DS4830A
Optical Microcontroller
User's Guide

Rev 0; 12/13

Maxim Integrated Products, Inc. 160 Rio Robles, San Jose, CA 95134 USA 1-408-601-1000

Contents

SECTION 1 – OVERVIEW

The DS4830A optical microcontroller is a low-power, 16-bit microcontroller with a unique peripheral set supporting a wide variety of optical transceiver controller applications. It provides a complete optical control, calibration, and monitor solution. The DS4830A is based on the high-performance, 16-bit, reduced instruction set computing (RISC) architecture with on-chip flash program memory and SRAM data memory.

The resources and features that the DS4830A provides for monitoring and controlling an optical system include the following:

- 16-Bit Low-Power Microcontroller
- 400kHz I^2C-Compatible Slave Communication Interface
  - Four User-Programmable Slave Addresses
  - 8-Byte Transmit Page for Each Slave Address
  - 8-Byte Receive Page Shared Between All Slave Addresses
- 32KWords Flash Program Memory
- 2KWords Data RAM
- 32-Level Hardware Stack
- 13-Bit ADC with a 26 Input Mux
  - 16 Single or 8 Differential Mode ADC Channels
  - Four User-Selectable Gains for Individual Channel
  - V_{DD}, Internal Reference, and DAC External References Measurement
  - ADC Samples Averaging Options
- 10 PWM Channels
  - Pulse Spreading Using Delta-Sigma Algorithm
  - PWM Output Synchronization
  - User-Selectable 7- to 16-Bit Resolution
  - 1MHz Switching Using 133MHz External Clock
- 10-Bit Fast Comparator with 16 Input Mux
  - Single and Differential Mode
  - Low and High Threshold Configurations
  - 3.2µs Conversion Time per Channel
- Two Independent Sample and Hold (S/H)
  - Single, Fast, and Dual Mode Operation
  - Internal and External Trigger Option
  - Pin Discharge
  - S/H Samples Averaging Options
- Fast Internal Die Temperature Sensors with Averaging Option
- 12-Bit, 8 Voltage DAC Channels Selectable Internal or External Reference Option
- Serial Interfaces
  - SPI Master and Slave Interface
  - 400kHz I^2C-Compatible Master with Alternate Location Option
  - 3-Wire Master Interface
- Dual Hardware Multiplier Unit
- Two 16-Bit Timers with Synchronous and Compare Modes
- Watchdog Timer
- Maskable Interrupt Sources
- Brownout Monitor
- 31 GPIO pins
- Supply Voltage Monitoring
- Internal 20MHz Oscillator, CPU Core Frequency 10MHz
- Included ROM Routines that allow Bootloading and In-Application Programming of Flash Memory
- In-System Debugging
- Four Software Interrupts
- Fast Hardware CRC-8 for Packet Error Checking (PEC)

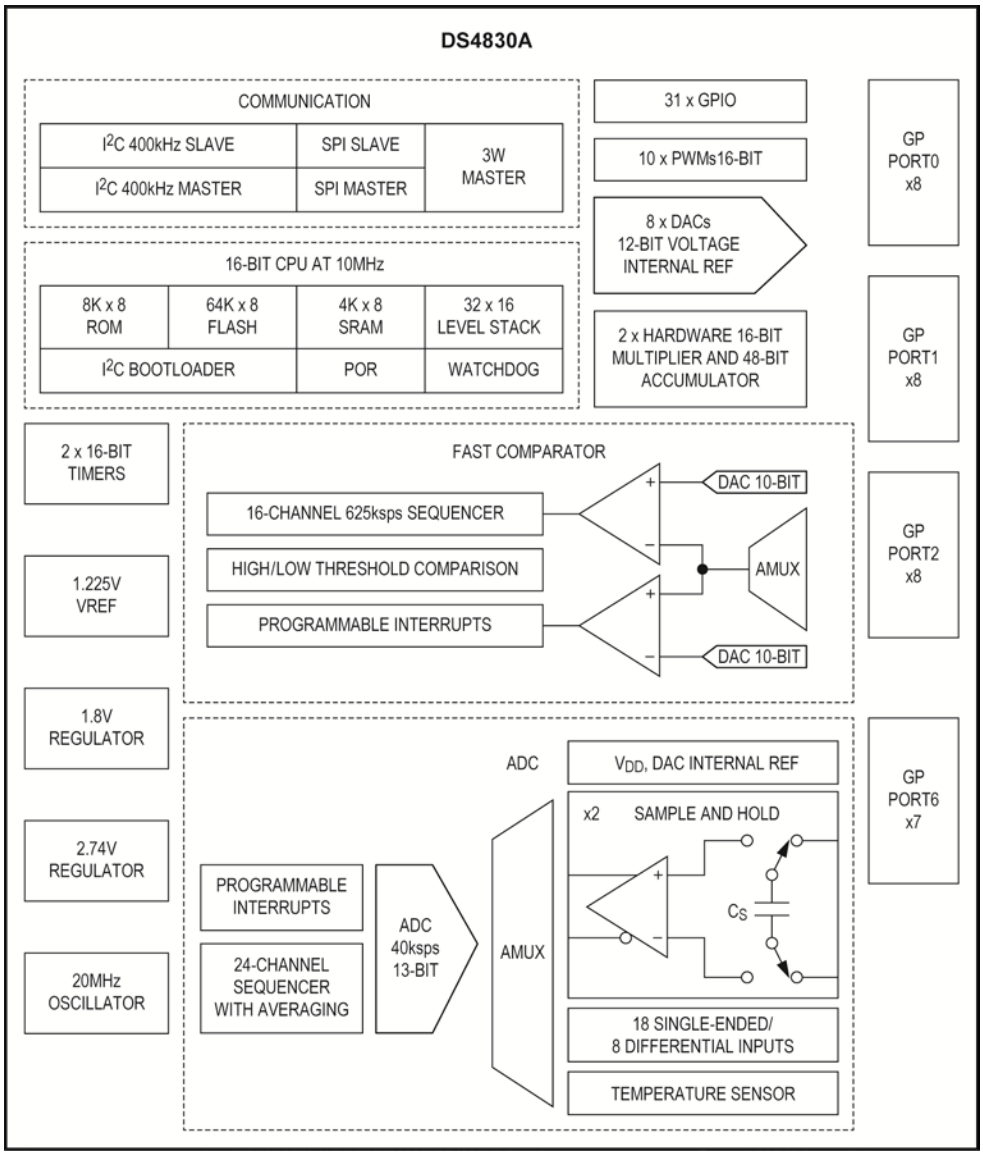


Figure 1-1: DS4830A Block Diagram

This document is provided as a supplement to the DS4830A IC data sheet. This user's guide provides the information necessary to develop applications using the DS4830A. All electrical and timing specifications, pin descriptions, package information, and ordering information can be found in the DS4830A IC data sheet.

SECTION 2 – ARCHITECTURE

The DS4830A contains a low-cost, high-performance microcontroller with flash memory. It is structured on a highly advanced, 16-accumulator-based, 16-bit RISC architecture. Fetch and execution operations are completed in one cycle without pipelining, since the instruction contains both the opcode and data. The highly efficient core is supported by 16 accumulators and a 32-level hardware stack, enabling fast subroutine calling and task switching.

Data can be quickly and efficiently manipulated with three internal data pointers. Two of these data pointers, DP0 and DP1, are stand-alone 16-bit pointers. The third data pointer, Frame Pointer, is composed of a 16-bit base pointer (BP) and an 8-bit offset register (OFFS). All three pointers support post-increment/decrement functionality for read operations and pre-increment/decrement for write operations. For the Frame Pointer (FP=BP[OFFS]), the increment/decrement operation is executed on the OFFS register and does not affect the base pointer. Multiple data pointers allow more than one function to access data memory without having to save and restore data pointers each time.

Stack functionality is provided by dedicated memory with a 16-bit width and a depth of 32. An on-chip memory management unit (MMU) allows logical remapping of the program and data spaces, and thus facilitates in-system programming and fast access to data tables, arrays, and constants located in flash memory.

This section provides details on the following topics.

1. Instruction decoding
2. Register space
3. Memory types
4. Program and data memory mapping and access
5. Data alignment
6. Reset conditions
7. Clock generation

2.1 – Instruction Decoding

The DS4830A uses the standard 16-bit core instruction set, which is described in the Instruction Set section. Every instruction is encoded as a single 16-bit word. The instruction word format is shown in Figure 2-1.

FORMAT	DESTINATION							SOURCE							
f	d	d	d	d	d	d	d	s	s	s	s	s	s	s	s

Figure 2-1: Instruction Word Format

- Bit 15 (f) indicates the format for the source field of the instruction as follows:
  - If f equals 0, the instruction is an immediate source instruction. The source field represents an immediate 8-bit value.
  - If f equals 1, the instruction is a register source instruction. The source field represents the register that the source value will be read from.
- Bits 14 to 8 (ddddddd) represent the destination for the transfer. This value always represents a destination register. The lower four bits contain the module specifier and the upper three bits contain the register index in that module.
- Bits 7 to 0 (ssssssss) represent the source for the transfer. Depending on the value of the format field, this can either be an immediate value or a source register. If this field represents a register, the lower four bits contain the module specifier and the upper four bits contain the register index in that module.

This instruction word format presents the following limitations.

1. There are 32 registers per register module, but only four bits are allocated to designate the source register and only three bits are allocated to designate the destination register.
2. The source field only provides 8 bits of data for an immediate value; however a 16-bit immediate value may be required.

The DS4830A uses a prefix register (PFX) to address these limitations. The prefix register provides the additional bits required to access all 32 register within a module. The prefix register also provides the additional 8 bits of data required to make a 16-bit immediate data source. The data that is written to the prefix register survives for only one clock cycle. This means the write to the prefix register must occur immediately prior to the instruction requiring the prefix register. The prefix register is cleared to zero after one cycle so it will not affect any other instructions. The write to the prefix register is

done automatically by the assembler and requires one additional execution cycle. So, while most instructions execute in a single cycle, two cycles are needed for instructions that require the prefix register.

The architecture of the DS4830A is transport-triggered. This means that writing to or reading from certain register locations will also cause side effects to occur. These side effects form the basis of the DS4830A's higher level opcodes, such as ADDC, OR, and JUMP. While these opcodes are actually implemented as MOVE instructions between certain register locations, the encoding is handled by the assembler and need not be a concern to the programmer. The unused "empty" locations in the System Register Modules are used for these higher level opcodes.

The instruction set is designed to be highly orthogonal. All arithmetic and logical operations that use two registers can use any register along with the accumulator. Data can be transferred between any two registers in a single instruction.

2.2 – Register Space

The DS4830A provides a total of 13 register modules broken up into two different groups. These groupings are descriptive only, as there is no difference between accessing the two register groups from a programming perspective.

The two groups are:

1. **System Registers:** These are modules 8h, 9h, and Bh through Fh. The System Registers in the DS4830A are used to implement higher level opcodes as well as the following common system features.
    - 16-bit ALU and associated status flags (zero, equals, carry, sign, overflow)
    - 16 working accumulator registers, each 16-bit, along with associated control registers
    - Instruction pointer
    - Registers for interrupt control, handling, and identification
    - Auto-decrementing Loop Counters for fast, compact looping
    - Two Data Pointer registers and a Frame Pointer for data memory access

2. **Peripheral Registers:** These are the lower six modules (Modules 0h through 5h). The Peripheral Registers in the DS4830A are used for functionalities such as ADC, Fast Comparator, DAC, PWM Outputs, Timers, Sample and Hold, 3-Wire, I^2C Master and Slave, SPI Master and Slave, 31-GPIO pins, etc. The Peripheral Registers are not used to implement opcodes.

Each System Register module has 16 registers, while each Peripheral Register module has 32 registers. The number of cycles required to access a particular register depends upon the register's index within the module. The access times based upon the register index are grouped as follows:

- The first eight registers (index 0h to 7h) in each module may be read from or written to in a single cycle
- The second eight registers (index 8h to 0Fh) may be read from in a single cycle and written to in two cycles (by using the prefix register PFX).
- The last sixteen registers (10h to 1Fh) in Peripheral Register modules may be read or written in two cycles (always requiring use of the prefix register PFX).

Registers may be 8 or 16 bits in length. Some registers may contain reserved bits. The user should not write to any reserved bits. Data transfers between registers of different sizes are handled as shown in Table 2-1.

- If the source and destination registers are both 8 bits wide, data is copied bit to bit.
- If the source register is 8 bits wide and the destination register is 16 bits wide, the data from the source register is transferred into the lower 8 bits of the destination register. The upper 8 bits of the destination register are set to the current value of the prefix register; this value is normally zero, but it can be set to a different value by the previous instruction if needed. The prefix register reverts back to zero after one cycle, so this must be done by the instruction immediately before the one that will be using the value.
- If the source register is 16 bits wide and the destination register is 8 bits wide, the lower 8 bits of the source are transferred to the destination register.
- If both registers are 16 bits wide, data is copied bit to bit.

The above rules apply to all data movements between defined registers. Data transfer to/from undefined register locations has the following behavior:

- If the destination is an undefined register, the MOVE is a dummy operation but may trigger an underlying operation according to the source register (e.g., @DPn--).
- If the destination is a defined register and the source is undefined, the source data for the transfer will depend upon the source module width. If the source is from a module containing 8-bit or 8-bit and 16-bit source registers,

the source data will be equal to the prefix data as the upper 8 bits and 00h as the lower 8 bits. If the source is from a module containing only 16-bit source registers, 0000h source data is used for the transfer.

Table 2-1. Register-to-Register Transfer Operations

SOURCE REGISTER SIZE (BITS)	DESTINATION REGISTER SIZE (BITS)	PREFIX SET?	DESTINATION SET TO VALUE	
			HIGH 8 BITS	LOW 8 BITS
8	8	X	—	Source [7:0]
8	16	No	00h	Source [7:0]
8	16	Yes	PFX [7:0]	Source [7:0]
16	8	X	—	Source [7:0]
16	16	X	Source [15:8]	Source [7:0]

2.3 – Memory Types

In addition to the internal register space, the DS4830A incorporates the following memory types:

- 32KWords of flash memory
- 4KWords of utility ROM contain a debugger and program loader
- 2KWords of SRAM
- 32-level hardware stack for storage of program return addresses

The memory on the DS4830A is organized according to Harvard architecture. This means that there are separate busses for both program and data memory. Stack memory is also separate and is accessed through a dedicated register set.

2.3.1 – Flash Memory

The DS4830A contains 32KWords (32K x 16) of flash memory. The flash memory begins at address 0000h and is contiguous through word address 7FFFh. The flash memory can also be used for storing lookup tables and other non-volatile data.

The incorporation of flash memory allows the contents of the flash memory to be upgraded in the field, either by the application or by one of the bootloaders (JTAG or I^2C). Writing to flash memory must be done indirectly by using routines that are provided by the utility ROM. See the Utility ROM and In-System Programming sections for more details.

2.3.2 – SRAM Memory

The DS4830A contains 2KWords (2K x 16) of SRAM memory. The SRAM memory address begins at address 0000h and is contiguous through word address 07FFh. The contents of the SRAM are indeterminate after power-on reset, but are maintained during non-POR resets.

When using the in-circuit debugging features, the highest 19 bytes of the SRAM must be reserved for saved state storage and working space for the debugging routines. If in-circuit debug is not used, the entire 2KWords of SRAM is available for application use.

2.3.3 – Utility ROM

The utility ROM is a 4kWord segment of memory. The utility ROM memory address begins at word address 8000h and is contiguous through word address 8FFFh. The utility ROM is programmed at the factory and cannot be modified. The utility ROM provides the following system utility functions:

- Reset vector (not user code reset vector)
- In-system programming (bootstrap loader) over JTAG or I^2C-compatible interfaces
- In-circuit debug routines
- Routines for in-application flash programming

Following any reset, the DS4830A automatically starts execution at the Reset Vector which is address 8000h in the utility ROM. The ROM code determines whether the program execution should immediately jump to the start of application code (flash address 0000h), or to one of the special routines mentioned. Routines within the utility ROM are firmware-accessible and can be called as subroutines by the application software. See the Utility ROM, In-System Programming, and In-Circuit Debug sections for more information on the routines provided by the utility ROM.

2.3.4 – Stack Memory

A 16-bit, 32-level on-chip stack provides storage for program return addresses and temporary storage of system registers. The stack is used automatically by the processor when the CALL, RET, and RETI instructions are executed, and when an interrupt is serviced. The stack can also be used explicitly to store and retrieve data by using the @SP- - source, @++SP destination, or the PUSH, POP, and POPI instructions. The POPI instruction acts identically to the POP instruction except that it additionally clears the INS bit.

The width of the stack is 16 bits to accommodate the instruction pointer size. On reset, the stack pointer SP initializes to the top of the stack (1Fh). The CALL, PUSH, and interrupt vectoring operations first increment SP and then store a value at @SP. The RET, RETI, POP, and POPI operations first retrieve the value at @SP and then decrement SP.
The stack memory is initialized to indeterminate values upon reset or power-up. Stack memory is dedicated for stack operations only and cannot be accessed by the DS4830A program or data busses.

When using the in-circuit debugging features, one word of the stack must be reserved for the debugging routines. If in-circuit debug is not used, the entire stack is available for application use.

2.4 – Program and Data Memory Mapping and Access

The memory on the DS4830A is implemented using Harvard architecture, with separate busses for program and data memory. The Memory Management Unit (MMU) allows the DS4830A to also support a pseudo-Von Neumann memory map. The pseudo Von Neumann memory map allows each of the memory segments (flash, SRAM, and utility ROM) to be logically mapped into a single contiguous memory map. This allows all of the memory segments to be accessed as both program and memory data. The pseudo-Von Neumann memory map provides the following advantages:

- Program execution can occur from the flash, SRAM, or utility ROM memory segments.
- The SRAM and flash memory segments can both be used for data memory.

Using the pseudo-Von Neumann memory map does have one restriction. This restriction is that a particular memory segment cannot be simultaneously accessed as both program and data memory.

2.4.1 – Program Memory Access

The instructions that the DS4830A is executing reside in what is defined as the program memory. The MMU fetches the instructions using the program bus. The Instruction Pointer (IP) register designates the program memory address of the next instruction to fetch. The Instruction Pointer is read/write accessible by the user software. A write to the Instruction Pointer will force program flow to the new address on the next cycle following the write. The content of the Instruction Pointer will be incremented by 1 automatically after each fetch operation. From an implementation perspective, system interrupts and branching instructions simply change the contents of the Instruction Pointer and force the opcode to fetch from a new program location.

2.4.2 – Program Memory Mapping

The DS4830A's mapping of the three memory segments (flash, SRAM, and utility ROM) as program memory is shown in Figure 2-2. The mapping of memory segments into program space is always the same. When referring to memory as program memory, all addresses are given as word addresses. The 32KWord flash memory segment is located at memory location 0000h through 7FFFh and is logically divided into two pages, each containing 16KWords. The utility ROM is located from location 8000h through 8FFFh, followed by the SRAM memory segment at location A000h through A7FFh. The user code reset vector, which is the first instruction of user program code that is executed, is located at flash memory address 0000h. User program code should always begin at this address.

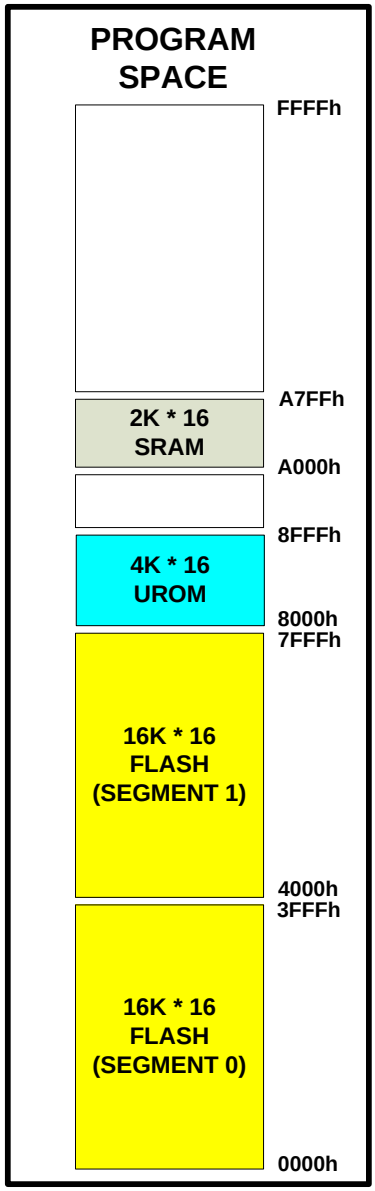


Figure 2-2: Program Memory Mapping

2.4.3 – Data Memory Access

Data memory mapping and access control are handled by the memory management unit (MMU). Read/write access to data memory can be in word or in byte mode. The DS4830A provides three pointers that can be used for indirect accessing of data memory. The DS4830A has two data pointers (@DPn) and one frame pointer (@BP[OFFS]). These pointers are implemented as registers that can be directly accessed by user software. A data memory access requires only one system clock period.

2.4.3.1 – Data Pointers

To access data memory, the data pointers are used as one of the operands in a MOVE instruction. If the data pointer is used as a source, the core performs a load operation that reads data from the memory location addressed by the data

pointer. If the data pointer is used as destination, the core performs a store operation that writes data to the memory location addressed by the data pointer. Following are some examples of setting and using a data pointer.

```
move DP[0], #0100h          ; set pointer DP[0] to address 100h
move Acc, @DP[0]            ; read data from location 100h
move @DP[0], Acc            ; write to location 100h
```

The address pointed to by the data pointers can be automatically incremented or decremented. If the data pointer is used as a source, the pointer can be incremented or decremented after the data access. If the data pointer is used as a destination, the increment or decrement can occur prior to the data access. Following are examples of using the data pointers increment/decrement features.

```
move Acc, @DP[0]++          ; increment DP[0] after read
move Acc, @DP[1]--          ; decrement DP[1] after read
move @++DP[0], Acc          ; increment DP[0] before write
move @--DP[1], Acc          ; decrement DP[0] before write
```

2.4.3.2 – Frame Pointer

The frame pointer (BP[OFFS]) is formed by the 16-bit unsigned addition of the 16-bit Frame Pointer Base Register (BP) and the 8-bit Frame Pointer Offset Register (OFFS). The method the DS4830A uses to access data using the frame pointer is similar to the data pointers. When increments or decrements are used, only the value of OFFS is incremented or decremented. The base pointer (BP) will remain unaffected by increments or decrements of the OFFS register, including when the OFFS register rolls over from FFh to 00h or from 00h to FFh. Following are examples of how to use the frame pointer.

```
move BP, #0100h             ; set base pointer to address 100h
move OFFS, #10h             ; set the offset to 10h,
move Acc, @BP[OFFS]         ; read data from location 0110h
move @BP[OFFS], Acc         ; write data to location 0110h
move Acc, @BP[OFFS++]       ; increment OFFS after read
move Acc, @BP[OFFS++]       ; decrement OFFS after read
move @BP[++OFFS], Acc       ; increment OFFS before write
move @BP[--OFFS], Acc       ; decrement OFFS before write
```

2.4.4 – Data Memory Mapping

The DS4830A's pseudo-Von Neumann memory map allows the MMU to read data from each of the three memory segments (flash, SRAM, utility ROM). The MMU can also write data directly to the SRAM memory segment. Data memory can be written to the flash memory segment, but because writing to flash requires the use of the utility ROM routines, this is not a direct access. The logical mapping of the three memory segments as data memory varies depending upon:

- which memory segment instructions are currently being executed from
- if data memory is being accessed in word or byte mode

In all cases, whichever memory segment is currently being used, program memory cannot be accessed as data memory.

When the program is currently executing instructions from either the SRAM or utility ROM memory segments, the flash memory will be mapped to half of the data memory space. If word access mode is selected, both pages (32KWords) can be logically mapped to data memory space. If byte access mode is selected, only one page (32KBytes) can be logically mapped to half of the data memory space. When operating in byte access mode, the selection of which flash page is mapped into data memory space is determined by the Code Data Access bit (CDA0):

CDA0	Selected Page in Byte Mode	Selected Page in Word Mode
0	P0	P0 and P1
1	P1	P0 and P1

The next three sections detail the mapping of the different memory segments as data memory depending upon which memory segment instructions are currently being executed from.

2.4.4.1 – Memory Map When Executing from Flash Memory

When executing from the flash memory:

- Read and write operations of SRAM memory are executed normally.
- The utility ROM can be read as data, starting at 8000h of the data space. The utility ROM cannot be written.

Figure 2-3 illustrates the mapping of the SRAM and utility ROM memory segments into data memory space when code is executing from the flash memory segment.

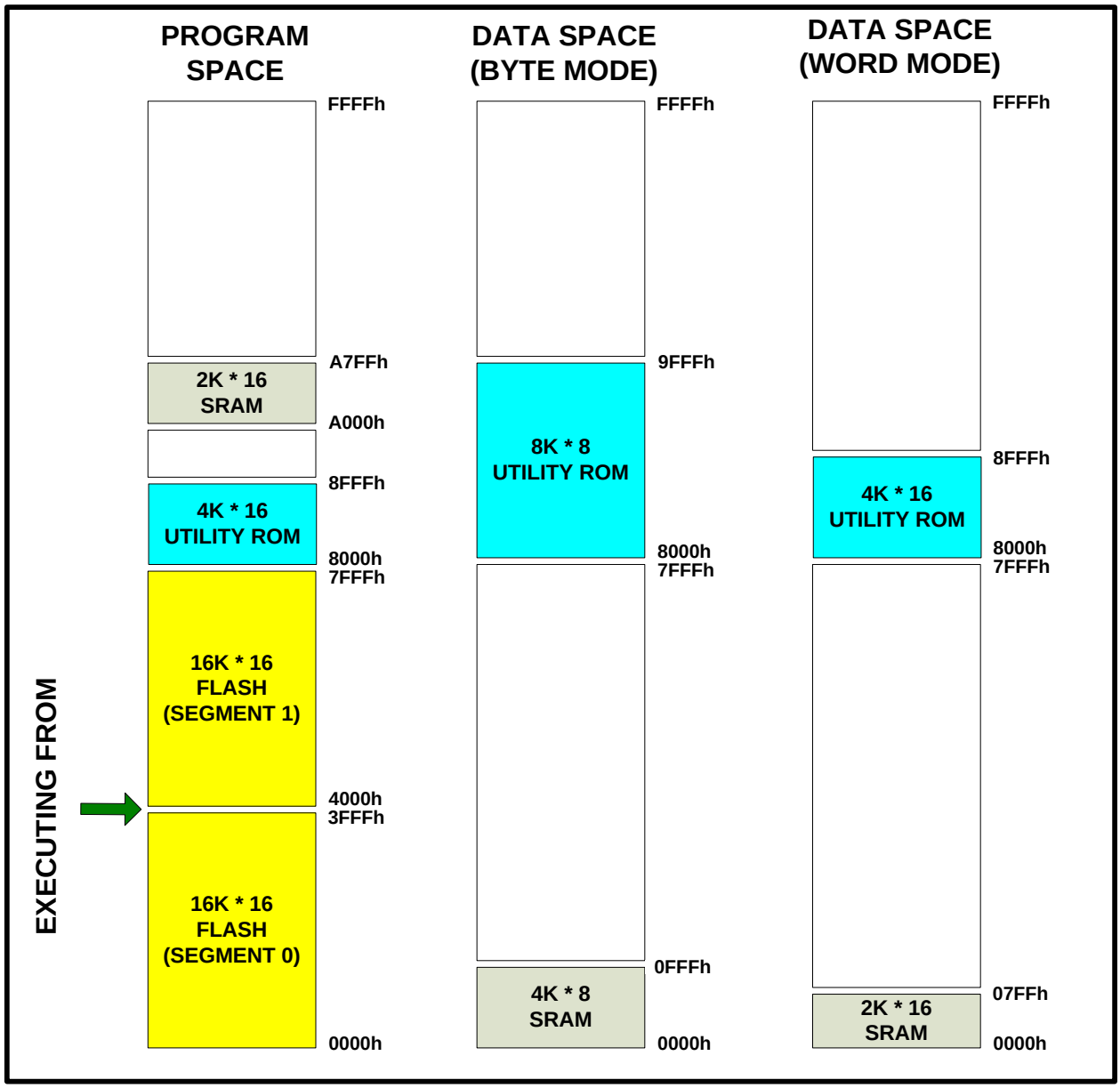


Figure 2-3: Memory Map When Executing from Flash Memory

2.4.4.2 – Memory Map When Executing from Utility ROM

When executing from the utility ROM:

- Read and write operations of SRAM memory are executed normally.
- Reading of flash memory is executed normally. Writing to flash memory requires the use of the utility ROM routines.
- One page (byte access mode) or both pages (word access mode) of the flash memory can be accessed as data with an offset of 8000h as determined by the CDA0 bit.

Figure 2-4 illustrates the mapping of the SRAM and flash memory segments into data memory space when code is executing from the utility ROM memory segment.

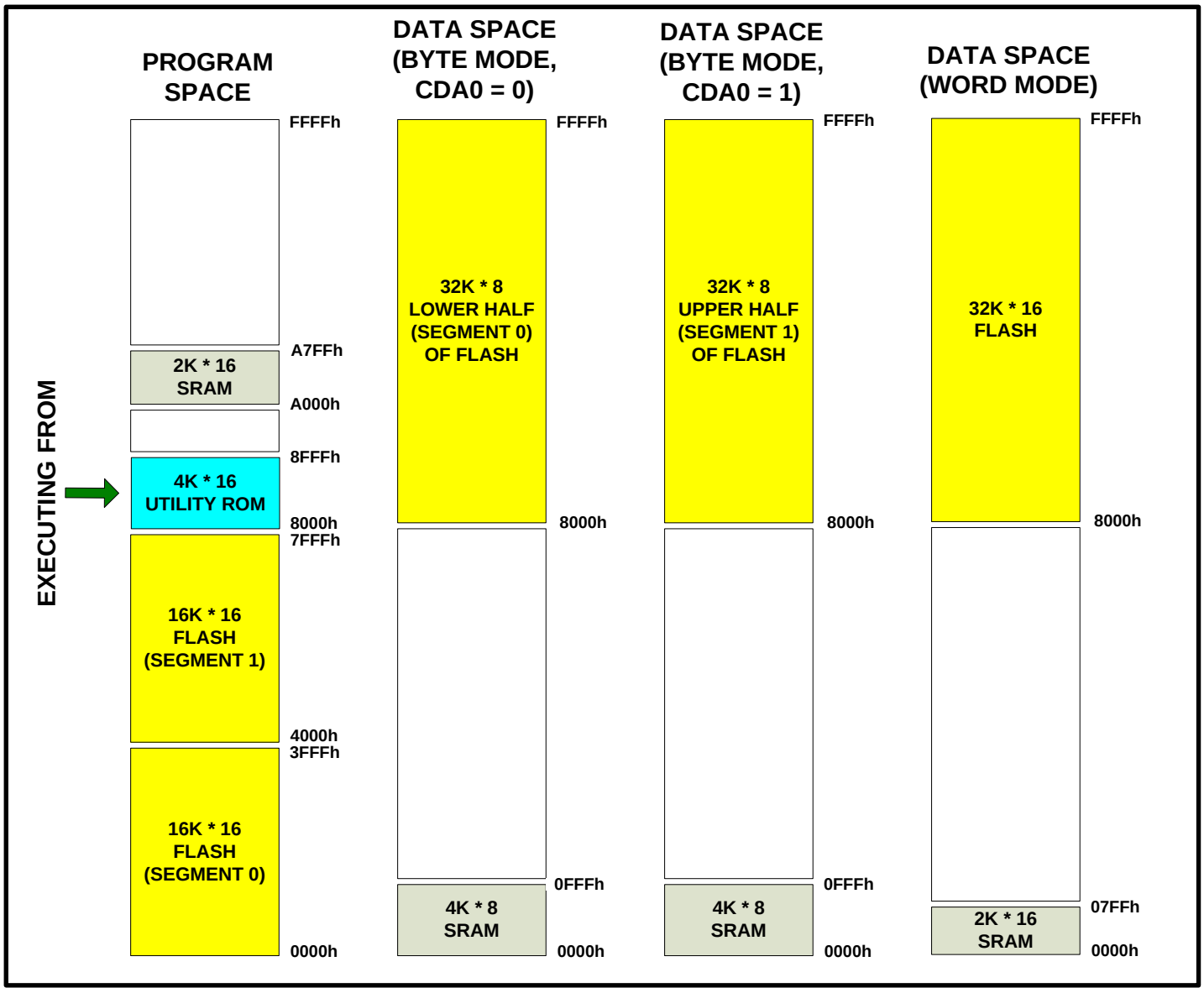


Figure 2-4: Memory Map When Executing from Utility ROM

2.4.4.3 – Memory Map When Executing from SRAM

When executing from the SRAM:

- The utility ROM can be read as data, starting at 8000h of the data space. The utility ROM cannot be written.
- Reading of flash memory is executed normally. Writing to flash memory requires the use of the utility ROM routines.
- One page (byte access mode) or both pages (word access mode) of the flash memory can be accessed as data with an offset of 0000h. For byte access mode, the page of flash accessed is determined by the CDA0 bit.

Figure 2-5 illustrates the mapping of the flash and utility ROM memory segments into data memory space when code is executing from the SRAM memory segment.

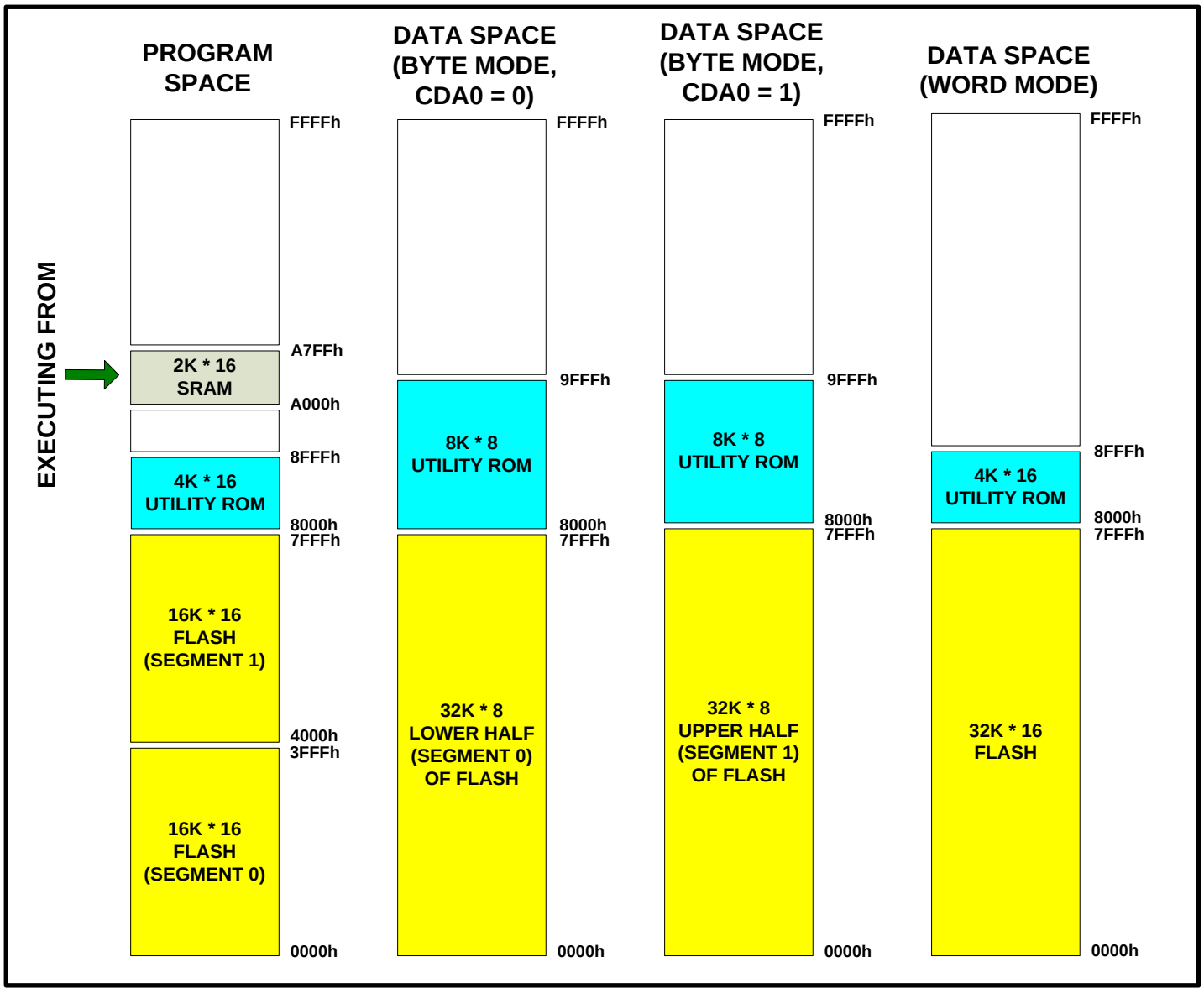


Figure 2-5: Memory Map When Executing from SRAM

2.5 – Data Alignment

To support merged program and data memory operation while maintaining efficient memory space usage, the data memory must be able to support both byte and word mode accessing. Data is aligned in data memory as words, but the effective data address is resolved to bytes. This data alignment allows program instruction fetching in words while maintaining data accessibility at the byte level. It is important to realize that this accessibility requires strict word alignment. All executable or data words must align to an even address in byte mode. Care must be taken when updating a code segment as misalignment of words will likely result in loss of program execution control.

Memory will always be read as a complete word, whether for program fetch or data access. The program decoder always uses a full 16-bit word. The data access can utilize a word or an individual byte. Data memory is organized as two byte-wide memory banks with common word address decode but two 8-bit data buses. In byte mode, data pointer hardware reads out the full word containing the selected byte using the effective data word address pointer (the least significant bit of the byte data pointer is not initially used). Then, the least significant data pointer bit functions as the byte select that is used to place the correct byte on the data bus. For write access, data pointer hardware addresses a particular word using the effective data word address while the least significant bit selects the corresponding data bank for write. The contents of the other byte are left unaffected.

2.6 – Reset Conditions

The DS4830A has several possible sources of reset.

- Power-On/Brownout Reset
- Watchdog Timer Reset
- External Reset
- Internal System Reset
- Soft Reset

Once a reset condition has completed or been removed, code execution begins at the beginning of utility ROM, which is address 8000h. The utility ROM code interrogates the I2C_SPE, JTAG_SPE, and PWL bits to determine if bootloading is necessary. If bootloading is not required, execution will jump to the user code reset vector, which is at flash memory address 0000h.

The $\overline{RST}$ pin is an input only.

2.6.1 – Power-On/Brownout Reset

The DS4830A provides a power-on reset (POR) circuit to ensure proper initialization of internal device states and analog circuits. The POR voltage threshold range is between approximately 1.1V and 1.7V. When V_{DD} is below the POR level, the state of all the DS4830A pins (except DAC port pins), including $\overline{RST}$, is weak pullup. The port pins having DAC function are high impedance on POR.

The DS4830A also includes brownout detection capability. This is an on-chip precision reference and comparator that monitors the supply voltage, V_{DD}, to ensure that it is within acceptable limits. If V_{DD} is below the brownout level (V_{BO}), the power monitor generates a reset. This can occur when:

- The DS4830A is being powered up and V_{DD} is above the POR level but still less than V_{BO}.
- V_{DD} drops from an acceptable level to less than V_{BO}.

Once V_{DD} exceeds V_{BO}, the DS4830A exits the reset condition and the internal oscillator starts up. After approximately 1ms the DS4830A performs the following tasks.

- All registers and circuits enter their reset state
- The POR flag in the Watchdog Control Register is set to indicate the source of the reset
- The DS4830A begins normal operation (CPU State)
- Code execution begins at utility ROM location 8000h

The transition between POR, Brownout, and normal operation is detailed in Figure 2-6: DS4830A State Diagram.

Note: If V_{DD} is below V_{BO}, there is a chance that the SRAM gets corrupted. If the POR flag in WDCN is set, all data in SRAM should be re-initialized.

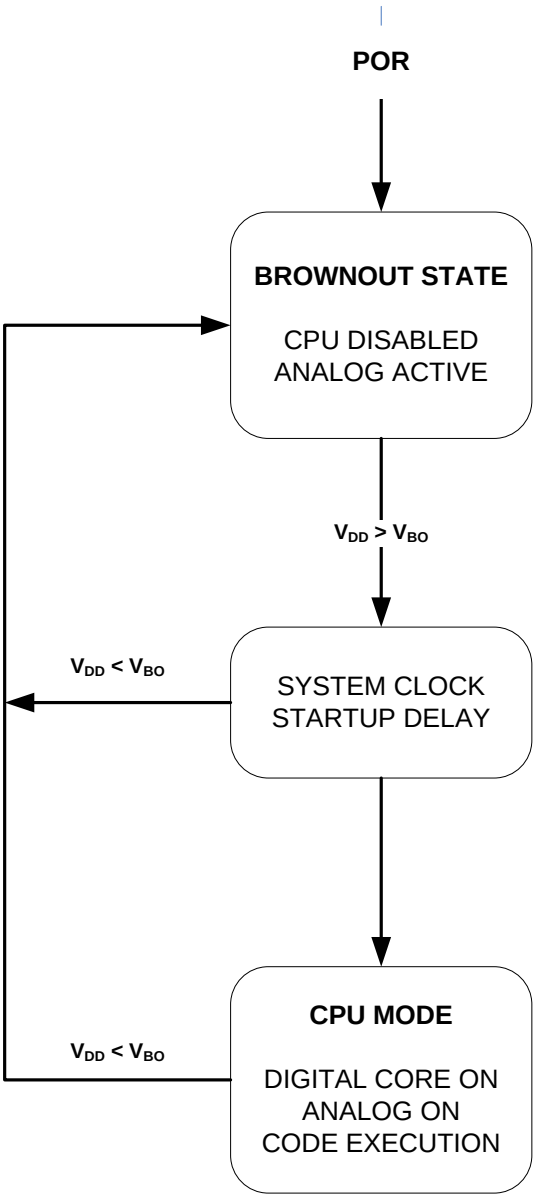


Figure 2-6: DS4830A State Diagram

2.6.2 – Watchdog Timer Reset

The watchdog timer is a programmable hardware timer that can be used to reset the processor in case a software lockup or other unrecoverable error occurs. Once the watchdog is enabled, software must reset the watchdog timer periodically. If the processor does not reset the watchdog timer before it elapses, the watchdog can initiate a reset.

If the watchdog resets the processor, the DS4830A will remain in reset for 12 clock cycles. When a reset occurs due to a watchdog timeout, the Watchdog Timer Reset Flag (WTRF) in the WDCN register is set to indicate the source of the reset.

2.6.3 – External Reset

During normal operation, the DS4830A is placed into external reset when the $\overline{RST}$ pin is held at logic 0 for at least four clock cycles. Once the DS4830A enters reset mode, it remains in reset as long as the $\overline{RST}$ pin is held at logic 0. After the $\overline{RST}$ pin returns to logic 1, the processor exits reset within 12 clock cycles.

An external reset pulse on the $\overline{RST}$ pin will reset the DS4830A and return to normal CPU mode operation within 10 clock cycles.

2.6.4 – Internal System Resets

There are two possible sources of internal system resets. An internal reset will hold the DS4830A in reset mode for 12 clock cycles.

1. When data BBh is written to the special I^2C slave address 34h.
2. When in-system programming is complete and the ROD bit is set to 1.

2.6.5 – Software Reset

The device UROM provides option to soft reset through the application program. The application program jumps to UROM code which generates the internal system reset. UROM location 8854h has code when executed generates internal reset. Application program can jump to this location to generate software reset.

asm ("LJUMP #8854h")

2.7 – Clock Generation

The DS4830A generates its 20MHz peripheral clock using an internal oscillator and generates 10MHz instruction clock using divide by 2 circuit. This oscillator starts up when V_{DD} exceeds the brownout voltage level, V_{BO}. There is a delay of approximately 1ms in the oscillator start up and beginning of clock. This delay ensures that the clock is stable prior to beginning normal operation.

SECTION 3 – SYSTEM REGISTER DESCRIPTIONS

Most functions of the DS4830A are controlled by sets of registers. These registers provide a working space for memory operations as well as configuring and addressing peripheral registers on the device. Registers are divided into two major types: system registers and peripheral registers. The common register set, also known as the system registers, includes ALU access and control registers, accumulator registers, data pointers, interrupt vectors and control, and stack pointer. The peripheral registers define additional functionality and the functionality is broken up into discrete modules.

This section describes the DS4830A's system registers. Table 3-1 shows the DS4830A system register map. Table 3-2 explains system register bit functions. This is followed by a detailed bit description.

Table 3-1: System Register Map

REGISTER INDEX	REGISTER MODULE						
	AP (08h)	**A (09h)**	**PFX (0Bh)**	**IP (0Ch)**	**SP (0Dh)**	**DPC (0Eh)**	**DP (0Fh)**
00h	AP	A[0]	PFX[0]	IP			
01h	APC	A[1]	PFX[1]		SP		
02h		A[2]	PFX[2]		IV		
03h		A[3]	PFX[3]			OFFS	DP[0]
04h	PSF	A[4]	PFX[4]			DPC	
05h	IC	A[5]	PFX[5]			GR	
06h	IMR	A[6]	PFX[6]		LC[0]	GRL	
07h		A[7]	PFX[7]		LC[1]	BP	DP[1]
08h	SC	A[8]				GRS	
09h		A[9]				GRH	
0Ah		A[10]				GRXL	
0Bh	IIR	A[11]				FP	
0Ch		A[12]					
0Dh		A[13]					
0Eh		A[14]					
0Fh	WDCN	A[15]					

Table 3-2. System Register Bit Functions

REGISTER	REGISTER BIT NUMBER															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AP									—	—	—	—	AP (4 bits)			
APC									CLR	IDS	—	—	—	MOD2	MOD1	MOD0
PSF									Z	S	—	GPF1	GPF0	OV	C	E
IC									—	—	—	—	—	—	INS	IGE
IMR									IMS	—	IM5	IM4	IM3	IM2	IM1	IM0
SC									TAP	—	—	CDA0	—	ROD	PWL	—
IIR									IIS	—	II5	II4	II3	II2	II1	II0
WDCN									POR	EWDI	WD1	WD0	WDIF	WTRF	EWT	RWT
A[n] (n=15:0)	A[n] (16 bits)															
PFX[n] (n=7:0)	PFX[n] (16 bits)															
IP	IP (16 bits)															
SP	—	—	—	—	—	—	—	—	—	—	—	SP (5 bits)				
IV	IV (16 bits)															
LC[0]	LC[0] (16 bits)															
LC[1]	LC[1] (16 bits)															
OFFS									OFFS (8 bits)							
DPC	—	—	—	—	—	—	—	—	—	—	—	WBS2	WBS1	WBS0	SDPS1	SDPS0
GR	GR (16 bits)															
GRL									GRL (8 bits)							
BP	BP (16 bits)															
GRS	GRS (16 bits) = (GRL : GRH)															
GRH									GRH (8 bits)							
GRXL	GRXL (16 bits) = (GRL.7, 8 bits) : (GRL, 8 bits)															
FP	FP = BP[OFFS] (16 bits)															
DP[0]	DP[0] (16 bits)															
DP[1]	DP[1] (16 bits)															

3.1 – Accumulator Pointer Register (AP, 08h[00h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

Bit	Name	Function
7:4	Reserved	Reserved. All reads return 0.
3:0	AP[3:0]	Active Accumulator Select. These bits select which of the 16 accumulator registers are used for arithmetic and logical operations. If the APC register has been set to perform automatic increment/decrement of the active accumulator, this setting will be automatically changed after each arithmetic or logical operation. If a 'MOVE AP, Acc' instruction is executed, any enabled AP inc/dec/modulo control will take precedence over the transfer of Acc data into AP.

3.2 – Accumulator Pointer Control Register (APC, 08h[01h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

Bit	Name	Function
7	CLR	AP Clear. Writing this bit to 1 clears the accumulator pointer AP to 0. Once set, this bit will automatically be reset to 0 by hardware. If a 'MOVE APC, Acc' instruction is executed requesting that AP be set to 0 (i.e., CLR = 1), the AP clear function overrides any enabled inc/dec/modulo control. All reads from this bit return 0.
6	IDS	Increment/Decrement Select. If this bit is set to 0, the accumulator pointer AP is incremented following each arithmetic or logical operation according to MOD[2:0]. If this bit is set to 1, the accumulator pointer AP is decremented following each arithmetic or logical operation according to MOD[2:0]. If MOD[2:0] is set to 000, the setting of this bit is ignored.
5:3	Reserved	Reserved. All reads return 0.
2:0	MOD[2:0]	Accumulator Pointer Auto Increment/Decrement Modulus. If these bits are set to a nonzero value, the accumulator pointer (AP[3:0]) will be automatically incremented or decremented following each arithmetic or logical operation. The mode for the auto-increment/ decrement is determined as follows:

MOD[2:0]	AUTO INCREMENT/DECREMENT MODE
000	No auto-increment/decrement (default)
001	Increment/decrement AP[0] modulo 2
010	Increment/decrement AP[1:0] modulo 4
011	Increment/decrement AP[2:0] modulo 8
100	Increment/decrement AP modulo 16
101 to 111	Reserved (modulo 16 when set)

3.3 – Processor Status Flags Register (PSF, 08h[04h])

Initialization: This register is cleared to 80h on all forms of reset.
Access: Bit 7 (Z), bit 6 (S), and bit 2 (OV) are read only. Bits [4:3] (GPF[1:0]), bit 1 (C), and bit 0 (E) are unrestricted read/write.

Bit	Name	Function
7	Z	Zero Flag. The value of this bit flag equals 1 whenever the active accumulator is equal to zero. This bit equals 0 if the active accumulator is not equal to 0.
6	S	Sign Flag. This bit flag mirrors the current value of the high bit of the active accumulator (Acc.15).
5	Reserved	Reserved. All reads return 0.
4:3	GPF[1:0]	General-Purpose Flags. These general-purpose flag bits are provided for user software control.
2	OV	Overflow Flag. This flag is set to 1 if there is a carry out of bit 14 but not out of bit 15, or a carry out of bit 15 but not out of bit 14 from the last arithmetic operation, otherwise, the OV flag remains as 0. OV indicates a negative number resulted as the sum of two positive operands, or a positive sum resulted from two negative operands.
1	C	Carry Flag. This bit flag is set to 1 whenever an add or subtract operation (ADD, ADDC, SUB, SUBB) returns a carry or borrow. This bit flag is cleared to 0 whenever an add or subtract operation does not return a carry or borrow. Many other instructions potentially affect the carry bit. Reference the instruction set documentation for details.
0	E	Equals Flag. This bit flag is set to 1 whenever a compare operation (CMP) returns an equal result. If a CMP operation returns not equal, this bit is cleared.

3.4 – Interrupt and Control Register (IC, 08h[05h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

Bit	Name	Function
7:2	Reserved	Reserved. All reads return 0.
1	INS	Interrupt In Service. The INS is set by hardware automatically when an interrupt is acknowledged. No further interrupts occur as long as the INS remains set. The interrupt service routine can clear the INS bit to allow interrupt nesting. Otherwise, the INS bit is cleared by hardware upon execution of an RETI or POPI instruction.
0	IGE	Interrupt Global Enable. If this bit is set to 1, interrupts are globally enabled, but still must be locally enabled to occur. If this bit is set to 0, all interrupts are disabled.

3.5 – Interrupt Mask Register (IMR, 08h[06h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted read/write access.

Bit	Name	Function
7	IMS	Interrupt Mask for System Modules
6	Reserved	Reserved. All reads return 0.
5	IM5	Interrupt Mask for Register Module 5
4	IM4	Interrupt Mask for Register Module 4
3	IM3	Interrupt Mask for Register Module 3
2	IM2	Interrupt Mask for Register Module 2
1	IM1	Interrupt Mask for Register Module 1
0	IM0	Interrupt Mask for Register Module 0

The first six bits in this register are interrupt mask bits for modules 0 to 5, one bit per module. The eighth bit, IMS, serves as a mask for any system module interrupt sources. Setting a mask bit allows the enabled interrupt sources for the associated module or system (for the case of IMS) to generate interrupt requests. Clearing the mask bit effectively disables all interrupt sources associated with that specific module or all system interrupt sources (for the case of IMS). The interrupt mask register is intended to facilitate user-definable interrupt prioritization.

3.6 – System Control Register (SC, 08h[08h])

Initialization: This register is reset to 1000 00s0b on all reset. Bit 1 (PWL) is set to 1 on a power-on reset only.
Access: Unrestricted read/write access.

<table>
<tr><th>Bit</th><th>Name</th><th>Function</th></tr>
<tr><td>7</td><td>TAP</td><td>Test Access Port (JTAG) Enable. This bit controls whether the Test Access Port special-function pins are enabled. The TAP defaults to being enabled. Clearing this bit to 0 disables the TAP special function pins.</td></tr>
<tr><td>6:5</td><td>Reserved</td><td>Reserved. All reads return 0.</td></tr>
<tr><td>4</td><td>CDA0</td><td>Code Data Access Bit 0.
The CDA0 bit is used to logically map the flash memory pages to the data space for read/write access. The logical data memory addresses of the flash depend on whether execution is from Utility ROM or SRAM. The CDA0 bit is not needed if data memory is accessed in word mode.
<table>
<tr><th>CDA0</th><th>Byte Mode Active Page</th><th>Word Mode Active Page</th></tr>
<tr><td>0</td><td>P0</td><td>P0 and P1</td></tr>
<tr><td>1</td><td>P1</td><td>P0 and P1</td></tr>
</table></td></tr>
<tr><td>3</td><td>Reserved</td><td>Reserved. All reads return 0.</td></tr>
<tr><td>2</td><td>ROD</td><td>ROM Operation Done. This bit is used to signify completion of a ROM operation sequence to the control units. This allows the Debug engine to determine the status of a ROM sequence. Setting this bit to logic 1 causes an internal system reset if the JTAG SPE bit is also set. Setting the ROD bit will clear the JTAG SPE and I2C_SPE bits if set. The ROD bit will be automatically cleared by hardware once the control unit acknowledges the done indication.</td></tr>
<tr><td>1</td><td>PWL</td><td>Password Lock. This bit defaults to 1 on a power-on reset. When this bit is 1, it requires a 32-byte password to be matched with the password in the program space before allowing access to the password protected in-circuit debug or bootstrap loader ROM routines. Clearing this bit to 0 disables the password protection for these ROM routines.</td></tr>
<tr><td>0</td><td>Reserved</td><td>Reserved. All reads return 0.</td></tr>
</table>

3.7 – Interrupt Identification Register (IIR, 08h[0Bh])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Read only.

Bit	Name	Function
7	IIS	Interrupt Identifier Flag for System Modules
6	Reserved	Reserved. All reads return 0.
5	II5	Interrupt Identifier Flag for Register Module 5
4	II4	Interrupt Identifier Flag for Register Module 4
3	II3	Interrupt Identifier Flag for Register Module 3
2	II2	Interrupt Identifier Flag for Register Module 2
1	II1	Interrupt Identifier Flag for Register Module 1
0	II0	Interrupt Identifier Flag for Register Module 0

The first six bits in this register indicate interrupts pending in modules 0 to 5, one bit per module. The eighth bit, IIS, indicates a pending system interrupt, such as from the watchdog timer. The interrupt pending flags will be set only for enabled interrupt sources waiting for service. The interrupt pending flag will be cleared when the pending interrupt sources within that module are disabled or when the interrupt flags are cleared by software

3.8 – Watchdog Control Register (WDCN, 08h[0Fh])

Initialization: Bits 5, 4, 3 and 0 are cleared to 0 on all forms of reset; for others, see individual bit descriptions.
Access: Unrestricted direct read/write access.

See the watchdog section for WDCN register description and further detail.

3.9 – Accumulator n Register (A[n], 09h[nh])

Initialization: These registers are cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
A[n][15:0]	These registers (n=0 to 15) act as the accumulator for all ALU arithmetic and logical operations when selected by the accumulator pointer (AP). They can also be used as a general-purpose working register.

3.10 – Prefix Register (PFX[n], 0Bh[n])

Initialization: This register is cleared to 0000h on all forms of reset.
yAccess: Unrestricted direct read/write access.

<table>
<tr><th>BIT</th><th>NAME</th><th>DESCRIPTION</th></tr>
<tr><td>15:0</td><td>PFX[n][15:0]</td><td>The Prefix register provides a means of supplying an additional 8 bits of high-order data for use by the succeeding instruction as well as providing additional indexing capabilities. This register will only hold any data written to it for one execution cycle, after which it will revert to 0000h. Although this is a 16-bit register, only the lower 8 bits are actually used for prefixing purposes by the next instruction. Writing to or reading from any index in the Prefix module will select the same 16-bit register. However, when the Prefix register is written, the index n used for the PFX[n] write also determines the high-order bits for the register source and destination specified in the following instruction.
The index selection reverts to 0 (default mode allowing selection of registers 0h to 7h for destinations) after one cycle in the same manner as the contents of the Prefix register.
<table>
<tr><th>WRITE TO</th><th>SOURCE REGISTER RANGE</th><th>DESTINATION REGISTER RANGE</th></tr>
<tr><td>PFX[0]</td><td>0h to Fh</td><td>0h to 7h</td></tr>
<tr><td>PFX[1]</td><td>10h to 1Fh</td><td>0h to 7h</td></tr>
<tr><td>PFX[2]</td><td>0h to Fh</td><td>8h to Fh</td></tr>
<tr><td>PFX[3]</td><td>10h to 1Fh</td><td>8h to Fh</td></tr>
<tr><td>PFX[4]</td><td>0h to Fh</td><td>10h to 17h</td></tr>
<tr><td>PFX[5]</td><td>10h to 1Fh</td><td>10h to 17h</td></tr>
<tr><td>PFX[6]</td><td>0h to Fh</td><td>18h to 1Fh</td></tr>
<tr><td>PFX[7]</td><td>10h to 1Fh</td><td>18h to 1Fh</td></tr>
</table>
</td></tr>
</table>

3.11 – Instruction Pointer Register (IP, 0Ch[00h])

Initialization: This register is cleared to 8000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
15:0	This register contains the address of the next instruction to be executed and is automatically incremented by 1 after each program fetch. Writing an address value to this register will cause program flow to jump to that address. Reading from this register will not affect program flow.

3.12 – Stack Pointer Register (SP, 0Dh[01h])

Initialization: This register is cleared to 001Fh on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
15:4	Reserved; all reads return 0.
4:0	These four bits indicate the current top of the hardware stack, from 0h to 1Fh. This pointer is incremented after a value is pushed on the stack and decremented before a value is popped from the stack.

3.13 – Interrupt Vector Register (IV, 0Dh[02h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
15:0	This register contains the address of the interrupt service routine. The interrupt handler will generate a CALL to this address whenever an interrupt is acknowledged.

3.14 – Loop Counter 0 Register (LC[0], 0Dh[06h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
15:0	This register is used as the loop counter for the DJNZ LC[0], src operation. This operation decrements LC[0] by one and then jumps to the address specified in the instruction by src if LC[0] = 0.

3.15 – Loop Counter 1 Register (LC[1], 0Dh[07h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
15:0	This register is used as the loop counter for the DJNZ LC[1], src operation. This operation decrements LC[1] by one and then jumps to the address specified in the instruction by src if LC[1] = 0.

3.16 – Frame Pointer Offset Register (OFFS, 0Eh[03h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
7:0	This 8-bit register provides the Frame Pointer (FP) offset from the base pointer (BP). The Frame Pointer is formed by unsigned addition of Frame Pointer Base Register (BP) and Frame Pointer Offset Register (Offs). The contents of this register can be post-incremented or post-decremented when using the Frame Pointer for read operations and may be pre-incremented or pre-decremented when using the Frame Pointer for write operations. A carry out or borrow resulting from an increment/decrement operation has no effect on the Frame Pointer Base Register (BP).

3.17 – Data Pointer Control Register (DPC, 0Eh[04h])

Initialization: This register is cleared to 001Ch on all forms of reset.
Access: Unrestricted direct read/write access.

<table>
<tr><th>BIT</th><th>NAME</th><th>DESCRIPTION</th></tr>
<tr><td>15:5</td><td>RESERVED</td><td>Reserved. All reads return 0.</td></tr>
<tr><td>4</td><td>WBS2</td><td>Word/Byte Select 2. This bit selects access mode for BP[OFFS]. When WBS2 is set to logic 1, the BP[Offs] is operated in word mode for data memory access; when WBS2 is cleared to logic 0, BP[Offs] is operated in byte mode for data memory access.</td></tr>
<tr><td>3</td><td>WBS1</td><td>Word/Byte Select 1. This bit selects access mode for DP[1]. When WBS1 is set to logic 1, the DP[1] is operated in word mode for data memory access; when WBS1 is cleared to logic 0, DP[1] is operated in byte mode for data memory access.</td></tr>
<tr><td>2</td><td>WBS0</td><td>Word/Byte Select 0. This bit selects access mode for DP[0]. When WBS0 is set to logic 1, the DP[0] is operated in word mode for data memory access; when WBS0 is cleared to logic 0, DP[0] is operated in byte mode for data memory access.</td></tr>
<tr><td>1:0</td><td>SDPS[1:0]</td><td>Source Data Pointer Select Bits[1:0]. These bits select one of the three data pointers as the active source pointer for the load operation. A new data pointer must be selected before being used to read data memory:
<table>
<tr><th>SDPS1</th><th>SDPS0</th><th>SOURCE POINTER SELECTION</th></tr>
<tr><td>0</td><td>0</td><td>DP[0]</td></tr>
<tr><td>0</td><td>1</td><td>DP[1]</td></tr>
<tr><td>1</td><td>0</td><td>FP (BP[Offs])</td></tr>
<tr><td>1</td><td>1</td><td>Reserved (select FP if set)</td></tr>
</table>
These bits default to 00b but do not activate DP[0] as an active source pointer until the SDPS bits are explicitly cleared to 00b or the DP[0] register is written by an instruction. Also, modifying the register contents of a data/frame pointer register (DP[0], DP[1], BP or Offs) will change the setting of the SDPS bits to reflect the active source pointer selection.</td></tr>
</table>

3.18 – General Register (GR, 0Eh[05h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
15:0	This register is intended primarily for supporting byte operations on 16-bit data. The 16-bit register is byte-readable, byte-writeable through the corresponding GRL and GRH 8-bit registers and byte-swappable through the GRS 16-bit register.

3.19 – General Register Low Byte (GRL, 0Eh[06h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
7:0	This register reflects the low byte of the GR register and is intended primarily for supporting byte operations on 16-bit data. Any data written to the GRL register will also be stored in the low byte of the GR register.

3.20 – Frame Pointer Base Register (BP, 0Eh[07h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
15:0	This register serves as the base pointer for the Frame Pointer (FP). The Frame Pointer is formed by unsigned addition of Frame Pointer Base Register (BP) and Frame Pointer Offset Register (Offs). The content of this base pointer register is not affected by increment/decrement operations performed on the offset (OFFS) register.

3.21 – General Register Byte-Swapped (GRS, 0Eh[08h])

Initialization: This register is cleared to 0000h on all forms of reset
Access: Unrestricted read-only access.

BIT	DESCRIPTION
15:0	This register is intended primarily for supporting byte operations on 16-bit data. This 16-bit read only register returns the byte-swapped value for the data contained in the GR register.

3.22 – General Register High Byte (GRH, 0Eh[09h])

Initialization: This register is cleared to 00h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
7:0	This register reflects the high byte of the GR register and is intended primarily for supporting byte operations on 16-bit data. Any data written to the GRH register will also be stored in the high byte of the GR register.

3.23 – General Register Sign Extended Low Byte (GRXL, 0Eh[0Ah])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read-only access.

BIT	DESCRIPTION
15:0	This register provides the sign extended low byte of GR as a 16-bit source.

3.24 – Frame Pointer Register (FP, 0Eh[0Bh])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read-only access.

BIT	DESCRIPTION
15:0	This register provides the current value of the frame pointer (BP[Offs]).

3.25 – Data Pointer 0 Register (DP[0], 0Fh[03h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
15:0	This register is used as a pointer to access data memory. DP[0] can be automatically incremented or decremented following each read operation or can be automatically incremented or decremented before each write operation.

3.26 – Data Pointer 1 Register (DP[1], 0Fh[07h])

Initialization: This register is cleared to 0000h on all forms of reset.
Access: Unrestricted direct read/write access.

BIT	DESCRIPTION
15:0	This register is used as a pointer to access data memory. DP[1] can be automatically incremented or decremented following each read operation or can be automatically incremented or decremented before each write operation.

SECTION 4 – PERIPHERAL REGISTER DESCRIPTIONS

Reg	M0	M1	M2	M3	M4	M5
0	PO2	I2CBUF_M	I2CBUF_S	MCNT	ADCN	QTDATA
1	PO1	I2CST_M	I2CST_S	MA	SENR	QTCN
2	PO0	I2CIE_M	MPNTR	MB	ADST	LTIL
3	EIF2	PO6	I2CTXFST	MC2	ADST1	HTIL
4	EIF1	CRC8IN	I2CTXFIE	MC1	ADDATA	SPIB_M
5	EIF0	MIIR1	I2CRXFST	MC0	SPIB_S	PWMDATA
6	GTV1	EIF6	I2CRXFIE	GTCN2	DADDR	PWMCN
7	GTCN1	EIE6	I2CST2_S	SHFT	MIIR4	PWMSYNC
8	PI2	PI6	RPNTR	MC1R	TEMPCN	LTIH
9	PI1	SVM	I2CCN_S	MC0R	SHCN	HTIH
10	PI0			GTC2		QTLST
11	GTC1			GTV2	PINSEL	
12		I2CCN_M	I2CSLA_S	GP_REG1	REFAVG	
13	EIE2	I2CCK_M	I2CSLA2_S	GP_REG2		
14	EIE1	I2CTO_M	I2CSLA3_S	MACSEL	TWR	MIIR5
15	EIE0	I2CSLA_M	I2CSLA4_S	USER_INT	RPCFG	
16	PD2	EIES6	I2CIE2_S	GP_REG3	SPICN_S	
17	PD1	PD6	MADDR	GP_REG4	SPICF_S	
18	PD0		MADDR2	GP_REG5	SPICK_S	SPICN_M
19	EIES2		MADDR3	GP_REG6	I2C_SPB	SPICF_M
20	EIES1		MADDR4	GP_REG7	DEV_NUM	SPICK_M
21	EIES0	CRC8OUT	CUR_SLA	GP_REG8	DACD0	
22			I2CIE_S	GP_REG9	DACD1	
23		ADCG1		GP_REG10	DACD2	
24		ADCG2	ICDT0	GP_REG11	DACD3	
25		ADVOFF	ICDT1	GP_REG12	DACD4	
26			ICDC	GP_REG13	DACD5	
27		ADCG3	ICDF	GP_REG14	DACD6	
28		ADCG4	ICDB	GP_REG15	DACD7	
29		CHIPREV	ICDA	GP_REG16	DACCFG	
30		I2CSLA2_M	ICDD		ADADDR	
31						

The DS4830A has sixteen 16-bit general-purpose registers GP_REG1-16 for application usage.

4.1 – Module 0 Peripheral Registers

MODULE 0																	
Register	**index**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PO2	00h									PO2[7:0]							
PO1	01h									PO1[7:0]							
PO0	02h									PO0[7:0]							
EIF2	03h									IFP2[7:0]							
EIF1	04h									IFP1[7:0]							
EIF0	05h									IFP0[7:0]							
GTV1	06h	GTV1[15:0]															
GTCN1	07h	-	-	-	GTR	MODE	CLK_SEL[1:0]		GTIE	-	-	-	GTIF	-	GTPS[2:0]		
PI2	08h									PI2[7:0]							
PI1	09h									PI1[7:0]							
PI0	0Ah									PI0[7:0]							
GTC1	0Bh	GTC1[15:0]															
EIE2	0Dh									IEP2[7:0]							
EIE1	0Eh									IEP1[7:0]							
EIE0	0Fh									IEP0[7:0]							
PD2	10h									PD2[7:0]							
PD1	11h									PD1[7:0]							
PD0	12h									PD0[7:0]							
EIES2	13h									IESP2[7:0]							
EIES1	14h									IESP1[7:0]							
EIES0	15h									IESP0[7:0]							

4.2 – Module 1 Peripheral Registers

MODULE 1																	
Register	**index**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
I2CBUF_M	00h	D[15:0]															
I2CST_M	01h	I2CBUS	I2CBUSY	-	I2CAMI2	I2CSPI	I2CSCL	I2CROI	I2CGCI	I2CNACKI	-	I2CAMI	I2CTOI	I2CSTRI	I2CRXI	I2CTXI	I2CSRI
I2CIE_M	02h	-	-	-	-	I2CSPIE	I2CAMI2E	I2CROIE	I2CGCIE	I2CNACKIE	-	I2CAMIE	I2CTOIE	I2CSTRIE	I2CRXIE	I2CTXIE	I2CSRIE
PO6	03h									-	PO6[6:0]						
CRC8IN	04h									CRC8IN[7:0]							
MIIR1	05h	-	-	-	-	-	-	-	I2CM	SVM	P6_6	P6_5	P6_4	P6_3	P6_2	P6_1	P6_0
EIF6	06h									-	IFP6[6:0]						
EIE6	07h									-	IEP6[6:0]						
PI6	08h									-	PI6[6:0]						
SVM	09h	-	-	-	-	SVMTH[3:0]				-	-	-	-	SVMI	SVMIE	SVMRDY	SVMEN
I2CCN_M	0Ch	-	-	-	I2CM_ALT	ADD2EN	SMB_MOD	I2CSTREN	I2CGCEN	I2CSTOP	I2CSTART	I2CACK	I2CSTRS	-	-	I2CMST	I2CEN
I2CCK_M	0Dh	I2CCKH[7:0]								I2CCKL[7:0]							
I2CTO_M	0Eh									I2CTO[7:0]							
I2CSLA_M	0Fh									SLAVE_ADDRESS[7:1]							I2CMODE
EIES6	10h									-	IESP6[6:0]						
PD6	11h									-	PD6[6:0]						
CRC8OUT	15h	-								CRC8OUT[7:0]							
ADCG1	17h	ADC VOLTAGE SCALE TRIM FOR GAIN1[13:0]														-	-
ADCG2	18h	ADC VOLTAGE SCALE TRIM FOR GAIN2[13:0]														-	-
ADVOFF	19h	ADC VOLTAGE OFFSET [15:0]															
ADCG3	1Bh	ADC VOLTAGE SCALE TRIM FOR GAIN3[13:0]														-	-
ADCG4	1Ch	ADC VOLTAGE SCALE TRIM FOR GAIN4[13:0]														-	-
CHIPREV	1Dh	CHIPREV[15:0]															
I2CSLA2_M	1Eh									SLAVE_ADDRESS[7:1]							I2CMODE

4.3 – Module 2 Peripheral Registers

MODULE 2																	
Register	**index**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
I2CBUF_S	00h	D[15:0]															
I2CST_S	01h	I2CBUS	I2CBUSY	-	-	-	I2CSCL	I2CROI	I2CGCI	I2CNACKI	-	I2CAMI	I2CTOI	I2CSTRI	I2CRXI	I2CTXI	I2CSRI
MPNTR	02h	-	-	-	-	-	PAGE[2:0]			MEM_PNTR[7:0]							
I2CTXFST	03h									-	-	-	-	-	-	THSH	-
I2CTXFIE	04h									TXPG_EN	-	-	-	-	-	THSH	-
I2CRXFST	05h									-	-	-	-	FULL	-	THSH	EMPTY
I2CRXFIE	06h									RXFIFO_EN	-	-	-	FULL	-	THSH	EMPTY
I2CST2_S	07h									-	-	I2CSPI	SADI	MADI	-	I2CXFRON	-
RPNTR	08h	-	-	-	-	-	PAGE[2:0]			MEM_PNTR[7:0]							
I2CCN_S	09h	-	-	ADDR4EN	ADDR3EN	ADDR2EN	SMB_MOD	I2CSTREN	I2CGCEN	I2CSTOP	I2CSTART	I2CACK	I2CSTRS	-	I2CMODE	-	I2CEN
I2CSLA_S	0Ch									SLAVE_ADDRESS[7:1]							I2CMODE
I2CSLA2_S	0Dh									SLAVE_ADDRESS[7:1]							I2CMODE
I2CSLA3_S	0Eh									SLAVE_ADDRESS[7:1]							I2CMODE
I2CSLA4_S	0Fh									SLAVE_ADDRESS[7:1]							I2CMODE
I2CIE2_S	10h									-	-	I2CSPIE	SADIE	MADIE	-	-	-
MADDR	11h	-	-	-	ROLLOVR	-	PAGE[2:0]			MEM_ADDR[7:0]							
MADDR2	12h	-	-	-	ROLLOVR	-	PAGE[2:0]			MEM_ADDR[7:0]							
MADDR3	13h	-	-	-	ROLLOVR	-	PAGE[2:0]			MEM_ADDR[7:0]							
MADDR4	14h	-	-	-	ROLLOVR	-	PAGE[2:0]			MEM_ADDR[7:0]							
CURR_SLA	15h									MADR_EN4	MADR_EN3	MADR_EN2	MADR_EN1	SLA4	SLA3	SLA2	SLA1
I2CIE_S	16h	-	-	-	-	-	-	I2CROIE	I2CGCIE	I2CNACKIE	-	I2CAMIE	I2CTOIE	I2CSTRIE	I2CRXIE	I2CTXIE	I2CSRIE
ICDT0	18h	ICDT0[15:0]															
ICDT1	19h	ICDT1[15:0]															
ICDC	1Ah									DME	-	REGE	-	CMD[3:0]			
ICDF	1Bh									-	-	-	-	PSS1	PSS0	JTAG_SPE	TXC
ICDB	1Ch									ICDB[7:0]							
ICDA	1Dh	ICDA[15:0]															
ICDD	1Eh	ICDD[15:0]															

4.4 – Module 3 Peripheral Registers

MODULE 3																	
Register	**index**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MCNT	00h									OF	MCW	CLD	SQU	OPCS	MSUB	MMAC	SUS
MA	01h	MA[15:0]															
MB	02h	MB[15:0]															
MC2	03h	MC2[15:0]															
MC1	04h	MC1[15:0]															
MC0	05h	MC0[15:0]															
GTCN2	06h	-	-	-	GTR	MODE	CLK_SEL[1:0]		GTIE	-	-	-	GTIF	-	GTPS[2:0]		
SHFT	07h									SHC	-	-	-	-	-	SR	SL
MC1R	08h	MC1R[15:0]															
MC0R	09h	MC0R[15:0]															
GTC2	0Ah	GTC2[15:0]															
GTV2	0Bh	GTV2[15:0]															
GP_REG1	0Ch	GP_REG1[15:0]															
GP_REG2	0Dh	GP_REG2[15:0]															
MACSEL	0Eh									-	-	-	-	-	-	-	MACRSEL
USER_INT	0Fh									SW_F3	SW_F2	SW_F1	SW_F0	SW_INT3	SW_INT2	SW_INT1	SW_INT0
GP_REG3	10h	GP_REG3[15:0]															
GP_REG4	11h	GP_REG4[15:0]															
GP_REG5	12h	GP_REG5[15:0]															
GP_REG6	13h	GP_REG6[15:0]															
GP_REG7	14h	GP_REG7[15:0]															
GP_REG8	15h	GP_REG8[15:0]															
GP_REG9	16h	GP_REG9[15:0]															
GP_REG10	17h	GP_REG10[15:0]															
GP_REG11	18h	GP_REG11[15:0]															
GP_REG12	19h	GP_REG12[15:0]															
GP_REG13	1Ah	GP_REG13[15:0]															
GP_REG14	1Bh	GP_REG14[15:0]															
GP_REG15	1Ch	GP_REG15[15:0]															
GP_REG16	1Dh	GP_REG16[15:0]															

4.5 – Module 4 Peripheral Registers

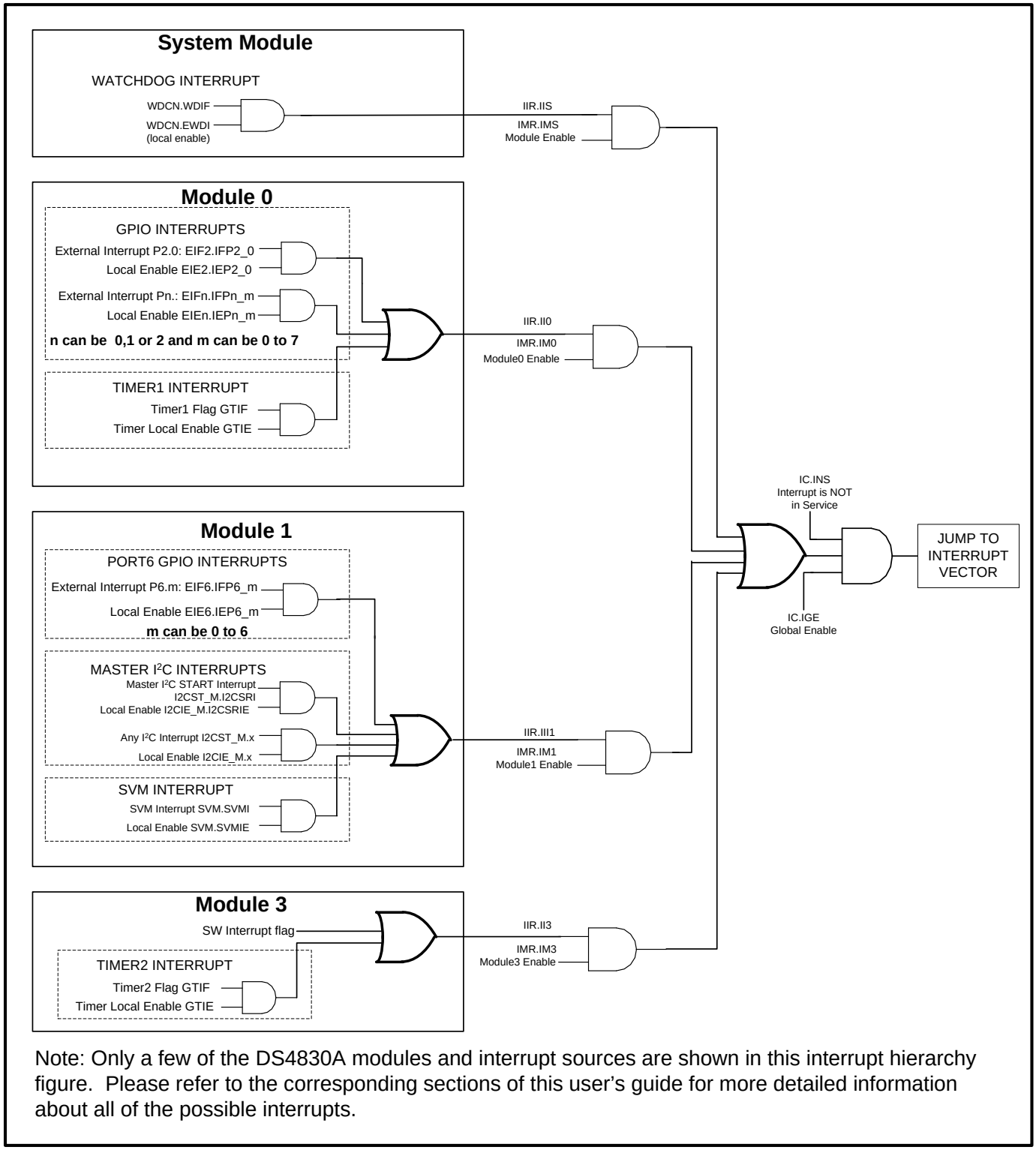
MODULE 4																	
Register	**index**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADCN	00h	ADCCLK[2:0]			NUM_SMP[4:0]					ADDAINV	ADCONT	ADDAIE	LOC_OVR	ADACQ[3:0]			
SENR	01h									-	-	INT_TRIG_EN1	INT_TRIG1	-	-	INT_TRIG_EN0	INT_TRIG0
ADST	02h	-	-	-	-	ENABLE_2X	-	-	-	ADCAVG	ADCONV	ADCFG	ADIDX[4:0]				
ADST1	03h									-	-	SH1DAI	SH0DAI			INTDAI	ADDAI
ADDATA	04h	ADDATA[15:0], SEE ADC SECTION FOR DETAILS															
SPIB_S	05h	SPIB[15:0]															
DADDR	06h	ADDR[6:0]							RWN	DATA[7:0]							
MIIR4	07h									-	-	-	-	-	SPI_S	TWI	ADC
TEMPCN	08h	-	-	-	-	-	INT_IEN	-	-	-	-	-	INT_ALIGN	-	-	-	INT_TEMP
SHCN	09h	SSC[3:0]				FAST_MODE	PIN_DIS1	PIN_DIS0	SH_DUAL	-	SH1_ALGN	SHDAI1_EN	SMP_HLD1	CLK_SEL	SH0_ALGN	SHDAI0_EN	SMP_HLD0
PINSEL	0Bh	PINSEL[15:0]															
REFAVG	0Ch	-	-	-	-	-	-	REFOUT	INTAVG	-	-	INTAVG[1:0]		SH1AVG[1:0]		SH0AVG[1:0]	
TWR	0Eh									TWEN	TWCP[2:0]			TWIE	TWCSDIS	TWI	BUSY
RPCFG	0Fh									-	-	-	-	-	-	REFB_CFG	REFA_CFG
SPICN_S	10h									STBY	SPIC	ROVR	WCOL	MODF	MODFE	MSTM	SPIEN
SPICF_S	11h									ESPII	SAS	-	-	-	CHR	CKPHA	CKPOL
SPICK_S	12h									SPICK[7:0]							
I2C_SPB	13h									-	-	-	-	-	-	-	I2C_SPE
DEV_NUM	14h									BOOT_DIS	DEVNUM[6:0]						
DACD0	15h	-	-	-	-	DACD0[11:0]											
DACD1	16h	-	-	-	-	DACD1[11:0]											
DACD2	17h	-	-	-	-	DACD2[11:0]											
DACD3	18h	-	-	-	-	DACD3[11:0]											
DACD4	19h	-	-	-	-	DACD4[11:0]											
DACD5	1Ah	-	-	-	-	DACD5[11:0]											
DACD6	1Bh	-	-	-	-	DACD6[11:0]											
DACD7	1Ch	-	-	-	-	DACD7[11:0]											
DACCFG	1Dh	DACCFG7[1:0]		DACCFG6[1:0]		DACCFG5[1:0]		DACCFG4[1:0]		DACCFG3[1:0]		DACCFG2[1:0]		DACCFG1[1:0]		DACCFG0[1:0]	
ADADDR	1Eh	-	-	-	ADSTART[4:0]					-	-	-	ADEND[4:0]				

4.6 – Module 5 Peripheral Registers

MODULE 5																	
Register	**index**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
QTDATA	00h	QTDATA[15:0] SEE QUICK TRIP FOR DETAILS															
QTCN	01h	-	-	-	QTEN	-	-	-	-	RW_LST	-	-	LTHT	QTIDX[3:0]			
LTIL	02h	LTIE[7:0]								LTIF[7:0]							
HTIL	03h	HTIE[7:0]								HTIF[7:0]							
SPIB_M	04h	SPIB[15:0]															
PWMDATA	05h	PWMDATA[15:0] SEE PWM SECTION FOR DETAILS															
PWMCN	06h	-	-	-	M_EN	-	-	-	UPDATE	PWM_SEL[3:0]				-	-	REG_SEL[1:0]	
PWMSYNC	07h	-	-	-	-	-	-	PWMSYNC[9:0]									
LTIH	08h	LTIE[15:8]								LTIF[15:8]							
HTIH	09h	HTIE[15:8]								HTIF[15:8]							
QTLIST	0Ah	-	-	-	-	-	-	-	-	QTSTART[3:0]				QTSTOP[3:0]			
MIIR5	0Eh									-	-	-	-	-	I2C_M	QT	SPI_M
SPICN_M	12h									STBY	SPIC	ROVR	WCOL	MODF	MODFE	MSTM	SPIEN
SPICF_M	13h									ESPII	SAS	-	-	-	CHR	CKPHA	CKPOL
SPICK_M	14h									SPICK[7:0]							

SECTION 5 – INTERRUPTS

The DS4830A provides a single, programmable interrupt vector (IV) that can be used to handle internal and external interrupts. Interrupts can be generated from system level sources (e.g., watchdog timer) or by sources associated with the peripheral modules. Only one interrupt can be handled at a time, and all interrupts naturally have the same priority. A programmable interrupt mask register allows software-controlled prioritization and nesting of high-priority interrupts. igure 5-1 shows a diagram of the interrupt hierarchy.



Note: Only a few of the DS4830A modules and interrupt sources are shown in this interrupt hierarchy figure. Please refer to the corresponding sections of this user's guide for more detailed information about all of the possible interrupts.

Figure 5-1: Interrupt Hierarchy

Note: Some of the DS4830A module and peripheral interrupts sources are shown in the Figure 5-1 interrupt hierarchy diagram. See the corresponding sections of this user's guide for more detailed information about all of the possible interrupts.

5.1 – Servicing Interrupts

For the DS4830A to service an interrupt, interrupts must be enabled locally, modularly, and globally. The Interrupt Global Enable (IGE) bit is located in the Interrupt Control (IC) register acts as a global interrupt mask. This bit defaults to 0, and it must be set to 1 before any interrupt takes place.

