DS4835/36 Microcontroller User's Guide

Rev 0.4; 3/21/2019

SECTION 1 – OVERVIEW

The DS4835/36 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 with built-in DC-DC converters and TEC controllers. The DS4835/36 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 DS4835/36 provides for monitoring and controlling an optical system include the following:

- Integrated PMIC and Analog Saves Space
  - 4 DC-DC Converters (3 Buck, 1 Buck/Boost*)
  - 2 TEC Controllers with ±0.05°C Accuracy
  - 4 Current DACs
  - Inrush Current Control
  - Internal 128MHz Oscillator
- Simplifies Complex Optical Module Design
  - 16-Bit 500ksps ADC (with up to 24 External Channels)
  - Internal and External Temperature Sensor, ±3°C
  - 16-Bit Fast Comparator with 8 Input Mux
  - 12 16-Bit Delta Sigma DAC
  - 4 Programmable logic Arrays
  - Slave Communication Interface: I^2C Compatible 2-Wire 400kHz without Clock Stretching
  - Master Communication Interface: 400kHz I^2C Compatible 2-Wire, SPI or Maxim 3-Wire Laser Driver
  - Two Independent Sample and Holds (300ns Sample Time)
  - 23 GPIO and 4 GPI pins
  - 8x16-Bit PWM Channels
- Microcontroller Features Enable Flexibility
  - 16-Bit Low Power Microcontroller
  - 64KBytes Flash Program Memory
  - 4KBytes Data RAM
  - 8KBytes ROM Memory
  - 32 Level Hardware Stack
  - Supply Voltage Monitor (SVM) and Brownout Monitor
  - I^2C Bootloader
  - 1-Wire In-System Debug and Programming
  - Dual Hardware MAC
  - 5mm x 5mm, 40-Pin TQFN Package
  - Three 16-Bit Timers and One Programmable Watchdog Timer

*DS4836 does not support 4V boost with internal FET.

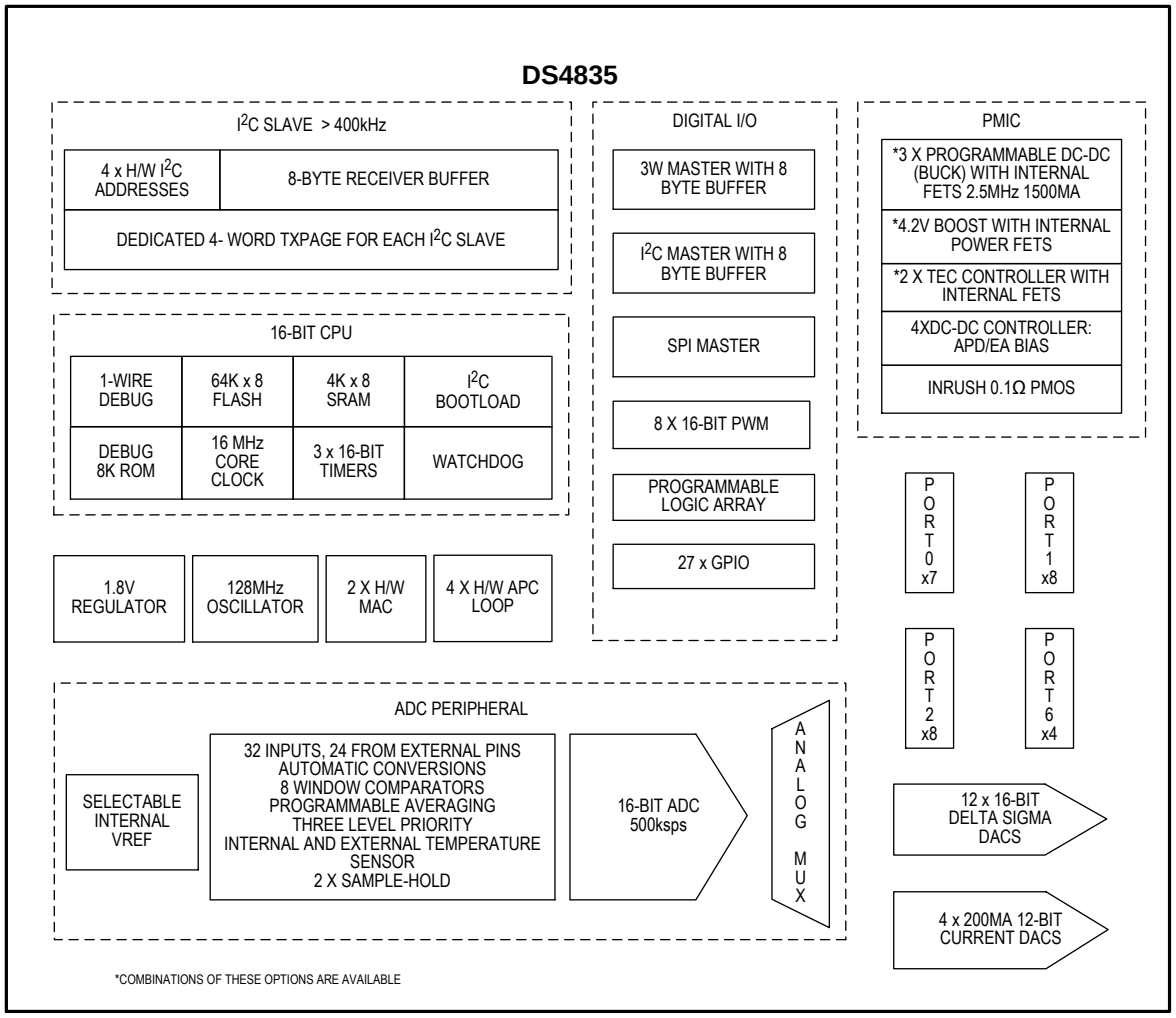


Figure 1-1: DS4835 Block Diagram

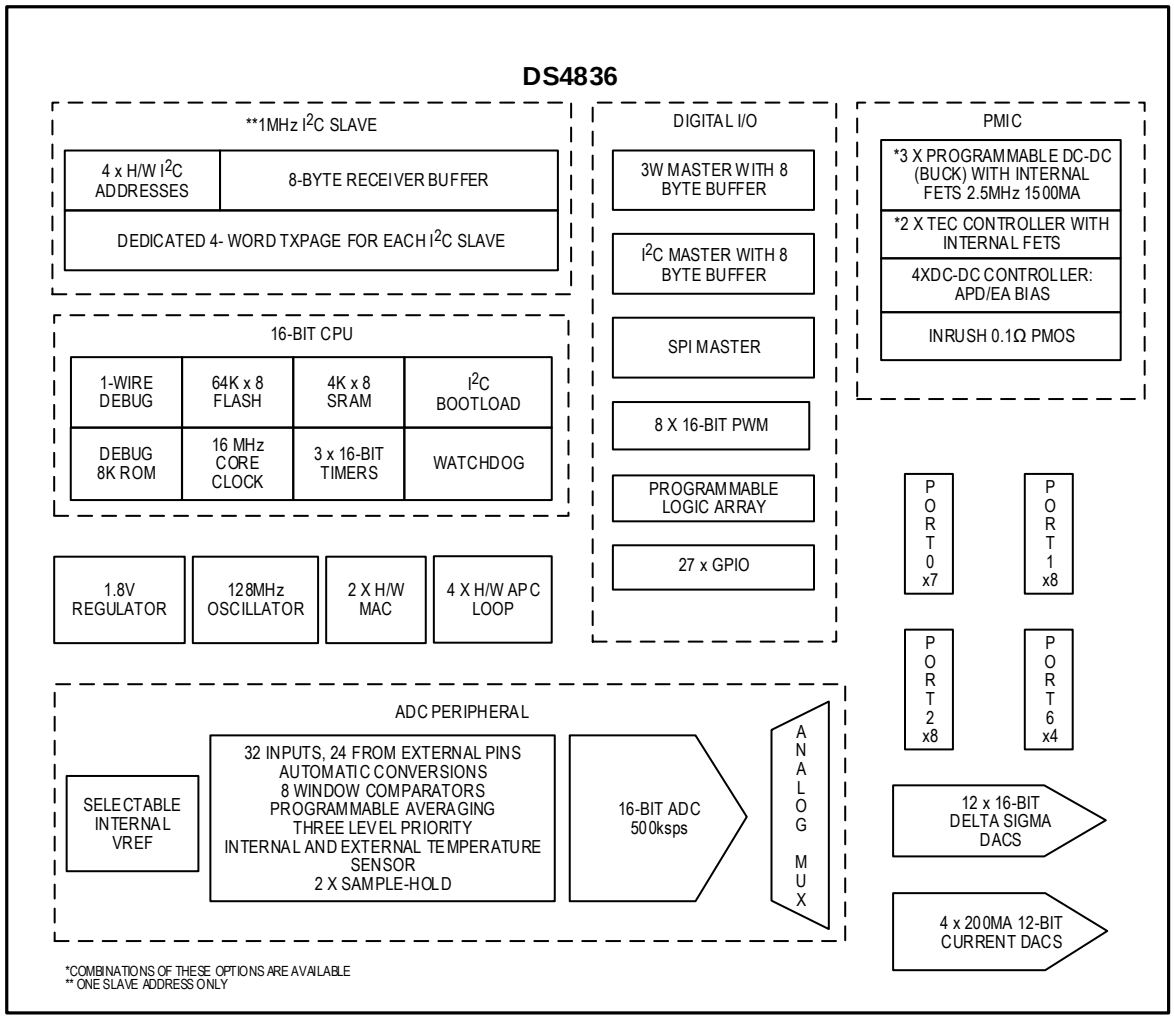


Figure 1-2: DS4836 Block Diagram

This document is provided as a supplement to the DS4835/36 IC data sheet. This user's guide provides the information necessary to develop applications using the DS4835/36. All electrical and timing specifications, pin descriptions, package information, and ordering information can be found in the DS4835/36 IC data sheet.

SECTION 2 – ARCHITECTURE

The DS4835/36 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 DS4835/36 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 DS4835/36 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 DS4835/36 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 DS4835/36'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 DS4835/36 provides a total of 16 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 DS4835/36 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 seven modules (Modules 0h through 6h). The Peripheral Registers in the DS4835/36 are used for functionalities such as ADC, Fast Comparator (Digital Comparator), Delta Sigma DAC, DC-DC Buck and Boost Converters, In-rush controller, Timers, 3-Wire, I^2C Master and Slave, SPI Master, GPIO, 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 DS4835/36 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 DS4835/36 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 DS4835/36 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 (1-Wire 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 DS4835/36 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 4 KWord 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 1-Wire or I^2C-compatible interfaces
- In-circuit debug routines
- Routines for in-application flash programming

Following any reset, the DS4835/36 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 DS4835/36 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 DS4835/36 is implemented using Harvard architecture, with separate busses for program and data memory. The Memory Management Unit (MMU) allows the DS4835/36 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 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 DS4835/36 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 DS4835/36'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. 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.

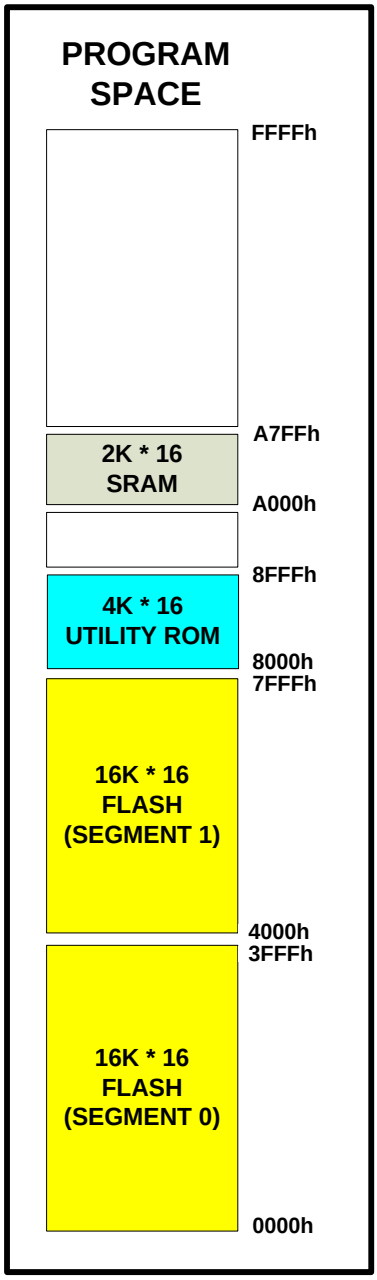


Figure 2-2: Program Memory Mapping

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 DS4835/36 provides three pointers that can be used for indirect accessing of data memory. The DS4835/36 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 DS4835/36 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 DS4835/36'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 (Lower Half)	P0 and P1 (All)
1	P1 (Upper Half)	P0 and P1 (All)

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.