The local interrupt-enable bit for a particular source is in one of the peripheral registers associated with that peripheral module, or in a system register for any system interrupt source. Between the global and local enables are intermediate per-module and system interrupt mask bits. These mask bits reside in the Interrupt Mask system register. By implementing intermediate per-module masking capability in a single register, interrupt sources spanning multiple modules can be selectively enabled/disabled in a single instruction. This promotes a simple, fast, and user-definable interrupt prioritization scheme. The interrupt source-enable hierarchy is illustrated in Figure 5-1 as well as Table 5-1.

Table 5-1: Interrupt Sources and Control Bits

INTERRUPT	INTERRUPT FLAG	LOCAL ENABLE BIT	MODULE INTERRUPT IDENTIFICATION BIT	INTERRUPT IDENTIFICATION BIT	MODULE ENABLE BIT
External Interrupt Pp.n (here p = 0,1,2 and n = 0 to 7)	EIFp.IEn	EIEp.EXn	-	IIR.II0	IMR.IM0
Timer1 Interrupt	GTCN1.GTIF	GTCN1.GTIE	-		
External Interrupt Pp.n (here p = 6 and n = 0 to 6)	EIFp.IEn	EIEp.EXn	MIIR1.Pp_n	IIR.II1	IMR.IM1
Supply Voltage Monitor Interrupt	SVM.SVMI	SVM.SVMIE	MIIR1.SVM		
I^2C Master Start Interrupt	I2CST_M.I2CSRI	I2CIE_M.I2CSRIE	MIIR1.I2CM		
I^2C Master Transmit Complete Interrupt	I2CST_M.I2CTXI	I2CIE_M.I2CTXIE			
I^2C Master Receive Ready Interrupt	I2CST_M. I2CRXI	I2CIE_M.I2CRXIE			
I^2C Master Clock Stretch Interrupt	I2CST_M.I2CSTRI	I2CIE_M.I2CSTRIE			
I^2C Master Timeout Interrupt	I2CST_M.I2CTOI	I2CIE_M.I2CTOIE			
I^2C Master NACK Interrupt	I2CST_M.I2CNACKI	I2CIE_M.I2CNACKIE			
I^2C Master Receiver Overrun Interrupt	I2CST_M.I2CROI	I2CIE_M.I2CROIE			
I^2C Master Stop Interrupt	I2CST_M.I2CSPI	I2CIE_M.I2CSPIE			
I^2C Slave Start Interrupt	I2CST_S.I2CSRI	I2CIE_S.I2CSRIE	-	IIR.II2	IMR.IM2
I^2C Slave Transmit Complete Interrupt	I2CST_S.I2CTXI	I2CIE_S.I2CTXIE			
I^2C Slave Receive Ready Interrupt	I2CST_S. I2CRXI	I2CIE_S.I2CRXIE			
I^2C Slave Clock Stretch Interrupt	I2CST_S.I2CSTRI	I2CIE_S.I2CSTRIE			
I^2C Slave Timeout Interrupt	I2CST_S.I2CTOI	I2CIE_S.I2CTOIE			
I^2C Slave Address Match Interrupt	I2CST_S.I2CAMI	I2CIE_S.I2CAMIE			
I^2C Slave NACK Interrupt	I2CST_S.I2CNACKI	I2CIE_S.I2CNACKIE			
I^2C Slave General Call Interrupt	I2ST_S.I2CGCI	I2CIE_S.I2CGCIE			
I^2C Slave Receiver Overrun Interrupt	I2CST_S.I2CROI	I2CIE_S.I2CROIE			
I^2C Slave Stop Interrupt	I2CST2_S.I2CSPI	I2CIE2_S.I2CSPIE			
I^2C Slave Start Address Interrupt	I2CST2_S.SADI	I2CIE2_S. SADIE			
I^2C Slave Memory Address Interrupt	I2CST2_S.MADI	I2CIE2_S. MADIE			
I^2C Slave Page Threshold Interrupt	I2CTXFST.THSH	I2CTXFIE.THSH			
I^2C Slave FIFO Threshold Interrupt	I2CRXFST.THSH	I2CRXFIE.THSH			
Timer2 Interrupt	GTCN1.GTIF	GTCN1.GTIE	-	IIR.II3	IMR.IM3
Software Interrupts	SW.Fn (n = 0,1,2,3)	-	-		
ADC Data Available Interrupt	ADST1.ADDAI	ADCN.ADDAIE	MIIR4.ADC	IIR.II4	IMR.IM4
Internal Temperature Interrupt	ADST1.INTDAI	TEMPCN.INT_IEN			
Sample and Hold 0 Interrupt	ADST1.SH0DAI	SHCN.SHDAI0_EN			
Sample and Hold 1 Interrupt	ADST1.SH1DAI	SHCN.SHDAI1_EN			
3- Wire Interrupt	TWR.TWI	TWR.TWIE	MIIR4.TW		
SPI Slave Transfer Complete	SPICN_S.SPIC	SPICF_S.ESPII	MIIR4.SPI_S		
SPI Slave Write Collision	SPICN_S.WCOL				
SPI Slave Receive Overrun	SPICN_S.ROVR				

INTERRUPT	INTERRUPT FLAG	LOCAL ENABLE BIT	MODULE INTERRUPT IDENTIFICATION BIT	INTERRUPT IDENTIFICATION BIT	MODULE ENABLE BIT
LT 0 Interrupt	LTIL.IF0	LTIL.IE0			
LT 1 Interrupt	LTIL.IF1	LTIL.IE1			
LT 2 Interrupt	LTIL.IF2	LTIL.IE2			
LT 3 Interrupt	LTIL.IF3	LTIL.IE3			
LT 4 Interrupt	LTIL.IF4	LTIL.IE4			
LT 5 Interrupt	LTIL.IF5	LTIL.IE5			
LT 6 Interrupt	LTIL.IF6	LTIL.IE6			
LT 7 Interrupt	LTIL.IF7	LTIL.IE7			
LT 8 Interrupt	LTIH.IF8	LTIH.IE8			
LT 9 Interrupt	LTIH.IF9	LTIH.IE9			
LT 10 Interrupt	LTIH.IF10	LTIH.IE10			
LT 11 Interrupt	LTIH.IF11	LTIH.IE11			
LT 12 Interrupt	LTIH.IF12	LTIH.IE12			
LT 13 Interrupt	LTIH.IF13	LTIH.IE13			
LT 14 Interrupt	LTIH.IF14	LTIH.IE14			
LT 15 Interrupt	LTIH.IF15	LTIH.IE15			
HT 0 Interrupt	HTIL.IF0	HTIL.IE0			
HT 1 Interrupt	HTIL.IF1	HTIL.IE1			
HT 2 Interrupt	HTIL.IF2	HTIL.IE2			
HT 3 Interrupt	HTIL.IF3	HTIL.IE3			
HT 4 Interrupt	HTIL.IF4	HTIL.IE4			
HT 5 Interrupt	HTIL.IF5	HTIL.IE5			
HT 6 Interrupt	HTIL.IF6	HTIL.IE6			
HT 7 Interrupt	HTIL.IF7	HTIL.IE7	MIIR5.QT	IIR.II5	IMR.IM5
HT 8 Interrupt	HTIH.IF8	HTIH.IE8			
HT 9 Interrupt	HTIH.IF9	HTIH.IE9			
HT 10 Interrupt	HTIH.IF10	HTIH.IE10			
HT 11 Interrupt	HTIH.IF11	HTIH.IE11			
HT 12 Interrupt	HTIH.IF12	HTIH.IE12			
HT 13 Interrupt	HTIH.IF13	HTIH.IE13			
HT 14 Interrupt	HTIH.IF14	HTIH.IE14			
HT 15 Interrupt	HTIH.IF15	HTIH.IE15			
SPI Master Transfer Complete	SPICN_M.SPIC	SPICF_M.ESPII	MIIR5.SPI_M		
SPI Master Write Collision	SPICN_M.WCOL				
SPI Master Receive Overrun	SPICN_M.ROVR				
SPI Master Mode Fault	SPICN_M.MODF	SPICN_M.MODFE			
Watchdog Interrupt	WDCN.WDIF	WDCN.EWDI	N/A	IIR.IIS	IMR.IMS

When an interrupt condition occurs, its individual flag is set, even if the interrupt source is disabled at the local, module, or global level. Interrupt flags must be cleared within the user interrupt routine to avoid repeated interrupts from the same source. Since all interrupts vector to the address contained in the Interrupt Vector (IV) register, the Interrupt Identification Register (IIR) may be used by the interrupt service routine to determine the module source of an interrupt. The IIR contains a bit flag for each peripheral module and one flag associated with all system interrupts; if the bit for a module is set, then an interrupt is pending that was initiated by that module.

In the DS4830A MIIR registers are defined for module 1, 4, and 5. In these modules the DS4830A provides two ways to determine which block inside a module (for module 1, 4, and 5 only) caused an interrupt to occur. Module 1, 4 and 5 has Module Interrupt Identification Registers MIIR1, MIIR4 and MIIR5 respectively that indicate which of the module's interrupt sources has a pending interrupt. The peripheral register bits inside the module also provide a way to differentiate among interrupt sources. Section 5.2 has more detail on the Module Interrupt Identification Registers.

The Interrupt Vector (IV) register provides the location of the interrupt service routine. It may be set to any location within program memory. The IV register defaults to 0000h on reset or power-up, so if it is not changed to a different address, the user program must determine whether a jump to 0000h came from a reset or interrupt source.

5.2 – Module Interrupt Identification Registers

The MIIR registers are implemented to indicate which particular function within a peripheral module has caused the interrupt. The DS4830A has 6 peripheral modules, M0 through M5. MIIR registers are implemented in peripheral module 1, 4 and 5. The MIIR registers are 16-bit read-only registers and all of them default to 0000h on system reset.

Each defined bit in an MIIR register is the final interrupt from a specific function, i.e., the interrupt enable bit(s) ANDed with the interrupt flag(s). A function can have multiple flags, but they all are ANDed with corresponding

enable bits and combined to create a single interrupt identification bit for that specific function. For example, the I^2C master has several interrupt sources; however, they all are combined to form a single identification bit, MIIR1.I2CM. The individual register bit functions are defined as follows.

Peripheral Module 1 Interrupt Identification Register (MIIR1)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	I2CM	SVM	P6_6	P6_5	P6_4	P6_3	P6_2	P6_1	P6_0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

BIT	NAME	DESCRIPTION
15:9	Reserved	Reserved. A read returns 0.
8	I2CM	This bit is set when there is an interrupt from the I^2C master block. The I^2C interrupt is a combination of all interrupts defined in the I2CST_M register for the I^2C master block. The Master I^2C section has more detail on the individual interrupts.
7	SVM	This bit is set when there is an interrupt from Supply Voltage Monitor (SVM).
6	P6_6	This bit is set when there is an External GPIO Interrupt at P6.6.
5	P6_5	This bit is set when there is an External Interrupt at P6_5.
4	P6_4	This bit is set when there is an External Interrupt at P6.4.
3	P6_3	This bit is set when there is an External Interrupt at P6.3.
2	P6_2	This bit is set when there is an External Interrupt at P6.2.
1	P6_1	This bit is set when there is an External Interrupt at P6.1.
0	P6_0	This bit is set when there is an External Interrupt at P6.0.

Peripheral Module 4 Interrupt Identification Register (MIIR4)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	-	-	-	-	-	I2CS	TW	ADC
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

BIT	NAME	DESCRIPTION
15:3	Reserved	Reserved. A read returns 0.
2	SPI_S	This bit is set when there is an interrupt at SPI Slave.
1	TW	This bit is set when there is an interrupt from the 3Wire Block.
0	ADC	This bit is set when there is an Interrupt from the ADC.

Peripheral Module 5 Interrupt Identification Register (MIIR5)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	-	-	-	-	-	-	QT	SPI_M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

BIT	NAME	DESCRIPTION
15:2	Reserved	Reserved. A read returns 0.
1	QT	This bit is set when there is an interrupt from the fast comparator
0	SPI_M	This bit is set when there is an interrupt at SPI Slave.

5.3 – Interrupt System Operation

The interrupt handler hardware responds to any interrupt event when it is enabled. An interrupt event occurs when an interrupt flag is set. All interrupt requests are sampled at the rising edge of the clock and can be serviced by the processor one clock cycle later, assuming the request does not hit the interrupt exception window. The one-cycle stall between detection and acknowledgement/servicing is due to the fact that the current instruction may also be accessing the stack. For this reason, the CPU must allow the current instruction to complete before pushing the stack and vectoring to IV. If an interrupt exception window is generated by the currently executing instruction, the following instruction must be executed, so the interrupt service routine will be delayed an additional cycle.

Interrupt operation in the DS4830A CPU is essentially a state machine generated long CALL instruction. When the interrupt handler services an interrupt, it temporarily takes control of the CPU to perform the following sequence of actions:

1. The next instruction fetch from program memory is cancelled.
2. The return address is pushed on to the stack.
3. The INS bit is set to 1 to prevent recursive interrupt calls.
4. The instruction pointer is set to the location of the interrupt service routine (contained in the Interrupt Vector register).
5. The CPU begins executing the interrupt service routine.

Once the interrupt service routine completes, it should use the RETI instruction to return to the main program. Execution of RETI involves the following sequence of actions:

1. The return address is popped off the stack.
2. The INS bit is cleared to 0 to re-enable interrupt handling.
3. The instruction pointer is set to the return address that was popped off the stack.
4. The CPU continues execution of the main program.

Pending interrupt requests will not interrupt an RETI instruction; a new interrupt will be serviced after first being acknowledged in the execution cycle which follows the RETI instruction and then after the standard one stall cycle of interrupt latency. This means there will be at least two cycles between back-to-back interrupts.

5.3.1 – Synchronous vs. Asynchronous Interrupt Sources

Interrupt sources can be classified as either asynchronous or synchronous. All internal interrupts are synchronous interrupts. An internal interrupt is directly routed to the interrupt handler that can be recognized in one cycle. All external interrupts are asynchronous interrupts by nature. When the device is not in stop mode, asynchronous interrupt sources are passed through a 3-clock sampling/glitch filter circuit before being routed to the interrupt handler. The sampling/glitch filter circuit is running on the system clock. An interrupt request with a pulse width less than three system clock cycles is not recognized. Note that the granularity of interrupt source is at module level. Synchronous interrupts and sampled asynchronous interrupts assigned to the same module produce a single interrupt to the interrupt handler.

5.3.2 – Interrupt Prioritization by Software

All interrupt sources of the DS4830A naturally have the same priority. However, when CPU operation vectors to the programmed Interrupt Vector address, the order in which potential interrupt sources are interrogated is left entirely up to the user, as this often depends upon the system design and application requirements. The Interrupt Mask system register provides the ability to knowingly block interrupts from modules considered to be of lesser priority and manually re-enable the interrupt servicing by the CPU (by setting INS = 0). Using this procedure, a given interrupt service routine can continue executing, only to be interrupted by higher priority interrupts. An example demonstrating this software prioritization is provided in the Handling Interrupts section of Section 19: Programming.

5.3.3 – Interrupt Exception Window

An interrupt exception window is a noninterruptible execution cycle. During this cycle, the interrupt handler does not respond to any interrupt requests. All interrupts that would normally be serviced during an interrupt exception window are delayed until the next execution cycle.

Interrupt exception windows are used when two or more instructions must be executed consecutively without any delays in between. Currently, there is a single condition in the DS4830A that causes an interrupt exception window: activation of the prefix (PFX) register.

When the prefix register is activated by writing a value to it, it retains that value only for the next clock cycle. For the prefix value to be used properly by the next instruction, the instruction that sets the prefix value and the instruction that uses it must always be executed back to back. Therefore, writing to the PFX register causes an interrupt exception window on the next cycle. If an interrupt occurs during an interrupt exception window, an additional latency of one cycle in the interrupt handling will be caused as the interrupt will not be serviced until the next cycle.

SECTION 6 – DIGITAL-TO-ANALOG CONVERTER (DAC)

The DS4830A contains eight 12-bit digital-to-analog converters (DACs). Each DAC has a voltage output buffer. Each DAC can independently select between a 2.5V internal reference and external reference at REFINA pin for DAC0 to DAC3 and at REFINB pin for DAC4 to DAC7.

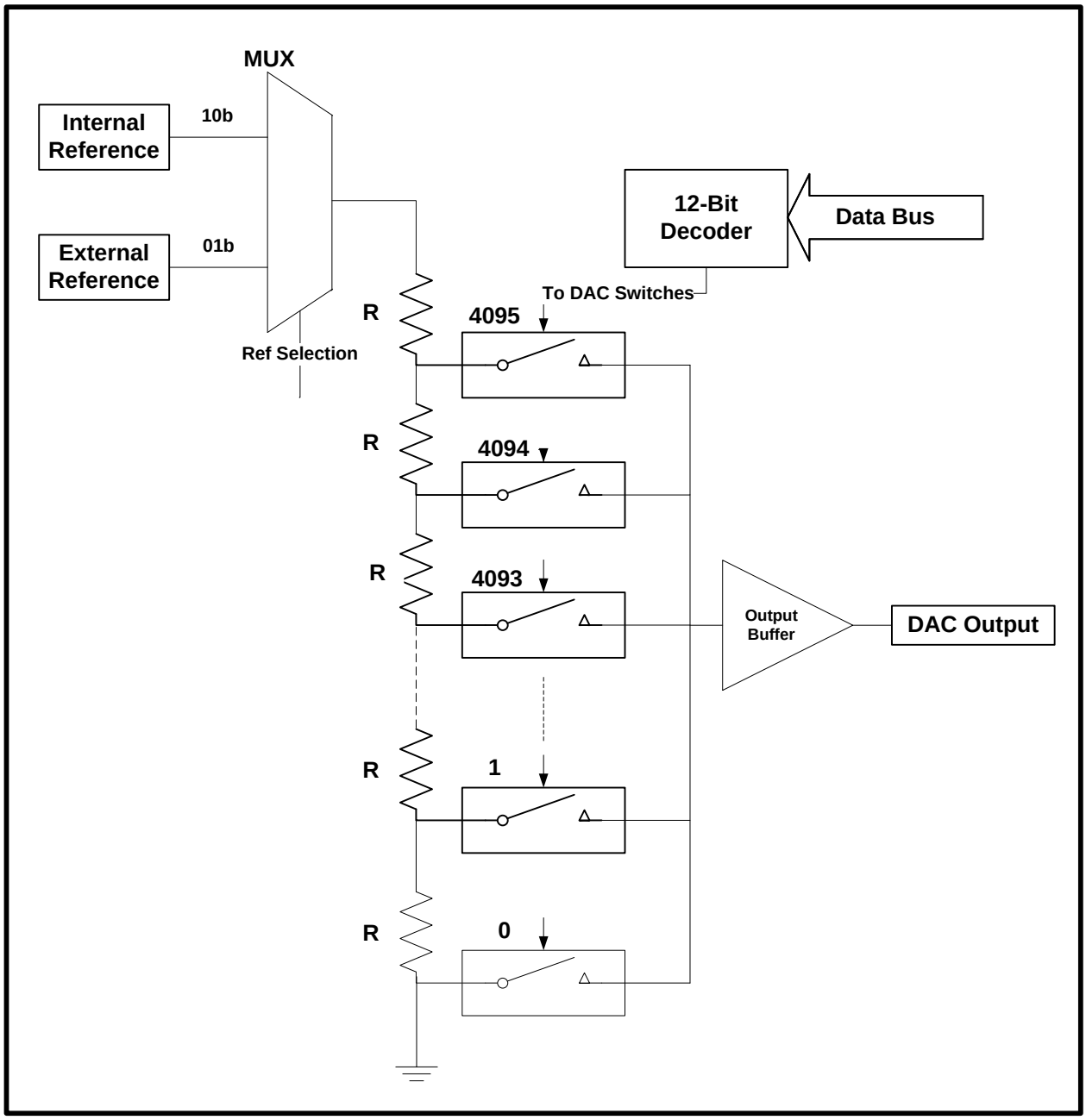


Figure 6-1: DAC Functional Diagram

6.1 – Detailed Description

The DS4830A DAC architecture consists of a resistor string with switches and decoder followed by a voltage buffer. The DS4830A has eight independent DACs, each having the same architecture. As shown in Figure 6-1, each DAC's reference is software selectable. Each DAC is independently configurable using the DAC configuration and DAC data registers. The DAC configuration register (DACCFG) provides the facility to enable or disable DACs independently and select the reference. Each DAC can be configured for either an internal (2.5V) or an external reference.

The DAC Data register programs the DAC for a particular voltage output depending on the value of this register and the reference setting. The DAC outputs are voltage buffered and have the capability to sink or source current. Each DAC output has output impedance which limits the DAC operating range if configured to sink current (refer to the

DS4830A IC data sheet). The DAC output voltage is maintained during any type of reset except POR. All DACs, REFINA and REFINB pins default to GPIO on reset.

6.1.1 – Reference Selection

Each DAC can be independently enabled with 2.5V internal reference or external reference. Each DAC has two bits in the DAC configuration register (DACCFG) that are used to enable or disable the DAC with either an internal or an external reference.

Any DAC can be enabled for using the internal reference by writing 10b at the corresponding location in the DACCFG register. The internal reference automatically powers-down when none of the 8 DACs use it as a reference source.

The external reference at REFINA (Port2.6) is selected by writing 01b at the corresponding location in the DACCFG for DAC0-3. The REFINA automatically becomes GPIO when none of the lower 4 DACs (DAC0 to DAC3) use REFINA as its reference. The external reference at REFINB (Port1.4) is selected by writing 01b at the corresponding location in the DACCFG register for DAC4-7. The REFINB pin automatically becomes GPIO when none of the upper 4 DACs (DAC4 to DAC7) use REFINB as its reference. The DAC internal or external references can be measured at the ADC. See ADC section for further detail information.

6.2 – DAC Register Descriptions

The DAC module has total 9 SFR registers. These are DAC Configuration register DACCFG and 8 DAC Data registers DACDx (DACD0 to DACD7). The DACCFG configures all DACs and the data register DACDx (DACD0-DACD7) controls the corresponding DAC output voltage. These SFRs are located in module 4.

6.2.1 – DAC Configuration Register (DACCFG)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	DACCFG7[1:0]		DACCFG6[1:0]		DACCFG5[1:0]		DACCFG4[1:0]		DACCFG3[1:0]		DACCFG2[1:0]		DACCFG1[1:0]		DACCFG0[1:0]	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:8	DACCFG7[1:0] DACCFG6[1:0] DACCFG5[1:0] DACCFG4[1:0]	**DAC Configuration:** These bits configure DAC7-4 and select the DAC reference for DAC7-4 when the corresponding DAC is enabled. **DACCFGx[1:0]** / **DACx Control/Reference Select** 00 – DACx is Disabled and is in power down mode. 01 – DACx is enabled and REFINB is selected as the external reference. To use the external reference, the REFB_CFG bit in the RPCFG register must be set to '1'. 10 – DACx is enabled and the 2.5V Internal Reference is selected as the DAC reference 11 – Reserved. (User should not write this value+) PIN 39 is REFINB (Port1.4).
7:0	DACCFG3[1:0] DACCFG2[1:0] DACCFG1[1:0] DACCFG0[1:0]	**DAC Configuration:** These bits configure DAC3-0 and select the DAC reference for DAC3-0 when DAC enabled. **DACCFGx[1:0]** / **DACx Control/Reference Select** 00 – DACx is Disabled and is in power down mode. 01 – DACx is enabled and REFINA is selected as the external reference. To external reference, the REFA_CFG bit in the RPCFG register must be set to '1'. 10 – DACx is enabled and the 2.5V Internal Reference is selected as the DAC reference 11 – Reserved. (User should not write this value+) PIN 31 is REFINA (Port2.6).

6.2.2 – DAC Data Registers (DACD0-DACD7)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	DACDx[11:0]											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:12	-	**Reserved**. The user should write zero to these bits.
11:0	DACDx*[11:0]	**DACDx:** These bits set the DACx output voltage according to reference selection and reference value. DACx Output voltage (in Volts) = (DAC Count / 4095) * Reference Voltage (in Volts)

* 'x' = 0 to 7

6.2.3 – Reference Pin Configuration Register (RPCFG)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	-	-	-	-	-	-	REFB_CFG	REFA_CFG
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rw	rw

BIT	NAME	DESCRIPTION
15:2	-	**Reserved**. The user should not write to these bits.
1	REFB_CFG	**REFINB Pin Configuration:** Setting this bit to '1' configures the DAC external reference pin for any DAC4-7 as analog input. PIN 39 is REFINB (Port1.4).
0	REFA_CFG	**REFINA Pin Configuration:** Setting this bit to '1' configures the DAC external reference pin for any DAC0-3 as analog input. PIN 31 is REFINA (Port2.6).

6.3 – DAC Code Examples

6.3.1 – DAC0 Enabled with Internal Reference and Output Voltage Configured for 50% (1.25V) of Internal Reference

```
RPCFG = 0x0000;
DACCFG = 0x0002;     //Only DAC0 enabled and internal reference is selected
DACD0 = 0x0800;      //DACD0 is set for 50%
```

6.3.2 – DAC2 Enabled with External Reference And Output Voltage Configured for 25% of External Reference at REFINA Pin

```
RPCFG = 0x0001;
DACCFG = 0x0010;    //Only DAC2 enabled and external reference is selected
DACD2 = 0x0400;     //DACD2 is set for 25%
```

6.3.3 – DAC6 Enabled with External Reference and Output Voltage Configured for 25% of External Reference at REFINB Pin

```
RPCFG = 0x0002;
DACCFG = 0x1000;    //Only DAC6 enabled and external reference is selected
DACD6 = 0x0400;     //DACD6 is set for 25%
```

SECTION 7 – ANALOG-TO-DIGITAL CONVERTER (ADC)

The DS4830A provides a 13-bit analog-to-digital converter (ADC) with 26-input MUX. As shown in Figure 7-1, the MUX selects the ADC input from 16 external channels, DAC external references at REFINA and REFINB, V_{DD}, DAC Internal Reference, Internal Die Temperature, Sample and Hold at GP2-GP3 and GP12-GP13 and ADC Internal Offset. The ADC external channels can operate in differential voltage mode or in single-ended voltage mode. An internal channel is used exclusively to measure the die temperature. The REFINA and REFINB pins can be used as analog channel independent to the DAC reference.

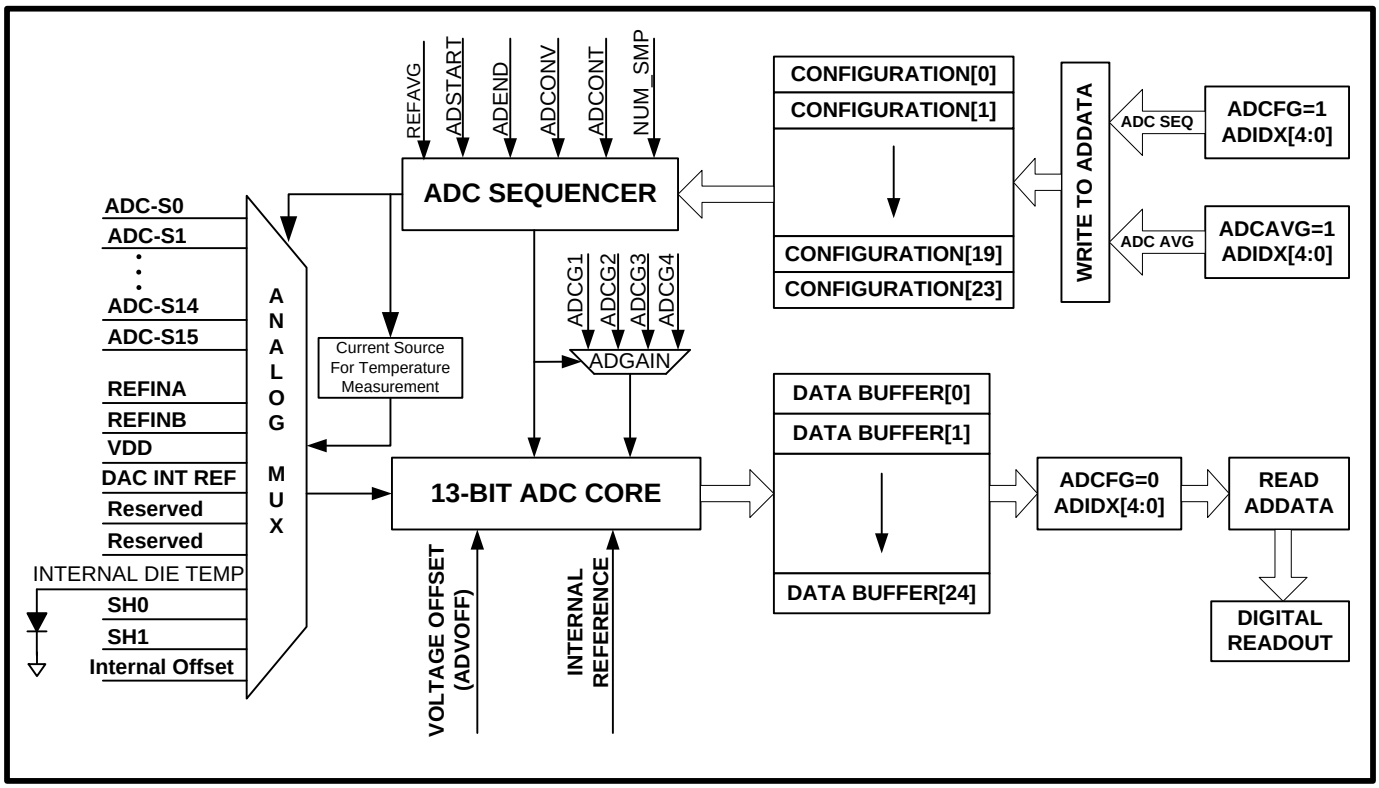


Figure 7-1: ADC Functional Diagram

7.1 – Detailed Description

7.1.1 – ADC Controller

The ADC controller is the digital interface block between CPU and the ADC. It provides all necessary controls to the ADC and the CPU interface. The ADC controller provides 25 buffers (0-24) for various configurations and data buffers. By default, the ADC conversion result corresponding to each channel is placed in data buffers at the location shown in Table 7-1. The user can override the default buffer locations and define alternate locations in the ADC Data and Configuration register (ADDATA) during configuration by settling the LOC_OVR bit to '1' in the ADC Control register (ADCN). The internal temperature sensor and Sample and Hold (S/H) use fixed data buffer locations and these locations should not be used for other channels if these peripherals are enabled. The ADC internal offset does not have any data buffer and its measurement is performed with location override enable. Table 7-1 has the default configuration and data buffer locations. The ADC controller provides various internal averaging options for individual ADC channels, internal die temperature and S/H. See Section 7.1.9 for ADC Averaging.

Table 7-1: ADC Configuration and Data Buffers

DATA BUFFER	CONFIGURATION/DATA BUFFER SELECTION
0-15	External Channels (0-15 in single-ended or 0-7 in differential)
16	REFINA
17	REFINB
18	V_{DD} (Supply Voltage)
19	DAC Internal Reference
20-21	Reserved, can be used with Location Override
22	Internal Die Temperature
23	Sample and Hold 0
24	Sample and Hold 1
0-24 (Any)	ADC Internal Offset (with Location Override)

By default, the external channels GP0-15 are general-purpose inputs. The DS4830A has the Pin Select Register (PINSEL) which is used to configure these external channels as analog pins for ADC or/and Quick Trip use. Each bit location in this register corresponds to the ADC/QT input pin. The ADC controller uses a set of Special Function Registers (SFRs) to configure the ADC for the desired mode of operation. The DS4830A ADC can operate in the three modes mentioned below.

1. ADC Sequence Mode Conversions
2. Temperature Mode Conversions
3. Sample and Hold Mode Conversions

7.1.2 – ADC Conversion Sequencing

The DS4830A ADC controller performs a user defined sequence for up to 16 single-ended or 8 differential external voltage channels. Additionally, the ADC controller allows the user to measure voltages of the DAC internal and external references (REFINA and REFINB) and V_{DD} . The REFINA and REFINB can be used as analog channels independent of DAC operation. Thus the DS4830A provides 18 analog channels for application usage. The ADC controller provides 24 ADC internal configuration and averaging configuration registers. The configuration registers are accessed by writing to the ADDATA register when ADST.ADCFG = 1 and ADST.ADCAVG = 0. The averaging configuration registers are accessed by writing to the ADDATA register when ADST.ADCAVG = 1 and ADST.ADCFG = 0. Each conversion in a sequence is setup using one of the ADC configuration and averaging configuration registers. The results from the ADC converter are located in the 25 data buffers. These are accessed by reading from the ADDATA register when ADST.ADCFG = 0 and ADST.ADCAVG = 0. See Figure 7-2 for ADC configurations and data buffers.

The configuration register pointed to by ADDATA is selected using the ADIDX bits in the ADST register when ADCFG = 1 and ADCAVG = 0. The individual configuration registers allows each of the conversions in a sequence to select from the following options.

- ADC channel selection
- Differential or single-ended conversion
- Full scale range
- Extended acquisition enable
- ADC conversion data alignment (left or right)
- Alternate location

For more information, see the configuration register description for the ADDATA register.

A sequence is setup in the ADC Address register (ADADDR) by defining the starting conversion configuration address (ADSTART) and an ending conversion configuration address (ADEND). The configuration start address designates the configuration register to be used for the first conversion in a sequence. The configuration end address designates the configuration register used for the last conversion in a sequence. A single channel conversion can be viewed as a special case for sequence conversion, where the starting and ending configuration address is the same. The configuration registers can be viewed as a circular register array where ADSTART does not have to be less than ADEND. For example, if ADSTART = 1 and ADEND = 5, then the sequence of conversions would be configurations 1, 2, 3, 4, 5. If ADSTART = 5 and ADEND = 1, then the sequence of conversions would be configurations 5, 6, 7 . . . 23 , 0, 1.

The ADC has two conversion sequence modes, single and continuous which are set by the ADCONT bit. When the start conversion bit (ADCONV) is set to '1', the ADC controller starts the ADC conversion sequence. In single sequence mode (ADCONT=0), the ADCONV bit remains set until the ADC has finished the conversion of the last channel in the sequence. In continuous mode (ADCONT=1), the ADCONV bit remains set until the continuous mode

is stopped. Writing a '0' to the ADCONV bit stops the ADC operation at the completion of the current ADC conversion. Writing a '1' to the ADCONV bit when ADCONV bit is already set to '1' is ignored by the ADC controller.

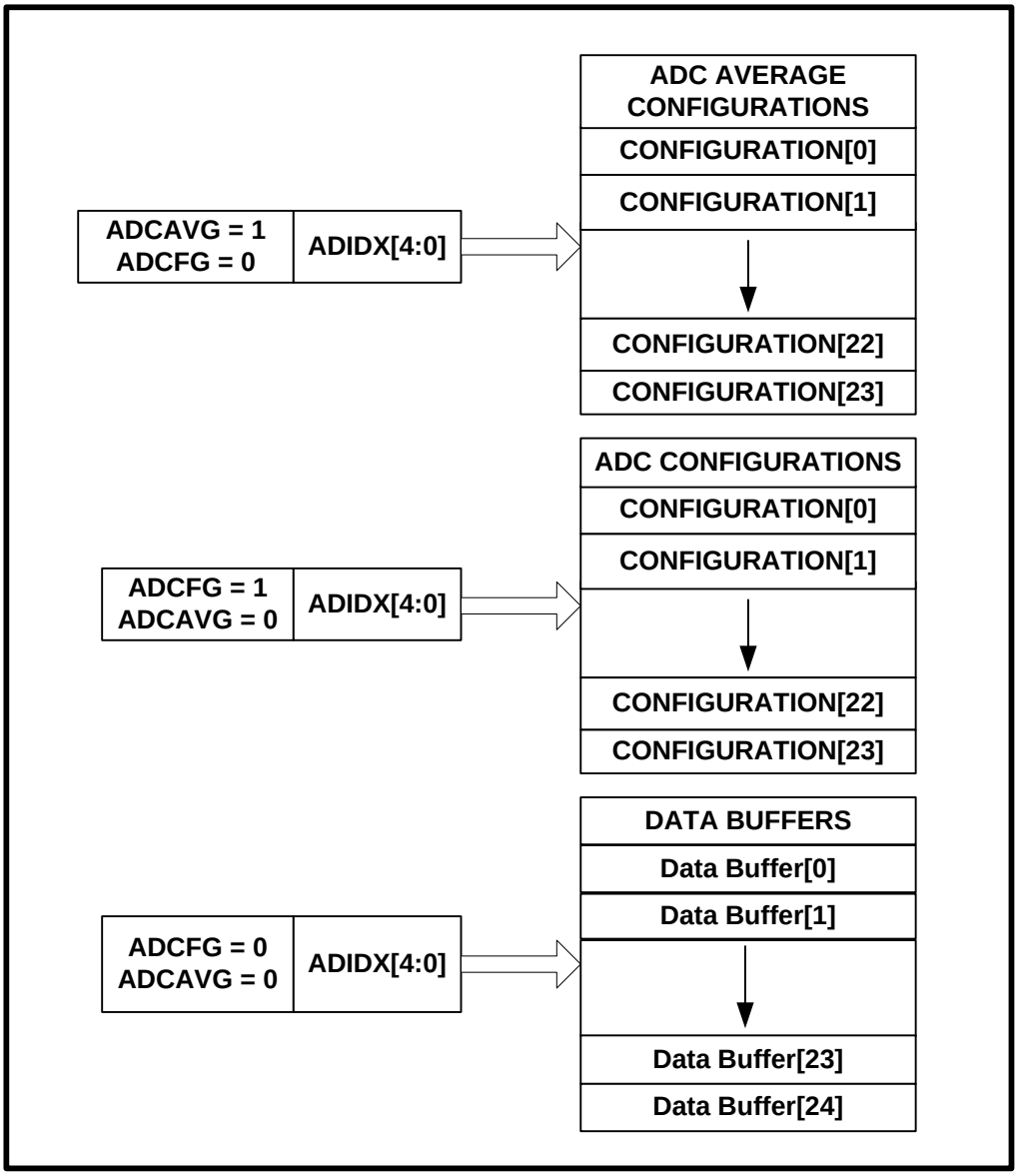


Figure 7-2: ADC Configurations and Data Buffers

Note: With location override enabled, a single channel can be added multiple times as demonstrated in Example 7.3.2.

7.1.3 – Internal Die Temperature Conversion

The DS4830A allows monitoring of internal die temperature. The internal temperature channel can be independently enabled by writing a '1' to the bit 0 in the Temperature Control register (TEMPCN). The internal die temperature has a temperature conversion complete flag located in the ADST register. Data buffer 22 is reserved for the result of the internal die temperature sensor. The TEMPCN register has separate bits for interrupt enable and data alignment.

A DS4830A temperature conversion provides 0.062 °C of resolution. The time required for a temperature conversion is approximately 42µsec at the default ADC Clock. If temperature conversion is enabled simultaneously with voltage conversions, the temperature conversion gets time slots at the end of ADC sequence. See Figure 7-3 ADC Frame Sequence for more details.

Note: If only internal temperature conversions are being performed (no voltage or sample/hold conversions are enabled), to disable the temperature conversion, a dummy ADC conversion must be performed by setting ADCONV=1.

7.1.4 – Sample and Hold Conversion

The DS4830A has two Sample and Hold (S/H) inputs at pins GP2-GP3 and GP12-GP13. These can be independently enabled or disabled by writing to their corresponding bit locations in the Sample and Hold Control register (SHCN). See the Sample and Hold description in Section 8. The Sample and Hold uses data buffer 23 and 24 for S/H0 and S/H1 respectively. The Sample and Hold conversion complete flags are located in the ADST register. When enabled with voltage conversions, the sample and hold conversions get time slots in between each voltage conversion. See Figure 7-3, ADC Frame Sequence for more details.

7.1.5 – ADC Frame Sequence

When all modes (voltage, temperature, and sample and hold) are used simultaneously, the ADC controller uses time slicing. The ADC controller uses the ADC sequence of voltage conversions as "primary channels" and sample and hold as secondary channels. The time slicing rules are

1. The primary channels (ADC voltage channels) have priority over the secondary channels (S/Hs).
2. S/H0 has priority over S/H1 if both S/Hs are ready for conversion. However, in next slot for S/H, the S/H1 will get slot even if S/H0 is also ready.
3. The internal die temperature gets the conversion slots at the end of ADC sequence.

For example, if the ADC sequence mode conversion is enabled for channel 0, 4, 5, 6, both S/Hs and internal die temperature are enabled and ready for conversion then the sequence of conversion is performed as shown in Figure 7-3.

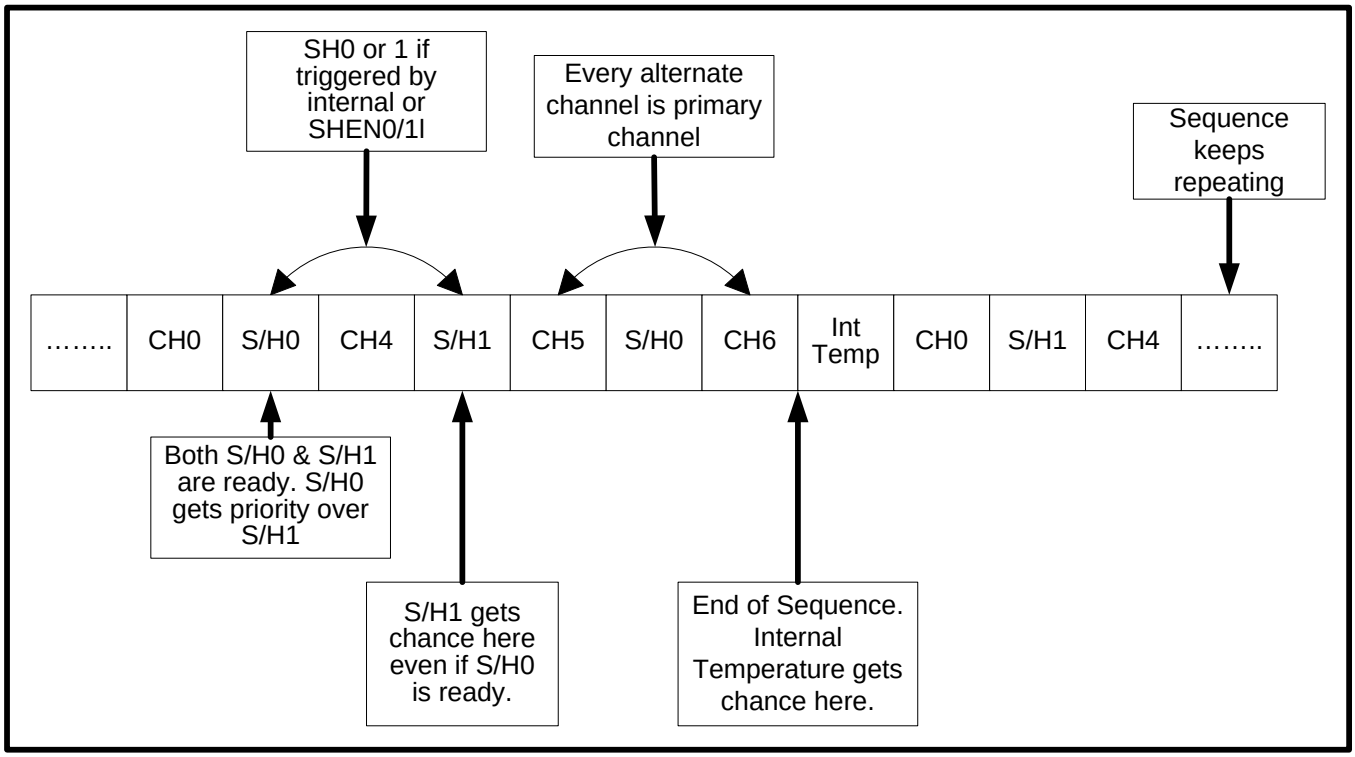


Figure 7-3: ADC Frame Sequence

Notes:

1. Both Sample and Hold channels can occur simultaneously as they have dedicated resources.
2. Averaging is disabled.

7.1.6 – ADC Reference

The ADC has a 1.2V internal reference that must be enabled before the start of ADC conversion sequence. The ADC controller provides INT_REF bit in the REFAVG register to control the ADC internal reference. By setting this bit to '1', the internal reference is enabled. The ADC internal reference needs approximate 1ms of stabilization time. The ADC conversion should be started only after this stabilization time.

The ADC controller provides an option to bring out the ADC internal reference at GP1 pin (PIN6, Port2.1). By setting REF_OUT bit in the REFAVG register and the bit 1 of the PINSEL register, the ADC internal reference is brought out at GP1 pin.

7.1.7 – ADC Conversion Time

The ADC clock is derived from the system clock with a divide ratio defined by the ADC Clock Divider Bits ADCCLK [2:0] in the ADC Control register (ADCN). Each sample takes 15 ADC clock cycles to complete. Two of the 15 ADC clock cycles are used for sample acquisition, and the remaining 13 clocks are used for data conversion. The ADC automatically reads each measurement twice and outputs the average of the two readings. This makes the resulting time for one complete conversion to be 30 ADC clock cycles. Additionally, 4 core clocks are used in data processing for each of the two readings.

Knowing this, it is possible to calculate the fastest ADC sample rate. The fastest ADC clock is:

ADC Clock = Core Clock / 8 = 10 MHz / 8 = 1250 kHz = 0.8 µs
One conversion requires 30 ADC Clocks + 8 Core Clocks
Conversion Time = (30 ADC Clocks Time+ 8 Core Clocks Time)
= 30 * 0.8 + 0.8 µs
= 24.8 µs per ADC Conversion
Sample Rate = 40.3 ksps

The ADC has an internal power management system that automatically shuts down the ADC when conversions are complete by clearing ADCONV to 0. After being shut down, the ADC begins conversions again when the ADCONV bit is set to 1 again. After ADCONV is set to 1, the ADC requires 20 ADCCLK cycles to setup and power up prior to beginning the first conversion of the sequence. So the first ADC conversion time is ~40µs at the fastest ADC Clock. If the quick trip is also enabled and if the ADC controller and the quick trip are sampling the same channel, the ADC sampling is delayed by two quick trip conversions (3.2µs) to prevent collision.

In applications where extending the acquisition time is desired, the user can make use of the ADC Acquisition Extension Bits (ADACQ[3:0] in the ADCN register). When the ADC Acquisition Extension is enabled (ADACQEN=1), the sample is acquired over a prolonged period during the sample acquisition. The extended acquisition time is determined by ADACQ[3:0]. Table 7-2 shows the extended acquisition time in terms of core clocks at different ADACQ[3:0] The total acquisition time, ACQ, is two ADC clocks plus the Extended Acquisition Time (ADACQ, as listed in Table 7-2). Figure 7-4 shows the clocking required for one conversion.

Table 7-2: Extended Acquisition Time in Terms of Core Clock and Time (µs)

ADACQ[3:0]	# of Core Clocks	Time (µs)
0	2	0.2
1	6	0.6
2	14	1.4
3	30	3.0
4	62	6.2
5	126	12.6
6	254	25.4
7	520	52
8	1032	103.2
9	2056	205.6
10	4104	410.4
11	8190	819
12	16382	1638.2
13	32766	3276.6
14	65534	6553.4
15	131070	13107

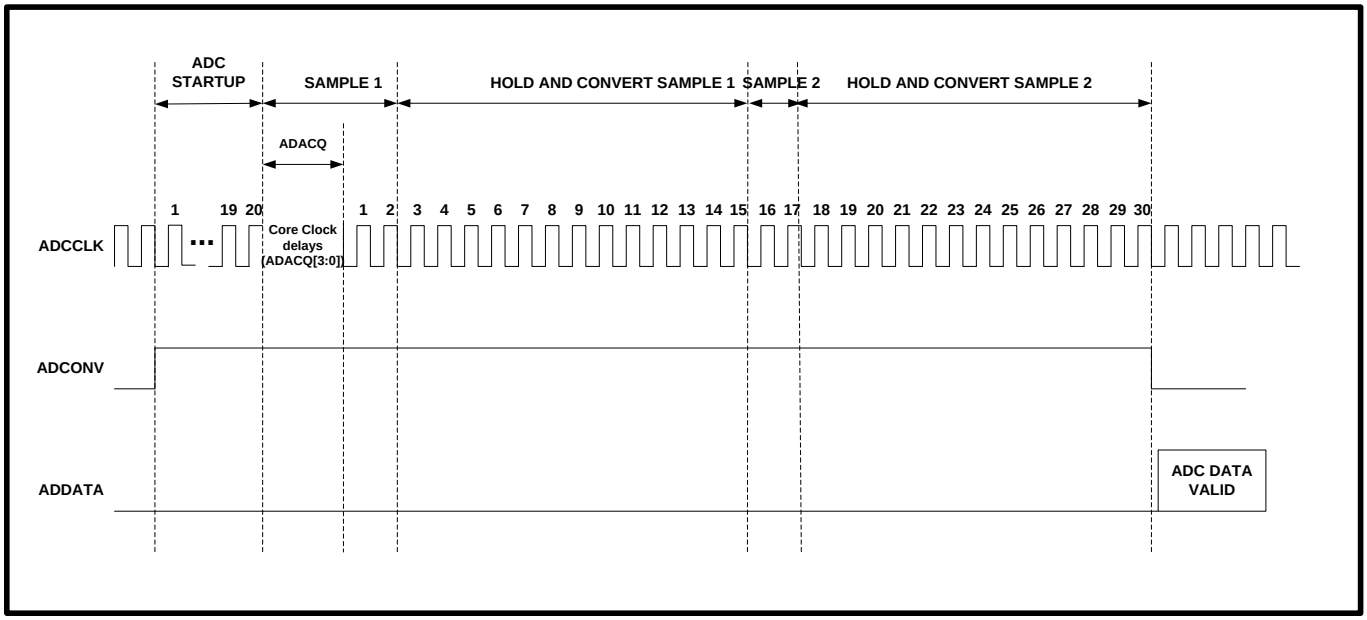


Figure 7-4: Extended Acquisition Time

7.1.8 – Location Override

By default, the ADC controller stores ADC conversion results in the ADC buffer location corresponding to the channel number (as defined in Table 7.1). The ADC controller allows the user to override the default data buffer location and store the ADC result at any of the buffer location (0-24). The location override is enabled by setting the LOC_OVR bit to '1' in the ADCN register. The user has to define the alternate location for storing the ADC conversion result during ADC configuration (when ADST.ADCFG = 1). The alternate location is defined by ADDATA[12:8] (ALT_LOC). Location override is demonstrated in Example 7.3.2,

Note: If the location override will be using the buffer locations designed for internal temperature or sample and hold, these corresponding peripherals should be disabled (as mentioned in 7.1.1). Example, if the buffer location 22 is used in the ADC sequence with the location override option, the internal die temperature should be disabled.

7.1. 9 – Averaging

The ADC controller supports various averaging options for each ADC channel, internal die temperature and S/Hs. This averaging is performed automatically by the ADC controller which reduces application overheads. The ADC controller has ADCAVG bit in the ADST register which is used to configure number of ADC samples to be averaged for each channel. When the ADCFG bit is set to 0 and ADCAVG bit is set to '1', writing to ADDATA [1:0] configures the number of ADC samples to be averaged. User can write any value between 0-3 to select 1, 4, 8 or 16 ADC samples averaged. See Section 7.1.2 for averaging configuration register and 7.3.3 for ADC averaging example code.

The ADC controller has the REFAVG register to configure different averaging options for internal die temperature and S/H. Each sample of the internal temperature is converted after the ADC sequence. See the REFAVG register description for detailed information about averaging options for internal die temperature and sample and hold channels.

When averaging configuration is enabled in the ADC sequence for ADC channels, internal die temperature and S/Hs, the ADC frame sequence is changed and explained in Figure 7-5. The ADC and S/H samples are converted back to back by the ADC controller and averaged values are reported in the data buffers. After every end of sequence, the ADC controller converts a sample of internal die temperature.

Figure 7-5 shows the ADC frame sequence for the following programmed sequence of ADC channels.

1. CH0: Average of 4 Samples
2. CH4: Average of 8 Samples
3. CH5: Average of 16 Samples
4. CH6: Average of 1 Sample
5. S/H0: Average of 2 Samples
6. S/H1: Average of 4 Samples
7. Internal Temperaute: Average of 16 Samples

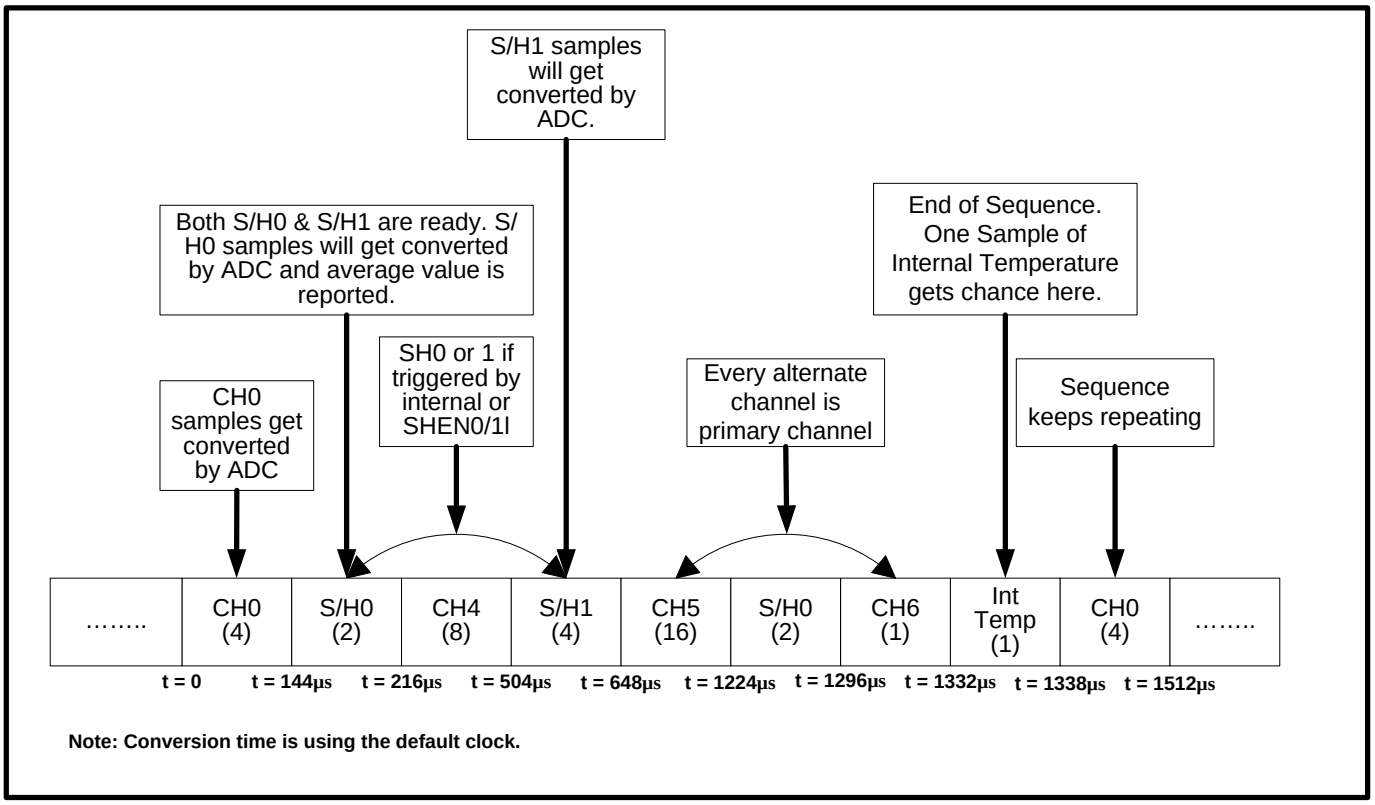


Figure 7-5: ADC Frame Sequence with Averaging

7.1.10 – ADC Data Reading

The ADC has a circular data buffer that can hold the results from 25 conversions. When the location override (LOC_OVR = 0) is disabled, the ADC controller writes the ADC conversion result at the data buffer location corresponding to ADC channel number, see Table 7-1. When location override is enabled, the ADC controller writes the result to the data buffer location configured in the ALT_LOC[4:0] bits in the ADDATA during ADC configuration (ADST.ADCFG = 1). Using the location override feature, multiple conversions for a single channel can be stored to data buffers as explained in example code 7.3.2. This buffer is accessed by reading the ADDATA register when ADCFG is set to 0. The data buffer pointed to by ADST.ADIDX [4:0] is the buffer returned when ADDATA is read. The ADIDX is automatically incremented following a read of ADDATA. This allows repeated reads of ADDATA to return the results from multiple conversions. The ADC continues writing to the data buffer until the end of the buffer. Once the end of the data buffer is reached, the ADC index rolls over and reading continues from data buffer 0.

7.1.11 – ADC Interrupts

The ADC Data Available Ready ADDAI bit in the ADST1 register is set when conversions are complete. This flag generates an interrupt if enabled by setting the ADCN.ADDAIE interrupt enable bit. The condition that causes the ADDAI flag to be set can be selected using the ADCN.ADDAINV bit.

Table 7-3: ADC Interrupt Intervals

ADDAINV	SET ADDAI AFTER
0	End of Every Sequence (ADSTART to ADEND)
1	After End of Every Sequence (ADSTART to ADEND) and After (NUM_SMP + 1) ADC Conversions

For example, if ADSTART = 0, ADEND = 6 and NUM_SMP = 3 with ADDAINV = 1, then ADDAI is set to '1' after every (NUM_SMP + 1) ADC conversions and every End of Sequence. In the given example, ADDAI is set after 4,7,8,12,14… ADC Samples. Interrupts after 4, 8 and 12 ADC Samples are because of (NUM_SMP+1) configurations and interrupts after 7 and 14 are because of "End of Sequence". Figure 7-6 demonstrates above ADC example sequence.

If ADC averaging is used, each of the converstions for an average is counted as a sample for interrupts. For example, if four samples are being averaged for each channel and interrupts are set to trigger every four converstions, then an interrupt will occur after each channel completes its four samples.

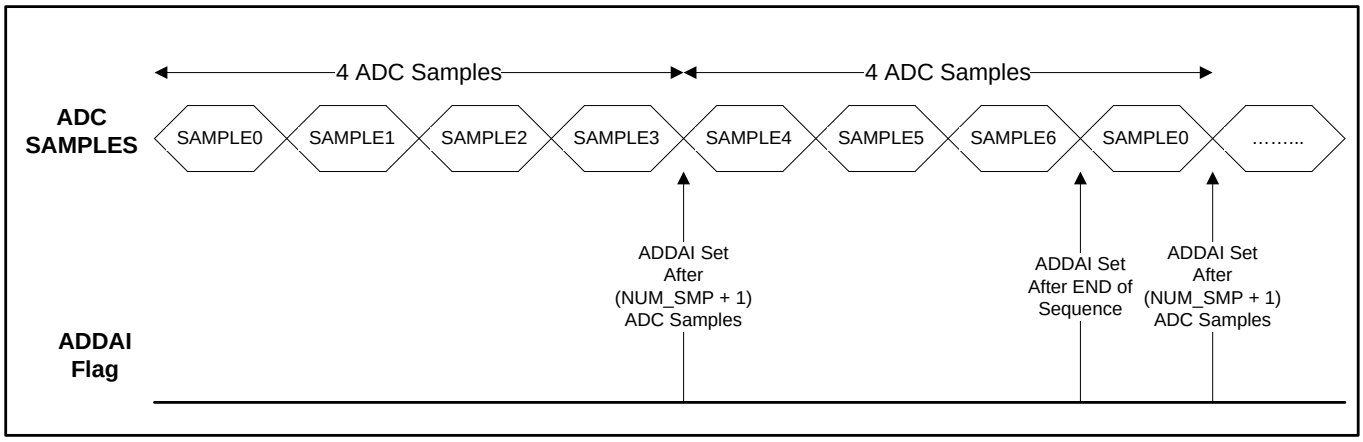


Figure 7-6: ADC Interrupt Intervals with NUM_SMP

The ADDAI flag is cleared by software by writing a '0', or it is automatically cleared when a new conversion sequence is started by setting the ADCONV bit to a '1'.

Note: The ADC controller processes ADC, internal die temperature and sample and holds conversions according to ADC frame sequence and sets the corresponding flags in the ADST1 flag. The user should process and clear an interrupt flag when it is set before another flag in the ADST1 is set by the ADC controller.

7.1.12 – ADC Internal Offset

The DS4830A ADC controller allows for ADC internal offset measurement. The ADC controller does not have a dedicated buffer for the internal offset so it can only be accessed with location override enabled. For measurement of ADC internal offset, the ADC controller connects internal ground to the ADC input and performs an ADC conversion. Using this feature, software can calibrate the ADC internal offset.

Refer to Application Note 5321: Calibrating the ADC Internal Offset of the DS4830 Optical Microcontroller.

7.1.13 – DAC External Reference Pins (REFINA and REFINB) as ADC Channels

The DS4830A provides an option to measure the voltage applied to the DAC external reference pins REFINA and REFINB without enabling any DACs. The ADC controller has RPCFG register to configure REFINA and REFINB as analog pins. This allows flexibility to use the REFINA and REFINB pins as two additional analog input channels and can also be used as DAC external reference.

7.1.14 – Fast Conversion Mode (ADST.ENABLE_2X)

The DS4830A ADC controller can be used in fast mode to reduce the sample conversion time. The Enable_2x bit in ADST register has to be set to 1 to use the ADC in fast mode. In normal operating mode, the ADC reads two input samples and outputs the average of the results of both the samples. In Fast conversion mode, the ADC reads only one input sample and outputs the result as such. The ADC conversion time is reduced by half when operating in fast mode.

7.2 – ADC Register Descriptions

The ADC is controlled by the ADC SFR registers. The PINSEL register is used to configure pins as analog pins for ADC use. Six of the registers, ADST, ADST1, ADADDR, ADCN, RPCFG, REFAVG and ADDATA are used for setup, control, and reading from the ADC. Registers ADCG1-4 and ADVOFF which are used to adjust the gains and offsets applied to ADC results. To avoid undesired operations, the user should not write to bits labeled as "Reserved".

7.2.1 – ADC Control Register (ADCN)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	ADCCLK[2:0]			NUM_SMP[4:0]					ADDAINV	ADCONT	ADDAIE	LOC_OVR	ADACQ[3:0]			
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*

* Unrestricted read, but can only be written to when ADCONV = 0 except ADDAIE bit.

BIT	NAME	DESCRIPTION
15:13	ADCCLK[2:0]	**ADC Clock Divider.** These bits select the ADC conversion clock in relationship to the Core Clock.
12:8	NUM_SMP[4:0]	**Interrupt After Number of Sample.** These bits define the Number of ADC samples required for an ADC interrupt when ADDAINV = 1. If ADDAINV is set to '1', then ADC Interrupt occurs after (NUM_SMP + 1) ADC samples and End of Sequence.
7	ADDAINV	**ADC Data Available Interrupt Interval.** This bit selects the condition for setting the data available interrupt flag (ADDAI). When ADDAINV = 0, ADDAI is set after End of Sequence. When ADDAINV = 1, ADDAI is set after End of Sequence and after ADC Samples = (NUM_SMP + 1).
6	ADCONT	**ADC Continuous Sequence Mode.** Setting this bit to '1' enables the continuous sequence mode. Clearing this bit to '0' disables the continuous sequence mode. In single sequence mode, the ADC conversion is stopped after the end of the sequence. The user should set this bit to '1', when temperature and sample and hold are also enabled.
5	ADDAIE	**ADC Data Available Interrupt Enable.** Setting the ADDAIE bit to '1' enables an interrupt to be generated when the ADDAI=1. Clearing this bit to '0' disables an interrupt from generating when ADDAI=1. This bit is unconditional writable.
4	LOC_OVR	**Location override bit**. Setting this bit to '1' enables the user to select an alternate location for storing ADC conversion results. The alternate location is defined by ADDATA[12:8] (ALT_LOC). By default, the ADC conversion results are stored in ADC buffer location corresponding to channel number. See Table 7-1.
3:0	ADACQ[3:0]	**ADC Acquisition Extension Bits [3:0].** These bits are used to extend sample acquisition time if the corresponding ADC Acquisition Extension is enabled (ADDATA.ADACQEN =1 when ADST.ADCFG is set to '1'). See ADC Conversion Time Section for details. The ADC acquisition extension should not be used when the fast comparator is used for the same channel.

ADCCLK[2:0]	ADC Clock
000	System Clock/8
001	System Clock/10
010	System Clock/12
011	System Clock/14
100	System Clock/16
101	System Clock/18
110	System Clock/20
111	System Clock/40

7.2.2 – ADC Status Register (ADST)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	ENALE_2X	-	-	-	ADCAVG	ADCONV	ADCFG	ADIDX[4:0]				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:12	-	**Reserved**. The user should not write to these bits.
11	ENABLE_2X	**ADC Fast Conversion Mode.** When ADST.ENABLE_2X = 1, the ADC operates in the fast mode. If reset to 0, normal conversion mode is used.
7	ADCAVG	**ADC Average Configuration Register Select.** When ADCAVG = 1 and ADCFG = 0, the ADDATA register points to the ADC Channel averaging configuration registers which allow configuration of averaging for each ADC channel. See 7.2.6.2 for ADC sample average configurations.
6	ADCONV	**ADC Start Conversion.** Setting this bit to '1' starts the ADC conversion process. This bit remains set until the ADC conversion process is finished. In single sequence mode, this bit is cleared to '0' when the ADC conversion sequence is finished. In continuous sequence mode, this bit remains set until the ADC conversion is stopped. To stop ADC conversion at any time, write '0' to this bit. The ADC stops acquiring data after the current conversion is finished or if the ADC is waiting during extended acquisition time, the ADC stops immediately.
5	ADCFG	**ADC Conversion Configuration Register Select.** ADCFG = 0: The ADDATA register points to the data buffers. The ADIDX[4:0] bits determine which data buffer is currently being accessed. When ADCFG=0 and ADCAVG = 0, ADDATA is read only. ADCFG = 1: The ADDATA register points to the ADC sequence configuration registers. The ADIDX[4:0] bits determine which configuration register is currently being accessed. When ADCFG=1, ADDATA has read/write access.
4:0	ADIDX[4:0]	**ADC Register Index Bits [4:0].** These bits together with ADCFG and ADCAVG select the source / destination for ADDATA access. This register value is auto-incremented on successive access (read/write) of ADDATA register. When ADCFG=1, ADIDX [4:0] are used to address one of 24 configuration registers. When ADCFG=0, ADIDX [4:0] are used to select one of 25 data buffers. ADCFG=1, ADCAVG=0: ADIDX[4:0] used to address one of 24 configuration registers ADCFG=0, ADCAVG=1: ADIDX[4:0] used to address one of 24 average configurations ADCFG=0, ADCAVG=0: ADIDX[4:0] used to select one of 25 data buffers.

7.2.3 – PIN Select Register (PINSEL)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	PINSEL[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Each bit in this register corresponds to an ADC input pin. When these bits are set the corresponding pins are dedicated for ADC use. On POR, the pin selection register is 0000h which corresponds to GP0 to GP15 being GPIO. For using these pins as ADC input, Sample and Hold or Quick Trip inputs the corresponding PINSEL bit should be set to '1'.

7.2.4 – ADC Status Register (ADST1)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-		-	-	-	-	-	-	SH1DAI	SH0DAI	-	-	INTDAI	ADDAI
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	rw	rw	r	r	rw	rw

BIT	NAME	DESCRIPTION
15:6	-	**Reserved**. The user should not write to these bits.
5	SH1DAI	**Sample and Hold 1 Data Available Interrupt Flag.** This bit is set to '1' when Sample and Hold is completed on GP12-GP13 in dual mode and data is ready at buffer location 24. This flag causes an interrupt if the SH1DAI_EN (SHCN.5) is set to '1'. This bit is cleared by software writing a '0'.
4	SH0DAI	**Sample and Hold 0 Data Available Interrupt Flag.** This bit is set to '1' when Sample and Hold is completed on GP2-GP3 if only S/H0 is used or after completion of S/H1 conversion on GP12-GP13 when both are used in single mode. The S/H0 and S/H1 data is ready at buffer location 23 and 24 respectively. This flag causes an interrupt if the SH0DAI_EN (SHCN.1) is set to '1'. This bit is cleared by software writing a '0'.
3:2	-	**Reserved**. The user should not write to these bits.
1	INTDAI	**Internal Temperature Data Available Interrupt Flag.** This bit is set to '1' when an internal temperature conversion is complete and data is ready in buffer location 22. This flag causes an interrupt if the INT_IEN (TEMPCN.10) is enabled. This bit is cleared by software writing a '0'.
0	ADDAI	**ADC Data Available Interrupt Flag.** This bit is set to '1' when the condition matching ADDAINV bit is met. This flag causes an interrupt if the ADDAIE bit is set. This bit is cleared by software writing a '0' or when software changes ADCONV bit from '0' to '1'.

7.2.5 – ADC Address Register (ADADDR)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	ADSTART[4:0]					-	-	-	ADEND[4:0]				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	rw*	rw*	rw*	rw*	rw*	r	r	r	rw*	rw*	rw*	rw*	rw*

* Unrestricted read, but can only be written to when ADCONV = 0.

BIT	NAME	DESCRIPTION
15:13	-	**Reserved**. The user should not write to these bits.
12:8	ADSTART[4:0]	**ADC Conversion Configuration Start Address Bits [4:0].** These bits select the first conversion configuration register.
7:5	-	**Reserved**. The user should not write to these bits.
4:0	ADEND[4:0]	**ADC Conversion Configuration Ending Address Bits [4:0].** These bits select the last conversion configuration register. This register is inclusive when defining the sequence.

7.2.6 – ADC Data and Configuration Register (ADDATA)

The ADDATA register is used to setup the ADC sequence configurations and also to read the results of the ADC conversions. If the ADST.ADCFG bit is set to a 1 and ADST.ADCAVG = 0, writing to ADDATA writes to one of the configuration registers. If ADST.ADCFG is set to 0 and ADST.ADCAVG is set to 1, writing to ADDATA writes to one of the averaging configuration registers. If ADST.ADCFG and ADST.ADCAVG is set to 0, reading from ADDATA reads one of the conversion results.

7.2.6.1 – ADC Configuration Register (ADDATA when ADCFG = 1 and ADCAVG = 0)

When ADCFG = 1 and ADCAVG = 0, writing to the ADDATA register writes to one of the configuration registers. The configuration register written to is selected by the ADIDX[4:0] bits. The ADIDX[4:0] bits are automatically incremented after a write to ADDATA. This allows consecutive writes of ADDATA to setup consecutive configuration registers. The configuration registers are reset to '0' on all forms of reset.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	ADGAIN[1:0]		ALT_LOC[4:0]					ADACQEN	ADALIGN	ADDIFF	ADCH[4:0]				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*

* When ADCFG = 1, unrestricted read, but can only be written to when ADCONV = 0.

BIT	NAME	DESCRIPTION
15	-	**Reserved**. The user should not write to this bit.
14:13	ADGAIN[1:0]	**ADC Gain Select.** This bit selects the ADC scale factor. **ADGAIN[1:0]** / **ADC SCALE** / **Full Scale (typ)** 00 / ADCG1 / 1.2V 01 / ADCG2 / 0.6V 10 / ADCG3 / 2.4V 11 / ADCG4 / 6.55* * When the ADCG4 select, the ADC input should not be above 3.6V. It is limited by V_{DD} operating range.
12:8	ALT_LOC[4:0]	**Alternate location for conversion result.** These bits specify the alternate location for storing the ADC conversion result when LOC_OVR bit in the ADCN register is set to '1'.
7	ADACQEN	**ADC Acquisition Extension Enable.** Setting this bit to '1' enables additional acquisition time to be inserted prior to this conversion. Clearing this bit to '0' disables the extended acquisition time.
6	ADALIGN	**ADC Data Alignment Select.** This bit selects the ADC data alignment mode. Setting this bit to '1' returns ADC data left aligned in ADDATA [15:2] with ADDATA[1:0] zero padded. Clearing this bit to '0' returns ADC data in right aligned format in ADDATA[13:0] with ADDATA[15:14] sign-extended by ADDATA[13].
5	ADDIFF	**ADC Differential Mode Select.** This bit selects the ADC conversion mode. When this bit is set to '1', the ADC conversion is in differential mode. When this bit is cleared to '0', the ADC conversion is performed in single-ended mode. In single-ended mode, the sample is measured between the ADC Channel and ground.
4:0	ADCH[4:0]	**ADC Channel Select.** These bits select the input channel source for configuration of ADC conversion. **ADCH [4:0]** / **ADDIFF = 0** / **ADDIFF=1** 00000 / ADC-S0 / ADC-D0P- ADC-D0N 00001 / ADC-S1 / ADC-D1P- ADC-D1N 00010 / ADC-S2 / ADC-D2P- ADC-D2N 00011 / ADC-S3 / ADC-D3P- ADC-D3N 00100 / ADC-S4 / ADC-D4P- ADC-D4N 00101 / ADC-S5 / ADC-D5P- ADC-D5N 00110 / ADC-S6 / ADC-D6P- ADC-D6N 00111 / ADC-S7 / ADC-D7P- ADC-D7N 01000 / ADC-S8 / NOT VALID 01001 / ADC-S9 / NOT VALID 01010 / ADC-S10 / NOT VALID 01011 / ADC-S11 / NOT VALID 01100 / ADC-S12 / NOT VALID 01101 / ADC-S13 / NOT VALID 01110 / ADC-S14 / NOT VALID 01111 / ADC-S15 / NOT VALID 10000 / ADC-REFINA / ADC-REFINA 10001 / ADC-REFINB / ADC-REFINB 10010 / VDD / VDD 10011 / DAC_INT_REF / DAC_INT_REF 10100- 11000 / NOT VALID / NOT VALID 11001 / ADC OFFSET / ADC OFFSET

7.2.6.2 – ADC Average Register (ADDATA when ADCAVG = 1 and ADCFG = 0)

When ADCAVG = 1 and ADCFG = 0, writing to the ADDATA register writes to one of the averaging configuration registers. The averaging configuration register written to is selected by the ADIDX[1:0] bits. The ADIDX[1:0] bits are automatically incremented after a write to ADDATA. This allows consecutive writes of ADDATA to setup consecutive average registers. The average registers are reset to '0' on all forms of reset.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	-	-	-	-	-	-	AVG[1:0]	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rw*	rw*

* When ADCAFG = 1, unrestricted read, but can only be written to when ADCONV = 0.

BIT	NAME	DESCRIPTION
15:2	-	**Reserved**. The user should not write to these bits.
1:0	AVG[1:0]	**ADC Average Select:** These bits select number of ADC samples to be averaged by the ADC controller.

AVG[1:0]	Samples Average
00	1
01	4
10	8
11	16

7.2.6.3 – ADC Data Buffer (ADDATA when ADCFG = 0 and ADCAVG = 0)

When ADCFG = 0 and ADCAVG = 0, reading from the ADDATA register reads the ADC results stored in one of the 25 data buffers. The ADIDX[4:0] bits point to the data buffer to be read. Reading ADDATA register returns the 14-bits (13 bits plus a sign bit) of ADC conversion data from the selected data buffer memory. The ADIDX[4:0] bits are automatically incremented after a read of ADDATA. This allows multiple reads of ADDATA to access consecutive data buffer locations without needing to change the ADIDX[4:0] bits. The data buffers are reset to 0 on all forms of reset and are not writable by the user.

The data that is read from the ADC Buffer may be from either a temperature or voltage conversion. Also, the data may be right or left aligned. Table 7-4 shows the returned bit weighting for each type of conversion.

Table 7-4: Voltage Data (ADC and Sample and Hold) and Temperature Bit Weighting with Alignment Option

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Temperature Right Aligned	S	S	S	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Temperature Left Aligned	S	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	0	0
Voltage Right Aligned	S	S	S	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Voltage Left Aligned	S	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	0	0

The ADC controller produces temperature, sample and hold and ADC data reading in the **2's complement format**.

7.2.7 – Reference Pin Configuration Register (RPCFG)

See Section 6.2.3 – Reference Pin Configuration Register (RPCFG) for detailed information about RPCFG SFR.

7.2.8 – Temperature Control Register (TEMPCN)

The Temperature Control register TEMPCN configures and enables internal die temperature. The Internal Temperature has a dedicated data buffer at address 22. The DS4830A ADC controller forces current into the internal diode and integrates voltage across diode. After integration the voltage is measured at ADC and the voltage is converted into temperature.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	INT_IEN	-	-	-	-	-	INT_ALIGN	-	-	-	INT_TEMP
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	rw	r	r	r	r	r	rw	r	r	r	rw

BIT	NAME	DESCRIPTION
15:11	-	**Reserved**. The user should not write to these bits.
10	INT_IEN	**Internal Temperature Interrupt Enable:** Setting this bit to '1' enables an interrupt generation on completion of an internal temperature conversion.
9:5	-	**Reserved**. The user should not write to these bits.
4	INT_ALIGN	**Internal Temperature Data Align.** Setting this bit to '1' configures internal temperature conversion data in left aligned mode. Setting this bit to '0' configures internal temperature conversion data in right aligned mode.
3:1	-	**Reserved**. The user should not write to these bits.
0	INT_TEMP	**Internal Temperature Enable.** Setting this bit to '1' initiates internal temperature conversion. The internal temperature typical conversion time is 42µs for default ADC clock. After internal temperature conversion, result is available in data buffer 22.

7.2.9 – Average and Reference Control Register (REFAVG)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	REFOUT	INTAVG	-	-	INTAVG[1:0]		SH1AVG[1:0]		SH0AVG[1:0]	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	rw	rw	r	r	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:10	-	**Reserved**. The user should not write to these bits.
9	REFOUT	**Internal Reference Control:** Setting this bit to '1' outputs the ADC internal reference at GP1 (Pin no 6, Port2.1).
8	INTREF	**Internal Reference Control:** Setting this bit to '1' enables the ADC internal reference and setting this bit to '0' disables the ADC internal reference.
7:6	-	**Reserved**. The user should not write to these bits.
5:4	INTAVG	**Internal Die Temperature Sample Average Control Register:** These bits configure the number of Internal Die Temperature samples to be averaged.

Internal Die Temperature	Number of Samples for Averaging
00b	1
01b	8
10b	16
11b	32

3:2	SH1AVG[1:0]	**SH1 Sample Average Control Register:** These bits configure the number of SH1 samples to be averaged.

SH1AVG	Number of Samples for Averaging
00b	1
01b	2
10b	4
11b	8

1:0	SH0AVG[1:0]	**SH0 Sample Average Control Register:** These bits configure the number of SH0 samples to be averaged.

SH0AVG	Number of Samples for Averaging
00b	1
01b	2
10b	4
11b	8

7.2.10 – ADC Voltage Offset Register (ADVOFF)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	S	S	S	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Reset	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

s = special, initial value is dependent on trim settings

This register contains the ADC voltage offset for the voltage mode. This is calibrated for ADCG1 at the factory to cancel out any offset that may be present in the ADC. The user can add or subtract any offset that they desire by altering this register. This offset is applied to the raw data from the ADC prior to the value being stored into the data buffer. The value stored in the data buffer will be raw_adc + ADVOFF, where raw_adc is the converted voltage without any offset compensation.

7.2.11 – ADC Voltage Scale Trim Registers (ADCG1, ADCG2, ADCG3 and ADCG4)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	ADCG[15:0]															
Reset	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

s = special, initial value is dependent on trim settings

These registers are used to adjust the ADC full scale by changing the gain applied to the ADC reference (internal). These registers are set at the factory to work with the internal reference. The internal reference voltage is set to 1.2V and cannot be changed by the user.

These gain registers are provided so the ADC full scale can be adjusted to meet the needs of the targeted application. Only bits ADCG[15:2] are used to adjust the full scale level. Some approximate settings are:

- ADCGx = 32A8h: The full scale is ~1X the reference level
- ADCGx = 1960h: The full scale is ~2X the reference level
- ADCGx = 0B90h: The full scale is ~4X the reference level
- ADCGx = 0328h: The full scale is ~6X the reference level

It is not recommended that a gain other than 1X, 2X, 4X or 6X be used. This is because the weightings of the ADCGx [15:0] bits are non-linear. An application specific program needs to be developed that tests the ADC full scale for each possible code setting until the proper full scale is achieved. It is recommended that the user should not change ADCG1. The ADC controller uses ADCG1 (not user selectable) for Sample and Hold.

7.3 – ADC Code Examples

7.3.1 – One Sequence of 4 Voltage Conversions for Ch0 (Diff), Ch1 (Diff), Ch14 (Single), and Ch15 (Single)

```
PINSEL = 0xC00F;              //Configure Pin as ADC Ch0 (Diff), Ch1 (Diff), Ch14 (Single) and Ch15(Single)

REFAVG_bit.INTREF = 1;        //Enable ADC internal reference

 for(iCounter = 0; iCounter < 1000; iCounter++); //Wait ~1ms to settle ADC internal reference

ADCN_bit.ADCONT = 0;          //run a single conversion sequence

ADST_bit.ADCFG = 1;           //set ADDATA for configuration (ADCFG)
ADST_bit.ADIDX = 0;           //ADIDX = 0, set to ADCFG [0]

ADDATA = 0x0020;              //ADCFG [0]: Differential voltage, CH0,     1.2V FS, Right Aligned
ADDATA = 0x2021;              //ADCFG [1]: Differential voltage, CH1,     0.6V FS, Right Aligned
ADDATA = 0x400E;              //ADCFG [2]: Single voltage,       CH14,    2.4V FS, Right Aligned
ADDATA = 0x600F;              //ADCFG [3]: Single voltage,       CH15,    6.55V FS, Right Aligned

ADST_bit.ADCFG = 0;           //set ADDATA to data buffer

ADADDR_bit.ADSTART = 0;       //start sequence with ADCFG [0]
ADADDR_bit.ADEND = 3;         //end sequence with ADCFG [3]

 ADST_bit.ADCONV = 1;         //start the conversions

while (!ADST_bit.ADDAI);      //wait for conversions to complete

ADST_bit.ADDAI = 0;           //Clear ADDAI flag

ADST_bit.ADIDX = 0;           //set ADDATA to data buffer [0]

ch0_volt = ADDATA;            //read and store ch0 voltage to variable
ch1_volt = ADDATA;            //read and store ch1 voltage to variable

ADST_bit.ADIDX = 14;          //set ADDATA to data buffer [14] (according to channel number)

ch14_volt = ADDATA;           //read and store ch14 voltage to variable
ch15_volt = ADDATA;           //read and store ch15 voltage to variable
```

7.3.2 – Continuous Conversion of 16 Samples of Ch0 with Location Override

```
PINSEL = 0x0003;              //Configure Pins as ADC Ch0 (Diff)

REFAVG_bit.INTREF = 1;        //Enable ADC internal reference

 for(iCounter = 0; iCounter < 1000; iCounter++); //Wait ~1ms to settle ADC internal reference

ADCN_bit.ADCONT = 1;          //run a continuous conversion sequence
ADCN_bit.LOC_OVR = 1;         //location override enable

ADST_bit.ADCFG = 1;           //set ADDATA as configuration (ADCFG)
ADST_bit.ADIDX = 0;           //ADIDX = 0, set to ADCFG [0]

ADDATA = 0x0020;              //ADCFG [0]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 0
ADDATA = 0x0120;              //ADCFG [1]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 1
ADDATA = 0x0220;              //ADCFG [2]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 2
ADDATA = 0x0320;              //ADCFG [3]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 3

ADDATA = 0x0420;              //ADCFG [4]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 4
ADDATA = 0x0520;              //ADCFG [5]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 5
ADDATA = 0x0620;              //ADCFG [6]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 6
ADDATA = 0x0720;              //ADCFG [7]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 7

ADDATA = 0x0820;              //ADCFG [8]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 8
ADDATA = 0x0920;              //ADCFG [9]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 9
ADDATA = 0x0A20;              //ADCFG [10]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 10
ADDATA = 0x0B20;              //ADCFG [11]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 11

ADDATA = 0x0C20;              //ADCFG [12]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 12
ADDATA = 0x0D20;              //ADCFG [13]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 13
ADDATA = 0x0E20;              //ADCFG [14]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 14
ADDATA = 0x0F20;              //ADCFG [15]: Differential voltage, CH0, 1.2 V FS, Right Aligned and Location override 15
```

```
ADST_bit.ADCFG = 0;          //set ADDATA to data buffer

ADADDR_bit.ADSTART = 0;  //start sequence with ADCFG [0]
ADADDR_bit.ADEND = 15;    //end sequence with ADCFG [15]

ADST_bit.ADCONV = 1;         //start the conversions

while (1)
{
   while (!ADST_bit.ADDAI);   //wait for conversions to complete

   ADST_bit.ADDAI = 0;

   ADST_bit.ADIDX = 0;         //set ADDATA to data buffer [0]

   for (iCount = 0; iCount < 16; iCount++)
       ch0 [iCount]= ADDATA;  //read and store ch0 voltage to variable
 }
```

7.3.3 – Continuous Conversion of 16 Samples of Ch0 Using ADC Averaging

```
PINSEL = 0x0003;                              //Configure Pins as ADC Ch0 (Diff)

REFAVG_bit.INTREF = 1;                                  //Enable ADC internal reference

for(iCounter = 0; iCounter < 1000; iCounter++); //Wait ~1ms to settle ADC internal reference

ADCN_bit.ADCONT = 1;                          //run a continuous conversion sequence

ADST_bit.ADCFG = 1;                           //set ADDATA as configuration (ADCFG)
ADST_bit.ADIDX = 0;                           //ADIDX = 0, set to ADCFG [0]

ADDATA = 0x0020;                              //ADCFG [0]: Differential voltage, CH0, 1.2 V FS, Right Aligned

ADST_bit.ADCFG = 0;                           //set ADDATA to data buffer

ADST_bit.ADCAVG = 1;                          //set ADDATA to data buffer
ADDATA = 0x0003;                              // Average of 16 samples of Ch0
ADST_bit.ADCAVG  = 0;

ADADDR_bit.ADSTART = 0;                        //start sequence with ADCFG[0]
ADADDR_bit.ADEND = 0;                         //end sequence with ADCFG[0]

ADST_bit.ADCONV = 1;                          //start the conversions

while (1)
{
      while (!ADST1_bit.ADDAI);               //wait for conversions to complete

      ADST_bit.ADIDX = 0;                     //set ADDATA to data buffer [0]

      ch0 = ADDATA;                           //read and store ch0 voltage to variable

      ADST1_bit.ADDAI = 0;                    //clear ADDAI flag
}
```

SECTION 8 – SAMPLE AND HOLD

The DS4830A has two independent, but identical, Sample and Hold differential channels. Sample and Hold 0 (S/H0) is on GP2-GP3 and Sample and Hold 1 (S/H1) is on GP12-GP13. The sample and hold function can be configured for internal or external triggering. Each sample and hold has a dedicated pin for external trigger.

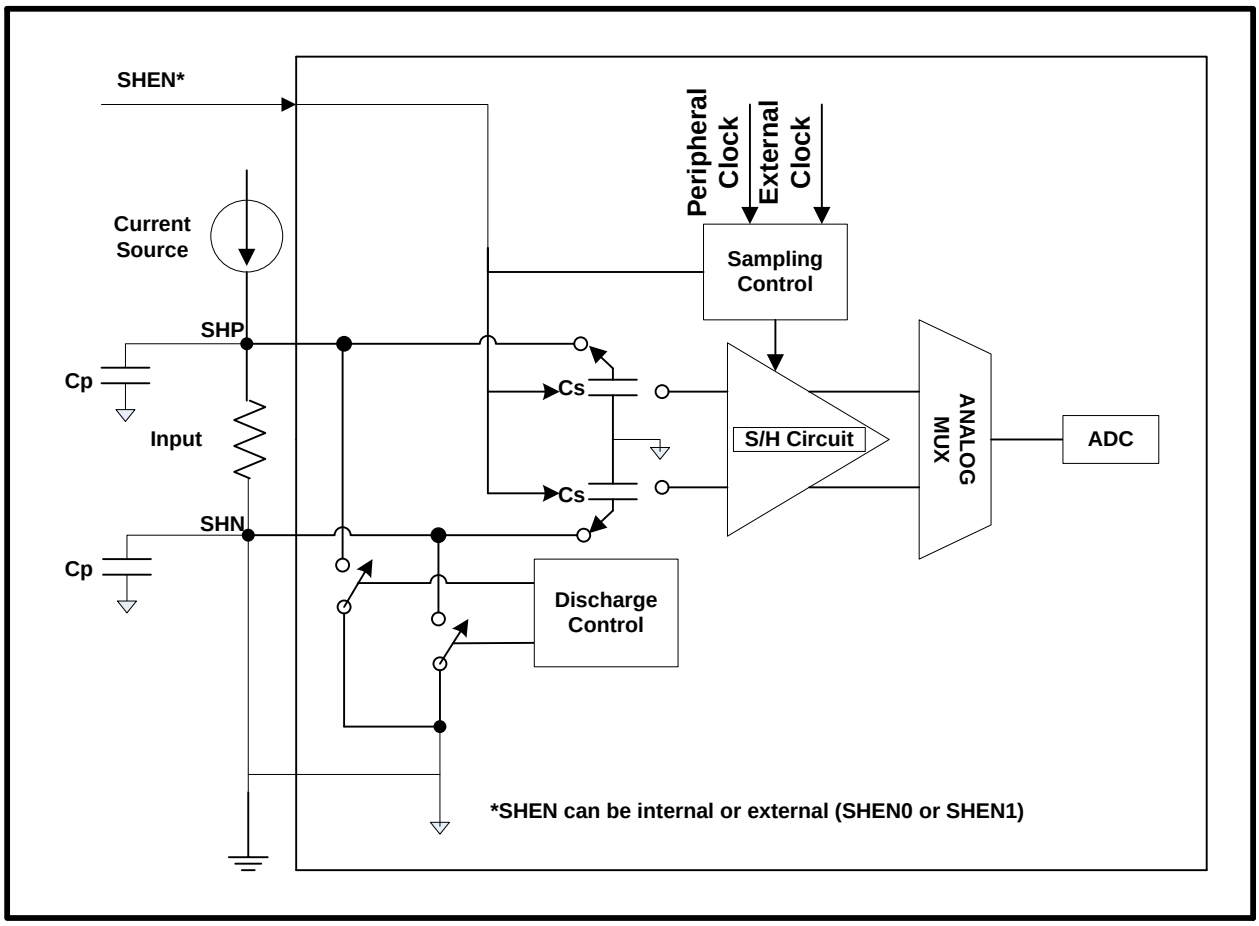


Figure 8-1: Sample and Hold Functional Block Diagram

8.1 – Detailed Description

As shown in Figure 8-1, each Sample and Hold consists of fully differential sampling capacitors (C_s), control logic and a differential output buffer. The sample and hold also contains a charge injection nulling circuit. Additionally, it has a discharge circuit to discharge parasitic capacitance on the input node and the sample capacitor before it starts sampling. The input voltage is sampled using 5pF capacitor on the positive input and another 5pF capacitor on the negative input. The negative input pin is used to reduce ground offset and noise. The capacitors are connected to the input pins when sample trigger signal SHEN (either internal or external) is high. During high period of sample pulse, the sample and hold performs sampling which ends at negative edge of the sample pulse SHEN. In addition to the sampling capacitors, the input pins also have parasitic capacitance. When the sample and hold is configured for internal triggering, the sample pulse is internally generated by the sample and hold hardware.

8.1.1 – Operation

When the SHEN signal goes high, the sample-and-hold capacitors are connected to the sample-and-hold input pins (GPx) for sampling of the input signal. The minimum sample time should be 300ns for proper sampling. When the SHEN signal goes low, the sampling is stopped and voltage stored at sampling capacitors are converted by the ADC controller. See Figure 8-2 for Sample and Hold Timings. Each Sample and Hold can be independently enabled by setting their respective enable bit in the Sample and Hold Control Register (SHCN). The sample and hold has two modes of operation "Single Mode" and "Dual Mode".

For proper first sample capturing on power up, the sample and hold should be initialized as explained below.

1. Enable sample and hold for internal sample
2. Apply internal pulse for few µs
3. Wait for conversion to complete, clear the flags and discard the result.
4. Configure S/H according to application requirement without disabling the S/H.

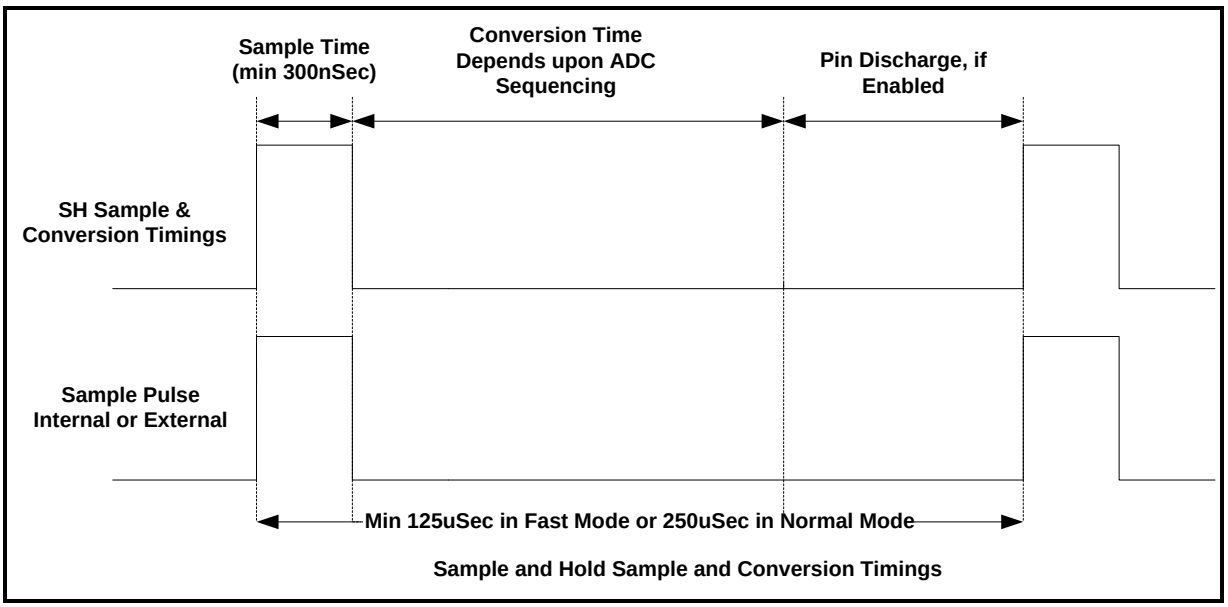


Figure 8-2: Sample and Hold Conversion Timings without Averaging

8.1.1.1 – Single Mode Operation

During the single mode operation, the SHEN signal (either internal trigger or external trigger at SHEN0) acts as a sample pulse for both sample and hold 0 and 1. The SH0DAI bit in the ADST register is set to '1', after conversion of both sample and holds by the ADC and an interrupt is generated if enabled. The results are available at data buffer locations 23 and 24 respectively for both sample and holds after the ADC conversion is complete.

In the single mode operation the SH0DAI bit is set to '1'

a. At the completion of both sample and hold channels ADC conversion, if both sample and holds are enabled.
b. At the completion of only enabled sample and hold channel if any one sample and hold enabled.

The sample and hold interrupt for both sample and hold circuits can be enabled by the setting the SHDAI0_EN bit in the SHCN register. In single mode operation, the SENR[1:0] register bits control the SHEN source for both of the sample and holds.

8.1.1.2 – Dual Mode Operation

Dual mode operation is selected when SH_DUAL bit in the SHCN register is set to '1'. In this mode of operation, both the sample and hold circuits work independently. Each sample and hold can have separate internal or external triggers. The SHEN0 and SHEN1 provide sample pulses to Sample and Hold 0 and Sample and Hold 1 respectively for external trigger. The Sample and Hold Internal Trigger Enable Register (SENR) has bits to enable the internal trigger for both sample and hold circuits individually. In the dual mode operation each sample and hold generates its own Sample and Hold Data Available Interrupt Flag (SH0DAI and SH1DAI) in the ADST register. Each of these flags can generate an interrupt if enabled. The results are available in ADC data buffer (ADDATA, see the ADC SFR description for detail) 23 and 24, respectively.

8.1.2 – Fast Mode Operation

The DS4830A Sample and Hold provides a special "Fast Mode" feature which gives priority to a sample and hold conversion over an ADC voltage conversion. The "Fast Mode" is enabled by setting the FAST_MODE bit to '1' in the SHCN register. This mode is useful when only Sample and Hold 0 is used. In fast mode operation the Sample and Hold 0 is guaranteed to get a conversion slot in the ADC conversion sequence every 125µs (If averaging is not enabled). In this mode, the user is allowed to issue SHEN pulses (either internal of external pulse) at every 125µs interval. This bit should be used with care, as it creates priority for the Sample and Hold0 over other sequence mode

channels and hence their ADC conversion will be delayed. When the FAST_MODE bit is set to '0', the user can issue SHEN pulse every 250μs time interval.

Note: When averaging is used for ADC channels or S/H's, the S/H conversion time slot changes as shown in Figure 7-5 and cannot be guaranteed to get conversion slot in 125μs or 250μs. The S/H conversion time depends upon number of ADC samples to be averaged.

8.1.3 – Sampling Control

The sample and hold circuitry provides the option to select the internal peripheral clock or the external clock. When the clock select bit CLK_SEL (located in the SHCN register) is set to '0', the peripheral clock is used for the sample and hold circuit. When the clock select bit CLK_SEL is set to '1', the external clock (CLKIN on the DACPW2 pin) is used for the sample and hold circuit.

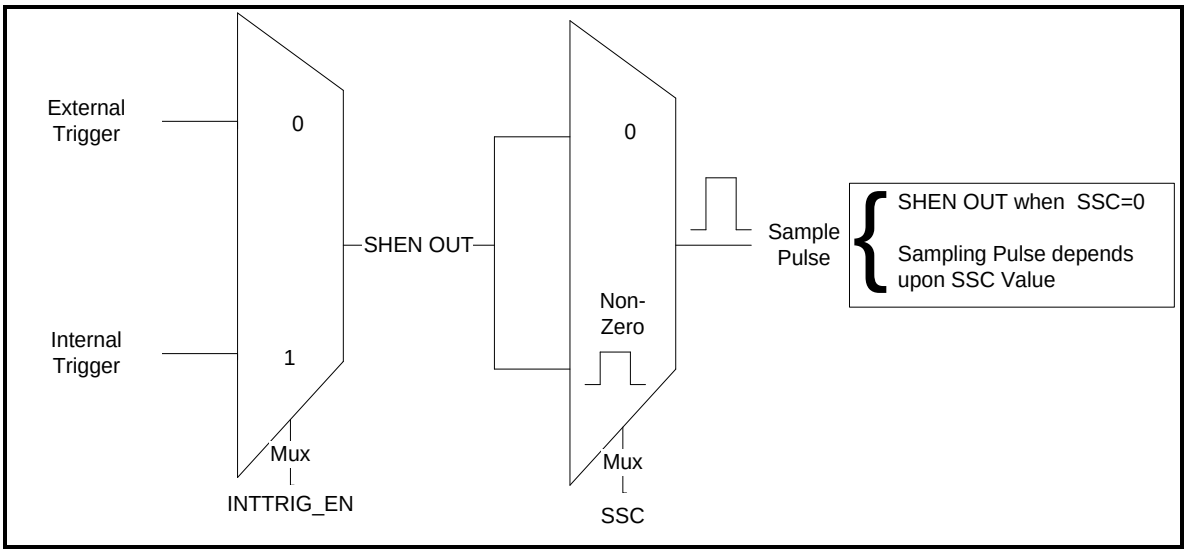


Figure 8-3: Sample Pulse

The end of the sample and hold sample time is controlled by the Sampling Stop Control bits SSC[3:0] in the SHCN register. These bits are used along with the CLK_SEL bit to determine the length of the sample pulse. When the SSC[3:0] bits have non-zero values and the CLK_SEL bit is set to '1', the stop sampling depends upon the number of external clock cycles. When the SSC[3:0] bits have non-zero values and the CLK_SEL bit is '0', the stop sampling depends upon the time from the rising edge of SHEN0/1 (See Figure 8-3 for Sample Pulse). See SSC[3:0] bit description for stop sampling timings.

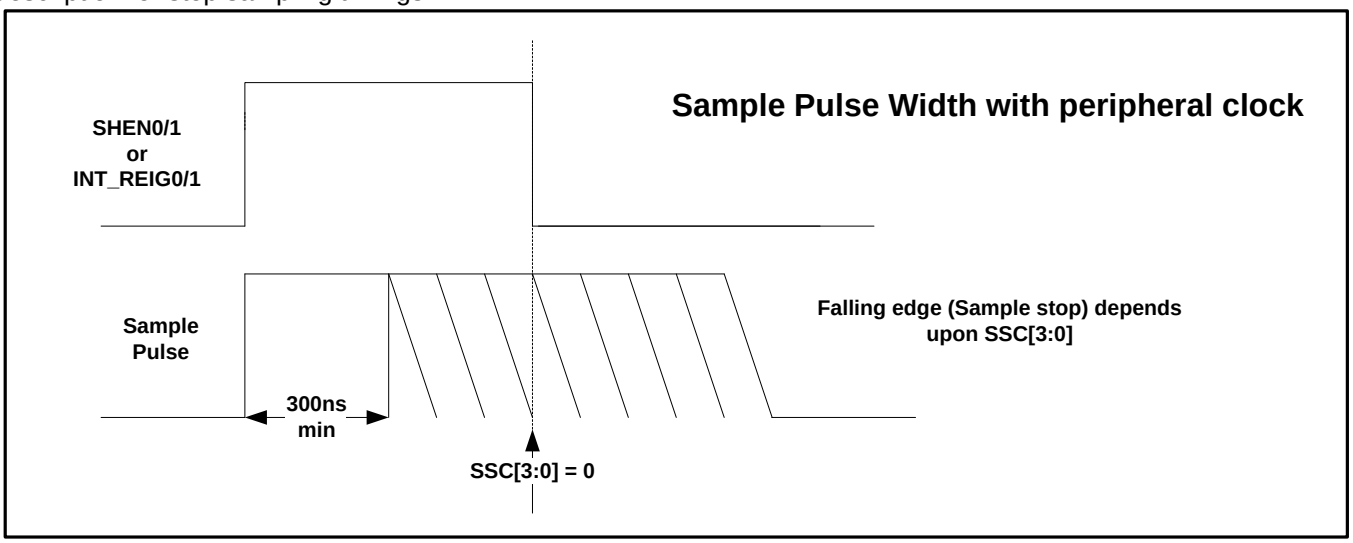


Figure 8-4: Sample Pulse Width with the Peripheral Clock

As shown in Figure 8-4, the sample pulse width time depends upon the SSC bits value when the peripheral clock is selected (CLK_SEL = 0).

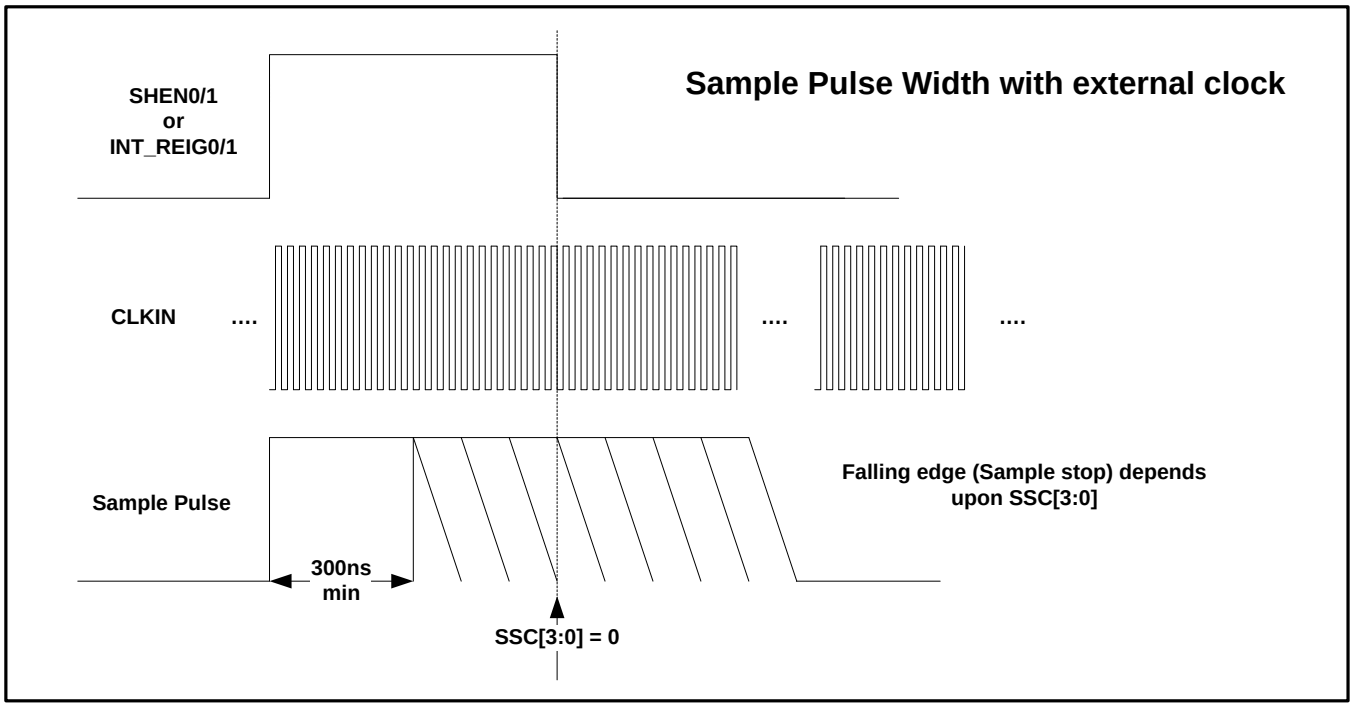


Figure 8-5: Sample Pulse Width with the External Clock

As shown in Figure 8-5, the sample pulse width time depends upon the SSC bits value when the external clock is selected (CLK_SEL = 1).

8.1.4 – Pin Capacitance Discharge

Before the sample and hold circuitry start sampling, the DS4830A has an option to discharge pin capacitance. The SHCN register has PIN_DIS0 and PIN_DIS1 bits to enable the pin discharge function before sampling begins. This is an optional feature, which generates a discharge pulse that discharges the pin or PCB capacitance for the sample and hold channels. The discharge pulse is active after the corresponding sample and hold channel's conversion is complete and goes inactive on the rising edge of SHEN0 or SHEN1 pulse. See pin discharge timing is shown in Figure 8-6.

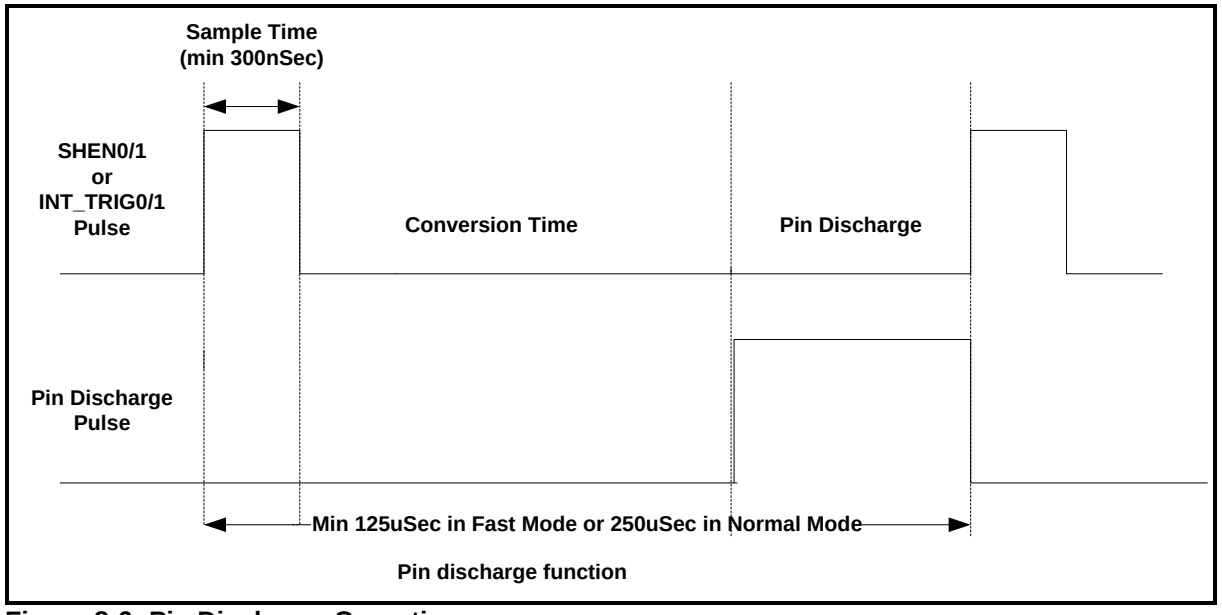


Figure 8-6: Pin Discharge Operation

8.1.5 – Sample and Hold Data Reading

Each sample and hold has defined data buffer locations where the ADC controller writes sample and hold results after the ADC conversion. The data buffer location 23 and 24 are reserved for Sample and Hold 0 and 1 respectively. The ADC controller uses ADCG1 (1.2V full scale) for ADC conversion of the sampled signal of both sample and holds.

8.1.6 – Sample and Hold Interrupts

The DS4830A sample and hold has two interrupt flags SH0DAI and SH1DAI in the ADST register. The SH1DAI bit is used only when both Sample and Hold are enabled in the dual mode operation. In single mode operation, SH0ADI is set only when:

1. Both sample and holds are enabled, then after the ADC conversion of both samples.
2. If only one sample and hold is enabled, then after the ADC conversion of the enabled sample and hold.

8.2 – Sample and Hold Register Descriptions

The sample and hold has two SFRs. These are Sample and Hold Control Register (SHCN) and Sample and Hold Internal Trigger Enable register (SENR). The SHCN register controls both sample and holds. The SENR controls the internal sample pulse for both sample and holds. The sample and hold SFRs are located in module 4.

8.2.1 – Sample and Hold Control Register (SHCN)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	SSC[3:0]				FAST_MODE	PIN_DIS1	PIN_DIS0	SH_DUAL	-	SH1_ALIGN	SHDAI1_EN	SMP_HLD1	CLK_SEL	SH0_ALIGN	SHDAI0_EN	SMP_HLD0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw

BIT | **NAME** | **DESCRIPTION**

15:12 | SSC[3:0] | **STOP Sample Control.** These bits control the end of the sample and hold sampling relative to the SHEN0 and SHEN1 pulse.

CLK_SEL = 0	
SSC[3:0]	**STOP Sampling**
0000	Falling Edge of SHEN0/SHEN1
0001	Reserved
0010	Reserved
0011	Reserved
0100	300ns after rising edge of SHEN0/SHEN1
0101	350ns after rising edge of SHEN0/SHEN1
0110	450ns after rising edge of SHEN0/SHEN1
0111	550ns after rising edge of SHEN0/SHEN1
1000	750ns after rising edge of SHEN0/SHEN1
1001	1us after rising edge of SHEN0/SHEN1
1010	1.5us after rising edge of SHEN0/SHEN1
1011	1.75us after rising edge of SHEN0/SHEN1
1100	2us after rising edge of SHEN0/SHEN1
1101	2.5us after rising edge of SHEN0/SHEN1
1110	4us after rising edge of SHEN0/SHEN1
1111	5us after rising edge of SHEN0/SHEN1

CLK_SEL = 1	
SSC[3:0]	**STOP Sampling**
0000	Falling Edge of SHEN0/SHEN1
0001	21 ext-clock after rising edge of SHEN0/SHEN1
0010	22 ext-clock after rising edge of SHEN0/SHEN1
0011	23 ext-clock after rising edge of SHEN0/SHEN1
0100	24 ext-clock after rising edge of SHEN0/SHEN1
0101	25 ext-clock after rising edge of SHEN0/SHEN1
0110	26 ext-clock after rising edge of SHEN0/SHEN1
0111	27 ext-clock after rising edge of SHEN0/SHEN1
1000	28 ext-clock after rising edge of SHEN0/SHEN1
1001	29 ext-clock after rising edge of SHEN0/SHEN1
1010	30 ext-clock after rising edge of SHEN0/SHEN1
1011	31 ext-clock after rising edge of SHEN0/SHEN1
1100	32 ext-clock after rising edge of SHEN0/SHEN1
1101	33 ext-clock after rising edge of SHEN0/SHEN1
1110	34 ext-clock after rising edge of SHEN0/SHEN1
1111	35 ext-clock after rising edge of SHEN0/SHEN1

Note: A minimum sample time of 300nSec must be used when external clock is used to guarantee accurate results.

11 | FAST_MODE | **Fast Mode Enable.** Setting this bit to '1' enables the fast operation for Sample and Hold 0. In this mode, Sample and Hold 0 is guaranteed to get a conversion slot in the ADC conversion sequence every 125μs and the user can issue sample pulses at an interval of 125μs. During fast mode, the sample and hold conversion priority is increased over voltage channels in the sequence and the voltage conversions will be delayed. When

		this bit is '0', Sample and Hold 0 acts in the normal mode in which Sample and Hold 0 gets a conversion slot in the ADC sequence every 250μs.
10	PIN_DIS1	**Pin Discharge Enable 1.** Setting this bit to '1' enables the pin discharge function for Sample and Hold 1. The discharge function discharges pin capacitances (GP12-GP13) after the Sample and Hold 1 ADC conversion.
9	PIN_DIS0	**Pin Discharge Enable 0.** Setting this bit to '1' enables pin discharge function at Sample and Hold 0. The discharge function discharges pin capacitances (GP2-GP3) after the Sample and Hold 0 ADC conversion.
8	SH_DUAL	**Sample and Hold Dual Mode.** Setting this bit to '1' configures in "Dual Mode" Sample and Hold operation. In dual mode, both sample and holds act independently and use different sample trigger input signals. SHEN0 (pin 23) acts as the sample trigger input signal for Sample and Hold 0. SHEN1 (pin 21) acts as the sample trigger input signal for Sample and Hold 1. In single mode operation both sample and hold circuits are triggered by the SHEN0 signal.
7	-	**Reserved**. The user should write 0 to this bit.
6	SH1_ALGN	**Sample and Hold 1 Data Alignment Select.** This bit selects the Sample and Hold 1 data alignment mode. Setting this bit to '1' returns data left aligned in ADDATA[15:2] with ADDATA[1:0] zero padded. Clearing this bit to '0' returns data in right aligned format in ADDATA[13:0] with ADDATA[15:14] sign-extended by ADDATA[13].
5	SHDAI1_EN	**Sample and Hold 1 Interrupt Enable.** Setting this bit to '1' enables interrupt generation on the completion of Sample and Hold 1 ADC conversion in the dual mode.
4	SMP_HLD1	**Sample and Hold 1 Enable.** Setting this bit to '1' enables Sample and Hold 1 operation on GP12-GP13 input pins. The conversion results are available in ADC data buffer location 24.
3	CLK_SEL	**Clock Select for Sample and Holds Trigger delayed rising edge control**. This bit selects the clock used to stop sampling when operating in SSC mode. During this mode SSC[3:0] bits controls the delay from the start to stop of sampling.. When this bit is set to '0', the peripheral clock is used for generating the SHEN pulse. When this bit is set to '1', the External Clock (CLKIN pin) is used for generating the SHEN pulse. See the SSC[3:0] bit description to see the effect of CLK_SEL on the SHEN0/SHEN1 pulse generation.
2	SH0_ALGN	**Sample and Hold 0 Data Alignment Select.** This bit selects the Sample and Hold 0 data alignment mode. Setting this bit to '1' returns data left aligned in ADDATA[15:2] with ADDATA[1:0] zero padded. Clearing this bit to '0' returns data in right aligned format in ADDATA[13:0] with ADDATA[15:14] sign-extended by ADDATA[13].
1	SHDAI0_EN	**Sample and Hold 0 Interrupt Enable.** Setting this bit to '1' enables interrupt generation on the completion of Sample and Hold 0 ADC conversion when operating in dual mode operation. In the single mode operation, this bit is set at the completion of both sample and hold conversions.
0	SMP_HLD0	**Sample and Hold 0 Enable.** Setting this bit to '1' enables Sample and Hold 0 operation on GP2-GP3 input pins. The conversion results are available in ADC data buffer location 23.

8.2.2 – Sample and Hold Internal Trigger Enable Register (SENR)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name											INT_TRIG_EN1	INT_TRIG1	-	-	INT_TRIG_EN0	INT_TRIG0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	rw	rw	r	r	rw	rw

BIT	NAME	DESCRIPTION
15:6	-	**Reserved**. The user should write 0 to these bits.
5	INT_TRIG_EN1	**Sample and Hold 1 Internal Trigger Enable**. Setting this bit to '1' enables internal trigger mode for Sample and Hold 1. When this bit is set to '1', writing a '1' to INT_TRIG1 starts an internal sample pulse for Sample and Hold 1. When this bit is '1', sample pulses on SHEN1 are igonred. Setting this bit to '0' configures Sample and Hold 1 for external sample pulse. This bit is used in the dual mode operation only.
4	INT_TRIG1	**Sample and Hold 1 Internal Trigger.** This bit is used when INT_TRIG_EN1 is set to '1'. Setting this bit to '1' starts internal sample pulse for Sample and Hold 1. The sample pulse will end when this bit is set back to 0 if SSC[3:0] = 0, or after the time defined by SSC[3:0] if these bits are not equal to 0. This bit is used in the dual mode operation only.
3:2	-	**Reserved**. The user should write 0 to these bits.
1	INT_TRIG_EN0	**Sample and Hold Internal Trigger Enable**. Setting this bit to '1' enables internal trigger mode for Sample and Hold 0. When this bit is set to '1', writing a '1' to INT_TRIG0 starts an internal sample pulse for Sample and Hold 0. When this bit is '1', sample pulses on SHEN0 are igonred. Setting this bit to '0' configures Sample and Hold 0 for external sample pulse. In the single mode operation, this bit is used for both sample and holds.
0	INT_TRIG0	**Sample and Hold0 Internal Trigger.** This bit is used when the INT_TRIG_EN0 is set to '1'. Setting this bit to '1' starts internal sample pulse for Sample and Hold 0. The sample pulse will end when this bit is set back to 0 if SSC[3:0] = 0, or after the time defined by SSC[3:0] if these bits are not equal to 0.In the single mode operation, this bit is used for both sample and holds.

8.2.3 – Sample and Hold Interrupt flag

See ADST1 description for Sample and Hold interrupts flags SH0DAI and SH1DAI descriptions.

8.2.4 – Sample and Hold Averaging

See REFAVG description in ADC section for sample and hold averaging options.

SECTION 9 – QUICK TRIP (FAST COMPARATOR)

The DS4830A has 10-bit quick trips with a 16-input analog MUX (Figure 9-1). The MUX selects the quick trip analog input from 16 external channels. The quick trip external channels can be configured to operate as eight fully differential inputs or sixteen single-ended inputs. The quick trip monitors all configured quick trip channels in a round robin sequence.

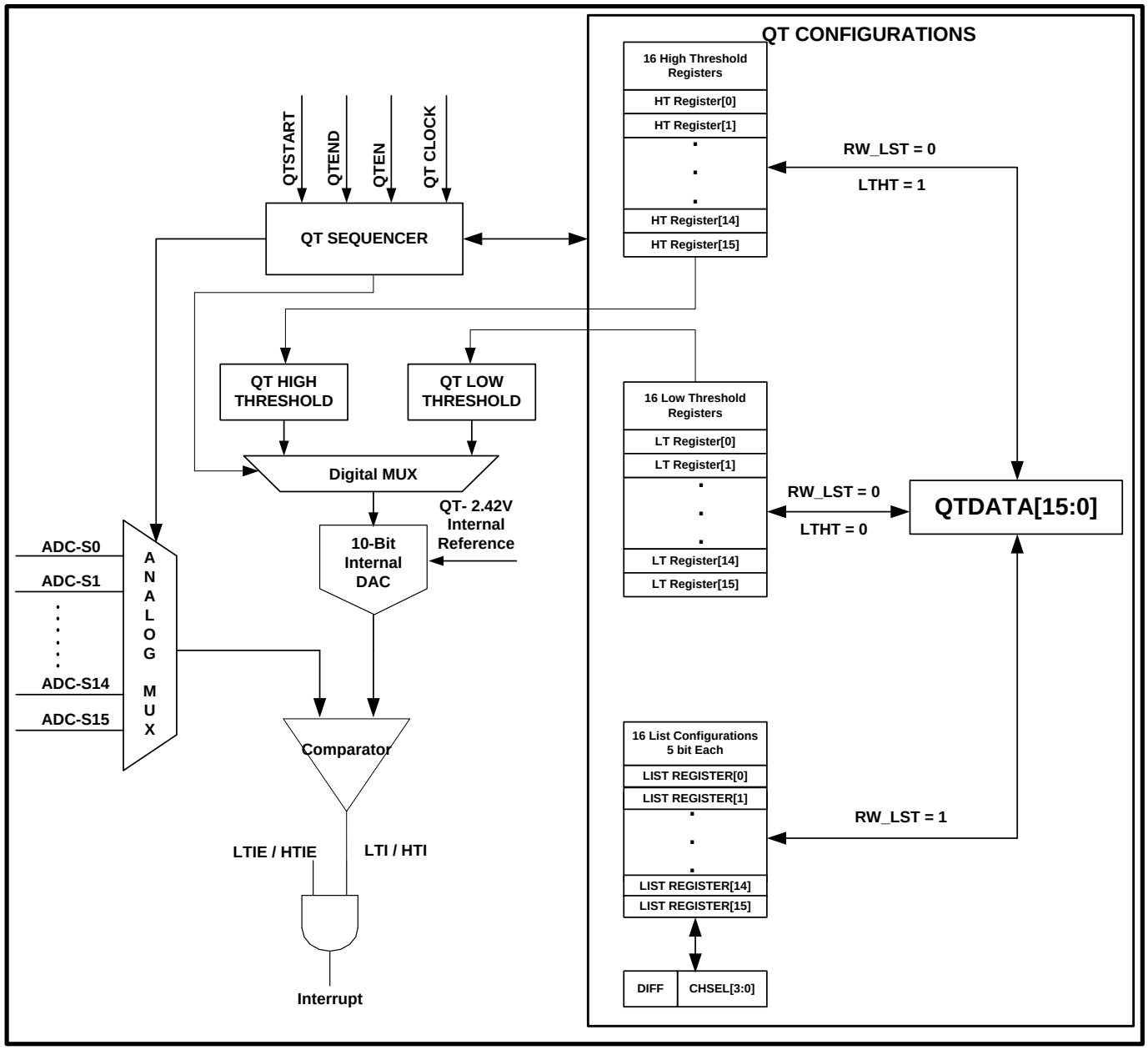


Figure 9-1: Quick Trip Functional Diagram

9.1 – Detailed Description

As shown in Figure 9-1, the DS4830A Quick Trip (QT) controller has a 16-input analog MUX and Quick Trip Sequencer. The QT sequencer creates a list of configurations and sets user defined low and high threshold for external channels. The quick trip controller has 16 low trip threshold and 16 high trip threshold internal registers.

The Quick Trip Control Register (QTCN) has two bits RW_LST and LTHT which are used to configure thresholds and list creation. The QTIDX[3:0] bits (located in the QTCN register) together with LTHT and RW_LST bits select the source or destination address for the QTDATA register access. Figure 9-1 illustrates the threshold configuration and list creation.

By default, the external channels GP0-15 are general-purpose input. The DS4830A has the Pin Select Register (PINSEL). The PINSEL register is used to configure the external channels as an analog pin for ADC or/and Quick Trip use. Each bit location in this register corresponds to the ADC/Quick Trip input pin.

Table 9-1: Low and High Thresholds Configuration and List Creation

RW_LST	LTHT	QTIDX	REGISTER SELECTED
0	0	N(0 to 15)	Low threshold configuration for the channel defined in list N
0	1	N(0 to 15)	High threshold configuration for the channel defined in list N
1	X	N(0 to 15)	N^{th} register of list configuration

Thresholds Configuration
Each configuration has two threshold registers to configure low and high threshold. Each threshold register is addressed by the QTIDX[3:0] bits. These bits are auto incremented on any read or write operation to the QTDATA register. The low trip thresholds are configured by writing to the QTDATA register when the RW_LST and LTHT bits are set to '0'. The high trip thresholds are configured by writing to the QTDATA register when the RW_LST bit is set to '0' and LTHT bit is set to '1'.

List Creation
As shown in Figure 9-1, the quick trip controller has 16 list registers. These are configured by writing to the QTDATA register when the RW_LST bit is set to '1'. The list address is addressed by the QTIDX[3:0] bits. Each list register uses only lower 5 bits. The first 4 lower bits CHSEL [3:0] specifies the quick trip input channel. The DIFF bit selects between single-ended mode (when DIFF bit is set to '0') and differential mode (when DIFF bit is set to '1') quick trip comparison. The start and stop addresses of the list are provided by the Quick Trip List Register (QTLST). Any channel can be used multiple times at any location in the list.

See Section 9.2 - Quick Trip Register Descriptions for details.

As shown in Figure 9-1, the quick trip sequencer selects a channel from 16 external channels. The quick trip controller has an internal 10-bit DAC which generates voltage for low and high threshold comparisons with the external channel input. The quick trip is also called a "Fast Comparator" as it compares the input with threshold using the fast comparator. The conversion time is 1.6µs for each threshold; so each channel's thresholds are compared in 3.2µs (1.6µs for low trip threshold + 1.6µs for high trip threshold).

9.1.1 – Quick Trip List Sequencing
The DS4830A quick trip controller performs the user defined sequence of up to 16 single-ended or 8 differential external channels conversions.

A sequence is setup in the QTLST register by defining the starting conversion configuration address (QTSTART) and an ending conversion configuration address (QTEND). The configuration start address designates the configuration register to be used for the first conversion in a sequence. The configuration end address designates the configuration register used for the last conversion in a sequence. A single channel conversion can be viewed as a special case for sequence conversion, where the starting and ending configuration address is the same. The configuration registers can be viewed as a circular register array where QTSTART does not have to be less than QTEND. For example, if QTSTART = 1 and QTEND = 5, then the sequence of conversions would be configurations 1, 2, 3, 4, 5. If QTSTART = 5 and QTEND = 1, then the sequence of conversions would be configurations 5, 6, 7 . . . 15, 0, 1.

9.1.2 – Operation
The quick trip is enabled by setting the Quick Trip Enable (QTEN) bit to '1' in the QTCN register. The Quick Trip Controller takes ~120 core clocks to wake up after enable and then starts scanning through the list of channels specified in the channel list register QTLST continuously in the round robin sequence. The quick trip sequence reads the list, selects the input channel and reads the low trip threshold and performs 10-bit comparison, then reads the high trip threshold and again performs 10-bit comparison. The quick trip has separate interrupt flag registers for the low and high trip threshold. The low trip interrupt flag is set when the input voltage is less than the configured low threshold. Similarly, the high trip interrupt flag is set when the input voltage is greater than the configured high threshold. The interrupt can be generated if enabled.

The channel list can be filled up using the QTDATA register by setting the RW_LST bit to '1' in the QTCN register. For example to scan channels S5, S6 and S14-15 having configurations for channels 5 & 6 in the single-ended mode, channel 7 (S14-S15) in the differential mode and channel 6 again (any channel can be configured multiple

times in the QT list). The quick trip list can be filled sequentially with data 05h (channel 5 + single-ended), 06h (channel 6 + single-ended), 17h (channel 7 + differential mode) and 06h (channel 6 + single-ended). See Table 9-2 for the quick trip list configurations.

To scan these list registers shown in Table 9-2, the QTSTART bits are set to 0 (0000b) and the QTSTOP bits are set to 3 (0011b). Each channel is compared twice (see Figure 9-2). First the low trip threshold (LT) is compared and then the high trip threshold (HT). The sequence of comparisions is shown is Figure 9-2.

Table 9-2: Quick Trip List Configuration

QT LIST NUMBER	QTDATA	DESCRIPTION	LIST REGISTERS USED FOR COMPARISON
0	05h	Channel 5 (S5) in single-ended mode	LT0 and HT0
1	06h	Channel 6 (S6) in single-ended mode	LT1 and HT1
2	17h	Channel 7(S14-S15) in differential mode	LT2 and HT2
3	06h	Channel 6 (S6) in single-ended mode	LT3 and HT3

Quick Trip Start address (QTSTART) = 0
Quick Trip Stop address (QTEND) = 3

Channel 5		Channel 6		Channel 7		Channel 6*		.....	Channel 5		Channel 6		Channel 7		Channel 6*		.....
LT	HT	LT	HT	LT	HT	LT	HT		LT	HT	LT	HT	LT	HT	LT	HT	

*** Note: Channels can be defined multiple times in the list.**

Figure 9-2: Quick Trip Operation

9.1.3 – Setting Quick Trip Thresholds

The quick trip threshold can be calculated by using the following formula.

$$\text{Threshold} = \frac{\text{Threshold Voltage}}{\text{Full Scale Voltage}(2.42\text{V})} * 1024$$

Table 9-3 demonstrates Quick Trip low and high threshold configuration.

Table 9-3: Quick Trip Low Threshold Configuration

QT LIST NUMBER	LOW THRESHOLD VALUE (AS EXAMPLE)	QTDATA
0	0.6V	0x00FE
1	0.8V	0x0153
2	1.0V	0x01A7
3	1.1V	0x01D1

```
QTCN = 0x0000;          //Low Threshold Configuration Register, Index = 0
QTDATA = 0x00FE;        //0.6V Low Threshold Configuration for List0 Configuration
QTDATA = 0x0153;        //0.8V Low Threshold Configuration for List1 Configuration
QTDATA = 0x01A7;        //1.0V Low Threshold Configuration for List2 Configuration
QTDATA = 0x01D1;        //1.1V Low Threshold Configuration for List3 Configuration
```

Table 9-4: Quick Trip High Threshold Configuration

QT LIST NUMBER	HIGH THRESHOLD VALUE (AS EXAMPLE)	QTDATA
0	2.2V	0x03A3
1	2.0V	0x034E
2	1.8V	0x02FA
3	1.6V	0x02A5

```
QTCN = 0x0010;          //High Threshold Configuration Register, Index = 0
QTDATA = 0x03A3;        //2.2V High Threshold Configuration for List0 Configuration
QTDATA = 0x034E;        //2.0V High Threshold Configuration for List1 Configuration
QTDATA = 0x02FA;        //1.8V High Threshold Configuration for List2 Configuration
QTDATA = 0x02A5;        //1.6V High Threshold Configuration for List3 Configuration
```

9.1.4 – Quick Trip Interrupts

The DS4830A quick trip has four interrupt flag registers the Low Trip Interrupt Lower Flag Register (LTIL), High Trip interrupt Lower Flag Register (HTIL), Low Trip Interrupt High Register and High Trip Interrupt High Register. See the register descriptions for the quick trip interrupt operation.

9.2 – Quick Trip Register Descriptions

The quick trip has 7 SFRs. These are the Quick Trip Control Register (QTCN), Quick Trip List Register (QTLST), Quick Trip Data Register (QTDATA), Low Trip Interrupt Lower Flag Register (LTIL), High Trip Interrupt Lower Flag Register (HTIL), Low Trip Interrupt High Register (LTIH) and High Trip Interrupt High Register (HTIH). The QTCN register controls the quick trip operation. The QTLIST register defines the list for the quick trip controller. The QTDATA register is used to read and write list and threshold (high and low threshold) registers. The LTIL and HTIL are interrupt flag registers for high and low threshold. The LTIH and HTIH are the interrupt enable registers. The Quick Trip SFRs are located in module 5.

9.2.1 – Quick Trip Control Register (QTCN)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	QTEN	-	-	-	-	RW_LST	-	-	LTHT	QTIDX[3:0]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	rw	r	r	r	r	rw	r	r	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:13	-	**Reserved**. The user should write these bits to '0'.
12	QTEN	**Quick Trip Enable.** When this bit is set to '1', it enables the quick trip operation. After setting the QTEN bit to '1', there is an initial delay for 120 core clock to wake up the quick trip circuitry. When this bit is set to '0', it disables the quick trip operation.
11:8	-	**Reserved**. The user should write these bits to '0'.
7	RW_LST	**Read List Register:** When this bit is set to '1', it selects one of the sixteen list register (addressed by QTIDX[3:0], see below) in the list configuration. When this bit is set to '0', the low or high threshold register (depends upon the LTHT bit) are configured.
6:5	-	**Reserved**. The user should write these bits to '0'.
4	LTHT	**Low or High Threshold Select:** This bit is used only when RW_LST is set to '0'. This bit is used to select low or high threshold read or write. When the LTHT bit is set to '0', it points to the low threshold configuration register list. When this bit is set to '1', it points to the high threshold configuration register list. The address of low or high threshold configuration is addressed by QTIDX[3:0] bits.
3:0	QTIDX[3:0]	**Quick Trip Index Select.** These bits together with LTHT and RW_LST bits select the source or destination address for the QTDATA register access. When the RW_LST and LTHT bits are set to '0', the QTIDX[3:0] bits address to one of the sixteen low threshold register for read or write. When RW_LST = 0 and LTHT = 1, the QTIDX[3:0] bits address one of the sixteen high threshold register for read or write. When RW_LST = 1 (irrespective of LTHT bit), the QTDATA register selects the list register addressed by QTIDX[3:0] bits for read and write operation. A read or write operation on the QTDATA register reads or writes to the list register addressed by QTIDX[3:0]. These bits are auto incremented on any read or write operation to the QTDATA register.

9.2.2 Quick Trip Data Register (QTDATA)

The Quick Trip Data register is used with LTHT, RW_LST and QTIDX[3:0] bits to configure thresholds and list configurations. The QTDATA register selects the list register and threshold registers addressed by QTIDX[3:0] bits for read and write operation.

Threshold registers are selected when the bit RW_LST is set to '0'. List registers are selected when the bit RW_LST is set to '1'. See the following tables for RW_LST = 0 and RW_LST = 1.

QTDATA Register map when **RW_LST = 0** (in the QTCN Register)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-						LOW or HIGH THRESHOLD									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:10	-	**Reserved**. The user should write these bits to '0'.
9:0	QTDATA[9:0]	**a. Low Threshold Configuration** (When the LTHT bit in the QTCN register is set to '0') The QTDATA register selects low threshold register addressed by QTIDX[3:0] bits in the QTCN register for read and write operation. The low threshold registers are 10-bit wide and the upper QTDATA [15:10] bits are ignored and return 0. **b. High Threshold Configuration** (When the LTHT bit in the QTCN register is set to '1') The QTDATA selects high threshold register addressed by QTIDX[3:0] bits in the QTCN register for read and write operation. The high threshold registers are 10-bit wide and upper QTDATA [15:10] bits are ignored and return 0.

QTDATA Register map when **RW_LST = 1** (in the QTCN Register)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	-	-	-	DIFF	CHSEL[3:0]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:5	-	**Reserved**. The user should write these bits to '0'.
4	DIFF	**Mode Selection (DIFF):** This bit selects the Quick trip input channel source either as single-ended or differential mode. When this bit is set to '0', quick trip channel (addressed by CHSEL[3:0]) is selected as "single-ended" input. When this bit is set to '1', quick trip channel (addressed by CHSEL[3:0]) is selected as "Differential Mode" input. See the below table for various quick trip input channel configuration in single-ended as well as differential mode.
3:0	CHSEL [3:0]	**QT Channel Select (CHSEL [3:0]):** These bits select the Quick trip input channel source for the quick trip list configuration.

CHSEL[3:0]	DIFF = 0 Channel Selected Single-Ended	DIFF = 1 Channel Selected Differential Mode
0000	ADC-S0	ADC-D0P – ADC-D0N
0001	ADC-S1	ADC-D1P – ADC-D1N
0010	ADC-S2	ADC-D2P – ADC-D2N
0011	ADC-S3	ADC-D3P – ADC-D3N
0100	ADC-S4	ADC-D4P – ADC-D4N
0101	ADC-S5	ADC-D5P – ADC-D5N
0110	ADC-S6	ADC-D6P – ADC-D6N
0111	ADC-S7	ADC-D7P – ADC-D7N
1000	ADC-S8	NOT VALID
1001	ADC-S9	NOT VALID
1010	ADC-S10	NOT VALID
1011	ADC-S11	NOT VALID
1100	ADC-S12	NOT VALID
1101	ADC-S13	NOT VALID
1110	ADC-S14	NOT VALID
1111	ADC-S15	NOT VALID

9.2.3 Low Trip Interrupt Lower Register (LTIL)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	IE[7:0]								IF[7:0]							
Reset																
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:8	IE[7:0]	**Low Trip Interrupt Enable**. This register is used to enable/mask the corresponding LTIL register interrupts. For Example, if LTIL = 0x0100 then Quick Trip list 0 can generate an interrupt when LTIL LSB is set to '1' and all other interrupts from LTIL are ignored. Similarly, if LTIL = 0xFF00, then all 8 interrupts from LTIL generate interrupts.
7:0	IF[7:0]	**Low Trip Interrupt Flag**. The corresponding bit of the Low Trip Interrupt register is set when a low threshold trip is occurred on a channel list register. In other words, when voltage across channel is less than the low threshold configuration for the channel. For example, if a low trip occurs on the list register 0 then LTIL is set to 0x0001. If the corresponding IE bit is also '1', and then this generates an interrupt. Software should clear the Low Trip Interrupt Flag once it is set by hardware. Setting this bit to '1' by software generates an interrupt if enabled.

9.2.4 High Trip Interrupt Lower Register (HTIL)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	IE[7:0]								IF[7:0]							
Reset																
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:8	IE[7:0]	**High Trip Interrupt Enable**. This register is used to enable/mask the corresponding HTIL register interrupts. For example, if HTIL = 0x0100 then Quick Trip list 0 can generate an interrupt when HTIL LSB is set to '1' and all other interrupts from HTIL are ignored. Similarly, if HTIL = 0xFF00, then all 8 flags from HTIL generate interrupts.
7:0	IF[7:0]	**High Trip Interrupt Flag**. The corresponding bit of the High Trip Interrupt register is set when a high threshold trip is occurred on a channel list register. In other words, when voltage across channel is greater than the high threshold configuration for the channel. For example, if a high trip occurs on the list register 0 then HTIL is be set to 0x0001. If the corresponding IE bit is also '1', and then this generates an interrupt. Software should clear the how trip interrupt flag once it is set by hardware. Setting this bit to '1' by software generates an interrupt if enabled.

9.2.5 Low Trip Interrupt High Register (LTIH)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	IE[15:8]								IF[15:8]							
Reset																
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:8	IE[15:8]	**Low Trip Interrupt Enable**. This register is used to enable/mask the corresponding LTIH register interrupts for upper 8 comparisions. For example, if LTIH = 0x0100 then Quick Trip list 8 can generate an interrupt when LTIH LSB is set to '1' and all other interrupts from LTIH are ignored. Similarly, if LTIH = 0xFF00, then all 8 flags from LTIH generate interrupts.
7:0	IF[15:8]	**Low Trip Interrupt Flag**. The corresponding bit of the low trip interrupt register is set when a low threshold trip is occurred on a channel list register. In other words, when voltage across channel is less than the low threshold configuration for the channel. For example, if a low trip occurs on the list register 8 then LTHI is set to 0x0001. If the corresponding IE bit is also '1', and then this generates an interrupt. Software should clear the Low Trip Interrupt Flag once it is set by hardware. Setting this bit to '1' by software generates an interrupt if enabled.

9.2.6 High Trip Interrupt High Register (HTIH)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	IE[15:8]								IF[15:8]							
Reset																
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:8	IE[15:8]	**High Trip Interrupt Enable**. This register is used to enable/mask the corresponding HTIH register interrupts for the upper 8 comparisons. For Example, if HTIH = 0x0100 then Quick Trip list 8 can generate an interrupt when HTIH LSB is set to '1' and all other interrupts from HTIH are ignored. Similarly, if HTIH = 0xFF00, then all 8 flags from HTIH generate interrupts.
7:0	IF[15:8]	**High Trip Interrupt Flag**. The corresponding bit of the High Trip Interrupt register is set when a high threshold trip is occurred on a channel list register. In other words, when voltage across channel is more than the high threshold configuration for the channel. For example, if a high trip occurs on the list register 8 then HTIH is set to 0x0001. If the corresponding IE bit is also '1', and then this generates an interrupt. Software should clear the High Trip Interrupt Flag once it is set by hardware. Setting this bit to '1' by software generates an interrupt if enabled.

9.2.7 – Quick Trip List Register (QTLST)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	QTSTART[3:0]				-	-	-	-	QTEND[3:0]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw	r	r	r	r	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:12	-	**Reserved**. The user should write these bits to '0'.
11:8	QTSTART[3:0]	**Quick Trip Configuration Start Address Bits [3:0].** These bits select the start address of quick trip channel list.
7:4	-	**Reserved**. The user should write these bits to '0'.
3:0	QTEND[3:0]	**Quick Trip Configuration Ending Address Bits [3:0].** These bits select the stop address of quick trip channel list.

SECTION 10 – I²C-COMPATIBLE MASTER INTERFACE

The DS4830A provides an I²C-compatible master controller that allows the DS4830A to communicate with a slave device. The I²C master interface can be setup to provide system interrupts after each I²C event.

10.1 – Detailed Description

10.1.1 – Description of Master I²C Interface

The master I²C interface uses the MSDA and MSCL pins. These pins are the master I²C controller's connection to the SDA and SCL pins of an I²C bus. In addition to driving these pins, the I²C master port also senses the state of both MSDA and MSCL. This allows the I²C master port to offer bus error detection and allows a slave device to clock stretch.

Unless explicitly stated, all references to SDA and SCL in this section refer to the SDA and SCL lines of the I²C bus, not the DS4830A's I²C slave interface SDA and SCL pins.

10.1.2 – Default Operation

The I²C master controller is disabled by default. The I²C master controller is enabled by setting the I2CEN and I2CMST bits in the I2CCN_M register to a 1. Prior to the I²C master controller being used for communication, some software setup is required. This setup includes setting the clock rate, timeout period, and which I²C events should generate interrupts. The DS4830A master I²C controller is not intended to be used on an I²C bus that has multiple masters connected to the bus.

10.1.3 – I²C Clock Generation

In an I²C system, the master is responsible for generating the SCL signal. The DS4830A I²C Master Controller provides complete control over the clock rate and duty cycle. The I²C Master Controller generates SCL from the system clock. The bit rate is controlled by the I²C Clock Control Register (I2CCK_M).

The high period of SCL clock is defined by the high byte of the I²C Clock Control register (I2CCKH) whereas the low period of SCL is defined by the low byte (I2CCKL). The minimum clock high period is three system clocks while the minimum low period has to be at least five system clock periods. The I²C clock characteristics can be defined by the following equations:

- SCL Low Time = System Clock Period x (I2CCKL[7:0] + 1)
- SCL High Time = System Clock Period x (I2CCKH[7:0] + 1)
- I²C Clock Rate = System Clock Frequency/(I2CCLK[7:0] + I2CCKH[7:0] + 2)

One feature of the master I²C controller is that it also monitors SCL while the clock is being output. This allows the controller to ensure that the SCL level is at the desired level prior to beginning the count for SCL Low or High Time. Figure 10-1 illustrates the SCL sampling performed by the master I²C controller. When SCL is released by the master I²C controller, the rise time is determined by the capacitive loading and pullup resistance on the SCL line. When the controller senses the SCL line has reached a high logic level, the count for SCL High Time is started. The same is true for a falling edge. The SCL Low Time is only started after the controller senses the SCL line at a low logic level.

Figure 10-1 also illustrates that the calculated I²C clock period is not exactly accurate because the rise and fall time of SCL is not taken into consideration. The actual clock period will be the period set by the I2CCK_M register plus any rise and fall time.

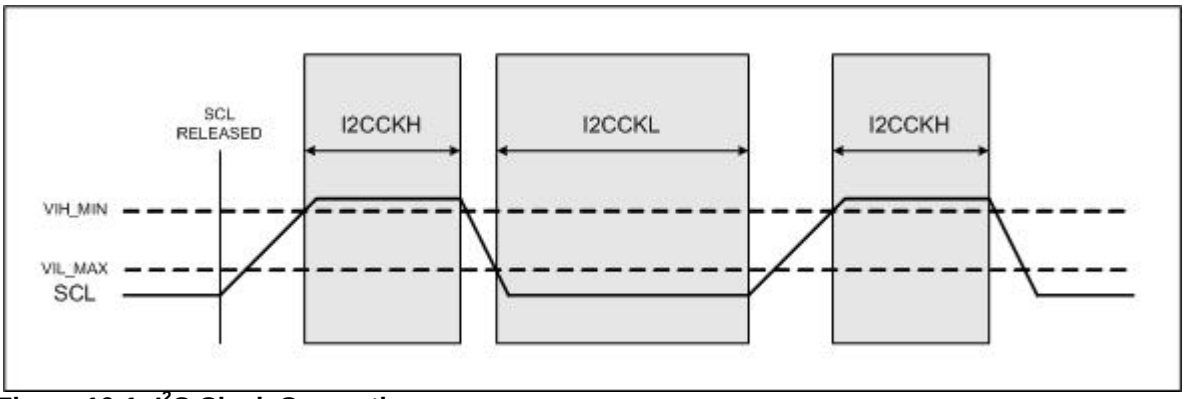


Figure 10-1: I²C Clock Generation

The master I²C controller's ability to monitor the state of SCL allows the master to operate with slave devices that stretch the clock. A slave device may clock stretch, or hold SCL low, while it is busy or processing data. The master I²C controller will always release SCL after holding it low for the SCL Low Time duration. By monitoring the state of SCL, the master I²C controller realizes that SCL has not been released and does not begin the SCL High Time count. Only after the master controller detects a high state on SCL will begin the I2CCKH count. This is illustrated in Figure 10-2.

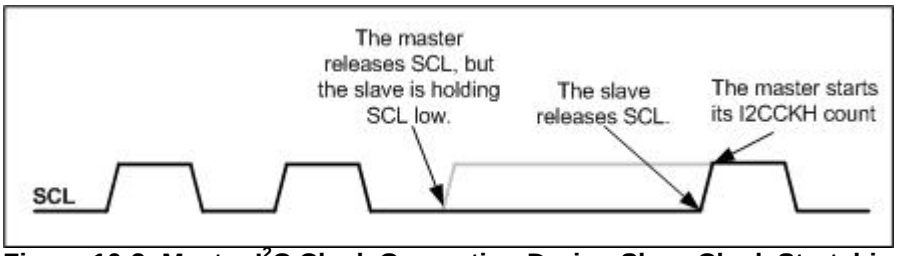


Figure 10-2: Master I²C Clock Generation During Slave Clock Stretching

10.1.4 – Timeout

The Master I²C Controller has a programmable timeout function that allows the controller to recover from a bus error. The timeout period is determined by the setting of the I²C Master Timeout Register (I2CTO_M) using the following equation:

$$\text{Timeout Period} = I^2C\ \text{Bit Rate} \times (I2CTO[7:0]+1)$$

where I²C Bit Rate is determined by the setting of the I2CCK_M register. The timeout can be disabled by clearing the I2CTO_M register to 0. The I²C timeout timer starts counting:

- When the I2CSTART bit is set to 1. The I²C controller will monitor the bus status until it can generate a START condition. The I²C bus is considered busy if another master has generated a START condition and no corresponding STOP has been detected (the I2CBUS bit is set to 1) or the SCL line is low. If the bus remains busy for a period longer than specified in the timeout register, the I²C controller concludes that there is a bus error and will set the I2CTOI flag.

  If the I²C Controller has started a transfer (after the first bit rising edge), it will wait for the current byte transfer to finish (after the 9th bit (acknowledge) has been transmit) before generating the START condition. In this case, the timeout timer will start counting after the end of the 9th bit low time.
- After the master I²C controller attempts to generate a STOP condition. If a STOP is not detected (I2CSPI = 1) during the timeout period, the I2CTOI flag will be set.

  If the I2C Controller has started a transfer (after the first bit rising edge), it will wait for the current byte transfer to finish (after the 9th bit (acknowledge) has been transmit) before generating the STOP condition. In this case, the timeout timer will start counting after the end of the 9th bit low time.

- Whenever SCL goes low. If the SCL line is low for a period longer than specified in the timeout register, the I²C controller concludes that there is a bus error and will set the I2CTOI flag.

For all of these cases, when the I^2C timeout period is reached, the I2CTOI flag will be set. The setting of I2CTOI can generate an interrupt if enabled. If the master I^2C controller is in the process of transferring data when the timeout occurs, the controller will abort the current transfer and clear the I2CBUSY flag. The I2CBUS flag will continue to be set until a STOP condition is detected or I2CEN is set to 0.

10.1.5 – Generating a START

To initiate a data transfer, the I^2C master controller must first issue a START command. The master I^2C controller's flow when attempting to issue a START command is shown in Figure 10-3. A START command is generated by setting the I2CSTART bit to 1. The I^2C controller will then determine the state of the I^2C bus. If the bus is busy (I2CBUS = 1), the controller will not generate a START until the bus is available. The I^2C bus is considered busy if another master has generated a START condition and no corresponding STOP has been detected (the I2CBUS bit is set to 1) or SCL is being held low.

If the bus is not busy, the I^2C master controller will attempt to generate a START. Because the SDA line is feedback into the device, when the master generates a START, it can also detect the START condition. When a START condition is detected, the I^2C START interrupt flag (I2CSRI) will be set and an interrupt will be generated if enabled. The I2CBUS bit will be set to indicate that the I^2C bus is now in use and the I2CSTART bit will be cleared.

When the I2CSTART bit is set to a 1, the I^2C controller will start its timeout timer if enabled (I2CTO_M ≠ 0). If the timer expires before the START can be generated, the I^2C timeout interrupt flag (I2CTOI) will be set and an interrupt is generated if enabled. If a timeout occurs, the I^2C master controller will reset to an idle state and the I2CSTART bit will be cleared.

If the I2CSTART bit is set when the I^2C Controller is in the middle of a byte transfer (after the first bit rising edge), the controller will wait for the current byte transfer to finish (after the 9th bit) before generating the START condition. In this case, the timeout timer will not start counting until after the end of the 9th bit low time.

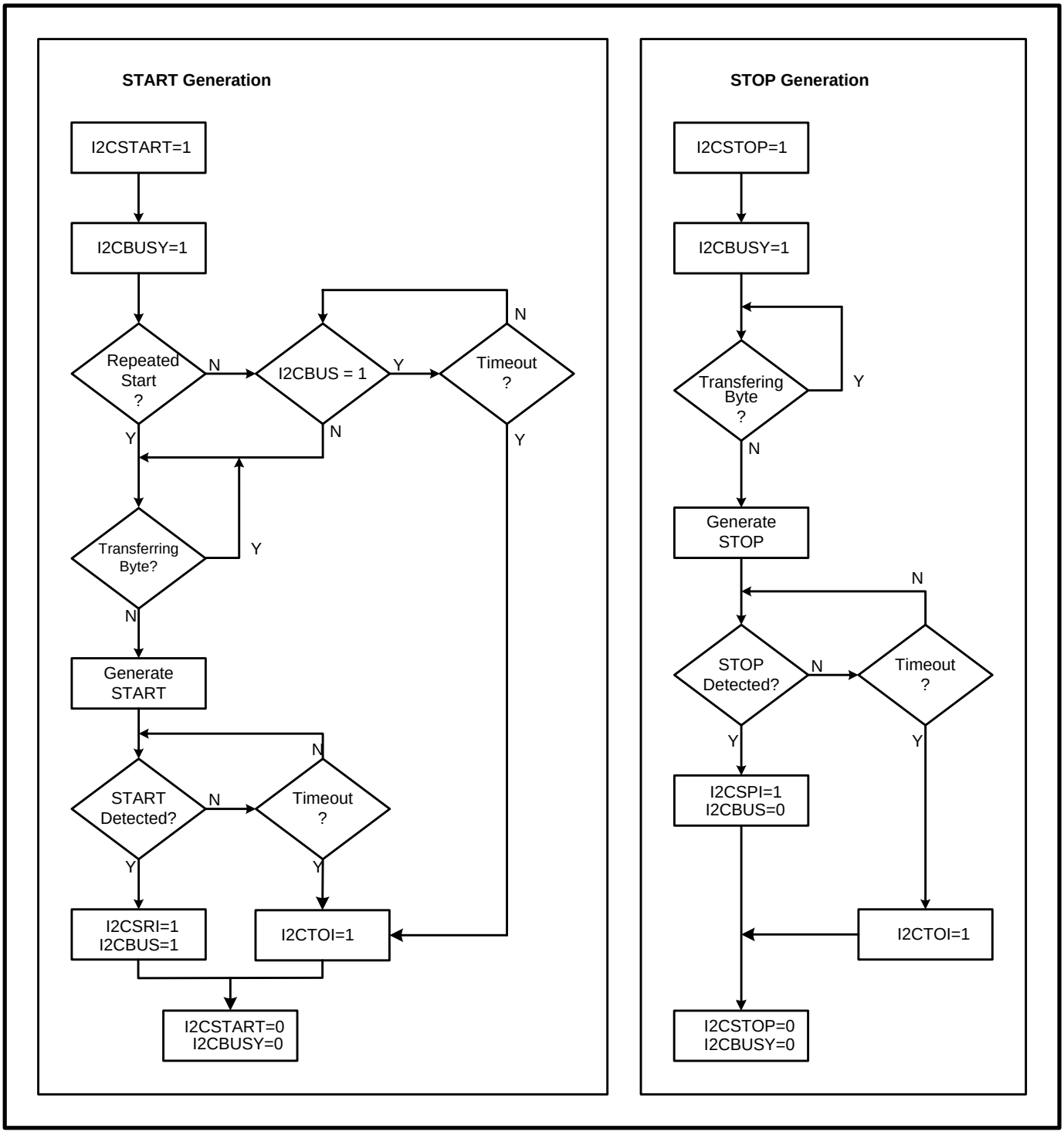


Figure 10-3: Master I²C-Generated START and STOP

10.1.6 – Generating a STOP

To end an I^2C transfer, a STOP must be transmitted. A STOP is generated by setting the I2CSTOP bit. The master I^2C controller's flow when attempting to issue a STOP command is shown in Figure 10-3.

If the I2CSTOP bit is set when the I^2C Controller is in the middle of a byte transfer (after the first bit rising edge), it will wait for the current byte transfer to finish (after the 9th bit) before generating the STOP condition.

Because the SDA line is feedback into the device, when the master generates a STOP, it will also detect the STOP condition. When a STOP condition is detected, the I^2C STOP interrupt flag (I2CSPI) will be set and an interrupt will be generated enabled. The I2CBUS bit will be cleared to indicate that the I^2C bus is now idle and the I2CSTOP bit will be cleared.

When the master I^2C controller attempts to generate the STOP condition, it will also start the timeout timer if this feature is enabled. If a timeout is generated before the STOP condition is detected, a timeout will occur. When a timeout occurs, the I2CTOI bit will be set, which can generate an interrupt if enabled, and the I2CSTOP bit will also be cleared to 0.

10.1.7 – Transmitting a Slave Address

The first byte after an I^2C START or restart condition is the slave address byte. This byte, which is transmitted by the master, contains seven bits of slave address followed by the R/$\overline{W}$ bit. The transmission of the slave address begins with writing 7-bit slave address + the R/$\overline{W}$ bit to I2CBUF_M.

Figure 10-4 shows the format for slave address 36h in write mode. The address bits A[6:0], which is the slave address the R/$\overline{W}$ bit is written to I2CBUF_M[6:0]. Bit 0 of I2CBUF_M is copied to bit 0 I2CMODE of the I2CSLA_M register. When bit 0 is '1', the I^2C master is operating in receiver mode (data read from slave). When bit 0 is '0', the I^2C master is operating in transmitter mode (data write to slave).

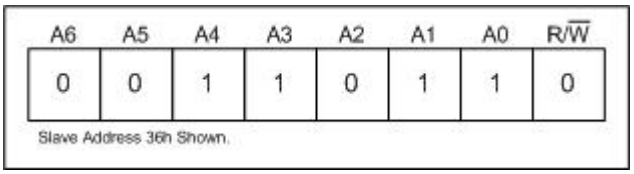


Figure 10-4: Slave Address Format

After the slave address has been written to I2CBUF_M, the I^2C master controller will set the I2CBUSY bit to indicate the controller is actively participating in a transaction. The eight bits in I2CBUF_M[7:0] will be transmitted on SDA. The data for the 8th bit transmit, which is the R/$\overline{W}$ bit, is copied in the I2CMODE bit of the I2CSLA_M register. The I^2C master then issues the 9th clock, which is for the acknowledge bit, and reads SDA for an acknowledgment from a slave device. The I^2C master controller then performs the following steps. This is illustrated in Figure 10-5.

- Set the I2CNACKI bit with the value of the received acknowledgement.
- The I2CTXI bit will then be set to indicate a byte was transmit.
- Clear the I2CBUSY flag.

Upon transmitting the slave data byte (7 bits of slave address + R/$\overline{W}$ bit + acknowledge), the I^2C master controller will enter one of the three states.

- Data Transmit: The I2CMODE (R/$\overline{W}$) bit was set to a 0, indicating that the master will be writing data to a slave device. The DS4830A will retain control of the SDA line.
- Data Receive: The I2CMODE (R/$\overline{W}$) bit was set to a 1, indicating that the master will be receiving data from a slave. The DS4830A releases control of SDA to allow a slave device to output data. The DS4830A Master I^2C controller automatically begins clocking bytes of data from the slave.
- The slave address was NACKed. The master I^2C controller will retain control of SDA and is able to transmit data.

10.1.8 – Transmitting Data

The DS4830A I^2C Master Controller enters into data transmission mode after transmitting a slave address with the R/$\overline{W}$ bit (I2CMODE) set to a 0. The steps of data transmission are shown in Figure 10-5. Data transmission is started by software loading a byte of data into the I2CBUF_M register. Loading I2CBUF_M causes the I2CBUSY bit

to be set. Once set, writes to I2CBUF_M will be ignored. The first bit of data (most significant bit) will be shifted to SDA when SCL is low. Each of the next seven bits will then be shifted following high to low transitions of SCL.

Following the 8th bit of data (least significant bit) being shifted to SDA, the SDA line will be released by the DS4830A master controller. This allows the slave to signal an ACK or NACK during the 9th clock cycle. The DS4830A I2C master controller samples the acknowledge bit following the 9th SCL rising edge. After the acknowledge bit is sampled, the DS4830A I2C master controller will perform the following tasks:

- Set or clear the I2CNACKI flag to reflect the received acknowledge bit. The setting of I2CNACKI can generate an interrupt if enabled.
- Set the I2CTXI flag to indicate that the I2C master controller transmit a complete byte. This can generate an interrupt if enabled.
- Clear the I2CBUSY flag to indicate that the I2C master controller is not actively participating in the transfer of data.

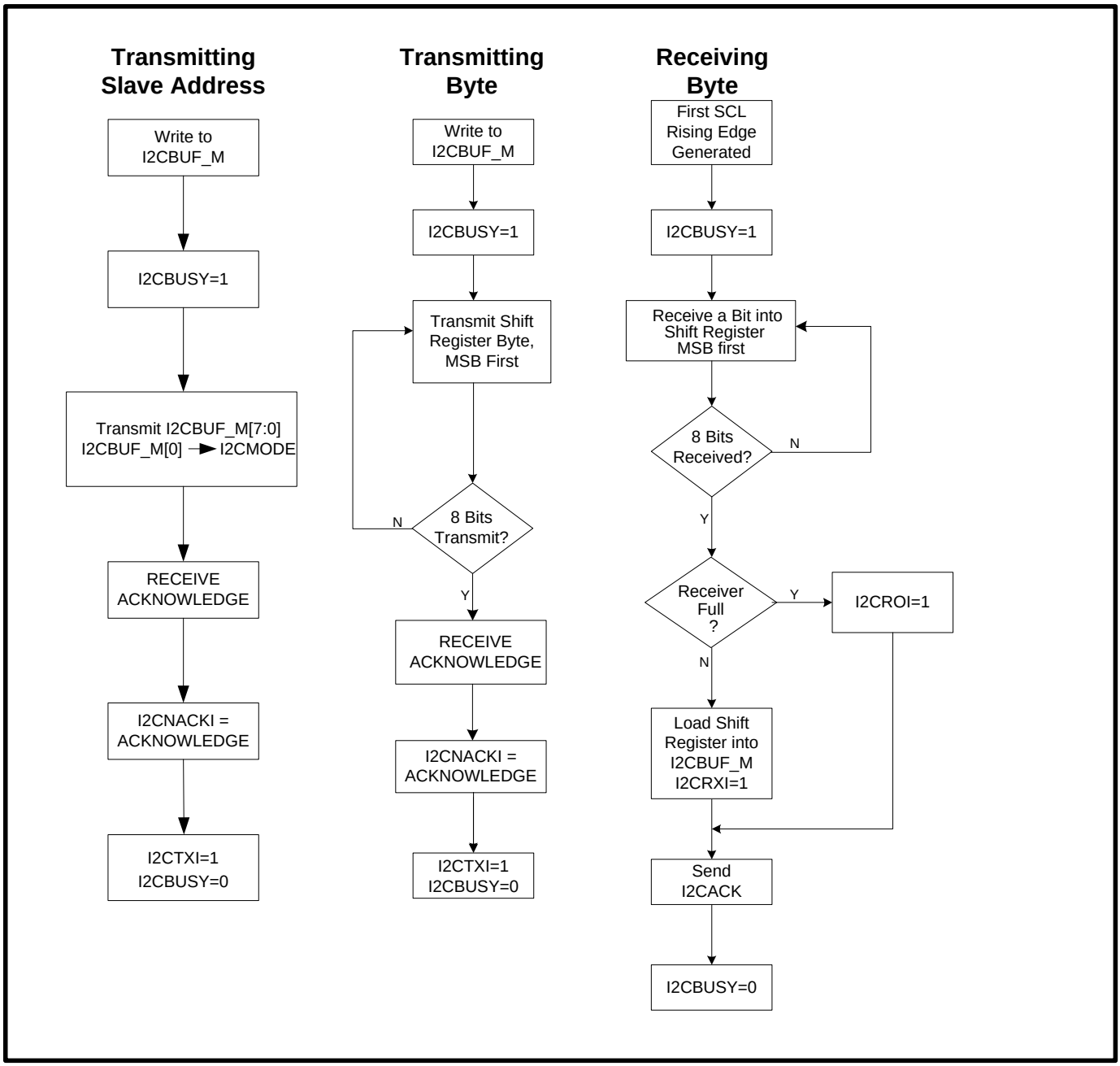


Figure 10-5: Master I2C Data Flow

10.1.9 – Receiving Data

The DS4830A I^2C Master Controller enters data reception mode after transmitting a slave address with the R/$\overline{W}$ bit (I2CMODE) set to a 1. The steps of data reception are shown in Figure 10-5. After transmitting the slave address, the master controller will switch to receiver mode and automatically begin outputting SCL clock pulses and shifting in data from SDA.

When receiving data, the DS4830A I^2C master controller uses a double buffer consisting of the I2CBUF_M register and the shift register. This allows the I^2C module to continue receiving data while the previous data byte is being processed. When a full byte of data (8 bits) has been received by the I^2C master controller, the master must send an acknowledgement to the slave. This occurs during the 9th clock cycle when the value in I2CACK is transmitted to the slave.

After a complete byte (8 bits) of data is received, the I^2C master controller will attempt to copy the received data from the shift register to I2CBUF_M. There are two possible results from the I^2C master controller's attempt to copy the shift register to I2CBUF_M.

1. If I2CBUF_M is empty, the I^2C master controller will copy the data from the shift register into I2CBUF_M. The I2CRXI flag will be set to indicate a received byte is ready to be read. The setting of I2CRXI can generate an interrupt if enabled.

2. If I2CBUF_M is full, the data in the shift register cannot be copied into I2CBUF_M. This causes a receive overrun condition. The receive overrun flag, I2CROI, will be set which can generate an interrupt if enabled. I2CBUF_M will be full if it was not read by software following the reception of a previous byte.

After receiving a byte of data and the I2CRXI flag being set, it is up to software to read I2CBUF_M prior to a second byte being received. Reading the I2CBUF_M register returns the received data and also clears I2CBUF_M. As long as the previous byte of data is read from I2CBUF_M before the next byte has completed, receive overrun will not occur.

When receive overrun is detected and I2CROI bit is set, the DS4830A master I^2C controller will stop outputting SCL clocks and not clock the acknowledge bit until the receive overrun condition is cleared. The receive overrun condition and the I2CROI flag can only be cleared by software reading the first byte received from I2CBUF_M. When the receive overrun condition is cleared, the I^2C master controller will copy the second byte that was received into I2CBUF_M, and again set I2CRXI to indicate a byte of data was received. The I^2C master controller will resume clocking SCL after satisfying SCL low time requirements.

The master I^2C controller will continue to automatically clock bytes of data until any of the following conditions occur.

1) A receive overrun condition occurs.
2) A STOP command is issued (I2CSTOP=1) prior to the master I^2C controller beginning to clock a new byte.
3) The master I^2C controller has clock stretching enabled and the clock is currently being held low by the master.

10.1.10 – I²C Master Clock Stretching

The Master I²C Controller is capable of clock stretching at the end of each transfer cycle. Clock stretching is when SCL is held low. If the I²C Clock Stretch Enable bit (I2CSTREN) is set to a 1, the I²C controller will hold SCL low after the clock pulse defined by the I²C Clock Stretch Select bit (I2CSTRS). If I2CSTRS=0, the I²C controller will hold SCL low after the falling edge of the 9th clock pulse. If I2CSTRS=1, the I²C controller will hold SCL low after the falling edge of the 8th clock pulse. When the I²C controller is holding SCL low, the I²C Clock Stretch Interrupt flag (I2CSTRI) will be set, which can generate an interrupt if enabled. The I²C slave controller will hold SCL low until I2CSTRI is cleared to 0 by software.

If clock stretching is enabled after the 8th clock pulse, the master I²C controller will continue outputting the value of the I2CACK bit until clock stretching is released by clearing I2CSTRI. This allows software time to examine the data that was received prior to sending an ACK or NACK to the slave. The continuous output of I2CACK will occur even if the master I²C controller is transmitting data. In this mode, the slave should be sending the acknowledgement. To allow the slave to send the proper acknowledgement, the I2CACK bit should be set to a 1, which prompts the master I²C controller to release SDA.

The Master I²C Controller may need to use clock stretching when receiving data from a slave. When receiving data, the master I²C controller automatically generates clock pulses. Without using clock stretching, this automatic clock generation is only halted when a STOP command is issued or a receive overrun occurs. If clock stretching is enabled, software can control when each byte of data is clocked from the slave device.

10.1.11 – Resetting the I²C Master Controller

The I²C master controller can be reset by disabling the I²C master controller by writing '0' at I2CEN = 0 in the I2CCN I2CCN_M register. A reset will force the master I²C controller to release both MSDA and MSCL if they are being held low by the I²C master controller. A reset may reset few or all bits of I2CCN, I2CST and I2CBUF I²C registers, and reset the I²C master controller's internal state machine. Following a reset, the I²C master controller must be re-initialized before it can be used again.

After a reset, the master I²C controller will be in a known state but the slave devices may be in an unknown state. It is recommended that the master I²C controller attempts to reset the slave devices prior to beginning communication. A reset of slave devices can be performed by outputting at least 9 clock pulses on the MSCL line while MSDA is high. This easiest way to achieve this is to use MSDA and MSCL as GPIO pins (see the GPIO section) while the master I²C controller is disabled (I2CEN=0). After the 9 clock pulses, a STOP command should be generated. This can be done either using GPIO, or by enabling the master I²C controller and generating a STOP.

10.1.12 – Alternate Location

The DS4830A has 3-Wire, SPI and I^2C Master on the same pins and some application may need the I^2C Master and 3-Wire or SPI interfaces. To support such applications, the DS4830A provides an I^2C Master alternate location. When I2CCN_M bit 12 is set to '1', the DACPW4 and DACPW5 pins are used as I^2C SDA and I^2C SCL pins as I^2C Master alternate locations.

10.1.13 – Operation as a Slave

The DS4830A contains two I^2C interfaces, the master (MSDA and MSCL) and slave (DS4830A SDA and SCL pins). These are two totally separate blocks within the DS4830A. However, both of the blocks are identical. Because of this, it is possible to operate the master as a slave and also operate the slave as a master.

To operate the master (MSDA and MSCL) as a slave I^2C interface, the I2CMST bit in I2CCN_M needs to be set to a 0. When the master is operating as a slave, it will use the same registers (I2CCN_M, I2CST_M, etc.) that it uses for master operation. However, the bits in these registers will have different functionality, as described in the I^2C Slave Interface Section. The I2CCN_M.SMB_MOD bit only affects the interface when it is operating as a slave. See the I^2C Slave Interface section for details on initializing and using a slave I^2C interface. The I^2C Master can be used in the slave mode and allows two user programmable slave addresses using I2CSLA_M and I2CSLA2_M slave address register. The I2CSLA2_M can be enabled by setting ADD2EN bit in the I2CCN_S register. When I^2C Master Interface is used as the I^2C slave mode, it does not have any TX Page or Receive FIFO which are available in the I^2C slave interface (Section 11) only.

10.1.14 – GPIO

When the I^2C master controller is disabled (I2CEN=0), the MSDA and MSCL pins can be used as GPIO pins. The MSDA pin is mapped to GPIO port P1.0 and MSCL is mapped to GPIO port P1.1. When used as GPIO outputs, the MSDA and MSCL pins are push pull outputs. See the General-Purpose I/O Section for more information on using MSDA and MSCL as GPIO pins.

10.2 – I²C Master Controller Register Description

Following are the registers that are used to control the I²C Master Interface, which is the MSDA and MSCL pins. These registers are used to control the I²C master interface if it is operating as either a master or slave. The bit descriptions below detail how to use these registers when operating in master mode. When operating in slave mode, some of the bits and registers have different functionality. See the I²C Slave Interface for more information on how to control the I²C Master Interface when it is operating as a slave.

10.2.1 – I²C Master Control Register (I2CCN_M)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	I2CM_ALT	ADD2EN	SMB_MOD	I2CSTREN	I2CGCEN	I2CSTOP	I2CSTART	I2CACK	I2CSTRS	-	-	I2CMST	I2CEN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Access	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw*	rw*

* Unrestricted Read. Unrestricted write access when I2CBUSY=0. Writes to I2CEN are disabled when I2CBUSY=1.