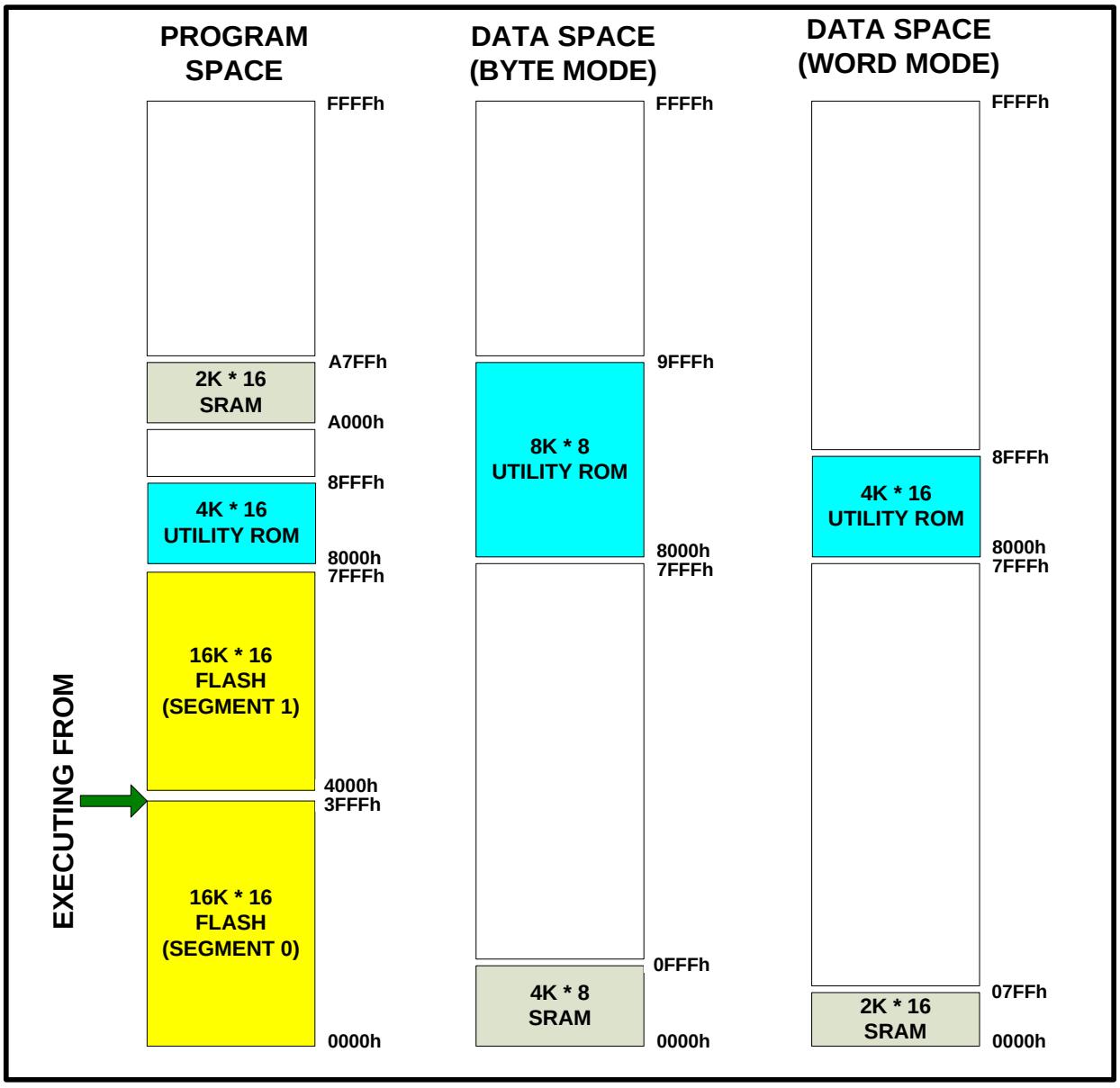


Figure 2-3: Memory Map When Executing from Flash Memory

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.

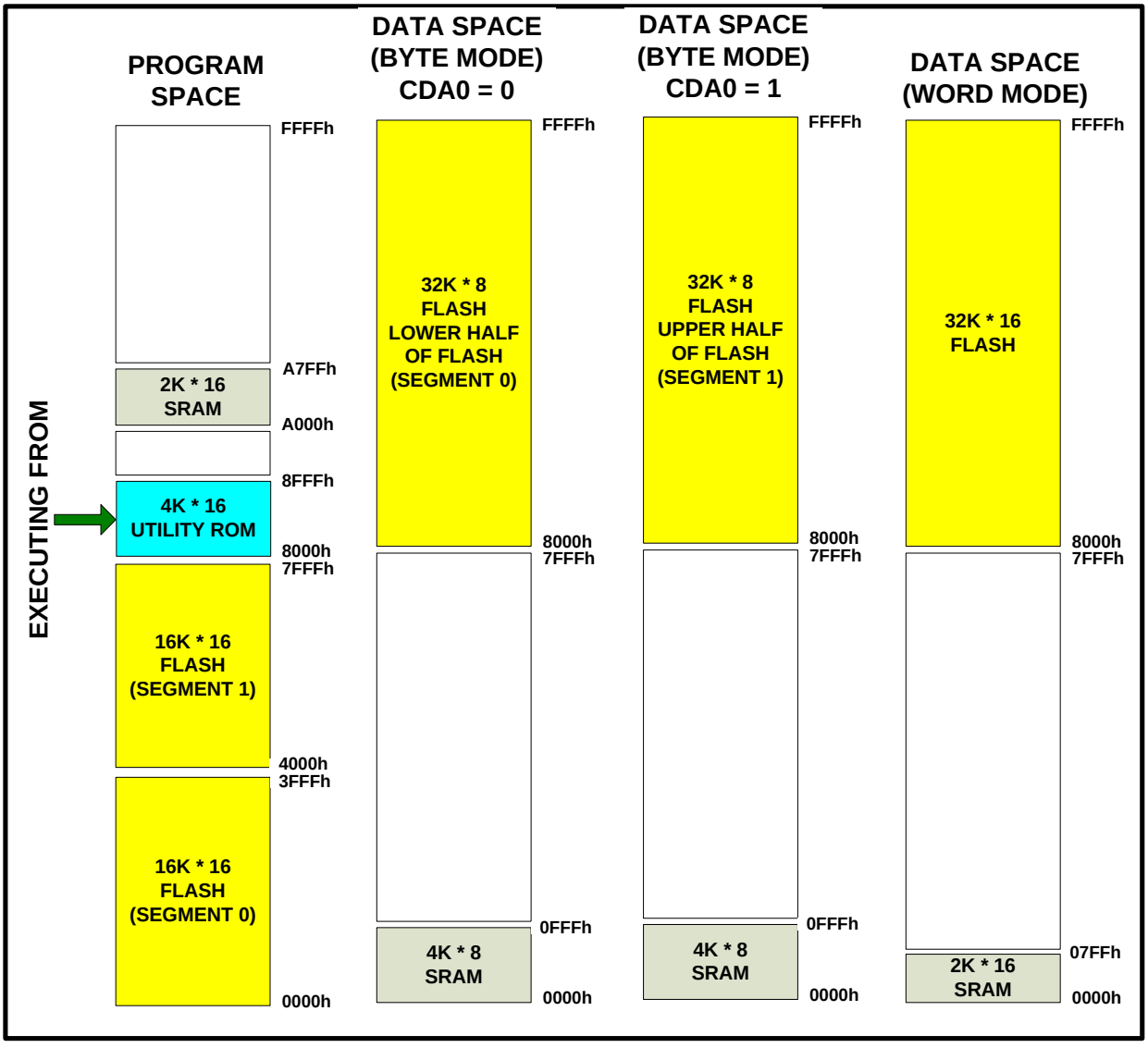


Figure 2-4: Memory Map When Executing from Utility ROM

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.

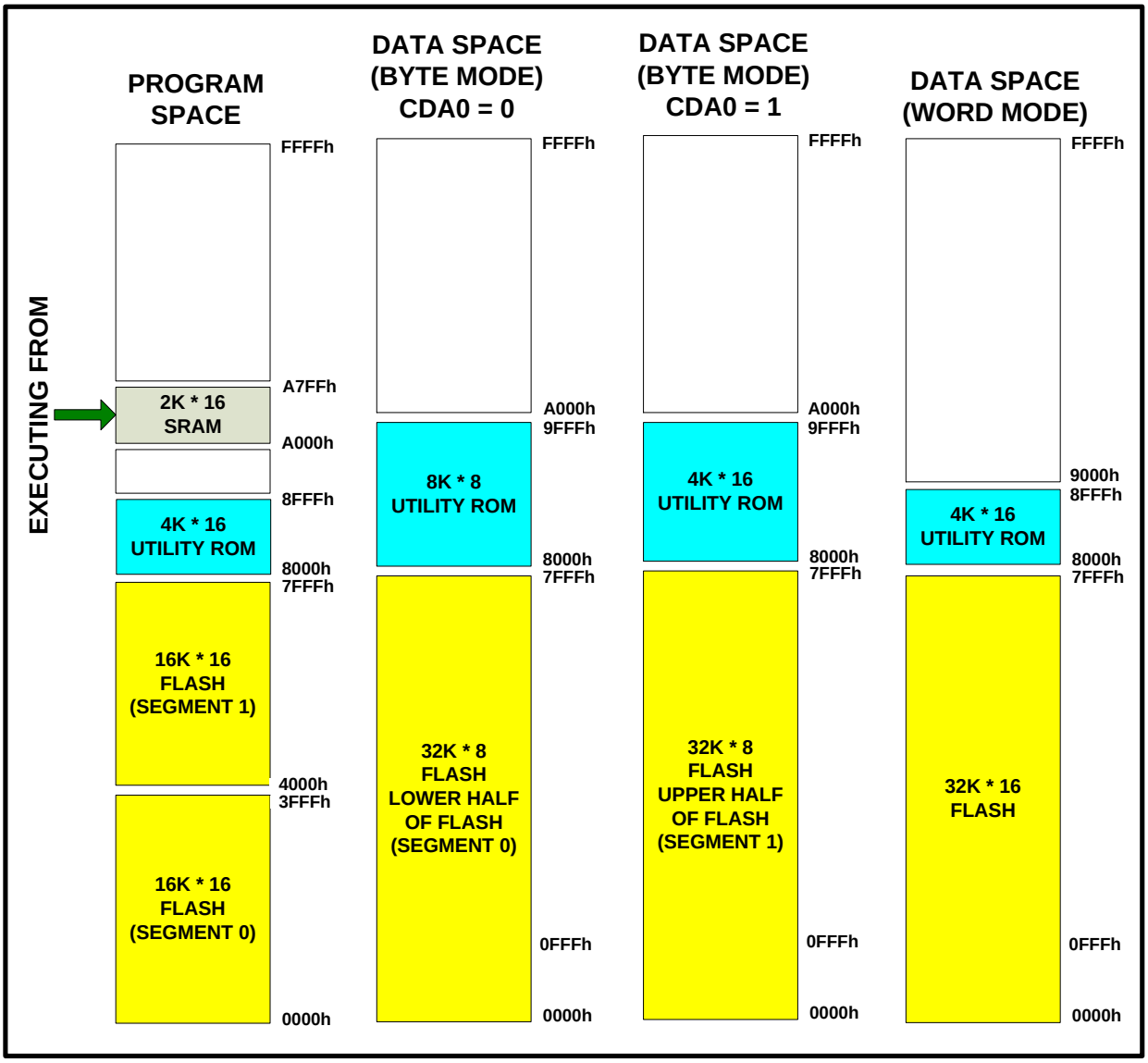


Figure 2-5: Memory Map When Executing from SRAM

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 DS4835/36 has several possible sources of reset.

- Power-On/Brownout Reset
- Watchdog Timer Reset
- External Reset
- Internal System Reset (from I2C bootloader or when ROD bit is set)
- Soft Reset (UROM command)

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, 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 RST pin is an input only.

2.6.1 – Power-On/Brownout Reset

The DS4835/36 provides a power-on reset (POR) circuit to ensure proper initialization of internal device states and analog circuits. When V_{DD} is greater than the V_{BO} level, the default state of all the DS4835/36 pins is high-impedance. When VDD is less than the V_{BO} level, the state of the DS4835/36 pins is undetermined.

The DS4835/36 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 DS4835/36 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 DS4835/36 exits the reset condition and the internal oscillator starts up. After approximately 3.2ms the DS4835/36 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 DS4835/36 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: DS4835/36 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.

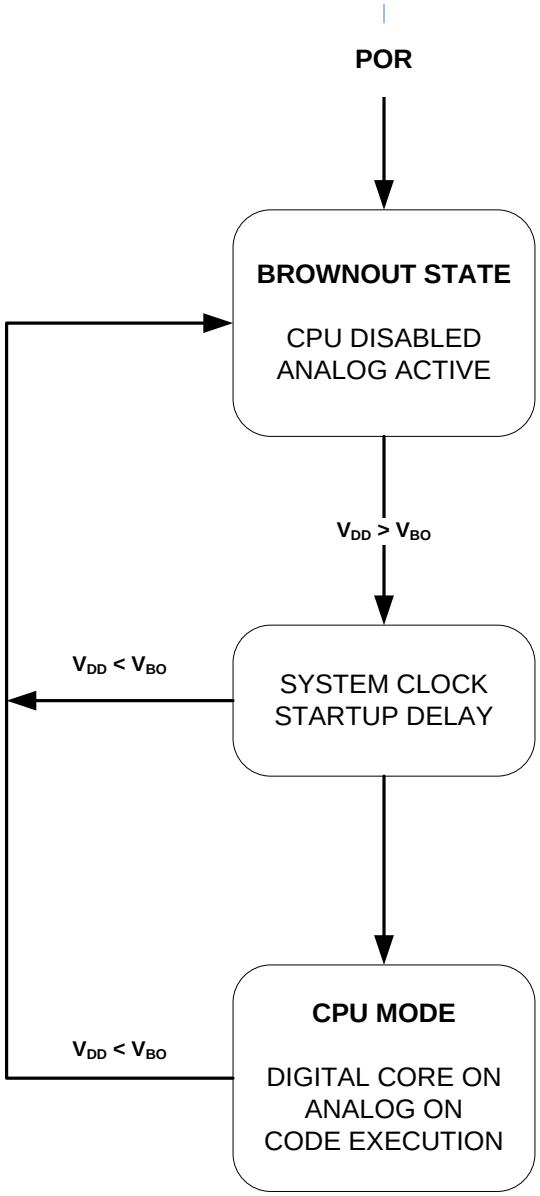


Figure 2-6: DS4835/36 State Diagram

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 DS4835/36 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 DS4835/36 is placed into external reset when the RST pin is held low. The required duration of the logic 0 pulse depends upon if the 1-Wire Test Access Port (TAP) is enabled. If the TAP is enabled, the RST pin must be held low for approximately 150us to initiate a reset. If the TAP is disabled, RST pin must be held low for 4 system clock cycles to initiate a reset. Once the DS4835/36 enters reset mode, it remains in reset if the RST pin is held at logic 0. After the RST pin returns to logic 1, the processor exits reset within 12 clock cycles.

An external reset pulse on the RST pin will reset the DS4835/36 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 DS4835/36 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 88C5h has code when executed generates internal reset. Application program can jump to this location to generate software reset.

asm ("LJUMP #88C5h")

2.7 – Clock Generation

The DS4835/36 generates a 128MHz for DC-DC, PWM, and Delta Sigma DAC operation. A 16MHz system clock is generated using a divide by 8 circuit. This oscillator starts up when V_{DD} exceeds the brownout voltage level, V_{BO}. There is a delay of approximately 3.2ms in the oscillator start up and beginning of clock. This delay ensures that the clock is stable prior to beginning normal operation.

SECTION 3 – PERIPHERAL REGISTER MAP

The following table is register map showing the location of all the peripheral registers. This map shows the module and register number for each of these registers. Detailed information for each register can be found in the description of the peripheral block associated with the register.

Reg	M0	M1	M2	M3	M4	M5	M6
0	PO2		I2CBUF_S	MCNT	ADCN	DAD2W1	
1	PO1	TECD1	I2CST_S	MA	ADST	DAD2W2	PICNT
2	PO0	TECD2	MPNTR	MB	ADDATA	IDCD1	PIDATA
3	EIF2	PO6	I2CTXFST	MC2		IDCD2	DPDATA
4	EIF1	CRCIN	I2CTXFIE	MC1	HTI	SPIB	APDDAT1
5	EIF0		I2CRXFST	MC0		IDCD3	APDDAT2
6	TBV1	EIF6	I2CRXFIE	TBCN3	LTI	IDCD4	APDDAT3
7	TBCN1	EIE6	I2CST2_S	SHFT		DADDR	APDDAT4
8	PI2	PI6	RPNTR	MC1R	TEMPCN		DCDC_SEL
9	PI1	SVM	I2CCN_S	MC0R	SH0_DATA	MTSPEC1	DCCN
10	PI0	FCNTL	I2CCN2_S*	TBC3		CNT2W1	VRDCDC
11	TBC1	FDATA	GP_REG++	TBV3		CNT2W1	IZDAC
12	TBV2		I2CSLA_S	APCDAT2	SHCN	MTSPEC2	PMOS_ON
13	EIE2	TECCN1	I2CSLA2_S	APCDAT3	SH1_DATA		NMOS_ON
14	EIE1	TECCN2	I2CSLA3_S	MACSEL	ITEMP_DATA	TWR	PMOS_ON_ST
15	EIE0		I2CSLA4_S	APCDAT1			NMOS_ON_ST
16	PD2	EIES6	I2CIE2_S	APCCN1		SLA2W1	
17	PD1	PD6	MADDR	APCIDX1		SLA2W2	
18	PD0		MADDR2	APCDAT4		IDCN1	DCDC_OPT
19	EIES2		MADDR3		SPB	IDCN2	
20	EIES1		MADDR4	APCCN2	DEV_NUM	SPICK	APDCN1
21	EIES0	CRCOUT	CUR_SLA	APCIDX2		SPICN	APDIDX1
22	PLACNT1	GP_REG	I2CIE_S	APCCN3	IEN0	SPICF	
23	PLADAT1		BRKPNT	APCIDX3	IEN1	IDCN3	APDCN2
24	PLACNT2		ICDT0	APCCN4	IEN2	IDCN4	APDIDX2
25	PLADAT2		ICDT1	APCIDX4	IEN6		
26	PLACNT3		ICDC		INRSH	MIS1	APDCN3
27	PLADAT3		ICDF			MIS2	APDIDX3
28	PLACNT4		ICDB			CCK2W1	
29	PLADAT4		ICDA			CCK2W2	APDCN4
30	TBCN2	NORMV	ICDD	APCOPT	ETEMP_DATA		APDIDX4
31	TBC2				IV2*		DPCN

Note: Register bits in Module 6 cannot be individually written, instead the entire register must be written.

*These registers are only present on the DS4836
++These registers are only present on the DS4835.

SECTION 4 – GENERAL PURPOSE INPUT/OUTPUT (GPIO) PINS

4.1 – Overview

The DS4835/36 provides general-purpose input/output (GPIO) functionality on 23 pins and GPI functionality on 4 more pins. In addition to the GPIO functionality, each of these pins is also multiplexed with at least one other function, which is classified as a "Special Function."

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

Except for a few pins which are described later in further detail, the GPIO pins either have CMOS push-pull outputs or open-drain outputs. Figure 4-1 shows a block diagram of the CMOS push-pull output pins and Figure 4-2 shows a diagram for the open drain pins.

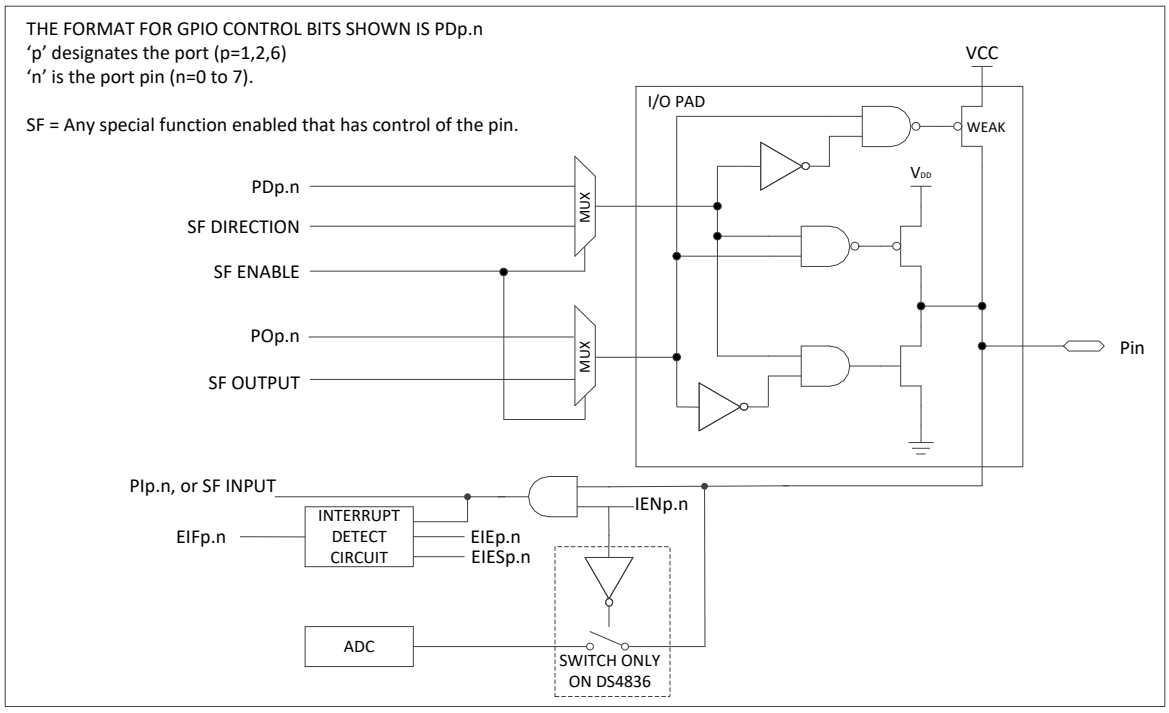


Figure 4-1 GPIO Pin Block Diagram

Some of the features of these CMOS push-pull GPIO pins are:

- CMOS output drivers
- Schmitt trigger inputs
- Optional weak pullup to VCC when operating in input mode

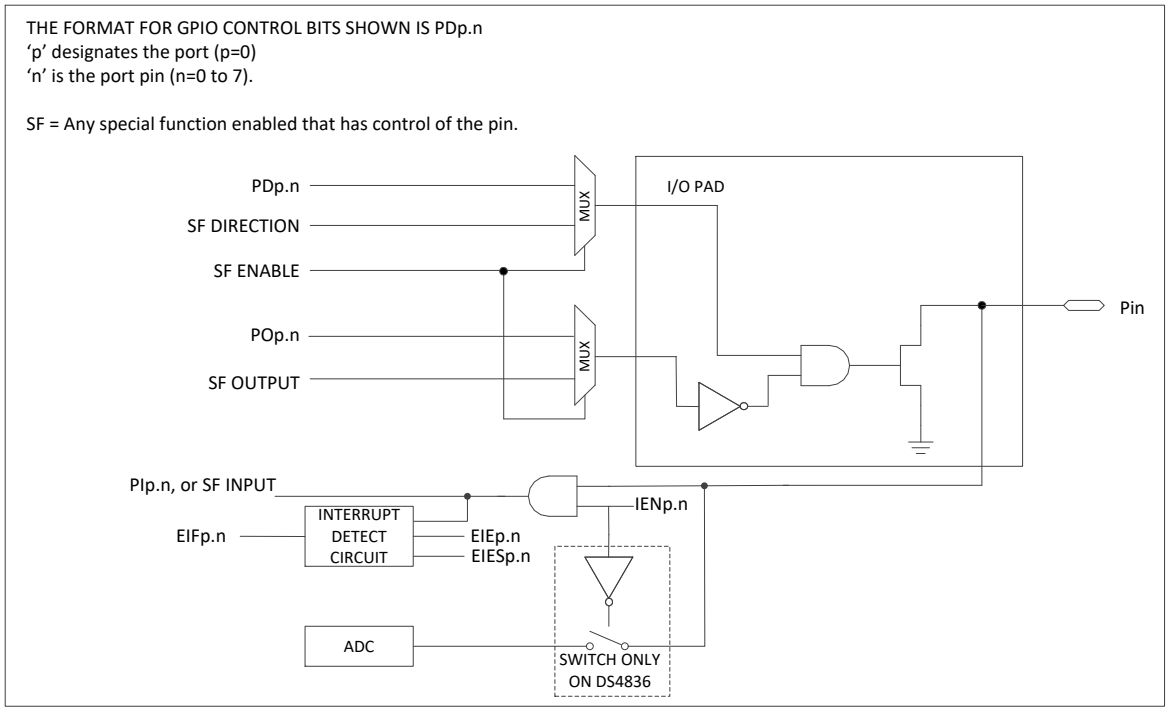


Figure 4-2 GPIO Pin Open-Drain Block Diagram

Some of the features of these open-drain output GPIO pins are:

- True open-drain output structure
- Schmitt trigger inputs

Table 4-1 details all the DS4835/36 pins as well as what functions are multiplexed with each pin. The column labeled "Default" shows the pin's default function if none of the special functions have been enabled.