BIT	NAME	DESCRIPTION
15:11	Reserved	**Reserved**. The user should write 0 to these bits.
12	I2CM_ALT	**I²C Master Alternate Location:** When this bit is set to '1', the DACPW4 and DACPW5 will be used as SDA and SCL respectively as I²C Master alternative location.
11	ADD2EN	**Slave Address2 Enable:** This bit has no function in master mode. In the slave mode, setting this bit to '1', enables I2CSLA2_M slave address.
10	SMB_MOD	**SMBus Mode Enable.** This bit enables the SMBUS timeout feature only when the master I²C interface (MSDA and MSCL) is enabled to be a slave interface. See the Operation as a Slave section for more details.
9	I2CSTREN	**I²C Master Clock Stretch Enable.** Setting this bit to '1' will stretch the clock (hold SCL low) at the end of the clock cycle specified by I2CSTRS. Clearing this bit disables clock stretching.
8	I2CGCEN	**I²C General Call Enable.** This bit has no function when operating in master mode.
7	I2CSTOP	**I²C STOP Enable.** Setting this bit to '1' generates a STOP condition. This bit is automatically cleared to '0' after the STOP condition has been generated. The setting of I2CSTOP will start the timeout timer if enabled. If the timeout timer expires before the STOP condition is generated, the I2CTOI flag is set, which can generate an interrupt if enabled. A timeout will also clear the I2CSTOP bit.
6	I2CSTART	**I²C START Enable.** Setting this bit to '1' generates a START or repeated START condition. This bit is automatically cleared to '0' after the START condition has been generated. The setting of I2CSTART will start the timeout timer if enabled. If the timeout timer expires before the START condition is generated, the I2CTOI flag is set, which can generate an interrupt if enabled. A timeout will also clear the I2CSTART bit.
5	I2CACK	**I²C Master Data Acknowledge Bit.** This bit selects the acknowledge bit returned by the master I²C controller while acting as a receiver. Setting this bit to '1' will generate a NACK (leaving SDA high). Clearing the I2CACK bit to '0' will generate an ACK (pulling SDA LOW) during the acknowledgement cycle. This bit will retain its value unless changed by software or hardware.
4	I2CSTRS	**I²C Master Clock Stretch Select.** Setting this bit to '1' will enable clock stretching after the falling edge of the 8th clock cycle. Clearing this bit to '0' will enable clock stretching after the falling edge of the 9th clock cycle. This bit has no effect when clock stretching is disabled (I2CSTREN=0).
3:2	Reserved	**Reserved**. The user should write 0 to these bits.
1	I2CMST	**I²C Master Mode Enable.** Setting this bit to '1' will enable I²C master functionality on the MSDA and MSCL pins. Setting this bit to '0' enables I²C slave functionality. See the I²C Slave Interface section for more details.
0	I2CEN	**I²C Enable.** This bit enables the I²C Master interface. When set to '1', the I²C Master Interface is enabled. When cleared to '0', the I²C function is disabled.

Notes: The I2CSTART and I2CSTOP are mutually exclusive. If both bits are set at the same time, it is considered an invalid operation and the I²C controller ignores the request and resets both bits to 0. Setting the I2CSTART bit to 1 while I2CSTOP = 1 is an invalid operation and is ignored, leaving the I2CSTART bit cleared to 0.

10.2.2 – I²C Master Status Register (I2CST_M)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	I2CBUS	I2CBUSY	-	I2CAMI2	I2CSPI	I2CSCL	I2CROI	I2CGCI	I2CNACKI	-	I2CAMI	I2CTOI	I2CSTRI	I2CRXI	I2CTXI	I2CSRI
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r*	r*	r	rw	rw	r*	rw	rw	rw*	r	rw	rw	rw*	rw*	rw	rw

* Set by hardware only.

BIT	NAME	DESCRIPTION
15	I2CBUS	**I²C Master Bus Busy.** This bit is set to '1' when a START/repeated START condition is detected and cleared to 0 when the STOP condition is detected. This bit is reset to '0' when I2CEN=0. This bit is controlled by hardware and is read only.
14	I2CBUSY	**I²C Master Busy.** This bit is used to indicate the current status of the I²C controller. The I2CBUSY is set to '1' when the I²C controller is actively participating in a transaction. This bit is controlled by hardware and is read only.
13	Reserved	**Reserved**. The user should write 0 to this bit.
12	I2CAMI2	**I²C Address Match2 Interrupt Flag.** This bit has no function when operating in master mode. In the slave mode, this bit is set when I2CSLA2_M address is matched. This bit must be cleared to '0' by software once set.
11	I2CSPI	**I²C Master STOP Interrupt Flag.** This bit is set to '1' when a STOP condition is detected. This bit must be cleared to '0' by software once set. Setting this bit to '1' by software will cause an interrupt if enabled.
10	I2CSCL	**I²C Master SCL Status.** This bit reflects the logic state of the SCL signal. This bit is set to '1' when SCL is at a high logic level and cleared to '0' when SCL is at a low logic level. This bit is controlled by hardware and is read only.
9	I2CROI	**I²C Master Receiver Overrun Flag.** This bit indicates a receive overrun when set to '1'. This bit is set to '1' if the receiver has received two bytes since the last software reading of I2CBUF_M. This bit can only be cleared to '0' by software reading I2CBUF_M. Setting this bit to '1' by software will cause an interrupt if enabled.
8	I2CGCI	**I²C General Call Interrupt Flag.** This bit has no function when operating in master mode.
7	I2CNACKI	**I²C Master NACK Interrupt Flag.** This bit is set by hardware to '1' if a NACK was received from a slave or a 0 if an ACK was received from a slave. The setting of this bit to '1' will cause an interrupt if enabled. This bit can be cleared to '0' by software once set. This bit is set by hardware only.
6	Reserved	**Reserved.** The user should write 0 to this bit.
5	I2CAMI	**I²C Address Match Interrupt Flag.** This bit has no function when operating in master mode and is set when I2CSLA_M address matched in the slave mode.
4	I2CTOI	**I²C Master Timeout Interrupt Flag**. This bit is set to '1' if the I²C controller cannot generate a START or STOP condition or the SCL low time is greater than the timeout value specified in the I2CTO_M register. This bit must be cleared to '0' by software once set. Setting this bit to '1' by software causes an interrupt if enabled.
3	I2CSTRI	**I²C Master Clock Stretch Interrupt Flag.** This bit is set to '1' to indicate that the I²C master controller is operating with clock stretching enabled and is currently holding the SCL clock signal low. The I²C controller will release SCL after this bit has been cleared to '0'. This bit must be cleared to '0' by software once set. This bit is set by hardware only.
2	I2CRXI	**I²C Master Receive Ready Interrupt Flag.** This bit is set to '1' to indicate that a data byte has been received in I2CBUF_M. This bit must be cleared to '0' by software once set. This bit is set by hardware only.
1	I2CTXI	**I²C Master Transmit Complete Interrupt Flag.** This bit is set to '1' to indicate that an address or a data byte has been successfully shifted out and the I²C controller has received an acknowledgment from the receiver (ACK or NACK). This bit must be cleared to '0' by software once set. Setting this bit to '1' by software will cause an interrupt if enabled.
0	I2CSRI	**I²C Master START Interrupt Flag.** This bit is set to '1' when a START condition (or restart) is detected. This bit must be cleared to '0' by software once set. Setting this bit to '1' by software will cause an interrupt if enabled.

10.2.3 – I²C Master Interrupt Enable Register (I2CIE_M)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	I2CSPIE	I2CAMI2IE	I2CROIE	I2CGCIE	I2CNACKIE	-	I2CAMIE	I2CTOIE	I2CSTRIE	I2CRXIE	I2CTXIE	I2CSRIE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:12	Reserved	**Reserved**. The user should write 0 to these bits.
11	I2CSPIE	**I²C Master STOP Interrupt Enable.** Setting this bit to '1' will enable an interrupt when a STOP condition is detected (I2CSPI=1). Clearing this bit to '0' will disable the STOP detection interrupt.
10	I2CAMI2IE	**I²C Address Match2 Interrupt Enable.** This bit has no function when operating in master mode and is used in slave mode for interrupt enable for I2CSLA2_M slave address.
9	I2CROIE	**I²C Master Receiver Overrun Interrupt Enable.** Setting this bit to '1' will enable an interrupt when a receiver overrun condition is detected (I2ROI=1). Clearing this bit to '0' will disable the receiver overrun detection interrupt.
8	I2CGCIE	**I²C General Call Interrupt Enable.** This bit has no function when operating in master mode.
7	I2CNACKIE	**I²C Master NACK Interrupt Enable.** Setting this bit to '1' will enable an interrupt when a NACK is detected (I2CNACKI=1). Clearing this bit to '0' will disable the NACK detection interrupt.
6	Reserved	**Reserved.** The user should write 0 to this bit.
5	I2CAMIE	**I²C Address Match Interrupt Enable.** This bit has no function when operating in master mode and used in slave mode for interrupt enable for I2CSLA_M slave register.
4	I2CTOIE	**I²C Master Timeout Interrupt Enable.** Setting this bit to '1' will enable an interrupt when a timeout condition is detected (I2CTOI=1). Clearing this bit to '0' will disable the timeout interrupt.
3	I2CSTRIE	**I²C Master Clock Stretch Interrupt Enable.** Setting this bit to '1' will enable an interrupt when the clock stretch interrupt flag is set (I2CSTRI=1). Clearing this bit will disable the clock stretch interrupt.
2	I2CRXIE	**I²C Master Receive Ready Interrupt Enable.** Setting this bit to '1' will enable an interrupt when receive ready interrupt flag is set (I2CRXI=1). Clearing this bit to '0' will disable the receive ready interrupt.
1	I2CTXIE	**I²C Master Transmit Complete Interrupt Enable.** Setting this bit to '1' will enable an interrupt when transmit complete interrupt flag is set (I2CTXI=1). Clearing this bit to '0' disables transmit complete interrupt.
0	I2CSRIE	**I²C Master START Interrupt Enable.** Setting this bit to '1' will enable an interrupt when a START condition is detected (I2CSRI=1). Clearing this bit to '0' will disable the START detection interrupt.

10.2.4 – I²C Master Data Buffer Register (I2CBUF_M)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	D7	D6	D5	D4	D3	D2	D1	D0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*

* Unrestricted read access. This register can be written to only when I2CBUSY = 0.

BIT	NAME	DESCRIPTION
15:8	Reserved	**Reserved**. The user should write 0 to these bits.
7:0	D[7:0]	Data for I²C transfer is read from or written to this location. The I²C transmit and receive buffers are separate but both are addressed at this location.

10.2.5 – I²C Master Clock Control Register (I2CCK_M)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	I2CCKH[7:0]								I2CCKL[7:0]							
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:8	I2CCKH[7:0]	**I²C Clock High Period.** These bits define the high period of the I²C clock. This period is defined by the number of system clocks. The high time duration is calculated using the following equation: I²C High Time Period = System Clock Period x (I2CCKH[7:0] + 1) I2CCKH[7:0] must be set to a minimum value of 2 to ensure proper operation. Any value less than 2 is set to 2.
7:0	I2CCKL[7:0]	**I²C Clock Low Period.** These bits define the low period of the I²C clock. This period is defined by the number of system clocks. The low time duration is calculated using the following equation: I²C Low Time Period = System Clock Period x (I2CCKL[7:0] + 1) I2CCKL[7:0] must be set to a minimum value of 4 to ensure proper operation. Any value less than 4 is set to 4.

10.2.6 – I²C Master Timeout Register (I2CTO_M)

Bit	7	6	5	4	3	2	1	0
Name	I2CTO7	I2CTO6	I2CTO5	I2CTO4	I2CTO3	I2CTO2	I2CTO1	I2CTO0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

The I2CTO_M register determines the length of the timeout interval. The timeout interval is defined by the number of I²C bit periods (SCL high + SCL low). When cleared to 00h, the timeout function is disabled. When set to any other value, the I²C controller waits until the timeout expires and sets the I2CTOI flag. The timeout period is:

I²C Timeout = I²C Bit Rate x (I2CTO[7:0] + 1)

The timeout timer resets to 0 and starts to count after each of the following events.

- The I2CSTART bit is set.
- The I2CSTOP bit is set.
- Any time that SCL goes low.

10.2.7 – I²C Slave Address Register (I2CSLA_M and I2CSLA2_M)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	A6	A5	A4	A3	A2	A1	A0	I2CMODE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

These register have no function when operating in master mode and are used in slave mode to program the slave address.

SECTION 11 – I²C-COMPATIBLE SLAVE INTERFACE

The DS4830A provides an I²C-compatible slave controller that allows communication with a host device and supports four user-programmable slave addresses. The DS4830A I²C slave controller can support 400kHz I²C operation with a host without clock stretching. The DS4830A I²C slave interface also has a dedicated 8-byte transmit page for each slave and 8-byte receive FIFO (shared between all four slaves). The DS4830A can also have flash programming using I²C bootloading functionality provided by the slave controller. This interface can be set up to provide system interrupts after each I²C event. Figure 11-1 shows the basic operation flow of the I²C slave controller. The blocks in Figure 11-1 that are shaded are shown in more detail in Figure 11-2.

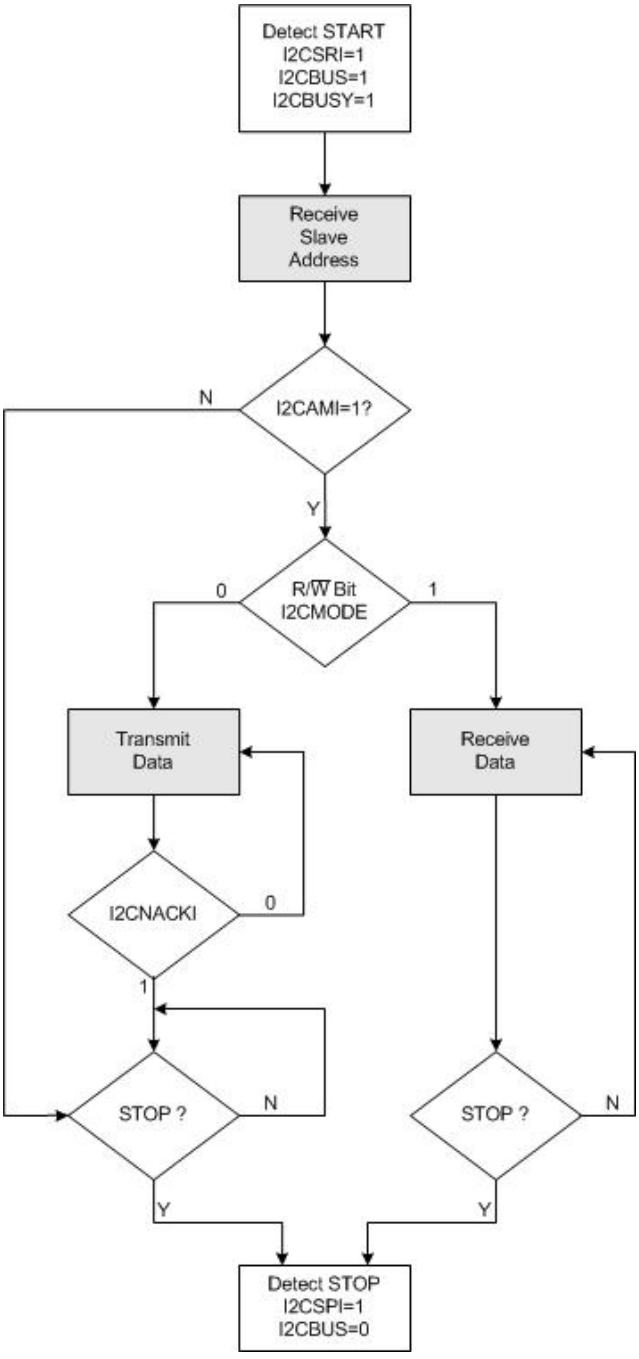


Figure 11-1: Slave I²C Flow

11.1 – Detailed Description

The I^2C slave controller has two different modes that can be used to transmit and receive data. The first option transmits and received data one byte at a time. An advanced mode uses 8-byte buffers for transmiting and receiving data, which is enabled by setting the TXPG_EN bit in the I2CTXFIE and the FIFO_EN bit in the I2CRXFIE registers. Using this advanced mode of operation, the DS4830A can support 400kHz I^2C operation without clock stretching.

11.1.1 – Default Operation

The I^2C slave controller is enabled (I2CCN_S.I2CEN=1) by default. As long as the I^2C slave controller is enabled, the DS4830A I^2C bootloader can operate. This allows bootloading of blank devices without any setup of the I^2C slave controller. Prior to the I^2C slave controller being used for normal data communication, the I^2C SFRs should be configured for necessary I^2C communication. These configurations include setting an I^2C slave address and enabling the slave controller to generate interrupts during I^2C events. This controller can also operate as an SMBUS slave.

11.1.2 – Slave Addresses

Prior to communication, an I^2C slave address may need to be selected. By default, the I^2C slave controller normally responds to two slave addresses. The I^2C bootloader uses address 34h. This bootloader address cannot be changed and should not be used as the device slave address for normal communication. The second slave address (default address 36h) is the address used for communication with the host. This slave address can be programmed by writing the desired slave address to the I2CSLA_S register. The address contained in the I2CSLA_S register is the address with the R/$\overline{W}$ bit. If an address other than 36h is desired, the I2CSLA_S register can be programmed with this new address.

The DS4830A has three more user-programmable slave addresses that can be programmable using the I2CSLA2_S, I2CSLA3_S, and I2CSLA4_S registers, respectively. By default, these slave addresses are disabled and can be individually enabled by writing '1' to the ADDR2EN, ADDR3EN, and ADDR4EN bits, which are defined in the I2CCN_S register.

The I^2C slave controller supports the General Call Address, which is 00h with the I2CSLA_S slave register. This feature can be enabled by setting the I2CCN_S.I2CGCEN bit to a 1.

11.1.3 – I^2C START Detection

The I^2C Slave Controller always monitors the I^2C bus for an I^2C START, which is a high to low transition on SDA while SCL is held high. If an I^2C START (or restart) condition is detected, the I^2C slave sets the I2CSRI bit in the I2CST_S register, which can cause an interrupt if enabled. The detection of a START brings the I^2C controller out of its idle state. Following a START, the I^2C controller begins to monitor data on the I^2C bus and the I2CBUSY bit is set to a 1. The I2CBUS bit is also set to a 1 to indicate that the I^2C bus is currently busy.

11.1.4 – I^2C STOP Detection

The I^2C Slave Controller also always monitors the I^2C bus for an I^2C STOP, which is a low to high transition on SDA while SCL is held high. If an I^2C STOP condition is detected, the I^2C slave controller sets the I2CSPI bit in the I2CST_S register, which can cause an interrupt if enabled. The I2CBUS bit is cleared to 0 following a STOP to indicate that the I^2C bus is no longer busy.

11.1.5 – Slave Address Matching

Following an I^2C START or restart, the I^2C slave controller knows that the next byte of data to be transmitted by the host should be the slave address. The I^2C slave automatically monitors for the slave address without any software interaction. The I^2C slave controller compares the first 7 bits received to the slave address programmed in the I2CSLA_S register. It also compares the first 7 bits received to the slave addresses programmed in the I2CSLA2_S, I2CSLA3_S, and I2CSLA4_S registers, if they are enabled. If the received slave address matches with one of enabled I^2C Slave addresses, the I^2C slave controller does the following steps. This is illustrated in Figure 11-2 (without RX FIFO and TX Pages) and Figure 11-4 (with RX FIFO and TX Pages).

- Transmit an ACK or NACK on the 9th clock based upon the setting of the I2CCN_S.I2CACK bit.
- Set the matched slave address I2CMODE bit with the value of the received R/$\overline{W}$ bit. This bit can be used by software to determine if the I^2C slave controller should receive or transmit data.
- Sets the I2CST_S.I2CAMI bit to indicate that a slave address match was made. The setting of this bit can generate an interrupt if enabled. Additionally, the I^2C slave controller sets following values in SLA [3:0] bits in CUR_SLA register according to the matched slave address.

Matched Slave Address	CUR_SLA.SLA[3:0]
I2CSLA_S	1
I2CSLA2_S	2
I2CSLA3_S	4
I2CSLA4_S	8

- Clears the I2CBUSY flag.

Upon completion of the slave data byte (7 bits of slave address + R/$\overline{W}$ bit + ACK/NACK), the I^2C slave controller enters one of the following three states.

- **Data Transmit**: The slave address matched and the R/$\overline{W}$ bit is '1'. The host is now expecting data from the DS4830A. The I^2C slave controller retains control of the SDA line so data can be transmitted to the host. The host can start clocking data from the slave at any time.
- **Data Receive**: The slave address is matched and the R/$\overline{W}$ bit is '0'. The host wants to write data to the I^2C slave. After sending the ACK/NACK bit, the DS4830A releases SDA and is ready to receive a byte of data.
- **Wait for START/STOP**: The received slave address did not match any enabled slave addresses. The I^2C controller enters idle state and waits for the next START or STOP condition.

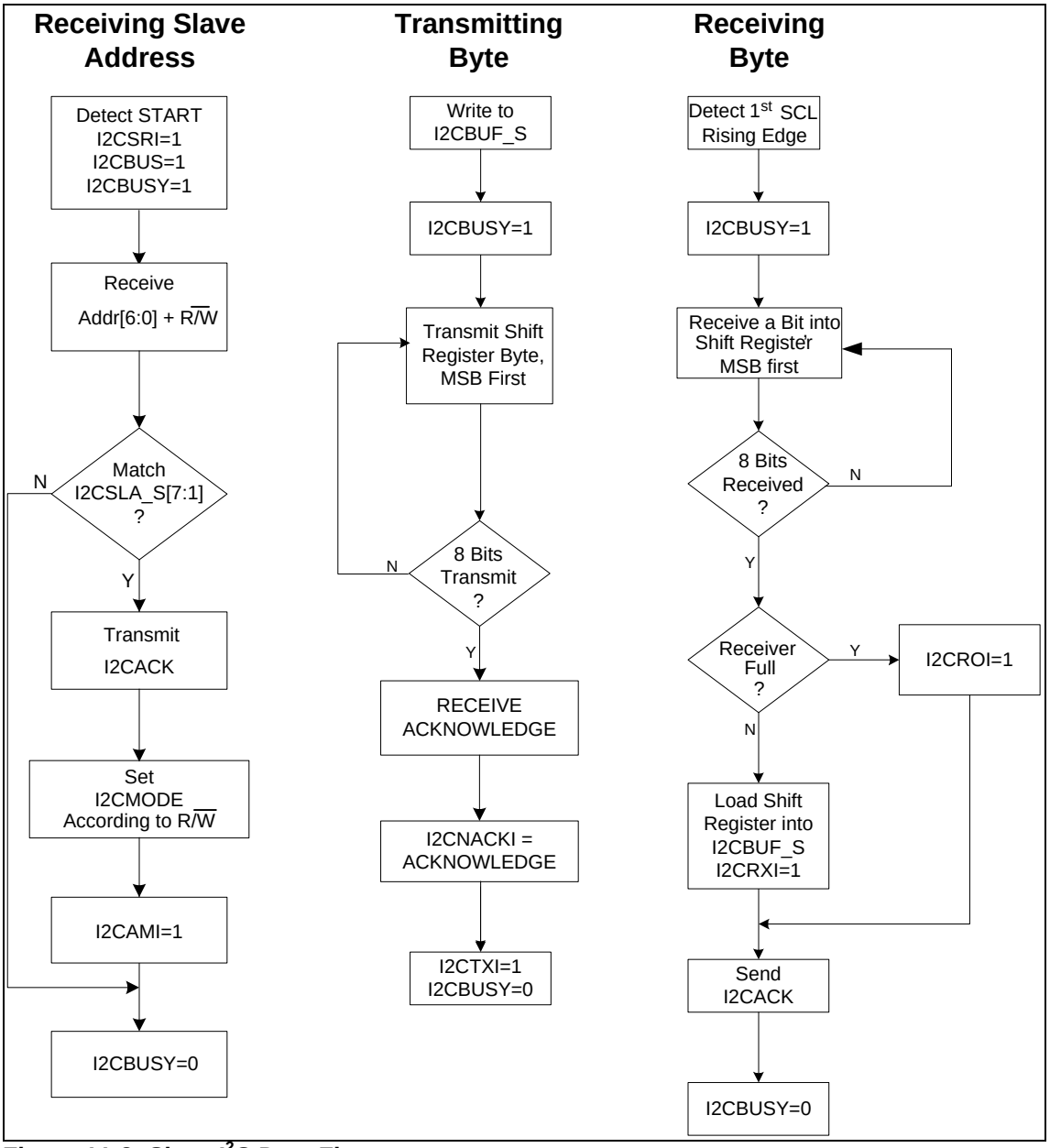


Figure 11-2: Slave I^2C Data Flow

11.1.6 – Advanced Mode Operation RX FIFO and TX Pages

The DS4830A I²C slave controller has a few features that make 400kHz I²C communication without clock stretching possible.

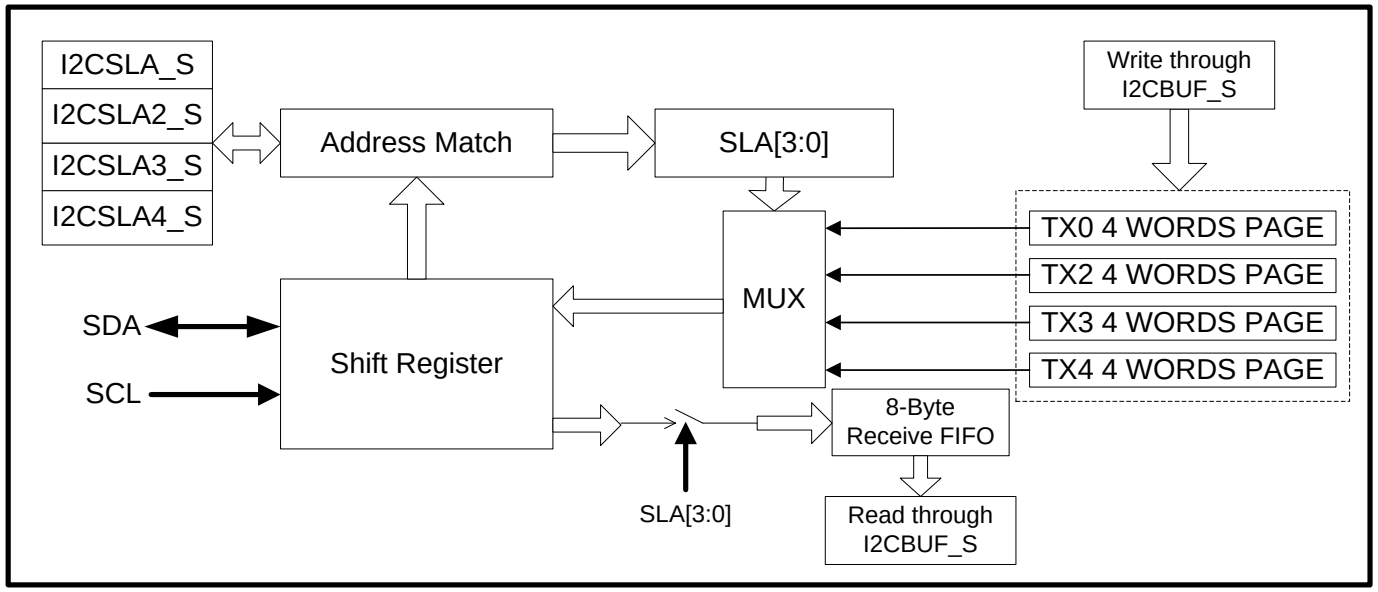


Figure 11-3: I²C Slave Block Diagram with RX FIFO and TX Pages

The I²C controller allows the user to define a memory map structure in the user SRAM for each individual slave address. This is done using the MEM_ADDR[7:0] and PAGE[2:0] bits in the MADDR, MADDR2, MADDR3, and MADDR4 registers. These register bits 10:0 are used to define start address (SRAM Address) of the memory map structure and bit 12 is used to define memory rollover boundary between 128 and 256. The I²C controller maintains the memory address of the individual slave address in the read memory address pointer RPNTR register. Each slave address has dedicated RPNTR, which is selected based on the SLA[3:0] bits. The read address (maintained by RPNTR) is automatically incremented by 1 word after every write to the I2CBUF_S. The I²C controller handles 128 or 256 boundary rollover internally on the read memory address.

11.1.6.1 – RX FIFO

The DS4830A I²C controller has an 8-byte receive FIFO. This FIFO is shared among the enabled slave addresses. The receive FIFO is controlled using the I2CRXFIE (I²C Receive FIFO Interrupt Enable) and I2CRXST (I²C Receive FIFO Status Flags) registers and is read from the I2CBUF_S register. See the individual bit description in I²C Slave Controller Register Description section. This FIFO is shown in Figure 11-3.

11.1.6.2 – Transmit Pages

The I²C controller has four Transmit (TX) pages, each dedicated to a specific slave address. Each of the TX pages holds 4 16-bit words. When transmitting data, the controller automatically selects one of the TX pages based upon the SLA[3:0] bits in the CUR_SLA (Current Slave Address) which is set during a successful slave address match event. The TX Pages are filled by first setting the SLA[3:0] bits, then writing data to the I2CBUF_S register. I²C transmission using the TX Pages is controlled using the I2CTXFIE (I²C Transmit Interrupt Enable) and I2XTXFST (I²C Transmit Page Status Flags) registers. See the individual bit description in I²C Slave Controller Register Description section. The TX pages are shown in Figure 11-3.

11.1.6.3 Advanced Mode Memory Address Detection

The I²C Slave Controller provides an option to automaticcaly detect the memory address being accessed by the host. The MADDR_EN bits in the CUR_SLA register enable the memory address to be automatically captured by the I²C controller. Following an address match with I2CMODE = 0 (Write), the I²C slave controller knows that the next byte of data to be received is the memory address of the memory map and copies the received byte into the MEM_ADDR[7:0] bits in the MPNTR (Memory address pointer) register with PAGE[2:0] from active slave address. When the memory address is captured, the MADI bit in the I2CST2_S register will be set, which can generate an interrupt if enabled. The MPNTR shows the current memory address of the active slave address. To enable memory address dection, the proper MADDR_EN bit must be set and the RX FIFO must be enabled.

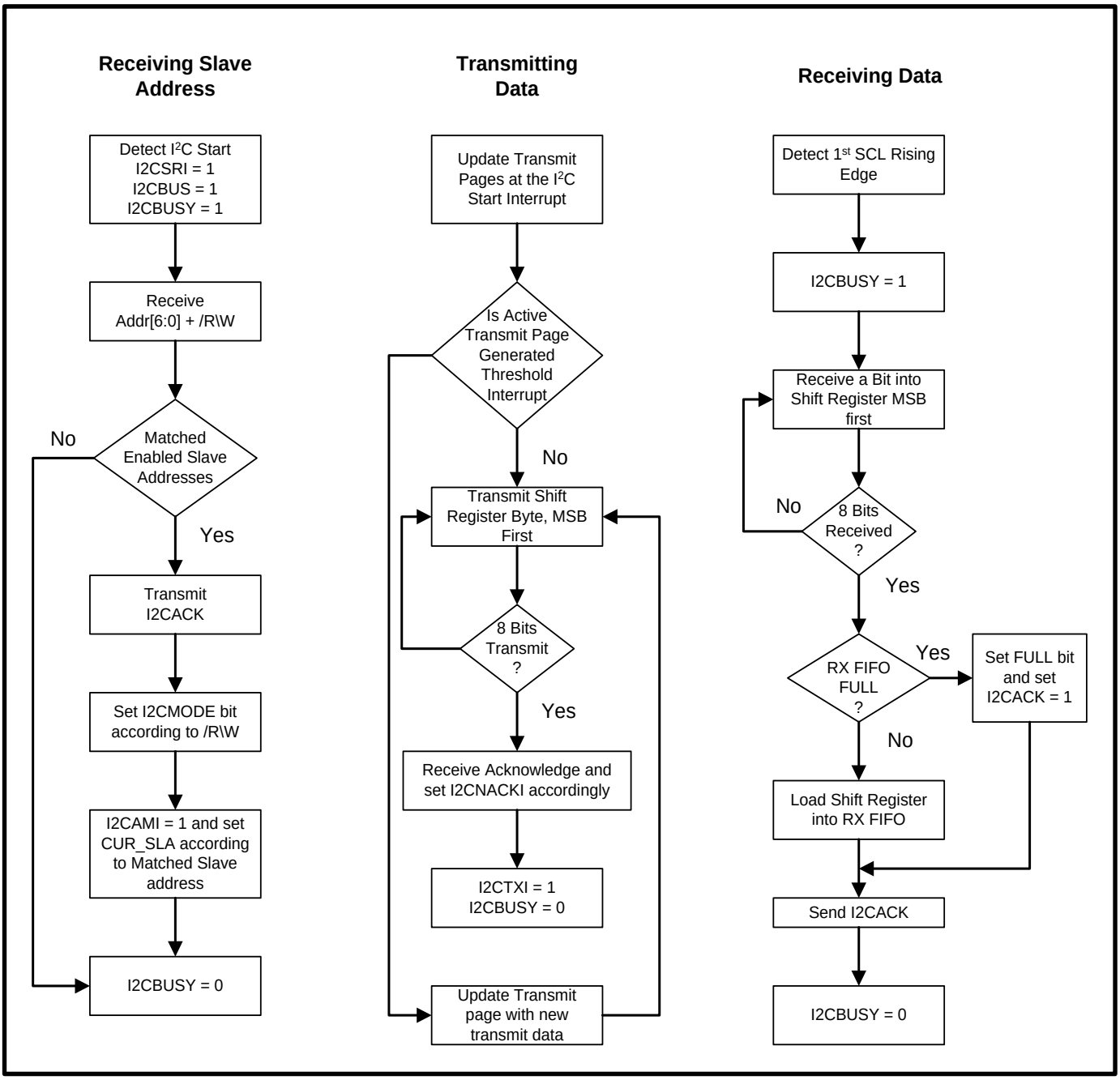


Figure 11-4: Slave I²C Data Flow Using 8-Byte Transmit Page and 8-Byte Receive FIFO

11.1.7 – Transmitting Data

The DS4830A I²C Slave Controller enters into data transmission mode after receiving a matching slave address with the R/$\overline{W}$ bit set to 1.

11.1.7.1 – Normal Mode Data Transmission

The steps of data transmission are shown in Figure 11-2. Data transmission is started by software loading data into the I2CBUF_S register. Loading I2CBUF_S causes the I2CBUSY bit in I2CST_S to be set. Once I2CBUSY bit is set, a write to I2CBUF_S is ignored. The first bit of data (most significant bit) is shifted to SDA when SCL is low. Each of the next seven bits is then shifted following high to low transitions of SCL.

Following the 8th bit data (least significant bit) being shifted to SDA, the SDA line is released by the slave controller. This allows the host to signal an ACK or NACK during the 9th clock cycle. The I²C slave controller samples the acknowledge bit following the 9th SCL rising edge. After the acknowledge bit is sampled, the I²C slave controller performs the following tasks:

- Sets the I2CST_S.I2CTXI flag to indicate that the I^2C slave controller has transmitted a byte. This can generate an interrupt if enabled.
- Sets or clears the I2CST_S.I2CNACKI flag to reflect the received acknowledge bit. The setting of I2CNACKI can generate an interrupt if enabled.
- Clears the I2CST_S.I2CBUSY flag to indicate that the I^2C slave controller is not actively participating in the transfer of data.

The detection of an ACK by the I^2C slave controller indicates that the host wants to receive another byte of data. The I^2C slave controller maintains control of SDA following the ACK. The next byte to transmit needs to be loaded into I2CBUF_S prior to the host starting to clock this next byte.

The detection of a NACK indicates that the host does not want to receive any additional data. The I^2C slave controller releases control of SDA following the reception of NACK bit. After the NACK, the slave controller enters idle state and monitors the I^2C bus for a START or STOP condition.

11.1.7.2 – **Advanced Mode Data Transmission**

To achieve 400kHz I^2C without clock stretch, the DS4830A I^2C Controller has 4-word TX Pages for each slave address. The TXPG_EN bit in the I2CTXFIE register enables the TX PAGEs of the all enabled slave addresses. The user should pre-fill these 4-word pages to ensure data is available to transmit immedialty following a slave address match. When data is being transmit, the I^2C controller automatically selects one of the four TX Pages depending upon which SLA [3:0] bits are set during the slave address match event.

The individual TX page should be written in the word mode using the I2CBUF_S. See below pseudo code to write the TX page of I2CSLA2_S address

```
MOVE  DP[0],    #01Ch          //DP[0] in word mode
MOVE  M2[21],  #00F2h          //Select TX PAGE2  in CUR_SLA

MOVE  RPNTR, #0000h            //Initialize RPNTR to current read address.  When written to 0000h,
                               //RPNTR will populate with the correct SRAM memory location for
                               //read data

                               //Copy word 1
MOVE  DP[0],   RPNTR           //Copy current memory address to the data pointer
MOVE  M2[0],   @DP[0]          //Copy data from @DP[0] to I2CBUF_S register (M2[0])
                               //I2CBUF_S will load data into TX PAGE
                               // RPNTR = RPNTR + 1 automatically when data is loaded
                               //into I2CBUF_S.  Rollover handled internally.

                               //Copy word 2
MOVE  DP[0],   RPNTR           //Copy current memory address in the data pointer
MOVE  M2[0],   @DP[0]          //Copy data from @DP[0] to TX PAGE via I2CBUF_S register

                               //Copy word 3
MOVE  DP[0],   RPNTR           //Copy current memory address in the data pointer
MOVE  M2[0],   @DP[0]          //Copy data from @DP[0] to TX PAGE via I2CBUF_S register

                               //Copy word 4
MOVE  DP[0],   RPNTR           //Copy current memory address in the data pointer
MOVE  M2[0],   @DP[0]          // Copy data from @DP[0] to TX PAGE via I2CBUF_S register
```

When TX page is enabled, the SLA[3:0] bits in the CUR_SLA register selects one of the TX pages as shown in Figure 11-3. The I^2C controller reads data from the selected TX page and writes to the shift register. When the I^2C controller is transmitting data, the threshold interrupt flag (THSH) in the I2CTXST register will be set when there are 4 bytes are remaining. This can generate an interrupt, if enabled.

11.1.8 – Receiving Data

The I²C Slave Controller enters data reception mode after receiving a matching slave address with the R/$\overline{W}$ bit set to 0. The steps of data reception are shown in Figure 11-2 and Figure 11-4. The reception process begins when the I²C slave controller detects the first rising edge of SCL. This rising edge sets I2CBUSY bit to '1' and clocks the first bit (MSB) of data from SDA into the data shift register.

11.1.8.1 – Receiving Data in Normal Mode

When receiving data, the I²C slave controller uses a double buffer consisting of the I2CBUF_S register and the shift register. This allows the I²C module to continue receiving data while the previous data byte is being processed. After a byte (8 bits) of data is received, the I²C slave controller attempts to copy the received data from the shift register to I2CBUF_S and two possible events can occur during this attempt.

1. If I2CBUF_S is empty, the I²C slave controller copies the data from the shift register into I2CBUF_S. The I2CRXI flag is set to indicate a received byte is ready for reading. The setting of I2CRXI can generate an interrupt if enabled. Software can now read data from the I2CBUS_S.
2. If I2CBUF_S is full, the data in the shift register cannot be copied into I2CBUF_S. This causes a receive overrun condition. The receive overrun flag, I2CROI is set which can generate an interrupt if enabled. I2CBUF_S can be full if it is not read by software following the reception of a previous byte.

When the receive overrun occurs (I2CROI = 1), any new incoming data is not shifted into the I²C slave controller. The controller responds to any bytes received with a NACK regardless of the setting of the I2CACK bit. The receive overrun condition and the I2CROI flag can only be cleared by software reading received first byte from I2CBUF_S. When the receive overrun condition is cleared, the I2C slave controller copies the second byte that is received into I2CBUF, and again sets I2CRXI to indicate a byte of data is received. The I²C slave controller resumes its normal operation in the next SCL clock cycle after I2CROI is cleared. To avoid losing any data, I2CROI must to be cleared prior to the first SCL clock rising edge of the next byte.

After the 9th bit of any byte has been received, the I2CBUSY bit is cleared to indicate that the controller is no longer participating in a data transaction. The value in I2CACK is transmitted to the host on the 9th SCL clock cycle, assuming the I²C slave controller is not operating in receive overrun.

11.1.8.2 – Receiving Data in Advanced Mode

As shown in Figure 11-4, when receive FIFO is enabled, the incoming data is copied into the FIFO. The receive FIFO will set flags in the I2CRXFST register when the FIFO is empty, half full with 4 bytes, or full with 8 bytes of received data. Interrupts can be generated for these events if the appropriate bits are set in the I2CRXFIE register. The receive FIFO is read one word at a time by reading the I2CBUF_S register.

11.1.9 – Clock Stretching

If slave device cannot receive or transmit another complete byte of data, it may hold SCL low, forcing the master to wait. Data transfer continues when the slave is ready for next byte of data after releasing SCL.

The I²C slave controller is capable of holding SCL low at the completion of each byte being transferred. If the I²C Clock Stretch Enable bit (I2CSTREN) is set to a 1, the I²C controller holds SCL low after the 8th or 9th clock pulse as configured in the I²C Clock Stretch Select bit (I2CSTRS). If I2CSTRS=0, the I²C controller holds SCL low after the falling edge of the 9th clock pulse. If I2CSTRS=1, the I²C controller holds SCL low after the falling edge of the 8th clock pulse. When the I²C controller is holding SCL low, the I²C Clock Stretch Interrupt bit (I2CSTRI) is set. The I²C slave controller holds SCL low until I2CSTRI is cleared to '0' by software. Figure 11-5 shows the I²C slave controller clock stretching after receiving the 9th clock of a byte.

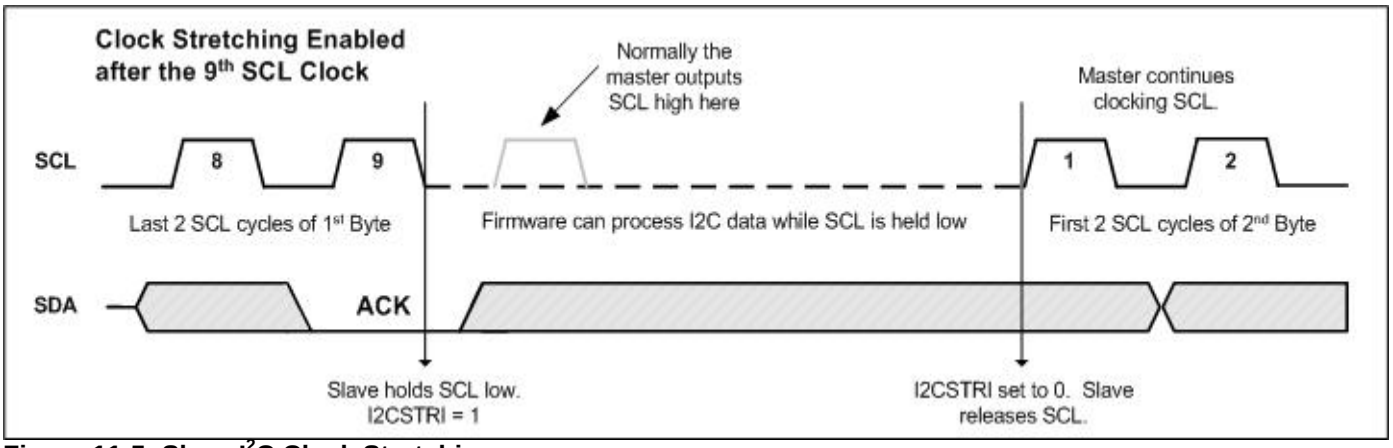


Figure 11-5: Slave I²C Clock Stretching

Normally when the I²C slave controller is receiving data, the value of I2CACK is sent after the falling edge of the 8th clock. However, if clock stretching is enabled after the 8th clock, the I²C slave controller continues to output the I2CACK bit until clock stretching is released by software. This allows software time to inspect data that is received before responding with an appropriate acknowledge bit.

The applications should use clock stretching if the I²C slave interrupts are not assigned the highest priority. Generally the application is set to respond only to interrupts from the I²C slave controller, thus not having to continuously poll the slave I²C controller. After each byte transfer is complete, the I²C slave controller needs to either read the received byte from I2CBUF_S or write the next byte to transmit to I2CBUF_S. Without using clock stretching, the host can begin clocking the next byte before the I²C slave controller is prepared. A few conditions that may require clock stretching to be enabled are listed below when used without RX FIFO and TX Pages.

- When a slave address match is made and the R/$\overline{W}$ bit is set, the I²C slave controller is expected to transmit a byte of data to the host. This byte of data needs to be written to I2CBUF_S. If clock stretching is not used, software may not be able to write the correct data into I2CBUF_S prior to the first clock of the data byte.
- Following the transmission of data to the host, another byte may be requested by the host sending an ACK bit. The I²C slave controller has to write next data to the I2CBUF_S prior to the first clock of the second byte which sometimes may not be possible.
- After a byte is received by the I²C slave controller it may be necessary to stretch the clock. This allows software to read the byte from I2CBUF_S and perform data processing.

11.1.10 – SMBus Timeout

The I^2C slave controller can also be used for SMBus or PMBus™ communication. To maintain SMBus compatibility, a 30ms timer is implemented by the I^2C slave controller. The purpose of this timer is to issue a timeout interrupt when SCL is low for greater than 30ms. The timer only starts when **none** of the following conditions are true:

1. The I^2C slave controller is in the idle state and there are no communications on the I^2C bus. The timer should not generate interrupts regardless of how long SCL is low.
2. The SMBUS mode bit is not set. This ensures the SMBUS timeout functionality does not interfere with normal I^2C functionality.
3. SCL is high. The timer is inactive whenever SCL is high. The timer is reset when it is inactive.
4. The I^2C slave controller is disabled or used as a master I^2C controller. The timer is not needed in this case.

The following description explains when the SMBus timer starts, assuming that all other START conditions are met. When the I^2C slave controller is idle and it receives a START, it exits the idle state and the timer becomes active (starts counting) any time SCL goes low. If following the START, the master addresses a different slave on the bus, the I^2C slave controller is returned to the idle state and the timer is reset and becomes inactive. In short, as soon as SCL goes low following a START, the SMBus timer becomes active until the I^2C slave controller re-enters into idle state.

When a timeout occurs, the timeout bit (I2CTOI) is set, which can generate an interrupt if enabled. If a timeout occurs, it may be necessary to reset the I^2C slave controller. See the *Resetting the I^2C Slave Controller* section for more details. SMBus mode selection is controlled by the SMB_MOD bit in I2CCN_S register. When the Slave SMBus Mode Operation bit (SMB_MOD) is set to 1, the SMBUS timeout functionality is enabled.

11.1.11 – Resetting the I^2C Slave Controller

The I^2C Slave Controller can be reset by disabling the I^2C Slave controller by writing '0' at I2CEN bit in the I2CCN_S register. A reset forces the I^2C slave controller to release both SDA and SCL if they are being held low by the I^2C slave controller. The reset may reset few or all bits of I2CCN, I2CST and I2CBUF_S registers and reset the internal state machine of the I^2C slave controller. Following a reset, the I^2C slave controller must be re-initialized.

Note: When the I^2C slave interface is disabled, the I^2C bootloader is not available.

PMBus is a trademark of SMIF, Inc.

11.2 – I²C Slave Controller Register Description

Following are the registers that are used to control the I²C Slave Interface.

11.2.1 – I²C Slave Control Register (I2CCN_S)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	ADDR4EN	ADDR3EN	ADDR2EN	SMB_MOD	I2CSTREN	I2CGCEN	-	-	I2CACK	I2CSTRS	-	I2CMODE	-	I2CEN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Access	r	r	rw	rw	rw	rw	rw*	rw*	r	r	rw*	rw*	r	r	r	rw*

* Unrestricted Read. Unrestricted write access when I2CBUSY=0. Writes to I2CEN are disabled when I2CBUSY=1.

BIT	NAME	DESCRIPTION
15:14	Reserved	**Reserved.** The user should not write to these bits.
13	ADDR4EN	**I²C Slave Address 4 Enable:** Setting this bit to '1', enables slave address I2CSLA4_S and the I²C controller uses this slave address during the address match event. When this bit is set to '0', disables slave address I2CSLA4_S.
12	ADDR3EN	**I²C Slave Address 3 Enable:** Setting this bit to '1', enables slave address I2CSLA3_S and the I²C controller uses this slave address during the address match event. When this bit is set to '0', disables slave address I2CSLA3_S.
11	ADDR2EN	**I²C Slave Address 2 Enable:** Setting this bit to '1', enables slave address I2CSLA2_S and the I²C controller uses this slave address during the address match event. When this bit is set to '0', disables slave address I2CSLA2_S.
10	SMB_MOD	**Slave SMBUS Mode Operation.** When this bit is set to a '1', SMBus timeout functionality is enabled for the I2C slave interface. When this bit is cleared to '0', the SMBus timeout functionality is disabled. See the SMBUS Timeout section for more details.
9	I2CSTREN	**I²C Slave Clock Stretch Enable.** Setting this bit to '1' stretches the clock (holds SCL low) at the end of the clock cycle specified in I2CSTRS. Clearing this bit disables clock stretching.
8	I2CGCEN	**I²C Slave General Call Enable.** Setting this bit to '1' enables the I²C to respond to a general call address (address = 0000 0000). Clearing this bit to '0' disables response to general call address.
7:6	Reserved	**Reserved.** The user should not write to these bits.
5	I2CACK	**I²C Slave Data Acknowledge Bit.** This bit selects the acknowledge bit returned by the I²C controller while acting as a receiver. Setting this bit to '1' generates a NACK (leaving SDA high). Clearing the I2CACK bit to '0' generates an ACK (pulling SDA LOW) during the acknowledgement cycle. This bit retains its value unless changed by software or hardware.
4	I2CSTRS	**I²C Slave Clock Stretch Select.** Setting this bit to '1' enables clock stretching after the falling edge of the 8th clock cycle. Clearing this bit to '0' enables clock stretching after the falling edge of the 9th clock cycle. This bit has no effect when clock stretching is disabled (I2CSTREN=0).
3	Reserved	**Reserved.** The user should not write to this bit.
2	I2CMODE	**I²C Transfer Mode Select.** This bit reflects the actual R/$\overline{W}$ bit value in current I²C transfer and is set by hardware. The same bit is set by hardware for corresponding slave address register following a successful slave address match.
1	Reserved	**Reserved.** The user should not write to this bit.
0	I2CEN	**I²C Slave Enable.** This bit enables the I²C Slave function. When set to '1', I²C Slave communication is enabled. When cleared to '0', the I²C function is disabled.

11.2.2 – I²C Slave Status Register (I2CST_S)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	I2CBUS	I2CBUSY	-	-	-	I2CSCL	I2CROI	I2CGCI	I2CNACKI	-	I2CAMI	I2CTOI	I2CSTRI	I2CRXI	I2CTXI	I2CSRI
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r*	r*	r	r	r	r*	rw	rw	rw*	r	rw	rw	rw*	rw*	rw	rw

* Set by hardware only.

BIT	NAME	DESCRIPTION
15	I2CBUS	**I²C Slave Bus Busy.** This bit is set to '1' when a START/repeated START condition is detected and cleared to '0' when the STOP condition is detected. This bit is reset to '0' on all forms of reset or when I2CEN=0. This bit is controlled by hardware and is read only.
14	I2CBUSY	**I²C Slave Busy.** This bit is used to indicate the current status of the I²C module. The I2CBUSY is set to '1' when the I²C controller is actively participating in a transaction. This bit is controlled by hardware and is read only.
13:11	Reserved	**Reserved.** The user should not write to these bits.
10	I2CSCL	**I²C Slave SCL Status.** This bit reflects the logic state of SCL signal. This bit is set to '1' when SCL is at a logic high (1), and cleared to '0' when SCL is at a logic low (0). This bit is controlled by hardware and is read only.
9	I2CROI	**I²C Slave Receiver Overrun Flag.** This bit indicates a receive overrun when set to '1'. This bit is set to '1' if the receiver has received two bytes since the last CPU read of I2CBUF_S. This bit can only be cleared to '0' by software reading the I2CBUF_S. Setting this bit to 1 by software causes an interrupt if enabled.
8	I2CGCI	**I²C Slave General Call Interrupt Flag.** This bit is set to '1' when the general call is enabled (I2CGCEN=1) and the general call address (00h) is received. This bit must be cleared to '0' by software once set. Setting this bit to '1' by software causes an interrupt if enabled.
7	I2CNACKI	**I²C Slave NACK Interrupt Flag.** This bit is set by hardware to either a '1' if a NACK was received from the host or a '0' if an ACK was received from the host. The setting of this bit to a '1' causes an interrupt if enabled. This bit can be cleared to '0' by software once set.
6	Reserved	**Reserved.** The user should not write to this bit.
5	I2CAMI	**I²C Slave Address Match Interrupt Flag.** This bit is set to '1' when the I²C controller receives an address that matches the contents of the slave address register (I2CSLA_S). This bit must be cleared to '0' by software once set. Setting this bit to '1' by software causes an interrupt if enabled.
4	I2CTOI	**I²C Slave Timeout Interrupt Flag.** This bit is set to '1' if SMBUS timeout is enabled and SCL is low longer than 30ms. This bit must be cleared to '0' by software once set. Setting this to '1' causes an interrupt if enabled.
3	I2CSTRI	**I²C Slave Clock Stretch Interrupt Flag**. This bit indicates that the I²C slave controller is operating with clock stretching enabled and is currently holding the SCL clock signal low. The I²C controller releases SCL after this bit has been cleared to '0'. This bit must be cleared to '0' by software once set. This bit is set by hardware only.
2	I2CRXI	**I²C Slave Receive Ready Interrupt Flag.** This bit indicates that a data byte has been received in the I2C buffer. This bit must be cleared by software once set. This bit is set by hardware only.
1	I2CTXI	**I²C Slave Transmit Complete Interrupt Flag.** This bit indicates that an address or a data byte has been successfully shifted out and the I²C controller has received an acknowledgment from the receiver (NACK or ACK). This bit must be cleared by software once set. Setting this bit to '1' by software causes an interrupt if enabled.
0	I2CSRI	**I²C Slave START Interrupt Flag.** This bit is set to '1' when a START condition (or restart) is detected. This bit must be cleared to '0' by software once set. Setting this bit to '1' by software causes an interrupt if enabled.

11.2.3 – I²C Slave Interrupt Enable Register (I2CIE_S)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	I2CROIE	I2CGCIE	I2CNACKIE	-	I2CAMIE	I2CTOIE	I2CSTRIE	I2CRXIE	I2CTXIE	I2CSRIE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	rw	rw	rw	r	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:10	Reserved	**Reserved.** The user should not write to these bits.
9	I2CROIE	**I²C Slave Receiver Overrun Interrupt Enable.** Setting this bit to '1' causes an interrupt to the CPU when a receiver overrun condition is detected (I2ROI=1). Clearing this bit to '0' disables the receiver overrun detection interrupt.
8	I2CGCIE	**I²C Slave General Call Interrupt Enable.** Setting this bit to '1' causes an interrupt to the CPU when a general call is detected (I2CGCI=1). Clearing this bit to '0' disables the general call interrupt.
7	I2CNACKIE	**I²C Slave NACK Interrupt Enable.** Setting this bit to '1' causes an interrupt to the CPU when a NACK is detected (I2CNACKI=1). Clearing this bit to '0' disables the NACK detection interrupt.
6	Reserved	**Reserved.** The user should not write to this bit.
5	I2CAMIE	**I²C Slave Address Match Interrupt Enable.** Setting this bit to '1' causes an interrupt to the CPU when the I²C controller detects an address that matches the I2CSLA_S value (I2CAMI=1). Clearing this bit to '0' disables the address match interrupt.
4	I2CTOIE	**I²C Slave Timeout Interrupt Enable.** Setting this bit to '1' causes an interrupt to the CPU when an SMBUS timeout condition is detected (I2CTOI=1). Clearing this bit to '0' disables the timeout interrupt.
3	I2CSTRIE	**I²C Slave Clock Stretch Interrupt Enable.** Setting this bit to '1' generates an interrupt to the CPU when the clock stretch interrupt flag is set (I2CSTRI=1). Clearing this bit disables the clock stretch interrupt.
2	I2CRXIE	**I²C Slave Receive Ready Interrupt Enable.** Setting this bit to '1' causes an interrupt to the CPU when receive ready interrupt flag is set (I2CRXI=1). Clearing this bit to '0' disables the receive ready interrupt.
1	I2CTXIE	**I²C Slave Transmit Complete Interrupt Enable.** Setting this bit to '1' causes an interrupt to the CPU when transmit complete interrupt flag is set (I2CTXI=1). Clearing this bit to '0' disables transmit complete interrupt.
0	I2CSRIE	**I²C Slave START Interrupt Enable.** Setting this bit to '1' causes an interrupt to the CPU when a START condition is detected (I2CSRI=1). Clearing this bit to '0' disables the START detection interrupt.

11.2.4 – I²C Slave Status2 Register (I2CST2_S)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	-	-	I2CSPI	SADI	MADI	-	I2CXFRON	-
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	r	rw	r

BIT	NAME	DESCRIPTION
15:6	Reserved	**Reserved.** The user should not write to these bits.
5	I2CSPI	**I²C Slave STOP Interrupt Flag.** This bit is set to '1' when a STOP condition is detected. This bit must be cleared to '0' by software once set. Setting this bit to '1' by software causes an interrupt if enabled.
4	SADI	**I²C START and Start of Address Cycle Flag.** This bit is set to '1' if the I²C controller has detected an I2C START and 2 cycles of SCL clock. Setting this to '1' causes an interrupt if enabled. This bit must be cleared to '0' by software once set.
3	MADI	**Memory Address Detected Interrupt Flag**. This bit indicates that the I²C slave controller has detected a memory address and copied address into bit [7:0] of MPNTR register. This bit must be cleared to '0' by software once set. Setting this bit to '1' by software causes an interrupt if enabled.
2	Reserved	**Reserved.** The user should not write to this bit.
1	I2CXFRON	**I²C Slave Transmit Complete Interrupt Flag.** This bit indicates that an address or a data byte has been successfully shifted out and the I²C controller has received an acknowledgment from the receiver (NACK or ACK). This bit must be cleared by software once set.
0	Reserved	**Reserved.** The user should not write to this bit.

11.2.5 – I²C Slave Interrupt Enable2 Register (I2CIE2_S)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	-	-	I2CSPIE	SADIE	MADIE	-	-	-
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	r	rw	r

BIT	NAME	DESCRIPTION
15:6	Reserved	**Reserved.** The user should not write to these bits.
5	I2CSPIE	**I²C Slave STOP Interrupt Enable.** Setting this bit to '1' causes an interrupt to the CPU when a STOP condition is detected (I2CSPI=1). Clearing this bit to '0' disables the STOP detection interrupt.
4	SADIE	**I²C Slave After Start Interrupt Enable.** Setting this bit to '1' causes an interrupt to the CPU after I²C start + two master clocks.
3	MADIE	**I²C Slave Memory Address Interrupt Enable.** Setting this bit to '1' causes an interrupt to the CPU when a memory address is detected on the I²C bus. The memory address cycle is detected by I²C controller after address match with write. The I²C controller looks for data after address match with write and copies into the MPNTR register. Clearing this bit to '0' disables the memory address detection interrupt.
2:0	Reserved	**Reserved.** The user should not write to these bits.

11.2.6 – I²C Slave Address Registers (I2CSLA_S, I2CSLA2_S, I2CSLA3_S and I2CSLA4_S)

I2CSLA_S

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	A6	A5	A4	A3	A2	A1	A0	I2CMode
Reset*	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0
Access	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

* Default value of I2CSLA_S is 36h.

I2CSLA2_S, I2CSLA3_S and I2CSLA4_S

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	A6	A5	A4	A3	A2	A1	A0	I2CMode
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:8	Reserved	**Reserved.** The user should not write to these bits.
7:1	A[6:0]	**I²C Slave Address.** These address bits contain the address of the I²C slave interface. When a match to this address is detected, the I²C controller automatically acknowledges the host with the I2CACK bit value and the I2CAMI flag is set to '1'. An interrupt is generated if enabled. The I2CSLA_S is enabled by default. Other slave address registers participate in the address match event only when the corresponding slave address enable bit in the I2CCN_S register is set to '1'.
0	I2CMode	**I²C Transfer Mode Select.** This bit reflects the actual R/$\overline{W}$ bit value in current value in I²C transfer and set by hardware.

11.2.7 – I²C Slave Data Buffer Register (I2CBUF_S)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:0	D[15:0]	Data for I²C transfer is read from or written to this register. The I²C transmit and receive buffers are different internal registers, however both are addressed at this register. The receive FIFO and TX pages are read and written using the I2CBUF_S register.

11.2.8 – Memory Map Address Register (MADDR)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	ROLLOVR	-	PAGE[2:0]			MEM_ADDR[7:0]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:13	Reserved	**Reserved.** The user should not write to these bits.
12	ROLLOVR	**Rollover Config:** Setting this bit to '1', enables boundary rollover at the memory address 256 and Setting this bit to '0', enables boundary rollover at the memory address 128 for the s;ave address defined by the I2CSLA_S register.
11	Reserved	**Reserved.** The user should not write to this bit.
10:8	PAGE	**PAGE:** These bits define the page of memory map structure for I2CSLA_S slave address.
7:0	MEM_ADDR	**Memory Address.** These bits define the start address of memory map structure for I2CSLA_S slave address.

11.2.9 – Memory Map Address Register (MADDR2)

BIT	NAME	DESCRIPTION
15:13	Reserved	**Reserved.** The user should not write to these bits.
12	ROLLOVR	**Rollover Config:** Setting this bit to '1', enables boundary rollover at the memory address 256 and Setting this bit to '0', enables boundary rollover at the memory address 128 for the s;ave address defined by the I2CSLA2_S register.
11	Reserved	**Reserved.** The user should not write to this bit.
10:8	PAGE	**PAGE:** These bits define the page of memory map structure for I2CSLA2_S slave address.
7:0	MEM_ADDR	**Memory Address.** These bits define the start address of memory map structure for I2CSLA2_S slave address.

11.2.10 – Memory Map Address Register (MADDR3)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	ROLLOVR	-	PAGE[2:0]			MEM_ADDR[7:0]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:13	Reserved	**Reserved.** The user should not write to these bits.
12	ROLLOVR	**Rollover Config:** Setting this bit to '1', enables boundary rollover at the memory address 256 and Setting this bit to '0', enables boundary rollover at the memory address 128 for the s;ave address defined by the I2CSLA3_S register.
11	Reserved	**Reserved.** The user should not write to this bit.
10:8	PAGE	**PAGE:** These bits define the page of memory map structure for I2CSLA3_S slave address.
7:0	MEM_ADDR	**Memory Address.** These bits define the start address of memory map structure for I2CSLA3_S slave address.

11.2.11 – Memory Map Address Register (MADDR4)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	ROLLOVR	-	PAGE[2:0]			MEM_ADDR[7:0]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:13	Reserved	**Reserved.** The user should not write to these bits.
12	ROLLOVR	**Rollover Config:** Setting this bit to '1', enables boundary rollover at the memory address 256 and Setting this bit to '0', enables boundary rollover at the memory address 128 for the s;ave address defined by the I2CSLA4_S register.
11	Reserved	**Reserved.** The user should not write to this bit.
10:8	PAGE	**PAGE:** These bits define the page of memory map structure for I2CSLA_S slave address.
7:0	MEM_ADDR	**Memory Address.** These bits define the start address of memory map structure for I2CSLA4_S slave address.

11.2.12 – Current Slave Address Register (CUR_SLA)

Bit	7	6	5	4	3	2	1	0
Name	MADDR_EN14	MADDR_EN3	MADDR_EN2	MADDR_EN1	SLA[3:0]			
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:8	Reserved	**Reserved.** The user should not write to these bits.
7	MADDR_EN4	**Memory Address Detection Enable 4:** Setting this bit to '1', enables the memory address detection as described in section 11.1.6.3 for the slave address defined by the I2CSLA4_S register.
6	MADDR_EN3	**Memory Address Detection Enable 3:** Setting this bit to '1', enables the memory address detection as described in section 11.1.6.3 for the slave address defined by the I2CSLA3_S register.
5	MADDR_EN2	**Memory Address Detection Enable 2:** Setting this bit to '1', enables the memory address detection as described in section 11.1.6.3 for the slave address defined by the I2CSLA2_S register.
4	MADDR_EN1	**Memory Address Det1ction Enable 1:** Setting this bit to '1' enables the memory address detection as described in section 11.1.6.3 for the slave address defined by the for I2CSLA_S register.
3:0	SLA[3:0]	**Slave Address Select.** These bits indicate the current active slave address. These bits are updated after the slave address match event by the I^2C controller. Using these bits, the TX Pages are selected by the I^2C controller during the I^2C transmits events. The I^2C controller allows writing to these bits. However, user should write to these bits before the address match event which allows I^2C controller to select intended TX page.

11.2.13 – Memory Address Pointer Register (MPNTR)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	PAGE[2:0]			MEM_PNTR[7:0]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:11	Reserved	**Reserved.** The user should not write to this bit.
10:8	PAGE	**PAGE:** These bits define the page of memory map structure for current active slave address.
7:0	MEM_ADDR	**Memory Address.** These bits store current address of memory map structure of the current active slave address. The I^2C controller automatically increments and performs boundary rollover for the active slave address according to ROLLOVER bit (ROLLOVR) defined in the corresponding MADDR register.

11.2.14 – Read Memory Address Pointer Register (RPNTR)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	PAGE[2:0]			MEM_PNTR[7:0]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:11	Reserved	**Reserved.** The user should not write to this bit.
10:8	PAGE	**PAGE:** These bits define the page of memory map structure for current active slave address.
7:0	MEM_ADDR	**Memory Address.** These bits maintain current read address of memory map structure for the current active slave address and is used in word mode.

Writing 0000h to RPTNR will cause this register to update with a pointer to the current SRAM location to store data based upon the memory location defined in the active slave address' MADDR register and the captured memory location.

11.2.15 – I²C TX Page Interrupt Enable Register (I2CTXFIE)

Bit	7	6	5	4	3	2	1	0
Name	TXPG_EN	-	-	-	-	-	THSH	-
Reset	0	0	0	0	0	0	0	0
Access	rw	r	r	r	r	r	rw	r

BIT	NAME	DESCRIPTION
7	TXPG_EN	**TX PAGE ENABLE:** Setting this bit to '1', enables the TX PAGE for all enabled slave addresses.
6:2	Reserved	**Reserved.** The user should not write to these bits.
1	THSH	**TX Page Threshold Reach Enable:** Setting this bit to '1', enables TX page threshold reach interrupt.
0	Reserved	**Reserved.** The user should not write to this bit.

11.2.16 – I²C TX Page Status Register (I2CTXFST)

Bit	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	THSH	-
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	rw	r

BIT	NAME	DESCRIPTION
7:2	Reserved	**Reserved.** The user should not write to these bits.
1	THSH	**TX Page Threshold Reach Enable:** The I²C controller sets this bit to '1' when number of bytes remaining in the TX page is 4 for the current active slave.
0	Reserved	**Reserved.** The user should not write to this bit.

11.2.17 – I²C Receive FIFO Interrupt Enable (I2CRXFIE)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	FIFO_EN	-	-	-	FULL	-	THSH	EMPTY
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	rw	r	r	r	rw	r	rw	rw

BIT	NAME	DESCRIPTION
15:8	Reserved	**Reserved.** The user should not write to these bits.
7	FIFO_EN	**FIFO Enable:** Setting this bit to '1', enables the receive FIFO.
6:4	Reserved	**Reserved.** The user should not write to these bits.
3	FULL	**FIFO FULL:** Setting this bit to '1', generates an interrupt when FIFO receives 8 bytes (FIFO FULL).
2	Reserved	**Reserved.** The user should not write to these bits.
1	THSH	**FIFO THSH:** Setting this bit to '1', generates an interrupt when FIFO receives 4 bytes.
0	EMPTY	**FIFO EMPTY:** Setting this bit to '1', generates an interrupt when receive FIFO is empty

11.2.18 – I²C Receive FIFO Interrupt Enable (I2CRXFST)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	-	-	-	-	FULL	-	THSH	EMPTY
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	r	r	rw	r	rw	rw

BIT	NAME	DESCRIPTION
15:4	Reserved	**Reserved.** The user should not write to these bits.
3	FULL	**FIFO FULL:** This bit indicates that the receive FIFO has received 8 bytes. This bit must be cleared to '0' by software once set. Setting this bit to '1' by software causes an interrupt if enabled.
2	Reserved	**Reserved.** The user should not write to these bits.
1	THSH	**FIFO EMPTY:** This bit indicates that the receive FIFO has received 4 bytes. This bit must be cleared to '0' by software once set. Setting this bit to '1' by software causes an interrupt if enabled.
0	EMPTY	**FIFO EMPTY:** This bit indicates that the receive FIFO is empty. This bit must be cleared to '0' by software once set. Setting this bit to '1' by software causes an interrupt if enabled.

SECTION 12 – SERIAL PERIPHERAL INTERFACE (SPI)

The DS4830A provides two independent Serial Peripheral Interfaces (SPI) – one defined as SPI Master and SPI Slave. Each SPI module of the DS4830A microcontroller provides an independent serial communication channel to communicate synchronously with peripheral devices in a multiple master or multiple slave system. Each interface allows independent access to a four-wire full-duplex serial bus that can be operated in either master mode or slave mode. The SPI functionality must be enabled by setting the SPI Enable (SPIEN) bit of the SPI Control register to '1'. The maximum data rate of the SPI interface is 1/2 the system clock frequency for master mode operation and 1/4 the system clock frequency for slave mode operation.

Note: Even though SPI Master and SPI Slave interfaces are defined, each interface can operate as SPI Master or SPI Slave or both.

The four external interface signals used by the SPI module are MOSI (Master Out Slave In), MISO (Master In Slave Out), SPI Clock (SPICK), and Slave Select (SSEL).

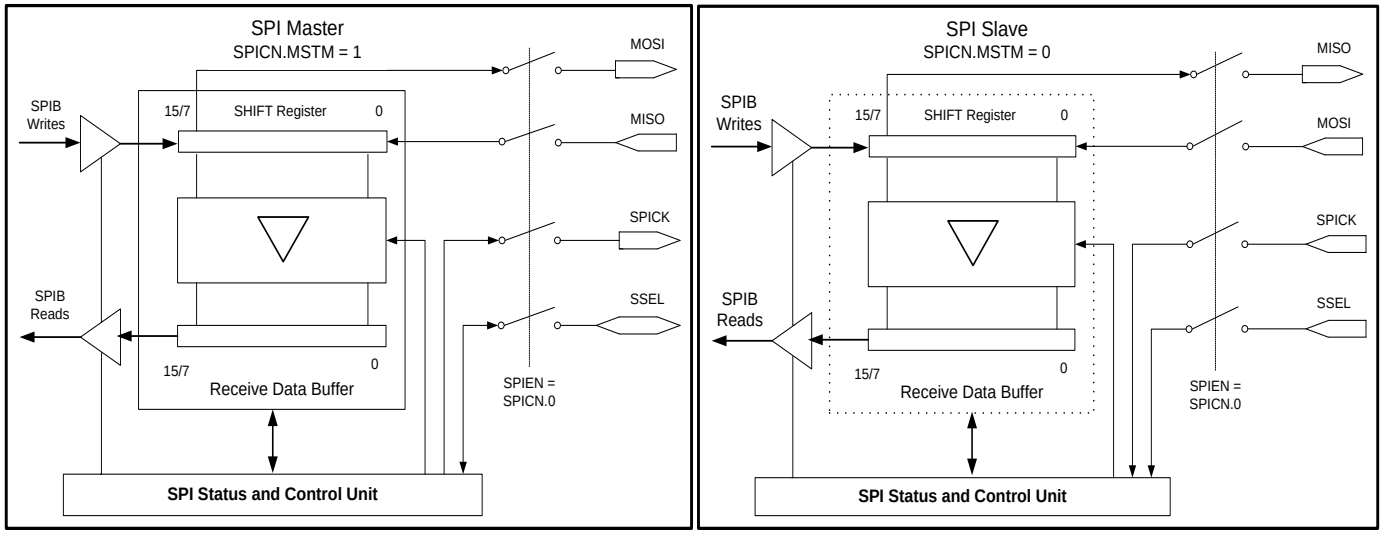


Figure 12-1: SPI Master and Slave Block Diagram

12.1 – Serial Peripheral Interface (SPI) Detailed Description

The block diagram Figure 12-1 shows the SPI external interface signals, control unit, read buffer, and single shift register common to the transmit and receive data path for both the master and slave blocks. SPI can be viewed as a synchronous serial I/O port that shifts data stream of variable length (8 or 16 bits) between peripheral devices. Data is shifted out of the SPI through the programmable shift register which is formed by serially connecting the master's shift register and a slave shift register.

Each time that an SPI transfer completes, the received character is transferred to the read buffer, giving double buffering on the receive side. The CPU has read/write access to the control unit and the SPI data buffer (SPIB). Writes to SPIB are always directed to the shift register while reads always come from the receive data buffer. During an SPI transfer, data is simultaneously transmitted and received. The serial clock signal (SPICK) synchronizes shifting and sampling of the bit stream on the two serial data pins.

For both the master and the slave, data is shifted out of the shift register on one edge of SPICK and latched into the shift register on the opposite SPICK clock edge. The master can initiate data transfer at any time since it controls the serial clock. The slave select signal (SSEL) allows individual selection of slave SPI device in the network.

12.1.1 – SPI Transfer Formats

During an SPI transfer, data is simultaneously transmitted and received over two serial data lines with respect to a single serial shift clock. The polarity and phase of the serial shift clock are the primary components in defining the SPI data transfer format. The polarity of the serial clock corresponds to the idle logic state of the clock line and therefore also defines which clock edge is the active edge. To define a serial shift clock signal that idles in a logic low state (active clock edge = rising), the Clock Polarity Select (CKPOL; SPICF.0) bit should be configured to a 0, while setting CKPOL = 1 causes the shift clock to idle in a logic high state (active clock edge = falling). The phase of the serial clock selects which edge is used to sample the serial shift data. The Clock Phase Select (CKPHA; SPICF.1) bit

controls whether the active or inactive clock edge is used to latch the data. When CKPHA is set to 1, data is sampled on the inactive clock edge (clock returning to the idle state). When CKPHA is set to 0, data is sampled on the active clock edge (clock transition to the active state). Together, the CKPOL and CKPHA bits allow four possible SPI data transfer formats illustrated in Figure 12-2 and Figure 12-3. The Slave Select signal can remain asserted between successive transfers. Table 12-1 illustrates the SPI modes.

Table 12-1: SPI Modes

CKPOL	CKPHA	MODE	SAMPLE POINT
0	0	Mode 0	Rising edge
0	1	Mode 1	Falling edge
1	0	Mode 2	Falling edge
1	1	Mode 3	Rising edge

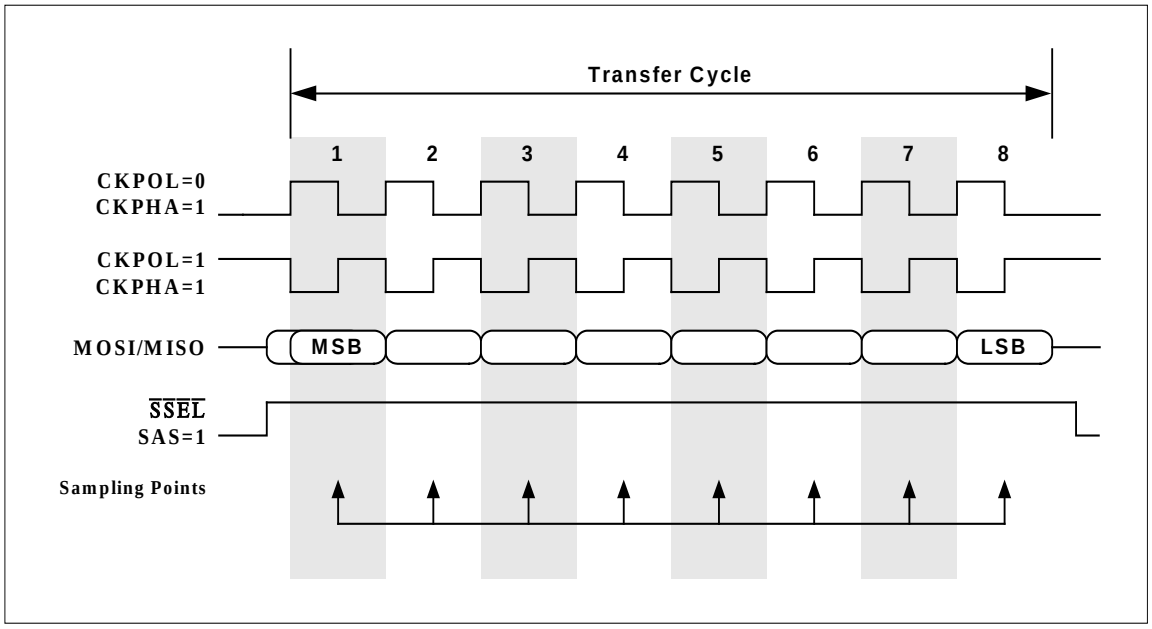


Figure 12-2: SPI Transfer Formats (CKPHA=1)

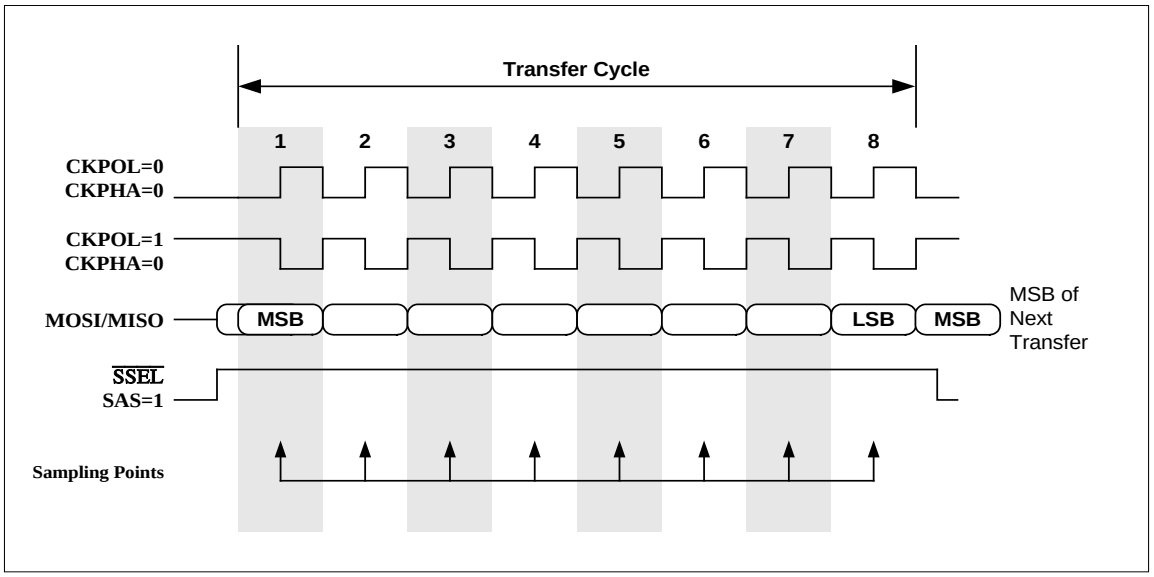


Figure 12-3: SPI Transfer Formats (CKPHA=0)

12.1.2 – SPI Character Lengths

To flexibly accommodate different SPI transfer data lengths, the character length for any transfer is user configurable via the Character Length Bit (CHR) in the SPI Configuration Register. These are independently configurable for the master and slave SPI. The CHR bit allows selection of either 8-bit or 16-bit transfers. When CHR is 0, the character length is 8-bits; when CHR is set to 1, the character length is 16 bits.

When loading 8-bit characters into the SPIB data buffer, the byte for transmission should be right-justified or placed in the least significant byte of the word. When a byte transfer completes, the received byte is right-justified and can be read from the least significant byte of the SPIB word. The most significant byte of the SPIB data buffer is not used when transmitting and receiving 8-bit characters.

12.2 – SPI System Errors

Three types of SPI system errors can be detected by the SPI module. A mode fault error arises in a multiple master system when more than one SPI device simultaneously tries to be a master. A receive overrun error occurs when an SPI transfer completes before the previous character has been read from the receive data buffer. The third kind of error, write collision, indicates that an attempted write to SPIB was detected while a transfer was in progress (STBY=1).

12.2.1 – Mode Fault

When a SPI device is configured as a master and its Mode Fault Enable bit (SPICN.2: MODFE) is also set, the Slave Select pin is configured as input for mode fault detection. The mode fault error occurs if Slave Select signal is asserted by an external device. This error can occur in multi master system when a second SPI device attempts to function as a master in the system. This causes the possibility of contention, which may damage the CMOS push pull drivers. The active state of Slave Select is defined by Slave Active Select bit (SPICF.6: SAS). If SAS is cleared to 0 and a low SSEL input signal is detected while MODFE is set, a mode fault error has occurred. If SAS is set to 1, a high SSEL signal indicates that a mode fault error has occurred. The mode fault error detection is to provide protection from such damage by disabling the bus drivers. When a mode fault is detected, the following actions are taken immediately by hardware:

1. The MSTM bit is forced to 0 to reconfigure the SPI device as a slave.
2. The SPIEN bit is forced to 0 to disable the SPI module.
3. The Mode Fault (SPICN.3: MODF) status flag is set. Setting the MODF bit can generate an interrupt if it is enabled.

The application software must correct the system conflict before resuming its normal operation. The MODF flag is set automatically by hardware but must be cleared by software or a reset once set. Setting the MODF bit to 1 by software causes an interrupt if enabled.

Mode fault detection is optional and can be disabled by clearing the MODFE bit to 0. Disabling the mode fault detection will disable the function of the Slave Select signal during the master mode operation, allowing the associated port pin to be used as a general-purpose I/O.

Note that the mode fault mechanism does not provide full protection from bus contention in multiple master, multiple slave systems. For example, if two devices are configured as master at the same time, the mode fault-detect circuitry offers protection only when one of them selects the other as slave by asserting its Slave Select signal. Also, if a master accidentally activates more than one slave and those devices try to simultaneously drive their output pins, bus contention can occur without a mode fault error being generated.

12.2.2 – Receive Overrun

Since the receive direction of SPI is double buffered, there is no overrun condition as long as the received character in the read buffer is read before the next character in the shift register is ready to be transferred to the read buffer. However, if previous data in the read buffer has not been read out when a transfer cycle is completed and the new character is loaded into the read buffer, a receive overrun occurs and the Receive Overrun flag (SPICN.5: ROVR) will be set. Setting the ROVR flag indicates that the newer received character has been overwritten and is lost. Setting the ROVR bit to 1 will cause an interrupt if enabled. Once set, the ROVR bit is cleared only by software or a reset.

12.2.3 – Write Collision While Busy

A write collision occurs if an attempt to write the SPIB data buffer is made during a transfer cycle (STBY=1). Since the shift register is single buffered in the transmit direction, writes to SPIB are made directly into the shift register. Allowing the write to SPIB while another transfer is in progress could easily corrupt the transmit/receive data. When such a write attempt is made, the current transfer continues undisturbed, the attempted write data is not transferred to the shift register, and the control unit sets the Write Collision flag (SPICN.4: WCOL). Setting the WCOL bit to 1 causes an interrupt if SPI interrupt sources are enabled. Once set, the WCOL bit is cleared only by software or a reset. Normally, write collisions are associated solely with slave devices since they do not control initiation of transfers and do not have access to as much information about the SPICK clock as the master. As a master, write collisions are completely avoidable, however, the control unit detects write collisions for both master and slave modes.

12.3 – SPI Interrupts

Four flags in the SPICN SFR can generate an SPI interrupt when enabled.

- Mode Fault (MODF) – This is applicable in Master mode only.
- Write Collision (WCOL)
- Receive overrun
- SPI Transfer Complete

These four bits serve as interrupts flags that allow the system programmer to specify the source of interrupts which may cause an interrupt request to the CPU. These bits default to 0 on reset and must be cleared by software when set. Once the SPI Interrupt is enabled by setting the ESPII bit to '1', any of the four SPI interrupt sources can cause an interrupt.

12.4 – SPI Master

The DS4830A has the following SPI interface signals.

FUNCTIONAL NAME	EXTERNAL PIN NAME
MSPIDI: Input to serial shift register (MISO)	MDI
MSPIDO: Output from serial shift register (MOSI)	MDIO
MSPICK: Serial shift clock sourced to slave device(s) (SPICK)	MCL
MSPICS: (Optional) Mode fault detection input if enabled (MODFE=1) (SSEL)	MCS

12.4.1 – SPI Transfer Baud Rates

When operating in the master mode, the SPI serial clock is sourced to the external slave device(s). The serial clock baud rate is determined by the clock divide ratio specified in the SPI Clock Divider Ratio (SPICK) register. The SPI module supports 256 different clock divide ratio selections for serial clock generation. The SPI Baud rate is determined by the following formula:

$$\text{SPI Baud Rate} = \frac{\text{Core Clock}}{2 * \text{Clock Divide Ratio}} \quad \text{where Clock Divider Ratio} = (\text{SPICK.7:0}) + 1$$

12.4.2 – SPI Master Operation

The SPI module is placed in master mode by setting the Master Mode Enable (MSTM) bit in the SPI Control register to 1. Only an SPI master device can initiate a data transfer. The master is responsible for manually selecting/deselecting slave(s) via the MSPICS signal or any GPIO pin. Writing a data character to the SPI shift register (SPIB) while in master mode starts a data transfer. The SPI master immediately shifts out the data serially on the MSPIDO pin, most significant bit first, while providing the serial clock on MSPICK output. New data is simultaneously received on the MSPIDI pin into the least significant bit of the shift register. The data transfer format (clock polarity and phase), character length, and baud rate are all configurable as described earlier in the section. During the transfer, the SPI Transfer Busy (SPICN.7:STBY) flag will be set to indicate that a transfer is in process. At the end of the transfer, the data contained in the shift register is moved into the receive data buffer, the STBY bit is cleared by hardware, and the SPI Transfer Complete flag (SPICN.6: SPIC) is set. Setting of the SPIC bit will generate an interrupt request if SPI interrupt sources are enabled (ESPII=1).