Notes:

- At POR, all GPIO pins default to high-impedance. The pin input functionality will be disabled until the pin's corresponding IEN bit is set.
- The DS4836 has an additional switch added between the pin and ADC. The corresponding IEN bit must be set to 0 to enable the connection from pin to ADC.

Table 4-1. GPIO Pins and Multiplexed Functions

Name	Pin	Type	Default	ADC	IDAC	DS DAC	PWM	DCDC	PLA	S/H	I2C	3 Wire	SSPI	MSPI	Port
GP00	**40**	Open Drain	GPIO, Hi-Z	ADC0					PLAOUT	SHEN1					GPIO00
GP01	**37**	Open Drain	GPIO, Hi-Z	ADC1					PLAOUT	SHEN1					GPIO01
GP02	**36**	Open Drain	GPIO, Hi-Z	ADC2					PLAOUT	SHEN1	MSDA2				GPIO02
GP03	**11**	Open Drain	GPIO, Hi-Z	ADC3					PLAOUT	SHEN1	MSCL2				GPIO03
GP05	**12**	Open Drain	GPIO, Hi-Z	ADC5					PLAIN	SHEN0					GPIO05
GP06	**13**	Open Drain	GPIO, Hi-Z	ADC6					PLAIN						GPIO06
GP07	**14**	Open Drain	GPIO, Hi-Z	ADC7				FB3	PLAIN						GPIO07
GP10	**32**	Push-Pull	GPIO, Hi-Z								MSDA1	SDA_3W	SSPI_MOSI	MSPI_MOSI	GPIO10
GP11	**31**	Push-Pull	GPIO, Hi-Z								MSCL1	SCL_3W	SSPI_CK	MSPI_CK	GPIO11
GP12	**30**	Push-Pull	GPIO, Hi-Z	ADC10				FB2			MSDA1 ALT	CS_3W	SSPI_CS		GPIO12
GP13	**7**	Push-Pull	GPIO, Hi-Z	ADC11	IDAC1	DS_DAC8					MSDA1 ALT		SSPI_ MISO	MSPI_MISO	GPIO13
GP14	**6**	Push-Pull	GPIO, Hi-Z	ADC12	IDAC2	DS_DAC9					MSDA1 ALT				GPIO14
GP15	**4**	Push-Pull	GPIO, Hi-Z	ADC13	IDAC3	DS_DAC10				SHEN0					GPIO15
GP16	**3**	Push-Pull	GPIO, Hi-Z	ADC14	IDAC4	DS_DAC11				SHEN0					GPIO16
GP17	**38**	Push-Pull	GPIO, Hi-Z	ADC15, TSENS				APD3	PLAIN	SHEN0					GPIO17
GP20	**1**	Push-Pull	GPIO, Hi-Z	ADC16		DS_DAC0	PWM0	APD4	PLAOUT	SHP0					GPIO20
GP21	**2**	Push-Pull	GPIO, Hi-Z	ADC17		DS_DAC1	PWM1		PLAOUT	SHN0					GPIO21
GP22	**8**	Push-Pull	GPIO, Hi-Z	ADC18		DS_DAC2	PWM2	APD1	PLAOUT						GPIO22
GP23	**9**	Push-Pull	GPIO, Hi-Z	ADC19		DS_DAC3	PWM3	APD2	PLAOUT						GPIO23
GP24	**10**	Push-Pull	GPIO, Hi-Z	ADC20		DS_DAC4	PWM4		PLAIN						GPIO24
GP25	**19**	Push-Pull	GPIO, Hi-Z	ADC21		DS_DAC5	PWM5		PLAIN						GPIO25
GP26	**20**	Push-Pull	GPIO, Hi-Z	ADC22		DS_DAC6	PWM6		PLAIN	SHP1					GPIO26
GP27	**21**	Push-Pull	GPIO, Hi-Z	ADC23		DS_DAC7	PWM7		PLAIN	SHN1					GPIO27
LX1	**29**	DCDC Output	LX1					TECP1							GPI60
LX2	**27**	DCDC Output	LX2	ADC8				TECN1							GPI61
LX3	**24**	DCDC Output	LX3					TECP2							GPI62
LX4	**22**	DCDC Output	LX4	ADC9				TECN2							GPI63

Notes:

- LX1, LX2, LX3 and LX4 only have GPI capabilities
- There is a diode connected internally between each IDAC output pin and VCCDAC. Please refer to the Current DAC section for more details. If these pins are not used for IDAC functionality and VCCDAC is less then VCCO, Maxim recommends the following.
    - They not be used as GPIO. Depending upon the level of VCCDAC, the pins not be able to reach VCC as either an input or output.
    - These pins should be used as ADC pins that have inputs that will not exceed VCCDAC.

4.2 – GPIO Port Register Descriptions

The DS4835/36 has 4 ports, P0, P1, P2 and P6. The GPIO operation is controlled and monitored through the PDp, POp and PIp and IENp registers, where p = 0, 1, 2 and 6. These ports are multiplexed with the various special functions such as ADC, DAC, PLA, DC-DC, I^2C, 3-Wire, SPI etc. Additionally, these ports also provide GPIO interrupts on all the pins. A GPIO interrupt can be generated when the pin is being operated as a GPIO, or a special function. Three additional registers, EIFp, EIEp, and EIESp are used to control the GPIO interrupts.

The GPIO registers and their module addresses are listed in Table 4-2. The user should not write to any reserved bits as this may cause undesired behavior.

Register	Function	Port 0	Port 1	Port 2	Port 6
POp	Port Output Register	M0[02h]	M0[01h]	M0[00h]	N/A
PIp	Port Input Register	M0[0Ah]	M0[09h]	M0[08h]	M1[08h]
PDp	Port Direction Register	M0[12h]	M0[11h]	M0[10h]	N/A
IENp	Input Enable Register	M4[16h]	M4[17h]	M4[18h]	M4[19h]
EIFp	External Interrupt Flag Register	M0[05h]	M0[04h]	M0[03h]	M1[06h]
EIEp	External Interrupt Enable Register	M0[0Fh]	M0[0Eh]	M0[0Dh]	M1[07h]
EIESp	External Interrupt Edge Select Register	M0[15h]	M0[14h]	M0[13h]	M1[10h]

Table 4-2. GPIO Registers

GPIO Port Direction Registers (PD0, PD1, PD2)

The PDp registers are 8-bit registers 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 = 0,1,2, n = 0 to 7) is set to '1', the pin is configured as an output and data in the POp.n bit will be output on the pin. When PDp.n is cleared to '0', the pin is an input and allows an external signal to drive the pin.

When operating as an input, the pins that are not open-drain outputs can have a weak internal pullup enabled. This P-channel pull-up transistor is controlled by the POp.n bit. If POp.n is set to '1', the corresponding weak pull-up is turned on, if the POp.n bit is cleared to '0', the weak pull-up is turned off and the pin's input is high-impedance.

Following are the default values for the PDp registers. These registers have read/write access. Note, Port 6 is GPI only and does not have a PD register.

	PDp7	PDp6	PDp5	PDp4	PDp3	PDp2	PDp1	PDp0
Port 0 (PD0)	0	0	0	N/A	0	0	0	0
Port 1 (PD1)	0	0	0	0	0	0	0	0
Port 2 (PD2)	0	0	0	0	0	0	0	0

GPIO Port Output Registers (PO0, PO1, PO2 and PO6)

The POp registers are 8-bit registers that control 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 pull-up if the pin is not open-drain only.

Following are the default values for the POp registers. These registers have read/write access. Note, Port 6 is GPI only and does not have a PO register.

	POp7	POp6	POp5	POp4	POp3	POp2	POp1	POp0
Port 0 (PO0)	0	0	0	N/A	0	0	0	0
Port 1 (PO1)	0	0	0	0	0	0	0	0
Port 2 (PO2)	0	0	0	0	0	0	0	0

GPIO Port Input Registers (PI0, PI1, PI2 and PI6)

The PIp registers are 8-bit registers which contain the data that is being 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.

The DS4835/36 GPIO input capabilities must first be enabled prior to being able to read the pin input value using the PIp registers. The GPIO input can be enabled by setting the associated bit in the Input Enable (IENp) register.

The following table shows which pins have GPIO input capabilities. "s" means the default for the bit depends upon the pin state. These registers are read only.

	PIp7	PIp6	PIp5	PIp4	PIp3	PIp2	PIp1	PIp0
Port 0 (PI0)	s	s	s	N/A	s	s	s	s
Port 1 (P16)	s	s	s	s	s	s	s	s
Port 2 (PI2)	s	s	s	s	s	s	s	s
Port 6 (PI6)	N/A	N/A	N/A	N/A	s	s	s	s

GPIO Input Enable Register (IEN0, IEN 1, IEN 2 and IEN 6)

The IENp registers are 8-bit registers which enable the GPIO input capabilities. Before any pin can have GPIO input or Special Function input capabilities, this bit must first be set for the associated pin.

The DS4836 has an additional switch added between the pin and ADC. The corresponding IEN bit must be set to 0 to enable the connection from pin to ADC.

Following are the default values for the four Input Enable registers. These registers have read/write access.

	IENp7	IENp6	IENp5	IENp4	IENp3	IENp2	IENp1	IENp0
Port 0 (PO0)	0	0	0	N/A	0	0	0	0
Port 1 (PO1)	0	0	0	0	0	0	0	0
Port 2 (PO2)	0	0	0	0	0	0	0	0
Port 6 (PO6)	N/A	N/A	N/A	N/A	0	0	0	0

GPIO Port External Interrupt Flag Register (EIF0, EIF1, EIF2, and EIF6)

The EIFp register bits are set when an edge is detected that generates an interrupt. Rising or falling edge selection is done using the corresponding bit in the EIESp register. The setting of these bits will enable an interrupt to the CPU if 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 if the bit remains set.

To use the GPIO interrupts, the port must first be enabled for input. The GPIO input can be enabled by setting the associated bit in the Input Enable (IENp) register.

Following are the default values for the four EIESp registers. These registers have read/write access.

	EIFp7	EIFp6	EIFp5	EIFp4	EIFp3	EIFp2	EIFp1	EIFp0
Port 0 (EIF0)	0	0	0	N/A	0	0	0	0
Port 1 (EIF1)	0	0	0	0	0	0	0	0
Port 2 (EIF2)	0	0	0	0	0	0	0	0
Port 6 (EIF6)	N/A	N/A	N/A	N/A	0	0	0	0

GPIO Port External Interrupt Enable Register (EIE0, EIE1, EIE2, and EIE6)

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.

Following are the default values for the four EIEp registers. These registers have read/write access.