The SPI master can be configured to transfer either 8 or 16 bits in an operation to accommodate network with different word length requirements. The data transfer rate for the network is determined by the divider ratio in the

master's SPI Clock SFR. The SPI transfer format is selected by the master device using two bits SPI Clock Polarity (CKPOL) and Clock Phase in the SPI Configuration Register.

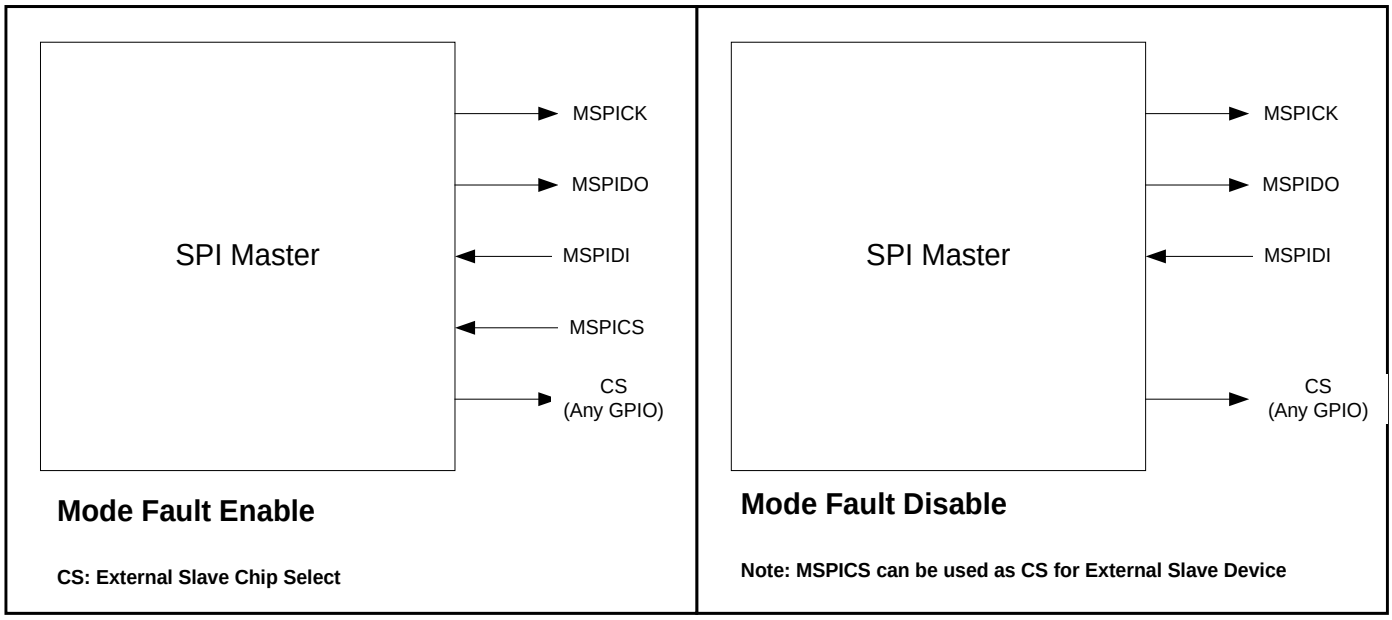


Figure 12-4: SPI Master Pin Configurations with Mode Fault Enable and Disable

In master mode, the MSPICS pin of the master defaults to general-purpose I/O pin. However, as shown in Figure 12-4 the MSPICS can be used for mode fault detection input if the Mode Fault Enable bit (MODFE) is set. When the SPI is configured as a master and the MSPICS pin is used as mode fault detection input, a mode fault condition occurs if an active signal is detected on MSPICS. This indicates that some other device on the network is attempting to be a master. The active signal is defined by the Slave Active Select (SAS) bit. When MODFE is set to 1 and SAS is cleared to 0, an active low signal on MSPICS will trigger a mode fault. If MODFE is set to 1 and SAS is set to 1, an active high signal on MSPICS will indicate a mode fault condition. Either way, the master device will sense the error and immediately disables the SPI device to avoid potentially damaging bus contentions.

To avoid unintentional mode fault error, prior to enabling the SPI peripheral as master with mode fault enabled, software should check the status of MSPICS. MSPICS should be held inactive for at least 2 system clocks before enabling the SPI master. Otherwise, mode fault will occur and the SPI MSTM bit will be cleared to 0 and the SPI disabled.

12.4.3 – SPI Master Register Descriptions

SPI Master Module has four SFR registers. These are SPICN_M, SPICF_M, SPICK_M and SPIB_M. The SPI control register SPICN_M and SPI configuration register SPICF_M controls and configures the Serial Peripheral Interface, respectively. The SPI Clock Register SPICK_M configures SPI Baud rate in Master mode. The SPI Buffer SPIB_M is used in SPI data transfer. SPI Master SFRs are located in Module 5.

12.4.3.1 – SPI Control Register (SPICN_M)

Bit	7	6	5	4	3	2	1	0
Name	STBY	SPIC	ROVR	WCOL	MODF	MODFE	MSTM	SPIEN
Reset	0	0	0	0	0	0	0	0
Access	r	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7	STBY	**Write Transfer Busy Flag.** This bit indicates the current status of the SPI module. STBY is set to '1' when SPI transfer cycle is started and is cleared to '0' when the transfer cycle is completed. This bit is controlled by hardware and is read only for user software.
6	SPIC	**SPI Transfer Complete Flag.** This bit indicates the completion of a transfer cycle when set to '1'. This bit must be cleared to '0' by software once set. Setting this bit to logic '1' by software will cause an interrupt if enabled.
5	ROVR	**Receive Overrun Flag.** This bit indicates a receive overrun when set to '1'. This is caused if two or more characters are received since the last read by the processor. The newer data is lost. This bit must be cleared to '0' by software once set. Setting this bit to logic '1' by software will cause an interrupt if enabled.
4	WCOL	**Write Collision Flag.** This bit indicates a write collision when set to '1'. This is caused by attempting to write to the SPIB while a transfer cycle is in progress. . This bit must be cleared to '0' by software once set. Setting this bit to logic '1' by software will cause an interrupt if enabled.
3	MODF	**Mode Fault**. This bit is the mode fault flag when the SPI is operating as a master If the MODFE bit is set, the active signal that causes a mode fault error is defined in the SAS bit. If the SAS bit is cleared to 0, a low MSPICS signal will trigger a mode fault error. If the SAS bit is set to 1, a high MSPICS signal will indicate that the mode fault error has occurred. This bit must be cleared to '0' by software once set. Setting this bit to logic '1' by software will cause an interrupt if enabled. This flag has no meaning in slave mode.
2	MODFE	**Mode Fault Enable.** When set to '1', MSPICS will be utilized for mode fault detection during SPI master mode operation. When cleared to '0', the MSPICS input has no function and its pin can be used for other purposes.
1	MSTM	**Master Mode Enable.** When set to '1', the SPI module will operate in Master mode when the SPI module is enabled (SPIEN = 1). When set to '0', SPI module will operate in Slave mode when the SPI module enabled (SPIEN = 1).
0	SPIEN	**SPI Enable.** Setting this bit to '1', enables the SPI Module. Setting this bit to '0', disables the SPI module.

12.4.3.2 – SPI Configuration Register (SPICF_M)

Bit	7	6	5	4	3	2	1	0
Name	ESPII	SAS	-	-	-	CHR	CKPHA	CKPOL
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	r	r	r	rw	rw	rw

BIT	NAME	DESCRIPTION
7	ESPII	**SPI Interrupt Enable.** Setting this bit to '1' enables the SPI interrupt when MODF, WCOL, ROVR or SPIC flags are set. Clearing this bit to '0' disables the SPI interrupt.
6	SAS	**Slave Active Select.** In Master mode, this is used only when mode fault is enabled. If SAS = 0, then mode fault is detected when active low is detected on MSPICS pin. If SAS = 1, then mode fault is detected when active high is detected on MSPICS pin.
5:3	Reserved	**Reserved, Read Returns 0.**
2	CHR	**Character Length Bit.** The CHR bit determines the character length for an SPI transfer cycle. A character can consist of 8 or 16 bits in length. When CHR bit is '0', the character is 8 bits; when CHR is set to '1', the character is 16 bits.
1	CKPHA	**SPI Clock Phase Select.** This bit is used with the CKPOL bit to determine the SPI transfer format. When the CKPHA is set to '1', the SPI will sample input data at an inactive edge. When the CKPOL is cleared to 0, the SPI will sample input at an active edge.
0	CKPOL	**SPI Clock Polarity Select.** This bit is used with the CKPHA bit to determine the SPI transfer format. When the CKPOL is set to '1', the SPI uses the clock falling edge as an active edge. When the CKPOL is cleared to 0, the SPI selects the clock rising edge as an active edge.

12.4.3.3 – SPI Clock Register (SPICK_M)

Bit	7	6	5	4	3	2	1	0
Name	SPICK_M[7:0]							
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7:0	SPICK_M[7:0]	**Clock Divide Ratio Bits.** These bits select one of the 256 divide ratios (0 to 255) used for the baud rate generator, with bit 7 as the most significant. The frequency of SPI baud rate is calculated using the following equation: **SPI Baud Rate = ½ x Core Clock / (SPICK[7:0] + 1)**

12.4.3.4 – SPI Data Buffer Register (SPIB_M)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	SPIB_M[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access*	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

*Unrestricted read, write is allowed outside of a transfer cycle; when the STBY bit is set, write is blocked and will cause write collision error.

BIT	NAME	DESCRIPTION
15:0	SPIB_M[15:0]	**SPI Data Buffer Bits.** Data for SPI is read from or written to this location. The serial transmit and receive buffers are separate but both are addressed at this location.

12.5 – SPI Slave

The DS4830A has the following SPI interface signals.

FUNCTIONAL NAME	EXTERNAL PIN NAME
SSPIDO: Output from serial shift register (MISO)	GP6
SSPIDI: Input to serial shift register (MOSI)	SDA
SSPICK: Serial shift clock from SPI master (SPICK)	SCL
SSPICS: Slave select input (CS)	GP7

12.5.1 – SPI Slave Select

The SPI Slave Select SSPICS can be configured to accept either an active low or active high signal via the Slave Active Select Bit (SAS) in the SPI Configuration Register. The SAS bit allows the selection of SSPICS active state. When SAS is cleared to 0, SSPICS is configured to be active low. When SAS is set to 1, SSPICS is configured to be active high.

12.5.2 – SPI Transfer Baud Rates

When operating as a slave device, the SPI serial clock is driven by an external master. For proper slave operation, the serial clock provided by the external master should not exceed the system clock frequency divided by 4.

12.5.3 – SPI Slave Operation

The SPI module operates in the slave mode when the MSTM bit is cleared to 0. In Slave mode, the SPI is dependent on the SSPICK sourced from the master to control the data transfer.

The Slave Select SSPICS input must be externally asserted by a master before data exchange can take place. SSPICS must be asserted before data transaction begin and must remain asserted for the duration of the transaction. If data is to be transmitted by the slave device, it must be written to its shift register before the beginning of a transfer cycle, otherwise the character already in the shift register will be transferred. The slave device considers a transfer to begin with the first clock edge or the active SSPICS edge, dependent on the data transfer format. When SAS is cleared to 0, the active SSPICS edge is the falling edge of SSPICS while if SAS is set to 1, the active SSPCIS edge is the rising edge of SSPICS.

The SPI slave receives data from the external master SSPIDI pin, most significant bit first, while simultaneously transferring the contents of its shift register to the master on the SSPIDO pin, also most significant bit first. Data received from the external master replaces data in the internal shift register until the transfer completes. Just like in the master mode of operation, received data is loaded into the read buffer and the SPI Transfer Complete flag is set at the end of transfer. The setting of the Transfer Complete flag will generate an interrupt request if enabled. Note also that when CKPHA=0, the most significant bit of the SPI data buffer will be shifted out on the 8^{th} shift clock edge.

When SSPICS is not asserted, the slave device ignores the SSPICK clock and the shift register is disabled. Under this condition, the device is basically idle, no data is shifted out from the shift register and no data is sampled from the SSPIDI pin. The SSPIDO pin is placed in an input mode and is weakly pulled high to allow other devices on the bus to drive the bus. De-assertion of the SSPICS signal by the master during a transfer (before a full character, as defined by CHR, is received) aborts the current transfer. When the transfer is aborted, no data is loaded into the read buffer, the SPIC flag is not set, and the slave logic and the bit counter are reset.

In slave mode, the Clock Divider Ratio bits (CKR7:0) have no function since the serial clock is supplied by an external master. The transfer format (CKPOL, CKPHA settings) and the character length selection (CHR) for the slave device, however, should match the master for a proper communication.

Slave mode is used when the SPI is controlled by another peripheral device. The SPI is in slave mode when the MSTM bit is cleared to logic 0.

Each SPI (named as SPI master or SPI slave in this section) can be used as either SPI master or Slave.

12.5.4 – SPI Slave Register Descriptions

SPI Slave Module has four SFR registers. These are SPICN_S, SPICF_S, SPICK_S, and SPIB_S. The SPI control register SPICN_S and SPI configuration register SPICF_S controls and configures the Serial Peripheral Interface respectively. The SPI Clock Register SPICK_S is not used in SPI Slave mode as SPI clock is driven by SPI Master. The SPI Buffer SPIB_S is used in SPI data transfer. SPI Slave SFRs are located in Module 1.

12.5.4.1 – SPI Control Register (SPICN_S)

Bit	7	6	5	4	3	2	1	0
Name	STBY	SPIC	ROVR	WCOL	MODF	MODFE	MSTM	SPIEN
Reset	0	0	0	0	0	0	0	0
Access	r	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7	STBY	**Write Transfer Busy Flag.** This bit indicates the current status of the SPI module. STBY is set to '1' when SPI transfer cycle is started and is cleared to '0' when the transfer cycle is completed. This bit is controlled by hardware and is read only for user software.
6	SPIC	**SPI Transfer Complete Flag.** This bit indicates the completion of a transfer cycle when set to '1'. This bit must be cleared to '0' by software once set. Setting this bit to logic '1' by software will cause an interrupt if enabled.
5	ROVR	**Receive Overrun Flag.** This bit indicates a receive overrun when set to '1'. This is caused if two or more characters are received since the last read by the processor. The newer data is lost. This bit must be cleared to '0' by software once set. Setting this bit to logic '1' by software will cause an interrupt if enabled.
4	WCOL	**Write Collision Flag.** This bit indicates a write collision when set to '1'. This is caused by attempting to write to the SPIB while a transfer cycle is in progress. . This bit must be cleared to '0' by software once set. Setting this bit to logic '1' by software will cause an interrupt if enabled.
3	MODF	**Mode Fault**. This flag has no meaning in slave mode.
2	MODFE	**Mode Fault Enable** This flag has no meaning in slave mode. In slave mode, the SSPICS pin always functions as a slave select input signal to the SPI module, independent of the MODFE bit.
1	MSTM	**Master Mode Enable.** When set to '1', SPI module will operate as Master mode when SPI module is enabled (SPIEN = 1). When set to '0', SPI module will operate as Slave mode when SPI module enabled (SPIEN = 1).
0	SPIEN	**SPI Enable.** Setting this bit to '1', enables SPI Module. Setting this bit to '0', disables the SPI module.

12.5.4.2 – SPI Configuration Register (SPICF_S)

Bit	7	6	5	4	3	2	1	0
Name	ESPII	SAS	-	-	-	CHR	CKPHA	CKPOL
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	r	r	r	rw	rw	rw

BIT	NAME	DESCRIPTION
7	ESPII	**SPI Interrupt Enable.** Setting this bit to '1' enables the SPI interrupt when MODF, WCOL, ROVR or SPIC flags are set. Clearing this bit to '0' disables the SPI interrupt.
6	SAS	**Slave Active Select.** In Slave Mode, this bit is used to determine the SSPICS active state. When the SAS is cleared to '0', the SSPICS is active low and will respond to an external low signal. When the SAS is set to '1', the SSPICS is active high.
5:3	Reserved	**Reserved, Read Returns 0.**
2	CHR	**Character Length Bit.** The CHR bit determines the character length for an SPI transfer cycle. A character can consist of 8 or 16 bits in length. When CHR bit is '0', the character is 8 bits; when CHR is set to '1', the character is 16 bits.
1	CKPHA	**SPI Clock Phase Select.** This bit is used with the CKPOL bit to determine the SPI transfer format. When the CKPHA is set to '1', the SPI will sample input data at an inactive edge. When the CKPOL is cleared to 0, the SPI will sample input at an active edge.
0	CKPOL	**SPI Clock Polarity Select.** This bit is used with the CKPHA bit to determine the SPI transfer format. When the CKPOL is set to '1', the SPI uses the clock falling edge as an active edge. When the CKPOL is cleared to 0, the SPI selects the clock rising edge as an active edge.

12.5.4.3 – SPI Clock Register (SPICK_S)

Bit	7	6	5	4	3	2	1	0
Name	SPICK_S[7:0]							
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7:0	SPICK_S[7:0]	The register has no function when operation in slave mode and clock generation circuitry is disabled.

12.5.4.4 – SPI Data Buffer Register (SPIB_S)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	SPIB_S[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access*	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

*Unrestricted read, write is allowed outside of a transfer cycle; when the STBY bit is set, write is blocked and will cause write collision error.

BIT	NAME	DESCRIPTION
15:0	SPIB_S[15:0]	**SPI Data Buffer Bits.** Data for SPI is read from or written to this location. The serial transmit and receive buffers are separate but both are addressed at this location.

SECTION 13 – 3-WIRE

The DS4830A has proprietary 3-Wire master interface for communication with MAXIM 3-wire laser drivers (which supports MSB first 3-wire protocol). The 3-wire communication mode operates similar to SPI mode. However, in the 3-wire mode, there is one bi-directional I/O instead of separate data in and data out signals. The 3-wire interface consists of the MCS, MDIO and MCL. The 3-Wire Master interface reads data on the falling edge of MCL. During 3-Wire write operation the 3-Wire master outputs the data on the falling edge of MCL.

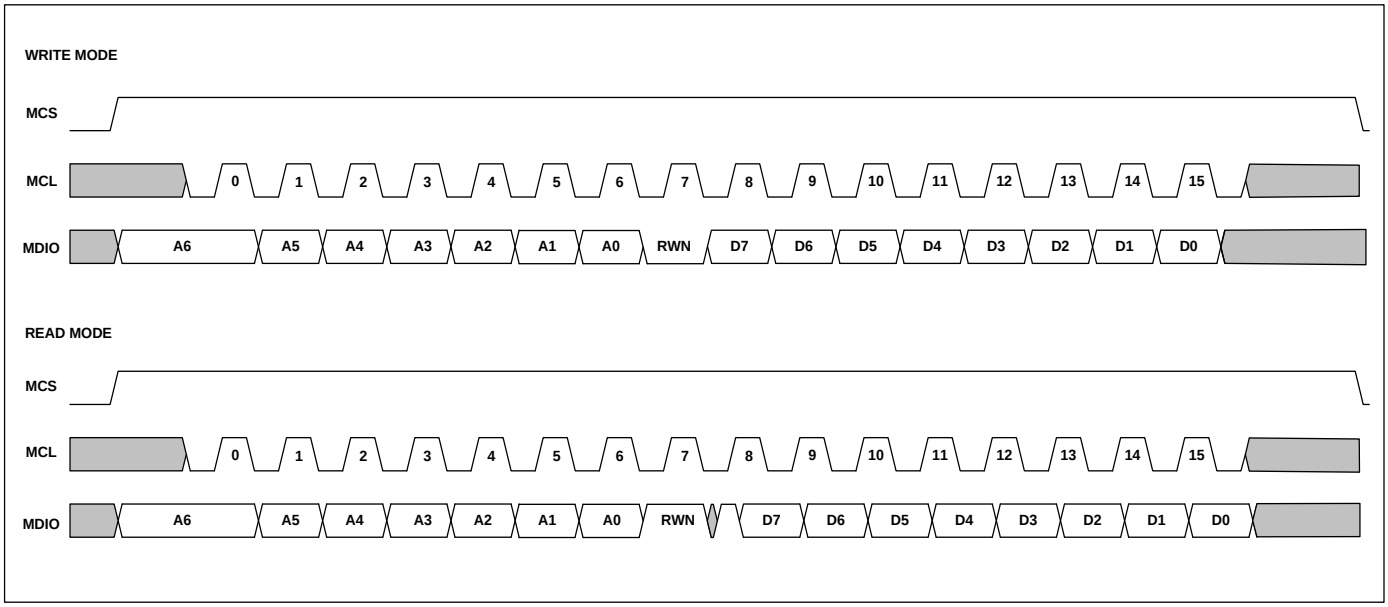


Figure 13-1: 3-Wire Write and Read Operation

13.1 – Detailed Description

The DS4830A has a proprietary 3-Wire digital serial interface and it is designed to interface with Maxim 3-wire slave devices (Laser drivers). The DS4830A acts as the 3-Wire master. It is a 3-pin interface consisting of MDIO a bidirectional data line, MCL clock signal and MCS chip select output. Chip select is active high. The 3-Wire master initiates communication by generating clock.

By default, 3-Wire Chip select is enabled and it is automatically controlled by 3-Wire interface during the communication. The DS4830A 3-Wire interface supports byte mode data transfer. The 3-Wire Control Register (TWR) is used to control and configure the 3-Wire interface. The 3-Wire interface provides 8 user selectable MCL clock frequencies. The 3-Wire communication is enabled by setting the TWEN bit to '1' in the TWR register and MCS goes to low. Data transfer is initiated on next core clock after writing to the Data and Address Register (DADDR).

13.1.1 – Operation

The DS4830A 3-wire master supports 8 user configurable communication clock frequencies. These are selected by writing to the TWCP [2:0] bits in the TWR register. Each 3-Wire packet consists of 16-bits (15-bit address/data, 1-bit RWN). See Table 13-1 for 3-Wire Data Packet.

Table 13-1: 3-Wire Data Packet

BIT NUMBER	NAME	DESCRIPTION
15 to 9	ADDR (Address)	7-bit Internal Register Address (3-Wire Slave)
8	RWN	0 - Write 1 – Read
7 to 0	DATA	8-bit Read or Write Data

The 3-Wire interface is enabled when the TWEN bit in the TWR register is set to '1'. Using the DADDR register 3-Wire write (RWN = 0) and read (RWN = 1) operations are performed. The 3-Wire master supports 7-bit read or write address and 8-bit data. Write to the DADDR register automatically starts the data transfer and the 3-Wire interface sets BUSY flag to '1'. The BUSY flag is reset to '0' when the data transfer is completed.

13.1.1.1 – Write Mode (RWN=0)

The 3-Wire master generates 16 clock cycles on MCL pin. It outputs 16-bits (MSB first DADDR data) to the MDIO line at the falling edge of the MCL. After completion of 16 clocks, the 3-Wire BUSY flag is cleared and the data transfer complete flag TWI is set to '1' which generates interrupt if enabled. The master closes the transmission by setting the MCS to '0'.

13.1.1.2 – Read Mode (RWN=1)

The 3-Wire master generates 16 clock cycles at MCL. It outputs 8-bits of ADDR + RWN (MSB first) to the MDIO line at the falling edge of the clocks. The MDIO line is released after the RWN bit is transmitted. The slave outputs 8-bits of data (MSB first) at rising edge of the clock. The master reads the data bits at the falling edge of the clocks. After the completion of 16 clocks, the 3-Wire BUSY flag is cleared and the data transfer complete flag TWI is set to '1' which generates interrupt if enabled. Read data is available in the DADDR [7:0] bits and the DADDR[8:15] bits set to 0.The master closes the transmission by setting the MCS to '0'.

13.1.1.3 – Chip Select Disable Mode (TWCDIS = 1)

The DS4830A 3-Wire master provides facility to disable MCS chip select. In this mode, any GPIO can be configured to function as chip select and the 3-Wire interface does not control MCS during the communication. In chip select disabled mode, the application program should control chip select during the 3-Wire communication. Using this feature, multiple 3-Wire slaves can be interfaced with the 3-Wire master.

13.2 – 3-Wire Register Descriptions

The 3-Wire interface is controlled by two SFR registers. These are the 3-Wire Control Register TWR and Data and Address Register DADDR. The TWR register configures and controls 3-Wire interface. The DADDR is used in 3-Wire read and write operation. These registers are located at Module 2.

13.2.1 – 3-Wire Control Register (TWR)

Bit	7	6	5	4	3	2	1	0
Name	TWEN	TWCP[2:0]			TWIE	TWCSDIS	TWI	BUSY
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	r

BIT	NAME	DESCRIPTION
7	TWEN	**3-Wire Enable.** This bit enables the 3-Wire interface. When this bit is set to '1', the 3-Wire interface is enabled. When this bit is cleared, the 3-Wire function is disabled.
6:4	TWCP[2:0]	**3-Wire Clock Period.** These bits are used for setting the 3-Wire MCL clock period.
3	TWIE	**3-Wire Interrupt Enable.** Setting this bit to '1' will enable an interrupt when the 3-Wire data transfer is completed. Clearing this bit will disable the 3-Wire data transfer complete interrupt.
2	TWCDIS	**3-Wire Chip Select disable.** Setting this bit to '1', will disable the chip select and the 3-Wire Master interface will not control the chip select MCS during the communication. In chip select disable mode, application program should control the 3-Wire chip select by any GPIO. Clearing this bit will enable MCS as active chip select and it is set to HIGH (See Figure 13-1) at start of 3-Wire data communication and set to LOW once the 3-Wire data communication is completed.
1	TWI	**3-Wire Interrupt.** This bit is set to '1' when data transfer is completed. This bit can generate interrupt if TWIE bit is enabled. Once set, it should be cleared by software.
0	BUSY	**3-Wire Busy.** This bit is set to '1' when data is written to the DADDR register and it indicates that the data transfer is in progress. This bit is reset to '0' once the data transfer is completed. This is also reset to zero when 3-Wire operation is disabled (the TWEN bit is '0'). This is read only bit.

TWCP[2:0]	MCL Clock Frequency (Period)
000	1 MHz (1000nSec)
001	714 KHz (1400nSec)
010	555 KHz (1800nSec)
011	454 KHz (2200nSec)
100	384 KHz (2600nSec)
101	333 KHz (3000nSec)
110	294 KHz (3400nSec)
111	263 KHz (3800nSec)

13.2.2 – Data and Address Register (DADDR)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	ADDR[15:9]							RWN	DATA[7:0]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:9	ADDR[6:0]	**3-Wire Address.** These bits specify Slave device internal register address.
8	RWN	**Read or Write Select.** When this bit is set to '1', the 3-Wire 'Read' operation is performed. When this bit is '0', the 3-Wire 'Write' operation is performed.
7:0	DATA[7:0]	**3-Wire Data.** During 3-Wire 'Read' operation (RWN = 1), the master writes the read data from the 3-Wire bus at these bits. During 3-Wire 'Write' operation (RWN = 0), the master sends data written at these bits on the 3-Wire bus.

Important Note: The entire DADDR register should be written at once instead of writing individual bits or fields.

SECTION 14 – PWM

The DS4830A provides 10 independent PWM output pins that can be used to create DC-DC power supply controllers or a Thermoelectric Cooler Controller (TECC)

Refer to Application Note 5424: Thermoelectric Cooler Control Using the DS4830 Optical Microcontroller.

14.1 – Detailed Description

The DS4830A provides 10 independently configurable PWM outputs. The DS4830A PWM controller has 3 SFRs PWMCN, PWMDATA and PWMSYNC for configuration and control of the 10 PWM outputs. Using PWMCN and PWMDATA, individual PWM channels can be programmed for unique duty cycles (DCYCn), configurations (PWMCFGn), and delays (PWMDLYn), where n represents the PWM channel number. The DS4830A provides three types of driving strength PWM outputs. Refer to the DS4830A IC data sheet for more information.

The PWM block has three SFRs that are accessed in module 5 (PWMCN, PWMDATA and PWMSYNC). All aspects of the PWM block can be programmed using these 3 SFRs.

14.1.1 – PWMCN and PWMDATA SFRs

The PWM Control SFR (PWMCN) along with the PWM Data SFR (PWMDATA) is used to configure and control individual PWM channels. All the channels can be independently configured. Figure 14-1 illustrates how this is accomplished.

The PWMCN SFR has 4 bits (PWM_SEL) that select a particular PWM channel to be configured (See PWM Register Descriptions for details). 2 bits (REG_SEL) within the PWMCN SFR allows for programming of 3 local registers for each PWM Channel:

- Duty Cycle (Register DCYCn),
- Configuration (Register PWMCFGn)
- Delay (Register PWMDLYn).

The PWMDATA SFR writes data to the particular local register pointed to by the PWM_SEL and REG_SEL bits as illustrated in Figure 14-1. PWM_SEL auto increments after each read or write operation to PWMDATA register allowing quick configuration.

The PWMCN SFR additionally allows enabling or disabling individual PWM Channels independently as well as update of the Duty Cycle programmed in the DCYCn local register. Table 14-1 explains how the different Local Registers are selected, and is discussed further in the Individual PWM detailed description section.

Table 14-1: Selecting the Local Registers

REG_SEL	LOCAL REGISTER SELECTED
00b	Duty Cycle Register (DCYCn)
01b	PWM Configuration Register (PWMCFGn)
1xb	Delay Setting Register (PWMDLYn)

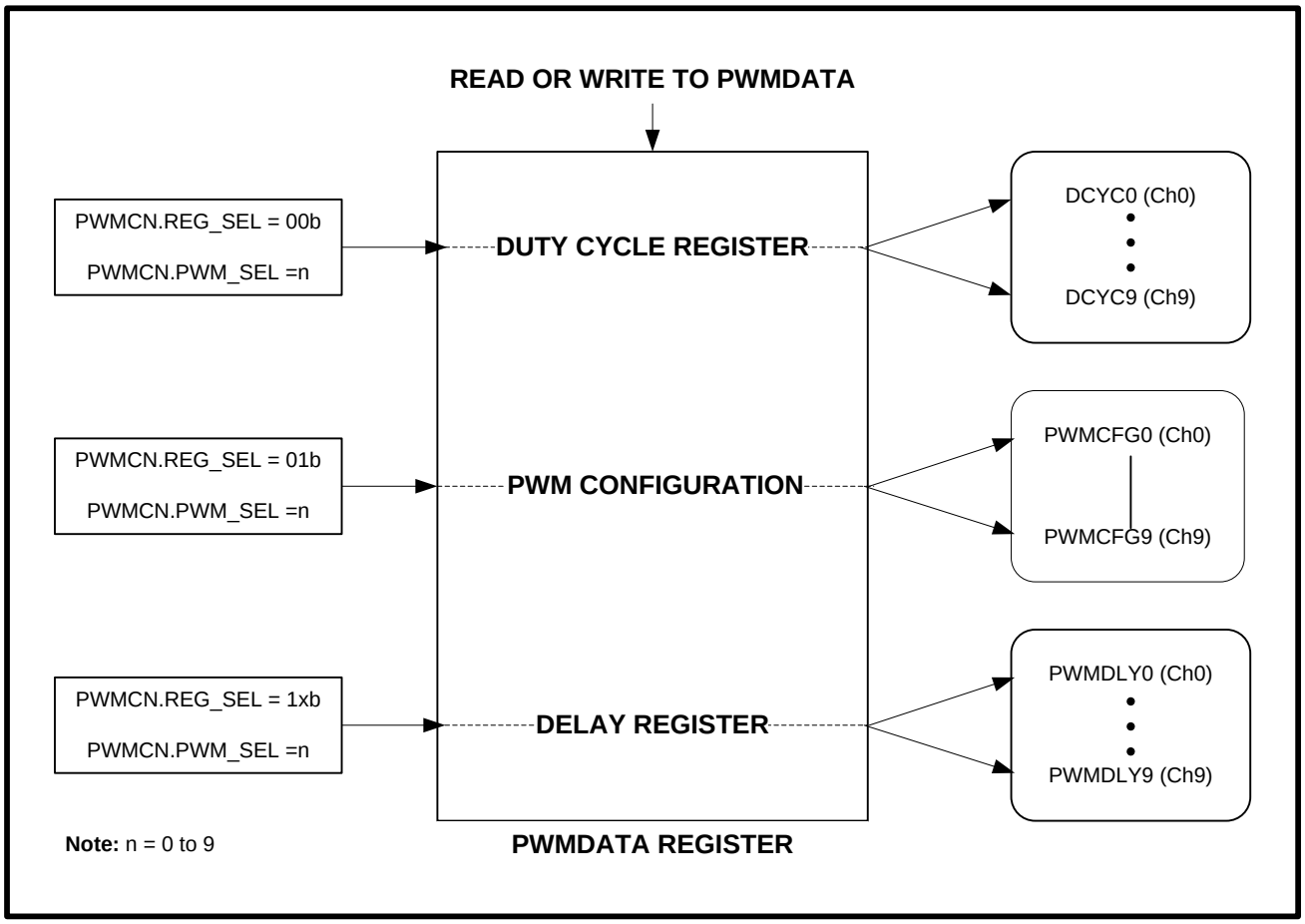


Figure 14-1: Illustration of PWMDATA and PWMCFG SFRs

14.1.2 – PWMSYNC SFR

Different channels can be synchronized using the PWMSYNC register. Doing so effectively brings the channels in phase by restarting the channels that are to be synchronized, without affecting the PWM operation. The PWM channels to be synchronized must have the same configurations (Resolution, Pulse Spreading option, Clock source etc.). The PWMSYNC register auto clears itself on the next core clock. Figure 14-2 shows an illustration of the PWMSYNC SFR operation. See PWM Delay section for more details.

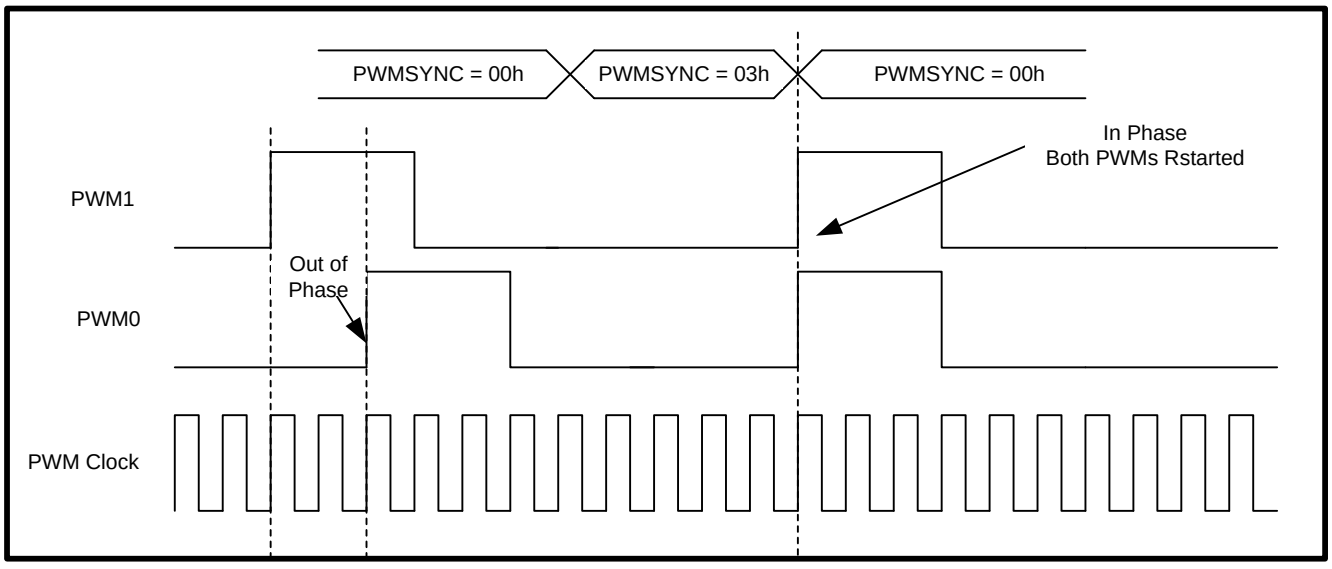


Figure 14-2: PWM Output Synchronization When the Same Delay is Programmed

14.2 – Individual PWM Channel Operation

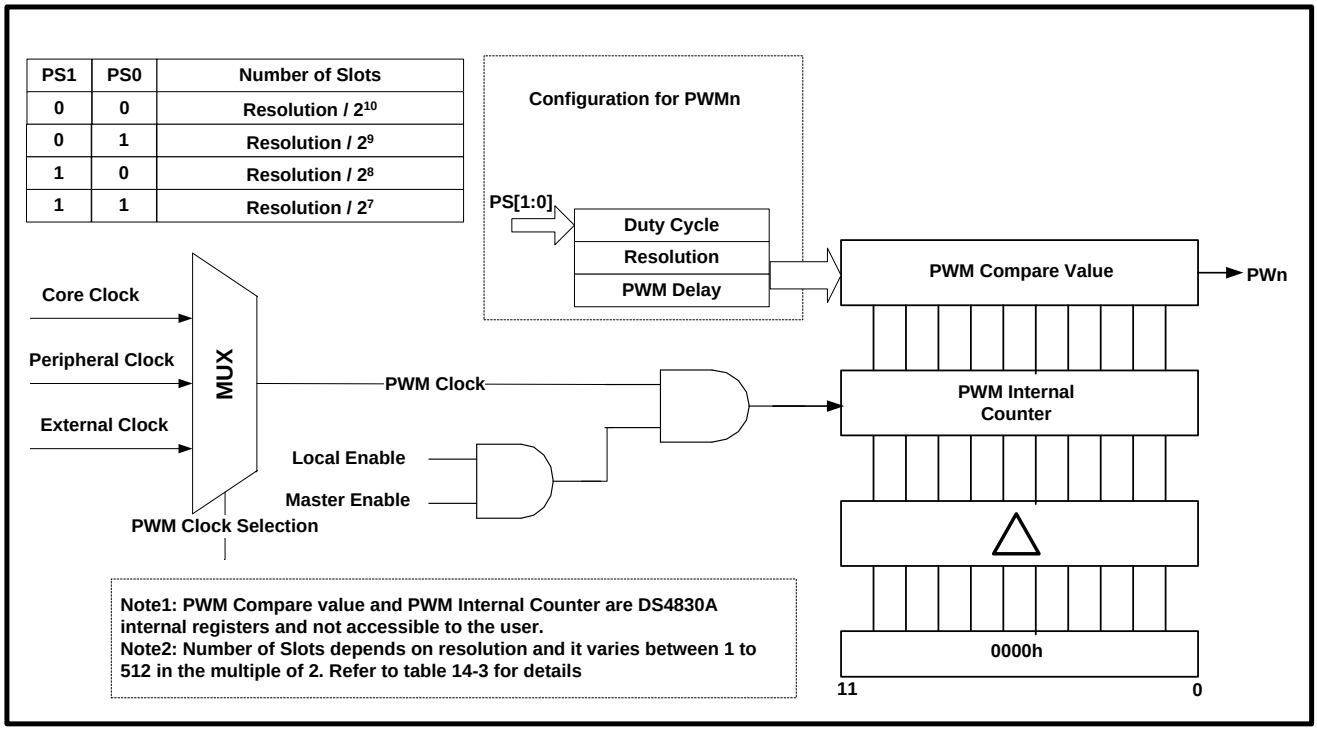


Figure 14-3: Block Diagram of One PWM Channel

The DS4830A has 10 PWMs which can provide up to 16 bits of resolution on each channel. Each channel can be independently enabled or disabled. Each PWM is configured using 3 Local Registers (for a total of 30 Local Registers for programming the 10 PWMs).

The source clock to PWM can be selected from Core clock, Peripheral Clock or External Clock. The external clock range is 20MHz to 133MHz. The PWM frame frequency is calculated from the below formula,

$$PWM\ Frame\ Frequency = \frac{PWM\ Clock\ Frequency}{2^N}$$, Where N is resolution

As explained above the PWMCN SFR points to a particular PWM channel. The local registers are then programmed by writing data to the PWMDATA SFR. The Local Register is selected based on the REG_SEL bits in the PWMCN SFR (See Table 14-1).

Details for programming of the Local Registers are in the "PWM Register Descriptions" section.

14.2.1 – Duty Cycle Register (DCYCn)

This register controls the Duty cycle of the PWM Channel. The number of bits used to program the Duty Cycle depends on the resolution programmed in the PWMCFG register. For 12 bits of resolution, the Duty cycle is the lower 12 bits of the PWMDATA register. However if only 7 bits of resolution is selected, only the lower 7 bits are used to control the Duty Cycle of the corresponding PWM Channel.

To achieve a particular duty cycle, the PWM output level is set to high and the internal counter starts counting from 0000h. The PWM output remains high untill the PWM count is equal to the value in the DCYC register. The PWM controller sets the PWM output to low for the remaining clock counts for the selected resolution. One such cycle represents one PWM frame and repeats until the PWM is disabled.

For example, when 9 bits of resolution is selected and the DCYC register is written to 128, the PWM controller sets the output high for the first 128 counts of PWM clock and sets output low for the remaining 384 PWM clock counts (2^9 =512, = 512 – 128 = 384). The PWM frame in this case is 512 clock cycles. The PWM output frequency depends upon the selected clock source in the PWMCFG register. Figure 14-4 illustrates this example.

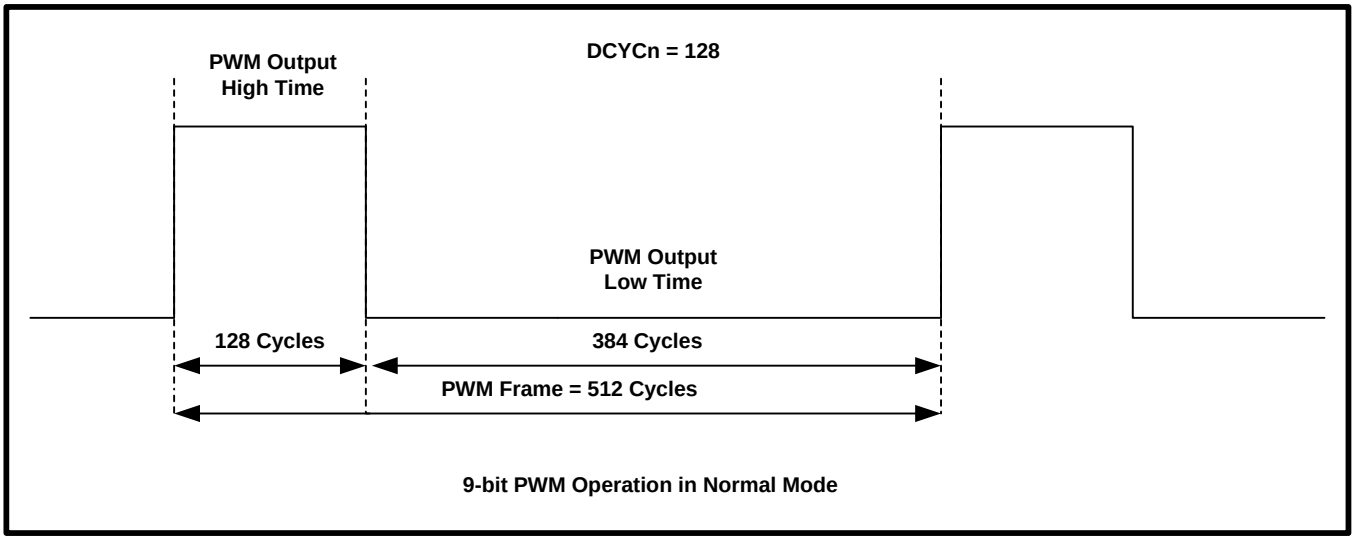


Figure 14-4: PWM Duty Cycle Set to 128 with 9-Bit Resolution

14.2.2 – PWM Configuration Register (PWMCFGn)

This register allows independent configuration of a PWM Channel. Each PWM Channel can be independently disabled or enabled. Each output can have from 7 to 16 bits of resolution and can be inverted.

The PWM channels 0 – 7 are multiplexed with the DAC outputs. The PWMCFGn register allows configuring the outputs to be present at an alternate location instead of the original location.

The PWMs can be clocked using the core clock, the peripheral clock or an external clock as defined by the PWMCFGn register.

14.2.2.1 – Pulse Spreading

The DS4830A PWMs have the ability to perform pulse spreading on the output stream. Pulse spreading divides the PWM frame into equal slots. The DS4830A PWM controller supports nine pulse spreading options with slots varying from 1 to 512 in the multiple of 2. For each resolution selection, up to four pulse spreading options are available. Pulse spreading options can be selected from 2 bits in the PWMCFGn register (PS[1:0]). The PWM controller distributes the duty cycle over the selected number of slots equally. If resolution bits are 12 and Pulse spreading option is 3 then the PWM controller distributes the PWM Frame over 32 equal slots. By doing this, the PWM output frequency becomes 32 times the PWM Frame frequency. In 12-bit resolution, the PWM clock period is 4096 counts long. If Pulse spreading option is set to 3, the PWM frame is divided into 32 slots with each slot taking 128 PWM counts. The duty cycle is equally distributed in each slot using dithering. Each slot frequency can be calculated from the below equation and see Table 14-2 for number of slots for each resolution.

$$PWM\ Slot\ Frequency = \frac{PWM\ Frame\ Frequency \times Number\ of\ Slots}{2^N}$$, where N is resolution.

Table 14-2: Number of Slots for Each Resolution

RES_SEL[3:0]	Resolution	PS[1:0]= 00	PS[1:0]= 01	PS[1:0]= 10	PS[1:0]= 11
		Number of Slots			
0000b	7	1	1	1	1
0001b	8	1	1	1	2
0010b	9	1	1	2	4
0011b	10	1	2	4	8
0100b	11	2	4	8	16
0101b	12	4	8	16	32
0110b	13	8	16	32	64
0111b	14	16	32	64	128
1000b	15	32	64	128	256
1001b	16	64	128	256	512
>1001b	16	64	128	256	512

Pulse Spreading Method

The DS4830A PWM controller uses a delta sigma algorithm to distribute the duty cycle uniformly among the slots. For example, a 10-bit PWM output with a DCYCn value of 128 with 8-slot pulse spreading enabled (PS[1:0] = b'11) produces a PWM output as shown in the Figure 14-5. The duty cycle of 128 in 1024 cycles (10-bit resolution) has been divided over 8 equal slots of 16 PWM clock cycles. As duty cycle increases by a count each time the pulse spread is implemented uniformly and the corresponding duty cycle is distributed among slots. Table 14-3 and Figure 14-5 explain this example. Example considers PWM operation in the positive polarity.

Table 14-3: Duty Cycle Distribution with 8-Slot Pulse Spreading for 10-Bit Resolution PWM Operation

Resolution	Duty Cycle	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7	Slot 8
10	**128**	16	16	16	16	16	16	16	16
	129	16	16	16	16	16	16	16	17
	130	16	16	16	17	16	16	16	17
	131	16	16	17	16	16	17	16	17
	132	16	17	16	17	16	17	16	17
	133	16	17	16	17	17	16	17	17
	134	16	17	17	17	17	16	17	17
	135	16	17	17	17	17	17	17	17
	136	17	17	17	17	17	17	17	17

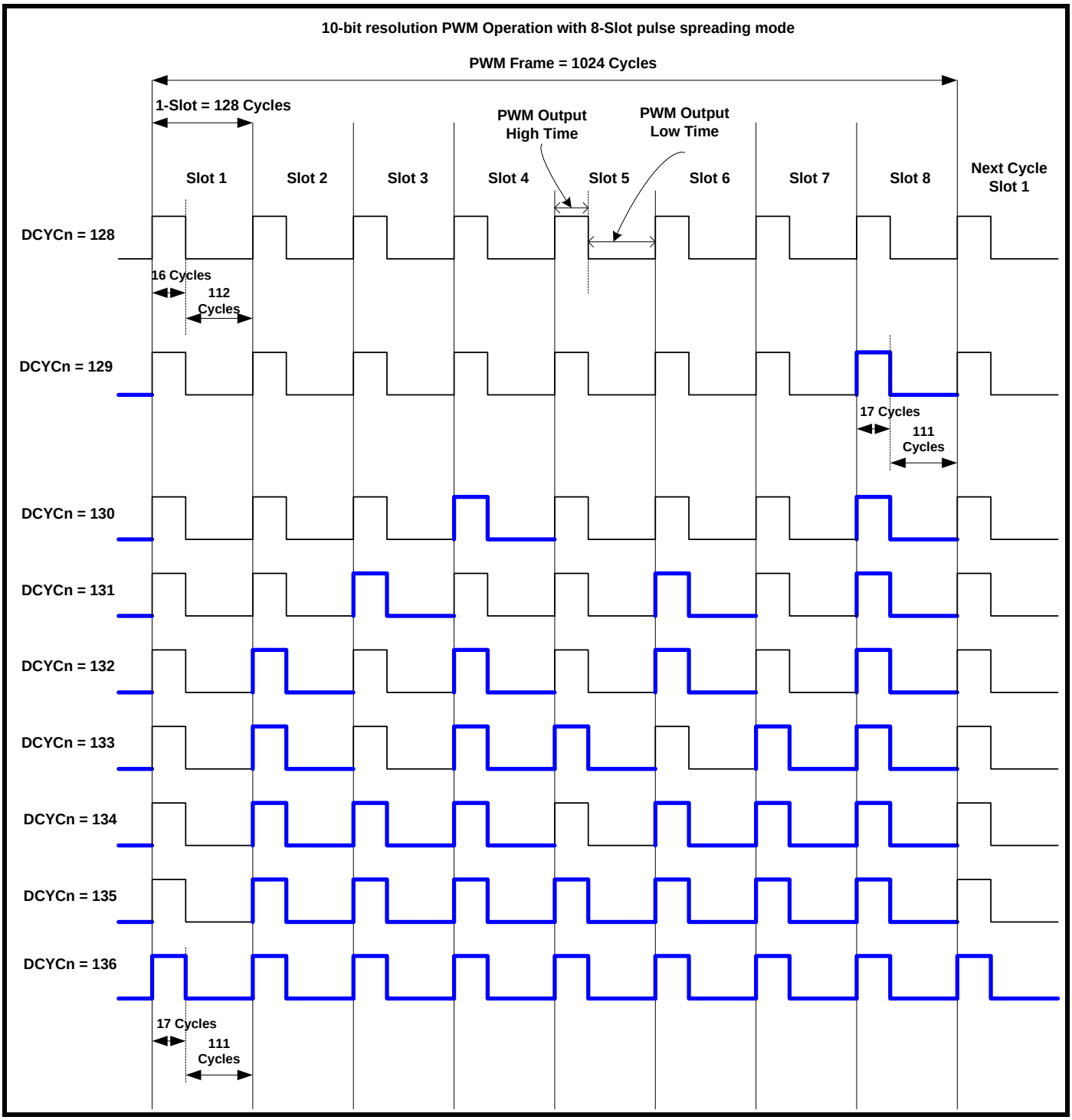


Figure 14-5: Duty Cycle Distribution with 8-Slot Pulse Spreading (PS[1:0] = 11b) for 10-Bit PWM Operation

See Tables 14-4a, 14-4b, and 14-4c for slot frequencies at various resolutions and pulse-spreading options with the different PWM source clock frequencies.

Table 14-4a: Slot Frequencies for Various Resolution and Pulse Spreading with Core Clock = 10MHz

Source = Core Clock (10MHz)					
Resolution	**Frame Frequency (Hz)**	**Pulse Spreading (PS[1:0])**			
		00	**01**	**10**	**11**
		Slot Frequency (Hz)			
7	78125	78125	78125	78125	78125
8	39062.5	39062.5	39062.5	39062.5	78125
9	19531.25	19531.25	19531.25	39062.5	78125
10	9765.625	9765.625	19531.25	39062.5	78125
11	4882.813	9765.625	19531.25	39062.5	78125
12	2441.406	9765.625	19531.25	39062.5	78125
13	1220.703	9765.625	19531.25	39062.5	78125
14	610.352	9765.625	19531.25	39062.5	78125
15	305.176	9765.625	19531.25	39062.5	78125
16	152.588	9765.625	19531.25	39062.5	78125

Table 14-4b: Slot Frequencies for Various Resolution and Pulse Spreading with Peripheral Clock = 20MHz

Source = Peripheral Clock (20MHz)					
Resolution	**Frame Frequency (Hz)**	**Pulse Spreading (PS[1:0])**			
		00	**01**	**10**	**11**
		Slot Frequency (Hz)			
7	156250	156250	156250	156250	156250
8	78125	78125	78125	78125	156250
9	39062.5	39062.5	39062.5	78125	156250
10	19531.25	19531.25	39062.5	78125	156250
11	9765.625	19531.25	39062.5	78125	156250
12	4882.813	19531.25	39062.5	78125	156250
13	2441.406	19531.25	39062.5	78125	156250
14	1220.703	19531.25	39062.5	78125	156250
15	610.352	19531.25	39062.5	78125	156250
16	305.176	19531.25	39062.5	78125	156250

Table 14-4c: Slot Frequencies for Various Resolution and Pulse Spreading with External Clock = 128MHz

Source = External Clock (128MHz)					
Resolution	**Frame Frequency (Hz)**	**Pulse Spreading (PS[1:0])**			
		00	**01**	**10**	**11**
		Slot Frequency (Hz)			
7	1000000	1000000	1000000	1000000	1000000
8	500000	500000	500000	500000	1000000
9	250000	250000	250000	500000	1000000
10	125000	125000	250000	500000	1000000
11	62500	125000	250000	500000	1000000
12	31250	125000	250000	500000	1000000
13	15625	125000	250000	500000	1000000
14	7812.5	125000	250000	500000	1000000
15	3906.25	125000	250000	500000	1000000
16	1953.125	125000	250000	500000	1000000

14.2.2.2 – Alternate PWM Output

Table 14-5 shows the mapping of each PWM Output. The PWM outputs PW0 to PW7 are also multiplexed with the DAC output pins. The DS4830A provides the option to select these alternate locations for PW0 to PW7 outputs if PWM functionality is required along with DAC outputs. When the ALT_LOC is set to '1' during PWM configuration for a PWM output, the PWM output will be available on this alternate pin. See Table 14-3 for details.

Table 14-5: Alternate PWM Output

PWM OUTPUT PIN	DS4830A PIN NUMBER WHEN ALT_LOC = 0	GPIO PIN	DS4830A PIN NUMBER WHEN ALT_LOC = 1	GPIO PIN
PW0	32	P0.4	4	P2.0
PW1	33	P0.5	6	P2.1
PW2	34	P6.5	12	P2.2
PW3	35	P1.5	13	P2.3
PW4	36	P1.6	24	P1.0
PW5	37	P1.7	25	P1.3
PW6	38	P6.6	26	P1.1
PW7	40	P2.7	27	P1.2
PW8	30	P0.6	30	P0.6
PW9	29	P0.7	29	P0.7

14.2.3 – PWM DELAY Register (PWMDLYn)

The Delay Register is used to provide a delay when starting the PWM output. By controlling the starting time for each individual PWM channel, multiphase operation can be achieved.

The number of bits used to program the Delay depends on the resolution programmed in the PWMCFG SFR. For 12 bits of resolution, the Delay is the lower 12 bits of the PWMDATA register. However if only 7 bits of resolution is selected, only the lower 7 bits are used to control the Delay of the corresponding PWM Channel. For example if 8-bit resolution is selected, the maximum delay programmed is limited to 255 (only lower 8 bits are considered).

The Delay resolution also depends on the selected Pulse spreading for the corresponding channel. The maximum delay is scaled correspondingly. With 10 bits of resolution and 4-slot pulse spreading (PS[1:0] = 2), the maximum delay programmed is limited to 1024/4 = 256.

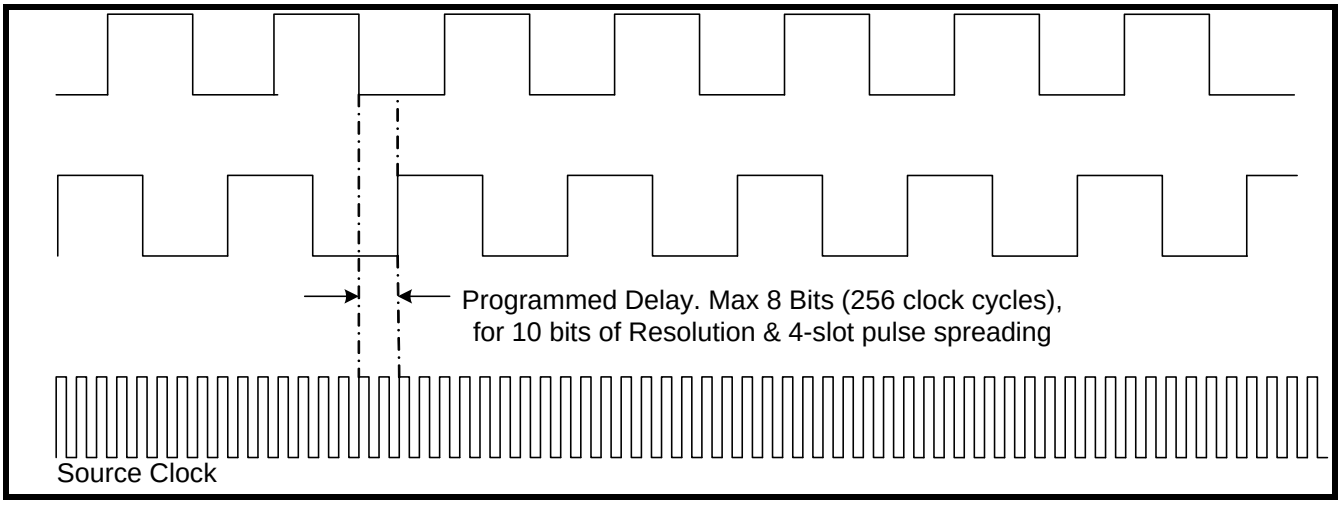


Figure 14-6: PWM Delay Operation without Pulse Spreading

14.2.3.1 – PWM DELAY with PWMSYNC SFR

The PWM channels to be synchronized must have the same configurations (Resolution, Pulse Spreading option, Clock source etc.). The delays on the two channels can be different. After the synchronization, the programmed delay is maintained as shown in Figure 14-7.

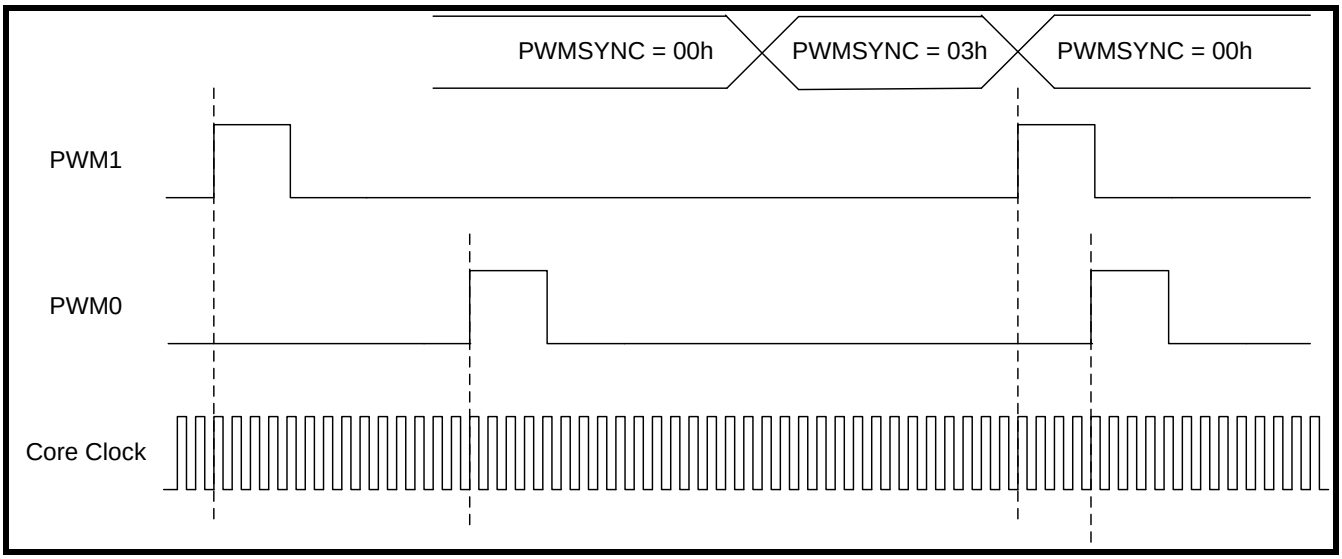


Figure 14-7: PWM Output Synchronization with 4 Clocks Delay

14.3 – PWM Output Register Descriptions

The DS4830A PWM controller has three SFRs. These are PWM Control Register PWMCN, PWM Data Register PWMDATA and PWM Synchronization Register PWMSYNC. The PWMCN configures and controls the various PWM operations. The PWMDATA register configures various PWM configurations and the PWMSYNC is used in PWM synchronization operation. The PWMCN, PWMDATA and PWMSYNC registers are cleared on POR only.

14.3.1 – PWM Control Register (PWMCN)

The PWMCN register is used to setup and start the PWM Output. To avoid undesired operation, the user should **not** modify the "Reserved" bits in the PWMCN registers.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	M_EN	-	-	-	UPDATE	PWM_SEL[3:0]				-	-	REG_SEL[1:0]	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	rw	r	r	r	rw	rw	rw	rw	rw	r	r	rw	rw

BIT	NAME	DESCRIPTION
15:13	-	**Reserved.** The user should not write to these bits.
12	M_EN	**Master Enable.** This is the master enable bit for all PWM channels. All the PWM channels will be enabled only after this bit is set to '1'. This bit should be set to '1', after configuring all local registers of all the required PWM channels.
11:9	-	**Reserved.** The user should not write to these bits.
8	UPDATE	**Update.** When this bit is set to '1', the duty cycle of all PWM channels are updated simultaneously. Writing a new value in the Duty Cycle register will not reflect in the PWM output until UPDATE is set to '1'. Once set, this bit will automatically clear after one core clock.
7:4	PWM_SEL[3:0]	**PWM Channel Select.** These bits select one of the 10 PWM channels for read or write to its local registers. These bits are used with REG_SEL[1:0] and provide access to 30 PWM local registers (3 local registers per channel). PWM_SEL auto increments after each read or write operation to PWMDATA register. DCYC = Duty Cycle Register PWMCFG = PWM Configuration Register PWMDLY = Delay Setting Register
3:2	-	**Reserved.** The user should not write to these bits.
1:0	REG_SEL[1:0]	**Register Select**. These bits are used to select one of three local registers of the selected PWM channel (selected by PWM_SEL[3:0] bits).

PWM_SEL	REG_SEL		
	00b	01b	1xb
0000b	DCYC0	PWMCFG0	PWMDLY0
0001b	DCYC1	PWMCFG1	PWMDLY1
0010b	DCYC2	PWMCFG2	PWMDLY2
0011b	DCYC3	PWMCFG3	PWMDLY3
0100b	DCYC4	PWMCFG4	PWMDLY4
0101b	DCYC5	PWMCFG5	PWMDLY5
0110b	DCYC6	PWMCFG6	PWMDLY6
0111b	DCYC7	PWMCFG7	PWMDLY7
1000b	DCYC8	PWMCFG8	PWMDLY8
1001b	DCYC9	PWMCFG9	PWMDLY9
>1001b	RESERVED	RESERVED	RESERVED

REG_SEL	Local Register Selected
00b	Duty Cycle Register (DCYCn)
01b	PWM Configuration Register (PWMCFGn)
1xb	Delay Setting Register (PWMDLYn)

14.3.2 – PWM Data Register (PWMDATA)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	PWMDATA[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:0	PWMDATA[15:0]	**PWM Data**. The PWM Data Register is used for configurations for various PWM channels. It is used to read or write various PWM local registers which are pointed by combinations of REG_SEL[1:0] and PWM_SEL[3:0] bits in the PWMCN register.

The PWMDATA Register is used to configure the local registers for each PWM channel. PWM channel is selected by PWM_SEL[3:0] bits in the PWMCN register. Individual local registers for a channel are selected by REG_SEL[1:0] bits in the PWMCN register. See below for the local register configurations.

14.3.2.1 – Local Register DCYCn

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	PWMDATA[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

PWMCN REG_SEL = 00b PWMDATA[15:0] → DCYCn[15:0]		
BIT	**NAME**	**DESCRIPTION**
15:0	DCYCn[15:0]	**Duty Cycle Register.** When REG_SEL[1:0] in the PWMCN SFR is set to 00b, the PWMDATA register points to the Duty Cycle Register of the PWM channel selected by PWM_SEL[3:0] bits in the PWMCN register. The number of bits used to program the Duty Cycle depends on the resolution programmed in the PWMCFG register. For 16 bits of resolution, the Duty cycle the complete 16 bits of the PWMDATA register. However if only 7 bits of resolution is selected, only the lower 7 bits are used. Example: If PWM_SEL[3:0] = 0101b and REG_SEL[1:0] = 00b, then the PWMDATA register points to the Duty Cycle Register of the PWM Channel 5. A read or write to/from PWMDATA register will read or write from the Duty Cycle Register of PWM Channel 5. If the resolution of channel 5 is set to 9 bits, only DCYCn[8:0] will be used for programming the Duty Cycle.

14.3.2.2 – Local Register PWMCFGn (through PWMDATA [15:0]

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	INV	-	ALT_ LOC	PWMEN	-		CLK_SEL		-		PS1	PS0	RES[3:0]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

PWMCN REG_SEL = 01b
PWMDATA[15:0] → PWMCFGn[15:0]

BIT	NAME	DESCRIPTION
15	INV	**Invert PWM Output.** When this bit is set to '1', PWM output is inverted for the selected PWM channel (determined by the PWM_SEL[3:0] bits).
14	-	**Reserved.** The user should not write to this bit.
13	ALT_LOC	**Alternate Location:** PWM outputs at channels 0 to 7 are multiplexed with the DAC outputs. By default, the PWM outputs appear at the DAC outputs. When ALT_LOC bit is set to '1', the PWM outputs will appear at the alternate location (See Table 14-3 for details).
12	PWMEN	**Local Enable:** Setting this bit to '1' will enable the individual PWM channel. PWM operation will be enabled only when both local enable and the Master Enable M_EN in PWMCN are enabled. Setting this bit to '0' will disable the individual PWM channel.
11:10	-	**Reserved.** The user should not write to these bits.
9:8	CLK_SEL[1:0]	**Clock Select**. These bits select the PWM clock for the selected PWM channel (which is selected by PWM_SEL[3:0] bits). **CLK_SEL** / **PWM Clock Source** 00b – Core Clock 01b – Peripheral Clock 1xb – External Clock The external clock range is 20MHz to 133MHz.
7:6	-	**Reserved.** The user should not write to these bits.
5:4	PS[1:0]	**Pulse Spreading:** These bits enable pulse spreading. The number of slots in a PWM frame is defined by these bits along with resolution. **PS[1:0]** / **Pulse spreading** 00b – Resolution / 2^{10} 01b – Resolution / 2^{9} 10b – Resolution / 2^{8} 11b – Resolution / 2^{7}
3:0	RES[3:0]	**Resolution Select**. These bits are used to configure PWM resolution (in bits) for selected PWM channel (which is selected by PWM_SEL[3:0] bits). The PWM Frame frequency is determined by the clock Frequency programmed and the resolution selected. $PWM\ Frame\ Frequency = \frac{PWM\ Clock\ Frequency}{2^N}$ Where N is the selected resolution. **RES [3:0]** / **PWM Resolution bits** 0000b – 7 0001b – 8 0010b – 9 0011b – 10 0100b – 11 0101b – 12 0110b – 13 0111b – 14 1000b – 15 >=1001b – 16

14.3.2.3 – Local Register PWMDLYn

PWMCN REG_SEL = 1xb PWMDATA[15:0] → PWMDLY[15:0]		
BIT	**NAME**	**DESCRIPTION**
15:0	PWMDLYn[15:0]	**Delay Setting Register.** When REG_SEL[1:0] is set to 1xb, the PWMDATA register points to the Delay Setting Register of PWM channel selected by PWM_SEL[3:0] bits in the PWMCN register. The Delay Setting Register is a 16-bit register, which is used for providing starting delay. Using this Delay Setting Register multiphase operation can be configured. If PWM_SEL[3:0] = 0101b and REG_SEL[1:0] = 1xb, then the PWMDATA register points to the Delay Register of the PWM Channel 5. A read or write to/from the PWMDATA register will read or write from the Duty Setting Register of PWM Channel 5. The Delay Setting Register is 16-bit register but number of bits is used by the PWM controller depends upon the selected resolution for given PWM channel. For Example, if resolution is 9- bits then only lower 9 bits PWMDLYn[8:0] are used in PWM operation and upper bits 10-15 bits will be ignored..

14.3.3 – PWM Synchronization Register (PWMSYNC)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	PWMSYNC[9:0]									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:10	-	**Reserved.** The user should not write to these bits. These bits are ignored by the PWM controller.
9:0	PWMSYNC[9:0]	**PWM Synchronization**. This register is used to provide synchronization among different PWM channels. Each bit of this register corresponds to a PWM channel. Setting any bit of this register will restart corresponding PWM channel. After a write to this register, it is cleared to 0x0000 on the next core cycle. For Example: When 0x0005 is written on the PWMSYNC register, PWM channel 0 and 2 will restart after current PWM counter (internal register) is over. This feature is used to bring different PWM channels in phase if they have the same PWM configurations (Resolution, Clock source, and Pulse spreading configuration, etc.). The Delay Register settings can be different for the PWM channels to be synchronized, and the settings are retained after the synchronization. See Figures 14.2 and 14.7 for details.

14.4 – PWM Output Code Examples

14.4.1 – 9-Bit PWM Output and Pulse Spreading (PS[1:0]= 11, 1-Slot) with Core Clock

Creating a 25% duty cycle with ~20kHz frequency PWM output at Channel 0 (default location):

```
//PWM Configuration for PW0

PWMCN = 0x0000;             //Channel 0, Duty cycle
PWMDATA = 0x007F;           //25% Duty cycle for 9-bit resolution

PWMCN = 0x0001;             //Channel 0, Config register
PWMDATA = 0x1002;           //Default location, 9-bit resolution with core clock with 1-slot pulse spreading, PWMEN

PWMCN = 0x0003;             //Channel 0, Delay Configuration
PWMDATA = 0x0000;           //No delay

PWMCN_bit.UPDATE = 1;    //Update PWM duty cycle
PWMCN_bit.M_EN = 1;         //Master Enable
```

14.4.2 – 9-Bit PWM Output and Pulse Spreading (PS[1:0]= 11, 4-Slots) with Core Clock

Creating a 40% duty cycle with 78.125kHz frequency PWM output at Channel 1 (default location):

```
//PWM Configuration for PW1

PWMCN = 0x0010;             //Channel1 Duty cycle
PWMDATA = 0x00CD;       //40% Duty cycle for 9-bit resolution with core clock

PWMCN = 0x0011;             //Channel 1Config register
PWMDATA = 0x1032;         //Default location, 9-bit resolution with core clock with 4-slot pulse spreading, PWMEN

PWMCN = 0x0013;             //Channel 1 Delay Configuration
PWMDATA = 0x0000;         //No delay

PWMCN_bit.UPDATE = 1; //Update PWM duty cycle
PWMCN_bit.M_EN = 1;       //Master Enable
```

14.4.3 – 12-Bit PWM Output and Pulse Spreading (PS[1:0]= 01, 8-Slots) with Alternate Location and Peripheral Clock

Creating a 25% duty cycle with 156.25kHz frequency PWM output at Channel 1 (alternate location) with Peripheral Clock:

```
//PWM Configuration for PW1

PWMCN = 0x0010;             //Channel 1 Duty cycle
PWMDATA = 0x03FF;         //25% Duty cycle for 12-bit resolution

PWMCN = 0x0011;             //Channel 1 Config register
PWMDATA = 0x3215;       //Alternate location (PW1, Port 2.1), 12-bit resolution with peripheral clock with 8-slot pulse spreading, PWMEN

PWMCN = 0x0013;             //Channel 1 Delay Configuration
PWMDATA = 0x0000;         //No delay

PWMCN_bit.UPDATE = 1;   //Update PWM duty cycle
PWMCN_bit.M_EN = 1;       //Master Enable
```

SECTION 15 – GENERAL-PURPOSE INPUT/OUTPUT (GPIO) PINS

15.1 – Overview

The DS4830A provides general-purpose input/output (GPIO) functionality on 31 pins. In addition to the GPIO functionality, each of these pins is multiplexed with at least one other function, which is classified as "Special Function."

Special functions override the GPIO register settings of the port pin when they are enabled. Once the special function takes control, normal control of the port pin is lost until the special function is disabled.

Table 15-1 details all of the GPIO pins as well as what other functions are multiplexed with each pin. With the exception of a few pins which are described further in detail later, the GPIO pins operate as shown in the GPIO Pin Block Diagram, Figure 15-1. Some of the features of these GPIO pins are:

- CMOS output drivers
- Schmitt trigger inputs
- Optional weak pullup to V_{DD} when operating in input mode

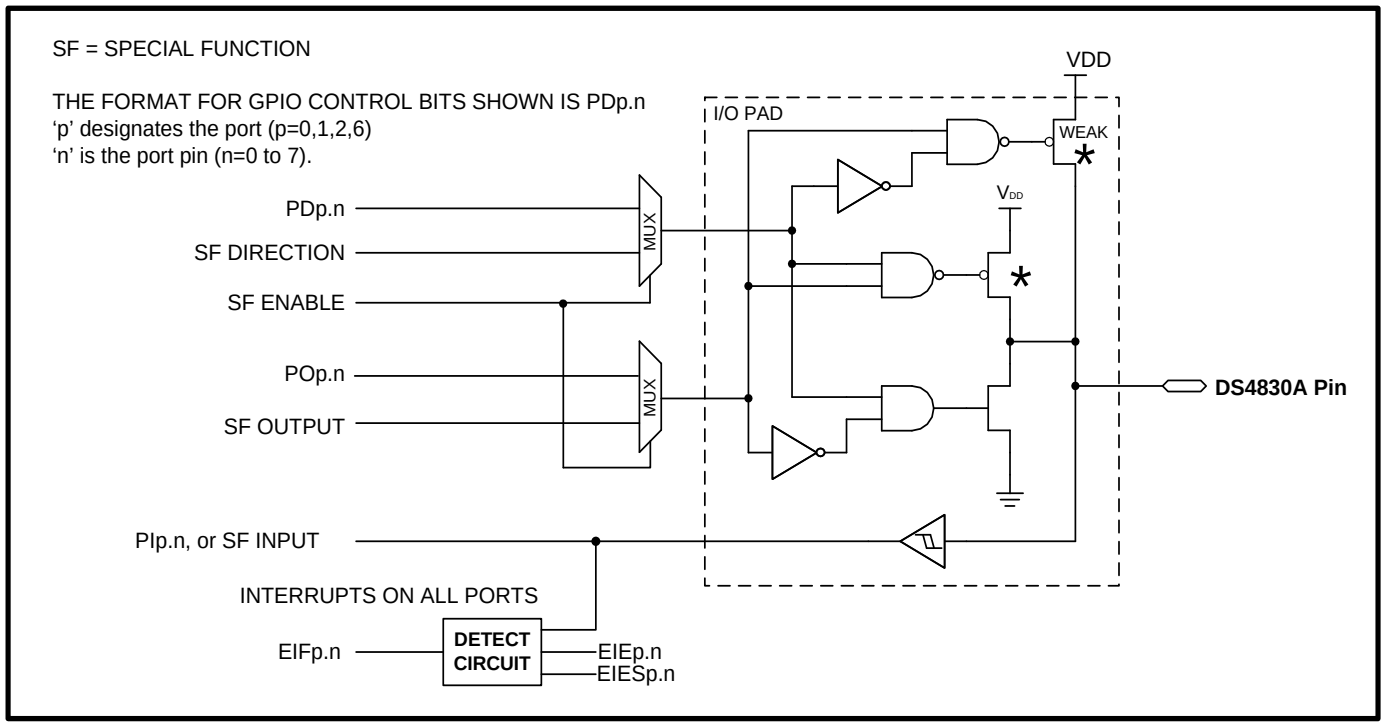


Figure 15-1: GPIO Pin Block Diagram

Table 15-1: GPIO Pins and Multiplexed Functions

Port Index	Pin Name	Pin No.	Default Function	Special Function-1	Special function-1 enable	Special Function-2	Special function-2 enable	Special Function-3	Special function-3 enable	Special function-4	Special function-4 enable
P0.0	GP12	19	GPIO	ADC-S12	PINSEL.12 = 1 & ADDATA.DIFF = 0	ADC-SHP1	SHCN.SMP_HLD1 = 1	ADC-D6P	PINSEL.12 = 1 & ADDATA.DIFF =1	-	-
P0.1	GP13	20	GPIO	ADC-S13	PINSEL.13 = 1 & ADDATA.DIFF = 0	ADC-SHN1	SHCN.SMP_HLD1 = 1	ADC-D6N	PINSEL.13 = 1 & ADDATA.DIFF =1	-	-
P0.2	GP14	21	GPIO	ADC-S14	PINSEL.14 = 1 & ADDATA.DIFF = 0	ADC-D7P	PINSEL.14 = 1 & ADDATA.DIFF =1	SHEN1	SENR.INT_TRIG_EN1 = 0	-	-
P0.3	GP15	22	GPIO	ADC-S15	PINSEL.15 = 1 & ADDATA.DIFF = 0	ADC-D7N	PINSEL.15 = 1 & ADDATA.DIFF =1	-	-	-	-
P0.4	DACPW0	32	GPIO	DAC0	DACCFG.CFG0 = 01b or 10b	PW0	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1	-	-	-	-
P0.5	DACPW1	33	GPIO	DAC1	DACCFG.CFG1 = 01b or 10b	PW1	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1	-	-	-	-
P0.6	PW8	30	GPIO	PW8	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1	-	-	-	-	-	-
P0.7	PW9	29	GPIO	PW9	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1	-	-	-	-	-	-
P1.0	MSDIO	24	GPIO	3W Data	TWR.TWEN = 1	I2CM-SDA	I2CCN_M.I2CEN=1	SPIM-DO	SPICN_M.SPIEN=1	PW4	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1 & PWMCFG.ALT_LOC = 1
P1.1	MSCL	26	GPIO	3W Clock	TWR.TWEN = 1	I2CM-CLK	I2CCN_M.I2CEN=1	SPIM-CL	SPICN_M.SPIEN=1	PW6	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1 & PWMCFG.ALT_LOC = 1
P1.2	MCS	27	GPIO	3W CS	TWR.TWEN = 1	-	-	SPIM-CS	SPICN_M.SPIEN=1	PW7	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1 & PWMCFG.ALT_LOC = 1
P1.3	MSDI	25	GPIO	-	-	-	-	SPIM-DI	SPICN_M.SPIEN=1	PW5	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1 & PWMCFG.ALT_LOC = 1
P1.4	REFINB	39	GPIO	ADC-REFB	DACCFG.CFG4-7* = 01b (any one or more DACs)	-	-	-	-	-	-
P1.5	DACPW3	35	GPIO	DAC3	DACCFG.CFG3 = 01b or 10b	PW3	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1	-	-	-	-
P1.6	DACPW4	36	GPIO	DAC4	DACCFG.CFG4 = 01b or 10b	PW4	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1	I2CM-SDA-ALT	i2CCN_M.I2CM_ALT = 1	-	-
P1.7	DACPW5	37	GPIO	DAC5	DACCFG.CFG5 = 01b or 10b	PW5	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1	I2CM-SCL-ALT	i2CCN_M.I2CM_ALT = 1	-	-
P2.0	GP0	4	GPIO	ADC-S0	PINSEL.0 = 1 & ADDATA.DIFF = 0	ADC-D0P	PINSEL.0 = 1 & ADDATA.DIFF =1	PW0	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1 & PWMCFG.ALT_LOC = 1	-	-
P2.1	GP1	6	GPIO	ADC-S1	PINSEL.1 = 1 & ADDATA.DIFF = 0	ADC-D0N	PINSEL.1 = 1 & ADDATA.DIFF =1	PW1	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1 & PWMCFG.ALT_LOC = 1	REFOUT	-
P2.2	GP6	12	GPIO	ADC-S6	PINSEL.6 = 1 & ADDATA.DIFF = 0	ADC-D3P	PINSEL.6 = 1 & ADDATA.DIFF =1	PW2	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1 & PWMCFG.ALT_LOC = 1	SDO	SPICN_S.SPIEN=1
P2.3	GP7	13	GPIO	ADC-S7	PINSEL.7 = 1 & ADDATA.DIFF = 0	ADC-D3N	PINSEL.7= 1 & ADDATA.DIFF =1	PW3	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1 & PWMCFG.ALT_LOC = 1	SCS	SPICN_S.SPIEN=1
P2.4	GP8	14	GPIO	ADC-S8	PINSEL.8 = 1 & ADDATA.DIFF = 0	ADC-D4P	PINSEL.8 = 1 & ADDATA.DIFF =1	-	-	-	-
P2.5	GP9	15	GPIO	ADC-S9	PINSEL.9 = 1 & ADDATA.DIFF = 0	ADC-D4N	PINSEL.9 = 1 & ADDATA.DIFF =1	-	-	-	-
P2.6	REFINA	31	GPIO	ADC-REFA	DACCFG.CFG0-3* = 01b (any one or more DACs)	-	-	-	-	-	-
P2.7	DACPW7	40	GPIO	DAC7	DACCFG.CFG7 = 01b or 10b	PW7	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1	-	-		
P6.0	GP4	10	TCK	ADC-S4	PINSEL.4 = 1 & ADDATA.DIFF = 0	ADC-D2P	PINSEL.4 = 1 & ADDATA.DIFF =1	-	-	-	-
P6.1	GP5	11	TDI	ADC-S5	PINSEL.5 = 1 & ADDATA.DIFF = 0	ADC-D2N	PINSEL.5 = 1 & ADDATA.DIFF =1	-	-	-	-
P6.2	GP10	17	TMS	ADC-S10	PINSEL.10 = 1 & ADDATA.DIFF = 0	ADC-D5P	PINSEL.10 = 1 & ADDATA.DIFF =1	-	-	-	-
P6.3	GP11	18	TDO	ADC-S11	PINSEL.11 = 1 & ADDATA.DIFF = 0	ADC-D5N	PINSEL.11 = 1 & ADDATA.DIFF =1	-	-	-	-
P6.4	SHEN0	23	GPIO	SHEN0	SENR.INT_TRIG_EN0 = 1	-	-	-	-	-	-
P6.5	DACPW2	34	GPIO	DAC2	DACCFG.CFG2 = 01b or 10b	PW2	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1	CLKIN	+	-	-
P6.6	DACPW6	38	GPIO	DAC6	DACCFG.CFG6 = 01b or 10b	PW6	PWMCFG.PWMEN = 1 & PWMCN.M_EN = 1	-	-	-	-

Notes:

- TCK: Test Access Port (TAP) Clock
- TDI: Test Access Port (TAP) Data Input
- TMS: Test Access Port (TAP) Mode Select
- TDO: Test Access Port (TAP) Data Output
- * One or more DACs should be enabled.
- + External Clock is enabled when the external clock source is selected by one or more peripherals among timers, PWM and Sample and Hold.

From a software perspective, each of the GPIO ports (Port0, Port1, Port2, and Port6) has six Special Function Registers (POp, PIp, PDp, EIFp, EIEp and EIESp where p=0, 1, 2, or 6). Each GPIO port is designed to provide

programming flexibility for any application. The associated registers and their module addresses are listed in Table 15-2. The user should not write to any reserved bits as this may cause undesired behavior.

Table 15-2: GPIO Registers

REGISTER	FUNCTION	PORT 0	PORT 1	PORT 2	PORT 6
POp	Port Output Register	M0[02h]	M0[01h]	M0[00h]	M1[03h]
PIp	Port Input Register	M0[0Ah]	M0[09h]	M0[08h]	M1[08h]
PDp	Port Direction Register	M0[12h]	M0[11h]	M0[10h]	M1[11h]
EIFp	Port External Interrupt Flag Register	M0[05h]	M0[04h]	M0[03h]	M1[06h]
EIEp	Port External Interrupt Enable Register	M0[0Fh]	M0[0Eh]	M0[0Dh]	M1[07h]
EIESp	Port External Interrupt Edge Select Register	M0[15h]	M0[14h]	M0[13h]	M1[10h]

15.2 – GPIO Port Register Descriptions

The DS4830A has 4 ports P0, P1, P2 and P6. Each port has 8 pins (exception is P6 which has 7 pin only). The GPIO operation is to control/monitor through PDp, POp and PIp (p = 0, 1, 2 and 6). These ports are multiplexed with various functions like ADC, DAC, Sample and Hold, PWM, I^2C, 3-Wire, SPI etc. Additionally, these ports also provide GPIO interrupts on all of the pins. A GPIO interrupt can be generated when the pin is being operated as a GPIO, or a special. Three additional registers, EIFp, EIEp, and EIESp are used to control the GPIO interrupts.

On device reset, the TAP port is active, allowing for in-circuit debugging and programming. The JTAG is active by default on Port6[3:0] and it is disabled when SC.TAP bit is set to '0'. Enabled special functions operate on the JTAG ports only if SC.TAP bit is set to '0'. Port 6 also provides GPIO interrupts on all of the pins. The GPIO works only when SC.TAP = 0. A GPIO interrupt can be generated when the pin is being operated as a GPIO, or a special or alternate function. Three additional registers, EIF6, EIE6, and EIES6 are used to control the GPIO interrupts. Port6.7 is not present in the Port6.