	EIEp7	EIEp6	EIEp5	EIEp4	EIEp3	EIEp2	EIEp1	EIEp0
Port 0 (EIE0)	0	0	0	N/A	0	0	0	0
Port 1 (EIE1)	0	0	0	0	0	0	0	0
Port 2 (EIE2)	0	0	0	0	0	0	0	0
Port 6 (EIE6)	N/A	N/A	N/A	N/A	0	0	0	0

GPIO Port External Interrupt Edge Select Register (EIES0, EIES1, EIES2, and EIES6)

The EIESp registers set the interrupt edge select to trigger an interrupt on either a rising or falling edge. Setting the EISPp_n bit 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.

Following are the default values for the four EIESp registers. These registers have read/write access.

	EIESp7	EIESp6	EIESp5	EIESp4	EIESp3	EIESp2	EIESp1	EIESp0
Port 0 (EIES0)	0	0	0	N/A	0	0	0	0
Port 1 (EIES1)	0	0	0	0	0	0	0	0
Port 2 (EIES2)	0	0	0	0	0	0	0	0
Port 6 (EIES6)	N/A	N/A	N/A	N/A	0	0	0	0

4.3 – GPIO Code Example

4.3.1 – GPIO Pin as Output

```
//set pin 1.4 as a high output
PD1 |= 0x10;          //set direction PD1.4 to 1 for an output
PO1 |= 0x10;          //set the output PO1.4 high
```

4.3.2 – GPIO High Impedance Input

```
//set pin 1.4 as a high-impedance input
IEN1 |= 0x10;         //enable input
PD1 &= ~0x10;         //set direction PD1.4 to 0 for input
PO1 &= ~0x10;         //set PO1.4 low to disable weak pullup
```

4.3.3 – GPIO Weak Pullup Input

```
//enable the pin 1.4 weak pullup
IEN1 |= 0x10;         //enable input
PD1 &= ~0x10;         //set direction PD1.4 to 0 for input
PO1 |= 0x10;          //set PO1.4 high to enable weak pullup
```