15.2.1 – GPIO Direction Register Port (PD0, PD1, PD2, and PD6)

Bit #	7	6	5	4	3	2	1	0
Name	PDp_7	PDp_6	PDp_5	PDp_4	PDp_3	PDp_2	PDp_1	PDp_0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

PDp is an 8-bit register used to determine the direction of the pins when they are used as GPIO pins. Each pin is independently controlled by its direction bit. When PDp.n (p = 0 to 7, n = 0 to 7) is set to '1', the pin is an output; data in the POp.n bit will be driven on the pin. When PDp.n is cleared to '0', the pin is an input and allows an external signal to drive the pin. Note that each port pin has a weak pullup circuit when functioning as an input. The P channel pullup transistor is controlled by the POp.n bit. If POp.n is set to '1', the corresponding weak pullup is turned on, if the POp.n bit is cleared to '0', the weak pullup is turned off and the pin's input is high-impedance.

15.2.2 – GPIO Output Register Port (PO0, PO1, PO2, and PO6)

Bit #	7	6	5	4	3	2	1	0
Name	POp_7	POp_6	POp_5	POp_4	POp_3	POp_2	POp_1	POp_0
Reset*	1	1	1	1	1	1	1	1
Access	rw	rw	rw	rw	rw	rw	rw	rw

*GPIO which are shared with DAC ports has POp.n = 0 on reset.

POp is an 8-bit register that controls the output data of a GPIO pin. If the pin is setup to be an output (PDp.n = 1), the data in POp.n will be output on the pin. If the pin is set as an input (PDp.n = 0), setting POp.n to a '1' enables a p-channel weak pullup, otherwise the pin's input is high impedance.
When the Port pins are operating as PWM pins, the data in POp will not affect PWM operation. Changing the direction of the pin does not change the data content of POp.n.

15.2.3 – GPIO Input Register for Port (PI0, PI1, PI2, and PI6)

Bit #	7	6	5	4	3	2	1	0
Name	PIp_7	PIp_6	PIp_5	PIp_4	PIp_3	PIp_2	PIp_1	PIp_0
Reset	s	s	s	s	s	s	s	s
Access	r	r	r	r	r	r	r	r

PIp is an 8-bit register which contains the data that is applied to the GPIO pins. The PIp input register contains valid input data even when the pin is not operating as a GPIO. The reset value for this register is dependent on the logical states applied to the pins. Note that each pin has a weak pullup circuit when functioning as an input and the P channel pullup transistor is controlled by the POp.n bit.

15.2.4 – GPIO Port External Interrupt Edge Select Register (EIES0, EIES1, EIES2, and EIES6)

Bit #	7	6	5	4	3	2	1	0
Name	IESPp_7	IESPp_6	IESPp_5	IESPp_4	IESPp_3	IESPp_2	IESPp_1	IESPp_0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

The EIESp register sets the interrupt edge select to trigger an interrupt on either a rising or falling edge. Setting the IESPp_n bits to 0 will trigger the corresponding interrupt on a positive edge. When these bits are set to a 1, the interrupt will be on a negative edge.

15.2.5 – GPIO Port External Interrupt Flag Register (EIF0, EIF1, EIF2, and EIF6)

Bit #	7	6	5	4	3	2	1	0
Name	IFPp_7	IFPp_6	IFPp_5	IFPp_4	IFPp_3	IFPp_2	IFPp_1	IFPp_0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

These bits are set when a negative edge (IESPp.n = 1) or a positive edge (IESPp.n = 0) is detected on the Pp.n pin. Setting any of the bits to 1 will generate an interrupt to the CPU if the corresponding interrupt is enabled. These bits will remain set until cleared by software or a reset. These bits must be cleared by software before exiting the interrupt service routine or another interrupt will be generated as long as the bit remains set.

15.2.6 – GPIO Port External Interrupt Enable Register (EIE0, EIE1, EIE2, and EIE6)

Bit #	7	6	5	4	3	2	1	0
Name	IEPp_7	IEPp_6	IEPp_5	IEPp_4	IEPp_3	IEPp_2	IEPp_1	IEPp_0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Setting any of these bits to 1 will enable the corresponding external interrupt. Clearing any of the bits to 0 will disable the corresponding interrupt function.

15.3 – GPIO Code Example

15.3.1 – GPIO Pin as Output

```
//set pin 6.4 as a high output
PD6 |= 0x10;          //set direction PD6.4 to 1 for an output
PO6 |= 0x10;          //set the output PO6.4 high
```

15.3.2 – GPIO High-Impedance Input

```
//set pin 6.4 as a high-impedance input
PD6 &= ~0x10;         //set direction PD6.4 to 0 for input
PO6 &= ~0x10;         //set PO6.4 low to disable weak pullup
```

15.3.3 – GPIO Weak Pullup Input

```
//enable the pin 6.4 weak pullup
PD6 &= ~0x10;         //set direction PD6.4 to 0 for input
PO6 |= 0x10;          //set PO6.4 high to enable weak pullup
```

15.3.4 – GPIO Open-Drain Output

```
//configure pin6.4 as port 'Open Drain'
PO6 &= ~0x10;         // set the PO6.4 to the logic '0'
PD6 |= 0x10;          // this will configure P6.4 as output and drive logic '0'
PD6 &= ~0x10;         // this will configure P6.4 as input with high impedance.
```

In summary, the GPIO output can be set to the '**Open Drain'** by doing the following method

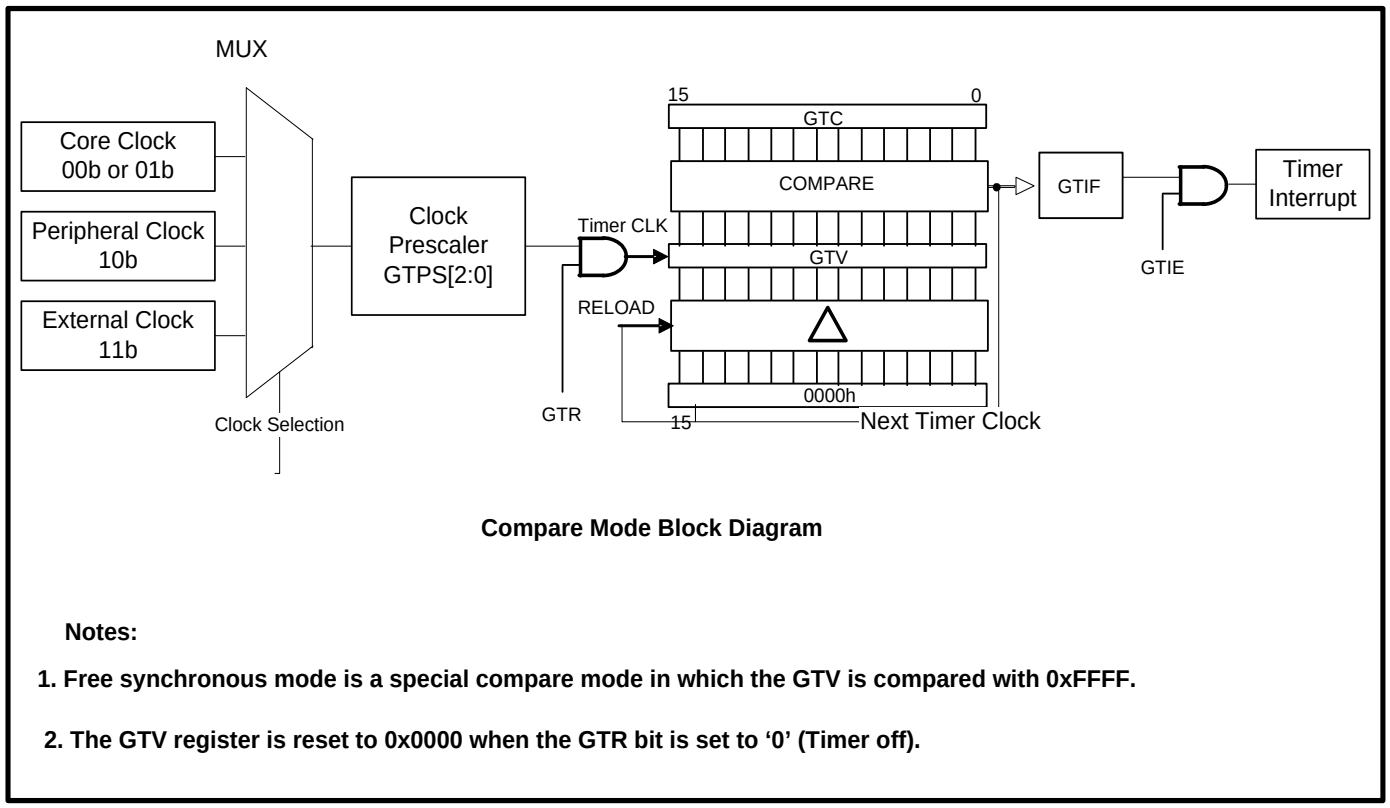
1. Set the POp.n to the logic '0'.
2. Toggle the direction register PDp.n between the input and output.

This causes the pin to alternate between logic '0' (PDp.n = 1) and 'high impedance' (PDp.n = 0).

SECTION 16 – GENERAL-PURPOSE TIMERS

The DS4830A has two identical 16-bit general-purpose timers. Each timer has the following,

- Two modes - Free synchronous and Compare
- Three Clock source selection options - Core clock, Peripheral clock and External clock
- 6 prescalers
- Interrupt feature in both modes.



Compare Mode Block Diagram

Notes:

1. Free synchronous mode is a special compare mode in which the GTV is compared with 0xFFFF.

2. The GTV register is reset to 0x0000 when the GTR bit is set to '0' (Timer off).

Figure 16-1: Timer Functional Diagram

16.1 – Detailed Description

The DS4830A has two 16-bit programmable timer modules. Each timer module supports input clock selection between Core, Peripheral and External clock sources. Each timer has two modes of operation i.e. free synchronous timer and compare mode. The timer is controlled by the General Timer Reset (GTR) bit in the General Timer Control Register (GTCN). When this bit is set to '1', it enables the timer and starts counting up. When this bit is set to '0', the timer is stopped. Each timer has six prescalers selection feature. Using various prescaler and input clock options, various timing loops can be generated.

16.1.1 – Timer Modes

Each timer has two modes of operation i.e. free synchronous timer and compare mode. The MODE bit in the GTCN register selects the timer mode. The 16-bit free synchronous mode is configured by setting the MODE bit to 0. When the Mode bit is set to '1', compare mode is configured.

In free synchronous mode, the timer module begins counting up from 0x0000. When the General Timer Value Register (GTV) value reaches to 0xFFFF, the GTIF interrupt flag is set to '1' which generates an interrupt if enabled, and the timer reloads the GTV register with 0x0000 at the next timer clock. The GTV register is a read only register and it resets to 0x0000 when the timer is stopped. (GTR = 0).

In compare mode, the timer module begins counting from 0x0000 and when the value in the GTV register matches the value in the General Timer Compare Register (GTC), the GTIF interrupt flag is set to '1 ' which generates an

interrupt if enabled. When the match occurs, the timer reloads the GTV register with 0x0000 at the next timer clock. In compare mode, the GTC register should be written first before setting the GTR bit.

16.1.2 – Clock Selection

There are three timer clock sources available in each timer module, core clock, peripheral clock and external clock. The peripheral clock is twice the core clock. The external clock can be between 20MHz to 133MHz. The timer clock source can be selected using CLK_SEL [1:0] bits in the GTCN register.

16.1.3 – Timer Clock Prescaler

Each timer has 6 different prescalers. The prescaler is selected by appropriately setting the GTPS [2:0] bits in the GTCN register. The prescaler divides the selected input clock by a value from 1 to 1024.

16.2 – Timer Register Descriptions

Each General Timer module has three independent SFR registers. These are GTCN, GTV and GTC. The General Timer Control Register GTCN controls the timer operation. The General Timer Value Register GTV is the Timer Value register and is incremented every timer clock when enabled. The General Timer Compare Register GTCx is used in the timer compare mode only. Timer 1 and 2 SFRs are located in module 0 and 3 respectively.

16.2.1 – General Timer Control Register (GTCN)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	GTR	MODE	CLK_SEL[1:0]		GTIE	-	-	-	GTIF	-	GTPS[2:0]		
Reset*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	rw	rw	rw	rw	rw	r	r	r	rw	r	rw	rw	rw

*These are default power on reset value.

BIT	NAME	DESCRIPTION
15:13	Reserved	**Reserved**. The user should write 0 to these bits.
12	GTR	**Timer Run Control.** Setting this bit to '1' will enable the timer. Clearing this bit to '0' will stop the timer and clear the GTV register.
11	MODE	**Timer Mode Select.** This bit selects the timer mode. When this bit is '0', free synchronous mode is selected. In this mode, the GTV register starts counting from 0x0000. When the GTV register reaches 0xFFFF, GTIF is set to '1' and the GTV register reloads to 0x0000 at the next timer clock. When the MODE bit is set to '1', compare mode is selected. In this mode, the GTV register starts counting from 0x0000. When the GTV register matches the value in the GTC register, GTIF is set to '1' and the GTV register reloads to 0x0000 at the next timer clock. **Note:** In the compare mode, the GTC register value should be set prior to write to the 'MODE' bit.
10:9	CLK_SEL[1:0]	**Timer Clock Select.** These bits select the timer clock source. **CLK_SEL** / **Clock Source** 0X / Core Clock 10 / Peripheral Clock 11 / External Clock* *The external clock range is 20MHz to 133MHz.
8	GTIE	**Timer Interrupt Enable.** Setting the GTIE bit to '1' causes an interrupt to be generated to the CPU when GTIF=1. Clearing this bit to '0' will not cause an interrupt when GTIF=1.
7:5	Reserved	**Reserved**. The user should write 0 to these bits.
4	GTIF	**Timer Matched Interrupt Flag.** This bit is set to '1' when 1. In free synchronous mode, the GTV register value reaches 0xFFFF. 2. In compare mode, the GTV register value matches the value in the GTC register. This flag generates an interrupt if the GTIE bit is enabled. This bit is cleared in software by writing '0'.
3	Reserved	**Reserved**. The user should write 0 to these bits.
2:0	GTPS[2:0]	**Timer Prescaler Select.** These bits configure the prescaler from the timer clock input to the timer. **Prescaler bits** / **Timer input clock** 000 / Timer Clock 001 / Timer Clock/4 010 / Timer Clock/16 011 / Timer Clock/64 100 / Timer Clock/256 101 / Timer Clock/1024 11X / Timer Clock

16.2.2 – General Timer Value Register (GTV1 and GTV2)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	GTV(1,2)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

16.2.3 – General Timer Compare Register (GTC1 and GTC2)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	GTC(1,2)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

SECTION 17 – SUPPLY VOLTAGE MONITOR (SVM)

The DS4830A provides feature to allow monitoring its power supply. The Supply Voltage Monitor (SVM) monitors the V_{DD} power supply and can alert the processor through an interrupt if V_{DD} falls below a programmable threshold.

The DS4830A provides the following power monitoring features:

- SVM compares V_{DD} against a programmable threshold from approximately 2.3V to 3.5V.
- Optional SVM interrupt can be triggered when V_{DD} drops below the programmed threshold.

The Supply Voltage Monitor is controlled by the peripheral register SVM. This register is located in Module 1, Index 9.

Supply Voltage Monitor Register (SVM)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	SVTH[3:0]				-	-	-	-	SVMI	SVMIE	SVMRDY	SVMEN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw*	rw*	rw*	rw*	r	r	r	rw	rw	rw	r	rw

*SVTH[3:0] bits can only be written when the SVM is not running (SVMEN=0)

BIT	NAME	DESCRIPTION
15:12	Reserved	Reserved. The user should write 0 to these bits.
11:8	SVTH[3:0]	**Supply Voltage Threshold Bits [3:0]**: These bits are used to select a user defined supply voltage threshold. If V_{DD} is below this threshold, the SVMI bit will be set and an interrupt can be generated if enabled. The threshold level can be adjusted from 2.3V to 3.5V in 0.1V increments. The default value is 00h (2.3V). Supply Voltage Monitor Threshold = 2.3V + SVTH[3:0] * 0.1V Note that the SVTH[3:0] bits can only be modified when SVMEN = 0. Writing to these bits is ignored if SVMEN = 1. The SVM thresholds 00h to 05h have no actual µs because these are lower than the DS4830A operating V_{DDMIN} range (which is 2.85V). In the upper side, setting 0Ch to 0Fh corresponds to 3.5V V_{DD}.
7:4	Reserved	Reserved. The user should write 0 to these bits.
3	SVMI	**Supply Voltage Monitor Interrupt:** This bit is set to '1' when the V_{DD} supply voltage falls below the threshold defined by SVTH[3:0]. If SVMIE = 1, setting this bit to 1 by either hardware or software triggers an interrupt. This bit must be cleared by software, but if V_{DD} is still below the threshold, the bit is immediately set again by hardware.
2	SVMIE	**Supply Voltage Monitor Interrupt Enable**: Setting this bit to 1 allows an interrupt to be generated (if not otherwise masked) when SVMI is set to 1. Clearing this bit to 0 disables the SVM interrupt.
1	SVMRDY	**Supply Voltage Monitor Ready:** This read-only status bit indicates whether the SVM is ready for use. 0 = The SVM is disabled (SVMEN = 0), or the SVM is in the process of powering up. 1 = The SVM is enabled and ready for use.
0	SVMEN	**Supply Voltage Monitor Enable:** Setting this bit to 1 enables the SVM and begins monitoring V_{DD} against the programmed (SVTH[3:0]) threshold. After SVMEN is set, SVMRDY will be set in approximately 20 us. Clearing this bit to 0 disables the SVM.

SECTION 18 – HARDWARE MULTIPLIER MODULE

The hardware multiplier module can be used by the DS4830A to support high-speed multiplications. The hardware multiplier module is equipped with two 16-bit operand registers, a 32-bit read-only result register, and an accumulator of 48-bit width. The multiplier can complete a 16-bit x 16-bit multiply-and-accumulate/subtract operation in a single cycle. The hardware multiplier module supports the following operations without interfering with the normal core functions:

- Signed or unsigned Multiply (16 bit x 16 bit)
- Signed or unsigned Multiply-Accumulate (16 bit x 16 bit)
- Signed or unsigned Multiply-Subtract (16 bit x 16 bit)
- Signed Multiply and Negate (16 bit x 16 bit)

18.1 – Hardware Multiplier Organization

The hardware multiplier consists of two 16-bit, parallel-load operand registers (MA, MB); a read-only result register formed by two parallel 16-bit registers (MC1R and MC0R); an accumulator, which is formed by three 16-bit parallel registers (MC2, MC1, and MC0); and a status/control register (MCNT). Figure 18-1 shows a block diagram of the hardware multiplier.

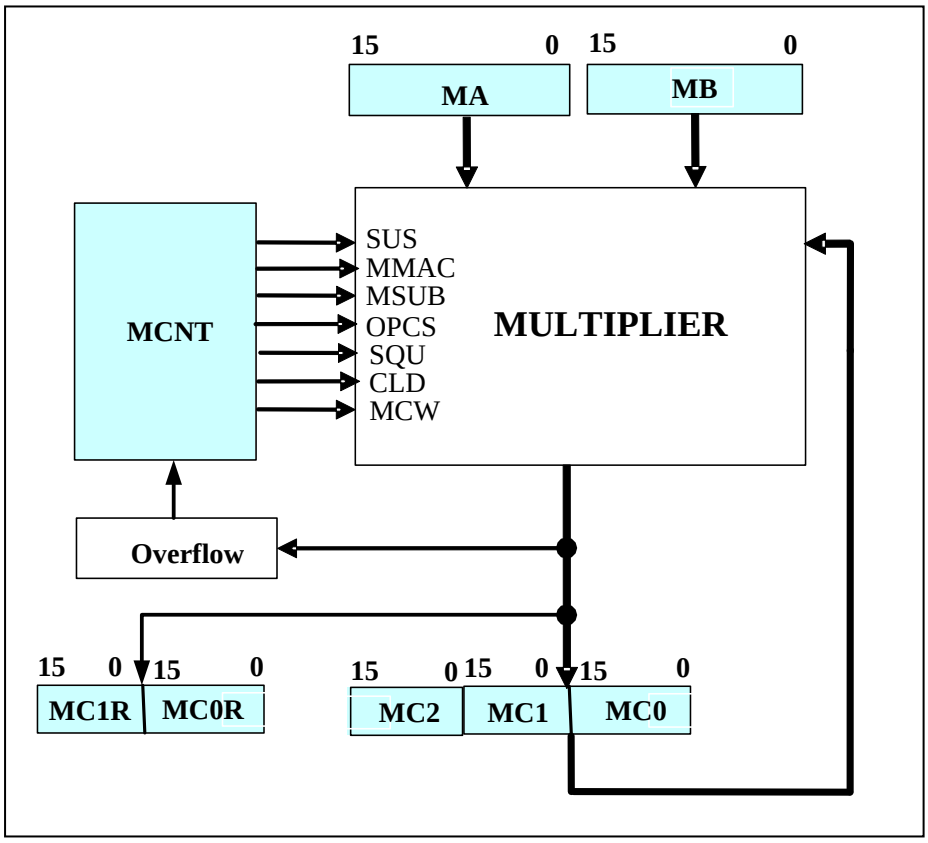


Figure 18-1: Multiplier Organization

18.2 – Hardware Multiplier Controls

The selection of operation to be performed by the multiplier is determined by four control bits in the MCNT register: SUS, MSUB, MMAC, and SQU. The number of operands that must be loaded to trigger the specified operation is dictated by the OPCS bit setting, except when the square function is enabled (SQU = 1). Enabling the square function implicitly defines that only a single operand (either MA or MB) needs to be loaded to trigger the square operation, independent of the OPCS bit setting. The MCNT register bits must be configured to select the desired operation and operand count prior to loading the operand(s) to trigger the multiplier operation. Any write to MCNT automatically resets the operand load counter of the multiplier, but does not affect the operand registers, unless such action is requested using the Clear Data Registers (CLD) control bit. Once the desired operation has been specified

via the MCNT register bits, loading the prescribed number of operands triggers the respective multiply, multiply-accumulate/subtract or multiply-negate operation.

18.3 – Register Output Selection

The Hardware Multiplier implements the MC Register Write Select (MCW) control bit so that writing of the result to the MC2:MC0 registers can be blocked to preserve the MC registers (accumulator). When the MCW bit is configured to logic 1, the result for the given operation is not written to the MC registers. When the MCW bit is configured to logic 0, the MC registers are updated with the result of the operation. The MC1R, MC0R read-only register pair are updated independent of the MCW bit setting. This register pair always reflects the output that would normally be placed in MC1:MC0, given that MCW = 1 or MMAC = 0. When MCW = 0 and MMAC = 1, the MC1R:MC0R content may not match the MC1:MC0 register content, but it will be predictable and may be useful in certain situations. See Table 18-1 for details.

18.3.1 – Signed-Unsigned Operand Selection

The operands can be either signed or unsigned numbers, but the data type must be defined by the user software via the Signed-Unsigned (SUS) bit prior to triggering the operation. For an unsigned operation, the Signed-Unsigned bit (SUS) in the MCNT register must be set to 1; for a signed operation, the SUS bit must be cleared to 0. The multiplier treats unsigned numbers as absolute magnitude. For a 16-bit positional binary number, this represents a value in the range 0 to 2^{16} - 1 (FFFFh). The signed number representation is a two's-complement value, where the most significant bit is defined as a sign bit. The range of a 16-bit two's-complement number is $-2^{(16-1)}$ (8000h) to $+2^{(16-1)}$ - 1 (7FFFh). The product of any signed operation will be sign extended before being stored or accumulated/subtracted into the MC registers. The SUS bit should always be configured to logic 0 (i.e., signed operands) for the multiply-negate operation. Attempting an unsigned multiply-negate operation results in incorrect results and setting of the OF bit. Modifying the operand data type selection via the SUS bit does not alter the contents of the MC registers. The MC registers are read/write accessible and can be modified by user code when necessary.

18.3.2 – Operand Count Selection

The OPCS bit allows selection of single operand or two operands operation for the multiply and multiply-accumulate/subtract operations. When the OPCS bit is cleared to 0, the multiply or multiply-accumulate/subtract operation established by the SUS, MSUB, and MMAC bits, is triggered once two operands are loaded-(MA and MB registers). When OPCS is set to 1, the operation commences once data is loaded to either MA or MB. The OPCS bit is ignored when the square operation is enabled (SQU), since loading of data to the MA or MB register actually writes to both registers.

18.4 – Hardware Multiplier Operations

The control bits, which specify data type (SUS), operand count (OPCS or SQU), and destination control (MCW), have already been described. However, there are two additional MCNT register bits that serve to define the Hardware Multiplier operation. The multiply-accumulate/subtract and multiply-negate operations are enabled by the Multiply-Accumulate Enable (MMAC) and Multiply Negate (MSUB) bits in the MCNT register. When the MMAC bit is set to 1, the multiplier performs a multiply-accumulate (if MSUB = 0) or a multiply-subtract (if MSUB = 1). If MMAC is configured to 0, the multiplier result is not accumulated or subtracted, but can be stored directly (if MSUB = 0) or negated (if MSUB = 1) before storage. The multiply-negate operation (MMAC = 0, MSUB = 1) is only allowable for signed data operands (SUS = 0). For unsigned multiply-accumulate/subtract operations, the OF bit is set when a carry-out/borrow-in from the most significant bit of the MC register occurs. For a signed two's-complement multiply-accumulate/subtract operations, the OF bit is set when the carry-out/borrow-in from the most significant magnitude position of the MC register is different from the carryout/ borrow-in of the sign position of the MC register. Since there is no overflow condition for multiply and multiply-negate operations, the OF bit is always cleared for these operations with one exception. The OF bit will be set to logic 1 if an unsigned multiply-negate (invalid operation) is requested. Table 18-1 shows the operations supported by the multiplier and associated MCNT control bit settings.

18.4.1 – Accessing the Multiplier

There are no restrictions on how quickly data is entered into the operand registers or the order of data entry. The only requirement to do a calculation is to perform the loading of MA and/or MB registers having specified data type and operation in the MCNT register. The multiplier keeps track of the writes to the MA and MB registers, and starts calculation immediately after the prescribed number of operands is loaded. If two operands are specified for the operation, the multiplier waits for the second operand to be loaded into the other operand register before starting the actual calculation. If for any reason software needs to reload the first operand, it should either reload that same operand register or use the CLD bit in the MCNT register to reinitialize the multiplier; otherwise, loading data to another operand register triggers the calculation. The CLD bit is a self-clearing bit that can be used for multiplier initialization. When it is set, it clears all data registers and the OF bit to zero and resets the multiplier operand write counter.

The specified hardware multiplier operation begins when the final operand(s) is loaded and will complete in a single cycle. The read-only MC1R, MC0R result registers can be accessed in the very next cycle unless accumulation/subtraction with MC2:0 is requested (MCW = 0 and MMAC = 1), in which case, one cycle is required so that stable data can be read. When MCW = 0, the MC2:0 registers always require one wait cycle before the operation result is accessible. The single wait cycle needed for updating the MC2:0 registers with a calculated result does not prevent initiating another calculation. Back-to-back operations can be triggered (independent of data type and operand count) without the need of wait state between the loadings of operands.

Table 18-1 Hardware Multiplier Operations

MCW:MSUB:MMAC	OPERATION	MC2	MC1	MC0	MC1R:MC0R	OF STATUS
000	Multiply	MA*MB			MA*MB	No
001	Multiply-Accumulate	MC+(MA*MB)			32lsbits of (MC+2*(MA*MB))	Yes
010	Multiply-Negate (SUS = 0 only)	-(MA*MB)			-(MA*MB)	No
011	Multiply-Subtract	MC-(MA*MB)			32lsbits of (MC-2*(MA*MB))	Yes
100	Multiply	MC2	MC1	MC0	MA*MB	No
101	Multiply-Accumulate	MC2	MC1	MC0	32lsbits of (MC+(MA*MB))	No
110	Multiply-Negate (SUS = 0 only)	MC2	MC1	MC0	-(MA*MB)	No
111	Multiply-Subtract	MC2	MC1	MC0	32lsbits of (MC-(MA*MB))	No

The DS4830A has two sets of internal MAC registers to allow interruptible MAC operation. The MACRSEL bit in the MACSEL register selects one of the MAC registers.

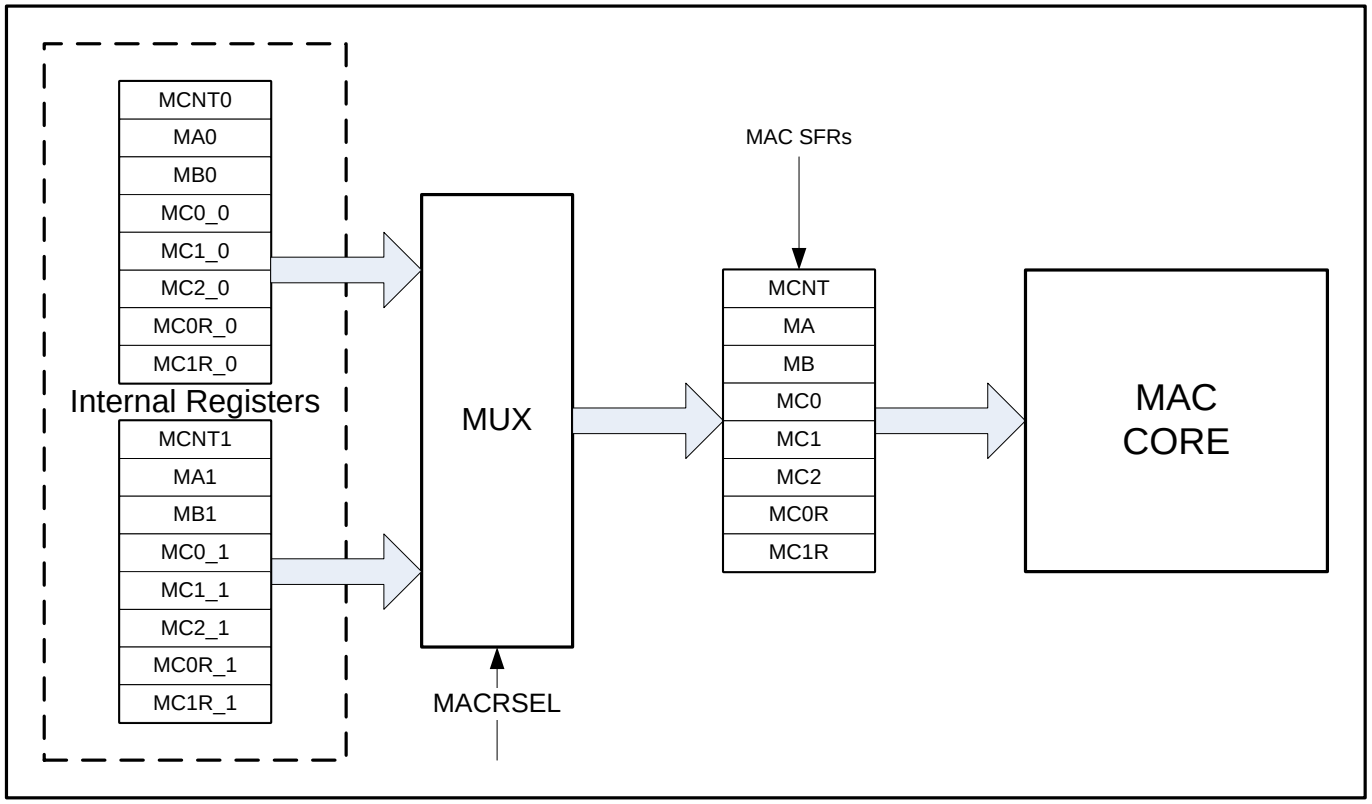


Figure 18-2: Dual MAC Registers Organization

18.5 – Hardware Multiplier Peripheral Registers

The hardware multiplier registers are detailed below. Addresses of registers are given as "M*x*[*yy*]" where *x* is the module number (from 0 to 5 decimal) and *yy* is the register index (from 00h to 1Fh hexadecimal).

Table 18-2: Hardware Multiplier Registers

REGISTER	ADDRESS	FUNCTION
MCNT	M3[00h]	Multiplier Control Register. Selects operation, data type, operand count, hardware square function, and write option on the MC register. Also contains the overflow flag and the clear control for operand registers and accumulator.
MA	M3[01h]	Multiplier Operand A Register. Used by the user software to load one of the 16-bit values for a hardware multiplier operation.
MB	M3[02h]	Multiplier Operand B Register. Used by the user software to load one of the 16-bit values for a hardware multiplier operation.
MC2	M3[03h]	Multiplier Accumulate Register 2. Contains the two most significant bytes of the accumulator register. The 48-bit accumulator is formed by MC2, MC1 and MC0. The most significant bit of this register is the signed bit for signed operations.
MC1	M3[04h]	Multiplier Accumulate Register 1. Contains bytes 3 and 2 of the accumulator register. The 48-bit accumulator is formed by MC2, MC1 and MC0.
MC0	M3[05h]	Multiplier Accumulate Register 0. Contains the two least significant bytes of the accumulator register. The 48-bit accumulator is formed by MC2, MC1 and MC0.
MC1R	M3[08h]	Multiplier Read Register 1. Contains bytes 1 and 0 result from the last operation when MCW bit is 1 or the last operation is either multiply-only or multiply-negate. The contents of this register will remain until an SFR related to the multiplier has been changed.
MC0R	M3[09h]	Multiplier Read Register 0. Contains bytes 3 and 2 result from the last operation when MCW bit is 1 or the last operation is either multiply-only or multiply-negate. The contents of this register will remain unchanged until an SFR related to the multiplier has been changed.
SHFT	M3[07h]	Right and Left Shift Register: The shift operations are implemented to help with fixed point math. These operations only work on the 48-bit accumulator, MC [2:0] registers. The MCR [1:0] registers are not affected by a shift operation.
MACSEL	M3[0Eh]	MAC Select Register. The device has internally two sets of MAC registers. Using this register one of two MAC registers is selected which allows uninterruptible MAC operation.

18.5.1 – Multiplier Control Register (MCNT)

Bit	7	6	5	4	3	2	1	0
Name	OF	MCW	CLD	SQU	OPCS	MSUB	MMAC	SUS
Reset	0	0	0	0	0	0	0	0
Access	r	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7	OF	**Overflow Flag**. This bit is set to logic 1 when an overflow occurred for the last operation. This bit can be set for accumulation/subtraction operations or unsigned multiply-negate attempts. This bit is automatically cleared to 0 following a reset, starting a multiplier operation, or setting of the CLD bit to 0.
6	MCW	**MC Register Write Select**. The state of the MCW bit determines if an operation result will be placed into the accumulator registers (MC). 0 = The result will be written to the MC registers. 1 = The result is not written to the MC registers (MC register content is unchanged).
5	CLD	**Clear Data register**. This bit initializes the operand registers and the accumulator of the multiplier. When it is set to 1, the contents of all data registers and the OF bit are cleared to 0 and the operand load counter is reset immediately. This bit is cleared by hardware automatically. Writing a 0 to this bit has no effect.
4	SQU	**Square Function Enable**. This bit supports the hardware square function. When this bit is set to logic 1, a square operation is initiated after an operand is written to either the MA or the MB register. Writing data to either of the operand registers writes to both registers and triggers the specified square or square-accumulate/subtract operation. Setting this bit to 1 also overrides the OPCS bit setting. When SQU is cleared to logic 0, the hardware square function is disabled. 0 = Square function disabled 1 = Square function enabled
3	OPCS	**Operand Count Select**. This bit defines how many operands must be loaded to trigger a multiply or multiply-accumulate/subtract operation (except when SQU = 1 since this implicitly specifies a single operand). When this bit is cleared to logic 0, both operands (MA and MB) must be written to trigger the operation. When this bit is set to 1, the specified operation is triggered once either operand is written. 0 = Both operands (MA and MB) must be written to trigger the multiplier operation. 1 = Loading one operand (MA or MB) triggers the multiplier operation.
2	MSUB	**Multiply-Accumulate Negate**. Configuring this bit to logic 1 enables negation of the product for signed multiply operations and subtraction of the product from the accumulator (MC[2:0]) when MMAC = 1. When MSUB is configured to logic 0, the product of multiply operations will not be negated and accumulation is selected when MMAC = 1.
1	MMAC	**Multiply-Accumulate Enable**. This bit enables the accumulate or subtract operation (as per MSUB) for the hardware multiplier. When this bit is cleared to logic 0, the multiplier will perform only multiply operations. When this bit is set to logic 1, the multiplier will perform a multiply-accumulate or multiply-subtract operation based upon the MSUB bit. 0 = Accumulate/subtract operation disabled 1 = Accumulate/subtract operation enabled
0	SUS	**Signed-Unsigned**. This bit determines the data type of the operands. When this bit is cleared to logic 0, the operands will be treated as two's complement values and the multiplier will perform a signed operation. When this bit is set to logic 1, the operands will be treated as absolute magnitudes and the multiplier will perform an unsigned operation. 0 = Signed Operands 1 = Unsigned Operands

18.5.2 – Multiplier Operand A Register (MA)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	MA[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Multiplier Operand A: This operand A register is used by the application code to load 16-bit values for multiplier operations.

18.5.3 – Multiplier Operand B Register (MB)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	MB[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Multiplier Operand B: This operand B register is used by the application code to load 16-bit values for multiplier operations.

18.5.4 – Multiplier Accumulator 2 Register (MC2)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	MC2[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Multiplier Accumulator 2 Register: The MC2 register represents the two most significant bytes of the accumulator register. The 48-bit accumulator is formed by MC2, MC1 and MC0. For a signed operation, the most significant bit of this register is the sign bit.

18.5.5 – Multiplier Accumulator 1 Register (MC1)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	MC1[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Multiplier Accumulator 1 Register: The MC1 register represents bytes 3 and 2 of the accumulator register. The 48-bit accumulator is formed by MC2, MC1, and MC0.

18.5.6 – Multiplier Accumulator 0 Register (MC0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	MC0[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Multiplier Accumulator 0 Register: The MC0 register represents the two least significant bytes of the accumulator register. The 48-bit accumulator is formed by MC2, MC1, and MC0.

18.5.7 – Multiplier Read Register 1 (MC1R)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	MC1R[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Multiplier Read Register 1: The MC1R register represents bytes 3 and 2 from the result of the last operation when MCW = 1 or the last operation was a multiply or multiply-negate. When MCW = 0 and the last operation was a multiply-accumulate/subtract, the contents of this register may or may not agree with the contents of MC1 due to the combinatorial nature of the adder. The content of this register may change if MCNT, MA, MB, or MC[2:0] is changed.

18.5.8 – Multiplier Read Register 0 (MC0R)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	MC0R[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Multiplier Read Register 0: The MC1R register represents bytes 1 and 0 from the result of the last operation when MCW = 1 or the last operation was a multiply or multiply-negate. When MCW = 0 and the last operation was a multiply-accumulate/subtract, the contents of this register may or may not agree with the contents of MC0 due to the combinatorial nature of the adder. The content of this register may change if MCNT, MA, MB or MC[2:0] is changed.

18.5.9 – MAC Select Register (MACSEL)

Bit	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	MACRSEL
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	rw

BIT	NAME	DESCRIPTION
7:1	-	Reserved
0	MACRSEL	MAC Registers Select Register. The device has internally two sets of MAC registers. Using this bit one of two MAC registers is selected which allows uninterruptible MAC operation.

18.5.10 – MAC Shift Register (SHIFT)

Bit	7	6	5	4	3	2	1	0
Name	SHC	-	-	-	-	-	SR	SL
Reset	0	0	0	0	0	0	0	0
Access	rw	r	r	r	r	r	rw	rw

BIT	NAME	DESCRIPTION
7	SHC	Shift Carry: This bit represents the carry out from last shift operation. For a left shift operation this bit will get MC2[15] (MSB of MC2 register). For a right shift operation this bit will get MC0[0] (LSB of MC0 register). This bit can be cleared by writing a 0 to it.
6:2	-	Reserved
1	SR	Shift Right: a 1 to this bit will cause one bit right shift operation on MC2-M0 register. This bit auto clears itself, so a read on SHFT register will always return 0 for this bit position.
0	SL	Shift Left: a 1 to this bit will cause one bit left shift operation on MC2-M0 register. This bit auto clears itself, so a read on SHFT register will always return 0 for this bit position.

The shift (right/left) operations are implemented for faster fixed point math operations. These operations only work on the 48-bit accumulator, MC [2:0] registers. The MCR [1:0] registers are not affected by a shift operation.

Right Shift Operation:
On doing a right shift the MC2-MC0 contents will be
MC2[15:0] = MC2[15],MC2[15:1] (MSB bit, sign bit, is preserved)
MC1[15:0] = MC2[0],MC1[15:1]
MC0[15:0] = MC1[0],MC0[15:1]
SHC = MC0[0]

Left Shift Operation:
On doing a left shift the MC2-MC0 contents will be
SHC = MC2[15] (shifted sign bit)
MC2[15:0] = MC2[14:0],MC1[15]
MC1[15:0] = MC1[14:0],MC0[15]
MC0[15:0] = MC0[14:0], 0

18.6 – Hardware Multiplier Examples

The following are code examples of multiplier operations.

```
;Unsigned Multiply 16-bit x 16-bit
move   MCNT, #21h       ; CLD=1, SUS=1 (unsigned)
move   MA, #0FFFh       ; MC2:0=0000_0000_0000h
move   MB, #1001h       ; MC1R:MC0R= 00FF_FFFFh
                        ; MC2:0=0000_00FF_FFFFh

;Signed Multiply 16-bit x 16-bit
move   MCNT, #20h       ; CLD=1, SUS=0 (signed)
move   MA, #F001h       ; MC2:0=0000_0000_0000h
move   MB, #1001h       ; MC1R:MC0R= FF00_0001h
                        ; MC2:0=FFFF_FF00_0001h

;Unsigned Multiply-Accumulate 16-bit x 16-bit
                        ; MC2:0=0000_0100_0001h
move   MCNT, #03h       ; MMAC=1, SUS=1 (unsigned)
move   MA, #0FFFh       ;
move   MB, #1001h       ;
                        ; MC1R:MC0R=02FF_FFFFh
                        ; MC2:0=0000_0200_0000h

;Signed Multiply-Accumulate 16-bit x 16-bit
                        ; MC2:0=0000_0100_0001h
move   MCNT, #02h       ; SUS=0 (signed)
move   MA, #F001h       ;
move   MB, #1001h       ;
                        ; MC1R:MC0R= FF00_0003h
                        ; MC2:0=0000_0000_0002h

;Unsigned Multiply-Subtract 16-bit x 16-bit
                        ; MC2:0=0000_0100_0001h
move   MCNT, #07h       ; MMAC=1, MSUB=1, SUS=1 (unsigned)
move   MA, #0FFFh       ;
move   MB, #1001h       ;
                        ; MC1R:MC0R=FF00_0003h
                        ; MC2:0=0000_0000_0002h

;Signed Multiply-Subtract 16-bit x 16-bit
                        ; MC2:0=0000_0100_0001h
move   MCNT, #06h       ; MMAC=1, MSUB=1, SUS=0 (signed)
move   MA, #F001h       ;
move   MB, #1001h       ;
                        ; MC1R:MC0R= 02FF_FFFFh
                        ; MC2:0=0000_0200_0000h

;Signed Multiply Negate 16-bit x 16-bit
move   MCNT, #24h       ; CLD=1, MSUB=1, SUS=0 (signed)
move   MA, #F001h       ; MC2:0=0000_0000_0000h
move   MB, #1001h       ; MC1R:MC0R =00FF_FFFFh
                        ; MC2:0=0000_00FF_FFFFh
```

SECTION 19 – WATCHDOG TIMER

19.1 - Overview

The Watchdog Timer is a user programmable clock counter that can serve as a time-base generator, an event timer, or a system supervisor. As can be seen in Figure 19-1, the timer is driven by the main system clock and is supplied to a series of dividers. If the watchdog interrupt and the watchdog reset are disabled (WDCN.EWDI = 0 and WDCN.EWT = 0), the watchdog timer and its input clock are disabled. Whenever the watchdog timer is disabled, the watchdog interval timer (per WDCN.WD[1:0] bits) and 512 clock reset counter will be reset if either the interrupt or reset function is enabled. When the watchdog timer is initially enabled, there will be a 1-clock to 3-clock cycle delay before it starts. The divider output is selectable, and determines the interval between timeouts. When the timeout is reached, an interrupt flag will be set, and if enabled, an interrupt occurs. A watchdog-reset function is also provided in addition to the watchdog interrupt. The reset and interrupt are completely discrete functions that may be acknowledged or ignored, together or separately for various applications.

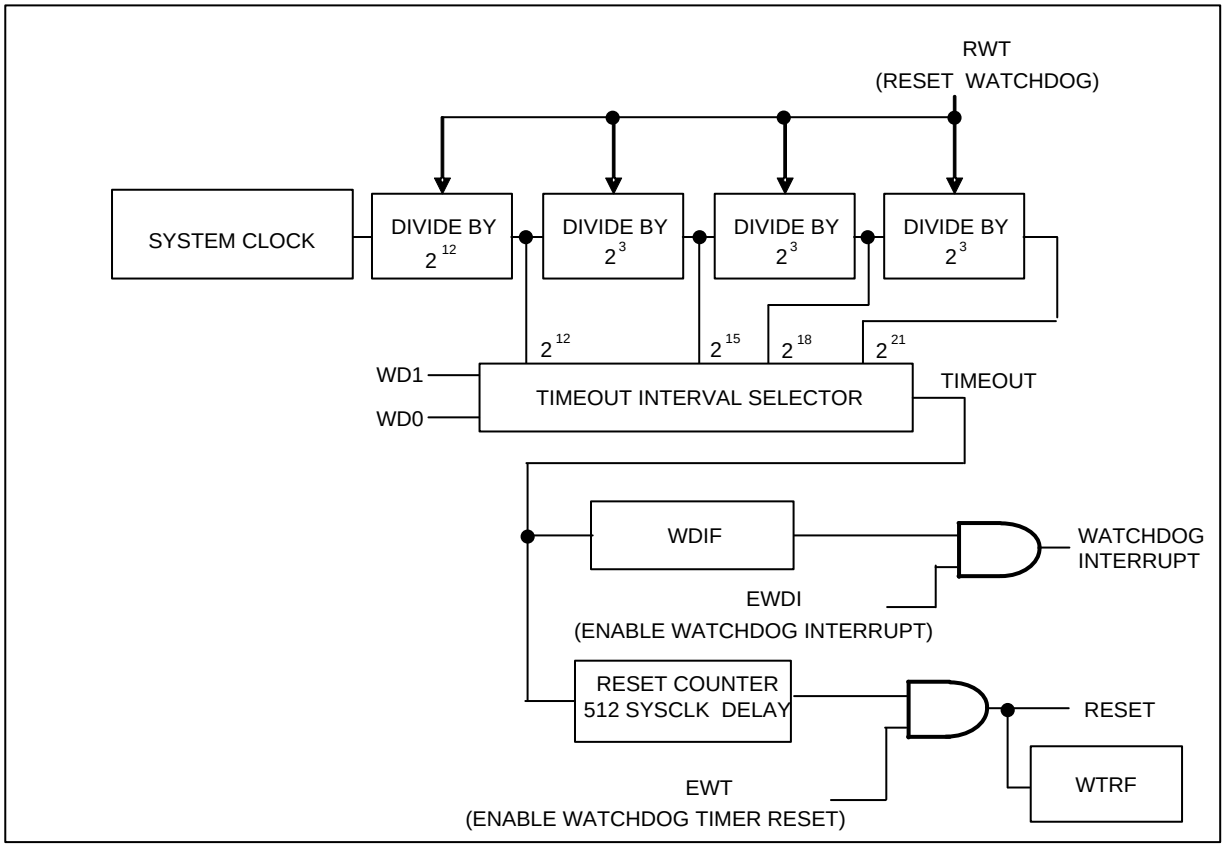


Figure 19-1: Watchdog Timer Block Diagram

19.2 – Watchdog Timer Description

When the watchdog timer is enabled, it begins counting system clock cycles. The watchdog count will be reset anytime RWT is set to 1. If the watchdog count reaches the time interval set by WD1:0], a watchdog timeout occurs, setting the Watchdog Interrupt Flag (WDCN.WDIF). A watchdog timeout will also generate an interrupt and/or reset the DS4830A. Table 19-1 describes the possible states of the watchdog timer.

Table 19-1: Watchdog Operating States

EWT	EWDI	WDIF	ACTIONS
x	X	0	No interrupt has occurred.
0	0	x	Watchdog disable, clock is gated off.
0	1	1	Watchdog interrupt has occurred.
1	0	1	No interrupt has been generated. Watchdog reset will occur in 512 system clock cycles if RWT is not set or WDIF not cleared.
1	1	1	Watchdog interrupt has occurred. Watchdog reset will occur in 512 system clock cycles if RWT is not set or WDIF not cleared.

19.2.1 – Watchdog Timer Interrupt Operation

The watchdog interrupt is enabled using the Enable Watchdog Timer Interrupt (WDCN.EWDI) bit. When the timeout occurs, the watchdog timer will set the Watchdog Interrupt Flag bit (WDCN.WDIF), and an interrupt will occur if the interrupt global enable (IC.IGE) and system interrupt mask (IMR.IMS) are set and an interrupt is not currently being serviced (IC.INS = 0). The Watchdog Interrupt Flag must be cleared by software.

19.2.2 – Watchdog Timer Reset Operation

In order to reset the DS4830A, the watchdog timer reset function must be enabled by setting the Enable Watchdog Timer Reset (WDCN.EWT) bit. When a watchdog timeout occurs, the WDIF flag will be set and an interrupt will be generated if enabled. Following the timeout, the watchdog will count an additional 512 system clock cycles. To avoid a reset, software must either set the RWT bit or clear the EWT bit. This can occur at any time during the watchdog timer interval or the additional 512 system clock cycles after WDIF is set. At the end of the 512 system clock cycles the DS4830A will be reset. When the reset occurs, the Watchdog Timer Reset Flag (WDCN.WTRF) will automatically be set to indicate the cause of the reset. Software must clear this bit manually.

19.2.3 – Watchdog Timer Applications

Using the watchdog interrupt during software development can allow the user to select ideal watchdog reset locations. Code is first developed without enabling the watchdog interrupt or reset functions. Once the program is complete, the watchdog interrupt function is enabled to identify the required locations in code to set the RWT bit. Incrementally adding instructions to reset the watchdog timer prior to each address location (identified by the watchdog interrupt) will allow the code to eventually run without receiving a watchdog interrupt. At this point the watchdog timer reset can be enabled without the potential of generating unwanted resets. At the same time the watchdog interrupt may also be disabled. Proper use of the watchdog interrupt with the watchdog reset allows interrupt software to survey the system for errant conditions.

When using the watchdog timer as a system monitor, the watchdog reset function should be used. If the interrupt function were used, the purpose of the watchdog would be defeated. For example, assume the system is executing errant code prior to the watchdog interrupt. The interrupt would temporarily force the system back into control by vectoring the CPU to the interrupt service routine. Restarting the watchdog and exiting by an RETI or RET, would return the processor to the errant code. By using the watchdog reset function, the processor is restarted from the beginning of the program, and therefore placed into a known state.

The watchdog timer is controlled by the Watchdog Timer Control Register, WDCN. The WDCN register is one of the system register and is located in Module 8, Register 19. The bit names and description of WDCN are listed in Table 19-2.

Table 19-2: Watchdog Timer Control Register Bits (WDCN)

Bit	7	6	5	4	3	2	1	0
Name	POR	EWDI	WD1	WD0	WDIF	WTRF	EWT	RWT
Reset	s*	s*	0	0	0	s*	s*	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

*Bits 5, 4, 3 and 0 are cleared to 0 on all forms of reset; for others, see individual bit descriptions.

<table>
<tr><th>BIT</th><th>NAME</th><th>DESCRIPTION</th></tr>
<tr><td>7</td><td>POR</td><td>Power-On Reset Flag: This bit is set to 1 whenever a power-on/brownout reset occurs. It is unaffected by other forms of reset. This bit can be checked by software following a reset to determine if a power-on/brownout reset occurred. It should always be cleared by software following a reset to ensure that the sources of following resets can be determined correctly.</td></tr>
<tr><td>6</td><td>EWDI</td><td>Enable Watchdog Timer Interrupt: If this bit is set to 1, an interrupt request can be generated when the WDIF bit is set to 1 by any means. If this bit is cleared to 0, no interrupt will occur when WDIF is set to 1, however, it does not stop the watchdog timer or prevent watchdog resets from occurring if EWT = 1. If EWT = 0 and EWDI = 0, the watchdog timer will be stopped. If the watchdog timer is stopped (EWT = 0 and EWDI = 0), setting the EWDI bit will reset the watchdog interval and reset counter, and enable the watchdog timer. This bit is cleared to 0 by power-on reset and is unaffected by other forms of reset.</td></tr>
<tr><td>5:4</td><td>WD[1:0]</td><td>Watchdog Timer Interval Control Bits: These bits determine the watchdog timeout interval. The timeout interval is set in terms of system clocks. Modifying the watchdog interval will automatically reset the watchdog timer unless the 512 system clock reset counter is already in progress, in which case, changing the WD[1:0] bits will not affect the watchdog timer or reset counter.
<table>
<tr><th>WD1</th><th>WD0</th><th>CLOCKS UNTIL INTERRUPT</th><th>CLOCKS UNTIL RESET</th></tr>
<tr><td>0</td><td>0</td><td>2^{12}</td><td>2^{12} + 512</td></tr>
<tr><td>0</td><td>1</td><td>2^{15}</td><td>2^{15} + 512</td></tr>
<tr><td>1</td><td>0</td><td>2^{18}</td><td>2^{18} + 512</td></tr>
<tr><td>1</td><td>1</td><td>2^{21}</td><td>2^{21} + 512</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>WDIF</td><td>Watchdog Interrupt Flag: This bit will be set to 1 when the watchdog timer interval has elapsed or can be set to 1 by user software. When WDIF = 1, an interrupt request will occur if the watchdog interrupt has been enabled (EWDI = 1) and not otherwise masked or prevented by an interrupt already in service (i.e., IGE = 1, IMS = 1, and INS = 0 must be true for the interrupt to occur). This bit should be cleared by software before exiting the interrupt service routine to avoid repeated interrupts. Furthermore, if the watchdog reset has been enabled (EWT = 1), a reset is scheduled to occur 512 system clock cycles following setting of the WDIF bit.</td></tr>
<tr><td>2</td><td>WTRF</td><td>Watchdog Timer Reset Flag: This bit is set to 1 when the watchdog resets the processor. Software can check this bit following a reset to determine if the watchdog was the source of the reset. Setting this bit to 1 in software will not cause a watchdog reset. This bit is cleared by power-on reset only and is unaffected by other forms of reset. It should also be cleared by software following any reset so that the source of the next reset can be correctly determined by software. This bit is only set to 1 when a watchdog reset actually occurs. If EWT is cleared to 0 when the watchdog timer elapses, this bit will not be set.</td></tr>
<tr><td>1</td><td>EWT</td><td>Enable Watchdog Timer Reset: If this bit is set to 1 when the watchdog timer elapses, the watchdog resets the DS4830A 512 system clock cycles later unless action is taken to disable the reset event. Clearing this bit to 0 prevents a watchdog reset from occurring but does not stop the watchdog timer or prevent watchdog interrupts from occurring if EWDI = 1. If EWT = 0 and EWDI = 0, the watchdog timer will be stopped. If the watchdog timer is stopped (EWT = 0 and EWDI = 0), setting the EWT bit will reset the watchdog interval and reset counter, and enable the watchdog timer. This bit is cleared on power-on reset and is unaffected by other forms of reset.</td></tr>
<tr><td>0</td><td>RWT</td><td>Reset Watchdog Timer: Setting this bit to 1 resets the watchdog timer count. If watchdog interrupt and/or reset modes are enabled, the software must set this bit to 1 before the watchdog timer elapses to prevent an interrupt or reset from occurring. This bit always returns 0 when read.</td></tr>
</table>

SECTION 20 – TEST ACCESS PORT (TAP)

The DS4830A incorporates a Test Access Port (TAP) and TAP controller for communication with a host device across a 4-wire synchronous serial interface. The TAP may be used by the DS4830A to support in-system programming and/or in-circuit debug. The TAP is compatible with the JTAG IEEE standard 1149 and is formed by four interface signals described in Table 20-1. For detailed information on the TAP and TAP controller, refer to IEEE STD 1149.1 "IEEE Standard Test Access Port and Boundary-Scan Architecture."

Table 20-1: Test Access Port Pins

EXTERNAL PIN SIGNAL	FUNCTION
TDO (Test Data Output)	Serial-Data Output. This signal is used to serially transfer internal data to the external host. Data is transferred least significant bit first. Data is driven out only on the falling edge of TCK, only during TAP Shift-IR or Shift-DR states and is otherwise inactive.
TDI (Test Data Input)	Serial-Data Input. This signal is used to receive data serially transferred by the host. Data is received least significant bit first and is sampled on the rising edge of TCK. TDI is weakly pulled high internally when TAP=1.
TCK (Test Clock Input)	Serial Shift Clock Provided by Host. When this signal is stopped at 0, storage elements in the TAP logic must retain their data indefinitely. TCK is weakly pulled high internally when TAP=1.
TMS (Test Mode Select Input)	Mode Select Input. This signal is sampled at the rising edge of TCK and controls movement between TAP states. TMS is weakly pulled high internally when TAP=1.

These pins default to the TAP/JTAG function on reset, which means that the part is always ready for in-circuit debugging or in-circuit programming operations following any reset. Once an application has been loaded and starts running, the TAP/JTAG port can still be used for in-circuit debugging operations. If in-circuit debugging functionality is not needed, the associated port pins can be reclaimed for application use by setting the TAP bit (SC.7) bit to 0. This disables the TAP/JTAG interface and allows the four pins to operate as normal port pins.

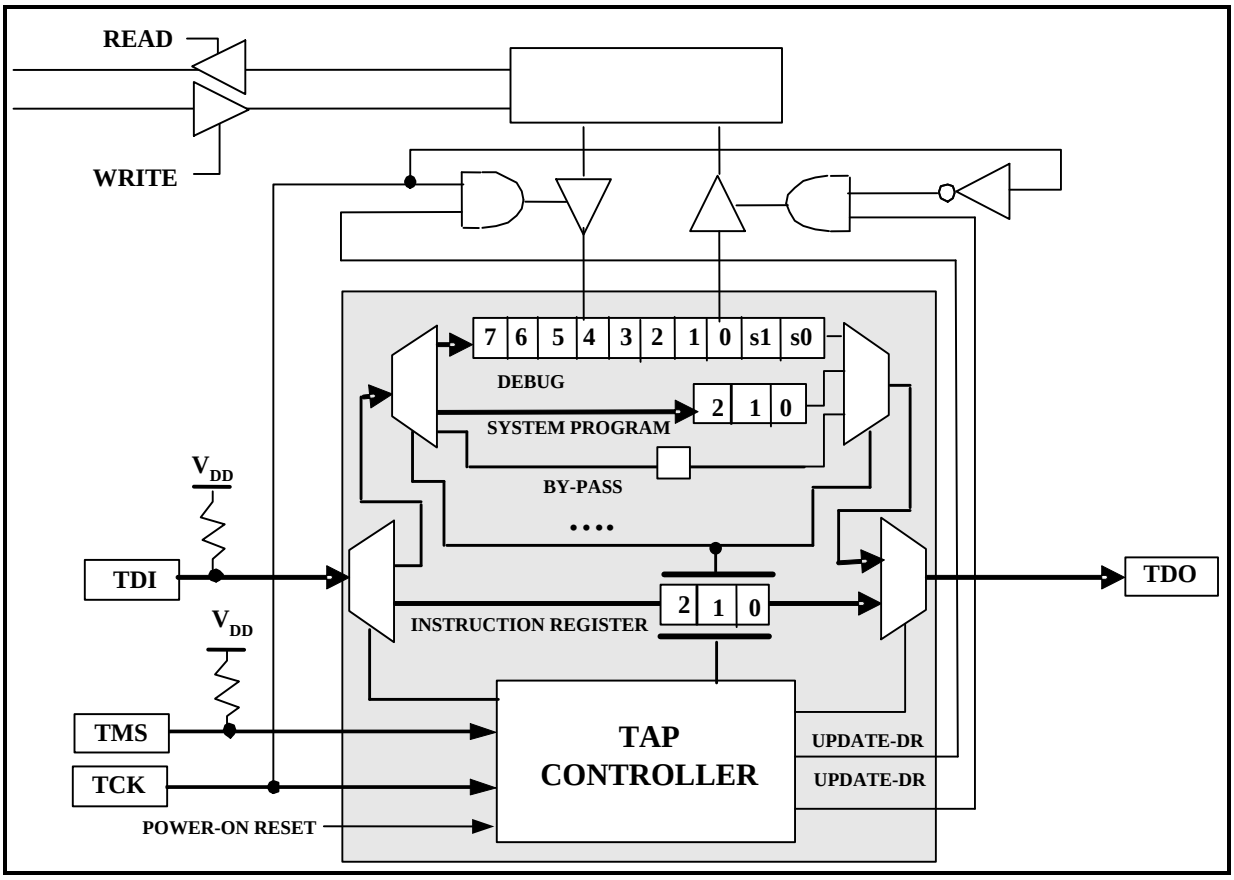


Figure 20-1: TAP and TAP Controller

20.1 – TAP Controller

The TAP controller is a synchronous state machine that responds to changes at the TMS and TCK signals. Based on its state transition, the controller provides the clock and control sequence for TAP operation. The performance of the TAP is dependent on the TCK clock frequency. The maximum TCK clock frequency should be limited to 1/8 the system clock frequency. This section provides a brief description of the state machine and its state transitions. The state diagram in Figure 20-1 summarizes the transitions caused by the TMS signal sampling on the rising edge at TCK. The TMS signal value is presented adjacent to each state transition in the figure.

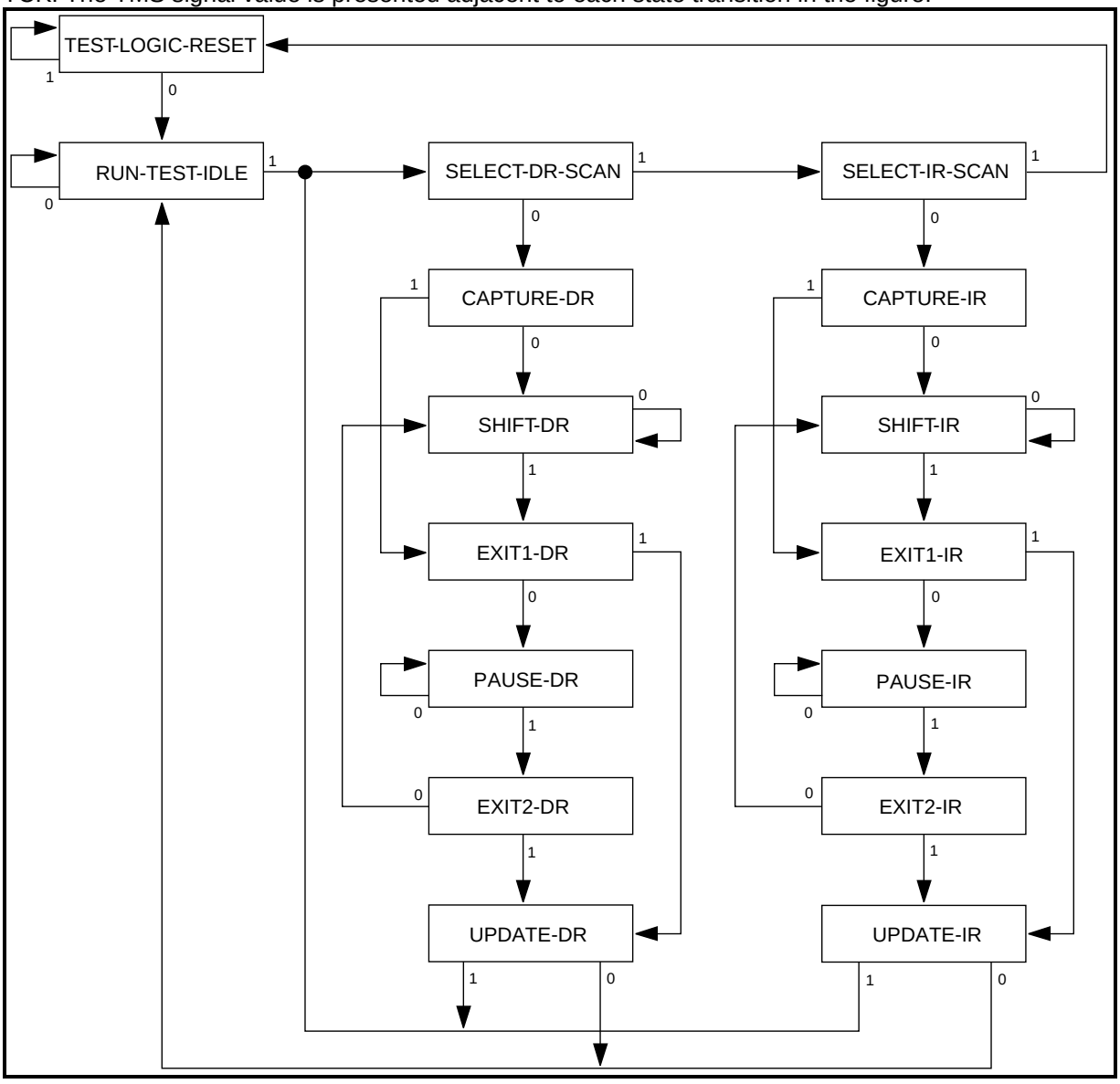


Figure 20-2: TAP Controller State Diagram

20.2 – TAP State Control

The TAP provides an independent serial channel to communicate synchronously with the host system. The TAP state control is achieved through host manipulation of the Test Mode Select (TMS) and Test Clock (TCK) signals. The TMS signal is sampled at the rising edge of TCK and decoded by the TAP controller to control movement between the TAP states. The TDI input and TDO output are meaningful once the TAP is in a serial shift state (i.e. Shift-IR or Shift-DR).

20.2.1 – Test-Logic-Reset

On a power-on reset, the TAP controller is initialized to the Test-Logic-Reset state and the instruction register (IR2:0) is initialized to the By-Pass instruction so that it will not affect normal system operation. No matter what the state of the controller, it will enter Test-Logic-Reset when TMS is held high for at least five rising edges of TCK. The controller remains in the Test-Logic-Reset state if TMS remains high. An erroneous low signal on the TMS may cause the controller to move into the Run-Test-Idle state but no disturbance is caused to system operation if the TMS signal is returned and kept at the intended logic high for three rising edges of TCK since this returns the controller to the Test-Logic-Reset state.

20.2.2 – Run-Test-Idle

As illustrated in Figure 20-2, the Run-Test-Idle state is simply an intermediate state for getting to one of the two state sequences in which the controller performs meaningful operations:

- Controller state sequence (IR-Scan), or
- Data register state sequence (DR-Scan)

20.2.3 – IR-Scan Sequence

The controller state sequence allows instructions (e.g. 'Debug' and 'System Programming') to be shifted into the instruction register starting from the Select-IR-Scan state. In the TAP, the instruction register is connected between the TDI input and the TDO output. Inside the IR-Scan Sequence, the Capture-IR state loads a fixed binary pattern (001b) into the 3-bit shift register and the Shift-IR state causes shifting of TDI data into the shift register and serial output to TDO, least significant bit first. Once the desired instruction is in the shift register, the instruction can be latched into the parallel instruction register (IR2:0) on the falling edge of TCK in the Update-IR state. The contents of the 3-bit instruction shift register and parallel instruction register (IR2:0) are summarized with respect to the TAP controller states in Table 20-2.

Table 20-2: Instruction Register Content vs. TAP Controller State

TAP CONTROLLER STATE	INSTRUCTION SHIFT REGISTER	PARALLEL (3-BIT) INSTRUCTION REGISTER (IR2:0)
Test-Logic-Reset	Undefined	Set to By-pass (011b) Instruction
Capture-IR	Load 001b at the rising edge of TCK	Retain last state
Shift-IR	Input data via TDI and Shift towards TDO at the rising edge of TCK	Retain last state
Exit1-IR Exit2-IR Pause-IR	Retain last state	Retain last state
Update-IR	Retain last state	Load from shift register at the falling edge of TCK
All other states	Undefined	Retain last state

When the parallel instruction register (IR2:0) is updated, the TAP controller decodes the instruction and performs any necessary operations, including activation of the data shift register to be used for the particular instruction during data register shift sequences (DR-Scan). The length of the activated shift register depends upon the value loaded to the instruction register (IR2:0). The supported instruction register encodings and associated data register selections are shown in Table 20-3.

Table 20-3: Instruction Register (IR2:0) Encodings

IR2:0	INSTRUCTION	FUNCTION	SERIAL DATA SHIFT REGISTER SELECTION
000	Extest	No operation	Unchanged. Retain previous selection
001	Sample/Preload	No operation	Unchanged. Retain previous selection
010	Debug	In-circuit debug mode	10-bit shift register
011	By-pass	No operation (default)	1-bit shift register
100	System Programming	Bootstrap function	3-bit shift register
101	By-pass	No operation (default)	1-bit shift register
110	Reserved		
111	By-pass	No operation (default)	1-bit shift register

The Extest (IR2:0 = 000b) and Sample/Preload (IR2:0 = 001b) instructions are mandated by the JTAG standard, however, the DS4830A does not intend to make practical use of these instructions. Hence, these instructions are treated as no operations but may be entered into the instruction register without affecting the on-chip system logic or pins and does not change the existing serial data register selection between TDI and TDO.

The By-pass (IR2:0 = 011b, 101b, or 111b) instruction is also mandated by the JTAG standard. The By-pass instruction is fully implemented by the DS4830A to provide a minimum length serial data path between the TDI and the TDO pins. This is accomplished by providing a single cell bypass shift register. When the instruction register is updated with the By-pass instruction, a single bypass register bit is connected serially between TDI and TDO in the Shift-DR state. The instruction register automatically defaults to the By-pass instruction when the TAP is in the Test-Logic-Reset state. The By-pass instruction has no effect on the operation of the on-chip system logic.

The Debug (IR2:0 = 010b) and System Programming (IR2:0 = 100b) instructions are private instructions which are intended solely for in-circuit debug and in-system programming operations respectively. If the instruction register is updated with the Debug instruction, a 10-bit serial shift register is formed between the TDI and TDO pins in the Shift-DR state. If the System Programming instruction is entered into the instruction register (IR2:0), a 3-bit serial data shift register is formed between the TDI and TDO pins in the Shift-DR state.

Instruction register (IR2:0) settings other than those listed and described above are reserved for internal use. As can be seen in Figure 20-2, the instruction register serves to select the length of the serial data register between TDI and TDO during the Shift-DR state.

20.2.4 – DR-Scan Sequence

Once the instruction register has been configured to a desired state (mode), transactions are performed via a data buffer register associated with that mode. These data transactions are executed serially in a manner analogous to the process used to load the instruction register and are grouped in the TAP Controller state sequence starting from the Select-DR-Scan state. In the TAP controller state sequence, the Shift-DR state allows internal data to be shifted out through the TDO pin while the external data is shifted in simultaneously via the TDI pin. Once a complete data pattern is shifted in, input data can be latched into the parallel buffer of the selected register on the falling edge of TCK in the Update-DR state. On the same TCK falling edge, in the Update-DR state, the internal parallel buffer is loaded to the data shift register for output. This Shift-DR/Update-DR process serves as the basis for passing information between the external host and the DS4830A. These data register transactions occur in the data register portion of the TAP controller state sequence diagram and have no effect on the instruction register.

20.3 – Communication via TAP

The TAP controller is in Test-Logic-Reset state after a power-on-reset. During this initial state, the instruction register contains By-pass instruction and the serial path defined between the TDI and TDO pins for the Shift-DR state is the 1-bit bypass register. All TAP signals (TCK, TMS, TDI, and TDO) default to being weakly pulled high internally on any reset. The TAP controller will remain in the Test-Logic-Reset state as long as TMS is held high. The TCK and TMS signals may be manipulated by the host to transition to other TAP states. The TAP controller will remain in a given state whenever TCK is held low.

For the host to establish a specific data communication link, a private instruction must be loaded into the IR2:0 register. Once the instruction is latched in the instruction parallel buffer at the Update-IR state, it is recognized by the TAP controller and the communication channel is established. In-Circuit Debug or In-System Programming

commands and data can be exchanged between the host and the DS4830A by operating in the data register portion of the state sequence (i.e. DR-Scan). The TAP retains the private instruction which was loaded into IR2:0 until a new instruction is shifted in or until the TAP controller returns to the Test-Logic-Reset state.

20.3.1 – TAP Communication Examples – IR-Scan and DR-Scan

Figures 20-3 and 20-4 illustrate examples of communication between the host JTAG controller and the Test Access Port (TAP) of the DS4830A. The host controls the TCK and TMS signals to move through the desired TAP states while accessing the selected shift register through the TDI input and TDO output pair.

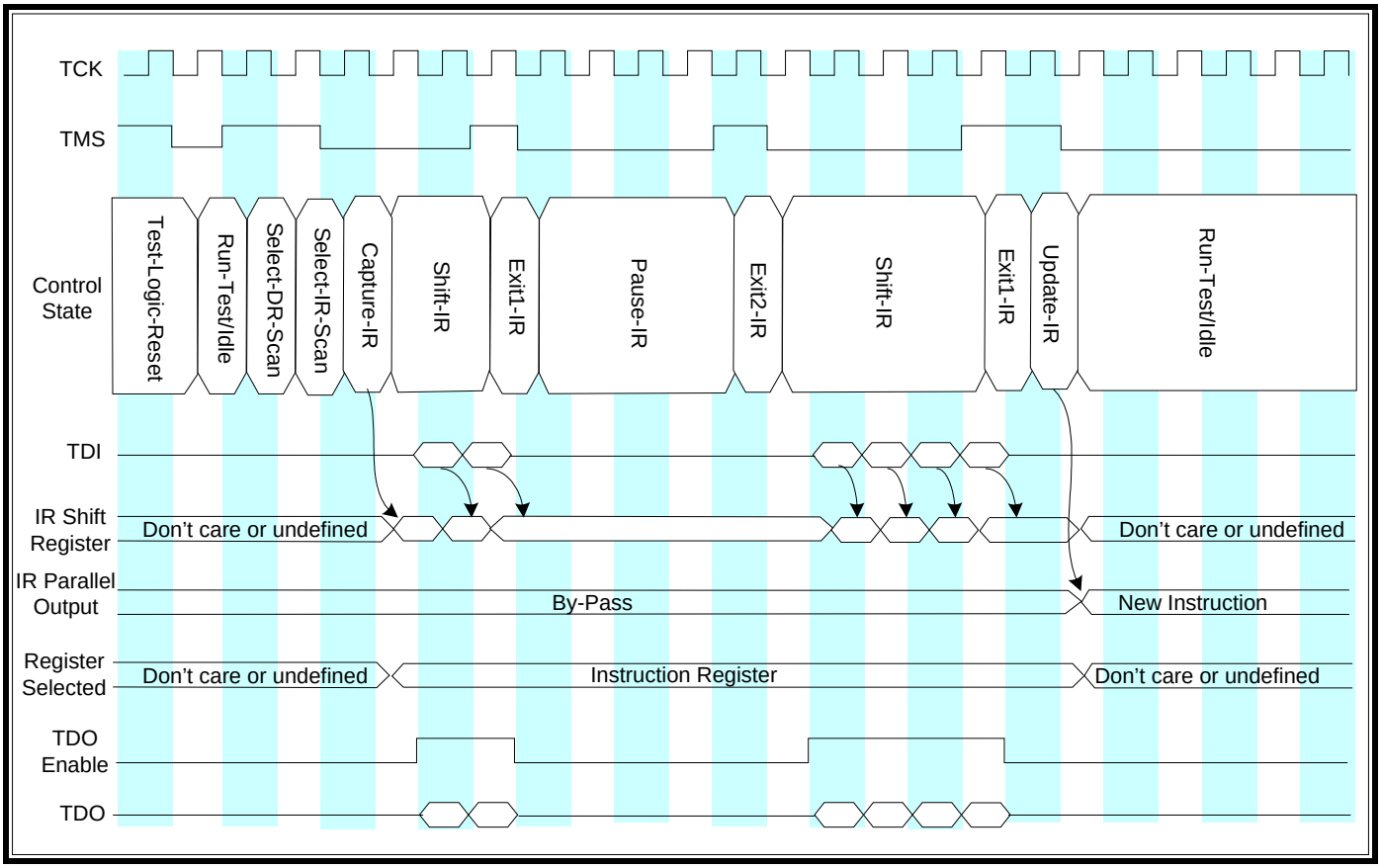


Figure 20-3: TAP Controller Debug Mode IR-Scan Example

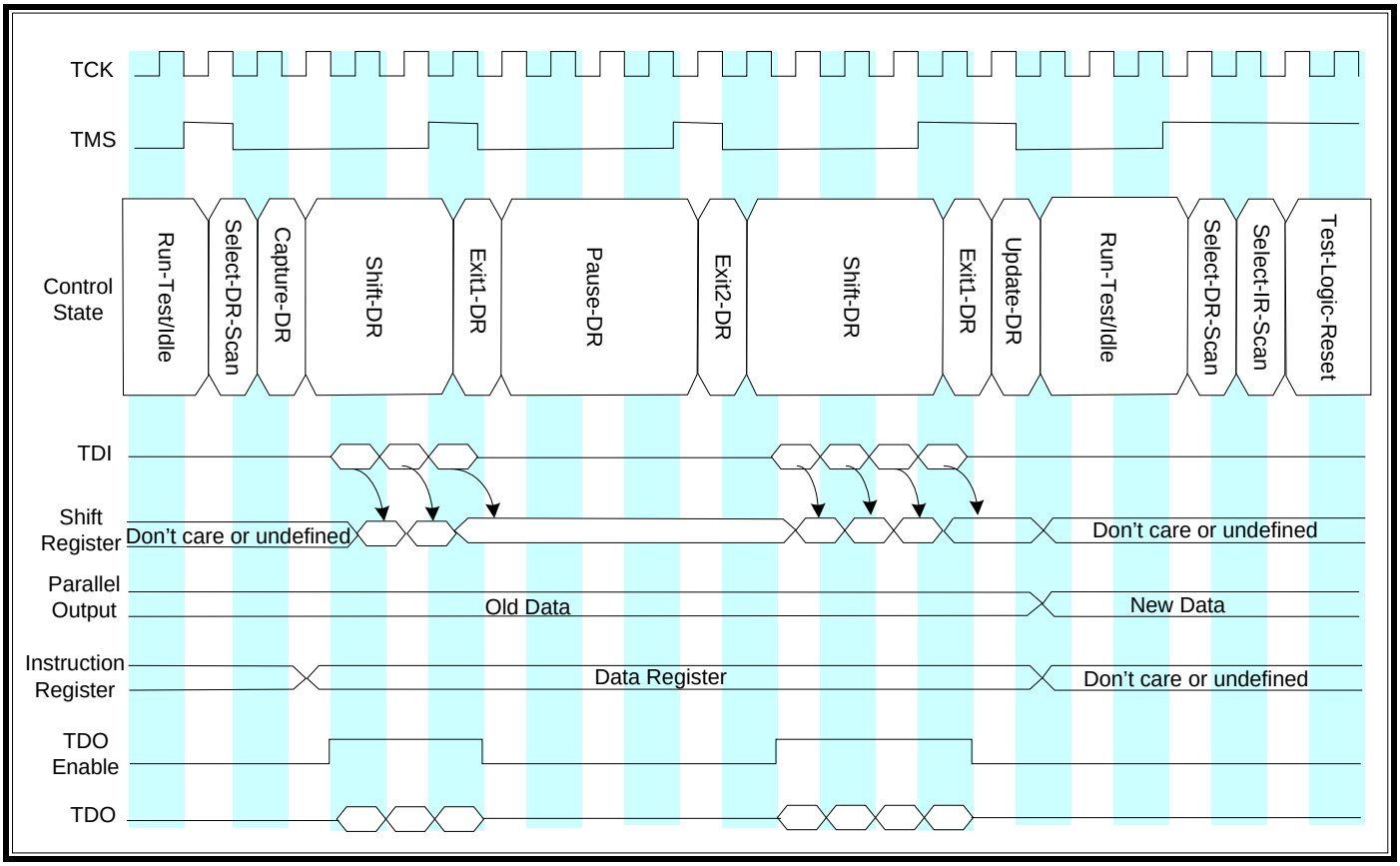


Figure 20-4: TAP Controller Debug Mode DR-Scan Example

SECTION 21 – IN-CIRCUIT DEBUG MODE

The DS4830A is equipped with embedded debug hardware and embedded ROM firmware developed for the purpose of providing in-circuit debugging capability to the user application. The in-circuit debug mode uses the JTAG-compatible Test Access Port (TAP) as its means of communication between the host and the DS4830A. Figure 21-1 shows a block diagram of the in-circuit debugger. The in-circuit debug hardware and software features include:

- a debug engine,
- a set of registers providing the ability to set breakpoints on register, code, or data,
- a set of debug service routines stored in a ROM.

Collectively, these hardware and software features allow two basic modes of in-circuit debugging:

- Background mode allows the host to configure and set up the in-circuit debugger while the CPU continues to execute the normal program. Debug mode can be invoked from Background mode.
- Debug mode allows the debug engine to take control of the CPU, providing read write access to internal registers and memory, and single step trace operation.

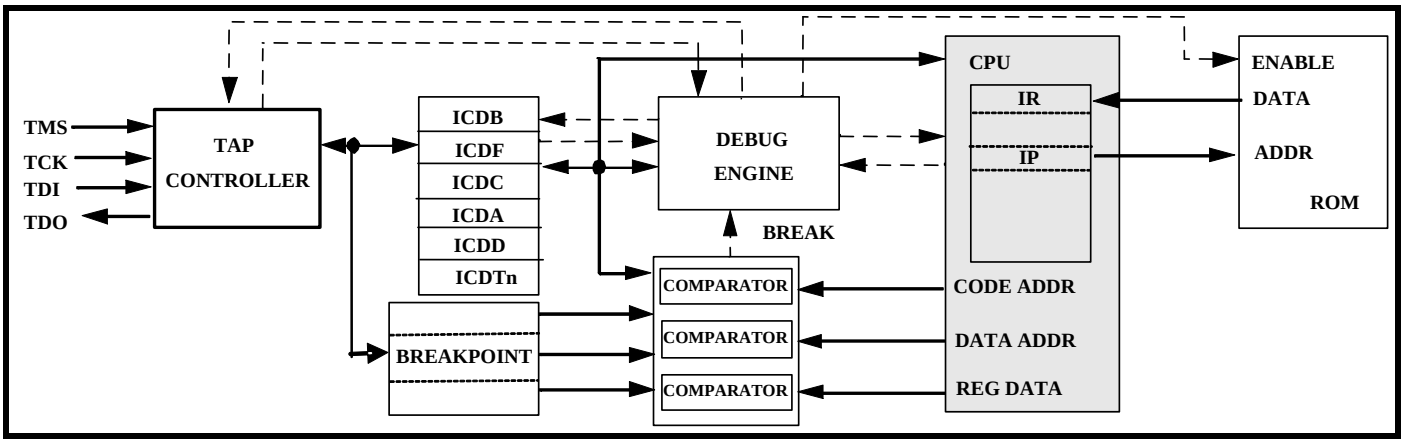


Figure 21-1: In-Circuit Debugger

The embedded hardware debug engine is implemented as a stand-alone hardware block in the DS4830A. The debug engine can be enabled for monitoring internal activities and interacting with selected internal registers while the CPU is executing user code. This capability allows the user to employ the embedded debug engine to debug the actual system, in place of the in-circuit emulator that uses external hardware to duplicate operation of the microcontroller outside of the real application environment.

To enable a communication link between the host and the microcontroller debug engine, the Debug instruction (010b) must be loaded into the TAP instruction register using the IR-Scan sequence. Once the instruction is latched in the instruction parallel buffer (IR2:0) and is recognized by the TAP controller in the Update-IR state, the 10-bit data shift register is activated as the communication channel for DR-Scan sequences. The TAP instruction register retains the Debug instruction until a new instruction is shifted via an IR-Scan or the TAP controller returns to the Test-Logic-Reset state.

The host now can transmit and receive serial data through the 10-bit data shift register that exists between the TDI input and TDO output during DR-Scan sequences. All background and debug mode communication (commands, data input/output, and status) occurs via this serial channel. Each 10-bit exchange of data between the host and the DS4830A internal hardware is composed of two status bits and a single byte of command or data. The 10-bit word is always transmitted least significant bit first with the format shown in Figure 21-2. The details of the two status bits are shown in Table 21-1.