4.3.4 – GPIO Open-Drain Output

```
//configure pin1.4 as port 'Open Drain'
PO1 &= ~0x10;         // set the PO1.4 to the logic '0'
PD1 |= 0x10;          // this will configure P1.4 as output and drive logic '0'
PD1 &= ~0x10;         // this will configure P1.4 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 5 – IN RUSH CONTROL

As per the MSA standards specifications, optical modules should limit the inrush current during hot-swap. The DS4835/36 device integrates a software programmable inrush control circuit, which is shown in Figure 5.1.

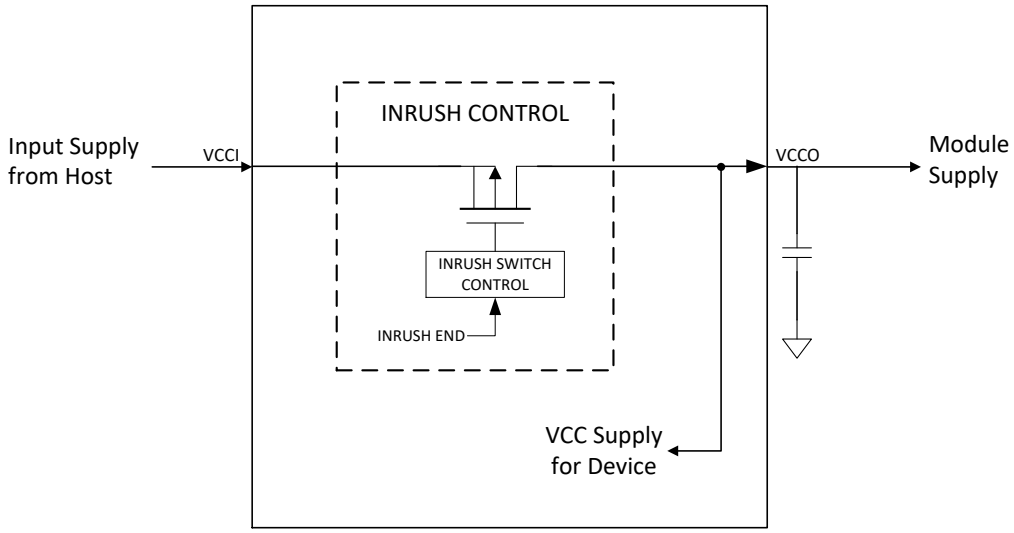


Figure 5-1: Inrush Control FET block

5.1 – Inrush Description

The DS4835/36 inrush control circuit is designed to power the entire optical module. The input to the inrush control circuit is the VCCI pin. This pin should be connected directly to the module's input power supply. The VCCI pin does not need any decoupling. The output of the inrush control circuit is the VCCO pin. This pin should be used to power the entire module. The core of the DS4835/36 is also powered by the VCCO output. A decoupling capacitor should be placed near the VCCO pin.

When VCCI is applied to the DS4835/36, the inrush FET is controlled by the DS4835/36 to limit the current to the VCCO node. Once VCCO has exceeded the DS4835/36 POR voltage, the DS4835/36 core will be start running and executing firmware. At this time, the current limited supply can be disabled and a low-impedance FET turned on between VCCI and VCCO. This is done by setting the INRUSH_END bit.

Note: If the current drawn from VCCO is greater than the inrush current control limit, then the DS4835/36 VCCO voltage will not be able to exceed V_{BO} and the device will not power up.

5.2 – Register Description

INRSH

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

BIT	NAME	DESCRIPTION
7	INRUSH_END	**Inrush End:** Setting this bit to 1 will disable the current limited current source and enable the low impedance FET between VCCI and VCCO.
6:0	RESERVED	**Reserved**. The user should write 0 to these bits.

SECTION 6 – ANALOG TO DIGITAL CONVERTER (ADC)

The DS4835/36 provides a 16-bit analog-to-digital converter (ADC) with a 32-channel analog input mux. The ADC performs differential conversions and the user can select the positive and negative ADC channels. Using this selection logic, the user can convert any external pin with respect to another pin or with respect to ground or VDD. In addition to external inputs the ADC also converts internal die temperature, external temperature, sample and hold 0 and 1 voltages, VDD, and can measure its own internal offset. This input mux, as well as the ADC controller is shown in Figure 6-1.

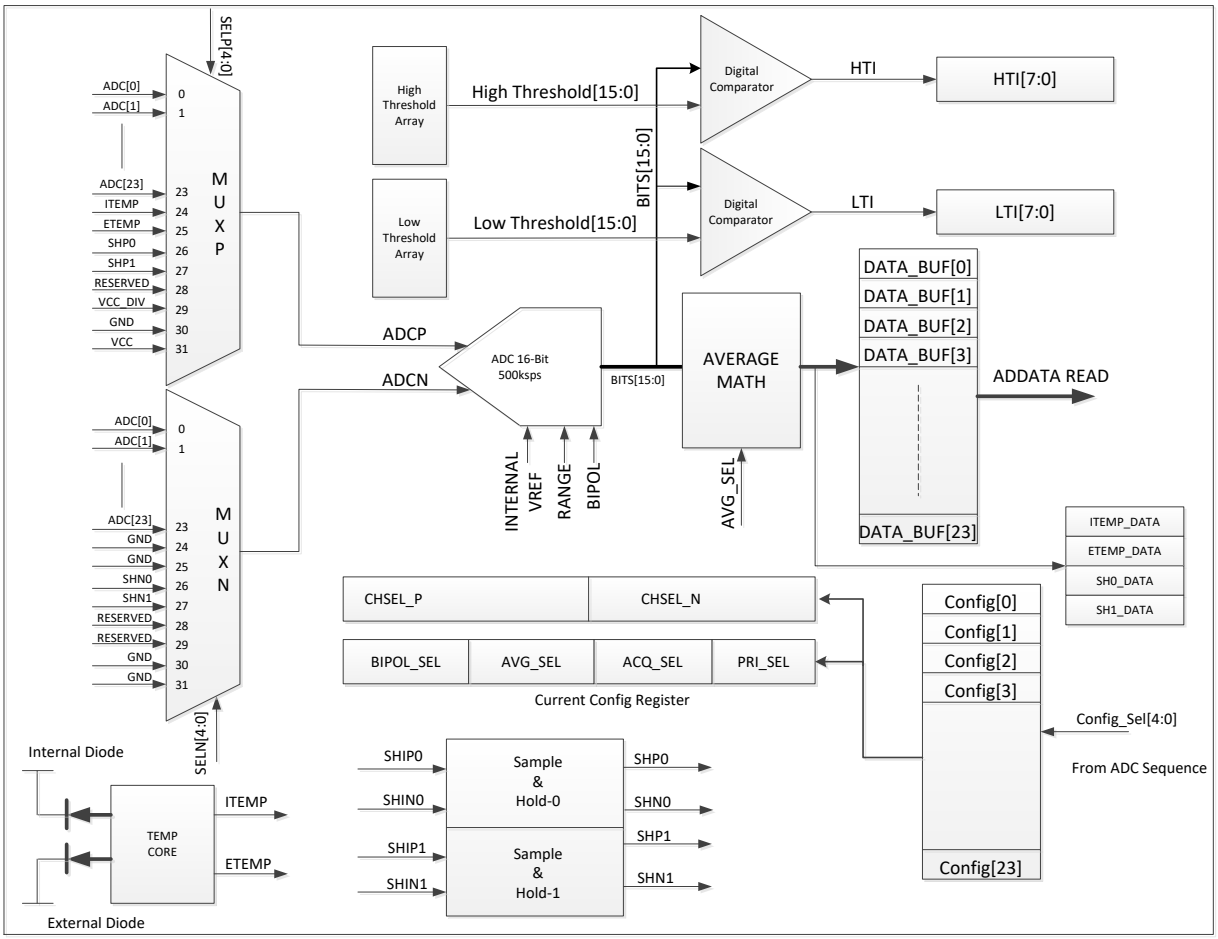


Figure 6.1: ADC Functional Block Diagram

6.1 ADC Controller

The ADC controller is the digital interface block between the CPU and the ADC, as shown in Figure 6.1. The controller is responsible for setting up and sequencing the ADC channels, averaging of channels, conversion mode, and storing converted data. The ADC controller can perform up to 24 different conversions on the external channels. Each of these conversions will have dedicated configuration and data buffer registers. In addition to these 24 conversions, ADC conversions can also be performed for internal and external temperature, and the two sample and hold channels. The ADC controller also provides various internal averaging options and extended acquisition times for individual ADC channels.

6.1.1 ADC Fullscale and Reference

The DS4835/36 ADC uses an internal reference (VREF) for its conversions.
The voltage reference for the ADC can be selected using the REF_BUF_SEL and REF_SEL bits in the ADCN register. The reference should be selected prior to starting ADC conversions. Once ADC conversions begin, the ADC reference should not be changed. The ADC voltage reference take approximately 10ms to startup.

The ADC can be configured to perform three different types of conversions. This setup is configured using the RANGE bit in the ADCN register and the BIPOL bit in the ADC_CONFIG register.

- Unipolar conversions: Will only perform a conversion with a positive result, for example if the ADC positive channel is greater than the ADC negative channel. All negative results will return 0. The full-scale range will be from 0 to VREF. This results in 1 LSB = VREF / 2^{16}
- Bipolar conversions: This mode can be used when a negative result (ADC positive channel < ADC negative channel) is expected. The ADC reports the result in 2's complement format. In this mode, the full-scale range will depend upon the RANGE bit. The BIPOL option is can be enabled individually for each ADC configuration.
    - RANGE = 0: Full scale = -VREF/2 to VREF/2. This results in 1 LSB = VREF / 2^{16}
    - RANGE = 1: Full scale = -VREF to + VREF. This results in 1 LSB = 2 * VREF / 2^{16}

6.1.2 ADC Conversion Configuration

The DS4835/36 ADC controller will perform a user defined conversion sequence for any of the 24 configurations that are enabled. The conversion sequence is setup using the ADC_CONFIG Registers. The ADC_CONFIG Registers are accessed by writing to the ADDATA register when ADCN.REGSEL = 1. Each conversion also needs to have a positive and negative channel assigned. This assignment is done using the CHSEL registers, which are accessed when ADCN.REGSEL = 2. The INDEX bits in ADCN determine which conversion's ADC_CONFIG and CHSEL register are being accessed. Table 6.1 details the available connections for both the positive and negative ADC input channels. For each conversion, the following must be configured.

- Conversion Priority. Setting this to 0 will disable the conversion
- The number of ADC conversions to average.
- The acquisition time for the conversion.
- Selection of the positive input for the ADC conversion
- Section of the negative input for the ADC conversion
- Selection of bipolar or unipolar conversion

For more information, refer to the ADC_CONFIG and CHSEL register descriptions.

CH_SEL	POSITIVE	NEGATIVE
0	ADC[0]	ADC[0]
1	ADC[1]	ADC[1]
2	ADC[2]	ADC[2]
↓	↓	↓
22	ADC[22]	ADC[22]
23	ADC[23]	ADC[23]
24	ITEMP	GND
25	ETEMP	GND
26	SHP0	SHN0
27	SHP1	SHN1
28	RESERVED	RESERVED
29	VCC_DIV	RESERVED
30	GND	GND
31	VCC	GND

Table 6.1: ADC Input Channel Selection

Notes:

- ITEMP, ETEMP, SHP0, SHP1, SHN0, SHN1 should not be chosen for an ADC configuration. These will be automatically selected by the ADC for temperature and sample and hold conversions.
- VCC_DIV is a divided version of VCCO and is intended for measuring the VCC voltage.
- VCC is the undivided VCCO voltage and is intended to be for differential conversions of channels that are referenced to VCCO.
- The DS4836 has an additional switch added between the pin and ADC. The corresponding IEN bit must be set to 0 to enable the connection from pin to ADC. Refer to the GPIO section for more details.

6.1.3 ADC Conversion Priority

The ADC controller allows priority to be given to each conversion. This allows critical channels to be sampled more often than less critical channels. There are three levels of priority, referred to as Priority Level 1 – 3, with Priority Level 1 being the highest priority. Following are the rules that the controller follows when determining the sequence of conversion. This is also illustrated in Figure 6.2.

1. All priority level 1 channels are converted once before any priority level 2 channels get a conversion slot.
2. After a conversion of a priority level 2 channel, all priority level 1 channels will be converted again.
3. All priority level 2 channels will be converted once before any priority level 3 channels get a conversion slot.
4. After a conversion of a priority level 3 channel, all priority 1 and 2 channels will be converted again according to rules 1 and 2 before the next priority level 3 channel will be configured.
5. For all channels of given priority level, the conversions will be done in numerical order.

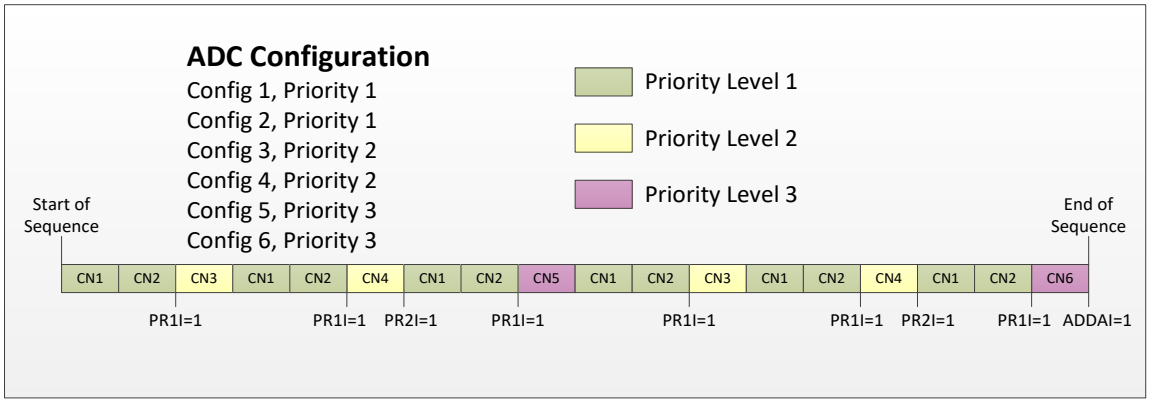


Figure 6.2: ADC Conversion Priority

6.1.4 ADC Interrupts

The ADC has three possible flags that will set when portions of an ADC sequence are complete. Each of these flags can also generate an interrupt if enabled. These flags are:

- PR1I: Will be set after all priority level 1 conversions have completed. This allows firmware to read and service the critical priority 1 conversions as soon as they have completed.
- PR2I: Will be set after all priority level 2 conversions have completed. This allows firmware to read and service the priority 2 conversion when they have completed.
- ADDAI: This flag will be set upon the full completion of an ADC sequence.

Figure 6.2 and Figure 6.3 illustrate when these ADC flags will be set during a conversion sequence.

The ADC controller also has separate interrupt flags that will set upon the completion of an internal temperature conversion (ITEMPI), external temperature conversion (ETEMPI) or sample and hold conversion (SH0I and SH1I).

6.1.5 ADC Averaging

The ADC controller provides options to average the ADC results of individual conversions. The device provides options to average 2, 4, 8, 16, 32, 64, and 128 results for each configuration. This is configured in each configuration's ADC_CONFIG register using the AVG_SEL[2:0] bits.

When averaging is enabled, the ADC interrupts will be set when averaging is complete for the last ADC configuration of a given priority. While using different averaging settings for configurations of the same priority, keep the configurations with the highest averaging required as the last channel in priority list. This will make sure that at the end of sequence interrupt all the data buffers will be ready for read. Figure 6.3 shows an example of an ADC sequence with averaging.

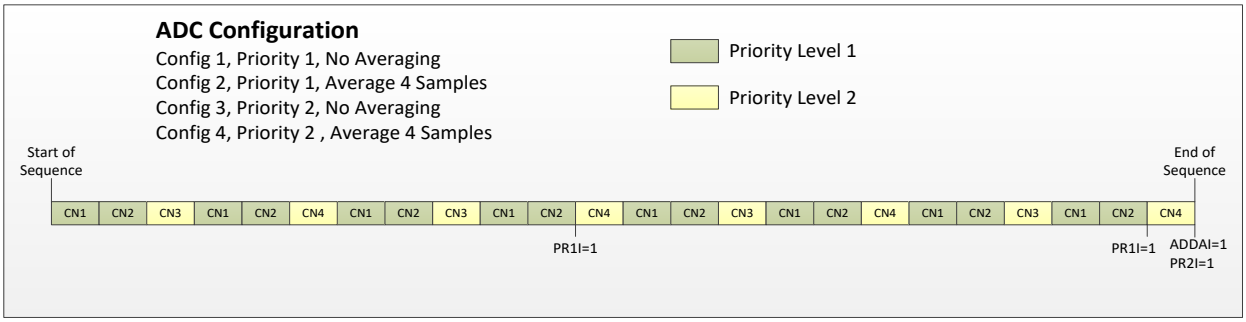


Figure 6.3: ADC Sequence with Averaging

As can be seen in Figure 6.3, the PR1I bit will be set after Config 2 completes its fourth conversion. In this time Config 1 has completed four conversions, but the flag is only set after the second conversion.

6.1.6 Starting and Stopping the ADC

The ADC conversion sequence starts when ADCN.ADCEN = 1. The ADC controller will always start converting the highest priority channel. At the end of an ADC sequence, the ADC controller will automatically start the ADC sequence over again. To disable the ADC, the ADCN.ADCEN bit must be set to 0. The ADC converter will stop at the end of the current conversion. The ADC must be stopped to make any changes to the ADC_CONFIG registers.

6.1.7 ADC Sample and Conversion Time

Each ADC conversion of a voltage takes approximately 2.0 us to complete.

When doing a temperature conversion, the ADC controller takes 128 us to force currents and integrate a temperature channel prior to a conversion. During this time, the ADC will continue processing other channels without any delays. After this 128 us time the temperature channel is ready to be converted. This conversion will take 2.0 us, the same as a voltage conversion channel. The ADC controller needs to capture two complete temperature conversions (integrate and convert) to calculate the temperature. This means that each temperature conversion takes 256us prior to averaging. The internal and external temperature conversions use the same temperature core which means these will not be converted simultaneously.

6.1.8 ADC Sample Time and Extended Acquisition

The DS4835/36 ADC utilizes a dual sample-and-hold architecture. Note that this is referring to the ADC architecture, not the S/H peripheral functionality that the DS4835/36 provides. The dual sample-and-hold architecture means that as one channel is being converted, the next channel is being sampled. By default, the next channel to be converted will be sampled during the last 180ns of the ongoing conversion.

The ADC sample caps must be given time to charge prior to a conversion being started. For 16-bits of resolution, this requires a settling time of greater than 11 tau (τ), where τ is the time constant of the ADC sample caps, which are approximately 6pF, and the source resistance. To calculate the maximum source impedance for the sample time, the following equation can be used. This will assume settling to 12 τ.

$$R = \frac{sample\ time}{12 \times 6pF}$$

So with the default 180ns sample time, and assuming settling to 12 τ , the maximum source resistance for an ADC channel is 2.5 kΩ.

Some ADC channels may have an impedance greater than the 2.5 kΩ supported by the default sample time. For these channels, the ADC controller has an option to extend the acquisition time. An extra 0.5us, 1.0us or 2.0us can be added to default 180ns sample time. This extended acquisition can be enabled individually for each voltage channel using the ACQ_SEL bits in the ADC_CONFIG registers. The temperature and sample and hold conversions can also select an extended acquisition using the TEMP_ACQ bits in the TEMPCN register. When using extended acquisition, the formula above can be used to calculate the maximum source resistance for each acquisition time.

Figure 6.4 shows the impact of using extended acquisition on the total round robin time for a sequence. ADC conversions will always take 2.0 us. But, if extended acquisition is enabled for a channel, this will delay when the ADC conversion is started for the given channel.

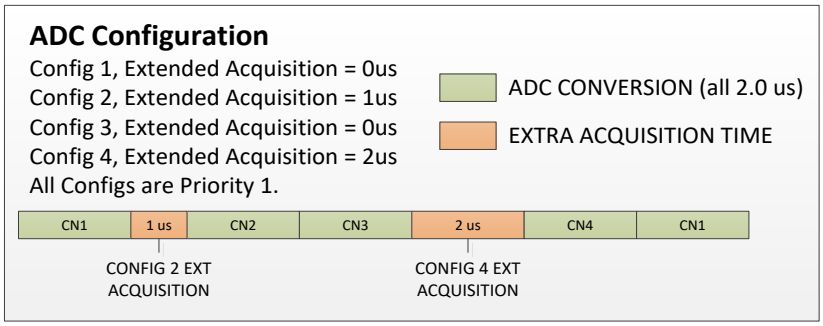


Figure 6.4: Timing Impact of Extended Acquisition

Some high impedance ADC sources may experience glitches as the ADC sample cap is connected to the ADC source. If this causes problems with the system, the DS4835/36 has an internal buffer than can be enabled between the ADC input and the ADC sample caps. This buffer is designed to charge the ADC input caps without causing glitches on the ADC source. This buffer is enabled by setting the MUXBUF_EN bit in the ADCN register. When enabled, this buffer will be used for all ADC conversions that have extended acquisition enabled. This buffer will also be used for temperature and sample and hold conversions if these conversions have selected to use extended acquisition.

6.1.9 Temperature Conversions

The DS4835/36 can sense the internal die temperature and the temperature of an external diode. The DS4835/36 ADC controller forces current into the internal diode and integrates voltage across diode. After integration, the voltage is measured by the ADC with the voltage then being converted into a temperature. During the time the controller is forcing current and integrating, the ADC controller will be running the ADC sequence as normal. When a temperature channel is ready for an ADC conversion, the conversion occur after the current conversion has completed. Figure 6.5 shows how a temperature conversion will be added to the ADC sequence. Unipolar conversions are automatically performed for temperature channels.

All temperature conversions are automatically averaged by the ADC controller. The number of averages can be changed with the TEMP_AVG bits in the TEMPCN register.

Temperature conversions are started by setting the ITEMP or ETEMP bits in the TEMPCN register. The temperature conversion will continue until the ITEMP or ETEMP bits are cleared. The ADC will set the ITMPI or ETEMPI bits, which can generate an interrupt if enabled, when a new temperature value is ready. The ITEMPI and ETEMPI flags will be set at the completion of the last temperature conversion in an average. The temperature results will be available in the ITEMP_DATA and ETEMP_DATA registers.

A moving average filter of 16 conversions can also be enabled for temperature conversions. When the moving average is enabled, a new temperature result will be reported at the completion when the last conversion of a temperature average is completed. The reported result will be the result from the moving average filter. When

initialized, the 16 samples in the moving average filter are initialized to 0. So, it will require 16 complete temperature conversions before the output from the moving average will be correct.

The sample acquisition time for a temperature conversion can be selected using the TEMP_ACQ bits in the TEMPCN register. If extended acquisition is selected with these bits, this acquisition time will apply to both internal and external temperature conversions, as well as sample and hold conversions. Also, if extended acquisition is enabled and the internal ADC buffer is enabled with the MUXBUF_EN bit, this buffer will be applied to the temperature and sample and hold channels.

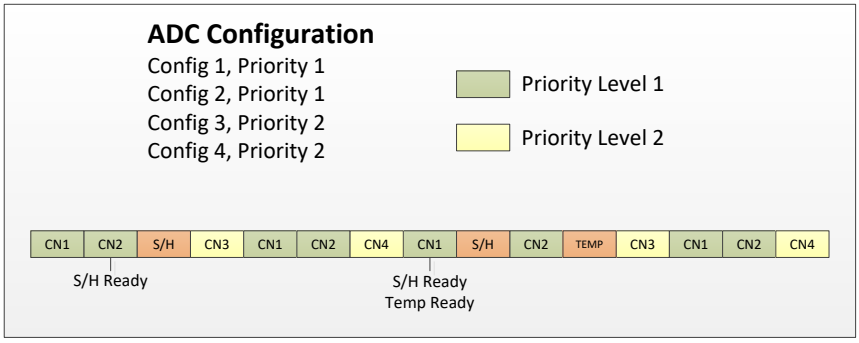


Figure 6.5: Temperature and S/H Conversion Inserting into ADC Sequence

6.1.10 Sample and Hold Conversions

The DS4835/36 has two sample-and-hold (S/H) channels that are also converted by the ADC. For detailed information on the S/H functionality, refer to the Sample and Hold section of this user's guide.

When a S/H channel is ready for an ADC conversion, the conversion will occur after the current conversion has completed. If a temperature conversion also happens to be ready for a conversion, the S/H channel will be converted first, followed by a normal voltage channel, then the temperature channel. Figure 6.5 shows how a S/H conversion will be added to the ADC sequence.

6.1.11 Applying Scale and Offset to ADC Results

6.1.11.1 Voltage Conversions

To obtain more accurate results when doing voltage conversions, the ADC can sample and convert its internal offset. This can be accomplished by setting both positive and negative ADC inputs to be a GND channel. There is not a scale factor for voltage conversions. The formula to calculate the result of a voltage conversion is given below.

Voltage Result = raw adc value – measure adc offset

ADC voltage channels that are using a bipolar conversion can have an automatic offset cancelation applied if the VOFF_CANCEL bit in ADST is set. If this is enabled, the ADC will subtract the measured offset from all bipolar ADC results. The ADC offset must be configured to be measured with ADC configuration 22 and should be setup as an internal ground signal for both the positive and negative input channels.

6.1.11.2 Internal Temperature Conversions

Scale and offset for internal temperature is calibrated at the factory and stored into info memory. These can be read from info memory using the UROM_infoRead command, refer to Utility ROM Section for more information. The scale factor is weighted so that after multiplying by the scale factor, the results must be divided by 2^{12} to get the correct value. To apply the scale and offset values, use the following equation.

$$Temperature\ Result = \frac{itemp\ value\ x\ temp\ scale}{2^{12}} - temp\ offset$$

The scale and offset factors set by the factory are for VREF = 2.4V. When VREF is set to something other than 2.4V and internal temperature conversions are being used, the user must perform their own calibration using the procedure documented below for external temperature conversions.

6.1.11.3 External Temperature Conversions

External temperature conversions must be calibrated in an application because all temperature sensing diodes are different. To do this calibration, the following steps can be used.

1) Do an external temperature conversion with the external diode at 25 °C.
2) The ADC will report a value in ETEMP_DATA : Call it etemp_val_1
3) Equation(i): (scale x etemp_val_1) – offset = 3200 -> (25 * 128 lsb per °C)
4) Do another external temperature conversion with the external diode at 85 °C.
5) The ADC will report a value in ETEMP_DATA : Call it etemp_val_2
6) Equation(ii): (scale x etemp_val_2) – offset = 10880 -> (85 * 128 lsb per °C)
7) Use equation (i) & (ii) to solve for external temperature scale and offset.

6.1.11 VCC Conversions

The DS4835/36 ADC can sample the VCC supply. The supply that is sampled is the voltage at the VCCO pin. The ADC controller has an internal resistor divider with a ratio of ¼ to scale the internal VCCO voltage to a level lower than the ADC full scale voltage.

6.1.12 Special Uses of the ADC

The results of the DS4835/36's ADC converter can be used for other special functions that the DS4835/36 provides. These special functions are quick trip comparisons, automatic power control (APC), external DCDC boost control (APD), and thermistor feedback for TEC control.

Each of these special functions, if it is enabled, will automatically take the corresponding raw data at the end of an ADC conversion. This data is moved to the special function by hardware, no firmware intervention is required. To use these special functions, the corresponding ADC conversion must be mapped to a particular configuration. Table 6.2 shows this mapping.

For these special functions, the output from the ADC will be used. If ADC offset cancelation is enabled, the value passed to the special function will be the value with offset removed. Note, ADC scale will not be applied to the value passed to the special function. The value passed to the special function will also be the averaged value, if averaging is enabled.

Configuration Register	Special Function
[7:0]	Quick Trip [7:0]
8	APC 1
9	APC 2
10	APC 3
11	APC 4
12	APD 1 FB
13	APD 2 FB
14	APD 3 FB
15	APD 4 FB and Digital Buck DCDC4 FB
16	TEC 1 Thermistor
17	TEC 2 Thermistor
18	DCDC1 PI Loop Feedback
19	DCDC2 PI Loop Feedback
20	DCDC3 PI Loop Feedback
22	ADC Internal Offset for offset cancelation
23	VCC conversion for VCC compensation of DS DAC and TEC

Table 6.2: Special ADC Configuration Assignment

6.1.13 Quick Trips

The DS4835/36 will perform a quick trip comparison to user defined thresholds at the completion of every ADC voltage conversion for configurations 0 through 7. This quick trip comparison will be done automatically and does not need to be enabled. The quick trip comparison is done using the raw ADC values, which do not have any scale or offset applied.

At the completion of an ADC conversion, the ADC results are compared to the corresponding high and low threshold registers.

- If the ADC result is greater than the value in the high threshold register, the corresponding flag in the HTI register will be set. This can generate an interrupt if enable.
- If the ADC result is less than the value in the low threshold register, the corresponding flag in the LTI register will be set. This can generate an interrupt if enable.

The quick trip threshold registers are 16-bit registers. A quick trip comparison can be masked by setting the threshold to the appropriate rail. This would be setting the high threshold to 0xFFFF and the lower threshold to 0x0000. Because of ADC noise, the user may elect to not use some of the lower LSBs of the QT threshold registers.

Quick Trips are only available for the 8 conversions that are assigned to configuration 0 through 7. Quick trips cannot be used for temperature or S/H channels.

The QT comparators can also operate in a window comparator mode. In this mode, the low and high threshold trips will reset each other. For example, if the low threshold flag is currently set and the input level then exceeds the high threshold, the high threshold flag is set and the low threshold flag will be automatically reset. This feature reduces the firmware processing of signals that have a hysteresis applied. An example of this would be RSSI based LOS. Window mode comparisons are enabled by setting the corresponding bit in the QT Window Mode Register.

6.2 ADC Register Descriptions

The ADC is controlled by ADC SFR registers. ADCN, ADDATA registers are used for setup, control, and reading data from the ADC.

ADC Control Register (ADCN)

Bit	15	14	13:12	11	10	9	8	7:5	4:0
Name	ADC_EN	RES	REF_SEL	MUXBUFEN	REFBUF_SEL	RES	RANGE	REG_SEL[2:0]	INDEX[4:0]
Reset	0	0	10	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15	ADC_EN	ADC Enable bit.
14	RESERVED	Reserved. This bit should be set to 0.
13:12	REF_SEL[1:0]	Selects the programmable reference for VREF. Only valid when REF_BUF_SEL = 0. REF_SEL[1:0] / VREF 00 / 1.2V 01 / 1.75V 10 / 2.4V 11 / Not Valid Note: The 1.2V and 1.75V references are not characterized or production tested.
11	MUXBUF_EN	MUX Buffer Enable: Setting this bit will enable the input buffer on the ADC Multiplexer. This feature allows high impedance nodes to settle quickly on the ADC caps while also not disturbing the input signal. The input buffer is only active during the ADC extended acquisition period. So this feature will only be applied to channels that have extended acquisition enabled.
10	REFBUF_SEL	0: Programmable reference is selected for VREF. 1: Fixed 2V reference is selected for VREF.
9	RES	Reserved. Set to 0.
8	RANGE	When operating in Bipolar mode, this bit sets the ADC full scale range. RANGE = 0, Full scale is +/- VREF/2 RANGE = 1, Full scale is +/- VREF
7:5	REGSEL[2:0]	Register selection of ADDATA. **REGSEL[2:0]** / **ADDATA Target Register** 000 / ADC Data Buffer 001 / ADC_CONFIG Registers 010 / ADC CHSEL Registers 011 / QT Low Threshold Registers 100 / QT High Threshold registers 101 / QT Window Mode Other / Undefined
4:0	INDEX[4:0]	Index pointer for ADDATA register. Auto increments on every read/write to ADDATA.

Note: The ADDATA register is read back by the debugger. This will result in the ADCN.INDEX bits incrementing at every debug mode breakpoint when the registers are read by the debugger.

ADC Status Register (ADST)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	VOFF_CANCEL	SH1IE	SH0IE	ITEMPIE	ETEMPIE	PR2IE	PR1IE	ADDAIE	-	SH1I	SH0I	ITEMPI	ETEMPI	PR2I	PR1I	ADDAI
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	VOFF_CANCEL	**Offset Cancelation.** Setting this bit will enable internal offset cancelation for channels configured in bipolar mode. To use this feature, configuration 22 should be used for offset with both positive and negative input channels set to internal ground.
14	SH1IE	**SH1 Interrupt Enable**. Setting this bit will enable an interrupt on completion of a S/H 1 conversion.
13	SH0IE	**SH0 Interrupt Enable**. Setting this bit will enable an interrupt on completion of a S/H 0 conversion.
12	ITEMPIE	**Internal Temp Interrupt Enable**. Setting this bit will enable an interrupt on completion of an internal temperature conversion.
11	ETEMPIE	**External Temp Interrupt Enable**. Setting this bit will enable an interrupt on completion of an external temperature conversion.
10	PR2IE	**Priority 2 Interrupt Enable**. Setting this bit will enable an interrupt on completion of all priority 2 channels in a sequence.
9	PR1IE	**Priority 1 Interrupt Enable**. Setting this bit will enable an interrupt on completion of all priority 1 channels in a sequence.
8	ADDAIE	**ADC Data Available Interrupt Enable**. Setting this bit will enable an interrupt on completion of an entire ADC sequence.
7	RESERVED	Returns 0
6	SH1I	**SH1 Interrupt Flag**. This bit is set when a S/H1 conversion has completed. This bit is cleared by software writing a '0'.
5	SH0I	**SH0 Interrupt Flag**. This bit is set when a S/H0 conversion has completed. This bit is cleared by software writing a '0'.
4	ITEMPI	**Internal Temperature Data Available Interrupt Flag.** This bit is set to '1' when an internal temperature conversion is complete. This bit is cleared by software writing a '0'.
3	ETEMPI	**External Temperature Data Available Interrupt Flag.** This bit is set to '1' when an external temperature conversion is complete. This bit is cleared by software writing a '0'.
2	PR2I	**Priority 2 Interrupt Flag**. This bit is set when all priority 2 conversions have completed. This bit is cleared by software writing a '0'.
1	PR1I	**Priority 1 Interrupt Flag**. This bit is set when all priority 1 conversions have completed. This bit is cleared by software writing a '0'.
0	ADDAI	**ADC Data Available Flag**. This bit is set when the ADC has finished all channels in a sequence with averaging. This bit is cleared by software writing a '0'.

Temperature Control Register (TEMPCN)

The Temperature Control Register TEMPCN configures and enables both the internal and external temperature measurements.

Bit	15:13	12	11	10:8	7	6:4	3:2	1	0
Name	RES	MAVG_EN	RES	RES	RES	TEMP_AVG[2:0]	TEMP_ACQ[1:0]	ITEMP	ETEMP
Reset	0	0	0	111	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:13	Reserved	**Reserved**. This bit should be set to 0.
12	MAVG_EN	**Moving Average Enable:** When set to 1, enables a 16 sample moving average for temperature conversions. This moving average is in addition to any averaging selected with the TEMP_AVG bits.
11	Reserved	**Reserved**. This bit should be set to 0.
10:8	Reserved	**Reserved.** These bits should be set to 1.
7	Reserved	**Reserved**. This bit should be set to 0.
6:4	TEMP_AVG[2:0]	Temperature averaging setting. **TEMP_AVG[2:0]** / **Number of Samples Averaged** 000 / 2 001 / 4 010 / 8 011 / 16 100 / 32 101 / 64 110 / 128 111 / Invalid
3:2	TEMP_ACQ[1:0]	**Temperature Extended Acquisition.** Selects the extra acquisition time for both internal and external temperature. This extra acquisition time will also be applied to sample and hold conversions. **ACQ_SEL[1:0]** / **Extra Acquisition Time** 00 / 0 01 / 0.5us 10 / 1us 11 / 2us
1	ITEMP	**Internal Temperature Enable.** Setting this bit to enables internal temperature conversions.
0	ETEMP	**External Temperature Enable.** Setting this bit to '1' enables external temperature conversions.

ADC_CONFIG Register (ADDATA when ADCN.REGSEL = 1)

When REGSEL = 1, writing to the ADDATA register writes to one of the 24 ADC_CONFIG registers. The configuration register written to is selected by the INDEX[4:0] bits. The INDEX[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	15:8	7	6:4	3:2	1:0
Name	RESERVED	BIPOL_SEL	AVG_SEL[2:0]	ACQ_SEL[1:0]	PRI_SEL[1:0]
Reset	0	0	0	0	0
Access	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:8	Reserved	**Reserved**. The user should not write to these bits.
7	BIPOL_SEL	**Bipolar Select:** Setting this bit to a 1 will set this configuration to do a bipolar conversion instead of unipolar conversion. If this bit is set to 0, a unipolar conversion will be done for this configuration.
6:4	AVG_SEL[2:0]	**ADC Average Select:** These bits select number of ADC samples to be averaged by the ADC controller.
3:2	ACQ_SEL[1:0]	**Extended Acquisition Select:** Selects the extra acquisition time for the current configuration
1:0	PRI_SEL[1:0]	**Priority Selection:** Defines the priority of this ADC configuration.

AVG_SEL[2:0]	Samples Average
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

ACQ_SEL[1:0]	Extra Acquisition Time
00	0
01	0.5us
10	1us
11	2us

PRI_SEL[1:0]	ADDATA Target Register
00	No Conversion
01	Level 1 priority
10	Level 2 priority
11	Level 3 priority

ADC_CHSEL Register (ADDATA when ADCN.REGSEL = 2)

When REGSEL = 2, writing to the ADDATA register writes to one of the 24 configuration channel select registers. The channel select register written to is selected by the INDEX[4:0] bits. The INDEX[4:0] bits are automatically incremented after a write to ADDATA. This allows consecutive writes of ADDATA to setup consecutive configuration registers. The channel select registers are reset to '0' on all forms of reset.

Bit	15:13	12:8	7:5	4:0
Name	RESERVED	CHSEL_P	RESERVED	CHSEL_N
Reset	0	0	0	0
Access	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:13	Reserved	**Reserved**. The user should not write to these bits.
12:8	CHSEL_P	**ADC Positive Channel Select** This selects which ADC channel (0 to 31) to use as the positive input for an ADC conversion.
7:5	Reserved	**Reserved**. The user should not write to these bits.
4:0	CHSEL_N	**ADC Negative Channel Select** This selects which ADC channel (0 to 31) to use as the negative input for an ADC conversion.

ADC Data Buffer (ADDATA when REGSEL = 0)

When REGSEL = 0, reading from the ADDATA register reads the ADC results stored in one of the 24 data buffers. The INDEX[4:0] bits point to the data buffer to be read. Reading ADDATA register returns the 16-bits of ADC conversion data from the selected data buffer memory. The INDEX[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 INDEX[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 either signed or unsigned, based upon the setting of the BIPOL bit. Following are the bit weightings for the ADC data buffers.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Unipolar	2^{15}	2^{14}	2^{13}	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}
Bipolar	S	2^{14}	2^{13}	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}

QT Low Threshold Register (ADDATA when REGSEL = 3)

When REGSEL = 3, writing to the ADDATA register will set the Quick Trip Low Threshold for an ADC channel. INDEX[4:0] is used to select which configuration's register is being accessed. Note, the Quick Trip functionality is only available for ADC configurations 0 through 7. The INDEX[4:0] bits are automatically incremented after an access of ADDATA.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	QT_LOW[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

The value in this register is automatically compared to the raw ADC data. If the ADC data is less than the QT_LOW value, the corresponding QT low interrupt flag will be set in the LTI register.

QT High Threshold Register (ADDATA when REGSEL = 4)

When REGSEL = 4, writing to the ADDATA register will set the Quick Trip High Threshold for an ADC channel. INDEX[4:0] is used to select which configuration's register is being accessed. Note, the Quick Trip functionality is only available for ADC configurations 0 through 7. The INDEX[4:0] bits are automatically incremented after an access of ADDATA.

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

The value in this register is automatically compared to the raw ADC data. If the ADC data is greater than the QT_HIGH value, the corresponding QT high interrupt flag will be set in the HTI register.

QT Window Register (ADDATA when REGSEL = 5)

When REGSEL = 5, the ADDATA register will access the QT Window Register.

Bit	15:8	7	6	5	4	3	2	1	0
Name	RESERVED	QTW7	QTW6	QTW5	QTW4	QTW3	QTW2	QTW1	QTW0
Reset	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw

When one of the QTWn bits is set, this will enable the QT to operate in Window Compare Mode. In this mode, the low and high threshold trips will reset each other. For example, if the low threshold flag is currently set and the input level then exceeds the high threshold, the high threshold flag will set and the low threshold flag will be automatically reset.

Low Trip Interrupt Register (LTI)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	LTIE[7:0]								LTI[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
15:8	LTIE[7:0]	**Low Trip Interrupt Enable**. This register is used to enable/disable interrupts from the corresponding LTI[7:0] flags in this register. Setting a bit to 1 will enable the interrupt, clearing the bit will disable the interrupt.
7:0	LTI[7:0]	**Low Trip Interrupt Flag**. These bits are set when the corresponding channel is less than the QT Low Threshold level. These bits must be cleared by software. These bits correspond to their respective ADC configurations. For example, LTI[1] is for ADC Configuration 1.

High Trip Interrupt Register (HTI)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	HTIE[7:0]								HTI[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
15:8	HTIE[7:0]	**High Trip Interrupt Enable**. This register is used to enable/disable interrupts from the corresponding HTI[7:0] flags in this register. Setting a bit to 1 will enable the interrupt, clearing the bit will disable the interrupt.
7:0	HTI[7:0]	**High Trip Interrupt Flag**. These bits are set when the corresponding channel is greater than the QT High Threshold level. These bits must be cleared by software. These bits correspond to their respective ADC configurations. For example, HTI[1] is for ADC Configuration 1.

Note: The HTI register is not read back by the debugger. When viewing this register in the debugger, it will always return as 0000h.

ITEMP_DATA and ETEMP DATA

These two registers are used to readback the values of an internal or external temperature conversion. After applying the temperature scale and offset as described, the bit weighting will be as shown below.

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

6.3 ADC Example Code