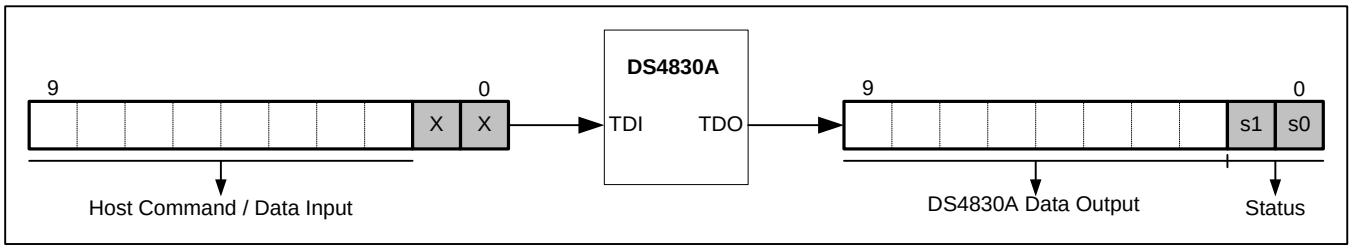


Figure 21-2: 10-Bit Word Format

Table 21-1: Status Bits

s1:s0	STATUS/CONDITION
00	**Non-Debug**. Default condition, Background mode, or debug engine inactive.
01	**Debug Idle**. Debug engine is ready to receive data from the host (command, data).
10	**Debug Busy**. Debug engine is busy without valid data (i.e. ROM code execution, trace operations).
11	**Debug Valid**. Debug engine is busy with valid data

The data byte portion of the 10-bit shift register is interfaced directly to the ICDB parallel register. The ICDB register functions as the holding data register for both transmit and receive operations. On the falling edge of TCK in the Update-DR state, the outgoing data is loaded from the ICDB parallel register to the debug shift register and the incoming shift register data is latched in the ICDB parallel register.

21.1 – Background Mode Operation

When the instruction register is loaded with the Debug instruction (IR2:0 = 010b), the host can communicate with the DS4830A in a background mode using TAP DR-Scan sequences without disturbing CPU operation. Note, however, that JTAG in-system programming also requires use of the 10-bit debug shift register and, if enabled (JTAG_SPE=1, PSS1:0= 0), takes precedence over background mode communication. When operating in background mode, the status bits are always cleared to 00b (non-debug), which indicates that the DS4830A is ready to receive background mode commands.

The host can perform the following operations from background mode:

- read/write internal breakpoint registers (BP0–BP5)
- read/write internal in-circuit debug registers (ICDC, ICDF, ICDA, ICDD)
- monitor to determine when a breakpoint match has occurred
- directly invoke debug mode

Table 21-2 shows the background mode commands supported by the DS4830A. Encodings not listed in this table are not supported in background mode and are treated as no operations.

Table 21-2: Background Mode Commands

OPCODE	COMMAND	OPERATION
0000-0000	No Operation	**No operation**. (Default state for Debug Shift register).
0000-0001	Read ICDC	**Read control data from the ICDC**. The contents of the ICDC register will be loaded into the Debug Shift Register via the ICDB register for host read. This command requires one follow-on transfer cycle.
0000-0010	Read ICDF	**Read flags from the ICDF.** The contents of the ICDF register (one byte) will be loaded into the Debug Shift Register via the ICDB register for host read. This command requires one follow-on transfer cycle.
0000-0011	Read ICDA	**Read data from the ICDA.** The contents of the ICDA register will be loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0100	Read ICDD	**Read data from the ICDD**. The contents of the ICDD register will be loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0101	Read BP0	**Read data from the BP0**. The contents of the BP0 register will be loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0110	Read BP1	**Read data from the BP1**. The contents of the BP1 register will be loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-0111	Read BP2	**Read data from the BP2.** The contents of the BP2 register will be loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1000	Read BP3	**Read data from the BP3**. The contents of the BP3 register will be loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1001	Read BP4	**Read data from the BP4.** The contents of the BP4 register will be loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0000-1010	Read BP5	**Read data from the BP5.** The contents of the BP5 register will be loaded into the Debug Shift Register via the ICDB register for host read. This command requires two follow-on transfer cycles with the least significant byte first.
0001-0001	Write ICDC	**Write control data to the ICDC.** The contents of ICDB will be loaded into the ICDC register by the debug engine at the end of the data transfer cycle.
0001-0011	Write ICDA	**Write data to the ICDA**. The contents of ICDB will be loaded into the ICDA register by the debug engine at the end of the data transfer cycles. Data is transferred with the least significant byte first.
0001-0100	Write ICDD	**Write data to the ICDD**. The contents of ICDB will be loaded into the ICDD register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0101	Write BP0	**Write data to the BP0.** The contents of ICDB will be loaded into the BP0 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0110	Write BP1	**Write data to the BP1**. The contents of ICDB will be loaded into the BP1 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-0111	Write BP2	**Write data to the BP2**. The contents of ICDB will be loaded into the BP2 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1000	Write BP3	**Write data to the BP3.** The contents of ICDB will be loaded into the BP3 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1001	Write BP4	**Write data to the BP4.** The contents of ICDB will be loaded into the BP4 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1010	Write BP5	**Write data to the BP5.** The contents of ICDB will be loaded into the BP5 register by the debug engine at the end of data transfer cycles. Data is transferred with the least significant byte first.
0001-1111	Debug	**Debug command.** This command forces the debug engine into debug mode and halts the CPU operation at the completion of the current instruction after the debug command is recognized by the debug engine.

21.1.1 – Breakpoint Registers

The DS4830A incorporates six breakpoint registers (BP0–BP5) that are configurable by the host for establishing different types of breakpoint mechanisms. The first four breakpoint registers (BP0–BP3) are 16-bit registers that are configurable as program memory address breakpoints. When enabled, the debug engine will force a break when a match between the breakpoint register and the program memory execution address occurs. The final two 16-bit breakpoint registers (BP4, BP5) are configurable in one of two possible capacities. They may be configured as data memory address breakpoints or may be configured to support register access breakpoints. In either case, if breakpoints are enabled and the defined breakpoint match occurs, the debug engine will generate a break condition. The six breakpoint registers are documented below.

21.1.1.1 – Breakpoint 0 Register (BP0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	BP0[15:0]															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

s = special

The Breakpoint 0 register is accessible only via background mode read/write commands. Breakpoint registers BP0, BP1, BP2, and BP3 serve as program memory address breakpoints. When DME bit is set in background mode, the debug engine monitors the program-address bus activity while the CPU is executing the user program. If an address match is detected, a break occurs, allowing the debug engine to take control of the CPU and enter debug mode.

21.1.1.2 – Breakpoint 1 Register (BP1)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	BP1[15:0]															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

s = special

The Breakpoint 1 register is accessible only via background mode read/write commands. Breakpoint registers BP0, BP1, BP2, and BP3 serve as program memory address breakpoints. When DME bit is set in background mode, the debug engine monitors the program-address bus activity while the CPU is executing the user program. If an address match is detected, a break occurs, allowing the debug engine to take control of the CPU and enter debug mode.

21.1.1.3 – Breakpoint 2 Register (BP2)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	BP2[15:0]															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

s = special

The Breakpoint 2 register is accessible only via background mode read/write commands. Breakpoint registers BP0, BP1, BP2, and BP3 serve as program memory address breakpoints. When DME bit is set in background mode, the debug engine monitors the program-address bus activity while the CPU is executing the user program. If an address match is detected, a break occurs, allowing the debug engine to take control of the CPU and enter debug mode.

21.1.1.4 – Breakpoint 3 Register (BP3)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	BP3[15:0]															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

s = special

The Breakpoint 3 register is accessible only via background mode read/write commands. Breakpoint registers BP0, BP1, BP2, and BP3 serve as program memory address breakpoints. When DME bit is set in background mode, the debug engine monitors the program-address bus activity while the CPU is executing the user program. If an address match is detected, a break occurs, allowing the debug engine to take control of the CPU and enter debug mode.

21.1.1.5 – Breakpoint 4 Register (BP4)

The Breakpoint 4 register is accessible only via background mode read/write commands.

When REGE = 0: This register serves as one of the two data memory address breakpoints. When DME is set in background mode, the debug engine will monitor the data memory address bus activity while the CPU is executing the user program. If an address match is detected, a break occurs, allowing the debug engine to take over control of the CPU and enter debug mode.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	BP4[15:0]															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

s = special

When REGE = 1: This register serves as one of the two register breakpoints. A break occurs when the destination register address for the executed instruction matches with the specified module and index. The destination module is indicated by the M[3:0] bits and the register within that module is defined by the r[4:0] bits.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	r.4	r.3	r.2	r.1	r.0	M.3	M.2	M.1	M.0
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

s = special

21.1.1.6 – Breakpoint 5 Register (BP5)

The Breakpoint 5 register is accessible only via background mode read/write commands.

When REGE = 0: This register serves as one of the two data memory address breakpoints. When DME is set in background mode, the debug engine will monitor the data memory address bus activity while the CPU is executing the user program. If an address match is detected, a break occurs, allowing the debug engine to take over control of the CPU and enter debug mode.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	BP5[15:0]															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s*	s*	s*	s*	s*	s**	s**	s**	s**

s = special

When REGE = 1: This register serves as one of the two register breakpoints. The destination module is indicated by the M[3:0] bits and the register within that module is defined by the r[4:0] bits. A break occurs when two following conditions are met:

- The destination register address for the executed instruction matches with the specified module and index.
- The bit pattern written to the destination register matches those bits specified for comparison by the ICDD data register and ICDA mask register. Only those ICDD data bits with their corresponding ICDA mask bits will be compared. When all bits in the ICDA register are cleared, Condition 2 becomes a don't care.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	r.4	r.3	r.2	r.1	r.0	M.3	M.2	M.1	M.0
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Access	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

s = special

21.1.2 – Using Breakpoints

All breakpoint registers (BP0-BP5) default to the FFFFh state on power-on reset or when the Test-Logic-Reset TAP state is entered. The breakpoint registers are accessible only with Background mode read/write commands issued over the TAP communication link. The breakpoint registers are not read/write accessible to the CPU.

Setting the Debug Mode Enable (DME) bit in the ICDC register to logic 1 enables all six breakpoint registers for breakpoint match comparison. The state of the Break-On Register Enable (REGE) bit in the ICDC register determines whether the BP4 and BP5 breakpoints should be used as data memory address breakpoints (REGE = 0) or as register breakpoints (REGE = 1).

When using the register matching breakpoints, it is important to realize that Debug mode operations (e.g., read data memory, write data memory, etc.) require use of ICDA and ICDD for passing of information between the host and DS4830A ROM routines. It is advised that these registers be saved and restored or be reconfigured before returning to the background mode if register breakpoints are to remain enabled.

When a breakpoint match occurs, the debug engine forces a break and the DS4830A enters Debug Mode. If a breakpoint match occurs on an instruction that activates the PFX register, the break is held off until the prefixed operation completes. The host can assess whether Debug mode has been entered by monitoring the status bits of the 10-bit word shifted out of the TDO pin. The status bits will change from the Non-debug (00b) state associated with background mode to the Debug-Idle (01b) state when Debug Mode is entered. Debug mode can also be manually invoked by host issuance of the 'Debug' background command.

21.2 – Debug Mode

There are two ways to enter the Debug Mode from Background Mode:

1. Issuance of the Debug command directly by the host via the TAP communication port, or
2. Breakpoint matching mechanism.

The host can issue the Debug background command to the debug engine. This direct Debug Mode entry is nondeterministic. The response time varies dependent on system conditions when the command is issued. The breakpoint mechanism provides a more controllable response, but requires that the breakpoints be initially configured in Background mode. No matter the method of entry, the debug engine takes control of the CPU in the same manner. Debug mode entry is similar to the state machine flow of an interrupt except that the target execution address is x8010h which resides in the Utility ROM instead of the address specified by the IV register that is used for interrupts. On debug mode entry, the following actions occur:

1. block the next instruction fetch from program memory
2. push the return address onto the stack
3. set the contents of IP to x8010h
4. clear the IGE bit to 0 to disable interrupt handler if it is not already clear.
5. halt CPU operation

Once in Debug mode, further breakpoint matches or host issuance of the Debug command are treated as no operations and will not disturb debug engine operation. Entering debug mode also stops the clocks to all timers, including the Watchdog Timer. Temporarily disabling these functions allows debug mode operations without disrupting the relationship between the original user program code and hardware timed functions. No interrupt request can be granted since the interrupt handler is also halted as a result of IGE = 0.

21.2.1 – Debug Mode Commands

The debug engine sets the data shift register status bits to 01b (debug-idle) to indicate that it is ready to accept debug commands from the host.

The host can perform the following operations from debug mode:

- read register map
- read program stack
- read/write register
- read/write data memory
- single step of CPU (trace)
- return to background mode
- unlock password

The only operations directly controlled by the debug engine are single step and return. All other operations are assisted by debug service routines contained in the Utility ROM. These operations require that multiple bytes be transmitted and/or received by the host, however each operation always begins with host transmission of a command byte. This command byte is decoded by the debug engine in order to determine the quantity, sequence, and destination for follow-on bytes received from the host. Even though there is no timing window specified for receiving the complete command and follow-on data, the debug engine must receive the correct number of bytes for a particular command before executing that command. If command and follow-on data are transmitted out of byte order or proper sequence, the only way to resolve this situation is to disable the debug engine by changing the instruction register (IR2:0) and reloading the Debug instruction. Once the debug engine has received the proper number of command and follow-on bytes for a given ROM assisted operation, it will respond with the following actions:

- update the Command bits (CMD3:0) in the ICDC register to reflect the host request,
- enable the ROM if it is not been enabled,
- force a jump to ROM address x8010h, and
- set the data shift register status bits to 10b (debug-busy)

The ROM code performs a read to the ICDC register CMD3:0 bits to determine its course of action. Some commands can be processed by the ROM without receiving data from the host beyond the initially supplied follow-on bytes, while others (e.g., Unlock Password) require additional data from the host. Some commands need only to

provide an indication of completion to the host, while others (e.g., Read Register Map) need to supply multiple bytes of output data. To accomplish data flow control between the host and ROM, the status bits should be used by the host to assess when the ROM is ready for additional data and/or when the ROM is providing valid data output. Internally, the ROM can ascertain when new data is available or when it may output the next data byte via the TXC flag. The TXC flag is an important indicator between the debug engine and the Utility ROM debug routines. The Utility ROM firmware sets the TXC flag to 1 to indicate that valid data has been loaded to the ICDB register. The debug engine clears the TXC flag to 0 to indicate completion of a data shift cycle, thus allowing the ROM to continue execution of a requested task that is still in progress. The Utility ROM signals that it has completed a requested task by setting the ROM Operation Done (ROD) bit of the SC register to logic 1. The ROD bit is reset by the debug engine when it recognizes the done condition.

Table 21-3 shows the debug mode commands supported by the DS4830A. Note that background mode commands are supported inside debug mode, however, the documentation of these commands can be found in the Background mode section of the document. Encodings not listed in this table are not supported in debug mode and are treated as no operations.

Table 21-3: Debug Mode Commands

OPCODE	COMMAND	OPERATION
0010-0000	No Operation	**No Operation.**
0010-0001	Read register Map	**Read data from internal registers.** This command forces the debug engine to update the CMD3:0 bits in the ICDC to 0001b and perform a jump to ROM code at x8010h. The ROM debug service routine will load register data to ICDB for host capture/read, starting at the lowest register location in module 0, one byte at a time in a successive order until all internal registers are read and output to the host.
0010-0010	Read data memory	**Read data from data memory.** This command requires four follow-on transfer cycles, two for the starting address and two for the word read count, starting with the LSB address and ending with the MSB read count. The input address must be based memory map when executing from utility ROM, as shown in Figure 2-4. The address is moved to the ICDA register and the word read count is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:0 bits to 0010b and performs a jump to ROM code at x8010h. The ROM debug service routine will load ICDB from data memory according to address and count information provided by the host.
0010-0011	Read program stack	**Read data from program stack**. This command requires four follow-on transfer cycles, two for the starting address and two for the read count, starting with the LSB address and ending with the MSB read count. The address is moved to the ICDA register and the read count is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:0 bits to 0011b and performs a jump to ROM code at x8010h. The ROM Debug service routine will pop data out from the stack according to the information received in the ICDA and ICDD register. The address input is the highest value that is used, as words are popped off the stack and returned in descending order.
0010-0100	Write register	**Write data to a selected register.** This command requires four follow-on transfer cycles, two for the register address and two for the data, starting with the LSB address and ending with the MSB data. The address is moved to the ICDA register and the data is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:0 bits to 0100b and performs a jump to ROM code at x8010h. The ROM Debug service routine will update the select register according to the information received in the ICDA and ICDD registers. Any register location can be written using this command, including reserved locations and those used for opcode support. No protection is provided by the debugging interface, and avoiding side effects is the responsibility of the host system communicating with the DS4830A. Writing to the IP register alters the address that execution resumes from when the debugging engine exits.

OPCODE	COMMAND	OPERATION
0010-0101	Write data memory	**Write data to a selected data memory location.** This command requires four follow-on transfer cycles, two for the memory address and two for the data, starting with the LSB address and ending with the MSB data. The input address must be based memory map when executing from utility ROM, as shown in Figure 2-4. The address is moved to the ICDA register and the data is moved to the ICDD register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:0 bits to 0101b and performs a jump to ROM code at x8010h. The ROM Debug service routine will update the selected data memory location according to the information received in the ICDA and ICDD registers.
0010-0110	Trace	**Trace command.** This command allows single stepping the CPU and requires no follow-on transfer cycle. The trace operation is a 'debug mode exit, one cycle CPU execution, debug mode entry' sequence.
0010-0111	Return	**Return command.** This command terminates the debug mode and returns the debug engine to background mode. This allows the CPU to resume its normal operation at the point where it has been last interrupted.
0010-1000	Unlock password	**Unlock the password lock.** This command requires 32 follow-on transfer cycles each containing a byte value to be compared with the program memory password for the purpose of clearing the PWL bit and granting access to protected debug and loader functions. When this command is received, the debug engine updates the CMD3:0 bit to 1000b and performs a jump to ROM code at x8010h. Data is loaded to the ICDB register when each byte of data is received, beginning with the LSB of the least significant word first and end with the MSB of the most significant word.
0010-1001	Read register	**Read from a selected internal register.** This command requires two follow-on transfer cycles, starting with the LSB address and ending with the MSB address. The address is moved to ICDA register by the debug engine. This information is directly accessible by the ROM code. At the completion of this command period, the debug engine updates the CMD3:0 bits to 1001b and performs a jump to ROM code at x8010h. The ROM Debug service routine will always assume a 16-bit register length and return the requested data LSB first. Reading a register through the debug interface returns the value that was in that register before the debugging engine was invoked. An exception to this rule is the SP register; reading the SP register through the debug interface actually returns the value (SP+1).

21.2.2 – Read Register Map Command Host-ROM Interaction

A read register map command reads out data contents for all implemented system and peripheral registers. The host does not specify a target register but instead should expect register data output in successive order, starting with the lowest order register in register module 0. Data is loaded by the ROM to the 8-bit ICDB register and is output one byte per transfer cycle. Thus, for a 16-bit register, two transfer cycles are necessary. The host initiates each transfer cycle to shift out the data bytes and will find valid data output tagged with a debug-valid (status = 11b). At the end of each transfer cycle, the debug engine clears the TXC flag to signal the ROM service routine that another byte may be loaded to ICDB. The ROM service routine sets the TXC flag each time after loading data to the ICDB register. This process is repeated until all registers have been read and output to the host. The host system recognizes the completion of the register read when the status debug-idle is presented. This indicates that the debug engine is ready for another operation.

This command outputs all peripheral registers in the range M0[00h] to M5[17h], along with a fixed set of system registers. The following formatting rules apply to the returned data:

- All peripheral registers are output as 16 bits, least significant byte first. If the register is an 8-bit register, the top is returned as 00h.
- System registers are output as 8 bits or 16 bits, least significant byte first.
- Registers I2CBUF_S, I2CBUF_M, SPIB_M, SPIB_S, QTDATA, PWMDATA and ADDATA are not read. Their values are returned as 0000h.
- Nonimplemented and reserved peripheral registers in the range M0[00h] to M5[17h] are represented as empty word values in Table 21-4. These values should be ignored.

The first byte output by this command is the value 180 (B4h), which represents the number of words output for peripheral register. There are a total of 216 words that are output by this command. Table 21-4 lists all of the registers output and the order in which they are output.

Table 21-4: Output from Read Register Map Command

WORD	REGISTER	WORD	REGISTER	WORD	REGISTER	WORD	REGISTER	WORD	REGISTER	WORD	REGISTER		WORD	REGISTER
0	PO2	32		64		96	MCNT	128	ADCN	160			192	A[3]
1	PO1	33	I2CST_M	65	I2CST_S	97	MA	129	SENR	161	QTCN		193	A[4]
2	PO0	34	I2CIE_M	66	MPNTR	98	MB	130	ADST	162	LTIL		194	A[5]
3	EIF2	35	PO6	67	I2CTXFST	99	MC2	131	ADST1	163	HTIL		195	A6[]
4	EIF1	36	CRC8IN	68	I2CTXFIE	100	MC1	132		164			196	A[7]
5	EIF0	37	MIIR1	69	I2CRXFST	101	MC0	133		165			197	A[8]
6	GTV1	38	EIF6	70	I2CRXFIE	102	GTCN2	134	DADDR	166	PWMCN		198	A[9]
7	GTCN1	39	EIE6	71	I2CST2_S	103	SHFT	135	MIIR4	167	PWMSYNC		199	A[10]
8	PI2	40	PI6	72	RPNTR	104	MC1R	136	TEMPCN	168	LTIH		200	A[11]
9	PI1	41	SVM	73		105	MC0R	137	SHCN	169	HTIH		201	A[12]
10	PI0	42	-	74		106	GTC2	138	ADMIS	170	QTLST		202	A[13]
11	GTC1	43	-	75		107	GTV2	139	PINSEL	171			203	A[14]
12		44	I2CCN_M	76	I2CSLA_S	108	GR_REG1	140	REFAVG	172			204	A[15]
13	EIE2	45	I2CCK_M	77	I2CSLA2_S	109	GR_REG2	141		173			205	IP
14	EIE1	46	I2CTO_M	78	I2CSLA3_S	110	MACRSEL	142	TWR	174	MIIR5		206	SP
15	EIE0	47	I2CSLA_M	79	I2CSLA4_S	111	USER_INT	143	RPCFG	175			207	IV
16	PD2	48	EIES6	80	I2CIE2_S	112	GR_REG3	144	SPICN_S	176			208	LC[0]
17	PD1	49	PD6	81	MADDR	113	GR_REG4	145	SPICF_S	177			209	LC[1]
18	PD0	50		82	MADDR2	114	GR_REG5	146	SPICK_S	178	SPICN_M		210	OFFS
19	EIES2	51		83	MADDR3	115	GR_REG6	147	I2C_SPB	179	SPICF_M		211	DPC
20	EIES1	52		84	MADDR4	116	GR_REG7	148	DEV_NUM	180	SPICK_M		212	GR
21	EIES0	53	CRC8OUT	85	CUR_SLA	117	GR_REG8	149	DACD0	181			213	BP
22		54		86	I2CIE_S	118	GR_REG9	150	DACD1	182			214	DP[0]
23		55	ADCG1	87		119	GR_REG10	151	DACD2	183			215	DP[1]
24		56	ADCG2	88	ICDT0	120	GR_REG11	152	DACD3	184	AP	APC		
25		57	ADVOFF	89	ICDT1	121	GR_REG12	153	DACD4	185	PSF	IC		
26		58		90	ICDC	122	GR_REG13	154	DACD5	186	IMR	SC		
27		59	ADCG3	91	ICDF	123	GR_REG14	155	DACD6	187	IIR	CKCN		
28		60	ADCG4	92	ICDB	124	GR_REG15	156	DACD7	188	WDCN	0		
29		61	CHIPREV	93	ICDA	125	GR_REG16	157	DACCFG	189	A[0]			
30		62	ICSLA2_M	94	ICDD	126		158	ADADDR	190	A[1]			
31		63		95		127		159		191	A[2]			

21.2.3 – Single Step Operation (Trace)

The debug engine supports single step operation in debug mode by executing a Trace command from the host. The debug engine allows the CPU to return to its normal program execution for one cycle and then forces a debug mode re-entry. The steps for the Trace command are:

1) Set status to 10b (debug-busy)
2) Pop the return address from the stack
3) Set the IGE bit to logic 1 if debug mode was activated when IGE=1.
4) Supply the CPU with an instruction addressed by the return address
5) Stall the CPU at the end of the instruction execution
6) Block the next instruction fetch from program memory
7) Push the return address onto the stack
8) Set the contents of IP to x8010h
9) Clear the IGE bit to 0 to disable the interrupt handler
10) Halt CPU operation
11) Set the status to debug-idle

Note that the trace operation uses a return address from the stack as a legitimate address for program fetching. The host must maintain consistency of program flow during the debug process. The Instruction Pointer is automatically incremented after each trace operation, thus a new return address will be pushed onto the stack before returning the control to the debug engine. Also, note that the interrupt handler is an essential part of the CPU and a pending interrupt could be granted during single step operation since the IGE bit state present on debug mode entry is restored for the single step.

21.2.4 – Return

To terminate the debug mode and return the debug engine to background mode, the host must issue a Return command to the debug engine. This command causes the following actions:

1) Pop the return address from the stack
2) Set the IGE bit to logic 1 if debug mode was activated when IGE=1.
3) Supply the CPU with an instruction addressed by the return address
4) Allow the CPU to execute the normal user program
5) Set the status to 00b (non-debug)

To prevent a possible endless breakpoint matching loop, no break will occur for a breakpoint match on the first instruction after returning from debug mode to background mode. Returning to background mode also enables all internal timer functions.

21.2.5 – Debug Mode Special Considerations

The following are special considerations when using Debug Mode.

- Special caution should be exercised when using the Write Register command on register bits that globally affect system operation (e.g., IGE, STOP). If the write register command is used to invoke stop mode (setting STOP = 1), the $\overline{RST}$ pin may be asserted to reset the debug engine and return to the background mode of operation.
- Single stepping ('Trace') through any IGE bit change operation results in the debug engine overriding the bit change since it retains the IGE bit setting captured when active debug mode was entered.
- Single stepping ('Trace') into an operation that sets STOP = 1 when IGE = 1 effectively allows enabled interrupts normally capable of causing exit from stop mode to do so.
- Single stepping ('Trace') through any memory read instruction that reads from the utility ROM (such as 'move Acc,' @DP[0] with DP[0] set to 8000h) will cause the memory read to return an incorrect value.
- Single stepping ('Trace') cannot be used when executing code from the utility ROM.
- Data memory allocation is important during system development if in-circuit debug is planned. The top 32-byte memory location may be used by the debug service routine during debug mode. The data contents in these locations may be altered and cannot be recovered.
- One available stack location is needed for debug mode. If the stack is full when entering debug mode, the oldest data in the stack will be overwritten.
- Any signal sampling that relies upon the internal system clock (e.g., counter inputs) can be unreliable since the system clock is turned off inside active debug mode between debug mode commands.

21.3 – In-Circuit Debug Peripheral Registers

The following peripheral registers are used to control the in-circuit debug mode of the DS4830A. Addresses of registers are given as "Mx[yy]," where x is the module number (from 0 to 5 decimal) and yy is the register index (from 00h to 1Fh hexadecimal). Fields in the bit definition tables are defined as follows:

- Name: Symbolic names of bits or bit fields in this register.
- Reset: The value of each bit in this register following a standard reset. If this field reads "unchanged," the given bit is unaffected by standard reset. If this field reads "s," the given bit does not have a fixed 0 or 1 reset value because its value is determined by another internal state or external condition.
- POR: If present this field defines the value of each bit in this register following a power-on reset (as opposed to a standard reset). Some bits are unaffected by standard resets and are set/cleared by POR only.
- Access: Bits can be read-only (r) or read/write (rw). Any special restrictions or conditions that could apply when reading or writing this bit are detailed in the bit description.

21.3.1 – In-Circuit Debug Temp 0 Register (ICDT0, M2[18h])

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	ICDT0[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

s = special

This register is read/write accessible by the CPU only in background mode or debug mode. This register is intended for use by the utility ROM routines as temporary storage to save registers that might otherwise have to be placed in the stack.

21.3.2 – In-Circuit Debug Temp 1 Register (ICDT1, M2[19h])

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	ICDT1[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s

s = special

This register is read/write accessible by the CPU only in background mode or debug mode. This register is intended for use by the utility ROM routines as temporary storage to save registers that might otherwise have to be placed in the stack.

21.3.3 – In-Circuit Debug Control Register (ICDC, M2[1Ah])

Bit	7	6	5	4	3	2	1	0
Name	DME	-	REGE	-	CMD3	CMD2	CMD1	CMD0
Reset	0	0	0	0	0	0	0	0
Access	rs	r	rs	r	rs	rs	rs	rs

r = read, s = special

BIT	NAME	DESCRIPTION
7	DME	Debug Mode Enable (DME). When this bit is cleared to 0, background mode commands may be executed, but breakpoints are disabled. When this bit is set to 1, breakpoints are enabled while background mode commands still may be entered. This bit may only be set or cleared from background debug mode. This bit has no meaning for the ROM code.
6	Reserved	Reserved. Do not write to this bit.
5	REGE	Break-On Register Enable. The REGE bit is used to enable the break-on register function. When REGE bit is set to 1, BP4 and BP5 are used as register breakpoints. A break occurs when the content of BP4 is matched with the destination address of the current instruction. For BP5, a break occurs only on a selected data pattern for a selected destination register addressed by BP5. The data pattern is determined by the contents in the ICDA and ICDD register. The REGE bit alone does not enable register breakpoints, but simply changes the manner in which BP4, BP5 are used. The DME bit still must be set to a logic 1 for any breakpoint to occur. This bit has no meaning for the ROM code.
4	Reserved	Reserved. Do not write to this bit.
3:0	CMD3:0	These bits reflect the current host command in debug mode. These bits are set by the debug engine and allow the ROM code to determine the course of action

CMD3:0	Action
0000	No operation
0001	Read register
0010	Read data memory
0011	Read stack memory
0100	Write register
0101	Write data memory
1000	Unlock password
1001	Read selected register
Other	Reserved

21.3.4 – In-Circuit Debug Flag Register (ICDF, M2[1Bh])

Bit	7	6	5	4	3	2	1	0
Name	-	-	-	-	PSS1	PSS0	JTAG_SPE	TXC
Reset	0	0	0	0	0	0	0	0
Access	r	r	r	r	rw	rw	rw	rw

r = read, s = special

BIT	NAME	DESCRIPTION
7:4	Reserved	Reserved. Do not write to these bits.
3:2	PSS[1:0]	Programming Source Select Bits [1:0]. These bits are used to select a programming interface during In-System programming when JTAG_SPE is set to 1, otherwise, the logic values of these bits have no meaning:
1	JTAG_SPE	System Program Enable. The JTAG_SPE bit is used for In-System programming support and its logical state, when read by the CPU, always reflects the logical-OR of the JTAG_SPE bit that is write accessible by the CPU and the SPE bit of the System Programming Buffer (SPB) Register in the TAP Module (which is accessible via JTAG). The logical state of this bit determines the program flow after a reset. When it is set to logic 1, In-System programming will be executed by the Utility ROM. When it is cleared to 0, execution will be transferred to user code if I^2C bootloading is not required. This bit allows read/write access by the CPU and is cleared to 0 only on a power-on reset or Test-Logic-Reset. The JTAG SPE bit will be cleared by hardware when the ROD bit is set. CPU writes to the JTAG_SPE bit (0 or 1) will result in clearing of the PSS[1:0] bits.
0	TXC	Serial Transfer Complete. This bit is set by hardware at the end of a transfer cycle at the TAP communication link. The TXC bit helps the debug engine to recognize host requests, either command or data. This bit is normally set by ROM code to signify or request the sending or receiving of data. The TXC bit is cleared by the debug engine once set. CPU writes to the TXC bit results in clearing of the PSS[1:0] bits.

PSS1	PSS0	Interface/Action
0	0	JTAG
0	1	I^2C Bootloader
1	x	Exit Loader

21.3.5 – In-Circuit Debug Buffer Register (ICDB, M2[1Ch])

Bit	7	6	5	4	3	2	1	0
Name	ICDB[7:0]							
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

This register serves as the parallel holding buffer for the debug shift register of the TAP. Data is read from or written to ICDB for serial communication between the debug routines and the external host.

21.3.6 – In-Circuit Debug Address Register (ICDA, M2[1Dh])

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	ICDA[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

This register is used by the debug engine to store addresses so that ROM code can view that information. This register is also used by the debug engine as a mask register to mask out don't care bits in the ICDD register when BP5 is used as a register breakpoint. When a bit in this register is set to 1, the corresponding bit location in the ICDD register will be compared to the data being written to the destination register to determine if a break should be generated. When a bit in this register is cleared, the corresponding bit in the ICDD register becomes a don't care and is not compared against the data being written. When all bits in this register are cleared, any updated data pattern will cause a break when the BP5 register matches the destination register address of the current instruction.

21.3.7 – In-Circuit Debug Data Register (ICDD, M2[1Eh])

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	ICDD[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

This register is used by the debug engine to store data or read count so that ROM code can view that information. This register is also used by the debug engine as a data register for content matching when BP5 is used as a register breakpoint. In this case, only data bits in this register with their corresponding mask bits in the ICDA register set will be compared with the updated destination data to determine if a break should be generated.

SECTION 22 – IN-SYSTEM PROGRAMMING

The DS4830A contains an internal bootstrap loader utilizing the JTAG or I²C interfaces. As a result, system software can be upgraded in-system, eliminating the need for a costly hardware retrofit when software updates are required. After each device reset, DS4830A ROM code is executed which determines if bootloader operation is desired. Figure 22-1 provides information on how the DS4830A enters into bootloader operation.

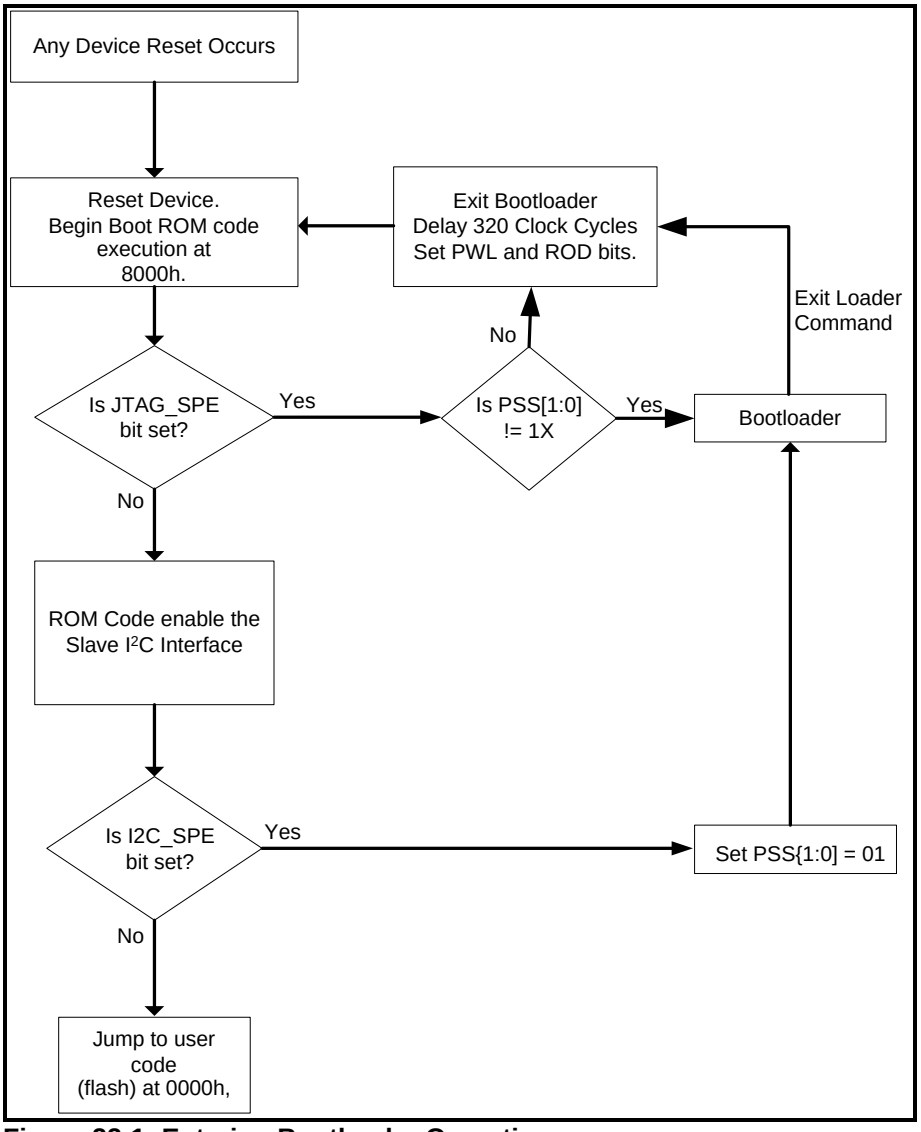


Figure 22-1: Entering Bootloader Operation

22.1 – Detailed Description

Following every reset, device ROM code is executed which determines if the DS4830A should enter into a bootloader mode. First, the ICDF register, which is not cleared by a reset, is read to see if the System Programming Enable (SPE) bit is set. See the *Entering JTAG Bootloader* section for more details on setting the SPE bit. If SPE is set, the DS4830A will enter into bootloader operation.

If SPE is not set, the DS4830A then enables the slave I²C interface. The I2C_SPE bit in the I2C_SPB register is read to determine if I²C bootloader operation is desired. The I2C_SPB register is not cleared by a reset. See the *Entering I²C Bootloader* section for more details on setting the I2C_SPE bit. If I2C_SPE is set, the DS4830A will set the PSS[1:0] bits to 01, which designates I²C bootloader, and enter bootloader operation.

If none of the preceding conditions have been met, the DS4830A ROM code will be complete. The DS4830A will then jump to program memory location 0000h and begin normal program execution.

22.1.1 – Password Protection

The DS4830A uses a password to protect the contents of the program memory from simple access and viewing. The password resides in the 32 bytes of program memory at byte address 0020h through 003Fh. A valid password is defined as any value that does not contain all 0000h or FFFFh. Following a reset, the Password Lock Bit (PWL) in the SC register will be set if the DS4830A contains a valid password.

To protect the program memory, DS4830A grants full access to in-system programming, in-application programming or in-circuit debugging only after a password match has occurred. When a password match occurs, the PWL bit will be cleared to 0. When bootloading the device, the password can be matched using the Password Match command, through either the JTAG or I^2C interface.

22.1.2 – Entering JTAG Bootloader

To enable the Bootstrap loader and establish a desired communication channel via JTAG, the System Programming instruction (100b) must be loaded into the TAP instruction register using the IR-Scan sequence. The TAP retains the System Programming instruction until a new instruction is shifted in or the TAP controller returns to the Test-Logic-Reset state. See Section 16 –Test Access Port for more information regarding the JTAG port.

Once the instruction is latched in the instruction parallel buffer (IR[2:0]) and is recognized by the TAP controller in the Update-IR state, a 3-bit data shift register is activated as the communication channel for DR-Scan sequences. This 3-bit shift register formed between the TDI and TDO pins is directly interfaced to the 3-bit Serial Programming Buffer (SPB). Table 22-1 provides a detailed description of the System Programming Buffer (SPB). The data content of the SPB is reflected in the ICDF register, which allows read and write access by the CPU. These bits are cleared by power-on reset or Test-Logic-Reset of the TAP controller.

Table 22-1: System Programming Buffer (SPB)

<table>
<tr><th>BIT</th><th>NAME</th><th>DESCRIPTION</th></tr>
<tr><td>2:1</td><td>PSS[1:0]</td><td>Programming Source Select (PSS1:PSS0). These bits select the programming interface source.
<table>
<tr><th>PSS1</th><th>PSS0</th><th>Programming Source</th></tr>
<tr><td>0</td><td>0</td><td>JTAG</td></tr>
<tr><td>0</td><td>1</td><td>I^2C</td></tr>
<tr><td>1</td><td>x</td><td>Exit loader</td></tr>
</table></td></tr>
<tr><td>0</td><td>SPE</td><td>System Programming Enable (SPE). Setting this bit to a logic 1 denotes that JTAG bootloading is desired upon exiting reset. The logic state of SPE is examined by the Utility ROM following a reset to determine the program flow. When SPE=1, the Bootstrap Loader selected by the PSS[1:0] bits will be activated to perform a Bootstrap Loader function. If SPE=0, the Utility ROM will determine if I^2C Bootloading is required before transferring execution control to the normal user program.</td></tr>
</table>

Following a reset, if the System Programming Buffer is set for JTAG bootloading, the bootload routine will be entered. The host must now load the Debug instruction (010b) into the TAP instruction register (IR[2:0]), which enables the 10-bit Debug shift register between TDI and TDO. When operating in JTAG bootloader mode, the debug state machines are disabled and the sole purpose of the debug hardware is to simultaneously transfer the data byte shifted in from the host to the In-Circuit Debug Buffer Register (ICDB) and transfer the contents of an internal holding register (loaded by ROM code writes of ICDB) into the shift register for output to the host. The 8 most significant bits of the 10-bit shift register interface directly to the ICDB. The transfer between the shift register and the ICDB register occurs on the falling edge of TCK at the Update-DR state. The debug hardware additionally clears the TXC bit in the ICDF register at this point. The ROM loader code controls the status bit output to the host by asserting TXC=1 when it has valid data to be shifted out.

The 2 least significant bits of the 10-bit shift register are status bits. The JTAG bootloader has the benefit of using the same status bit handshaking hardware that is used for in-circuit debugging. The description of the status bits is described in Table 22-2.

Note: When using the JTAG port, the clock rate (TCK) must be kept below 1/8 of the system clock rate.

Table 22-2: JTAG Bootloader Status Bits

BITS 1:0	STATUS	CONDITION
00	Reserved	Invalid condition.
01	Reserved	Invalid condition
10	Loader-Busy	ROM Loader is busy executing code or processing the current command.
11	Loader-Valid	ROM Loader is supplying valid output data to the host in current shift operation.

22.1.3 – Entering I²C Bootloader

The DS4830A also has built-in functionality that allows bootloading over I²C. Bootloading via I²C allows the system to update the firmware using only the I²C bus without JTAG or firmware intervention. To access the bootloading function, slave address 34h is used. This slave address is setup by hardware and cannot be changed through firmware. As long as the Slave I²C port is enabled, which is the default, the DS4830A will always respond to this slave address without any firmware interaction required. This address should not be used for any purpose other than the special bootloading features. Table 22-3 details the special functions that can be performed using slave address 34h.

Table 22-3: Special Functions of Address 34h

COMMAND BYTE	ACTION
F0h	Sets the I2C_SPE bit in the I2C_SPB register to enable bootloading via I²C. This bit will not be cleared on device reset.
BBh	Executes a reset of the DS4830A when an I²C STOP is received.
All other bytes	The I2C_SPE bit in I2C_SPB is cleared. The DS4830A will NACK this byte.

To enter I²C bootloader, the host must first write slave address 34h with data F0h and then issue a STOP command. When the STOP command is received, the I²C _SPE bit will be set. The DS4830A must then be reset. This can be done using either the $\overline{RST}$ pin or by using the I²C self-reset. To do an I²C self-reset, the host needs to write slave address 34h with data BBh. Upon receiving an I²C STOP, a reset will be performed.

22.1.4 – I²C Bootloader Disable

The DS4830A provides options to disable the bootloader slave address 34h. The device has DEV_NUM register which is cleared only on POR. The bit 7 of the DEV_NUM controls bootloader slave address. Setting DEV_NUM[7] disables the slave address. Application in which bootloader address are not required should set the DEV_NUM[7] in the top of initialization function at the earliest.

22.2 – Bootloader Operation

Once in bootloader mode, the JTAG and I^2C interfaces both use the same commands. How these commands are implemented will be different between the two interfaces. Table 22-4 shows an example command and parameters. The next two sections will detail how to implement these commands using either the JTAG or I^2C interface.

Table 22-4: Example Bootload Command

Byte(s)	Command	Data In	NOP	Data Out	Return
Input	Command	Data In	00h	00h	00h
Output	X	X	X	Data Out	3Eh

Byte Name	Description
Command	All bootloader commands begin with a single command byte. The upper four bits of this command byte define the command family (from 0 to 15) and the lower four bits define the specific command within that family.
Data In	Data bytes that are input to the bootloader that are required for the command. The number of Data In bytes varies for each command. Some commands do not require any Data In bytes.
NOP	The NOP byte is only used for JTAG mode. This is a byte of 00h that is clocked into TDI, while TDO is ignored.
Data Out	Data Out is any data that is returned by the bootloader. The number of Data Out bytes varies for each command. Some commands do not output any Data Out bytes.
Return	A return value of 3Eh is output by the bootloader at the start of first command and following the successful completion of every command thereafter. If the Return byte is read prior to 3Eh being loaded by the bootloader, the read will return the data that is currently in the shift register. The value 3Eh is only loaded into the shift register once. Any subsequent reads will return invalid data. In JTAG bootload mode, status bits will tell when ROM loader is sending valid 3Eh.

22.2.1 – JTAG Bootloader Protocol

The JTAG port consists of a shift register. As data is clocked into TDI, data will be clocked out of TDO. Each "byte" on the JTAG port is actually 10 bits. The two least significant bits are the status bits described in Table 22-2. The data that is input to the device on the TDI pin should have the two status bits set to 0. The following steps are required for each command.

1) Transmit the Command byte on TDI. Ignore the returned data on TDO.
2) Transmit any Data In bytes on TDI. Ignore the returned data on TDO.
3) Transmit the NOP byte of 00h, on TDI. Ignore the returned data on TDO.
4) Possibly poll returned data until command execution completes.
5) Transmit 00h on TDI for each Data Out byte. Read the Data Out byte on TDO.
6) Transmit 00h on TDI and verify that the Return byte output on TDO is 3Eh.

Some of the bootloader commands, such as the erase and CRC commands require extra time to execute. For these commands, the two status bits can be used to verify the state of the bootloader. After issuing any of these commands, the NOP command can continuously be sent to the bootloader. If the returned status bits are 10, the bootloader is still busy processing the command. If the status bits are 11, the bootloader has completed execution of the command. The first byte that was returned with status bits 11 will be the first byte of valid returned data from the bootloader.

22.2.2 – I²C Bootloader Protocol

After entering the I²C bootloader, all I²C communication takes place on the default I²C bootloader slave address 36h. When writing data to the DS4830A, slave address 36h (R/$\overline{W}$ bit = 0) is used. To read data from the DS4830A I²C bootloader, slave address 37h (R/$\overline{W}$ bit = 1) is used. The I²C bootloader does not return the status bits that are available from the JTAG bootloader. The following I²C steps are required to send each command

1) Send an I²C start, followed by writing slave address 36h(R/$\overline{W}$ bit set to write).
2) Write command byte.
3) Write any Data In bytes.
4) The NOP byte is not required for the I²C interface. Sending a NOP byte when using the I²C bootloader will place the bootloader into an unknown state. Instead, an I²C Restart needs to be issued, followed by writing slave address 37h (R/$\overline{W}$ bit set to read).
5) Possibly poll returned data until command execution completes.
6) Read and ACK all Data Out bytes.
7) Read and NACK the Return byte, verify that 3Eh was returned.
8) Send an I²C STOP.

Some of the bootloader commands, such as the erase and CRC commands require extra time to execute. For these commands, the I²C port can be continuously polled to determine when the command completes. This polling is done by reading the returned data bytes after sending slave address 37h. The I²C bootloader will return data B7h while it is currently busy. When data other than B7h is returned, the bootloader is returning valid data. An example of polling for the "Master Erase" command is shown in Figure 22-2. After sending slave address 37h, the I²C bootloader will output B7h until the command has finished execution. The I²C master needs to continue reading and returning ACK's until 3Eh is returned. The master then NACK's this byte (3E).

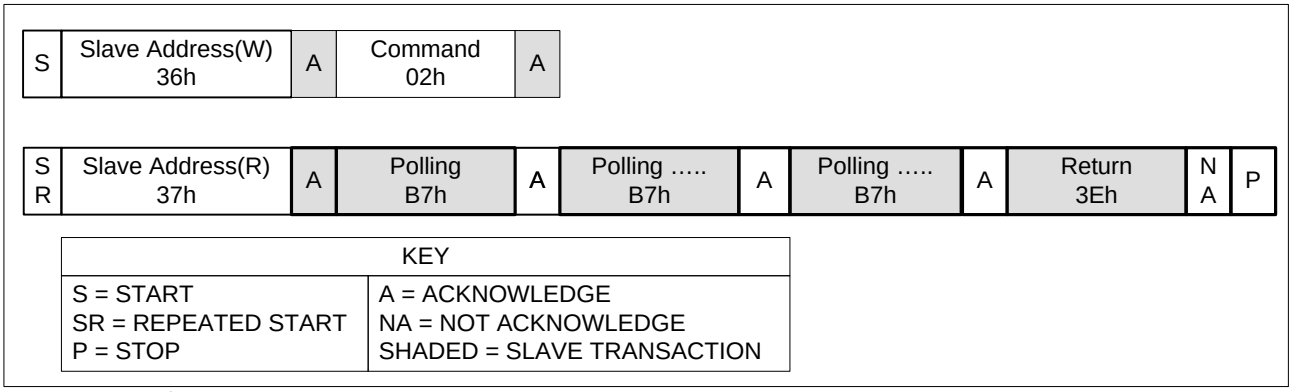


Figure 22-2: I²C Bootloader Polling

Refer to Application Note 5602: In-System Programming Using I²C Bootloader Commands for ISP using the I²C bootloader.

22.3 – Bootloader Commands

Commands for the DS4830A loader are grouped into families. All bootloader commands begin with a single command byte. The upper four bits of this command byte define the command family (from 0 to 15), while the lower four bits define the specific command within that family. The loader command families are shown in Table 22-5.

Table 22-5: Command Families

COMMAND FAMILY	FAMILY DESCRIPTION
0	Required
1	Load
2	Dump
3	CRC
4	Verify
5	Load and Verify
E	Fixed Length Erase

All commands, except those in Family 0, are password protected. The password must first be matched before these commands can be executed. This is done using the Password Match command, which will clear the PWL bit if a match is made.

Bootloader commands that fail for any reason set the bootloader status byte to an error code value describing the reason for the failure. This status byte can be read by means of the Get Status command.

For proper bootloader operation, all bytes of data listed for the command must be written or read from the bootloader. This includes the Return byte, and for the I^2C bootloader, the Dummy RX byte. If all bytes are not read, the bootloader will remain in an unknown state even after a new command is sent to the bootloader.

Following are descriptions of the bootloader commands that are available for use by the DS4830A bootloader.

22.3.1 – Command 00h – No Operation

	Byte 1
	Command
Input	00h
Output	X

This is a No Operation Command. This command can be sent at any time without the bootloader taking action. This command is not password protected.

22.3.2 – Command 01h – Exit Loader

	Byte 1
	Command
Input	01h
Output	X

This command causes the bootloader to exit. When exiting, the bootloader will clear the JTAG_SPE and I2C_SPE bits and then perform an internal reset of the device. Following the reset, code execution jumps to the beginning of application code at address 0000h. This command is not password protected.

22.3.3 – Command 02h – Master Erase

	Byte 1	Byte 2	Byte 3
	Command	**NOP**	**Return**
Input	02h	00h	00h
Output	X	X	3Eh

This command erases (sets to FFFFh) all words in the program flash memory and writes all words in the data SRAM to zero. This command is not password protected. After this command completes, the password lock bit is automatically cleared, allowing access to all bootloader commands. This command requires approximately 40ms to complete. Polling for a return value of 3Eh can be performed during this execution time to determine when the master erase has completed.

22.3.4 – Command 03h – Password Match

	Byte 1	Bytes 2 to 33	Byte 34	Byte 35
	Command	**Data In**	**NOP**	**Return**
Input	03h	32-Byte Password	00h	00h
Output	X	X	X	3Eh

This command accepts a 32-byte password value, which is matched against the password in program memory from byte address 0020h through 003Fh. If the entered value matches the password in program memory, the password lock bit will be cleared. This command is not password protected.

22.3.5 – Command 04h – Get Status

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
	Command	**NOP**	**Data Out**	**Data Out**	**Return**
Input	04h	00h	00h	00h	00h
Output	X	X	Flags	Status Code	3Eh

The Status Flags and Status Code returned by the Get Status command are defined in Tables 22-6 and 22-7. This command is not password protected. The Status Codes will be set whenever an error condition occurs and will only reflect the last error. The Status Codes will be cleared

- When the bootloader is initially entered
- At the start of execution of all commands except Family 0 commands
- At the start of execution of the Family 0 Master Erase.

Table 22-6: Bootloader Status Flags

FLAG BIT	MEANING
8:3	**Reserved.**
2	**Word/Byte Mode Supported.** 0 – The bootloader supports byte mode only. 1 – The bootloader supports word mode as well as byte mode. **(Note: The DS4830A supports byte mode only)**
1	**Word/Byte Mode.** 0 – The bootloader is currently in byte mode for memory reads/writes. 1 – The bootloader is currently in word mode for memory reads/writes. **(Note: The DS4830A supports byte mode only)**
0	**Password Lock.** This bit will match the SC.PWL bit. 0 – The password is unlocked or had a default value; password-protected commands may be used. 1 – The password is locked. Password-protected commands may not be used.

Table 22-7: Bootloader Status Codes

STATUS VALUE	MEANING
00	**No Error.** The last command completed successfully.
01	**Family Not Supported.** An attempt was made to use a command from a family which the bootloader does not support.
02	**Invalid Command.** An attempt was made to use a nonexistent command within a supported command family.
03	**No Password Match.** An attempt was made to use a password-protected command without first matching a valid password. Or, the Password Match command was called with an incorrect password value.
04	**Bad Parameter.** An input parameter passed to the command was out of range or otherwise invalid.
05	**Verify Failed.** The verification step failed on a Load/Verify or Verify command.
06	**Unknown Register.** An attempt was made to read from or write to a nonexistent register.
07	**Word Mode Not Supported.** An attempt was made to set word mode access, but the bootloader supports byte mode access only.
08	**Master Erase Failed.** The bootloader was unable to perform master erase.

22.3.6 – Command 05h – Get Supported Commands

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
	Command	**NOP**	**Data Out**	**Data Out**	**Data Out**	**Data Out**	**Return**
Input	05h	00h	00h	00h	00h	00h	00h
Output	X	X	SupportL	SupportH	00h	00h	3Eh

The SupportL (LSB) and SupportH (MSB) bytes form a 16-bit value that indicates which command families the bootloader supports. If bit 0 is set to 1, it indicates that Family 0 is supported. If bit 1 is set to 1, it indicates that Family 1 is supported. The value returned by the DS4830A is 403Fh, indicating that command families 0, 1, 2, 3, 4, 5 and E are supported. This command is not password protected.

22.3.7 – Command 06h – Get Code Size

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
	Command	**NOP**	**Data Out**	**Data Out**	**Return**
Input	06h	00h	00h	00h	00h
Output	X	X	SizeL	SizeH	3Eh

This command returns SizeH:SizeL, which represents the size of available code memory in words minus 1. The DS4830A will return a value of 7FFFh, which indicates 32k words of program memory are available. This command is not password protected.

22.3.8 – Command 07h – Get Data Size

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
	Command	**NOP**	**Data Out**	**Data Out**	**Return**
Input	07h	00h	00h	00h	00h
Output	X	X	SizeL	SizeH	3Eh

This command returns SizeH:SizeL, which represents the size of available data memory in words minus 1. The DS4830A will return a value of 07FFh, which indicates 2k words of data memory are available. This command is not password protected.

22.3.9 – Command 08h – Get Loader Version

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
	Command	**NOP**	**Data Out**	**Data Out**	**Return**
Input	08h	00h	00h	00h	00h
Output	X	X	VersionL	VersionH	3Eh

This command returns the device's bootloader version. The format of the version is VersionH.VersionL. For example, if VersionL returns 01h and VersionH returns 01h, this corresponds to bootloader version 1.1. This command is not password protected.

22.3.10 – Command 09h – Get Utility ROM Version

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
	Command	**NOP**	**Data Out**	**Data Out**	**Return**
Input	09h	00h	00h	00h	00h
Output	X	X	VersionL	VersionH	3Eh

This command returns the device's ROM code version. The format of the ROM version is VersionH.VersionL. For example, if VersionL returns 00h and VersionH returns 01h, this corresponds to ROM version 1.0. This command is not password protected.

22.3.11 – Command 10h – Load Code

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5	Byte Length+6
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**NOP**	**Return**
Input	10h	Length	AddressL	AddressH	Data to load	00h	00h
Output	X	X	X	X	X	X	3Eh

This command programs (Length) bytes of data into the program flash starting at byte address (AddressH:AddressL). The bootloader writes one 16-bit word to flash at a time. The low bit of the address will always be forced to zero because instructions in program flash are word aligned. If an odd number of bytes are input, the final word written to the program flash will have its most significant byte set to 00h. Memory locations in flash that have been previously loaded must be erased (Master Erase or Page Erase Command) before they can be loaded with a new value. The DS4830A uses a little-endian memory architecture where the least significant byte of each word is loaded first. For example, if you load bytes (11h, 22h, 33h, 44h) starting at address 0000h, the first two words of program space will be written to 2211h, 4433h. This command is password protected.

The time required to write 1 word of data to flash is approximately 80µs. To guarantee correct programming, a bootloading program will need to ensure that there is at least 100µs of time between when the bootloader receives two words of data. The easiest way to do this is to limit the clock rate to 100kHz. The time to transmit one word of data with a 100kHz clock exceeds 100µs, thus giving the previously transmitted word time to be programmed into flash prior to processing the next word. If a faster clock rate is used, delays will need to be added to ensure that words are not transmitting at rates faster than 100µs.

The JTAG bootloader also supports polling using the status bits as a method to determine when a word has successfully been written into flash. When sending the first two bytes of program data to load, the status bits should return as 11 to signify that the bootloader is valid. After sending the 2nd byte, the bootloader will begin writing this first word to flash and will be busy. If a 3rd byte of data is written while the bootloader is busy programming the first word, the status bits will return as 10, which is loader busy. Upon receiving a status of 10, the 3rd byte needs to be sent again until the status bits return as 11, or loader valid. When this code is returned the 3rd byte has been received and the 4th byte can now be sent. If using the JTAG bootloader with a clock faster than 100kHz, this polling method should be used for every byte that is transmit to the bootloader.

22.3.12 – Command 11h – Load Data

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5	Byte Length+6
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**NOP**	**Return**
Input	11h	Length	AddressL	AddressH	Data to load	00h	00h
Output	X	X	X	X	X	X	3Eh

This command writes (Length) bytes of data into the data SRAM starting at byte address (AddressH:AddressL). The DS4830A uses a little-endian memory architecture where the least significant byte of each word is loaded first. For example, if you load bytes (11h, 22h, 33h, 44h) starting at address 0000h, the first two words of memory space will be written to 2211h, 4433h. This command is password protected.

22.3.13 – Command 20h – Dump Code

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 5	Byte 6	Length Bytes	Byte Length+7
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Data In**	**NOP**	**Data Out**	**Return**
Input	20h	2	AddrL	AddrH	LengthL	LengthH	00h	00h	00h
Output	X	X	X	X	X	X	X	Memory	3Eh

This command returns the contents of the program flash memory. The memory dump begins at byte address AddrH:AddrL and will contain LengthH:LengthL bytes. This command is password protected.

22.3.14 – Command 21h – Dump Data

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 5	Byte 6	Length Bytes	Byte Length+7
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Data In**	**NOP**	**Data Out**	**Return**
Input	21h	2	AddrL	AddrH	LengthL	LengthH	00h	00h	00h
Output	X	X	X	X	X	X	X	Memory	3Eh

This command returns the contents of the SRAM memory. The memory dump begins at byte address AddrH:AddrL and will contain LengthH:LengthL bytes. This command is password protected.

22.3.15 – Command 30h – CRC Code

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Data In**	**NOP**	**Data Out**	**Data Out**	**Return**
Input	30h	2	AddrL	AddrH	LengthL	LengthH	00h	00h	00h	00h
Output	X	X	X	X	X	X	X	CRCL	CRCH	3Eh

This command returns the CRC-16 value (CRCH:CRCL) of the (LengthH:LengthL) bytes of program flash starting at (AddrH:AddrL). The formula for the CRC calculation is $X^{16} + X^{15} + X^2 + 1$. This command is password protected.

The CRC calculation takes approximately 45 system clock cycles per byte (4.5µs/byte). During this time polling should be performed to determine when the loader has finished executing the CRC calculation. If using the I^2C loader, user should wait for time according to given length and read CRCL, CRCH, 3Eh. If using the JTAG loader, the JTAG status bits can be used to determine when the CRC calculation is complete.

22.3.16 – Command 31h – CRC Data

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Data In**	**NOP**	**Data Out**	**Data Out**	**Return**
Input	31h	2	AddrL	AddrH	LengthL	LengthH	00h	00h	00h	00h
Output	X	X	X	X	X	X	X	CRCL	CRCH	3Eh

This command returns the CRC-16 value (CRCH:CRCL) of the (LengthH:LengthL) bytes of data memory starting at (AddrH:AddrL). The formula for the CRC calculation is $X^{16} + X^{15} + X^2 + 1$. This command is password protected.

The CRC calculation takes approximately 45 system clock cycles per byte (4.5µs/byte). During this time polling should be performed to determine when the loader has finished executing the CRC calculation. If using the I^2C loader, user should wait for time according to given length and read CRCL, CRCH, 3Eh. If using the JTAG loader, the JTAG status bits can be used to determine when the CRC calculation is complete.

22.3.17 – Command 40h – Verify Code

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5	Byte Length+6
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**NOP**	**Return**
Input	40h	Length	AddrL	AddrH	Data to Verify	00h	00h
Output	X	X	X	X	X	X	3Eh

This command operates in the same manner as the Load Code command, except that instead of programming the input data into flash memory, it verifies that the input data matches the data already in code space. If the data does not match, the status code is set to reflect this failure. This command is password protected.

22.3.18 – Command 41h – Verify Data

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5	Byte Length+6
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**NOP**	**Return**
Input	41h	Length	AddrL	AddrH	Data to Verify	00h	00h
Output	X	X	X	X	X	X	3Eh

This command operates in the same manner as the Load Data command, except that instead of writing the input data into SRAM, it verifies that the input data matches the data already in data space. If the data does not match, the status code is set to reflect this failure. This command is password protected.

22.3.19 – Command 50h – Load and Verify Code

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5	Byte Length+6
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**NOP**	**Return**
Input	50h	Length	AddrL	AddrH	Data to load and verify	00h	00h
Output	X	X	X	X	X	X	3Eh

This command provides the combined functionality of the Load Code and Verify Data commands. After each word of data is written to data memory, the loader will read this memory location and verify that the data matches the input data. If the verification fails, the status code will be set to reflect this failure. All the guidelines that are listed for the Load Code command must be followed for the Load and Verify Code command. This command is password protected.

22.3.20 – Command 51h – Load and Verify Data

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5	Byte Length+6
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**NOP**	**Return**
Input	51h	Length	AddrL	AddrH	Data to load and verify	00h	00h
Output	X	X	X	X	X	X	3Eh

This command provides the combined functionality of the Load Data and Verify Data commands. After each word of data is written to SRAM memory, the loader will read this memory location and verify that the data matches the input data. If the verification fails, the status code will be set to reflect this failure. The guidelines that are listed for the Load Data command must be followed for the Load and Verify Data command. This command is password protected.

22.3.21 – Command E0h – Code Page Erase

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
	Command	**Data In**	**Data In**	**Data In**	**NOP**	**Return**
Input	E0h	0	PageNum	0	00h	00h
Output	X	X	X	X	X	3Eh

This command erases (programs to FFFFh) all words in a 256 word (512 byte) page of the program flash memory. The DS4830A has 128 pages of flash. The input PageNum indicates which page to erase. For example, PageNum=0 would erase byte addresses 000h through 1FFh and PageNum=1 would erase byte addresses 200h through 3FFh. This command requires approximately 26ms to complete. Polling can be performed during this execution time to determine when the page erase has completed. This command is password protected.

SECTION 23 – PROGRAMMING

The following section provides a programming overview of the DS4830A. For full details on the instruction set, as well as System Register and Peripheral Register detailed bit descriptions, see the appropriate sections in this user's guide.

23.1 – Addressing Modes

The instruction set for the DS4830A provides three different addressing modes: direct, indirect and immediate.

The direct addressing mode can be used to specify either source or destination registers, such as:

```
move  A[0], A[1]          ; copy accumulator 1 to accumulator 0
push  A[0]                ; push accumulator 0 on the stack
add   A[1]                ; add accumulator 1 to the active accumulator
```

Direct addressing is also used to specify addressable bits within registers.

```
move  C, Acc.0            ; copy bit zero of the active accumulator to the carry flag
move  PO0.3, #1           ; set bit three of port 0 Output register
```

Indirect addressing, in which a register contains a source or destination address, is used only in a few cases.

```
move @DP[0], A[0]         ; copy accumulator 0 to the data memory location pointed to by data pointer 0
move  A[0], @SP--         ; where @SP-- is used to pop the data pointed to by the stack pointer register
```

Immediate addressing is used to provide values to be directly loaded into registers or used as operands.

```
move  A[0], #10h          ; set accumulator 1 to 10h/16d
```

23.2 – Prefixing Operations

All instructions on the DS4830A are 16 bits long and execute in a single cycle. However, some operations require more data than can be specified in a single cycle or require that high order register index bits be set to achieve the desired transfer. In these cases, the prefix register module PFX is loaded with temporary data and/or required register index bits to be used by the following instruction. The PFX module only holds loaded data for a single cycle before it clears to zero.

Instruction prefixing is required for the following operations, which effectively makes them two-cycle operations.

- When providing a 16-bit immediate value for an operation (e.g. loading a 16-bit register, ALU operation, supplying an absolute program branch destination), the PFX module must be loaded in the previous cycle with the high byte of the 16-bit immediate value unless that high byte is zero. One exception to this rule is when supplying an absolute branch destination to 0023h. In this case, PFX still must be written with 00h. Otherwise, the branch instruction would be considered a relative one instead of the desired absolute branch.
- When selecting registers with indexes greater than 07h within a module as destinations for a transfer or registers with indexes greater than 0Fh within a module as sources, the PFX[n] register must be loaded in the previous cycle. This can be combined with the previous item.

Generally, prefixing operations can be inserted automatically by the assembler as needed, so that (for example)

```
move  DP[0], #1234h
```

actually assembles as

```
move  PFX[0], #12h
move  DP[0], #34h
```

However, the operation

```
move  DP[0], #0055h
```

does not require a prefixing operation even though the register DP[0] is 16-bit. This is because the prefix value defaults to zero, so the line `move  PFX[0], #00h` is not required.

23.3 – Reading and Writing Registers

All functions in the DS4830A are accessed through registers, either directly or indirectly. This section discusses loading registers with immediate values and transferring values between registers of the same size and different sizes.

23.3.1 – Loading an 8-Bit Register with an Immediate Value

Any writeable 8-bit register with a sub-index from 0h to 7h within its module can be loaded with an immediate value in a single cycle using the MOVE instruction.

```
move  AP, #05h              ; load accumulator pointer register with 5 hex
```

Writeable 8-bit registers with sub-indexes 8h and higher can be loaded with an immediate value using MOVE as well, but an additional cycle is required to set the prefix value for the destination.

```
move  WDCN, #33h            ; assembles to:  move PFX[2], #00h
                            ;                move (WDCN-80h), #33h
```

23.3.2 – Loading a 16-Bit Register with a 16-Bit Immediate Value

Any writeable 16-bit register with a sub-index from 0h to 07h can be loaded with an immediate value in a single cycle if the high byte of that immediate value is zero.

```
move  LC[0], #0010h         ; prefix defaults to zero for high byte
```

If the high byte of that immediate value is not zero or if the 16-bit destination sub-index is greater than 7h, an extra cycle is required to load the prefix value for the high byte and/or the high order register index bits.

```
                            ; high byte <> #00h
move  LC[0], #0110h         ; assembles to:  move PFX[0], #01h
                            ;                move LC[0], #10h

                            ; destination sub-index > 7h
move  A[8], #0034h          ; assembles to:  move PFX[2], #00h
                            ;                move (A[8]-80h), #34h
```

23.3.3 – Moving Values Between Registers of the Same Size

Moving data between same-size registers can be done in a single-cycle MOVE if the destination register's index is from 0h to 7h and the source register index is between 0h and Fh.

```
move  A[0], A[8]            ; copy accumulator 8 to accumulator 0
move  LC[0], LC[1]          ; copy loop counter 1 to loop counter 0
```

If the destination register's index is greater than 7h or if the source register index is greater than Fh, prefixing is required.

```
move  A[15], A[0]           ; assembles to:  move PFX[2], #00h
                            ;                move (A[15]-80h), A[0]
```

23.3.4 – Moving Values Between Registers of Different Sizes

Before covering some transfer scenarios that might arise, a special register must be introduced that will be utilized in many of these cases. The 16-bit General Register (GR) is expressly provided for performing byte singulation of 16-bit words. The high and low bytes of GR are individually accessible in the GRH and GRL registers respectively. A read-only GRS register makes a byte-swapped version of GR accessible and the GRXL register provides a sign-extended version of GRL.

8-Bit Destination ← Low Byte (16-Bit Source)

The simplest transfer possibility would be loading an 8-bit register with the low byte of a 16-bit register. This transfer does not require use of GR and requires a prefix only if the destination or source register are outside of the single cycle write or read regions, 0-7h and 0-Fh, respectively.

```
move  OFFS, LC[0]           ; copy the low byte of LC[0] to the OFFS register
move  IMR, @DP[1]           ; copy the low byte @DP[1] to the IMR register
move  WDCN, LC[0]           ; assembles to: move PFX[2], #00h
                            ;               move (WDCON-80h), LC[0]
```

8-Bit Destination ← High Byte (16-Bit Source)

If, however, we needed to load an 8-bit register with the high byte of a 16-bit source, it would be best to use the GR register. Transferring the 16-bit source to the GR register adds a single cycle.

```
move  GR, LC[0]          ; move LC[0] to the GR register
move  IC, GRH            ; copy the high byte into the IC register
```

16-Bit Destination ← Concatenation (8-Bit Source, 8-Bit Source)

Two 8-bit source registers can be concatenated and stored into a 16-bit destination by using the prefix register to hold the high order byte for the concatenated transfer. An additional cycle may be required if either source byte register index is greater than 0Fh.

```
move  PFX[0], IC         ; load high order source byte IC into PFX
move  @++SP, AP          ; store @DP[0] the concatenation of IC:AP

                         ; 16-bit destination sub-index: dst=08h
                         ;  8-bit source sub-indexes:
                         ;  high=10h, low=11h
move  PFX[1], #00h       ;
move  PFX[3], high       ; PFX=00:high
move  dst, low           ; dst=high:low
```

Low (16-Bit Destination) ← 8-Bit Source

To modify only the low byte of a given 16-bit destination, the 16-bit register should be moved into the GR register such that the high byte can be singulated and the low byte written exclusively. An additional cycle is required if the destination index is greater than 0Fh.

```
move  GR, DP[0]          ; move DP[0] to the GR register
move  PFX[0], GRH        ; get the high byte of DP[0] via GRH
move  DP[0], #20h        ; store the new DP[0] value

                         ; 16-bit destination sub-index: dst=10h
                         ;  8-bit source sub-index: src=11h
move  PFX[1], #00h       ;
move  GR, dst            ; read dst word to the GR register
move  PFX[5], GRH        ; get the high byte of dst via GRH
move  dst, src           ; store the new dst value
```

High (16-Bit Destination) ← 8-Bit Source

To modify only the high byte of a given 16-bit destination, the 16-bit register should be moved into the GR register such that the low byte can be singulated and the high byte can be written exclusively. Additional cycles are required if the destination index is greater than 0Fh or if the source index is greater than 0Fh.

```
move  GR, DP[0]          ; move DP[0] to the GR register
move  PFX[0], #20h       ; get the high byte of DP[0] via GRH
move  DP[0], GRL         ; store the new DP[0] value

                         ; 16-bit destination sub-index: dst=10h
                         ;  8-bit source sub-index: src=11h
move  PFX[1], #00h       ;
move  GR, dst            ; read dst word to the GR register
move  PFX[1], #00h
move  PFX[4], src        ; get the new src byte
move  dst, GRL           ; store the new dst value
```

If the high byte needs to be cleared to 00h, the operation can be shortened by transferring only the GRL byte to the 16-bit destination (example follows):

```
move  GR, DP[0]          ; move DP[0] to the GR register
move  DP[0], GRL         ; store the new DP[0] value, 00h used for high byte
```

23.4 – Reading and Writing Register Bits

The MOVE instruction can also be used to directly set or clear any one of the lowest 8 bits of a peripheral register in module 0h-5h or a system register in module 8h. The set or clear operation will not affect the upper byte of a 16-bit register that is the target of the set or clear operation. If a set or clear instruction is used on a destination register that does not support this type of operation, the register high byte will be written with the prefix data and the low byte will be written with the bit mask (i.e. all 0's with a single 1 for the set bit operation or all ones with a single 0 for the clear bit operation).

Register bits may be set or cleared individually using the MOVE instruction as follows.

```
move  IGE, #1              ; set IGE (Interrupt Global Enable) bit
move  APC.6, #0            ; clear IDS bit (APC.6)
```

As with other instructions, prefixing is required to select destination registers beyond index 07h.

The MOVE instruction may also be used to transfer any one of the lowest 8 bits from a register source or any bit of the active accumulator (Acc) to the Carry flag. There is no restriction on the source register module for the 'MOVE C, src.bit' instruction.

```
move  C, IIR.3             ; copy IIR.3 to Carry
move  C, Acc.7             ; copy Acc.7 to Carry
```

Prefixing is required to select source registers beyond index 15h.

23.5 – Using the Arithmetic and Logic Unit

The DS4830A provides a 16-bit Arithmetic and Logic Unit (ALU) which allows operations to be performed between the active accumulator and any other register. The DS4830A is equipped with sixteen 16-bit working accumulators.

23.5.1 – Selecting the Active Accumulator

Any of the sixteen accumulator registers A[0] through A[15] may be selected as the active accumulator by setting the low four bits of the Accumulator Pointer Register (AP) to the index of the accumulator register you want to select.

```
move  AP, #01h             ; select A[1] as the active accumulator
move  AP, #0Fh             ; select A[15] as the active accumulator
```

The current active accumulator can be accessed as the Acc register, which is also the register used as the implicit destination for all arithmetic and logical operations.

```
move  A[0], #55h           ; set A[0] =   0055 hex

move  AP, #00h             ; select A[0] as active accumulator
move  Acc, #55h            ; set A[0] =   0055 hex
```

23.5.2 – Enabling Auto-Increment and Auto-Decrement

The accumulator pointer AP can be set to automatically increment or decrement after each arithmetic or logical operation. This is useful for operations involving a number of accumulator registers, such as adding or subtracting two multibyte integers. If auto-increment/decrement is enabled, the AP register will increment or decrement after any of the following operations:

- ADD src (Add source to active accumulator)
- ADDC src (Add source to active accumulator with carry)
- SUB src (Subtract source from active accumulator)
- SUBB src (Subtract source from active accumulator with borrow)
- AND src (Logical AND active accumulator with source)
- OR src (Logical OR active accumulator with source)
- XOR src (Logical XOR active accumulator with source)
- CPL (Bitwise complement active accumulator)
- NEG (Negate active accumulator)
- SLA (Arithmetic shift left on active accumulator)
- SLA2 (Arithmetic shift left active accumulator 2 bit positions)
- SLA4 (Arithmetic shift left active accumulator 4 bit positions)
- SRA (Arithmetic shift right on active accumulator)
- SRA2 (Arithmetic shift right active accumulator 2 bit positions)
- SRA4 (Arithmetic shift right active accumulator 4 bit positions)
- RL (Rotate active accumulator left)
- RLC (Rotate active accumulator left through Carry flag)
- RR (Rotate active accumulator right)
- RRC (Rotate active accumulator right through Carry flag)
- SR (Logical shift active accumulator right)

- MOVE Acc, src (Copy data from source to active accumulator)
- MOVE dst, Acc (Copy data from active accumulator to destination)
- MOVE Acc, Acc (Recirculation of active accumulator contents)
- XCHN (Exchange nibbles within each byte of active accumulator)
- XCH (Exchange active accumulator bytes)

The active accumulator may not be the source in any instruction where it is also the implicit destination.

There is an additional notation that can be used to refer to the active accumulator for the instruction "MOVE dst, Acc". If the instruction is instead written as "MOVE dst, A[AP]", the source value is still the active accumulator, but no AP auto-increment or auto-decrement function will take place, even if this function is enabled. Note that the active accumulator may not be the destination for the MOVE dst, A[AP] instruction (i.e. MOVE Acc, A[AP] is prohibited).

So, the two instructions

```
move  A[7], Acc
move  A[7], A[AP]
```

are equivalent except that the first instruction triggers auto-inc/dec (if it is enabled), while the second one will never do so.

The Accumulator Pointer Control Register (APC) controls the automatic increment/decrement mode as well as selects the range of bits (modulo) in the AP register that will be incremented or decremented. There are nine different unique settings for the APC register, as listed in Table 23-1.

Table 23-1. Accumulator Pointer Control Register Settings

APC.2 (MOD2)	APC.1 (MOD1)	APC.0 (MOD0)	APC.6 (IDS)	APC	AUTO-INCREMENT/-DECREMENT SETTING
0	0	0	0	00h	No auto-increment/decrement (default mode)
0	0	1	0	01h	Increment bit 0 of AP (modulo 2)
0	0	1	1	41h	Decrement bit 0 of AP (modulo 2)
0	1	0	0	02h	Increment bits [1:0] of AP (modulo 4)
0	1	0	1	42h	Decrement bits [1:0] of AP (modulo 4)
0	1	1	0	03h	Increment bits [2:0] of AP (modulo 8)
0	1	1	1	43h	Decrement bits [2:0] of AP (modulo 8)
1	0	0	0	04h	Increment all 4 bits of AP (modulo 16)
1	0	0	1	44h	Decrement all 4 bits of AP (modulo 16)

For the modulo increment or decrement operation, the selected range of bits in AP are incremented or decremented. However, if these bits roll over or under, they simply wrap around without affecting the remaining bits in the accumulator pointer. So, the operations can be defined as follows:

- Increment modulo 2: AP = AP[3:1] + ((AP[0] + 1) mod 2)
- Decrement modulo 2: AP = AP[3:1] + ((AP[0] – 1) mod 2)
- Increment modulo 4: AP = AP[3:2] + ((AP[1:0] + 1) mod 4)
- Decrement modulo 4: AP = AP[3:2] + ((AP[1:0] – 1) mod 4)
- Increment modulo 8: AP = AP[3] + ((AP[2:0] + 1) mod 8)
- Decrement modulo 8: AP = AP[3] + ((AP[2:0] – 1) mod 8)
- Increment modulo 16: AP = (AP + 1) mod 16
- Decrement modulo 16: AP = (AP – 1) mod 16

For this example, assume that all 16 accumulator registers are initially set to zero.

```
move  AP, #02h          ; select A[2] as active accumulator
move  APC, #02h         ; auto-increment AP[1:0] modulo 4
                        ;  AP   A[0]   A[1]   A[2]   A[3]
                        ;  02   0000   0000   0000   0000
add   #01h              ;  03   0000   0000   0001   0000
add   #02h              ;  00   0000   0000   0001   0002
add   #03h              ;  01   0003   0000   0001   0002
add   #04h              ;  02   0003   0004   0001   0002
add   #05h              ;  03   0003   0004   0006   0002
```

23.5.3 – ALU Operations Using the Active Accumulator and a Source

The following arithmetic and logical operations can use any register or immediate value as a source. The active accumulator Acc is always used as the second operand and the implicit destination. Also, Acc may not be used as the source for any of these operations.

```
add   A[4]          ; Acc = Acc + A[4]
addc  #32h          ; Acc = Acc + 0032h + Carry
sub   A[15]         ; Acc = Acc – A[15]
subb  A[1]          ; Acc = Acc – A[1] - Carry
cmp   #00h          ; If (Acc == 0000h), set Equals flag
and   A[0]          ; Acc = Acc AND A[0]
or    #55h          ; Acc = Acc OR
xor   A[1]          ; Acc = Acc XOR A[1]
```

23.5.4 – ALU Operations Using Only the Active Accumulator

The following arithmetic and logical operations operate only on the active accumulator.

```
cpl                 ; Acc = NOT Acc
neg                 ; Acc = (NOT Acc) + 1
rl                  ; Rotate accumulator left (not using Carry)
rlc                 ; Rotate accumulator left through Carry
rr                  ; Rotate accumulator right (not using Carry)
rrc                 ; Rotate accumulator right through Carry
sla                 ; Shift accumulator left arithmetically once
sla2                ; Shift accumulator left arithmetically twice
sla4                ; Shift accumulator left arithmetically four times
sr                  ; Shift accumulator right, set Carry to Acc.0, set Acc.15 to zero
sra                 ; Shift accumulator right arithmetically once
sra2                ; Shift accumulator right arithmetically twice
sra4                ; Shift accumulator right arithmetically four times
xchn                ; Swap low and high nibbles of each Acc byte
xch                 ; Swap low byte and high byte of Acc
```

23.5.5 – ALU Bit Operations Using Only the Active Accumulator

The following operations operate on single bits of the current active accumulator in conjunction with the Carry flag. Any of these operations may use an Acc bit from 0 to 15.

```
move  C, Acc.0      ; copy bit 0 of accumulator to Carry
move  Acc.5, C      ; copy Carry to bit 5 of accumulator
and   Acc.3         ; Acc.3 = Acc.3 AND Carry
or    Acc.0         ; Acc.0 = Acc.0 OR Carry
xor   Acc.1         ; Acc.1 = Acc.1 OR Carry
```

None of the above bit operations will cause the auto-increment, auto-decrement, or modulo operations defined by the accumulator pointer control (APC) register.

23.5.6 – Example: Adding Two 4-Byte Numbers Using Auto-Increment

```
move  A[0], #5678h  ; First number – 12345678h
move  A[1], #1234h
move  A[2], #0AAAAh ; Second number – 0AAAAAAAh
move  A[3], #0AAAh
move  APC, #81h     ; Active Acc = A[0], increment low bit = mod 2
add   A[2]          ; A[0] = 5678h + AAAAh = 0122h + Carry
addc  A[3]          ; A[1] = 1234h + AAAh + 1 = 1CDFh
                    ; 12345678h + 0AAAAAAAh = 1CDF0122h
```

23.6 – Processor Status Flag Operations

The Processor Status Flag (PSF) register contains five flags that are used to indicate and store the results of arithmetic and logical operations. Four of these flags can be used for conditional program branching.

23.6.1 – Sign Flag

The Sign flag (PSF.6) reflects the current state of the most significant bit of the active accumulator, (Acc.15). If signed arithmetic is being used, this flag indicates whether the value in the accumulator is positive or negative.

Since the Sign flag is a dynamic reflection of the high bit of the active accumulator, any instruction that changes the value in the active accumulator can potentially change the value of the Sign flag. Also, any instruction that changes which accumulator is the active one (including AP auto-increment/decrement) can also change the Sign flag.

The following operation uses the Sign flag:

```
JUMP S, src                    ; Jump if Sign flag is set
```

23.6.2 – Zero Flag

The Zero flag (PSF.7) is a dynamic flag that reflects the current state of the active accumulator, Acc. If all bits in the active accumulator are zero, the Zero flag will equal 1. Otherwise, it will equal 0.

Since the Zero flag is a dynamic reflection of (Acc == 0), any instruction that changes the value in the active accumulator can potentially change the value of the Zero flag. Also, any instruction that changes which accumulator is the active one (including AP auto-increment/decrement) can also change the Zero flag.

The following operations use the Zero flag:

```
JUMP Z, src                    ; Jump if Zero flag is set
JUMP NZ, src                   ; Jump if Zero flag is cleared
```

23.6.3 – Equals Flag

The Equals flag (PSF.0) is a static flag set by the CMP instruction. When the source given to the CMP instruction is equal to the active accumulator, the Equals flag is set to 1. When the source is different from the active accumulator, the Equals flag is cleared to 0.

The following instructions use the value of the Equals flag. Note that the 'src' for the JUMP E/NE instructions must be immediate.

```
JUMP E, src                    ; Jump if Equals flag is set
JUMP NE, src                   ; Jump if Equals flag is cleared
```

In addition to the CMP instruction, any instruction using PSF as the destination can alter the Equals flag.

23.6.4 – Carry Flag

The Carry flag (PSF.1) is a static flag indicating that a carry or borrow bit resulted from the last ADD/ADDC or SUB/SUBB operation. Unlike the other status flags, it can be set or cleared explicitly and is also used as a generic bit operand by many other instructions.

The following instructions can alter the Carry flag:

- ADD src (Add source to active accumulator)
- ADDC src (Add source and Carry to active accumulator)
- SUB src (Subtract source from active accumulator)
- SUBB src (Subtract source and Carry from active accumulator)
- SLA, SLA2, SLA4 (Arithmetic shift left active accumulator)
- SRA, SRA2, SRA4 (Arithmetic shift right active accumulator)
- SR (Shift active accumulator right)
- RLC / RRC (Rotate active accumulator left / right through Carry)
- MOVE C, Acc.<b> (Set Carry to selected active accumulator bit)
- MOVE C, #i (Explicitly set, i=1, or clear, i=0, the Carry flag)
- CPL C (Complement Carry)
- MOVE C, src.<b> (Copy bit addressable register bit to Carry)
- *any instruction using PSF as the destination*

The following instructions use the value of the Carry flag:

- ADDC src (Add source and Carry to active accumulator)
- SUBB src (Subtract source and Carry from active accumulator)
- RLC / RRC (Rotate active accumulator left / right through Carry)
- CPL C (Complement Carry)
- MOVE Acc.<b>, C (Set selected active accumulator bit to Carry)
- AND Acc.<b> (Carry = Carry AND selected active accumulator bit)
- OR Acc.<b> (Carry = Carry OR selected active accumulator bit)

- XOR Acc.<b> (Carry = Carry XOR selected active accumulator bit)
- JUMP C, src (Jump if Carry flag is set)
- JUMP NC, src (Jump if Carry flag is cleared)

23.6.5 – Overflow Flag

The Overflow flag (PSF.2) is a static flag indicating that the carry or borrow bit (Carry status Flag) resulting from the last ADD/ADDC or SUB/SUBB operation but did not match the carry or borrow of the high order bit of the active accumulator. The overflow flag is useful when performing signed arithmetic operations.

The following instructions can alter the Overflow flag:

- ADD src (Add source to active accumulator)
- ADDC src (Add source and Carry to active accumulator)
- SUB src (Subtract source from active accumulator)
- SUBB src (Subtract source and Carry from active accumulator)

23.7 – Controlling Program Flow

The DS4830A provides several options to control program flow and branching. Jumps may be unconditional, conditional, relative or absolute. Subroutine calls store the return address on the hardware stack for later return. Built-in counters and address registers are provided to control looping operations.

23.7.1 – Obtaining the Next Execution Address

The address of the next instruction to be executed can be read at any time by reading the Instruction Pointer (IP) register. This can be particularly useful for initializing loops. Note that the value returned is actually the address of the current instruction plus 1, so this will be the address of the next instruction executed as long as the current instruction does not cause a jump.

23.7.2 – Unconditional Jumps

An unconditional jump can be relative (IP +127/-128 words) or absolute (to anywhere in program space). Relative jumps must use an 8-bit immediate operand, such as

```
Label1:                     ; must be within +127/-128 words of the JUMP
        ....
        jump  Label1
```

Absolute jumps may use a 16-bit immediate operand, a 16-bit register, or an 8-bit register.

```
        jump  LongJump      ; assembles to: move PFX[0], #high(LongJump)
                            ;               jump    #low(LongJump)
        jump  DP[0]         ; absolute jump to the address in DP[0]
```

If an 8-bit register is used as the jump destination, the prefix value is used as the high byte of the address and the register is used as the low byte.

23.7.3 – Conditional Jumps

Conditional jumps transfer program execution based on the value of one of the status flags (C, E, Z, S). Except where noted for JUMP E and JUMP NE, the absolute and relative operands allowed are the same as for the unconditional JUMP command.

```
        jump c, Label1          ; jump to Label1 if Carry is set
        jump nc, LongJump       ; jump to LongJump if Carry is not set
        jump z, LC[0]           ; jump to 16-bit register destination if Zero is set
        jump nz, Label1         ; jump to Label1 if Zero is not set (Acc<>0)
        jump s, A[2]            ; jump to A[2] if Sign flag is set
        jump e, Label1          ; jump to Label1 if Equal is set
        jump ne, Label1         ; jump to Label1 if Equal is cleared
```

JUMP E and JUMP NE may only use immediate destinations.

23.7.4 – Calling Subroutines

The CALL instruction works the same as the unconditional JUMP, except that the next execution address is pushed on the stack before transferring program execution to the branch address. The RET instruction is used to return from a normal call, and RETI is used to return from an interrupt handler routine.

```
        call  Label1            ; if Label1 is relative, assembles to : call #immediate
        call  LongCall          ; assembles to:  move PFX[0], #high(LongCall)
                                ;                call #low(LongCall)
        call  LC[0]             ; call to address in LC[0]
LongCall:
        ret                     ; return from subroutine
```

23.7.5 – Looping Operations

Looping over a section of code can be performed by using the conditional jump instructions. However, there is built-in functionality, in the form of the 'DJNZ LC[n], *src*' instruction, to support faster, more compact looping code with separate loop counters. The 16-bit registers LC[0], and LC[1] are used to store these loop counts. The 'DJNZ LC[n], *src*' instruction automatically decrements the associated loop counter register and jumps to the loop address specified by *src* if the loop counter has not reached 0.

To initialize a loop, set the LC[n] register to the count you wish to use before entering the loop's main body. The desired loop address should be supplied in the *src* operand of the 'DJNZ LC[n], *src*' instruction. When the supplied loop address is relative (+127/-128 words) to the DJNZ LC[n] instruction, as is typically the case, the assembler automatically calculates the relative offset and inserts this immediate value in the object code.

```
        move  LC[1], #10h       ; loop 16 times
LoopTop:                        ; loop addr relative to djnz LC[n],src instruction
        call  LoopSub
        djnz  LC[1], LoopTop    ; decrement LC[1] and jump if nonzero
```

When the supplied loop address is outside of the relative jump range, the prefix register (PFX[0]) is used to supply the high byte of the loop address as required.

```
        move  LC[1], #10h        ; loop 16 times
LoopTop:                        ; loop addr not relative to djnz LC[n],src
        call  LoopSub
        ...
        djnz  LC[1], LoopTop    ; decrement LC[1] and jump if nonzero
                                ; assembles to: move PFX[0], #high(LoopTop)
                                ;               djnz LC[1], #low(LoopTop)
```

If loop execution speed is critical and a relative jump cannot be used, one might consider preloading an internal 16-bit register with the *src* loop address for the 'DJNZ LC[n], *src*' loop. This ensures that the prefix register will not be needed to supply the loop address and always yields the fastest execution of the DJNZ instruction.

```
        move  LC[0], #LoopTop   ; using LC[0] as address holding register
                                ; assembles to: move PFX[0], #high(LoopTop)
                                ;               move LC[0], #low(LoopTop)
        move  LC[1], #10h        ; loop 16 times
        ...
LoopTop:                        ; loop address not relative to djnz LC[n],src
        call  LoopSub
        ...
        djnz  LC[1], LC[0]      ; decrement LC[1] and jump if nonzero
```

If opting to preload the loop address to an internal 16-bit register, the most time and code efficient means is by performing the load in the instruction just prior to the top of the loop:

```
        move  LC[1], #10h       ; Set loop counter to 16
        move  LC[0], IP         ; Set loop address to the next address
LoopTop:                        ; loop addr not relative to djnz LC[n],src
        ...
```

23.7.6 – Conditional Returns

Similar to the conditional jumps, the DS4830A microcontroller also supports a set of conditional return operations. Based upon the value of one of the status flags, the CPU can conditionally pop the stack and begin execution at the address popped from the stack. If the condition is not true, the conditional return instruction does not pop the stack and does not change the instruction pointer. The following conditional return operations are supported:

```
        RET C                   ; if C=1, a RET is executed
        RET NC                  ; if C=0, a RET is executed
        RET Z                   ; if Z=1 (Acc=00h), a RET is executed
        RET NZ                  ; if Z=0 (Acc<>00h), a RET is executed
        RET S                   ; if S=1, a RET is executed
```

23.8 – Handling Interrupts

Handling interrupts in the DS4830A microcontroller is a three-part process.

First, the location of the interrupt handling routine must be set by writing the address to the 16-bit Interrupt Vector (IV) register. This register defaults to 0000h on reset, but this will usually not be the desired location since this will often be the location of reset / power-up code.

```
        move  IV, IntHandler    ; move PFX[0], #high(IntHandler)
                                ; move IV, #low(IntHandler)
                                ; PFX[0] write not needed if IntHandler addr=0023h
```

Next, the interrupt must be enabled. For any interrupts to be handled, the IGE bit in the Interrupt and Control register (IC) must first be set to 1. Next, the interrupt itself must be enabled at the module level and locally within the module itself. The module interrupt enable is located in the Interrupt Mask register, while the location of the local interrupt enable will vary depending on the module in which the interrupt source is located.

Once the interrupt handler receives the interrupt, the Interrupt in Service (INS) bit will be set by hardware to block further interrupts, and execution control is transferred to the interrupt service routine. Within the interrupt service routine, the source of the interrupt must be determined. Since all interrupts go to the same interrupt service routine, the Interrupt Identification Register (IIR) must be examined to determine which module initiated the interrupt. For example, the II0 (IIR.0) bit will be set if there is a pending interrupt from module 0. These bits cannot be cleared directly; instead, the appropriate bit flag in the module must be cleared once the interrupt is handled.

INS is set automatically on entry to the interrupt handler and cleared automatically on exit (RETI).

```
IntHandler:
        push  PSF               ; save C since used in identification process
        move  C, IIR.X          ; check highest priority flag in IIR
        jump  C, ISR_X          ; if IIR.X is set, interrupt from module X
        move  C, IIR.Y          ; check next highest priority int source
        jump  C, ISR_Y          ; if IIR.Y is set, interrupt from module Y
        ...
ISR_X:
        ...
        reti
```

To support high priority interrupts while servicing another interrupt source, the IMR register may be used to create a user-defined prioritization. The IMR mask register should not be utilized when the highest priority interrupt is being serviced because the highest priority interrupt should never be interrupted. This is default condition when a hardware branch is made the Interrupt Vector address (INS is set to 1 by hardware and all other interrupt sources are blocked). The code below demonstrates how to use IMR to allow other interrupts.

```
ISR_Z:
        pop   PSF               ; restore PSF
        push  IMR               ; save current interrupt mask
        move  IMR, #int_mask    ; new mask to allow only higher priority ints
        move  INS, #0           ; re-enable interrupts
```

```
 ...
(interrupt servicing code)
...
pop   IMR                   ; restore previous interrupt mask
ret                         ; back to code or lower priority interrupt
```

Note that configuring a given IMR register mask bit to '0' only prevents interrupt conditions from the corresponding module or system from generating an interrupt request. Configuring an IMR mask bit to '0' does not prevent the corresponding IIR system or module identification flag from being set. This means that when using the IMR mask register functionality to block interrupts, there may be cases when both the mask (IMR.x) and identifier (IIR.x) bits should be considered when determining if the corresponding peripheral should be serviced.

23.8.1 – Conditional Return from Interrupt

Similar to the conditional returns, the DS4830A microcontroller also supports a set of conditional return from interrupt operation. Based upon the value of one of the status flags, the CPU can conditionally pop the stack, clear the INS bit to 0, and begin execution at the address popped from the stack. If the condition is not true, the conditional return from interrupt instruction leaves the INS bit unchanged, does not pop the stack and does not change the instruction pointer. The following conditional return from interrupt operations are supported:

```
RETI C                      ; if C=1, a RETI is executed
RETI NC                     ; if C=0, a RETI is executed
RETI Z                      ; if Z=1 (Acc=00h), a RETI is executed
RETI NZ                     ; if Z=0 (Acc<>00h), a RETI is executed
RETI S                      ; if S=1, a RETI is executed
```

23.9 – Accessing the Stack

The hardware stack is used automatically by the CALL, RET and RETI instructions, but it can also be used explicitly to store and retrieve data. All values stored on the stack are 16 bits wide.

The PUSH instruction increments the stack pointer SP and then stores a value on the stack. When pushing a 16-bit value onto the stack, the entire value is stored. However, when pushing an 8-bit value onto the stack, the high byte stored on the stack comes from the prefix register. The @++SP stack access mnemonic is the associated destination specifier that generates this push behavior, thus the following two instruction sequences are equivalent:

```
move  PFX[0], IC
push  PSF                   ; stored on stack: IC:PSF

move  PFX[0], IC
move  @++SP, PSF            ; stored on stack: IC:PSF
```

The POP instruction removes a value from the stack and then decrements the stack pointer. The @SP-- stack access mnemonic is the associated source specifier that generates this behavior, thus the following two instructions are equivalent:

```
pop   PSF
move  PSF, @SP--
```

The POPI instruction is equivalent to the POP instruction but additionally clears the INS bit to '0'. Thus, the following two instructions would be equivalent:

```
popi  IP
reti
```

The @SP-- mnemonic can be utilized by the DS4830A microcontroller so that stack values may be used directly by ALU operations (e.g. ADD src, XOR src, etc.) without requiring that the value be first popped into an intermediate register or accumulator.

```
add   @SP--                 ; sum the last three words pushed onto the stack
add   @SP--                 ;  with Acc, disregarding overflow
add   @SP--
```

The stack pointer SP can be set explicitly. For a DS4830A, which has a stack depth of 16 words, only the lowest four bits are used and setting SP to 0Fh will return it to its reset state.

Since the stack is 16 bits wide, it is possible to store two 8-bit register values on it in a single location. This allows more efficient use of the stack if it is being used to save and restore registers at the start and end of a subroutine.

```
SubOne:
        move  PFX[0], IC
        push  PSF                   ; store IC:PSF on the stack
        ...
        pop   GR                    ; 16-bit register
        move  IC, GRH               ; IC was stored as high byte
        move  PSF, GRL              ; PSF was stored as low byte
        ret
```

23.10 – Accessing Data Memory

Data memory is accessed through the data pointer registers DP[0] and DP[1] or the Frame Pointer BP[Offs]. Once one of these registers is set to a location in data memory, that location can be read or written as follows, using the mnemonic @DP[0], @DP[1] or @BP[OFFS] as a source or destination.

```
        move  DP[0], #0000h         ; set pointer to location 0000h
        move  A[0], @DP[0]          ; read from data memory
        move  @DP[0], #55h          ; write to data memory
```

Either of the data pointers may be post-incremented or post-decremented following any read or may be pre-incremented or pre-decremented before any write access by using the following syntax.

```
        move  A[0], @DP[0]++        ; increment DP[0] after read
        move  @++DP[0], A[1]        ; increment DP[0] before write
        move  A[5], @DP[1]--        ; decrement DP[1] after read
        move  @--DP[1], #00h        ; decrement DP[1] before write
```

The Frame Pointer (BP[OFFS]) is actually comprised of a base pointer (BP) and an offset from the base pointer (OFFS). For the frame pointer, the offset register (OFFS) is the target of any increment or decrement operation. The base pointer (BP) is unaffected by increment and decrement operations on the Frame Pointer. Similar to DP[n], the OFFS register may be pre-incremented/decremented when writing to data memory and may be post-incremented/decremented when reading from data memory.

```
        move  A[0], @BP[OFFS--]         ; decrement OFFS after read
        move  @BP[++OFFS], A[1]         ; increment OFFS before write
```

All three data pointers support both byte and word access to data memory. Each data pointer has its own word/byte select (WBSn) special function register bit to control the access mode associated with the data pointer. These three register bits (WBS2 which controls BP[Offs] access, WBS1 which controls DP[1] access and WBS0 which control DP[0] access) reside in the Data Pointer Control (DPC) register. When a given WBSn control bit is configured to 1, the associated pointer is operated in the word access mode. When the WBSn bit is configured to 0, the pointer is operated in the byte access mode. Word access mode allows addressing of 64k words of memory while byte access mode allows addressing of 64k bytes of memory.

Each data pointer (DP[n]) and Frame Pointer base, BP register) is actually implemented internally as a 17-bit register (e.g. 16:0). The Frame Pointer offset register (OFFS) is implemented internally as a 9-bit register (e.g.8:0). The WBSn bit for the respective pointer controls whether the highest 16 bits (16:1) of the pointer are in use, as is the case for word mode (WBSn = 1) or whether the lowest 16 bits (15:0) are in use, as will be the case for byte mode (WBSn = 0). The WBS2 bit also controls whether the high 8 bits (8:1) of the offset register are in use (WBS2 = 1) or the low 8 bits (7:0) are used (WBS2 = 0). All data pointer register reads, writes, auto-increment/decrement operations occur with respect to the current WBSn selection. Data pointer increment and decrement operations only affect those bits specific to the current word or byte addressing mode (e.g., incrementing a byte mode data pointer from FFFFh does not carry into the internal high order bit that is utilized only for word mode data pointer access). Switching from byte to word access mode or vice versa does not alter the data pointer contents. Therefore, it is important to maintain the consistency of data pointer address value within the given access mode.

```
        move  WBS0, #0              ; DP[0] in byte mode
        move  DP[0], #1             ; DP[0]=0001h (byte mode, index 1)
        move  WBS0, #1              ; DP[0] in word mode, byte mode lsbit not visible
        move  DP[0], #1             ; DP[0]=0001h (word mode, index 1)
        move  WBS0, #0              ; DP[0] in byte mode
        move  GR, DP[0]             ; GR = 0003h (word index 1, byte index 1)
```

The three pointers share a single read/write port on the data memory and thus, the user must knowingly activate a desired pointer before using it for data memory read operations. This can be done explicitly using the data pointer

select bits (SDPS1:0; DPC.1:0), or implicitly by writing to the DP[n], BP or OFFS registers. Any indirect memory write operation using a data pointer will set the SDPS bits, thus activating the write pointer as the active source pointer.

```
move  SDPS1, #1          ; (explicit) selection of FP as the pointer
move  DP[0], src         ; (implicit) selection of DP[0]; set SDPS1:0=00b
move  DP[1], DP[1]       ; (implicit) selection of DP[1]; set SDPS1:0=01b
move  OFFS, src          ; (implicit) selection of FP; set SDPS1=1
move  WBS1, #0           ; (implicit) selection of byte access for DP[1]
```

Once the pointer selection has been made, it will remain in effect until:

- the source data pointer select bits are changed via the explicit or implicit methods described above (i.e. another data pointer is selected for use).
- the memory to which the active source data pointer is addressing is enabled for code fetching using the Instruction Pointer, or
- a memory write operation is performed using a data pointer other than the current active source pointer.

```
move  DP[1], DP[1]       ; select DP[1] as the active pointer
move  dst, @DP[1]        ; read from pointer
move  @DP[1], src        ; write using a data pointer
                         ; DP[0] is needed
move  DP[0], DP[0]       ; select DP[0] as the active pointer
```

To simplify data pointer increment / decrement operations without disturbing register data, a virtual NUL destination has been assigned to system module 6, sub-index 7 to serve as a bit bucket. Data pointer increment / decrement operations can be done as follows without altering the contents of any other register:

```
move  NUL, @DP[0]++      ; increment DP[0]
move  NUL, @DP[0]--      ; decrement DP[0]
```

The following data pointer related instructions are invalid:

```
move @++DP[0], @DP[0]++
move @++DP[1], @DP[1]++
move @BP[++Offs], @BP[Offs++]
move @--DP[0], @DP[0]--
move @--DP[1], @DP[1]--
move @BP[--Offs], @BP[Offs--]
move @++DP[0], @DP[0]--
move @++DP[1], @DP[1]--
move @BP[++Offs], @BP[Offs--]
move @--DP[0], @DP[0]++
move @--DP[1], @DP[1]++
move @BP[--Offs], @BP[Offs++]
move @DP[0], @DP[0]++
move @DP[1], @DP[1]++
move @BP[Offs], @BP[Offs++]
move @DP[0], @DP[0]--
move @DP[1], @DP[1]--
move @BP[Offs], @BP[Offs--]
move DP[0], @DP[0]++
move DP[0], @DP[0]--
move DP[1], @DP[1]++
move DP[1], @DP[1]--
move Offs, @BP[Offs--]
move Offs, @BP[Offs++]
```

SECTION 24 – INSTRUCTION SET

Table 24-1. Instruction Set Summary

	MNEMONIC	DESCRIPTION	16-BIT INSTRUCTION WORD	STATUS BITS AFFECTED	AP INC/DEC	NOTES
LOGICAL OPERATIONS	AND src	Acc ← Acc AND src	f001 1010 ssss ssss	S, Z	Y	1
	OR src	Acc ← Acc OR src	f010 1010 ssss ssss	S, Z	Y	1
	XOR src	Acc ← Acc XOR src	f011 1010 ssss ssss	S, Z	Y	1
	CPL	Acc ← ~Acc	1000 1010 0001 1010	S, Z	Y	
	NEG	Acc ← ~Acc + 1	1000 1010 1001 1010	S, Z	Y	
	SLA	Shift Acc left arithmetically	1000 1010 0010 1010	C, S, Z	Y	
	SLA2	Shift Acc left arithmetically twice	1000 1010 0011 1010	C, S, Z	Y	
	SLA4	Shift Acc left arithmetically four times	1000 1010 0110 1010	C, S, Z	Y	
	RL	Rotate Acc left (w/o C)	1000 1010 0100 1010	S	Y	
	RLC	Rotate Acc left (through C)	1000 1010 0101 1010	C, S, Z	Y	
	SRA	Shift Acc right arithmetically	1000 1010 1111 1010	C, Z	Y	
	SRA2	Shift Acc right arithmetically twice	1000 1010 1110 1010	C, Z	Y	
	SRA4	Shift Acc right arithmetically four times	1000 1010 1011 1010	C, Z	Y	
	SR	Shift Acc right (0 → msbit)	1000 1010 1010 1010	C, S, Z	Y	
	RR	Rotate Acc right (w/o C)	1000 1010 1100 1010	S	Y	
	RRC	Rotate Acc right (though C)	1000 1010 1101 1010	C, S, Z	Y	
BIT OPERATIONS	MOVE C, Acc.<b>	C ← Acc.<b>	1110 1010 bbbb 1010	C		
	MOVE C, #0	C ← 0	1101 1010 0000 1010	C		
	MOVE C, #1	C ← 1	1101 1010 0001 1010	C		
	CPL C	C ← ~C	1101 1010 0010 1010	C		
	MOVE Acc.<b>, C	Acc.<b> ← C	1111 1010 bbbb 1010	S, Z		
	AND Acc.<b>	C ← C AND Acc.<b>	1001 1010 bbbb 1010	C		
	OR Acc.<b>	C ← C OR Acc.<b>	1010 1010 bbbb 1010	C		
	XOR Acc.<b>	C ← C XOR Acc.<b>	1011 1010 bbbb 1010	C		
	MOVE dst.<b>, #1	dst.<b> ← 1	1ddd dddd 1bbb 0111	C,E		2
	MOVE dst.<b>, #0	dst.<b> ← 0	1ddd dddd 0bbb 0111	C,E		2
	MOVE C, src.<b>	C ← src.<b>	fbbb 0111 ssss ssss	C		
MATH	ADD src	Acc ← Acc + src	f100 1010 ssss ssss	C, S, Z, OV	Y	1
	ADDC src	Acc ← Acc + (src + C)	f110 1010 ssss ssss	C, S, Z, OV	Y	1
	SUB src	Acc ← Acc – src	f101 1010 ssss ssss	C, S, Z, OV	Y	1
	SUBB src	Acc ← Acc – (src + C)	f111 1010 ssss ssss	C, S, Z, OV	Y	1
BRANCHING	{L/S}JUMP src	IP ← IP + src or src	f000 1100 ssss ssss			6
	{L/S}JUMP C, src	If C=1, IP ← (IP + src) or src	f010 1100 ssss ssss			6
	{L/S}JUMP NC, src	If C=0, IP ← (IP + src) or src	f110 1100 ssss ssss			6
	{L/S}JUMP Z, src	If Z=1, IP ← (IP + src) or src	f001 1100 ssss ssss			6
	{L/S}JUMP NZ, src	If Z=0, IP ← (IP + src) or src	f101 1100 ssss ssss			6
	{L/S}JUMP E, src	If E=1, IP ← (IP + src) or src	0011 1100 ssss ssss			6
	{L/S}JUMP NE, src	If E=0, IP ← (IP + src) or src	0111 1100 ssss ssss			6
	{L/S}JUMP S, src	If S=1, IP ← (IP + src) or src	f100 1100 ssss ssss			6
	{L/S}DJNZ LC[n], src	If --LC[n] <> 0, IP← (IP + src) or src	f10n 1101 ssss ssss			6
	{L/S}CALL src	@++SP ← IP+1; IP ← (IP+src) or src	f011 1101 ssss ssss			6,7
	RET	IP ← @SP--	1000 1100 0000 1101			
	RET C	If C=1, IP ← @SP--	1010 1100 0000 1101			
	RET NC	If C=0, IP ← @SP--	1110 1100 0000 1101			
	RET Z	If Z=1, IP ← @SP--	1001 1100 0000 1101			
	RET NZ	If Z=0, IP ← @SP--	1101 1100 0000 1101			
	RET S	If S=1, IP ← @SP--	1100 1100 0000 1101			
	RETI	IP ← @SP-- ; INS← 0	1000 1100 1000 1101			
	RETI C	If C=1, IP ← @SP-- ; INS← 0	1010 1100 1000 1101			
	RETI NC	If C=0, IP ← @SP-- ; INS← 0	1110 1100 1000 1101			
	RETI Z	If Z=1, IP ← @SP-- ; INS← 0	1001 1100 1000 1101			
	RETI NZ	If Z=0, IP ← @SP-- ; INS← 0	1101 1100 1000 1101			
	RETI S	If S=1, IP ← @SP-- ; INS← 0	1100 1100 1000 1101			
DATA TRANSFER	XCH	Swap Acc bytes	1000 1010 1000 1010	S	Y	
	XCHN	Swap nibbles in each Acc byte	1000 1010 0111 1010	S	Y	
	MOVE dst, src	dst ← src	fddd dddd ssss ssss	C,S,Z,E	(Note 8)	7,8
	PUSH src	@++SP ← src	f000 1101 ssss ssss			7
	POP dst	dst ← @SP--	1ddd dddd 0000 1101	C,S,Z,E		7
	POPI dst	dst ← @SP-- ; INS ← 0	1ddd dddd 1000 1101	C,S,Z,E		7
	CMP *src*	E ← (Acc = src)	f111 1000 ssss ssss	E		
	NOP	No operation	1101 1010 0011 1010			

Note 1: The active accumulator (Acc) is not allowed as the src in operations where it is the implicit destination.
Note 2: Only module 8 and modules 0-5 are supported by these single-cycle bit operations. Potentially affects C or E if PSF register is the destination. Potentially affects S and/or Z if AP or APC is the destination.
Note 3: The terms Acc and A[AP] can be used interchangeably to denote the active accumulator.

Note 4: Any index represented by <b> or found inside [ ] brackets is considered variable, but required.
Note 5: The active accumulator (Acc) is not allowed as the dst if A[AP] is specified as the src.
Note 6: The '{L/S}' prefix is optional.
Note 7: Instructions that attempt to simultaneously push/pop the stack (e.g. PUSH @SP--, PUSH @SPI--, POP @++SP, POPI @++SP) or modify SP in a conflicting manner (e.g., MOVE SP, @SP--) are invalid.
Note 8: Special cases: If 'MOVE APC, Acc' sets the APC.CLR bit, AP will be cleared, overriding any auto-inc/dec/modulo operation specified for AP. If 'MOVE AP, Acc' causes an auto-inc/dec/modulo operation on AP, this overrides the specified data transfer (i.e., Acc will not be transferred to AP).

ADD / ADDC *src*

Add / Add with Carry

Description: The **ADD** instruction sums the active accumulator (Acc or A[AP]) and the specified *src* data and stores the result back to the active accumulator. The **ADDC** instruction additionally includes the Carry (C) Status Flag in the summation. For the complete list of *src* specifiers, reference the **MOVE** instruction. Because the source field is limited to 8 bits, the PFX[n] register is used to supply the high-byte of data for 16 bit sources.

Status Flags: C, S, Z, OV

ADD

Operation: Acc ← Acc + *src*

Encoding:

15			0
f100	1010	ssss	ssss

Example(s):

```
                    ; Acc = 2345h for each example
ADD A[3]            ; A[3]=FF0Fh
                    ; → Acc =2254h,C=1, Z=0, S=0, OV=0
ADD #0C0h           ; → Acc =2405h,C=0, Z=0, S=0, OV=0
ADD A[4]            ; A[4]=C000h
                    ; → Acc = E345h, C=0, Z=0, S=1, OV=0
ADD A[5]            ; A[5]=6789h
                    ; → Acc = 8ACEh, C=0, Z=0, S=1, OV=1
```

ADDC

Operation: Acc ← Acc + C + *src*

Encoding:

15			0
f110	1010	ssss	ssss

Example(s):

```
                    ; Acc = 2345h for each example
ADDC A[3]           ; A[3] = DCBAh, C=1
                    ; → Acc = 0000h, C=1, Z=1, S=0, OV=0
ADDC @DP[0]--       ; @DP[0] = 00EEh, C=1
                    ; → Acc = 2434h, C=0, Z=0, S=0, OV=0
```

Special Notes: The active accumulator (Acc) is not allowed as the *src* for these operations.

AND *src* Logical AND

Description: Performs a logical-AND between the active accumulator (Acc) and the specified *src* data. For the complete list of *src* specifiers, reference the **MOVE** instruction. Because the source field is limited to 8 bits, the PFX[n] register is used to supply the high-byte of data for 16 bit sources.

Status Flags: S, Z

Operation: Acc ← Acc AND *src*

Encoding:

15			0
f001	1010	ssss	ssss

Example(s):

```
                        ; Acc = 2345h for each example
AND A[3]                ; A[3]=0F0Fh
                        ; ➔ Acc = 0305h, S=0, Z=0
AND  #33h               ; ➔ Acc = 0001h
AND #2233h              ; generates object code below
                        ; MOVE PFX[0], #22h (smart-prefixing)
                        ; AND #33h
                        ; ➔ Acc = 2201h
MOVE PFX[0], #0Fh
AND M0[8]               ; M0[8]=0Fh (assume M0[8] is an 8-bit register)
                        ; ➔ Acc = 0305h
```

Special Notes: The active accumulator (Acc) is not allowed as the *src* for this operation.

AND *Acc.<b>* Logical AND Carry Flag with Accumulator Bit

Description: Performs a logical-AND between the Carry (C) status flag and a specified bit of the active accumulator (Acc.<b>) and returns the result to the Carry.

Status Flags: C

Operation: C ← C AND *Acc.<b>*

Encoding:

15			0
1001	1010	bbbb	1010

Example(s):

```
                        ; Acc = 2345h, C=1 at start
AND Acc.0               ; Acc.0=1  ➔ C=1
AND Acc.1               ; Acc.1=0  ➔ C=0
AND C, Acc.8            ; Acc.8=1  ➔ C=0
```

{L/S}CALL *src* {Long/Short} Call to Subroutine

Description: Performs a call to the subroutine destination specified by *src*. The **CALL** instruction uses an 8-bit immediate src to perform a relative short call (IP +127/-128 words). The **CALL** instruction uses a 16-bit immediate *src* to perform an absolute long **CALL** to the specified 16-bit address. The PFX[0] register is used to supply the high byte of a 16-bit immediate address for the absolute long **CALL**. Using the optional 'L' prefix (i.e. **LCALL**) will result in an absolute long call and use of the PFX[0] register. Using the optional 'S' prefix (i.e. **SCALL**) will attempt to generate a relative short call, but will be flagged by the assembler if the destination is out or range. Specifying an internal register *src* (no matter whether 8-bit or 16-bit) always produces an absolute **CALL** to a 16-bit address, thus the 'L' and 'S' prefixes should not be used. The PFX[n] register value is used to supply the high address byte when an 8-bit register *src* is specified.

Status Flags: None

Operation:

@++SP ← IP + 1	*PUSH*
IP ← *src*	*Absolute CALL*
IP ← IP + *src*	*Relative CALL*

Encoding:

15			0
f011	1101	ssss	ssss

Example(s):

```
CALL label1             ; relative call to label1 (must be within IP +127/ -
                        ; 128 address range)
CALL label1             ; absolute call to label1 = 0120h
                        ; MOVE PFX[0], #01h
                        ; CALL #20h.
CALL   DP[0]            ; DP[0] holds 16-bit address of subroutine
CALL  M0[0]             ; assume M0[0] is an 8-bit register
                        ; absolute call to addr16
                        ; high(addr16)=00h   (PFX[0])
                        ; low (addr16)=M0[0]
MOVE PFX[0], #22h       ;
CALL  M0[0]             ; assume M0[0] is an 8-bit register
                        ; high(addr16)=22h   (PFX[0])
                        ; low (addr16)=M0[0]

LCALL label1            ; label=0120h and is relative to this instruction
                        ; absolute call is forced by use of 'L' prefix
                        ; MOVE PFX[0], #01h
                        ; CALL #20h
SCALL label1            ; relative offset for label1 calculated and used
                        ; if label1 is not relative, assembler will generate an error
SCALL #10h              ; relative offset of #10h is used directly by the CALL
```

CMP *src* Compare Accumulator

Description: Compare for equality between the active accumulator and the least significant byte of the specified *src*. Because the source is limited to 8 bits, the PFX[n] register is used to supply the high-byte of data for 16 bit sources.

Status Flags: E

Operation: Acc = src: E ← 1
Acc <> src: E ← 0

Encoding:

15			0
f111	1000	Ssss	ssss

Example(s):

```
CMP  #45h          ; Acc = 0145h, E=0
CMP  #145h         ; PFX[0] register used
                   ; MOVE PFX[0], #01h (smart-prefixing)
                   ; CMP  #45h  E=1
```

CPL Complement Acc

Description: Performs a logical bitwise complement (1's complement) on the active accumulator (Acc or A[AP]) and returns the result to the active accumulator.

Status Flags: S, Z

Operation: Acc ← ~Acc

Encoding:

15			0
1000	1010	0001	1010

Example(s):

```
                   ; Acc = FFFFh, S=1, Z=0
CPL                ; Acc ← 0000h, S=0, Z=1
                   ; Acc = 0990h, S=0, Z=0
CPL                ; Acc ← F66Fh, S=1, Z=0
```

CPL C Complement Carry Flag

Description: Logically complements the Carry (C) Flag.

Status Flag: C

Operation: C ← ~C

Encoding:

15			0
1101	1010	0010	1010

Example(s):

```
                   ; C = 0
CPL C              ; C ← 1
```

{L/S}DJNZ LC*[n], src* Decrement Counter, {Long/Short} Jump Not Zero

Description: The **DJNZ LC**[n], *src* instruction performs a conditional branch based upon the associated Loop Counter (LC[n]) register. The **DJNZ LC**[n], *src* instruction decrements the LC[n] loop counter and branches to the address defined by *src* if the decremented counter has not reached 0000h. Program branches can be relative or absolute depending upon the *src* specifier and may be qualified by using the 'L' or 'S' prefixes as documented in the **JUMP** *src* opcode.

Status Flags: None

Operation:
LC[n] ← LC[n] –1
LC[n] <> 0: IP ← IP + *src (relative)* –or— *src (absolute)*
LC[n] = 0: IP ← IP + 1

Encoding:

15			0
f10n	1101	ssss	ssss

Example(s):

```
MOVE LC[1], #10h      ; counter = 10h
Loop:
ADD    @DP[0]++       ; add data memory contents to Acc, post-inc DP[0]
DJNZ  LC[1],  Loop    ; 16 times before falling through
```

{L/S} JUMP *src* Unconditional {Long/Short} Jump

Description: Performs an unconditional jump as determined by the *src* specifier. The JUMP instruction uses an 8-bit immediate *src* to perform a relative jump (IP +127/-128 words). The JUMP instruction uses a 16-bit immediate *src* to perform an absolute JUMP to the specified 16-bit address. The PFX[0] register is used to supply the high byte of a 16-bit immediate address for the absolute JUMP. Using the optional 'L' prefix (i.e. **LJUMP**) will result in an absolute long jump and use of the PFX[0] register. Using the optional 'S' prefix (i.e. **SJUMP**) will attempt to generate a relative short jump, but will be flagged by the assembler if the destination is out or range. Specifying an internal register *src* (no matter whether 8-bit or 16-bit) always produces an absolute **JUMP** to a 16-bit address, thus the 'L' and 'S' prefixes should not be used. The PFX[n] register value is used to supply the high address byte when an 8-bit register *src* is specified.

Status Flags: None

Operation:

IP ← *src* *Absolute JUMP*
IP ← IP + *src* *Relative JUMP*

Encoding:

15			0
f000	1100	ssss	ssss

Example(s):

```
JUMP label1          ; relative jump to label1 (must be within range
                     ;         IP +127/-128 words)
JUMP label1          ; absolute jump to label1= 0400h
                     ; MOVE PFX[0], #04h
                     ; JUMP #00h
JUMP   DP[0]         ; absolute jump to addr16 DP[0]
JUMP M0[0]           ; assume M0[0] is an 8-bit register
                     ; absolute jump to addr16
                     ; high(addr16)=00h   (PFX[0])
                     ; low (addr16)=M0[0]

LJUMP label1         ; label=0120h and is relative to this instruction
                     ; absolute jump is forced by use of 'L' prefix
                     ; MOVE PFX[0], #01h
                     ; JUMP #20h
SJUMP label1         ; relative offset for label1 calculated and used
                     ; if label1 is not relative, assembler will generate an error
SJUMP #10h           ; relative offset of #10h is used directly by the JUMP
```

{L/S} JUMP C / {L/S} JUMP NC, *src*
{L/S} JUMP Z / {L/S} JUMP NZ, *src*
{L/S} JUMP E / {L/S} JUMP NE, *src*
{L/S} JUMP S, *src*

Conditional {Long/Short} Jump on Status Flag

Description: Performs conditional branching based upon the state of a specific processor status flag. **JUMP C** results in a branch if the Carry flag is set while **JUMP NC** branches if the Carry flag is clear. **JUMP Z** results in a branch if the Zero flag is set while **JUMP NZ** branches if the Zero flag is clear. **JUMP E** results in a branch if the Equal flag is set while **JUMP NE** branches if the Equal flag is clear. **JUMP S** results in a branch if the Sign flag is set. Program branches can be relative or absolute depending upon the *src* specifier and may be qualified by using the 'L' or 'S' prefixes as documented in the **JUMP** *src* opcode. Special *src* restrictions apply to **JUMP E** and **JUMP NE**.

Status Flags: None

JUMP C

Operation:
C=1: IP ← IP + *src (relative)* –or— *src (absolute)*
C=0: IP ← IP + 1

Encoding:

15			0
f010	1100	ssss	ssss

Example(s):
```
JUMP C, label1          ; C=0, branch not taken
```

JUMP NC

Operation:
C=0: IP ← IP + *src (relative)* –or— *src (absolute)*
C=1: IP ← IP +1

Encoding:

15			0
f010	1100	ssss	ssss

Example(s):
```
JUMP NC, label1         ; C=0, branch taken
```

JUMP Z

Operation:
Z=1: IP ← IP + *src*
Z=0: IP ← IP + 1

Encoding:

15			0
f001	1100	ssss	ssss

Example(s):
```
JUMP Z, label1          ; Z=1, branch taken
```

JUMP NZ

Operation: Z=0: IP ← IP + *src* (relative) –*or*— *src* (absolute)
Z=1: IP ← IP + 1

Encoding:

15			0
f101	1100	ssss	ssss

Example(s): JUMP NZ, label1 ; Z=1, branch not taken

JUMP E

Operation: E=1: IP ← IP + *src* (relative) –or— *src* (absolute)
E=0: IP ← IP + 1

Encoding:

15			0
0011	1100	ssss	ssss

Example(s): JUMP E, label1 ; E=1, branch taken

Special Notes: The *src* specifier must be immediate data.

JUMP NE

Operation: E=0: IP ← IP + *src* (relative) –or— *src* (absolute)
E=1: IP ← IP + 1

Encoding:

15			0
0111	1100	ssss	ssss

Example(s): JUMP NE, label1 ; E=1, branch not taken

Special Notes: The *src* specifier must be immediate data.

JUMP S

Operation: S=1: IP ← IP + *src* (relative) –or— *src* (absolute)
S=0: IP ← IP + 1

Encoding:

15			0
f100	1100	ssss	ssss

Example(s): JUMP S, label1 ; S=0, branch not taken

MOVE *dst, src* Move Data

Description: Moves data from a specified source (*src*) to a specified destination (*dst*). A list of defined source, destination specifiers is given in the table below. Also, since *src* can be either 8-bit (byte) or 16-bit (word) data, the rules governing data transfer are also explained below in the encoding section.

Status Flags: S, Z (if *dst* is Acc or AP or APC)
C, E (if *dst* is PSF)

Operation: *dst* ← *src*

Encoding:

15			0
fddd	dddd	ssss	ssss

Source Specifier Codes

src	src Bit Encoding f ssss ssss	16 or 8 Bits	Description
#k	0 kkkk kkkk	8	kkkkkkkk = immediate (literal) data
MN[n]	1 nnnn 0NNN	8/16	nnnn selects one of first 16 registers in module NNN; where NNN= 0-5. Access to 2nd 16 using PFX[n].
AP	1 0000 1000	8	Accumulator Pointer
APC	1 0001 1000	8	Accumulator Pointer Control
PSF	1 0100 1000	8	Processor Status Flag Register
IC	1 0101 1000	8	Interrupt and Control Register
IMR	1 0110 1000	8	Interrupt Mask Register
SC	1 1000 1000	8	System Control Register
IIR	1 1011 1000	8	Interrupt Identification Register
CKCN	1 1110 1000	8	Clock Control Register
WDCN	1 1111 1000	8	Watchdog Control Register
A[n]	1 nnnn 1001	16	nnnn selects one of 16 accumulators
Acc	1 0000 1010	16	Active Accumulator = A[AP]. Update AP per APC
A[AP]	1 0001 1010	16	Active Accumulator = A[AP]. No change to AP
IP	1 0000 1100	16	Instruction Pointer
@SP--	1 0000 1101	16	16-bit word @SP, post-decrement SP
SP	1 0001 1101	16	Stack Pointer
IV	1 0010 1101	16	Interrupt Vector
LC[n]	1 011n 1101	16	n selects one of 2 loop counter registers
@SPI--	1 1000 1101	16	16-bit word @SP, post-decrement SP, INS=0
@BP[Offs]	1 0000 1110	8/16	Data memory @BP[Offs]
@BP[Offs++]	1 0001 1110	8/16	Data memory @BP[Offs]; post increment OFFS
@BP[Offs--]	1 0010 1110	8/16	Data memory @BP[Offs]; post decrement OFFS
OFFS	1 0011 1110	8	Frame Pointer Offset from Base Pointer (BP)
DPC	1 0100 1110	16	Data Pointer Control Register
GR	1 0101 1110	16	General Register
GRL	1 0110 1110	8	Low byte of GR register
BP	1 0111 1110	16	Frame Pointer Base Pointer (BP)
GRS	1 1000 1110	16	Byte-swapped GR register
GRH	1 1001 1110	8	High byte of GR register
GRXL	1 1010 1110	16	Sign Extended low byte of GR register
FP	1 1011 1110	16	Frame Pointer (BP[Offs])
@DP[n]	1 0n00 1111	8/16	Data memory @DP[n]
@DP[n]++	1 0n01 1111	8/16	Data memory @DP[n], post increment DP[n]
@DP[n]--	1 0n10 1111	8/16	Data memory @DP[n], post decrement DP[n]
DP[n]	1 0n11 1111	16	n selects 1 of 2 data pointers

MOVE *dst, src*
(continued)

Destination Specifier Codes dst	**dst Bit Encoding ddd dddd**	**16 or 8 Bits**	**Description**
NUL	111 0110	8/16	Null (virtual) destination. Intended as a bit bucket to assist software with pointer increments/decrements.
MN[n]	nnn 0NNN	8/16	nnnn selects one of first 8 registers in module NNN; where NNN= 0-5. Access to next 24 using PFX[n].
AP	000 1000	8	Accumulator Pointer
APC	001 1000	8	Accumulator Pointer Control
PSF	100 1000	8	Processor Status Flag Register
IC	101 1000	8	Interrupt and Control Register
IMR	110 1000	8	Interrupt Mask Register
A[n]	nnn 1001	16	nnn selects 1 of first 8 accumulators: A[0]..A[7]
Acc	000 1010	16	Active Accumulator = A[AP].
PFX[n]	nnn 1011	8	nnn selects one of 8 Prefix Registers
@++SP	000 1101	16	16-bit word @SP, pre-increment SP
SP	001 1101	16	Stack Pointer
IV	010 1101	16	Interrupt Vector
LC[n]	11n 1101	16	n selects one of 2 loop counter registers
@BP[Offs]	000 1110	8/16	Data memory @BP[Offs]
@BP[++Offs]	001 1110	8/16	Data memory @BP[Offs]; pre increment OFFS
@BP[--Offs]	010 1110	8/16	Data memory @BP[Offs]; pre decrement OFFS
OFFS	011 1110	8	Frame Pointer Offset from Base Pointer (BP)
DPC	100 1110	16	Data Pointer Control Register
GR	101 1110	16	General Register
GRL	110 1110	8	Low byte of GR register
BP	111 1110	16	Frame Pointer Base Pointer (BP)
@DP[n]	n00 1111	8/16	Data memory @DP[n]
@++DP[n]	n01 1111	8/16	Data memory @DP[n], pre increment DP[n]
@--DP[n]	n10 1111	8/16	Data memory @DP[n], pre decrement DP[n]
DP[n]	n11 1111	16	n selects one of 2 data pointers
2-Cycle Destination Access Using PFX[n] register (see Special Notes)			
SC	000 1000	8	System Control Register
CKCN	110 1000	8	Clock Control Register
WDCN	111 1000	8	Watchdog Control Register
A[n]	nnn 1001	16	nnn selects 1 of second 8 accumulators A[8]..A[15]
GRH	001 1110	8	High byte of GR register

Data Transfer Rules

dst (16-bit) ← *src* (16-bit):	dst[15:0] ← src[15:0]
dst (8-bit) ← *src* (8-bit):	dst[7:0] ← src[7:0]
dst (16-bit) ← *src* (8-bit):	dst[15:8] ← 00h *
	dst[7:0] ← src[7:0]
dst (8-bit) ← *src* (16-bit):	dst[7:0] ← src[7:0]

*** Note:** The PFX[0] register may be used to supply a separate high order data byte for this type of transfer.

Example(s):

```
MOVE  A[0], A[3]          ; A[0] ← A[3]
MOVE DP[0], #110h         ; DP[0] ← #0110h  (PFX[0] register used)
                          ; MOVE PFX[0], #01h (smart-prefixing)
                          ; MOVE DP[0], #10h
MOVE DP[0], #80h          ; DP[0] ← #0080h (PFX[0] register not needed)
```

Special Notes: Proper loading of the PFX[n] registers, when for the purpose of supplying 16-bit immediate data or accessing 2-cycle destinations, is handled automatically by the assembler and is therefore an optional step for the user when writing assembly source code. Examples of the automatic PFX[n] code insertion by the assembler are demonstrated below.

Initial Assembly Code	Assembler Output
MOVE DP[0], #0100h	MOVE PFX[0], #01h MOVE DP[0], #00h
MOVE A[15], A[7]	MOVE PFX[2], any src MOVE A[7], A[7]
MOVE A[8], #3040h	MOVE PFX[2], #30h MOVE A[0], #40h

MOVE *Acc.<b>,* Move Carry Flag to Accumulator Bit

Description: Replaces the specified bit of the active accumulator with the Carry bit.

Status Flags: S, Z

Operation: *Acc.<b>* ← C

Encoding:

15			0
1111	1010	bbbb	1010

Example(s):

```
                          ; Acc = 8000h, S=1, Z=0, C=0
MOVE Acc.15, C            ; Acc = 0000h, S=0, Z=1
```

MOVE C, *Acc.<b>* Move Accumulator Bit to Carry Flag

Description: Replaces the Carry (C) status flag with the specified active accumulator bit.

Status Flag: C

Operation: C ← Acc.<b>

Encoding:

15			0
1110	1010	bbbb	1010

Example(s):

```
                    ; Acc = 01C0h, C=0
MOVE C, Acc.8       ; C =1
```

MOVE C, *src.<b>* Move Bit to Carry Flag

Description: Replaces the Carry (C) status flag with the specified source bit *src.<b>.*

Status Flag: C

Operation: C ← src.<b>

Encoding:

15			0
fbbb	0111	ssss	ssss

Example(s):

```
                    ; M0[0] = FEh; C=1 (assume M0[0] is an 8-bit register)
MOVE C, M0[0].0     ; C=0
```

Special Notes: Only system module 8 and peripheral modules (0-5) are supported by **MOVE C***,src.<b>*.

MOVE C, #0 Clear Carry Flag

Description: Clears the Carry (C) processor status flag.

Status Flag: C ← 0

Operation: C ← 0

Encoding:

15			0
1101	1010	0000	1010

Example(s):

```
                    ; C = 1
MOVE  C, #0         ; C ← 0
```

MOVE C, #1 Set Carry Flag

Description: Sets the Carry (C) processor status flag.

Status Flags: C ← 1

Operation: C ← 1

Encoding:

15			0
1101	1010	0001	1010

Example(s):

```
                    ; C = 0
MOVE C, #1          ; C ← 1
```

MOVE *dst.<b>*, #0 Clear Bit

Description: Clears the bit specified by *dst.<b>.*

Status Flags: C, E (if *dst* is PSF)

Operation: *dst*.<b> ← 0

Encoding:

15			0
1ddd	dddd	0bbb	0111

Example(s):

```
                    ; M0[0] = FEh
MOVE  M0[0].1, #0   ; M0[0] = FCh
MOVE  M0[0].7, #0   ; M0[0] = 7Ch
```

Special Notes: Only system module 8 and peripheral modules (0-5) are supported by **MOVE** *dst.<b>, #0*.

MOVE *dst.<b>*, #1 Set Bit

Description: Sets the bit specified by *dst.<b>.*

Status Flags: C, E (if *dst* is PSF)

Operation: *dst*.<b> ← 1

Encoding:

15			0
1ddd	dddd	1bbb	0111

Example(s):

```
                    ; M0[0] = 00h
MOVE  M0[0].1, #1   ; M0[0] = 02h
MOVE  M0[0].7, #1   ; M0[0] = 82h
```

Special Notes: Only system module 8 and peripheral modules (0-5) are supported by **MOVE** *dst.<b>, #1*.

NEG — Negate Accumulator

Description: Performs a negation (two's complement) of the active accumulator and returns the result back to the active accumulator.

Status Flags: S, Z

Operation: Acc ← ~Acc + 1

Encoding:

15			0
1000	1010	1001	1010

Example(s):

```
                    ; Acc = FEEDh, S=1, Z=0
NEG                 ; Acc = 0113h,   S=0, Z=0
```

OR *src* — Logical OR

Description: Performs a logical-OR between the active accumulator (Acc or A[AP]) and the specified *src* data. For the complete list of *src* specifiers, reference the **MOVE** instruction. Because the source is limited to 8 bits, the PFX[n] register is used to supply the high-byte of data for 16 bit sources.

Status Flags: S, Z

Operation: Acc ← Acc OR *src*

Encoding:

15			0
f010	1010	ssss	ssss

Example(s):

```
                    ; Acc = 2345h for each example
OR A[3]             ; A[3]= 0F0Fh → Acc = 2F4Fh
OR #1133h           ; MOVE PFX[0], #11h (smart-prefixing)
                    ; OR #33h → Acc = 3377h
```

Special Notes: The active accumulator (Acc) is not allowed as the *src* for this operation.

OR *Acc.<b>* — Logical OR Carry Flag with Accumulator Bit

Description: Performs a logical-OR between the Carry (C) status flag and a specified bit of the active accumulator (Acc.<b>) and returns the result to the Carry.

Status Flags: C

Operation: C ← C OR *Acc.<b>*

Encoding:

15			0
1010	1010	bbbb	1010

Example(s):

```
                    ; Acc = 2345h, C=0 at start
OR Acc.1            ; Acc.1=0  → C=0
OR Acc.2            ; Acc.2=1  → C=1
```

POP *dst* Pop Word from the Stack

Description: Pops a single word from the stack (@SP) to the specified *dst* and decrements the stack pointer (SP).

Status Flags: S, Z (if *dst* = Acc or AP or APC)
C, E (if *dst* = PSF)

Operation: *dst* ← @ SP--

Encoding:

15			0
1ddd	dddd	0000	1101

Example(s):

```
POP GR              ; GR ← 1234h
POP @DP[0]          ; @DP[0] ← 76h (WBS0=0)
                    ; @DP[0] ← 0876h (WBS0=1)
```

Stack Data:

xxxxh	
1234h	← SP (initial)
0876h	← SP (after POP GR)
xxxxh	← SP (after POP @DP[0])
xxxxh	

POPI *dst* Pop Word from the Stack Enable Interrupts

Description: Pops a single word from the stack (@SP) to the specified *dst* and decrements the stack pointer (SP). Additionally, **POPI** returns the interrupt logic to a state in which it can acknowledge additional interrupts.

Status Flags: S, Z (if *dst* = Acc or AP or APC)
C, E (if *dst* = PSF)

Operation: *dst* ← @ SP--
INS ← 0

Encoding:

15			0
1ddd	dddd	1000	1101

Example(s): See **POP**

PUSH *src* Push Word to the Stack

Description: Increments the stack pointer (SP) and pushes a single word specified by *src* to the stack (@SP).

Status Flags: None

Operation: SP ← ++SP

Encoding:

15			0
f000	1101	ssss	ssss

Example(s):

```
PUSH GR                 ; GR=0F3Fh
PUSH #40h
```

Stack Data:

xxxxh	
0040h	← SP (after PUSH #40h)
0F3Fh	← SP (after PUSH GR)
xxxxh	← SP (initial)
xxxxh	

RET Return from Subroutine

Description: RET pops a single word from the stack (@SP) into the Instruction Pointer (IP) and decrements the stack pointer (SP). The decremented SP is saved as the new stack pointer (SP).

Status Flags: None

Operation: IP ← @ SP--

Encoding:

15			0
1000	1100	0000	1101

Example(s):

```
RET
```

Code Execution:

Addr (IP)	**Opcode**
0311h	…
0312h	RET
0103h	…

Stack Data:

xxxxh	
xxxxh	
0103h	← SP (before RET)
xxxxh	← SP (after RET)
xxxxh	

RET C / RET NC
RET Z / RET NZ
RET S

Conditional Return on Status Flag

Description: Performs conditional return (RET) based upon the state of a specific processor status flag. **RET C** returns if the Carry flag is set while **RET NC** returns if the Carry flag is clear. **RET Z** returns if the Zero flag is set while **RET NZ** returns if the Zero flag is clear. **RET S** returns if the Sign flag is set. See **RET** for additional information on the return operation.

Status Flags: None

RET C

Operation:
C=1: IP ← @SP--
C=0: IP ← IP + 1

Encoding:

15			0
1010	1100	0000	1101

Example(s):
```
RET C                ; C=1, return (RET) is performed.
```

RET NC

Operation:
C=0: IP ← @SP--
C=1: IP ← IP +1

Encoding:

15			0
1110	1100	0000	1101

Example(s):
```
RET NC               ; C=1, return (RET) does not occur
```

RET Z

Operation:
Z=1: IP ← @SP--
Z=0: IP ← IP + 1

Encoding:

15			0
1001	1100	0000	1101

Example(s):
```
RET Z                ; Z=0, return (RET) does not occur
```

RET NZ

Operation:
Z=0: IP ← @SP--
Z=1: IP ← IP + 1

Encoding:

15			0
1101	1100	0000	1101

Example(s):
```
RET NZ               ; Z=0, return (RET) is performed
```

RET S

Operation:
S=1: IP ← @SP--
S=0: IP ← IP + 1

Encoding:

15			0
1100	1100	0000	1101

Example(s):
```
RET S                ; S=0, return (RET) does not occur
```

RETI Return from Interrupt

Description: RETI pops a single word from the stack (@SP) into the Instruction Pointer (IP) and decrements the stack pointer (SP). Additionally, **RETI** returns the interrupt logic to a state in which it can acknowledge additional interrupts.

Status Flags: None

Operation:
IP ← @SP--
INS← 0

Encoding:

15			0
1000	1100	1000	1101

Example(s): see **RET**

RETI C / RETI NC
RETI Z / RETI NZ
RETI S Conditional Return from Interrupt on Status Flag

Description: Performs conditional return from interrupt (RETI) based upon the state of a specific processor status flag. **RETI C** returns if the Carry flag is set while **RETI NC** returns if the Carry flag is clear. **RETI Z** returns if the Zero flag is set while **RETI NZ** returns if the Zero flag is clear. **RETI S** returns if the Sign flag is set. See **RETI** for additional information on the return from interrupt operation.

Status Flags: None

RETI C

Operation:
C=1: IP ← @SP--
INS← 0
C=0: IP ← IP + 1

Encoding:

15			0
1010	1100	1000	1101

Example(s):
```
RETI C          ; C=1, return from interrupt (RETI) is performed.
```

RETI NC

Operation:
C=0: IP ← @SP--
INS← 0
C=1: IP ← IP +1

Encoding:

15			0
1110	1100	1000	1101

Example(s):
```
RETI NC         ; C=1, return from interrupt (RETI) does not occur
```

RETI Z

Operation:
Z=1: IP ← @SP--
INS← 0
Z=0: IP ← IP + 1

Encoding:

15			0
1001	1100	1000	1101

Example(s):
```
RETI Z                  ; Z=0, return from interrupt (RETI) does not occur
```

RETI NZ

Operation:
Z=0: IP ← @SP--
INS← 0
Z=1: IP ← IP + 1

Encoding:

15			0
1101	1100	1000	1101

Example(s):
```
RETI NZ                 ; Z=0, return from interrupt (RETI) is performed
```

RETI S

Operation:
S=1: IP ← @SP--
INS← 0
S=0: IP ← IP + 1

Encoding:

15			0
1100	1100	1000	1101

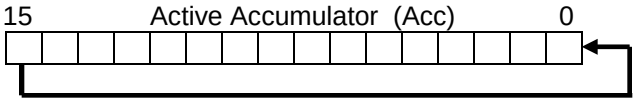
Example(s):
```
RETI S                  ; S=0, return from interrupt (RETI) does not occur
```

RL / RLC — Rotate Left Accumulator Carry Flag (Ex/In)clusive

Description: Rotates the active accumulator left by a single bit position. The **RL** instruction circulates the msb of the accumulator (bit 15) back to the lsb (bit 0) while the **RLC** instruction includes the Carry (C) flag in the circular left shift.

Status Flags: C (for RLC only), S, Z (for RLC only)

RL Operation:



Acc.[15:1]← Acc.[14:0]; Acc.0 ← Acc.15

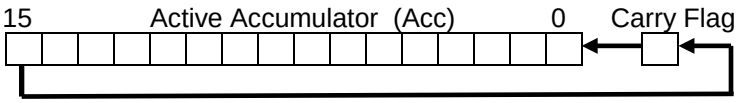
Encoding:

15			0
1000	1010	0100	1010

Example(s):

```
                    ; Acc = A345h, S=1, Z=0
RL                  ; Acc = 468Bh, S=0, Z=0
RL                  ; Acc = 8D16h, S=1, Z=0
```

RLC Operation:



Acc.[15:1]← Acc.[14:0]; Acc.0 ← C; C ← Acc.15

Encoding:

15			0
1000	1010	0101	1010

Example(s):

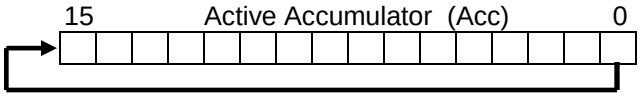
```
                    ; Acc = A345h, C=1, S=1, Z=0
RLC                 ; Acc = 468Bh, C=1, S=0, Z=0
RLC                 ; Acc = 8D17h, C=0, S=1, Z=0
```

RR / RRC — Rotate Right Accumulator Carry Flag (Ex/In)clusive

Description: Rotates the active accumulator right by a single bit position. The **RR** instruction circulates the lsb of the accumulator (bit 0) back to the msb (bit 15) while the **RRC** instruction includes the Carry (C) flag in the circular right shift.

Status Flags: C (for RRC only), S, Z (for RRC only)

RR Operation:



Acc.[14:0]← Acc.[15:1]; Acc.15 ← Acc.0

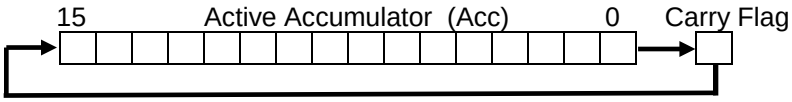
Encoding:

15			0
1000	1010	1100	1010

Example(s):

```
                    ; Acc = A345h, S=1, Z=0
RR                  ; Acc = D1A2h, S=1, Z=0
RR                  ; Acc = 68D1h, S=0, Z=0
```

RRC Operation:



Acc.[14:0]← Acc.[15:1]; Acc.15 ← C; C ← Acc.0

Encoding:

15			0
1000	1010	1101	1010

Example(s):

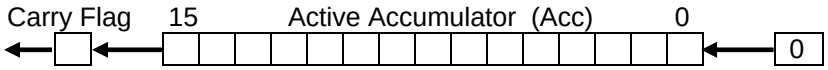
```
                    ; Acc = A345h, C=1, S=1, Z=0
RRC                 ; Acc = D1A2h, C=1, S=1, Z=0
RRC                 ; Acc = E8D1h, C=0, S=1, Z=0
```

SLA / SLA2 / SLA4 Shift Accumulator Left Arithmetically One, Two, or Four Times

Description: Shifts the active accumulator left once, twice, or four times respectively for **SLA, SLA2**, and **SLA4**. For each shift iteration, a '0' is shifted into the lsb and the msb is shifted into the Carry (C) flag. For signed data, this shifting process effectively retains the sign orientation of the data to the point at which overflow/underflow would occur.

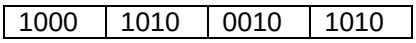
Status Flags: C, S, Z

SLA Operation:



C ← Acc.15; Acc.[15:1]← Acc.[14:0]; Acc.0 ← 0

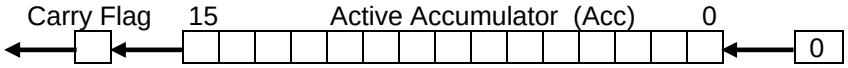
Encoding:



Example(s):

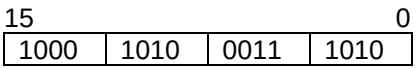
```
                    ; Acc = E345h, C=0, S=1, Z=0
SLA                 ; Acc = C68Ah, C=1, S=1, Z=0
SLA                 ; Acc = 8D14h, C=1, S=1, Z=0
```

SLA2 Operation:



C ← Acc.14; Acc.[15:2]← Acc.[13:0]; Acc.[1:0] ← 0

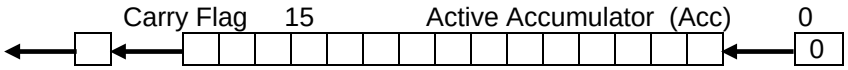
Encoding:



Example(s):

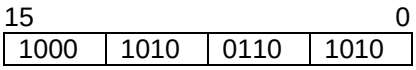
```
                    ; Acc = E345h, C=0, S=1, Z=0
SLA2                ; Acc = 8D14h, C=1, S=1, Z=0
```

SLA4 Operation:



C ← Acc.12; Acc.[15:4]← Acc.[11:0]; Acc.[3:0] ← 0

Encoding:



Example(s):

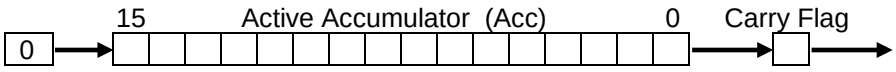
```
                    ; Acc = E345h, C=0, S=1, Z=0
SLA4                ; Acc = 3450h, C=0, S=0, Z=0
```

SR — Shift Accumulator Right
SRA / SRA2 / SRA4 — Shift Accumulator Right Arithmetically One, Two, or Four Times

Description: Shifts the active accumulator right once for the **SR**, **SRA** instructions and 2 or 4 times respectively for the **SRA2**, **SRA4** instructions. The **SR** instruction shifts a 0 into the accumulator msb while the **SRA**, **SRA2**, and **SRA4** instructions effectively shift a copy of the current msb into the accumulator, thereby preserving any sign orientation. For each shift iteration, the accumulator lsb is shifted into the Carry (C) flag.

Status Flags: C, S (changes for SR only), Z

SR Operation:



Acc.15 ← 0; Acc.[14:0]← Acc.[15:1]; C ← Acc.0

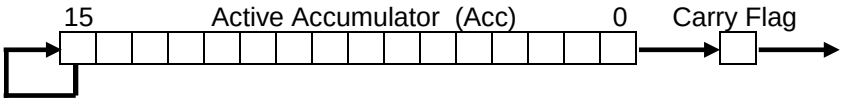
Encoding:

15			0
1000	1010	1010	1010

Example(s):

```
                    ; Acc = A345h, C=1, S=1, Z=0
SR                  ; Acc = 51A2h, C=1, S=0, Z=0
SR                  ; Acc = 28D1h, C=0, S=0, Z=0
```

SRA Operation:



Acc.[14:0]← Acc.[15:1]
Acc.15 ← Acc.15
C ← Acc.0

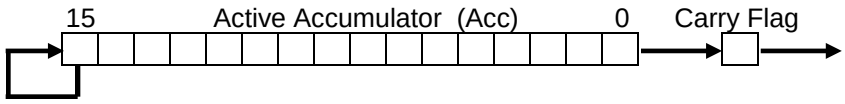
Encoding:

15			0
1000	1010	1111	1010

Example(s):

```
                    ; Acc = 0003h, C=0, Z=0
SRA                 ; Acc = 0001h, C=1, Z=0
SRA                 ; Acc = 0000h, C=1, Z=1
```

SRA2 Operation:



Acc.[13:0]← Acc.[15:2]
Acc.[15:14] ← Acc.15
C ← Acc.1

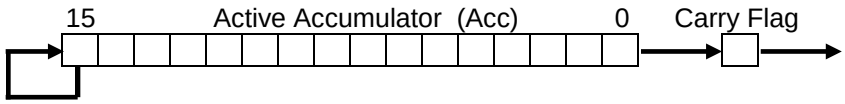
Encoding:

15			0
1000	1010	1110	1010

Example(s):

```
                    ; Acc = 0003h, C=0, Z=0
SRA2                ; Acc = 0000h, C=1, Z=1
```

SRA4 Operation:



Acc.[11:0]← Acc.[15:4]
Acc.[15:12] ← Acc.15
C ← Acc.3

Encoding:

15			0
1000	1010	1011	1010

Example(s):

```
                    ; Acc = 9878h, C=0, Z=0
SRA4                ; Acc = F987h, C=1, Z=0
SRA4                ; Acc = FF98h, C=0, Z=0
```

SUB / SUBB *src* — Subtract / Subtract with Borrow

Description: Subtracts the specified *src* from the active accumulator (Acc) and returns the result back to the active accumulator. The **SUBB** additionally subtracts the borrow (Carry Flag) which may have resulted from previous subtraction. For the complete list of *src* specifiers, reference the **MOVE** instruction. Because the source is limited to 8 bits, the PFX[n] register is used to supply the high-byte of data for 16 bit sources.

Status Flags: C, S, Z, OV

SUB Operation: Acc ← Acc - *src*

Encoding:

15			0
f101	1010	ssss	ssss

Example(s):

```
                        ; Acc = 2345h to start, A[1]= 1250h
SUB     A[1]            ; Acc = 10F5h, C=0, S=0, Z=0, OV=0
SUB     A[1]            ; Acc = FEA5h, C=1, S=1, Z=0, OV=0
SUB     A[2]            ; A[2] =7FFFh
                        ; ➔ Acc = 7EA6h; C=0, S=0, Z=0, OV=1
```

SUBB Operation: Acc ← Acc – (*src* + C*)*

Encoding:

15			0
f111	1010	ssss	ssss

Example(s):

```
                        ; Acc = 2345h, A[1]= 1250h, C=1
SUBB    A[1]            ; Acc = 10F4h, C=0, S=0, Z=0
SUBB    A[1]            ; Acc = FEA4h, C=1, S=1, Z=0
```

Special Notes: The active accumulator (Acc) is not allowed as the *src* for these operations.

XCH — Exchange Accumulator Bytes

Description: Exchanges the upper and lower bytes of the active accumulator.

Status Flags: S

Operation:
Acc.[15:8] ← Acc.[7:0]
Acc.[7:0] ← Acc.[15:8]

Encoding:

15			0
1000	1010	1000	1010

Example(s):

```
                    ; Acc = 2345h
XCH                 ; Acc = 4523h
```

XCHN — Exchange Accumulator Nibbles

Description: Exchanges the upper and lower nibbles in the active accumulator byte(s).
Status Flags: S

Operation:
Acc.[7:4] ← Acc.[3:0]
Acc.[3:0] ← Acc.[7:4]
Acc.[15:12] ← Acc.[11:8]
Acc.[11:8] ← Acc.[15:12]

Encoding:

15			0
1000	1010	0111	1010

Example(s):

```
                    ; Acc = 2345h
XCHN                ; Acc = 3254h
```

XOR *src* Logical XOR

Description: Performs a logical-XOR between the active accumulator (Acc or A[AP]) and the specified *src* data. For the complete list of *src* specifiers, reference the MOVE instruction. Because the source is limited to 8 bits, the PFX[n] register is used to supply the high-byte of data for 16 bit sources.

Status Flags: S, Z

Operation: Acc ← Acc XOR *src*

Encoding:

15			0
f011	1010	ssss	ssss

Example(s):

```
                    ; Acc = 2345h
XOR A[2]            ; A[2]=0F0Fh; Acc ← 2C4Ah
```

Special Notes: The active accumulator (Acc) is not allowed as the *src* for this operation.

XOR *Acc.<b>* Logical XOR Carry Flag with Accumulator Bit

Description: Performs a logical-XOR between the Carry (C) status flag and a specified bit of the active accumulator (Acc.<b>) and returns the result to the Carry.

Status Flags: C

Operation: C ← C XOR *Acc.<b>*

Encoding:

15			0
1011	1010	bbbb	1010

Example(s):

```
                    ; Acc = 2345h, C=1 at start
XOR Acc.1           ; Acc.1=0  → C=1
XOR Acc.2           ; Acc.2=1  → C=0
```

SECTION 25 – UTILITY ROM

25.1 – Overview

The DS4830A utility ROM includes routines that provide the following functions to application software:

- In-application programming routines for flash memory (program, erase, mass erase)
- Single word/byte copy and buffer copy routines for lookup tables in flash

To provide backwards compatibility among different versions of the utility ROM, a function address table is included that contains the entry points for all user-callable functions. With this table, user code can determine the entry point for a given function as follows:

1. Read the location of the function address table from address 0800Dh in the utility ROM.
2. The entry points for each function listed below are contained in the function address table, one word per function, in the order given by their function numbers.

For example, the entry point for the UROM_flashEraseAll function can be determined by the following procedure.

1. functionTable = romMemory[800Dh]
2. flashEraseAllEntry = romMemory[functionTable + 2]

It is also possible to call utility ROM functions directly, using the entry points given in Table 25-1. Calling a function directly will provide faster code execution.

Table 25-1: DS4830A Utility ROM Functions

INDEX	FUNCTION NAME	ENTRY POINT	SUMMARY
0	UROM_flashWrite	843Ch	Programs a single word of flash memory.
1	UROM_flashErasePage	845Fh	Erases (programs to FFFFh) a 256-word page of flash.
2	UROM_flashEraseAll	8475h	Erases (programs to FFFFh) all flash memory.
3	UROM_moveDP0	8484h	Reads a byte/word at DP[0].
4	UROM_moveDP0inc	8487h	Reads a byte/word at DP[0], then increments DP[0].
5	UROM_moveDP0dec	848Ah	Reads a byte/word at DP[0], then decrements DP[0].
6	UROM_moveDP1	848Dh	Reads a byte/word at DP[1].
7	UROM_moveDP1inc	8490h	Reads a byte/word at DP[1], then increments DP[0].
8	UROM_moveDP1dec	8493h	Reads a byte/word at DP[1], then decrements DP[0].
9	UROM_moveBP	8496h	Reads a byte/word at BP[OFFS].
10	UROM_moveBPinc	8499h	Reads a byte/word at BP[OFFS], then increments OFFS.
11	UROM_moveBPdec	849Ch	Reads a byte/word at BP[OFFS], then decrements OFFS.
12	UROM_copyBuffer	849Fh	Copies LC[0] bytes/words (up to 255) from DP[0] to BP[OFFS].

25.2 – In-Application Programming Functions

25.2.1 – UROM_flashWrite

Function	UROM_flashWrite
Summary	Programs a single word of flash memory
Inputs	A[0]: Word address in program flash memory to write. A[1]: Value to write to flash memory.
Outputs	Carry: Set on error and cleared on success
Destroys	PSF, LC[1]

Notes:

- This function uses two stack levels to save and restore values.
- If the watchdog reset function is active, it should be disabled before calling this function.
- Interrupts are disabled while in this function.
- If the flash location has already been programmed to a value other than FFFFh, this function returns with an error (Carry set). In order to reprogram a flash location, the location must first be erased by calling UROM_flashErasePage or UROM_flashEraseAll.

25.2.2 – UROM_flashErasePage

Function	UROM_flashErasePage
Summary	Erases (programs to FFFFh) a 512-byte page of flash memory.
Inputs	A[0]: Word address located in the page to be erased. (The page number is the high 8 bits of A[0].)
Outputs	Carry: Set on error and cleared on success.
Destroys	PSF, LC[1], GR, AP, APC, A[0]

Notes:

- If the watchdog reset function is active, it should be disabled before calling this function.
- Interrupts are disabled while in this function.
- When calling this function from flash, care should be taken that the return address is not in the page which is being erased.

25.2.3 – UROM_flashEraseAll

Function	UROM_flashEraseAll
Summary	Erases (programs to FFFFh) all locations in flash memory
Inputs	None
Outputs	Carry: Set on error and cleared on success.
Destroys	PSF, GR, LC[1], LC[0], AP, APC, A[0]

Notes:

- If the watchdog reset function is active, it should be disabled before calling this function.
- Interrupts are disabled while in this function..
- This function can only be called by code running from the RAM. Attempting to call this function while running from the flash results in an error.

25.3 – Data Transfer Functions

The DS4830A cannot access data from the same memory segment that is currently being used for instructions. For example, when instructions are executing from FLASH, data in FLASH cannot be accessed. The following utility ROM functions can be used to transfer data from one memory segment to another. For example, if data in FLASH needs to be copied to SRAM, one of these ROM functions can be called to do this transfer. This is useful when code is executing from FLASH and access to lookup tables or non-volatile data that is stored in FLASH is required. These functions can also be used by code running from SRAM to read data that is stored in SRAM.

Since these functions are executed from utility ROM, addresses must be specified correctly to point to the intended memory segments. When executing from utility ROM, the memory map is illustrated in Figure 2-4. For example, data located at word address 0100h in the FLASH must be accessed at word address 8100h (or byte address 8200h) when using any of the functions listed in the following sections.

25.3.1 – UROM_moveDP0

Function	**UROM_moveDP0**
Summary	Reads the byte/word value pointed to by DP[0].
Inputs	DP[0]: Address to read from data space (include 8000h offset if reading from flash).
Outputs	GR: Data byte/word read.
Destroys	None

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 25-1. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.
- This function automatically selects DP[0] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[0]

25.3.2 – UROM_moveDP0inc

Function	**UROM_moveDP0inc**
Summary	Reads the byte/word value pointed to by DP[0], then increments DP[0].
Inputs	DP[0]: Address to read from data space (include 8000h offset if reading from flash).
Outputs	GR: Data byte/word read.DP[0] is incremented.
Destroys	None

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 25-1. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.
- This function automatically selects DP[0] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[0]++

25.3.3 – UROM_moveDP0dec

Function	UROM_moveDP0dec
Summary	Reads the byte/word value pointed to by DP[0], then decrements DP[0].
Inputs	DP[0]: Address to read from data space (include 8000h offset if reading from flash).
Outputs	GR: Data byte/word read. DP[0] is decremented.
Destroys	None

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 25-1. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.
- This function automatically selects DP[0] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[0]--

25.3.4 – UROM_moveDP1

Function	UROM_moveDP1
Summary	Reads the byte/word value pointed to by DP[1].
Inputs	DP[1]: Address to read data space (include 8000h offset if reading from flash).
Outputs	GR: Data byte/word read.
Destroys	None

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 25-1. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.
- This function automatically selects DP[1] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[1]

25.3.5 – UROM_moveDP1inc

Function	UROM_moveDP1inc
Summary	Reads the byte/word value pointed to by DP[1], then increments DP[1].
Inputs	DP[1]: Address to read from data space (include 8000h offset if reading from flash).
Outputs	GR: Data byte/word read. DP[1] is incremented.
Destroys	None

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 25-1. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.
- This function automatically selects DP[1] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[1]++

25.3.6 – UROM_moveDP1dec

Function	UROM_moveDP1dec
Summary	Reads the byte/word value pointed to by DP[1], then decrements DP[1].
Inputs	DP[1]: Address to read from data space (include 8000h offset if reading from flash).
Outputs	GR: Data byte/word read. DP[1] is decremented.
Destroys	None

Notes:

- Before calling this function, DPC should be set appropriately to configure DP[1] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 25-1. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.
- This function automatically selects DP[1] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[1]--

25.3.7 – UROM_moveBP

Function	UROM_moveBP
Summary	Reads the byte/word value pointed to by BP[OFFS].
Inputs	BP[OFFS]: Address to read from data space (include 8000h offset if reading from flash).
Outputs	GR: Data byte/word read.
Destroys	None.

Notes:

- Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 25-1. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.
- This function automatically selects BP[OFFS] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @BP[OFFS]

25.3.8 – UROM_moveBPinc

Function	UROM_moveBPinc
Summary	Reads the byte/word value pointed to by BP[OFFS], then increments OFFS.
Inputs	BP[OFFS]: Address to read from data space (include 8000h offset if reading from flash).
Outputs	GR: Data byte/word read. OFFS is incremented.
Destroys	None

Notes:

- Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 25-1. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.
- This function automatically selects BP[OFFS] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @BP[OFFS++]

25.3.9 – UROM_moveBPdec

Function	UROM_moveBPdec
Summary	Reads the byte/word value pointed to by BP[OFFS], then decrements OFFS.
Inputs	BP[OFFS]: Address to read from data space (include 8000h offset if reading from flash).
Outputs	GR: Data byte/word read. OFFS is decremented.
Destroys	None

Notes:

- Before calling this function, DPC should be set appropriately to configure BP[OFFS] for byte or word mode.
- The address passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 25-1. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.
- This function automatically selects BP[OFFS] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @BP[OFFS--]

25.3.10 – UROM_copyBuffer

Function	UROM_copyBuffer
Summary	LC[0] bytes/words (up to 256) from DP[0] to BP[OFFS].
Inputs	DP[0]: Starting address to copy from. BP[OFFS]: Starting address to copy to. LC[0]: Number of bytes/words to copy.
Outputs	OFFS is incremented by LC[0]. DP[0] is incremented by LC[0].
Destroys	LC[0]

Notes:

- This function can be used to copy from program flash to data RAM, or from one part of data RAM to another. It cannot be used to copy data into flash memory; the UROM_writeFlash function should be used for this purpose.
- Before calling this function, DPC should be set appropriately to configure DP[0] and BP[OFFS] for byte or word mode. Both DP[0] and BP[OFFS] should be configured to the same mode (byte or word) for correct buffer copying.
- The addresses passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 25-1. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.
- This function automatically selects the data pointers before reading the byte/word values.

25.4 Special Functions

The DS4830A provides software reset and read single word functions.

25.4. 1 – UROM_copyWord

Function	UROM_copyWord
Summary	1 word DP[0] to A[0].
Inputs	DP[0]: Starting address to copy from.
Outputs	A[0]
Destroys	A[0], DP[0]
UROM Address	885Bh

Notes:

- This function can be used to copy a word from program flash to data RAM, or from one part of data RAM to another.
- Before calling this function, DPC should be set appropriately to configure DP[0] for byte or word mode.
- The addresses passed to this function should be based on the data memory mapping for the utility ROM, as shown in Figure 25-1. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.
- This function automatically selects the data pointers before reading the byte/word values.

25.4. 2 – Software Reset

UROM has necessary code at the location 8854h which can generate an internal reset when application software jumps to this location.

25.5 – Utility ROM Examples

25.5.1 – Reading Constant Word Data from Flash

```
UROM_moveDP0inc  equ  08487h

move  DPC, #1Ch                    ; Set all pointers to word mode
move DP[0], #(table + 8000h)       ; Point to address of data as viewed in the Utility ROM memory map
lcall #UROM_moveDP0inc
move A[0], GR
; A[0] = 1111h
lcall #UROM_moveDP0inc
move A[1], GR                      ; A[1] = 2222h
lcall #UROM_moveDP0inc
move A[2], GR
; A[0] = 3333h
lcall #UROM_moveDP0inc
move A[3], GR                      ; A[1] = 4444h
sjump $

org 0100h
table:
   dw 1111h, 2222h, 3333h, 4444h
```

25.5.2 – Reading Constant Byte Data from Flash (Indirect Function Call)

```
INDX_moveDP0inc  equ  4

move DPC, #1Ch                     ; Set all pointers to word mode
move DP[0], #800Dh                 ; Fetch location of function table from Utility ROM
move BP, @DP[0]                    ; Set base pointer to function table location
move Offs, #INDX_moveDP0inc        ; Set offset to moveDP0inc entry in table
move A[7], @BP[Offs]               ; Get address of moveDP0inc function
move DPC, #00h                     ; Set all pointers to byte mode
move DP[0], #((table * 2) + 8000h) ; Point to address of data as viewed in the Utility ROM memory map  and
convert
                                   ; to byte mode pointer
lcall A[7]                         ; moveDP0inc
move A[0], GR                      ; A[0] = 34h
lcall A[7]                         ; moveDP0inc
move A[1], GR                      ; A[1] = 12h
lcall A[7]                         ; moveDP0inc
move A[2], GR                      ; A[2] = 78h
lcall A[7]                         ; moveDP0inc
move A[3], GR                      ; A[3] = 56h
sjump $

org 0100h
table:
   dw 1234h, 5678h
```

SECTION 26 – MISCELLANEOUS

26.1 – Overview

Miscellaneous features of DS4830A are

- CRC8
- Software interrupts
- General-purpose registers.

26.2 – CRC8

DS4830A has an built-in hardware CRC8. The registers used for CRC8 are CRC8IN and CRC8OUT. They are defined in Module 1. SMBus 2.0 specification is followed for CRC algorithm (CRC polynomial is x^8+x^2+x+1).

26.2.1 – CRC Data In (CRC8IN)

Bit	7	6	5	4	3	2	1	0
Name	CRC8IN_7	CRC8IN_6	CRC8IN_5	CRC8IN_4	CRC8IN_3	CRC8IN_2	CRC8IN_1	CRC8IN_0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7:0	CRC8IN[7:0]	**CRC Data in.** The user program writes data to this register for which CRC8 should be applied to.

26.2.2 – CRC Data Out (CRC8OUT)

Bit	7	6	5	4	3	2	1	0
Name	CRC8OUT_7	CRC8OUT_6	CRC8OUT_5	CRC8OUT_4	CRC8OUT_3	CRC8OUT_2	CRC8OUT_1	CRC8OUT_0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7:0	CRC8OUT[7:0]	**CRC Data out.** The user program reads CRC8 result from this register for all the data that was written to CRC8IN. **Note:** This register has to be cleared to 0x00 by software before starting a CRC8 calculation.

26.2.3 – Example

```
unsigned char Calculate_CRC8(unsigned char* data, int length)
{
        unsigned int i = 0;
        unsigned char CRC_result;
        CRC8OUT = 0x00;
        for( ; i<length ; i++)
        {
                CRC8IN = data[i];
                //Incrementing i in the loop takes a cycle atleast. So CRC should have been completed in this time.
        }
        CRC_result = CRC8OUT;
        return CRC_result;
}
```

26.3 – Software Interrupts

The DS4830A has four software interrupts that the application program can use to generate interrupts for general-purpose application requirements. The user can generate an interrupt by setting a bit in the USER_INT[3:0]. The USER_INT[7:4] are single cycle read/write bits which can be used in the time critical interrupts.

26.3.1 – User Interrupt Register (USER_INT)

Bit	7	6	5	4	3	2	1	0
Name	SW_F3	SW_F2	SW_F1	SW_F0	SW_INT4	SW_INT 3	SW_INT 2	SW_INT 1
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7:4	SW_F3[3:0]	**Software flags:** Single cycle read/write bits for general-purpose flags for application usage.
3:0	SW_INT[3:0]	**Software Interrupt:** Setting this bit to '1' generates an interrupt.

26.4 – General-Purpose Registers

DS4830A has 16 general-purpose registers defined in Module 3. Reading from GP_REG1 and GP_REG2 take a single clock cycle and writing to these registers takes two clock cycles. For GP_REG3 to GP_REG16, reading from and writing to take two clock cycles. These registers can be used by time critical software in place of program variables to save clock cycles during memory access.

26.4.1 – General-Purpose Register

(GP_REG1, GP_REG2, GP_REG3, GP_REG4, GP_REG5, GP_REG6, GP_REG7, GP_REG8, GP_REG9, GP_REG10, GP_REG11, GP_REG12, GP_REG13, GP_REG14, GP_REG15, GP_REG16)

Bit	15	14	13	12	11	10	9	8
Name	GP_REGx_15	GP_REGx_14	GP_REGx_13	GP_REGx_12	GP_REGx_11	GP_REGx_10	GP_REGx_9	GP_REGx_8
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	Rw	rw	rw	rw

Bit	7	6	5	4	3	2	1	0
Name	GP_REGx_7	GP_REGx_6	GP_REGx_5	GP_REGx_4	GP_REGx_3	GP_REGx_2	GP_REGx_1	GP_REGx_0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	Rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:0	GP_REGx_n	**General-Purpose Register x Bit n.** The software can use these bits in place of variables.

26.5 – Device Number and I^2C Bootloader Address Disable

The DS4830A has DEV_NUM register which is used to disable the bootloader slave address (34h). On POR, this register is initialized to default value.

26.5.1 – Device Number Register (DEV_NUM)

Bit	7	6	5	4	3	2	1	0
Name	BOOT_DIS	DEV_NUM[6:0]						
Reset	0	x	x	x	x	x	x	x
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7	BOOT_DIS	**BOOT DIS flags:** Setting this bit to '1' will disable the bootloader slave address (34h). On POR, this bit is set to '0'.
6:0	DEV_NUM	**DEV_NUM:** The DEV_NUM[6:0] is configured in the production for indication and tracibility purpose.