```
unsigned int ADC_MONITORS[3];
signed int InternalTemp = 0;

void main(void)
{
   unsigned int i;
   //ADC Initialization for ADC Channel 1, 3 and 5 wrt GND.
   ADCN = 0x0020;                                  //Set Index = 0 for Config 0 and REG_SEL = 1 for Configuration
   ADDATA = 0x007E;                                //Priority 2, 2us extra Acquisition, 128 samples average
   ADCN = 0x0040;                                  //Set Index = 0 for Config 0 and REG_SEL = 2 for ADC Channel Configuration
   ADDATA = 0x011E;                                //Positive Channel = ADC1, Negative Channel = Ground Reference

   ADCN = 0x0021;                                  //Set Index = 1 for Config 1 and REG_SEL = 1 for Configuration
   ADDATA = 0x007E;                                //Priority 2, 2us extra Acquisition, 128 samples average
   ADCN = 0x0041;                                  //Set Index = 1 for Config 1 and REG_SEL = 2 for ADC Channel Configuration
   ADDATA = 0x031E;                                //Positive Channel = ADC3, Negative Channel = Ground Reference

   ADCN = 0x0022;                                  //Set Index = 2 for Config 2 and REG_SEL = 1 for Configuration
   ADDATA = 0x007E;                                //Priority 2, 2us extra Acquisition, 128 samples average
   ADCN = 0x0042;                                  //Set Index = 2 for Config 2 and REG_SEL = 2 for ADC Channel Configuration
   ADDATA = 0x051E;                                //Positive Channel = ADC5, Negative Channel = Ground Reference

   TEMPCN_bit.TEMP_AVG = 6;                        //128 Temp sample Average
   TEMPCN_bit.ITEMP = 1;                           //Enable temperature

   ADCN = 0x8000;    //ADC turns on the ADC

   while(1)
   {
    //ADC channels conversion
    if( ADST_bit.ADDAI )                           //Check if ADC Sequence is complete
    {
      ADST_bit.ADDAI = 0;                          //Clear ADC Interrupt flag ADDAI

      //Read data
      for( i = 0; i < 3; i++)
      {
        ADCN_bit.INDEX = i;                        //Point to corresponding ADDATA Config
        ADC_MONITORS[i] =  ADDATA;                         //Read the data
      }
    }

    //Internal Temperature
    if( ADST_bit.ITEMPI )                          //Check if Internal Temperature sensor is complete
    {
       ADST_bit.ITEMPI = 0;                        //Clear Internal temperature sensor conversion complete flag
       InternalTemp = ITEMP_DATA;                  //Read the internal temperature
    }
   }
}
```

SECTION 7 – DCDC BUCK CONVERTER

The DS4835/36 contains three synchronous buck DCDC converters, which includes the switching FETs and controller hardware. Shown in Figure 7-1 is a diagram showing one of the DS4835/36's buck DCDC converter and the required external components.

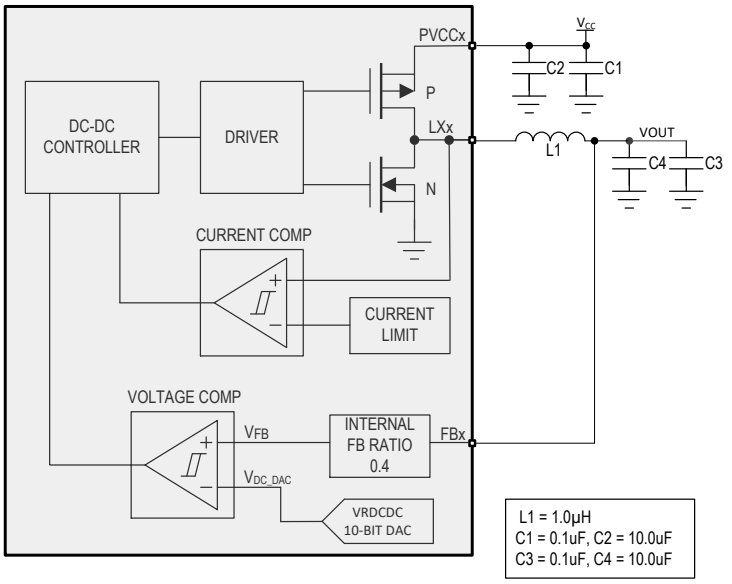


Figure 7-1: Buck DCDC Diagram

To operate the DS4835/36's buck controllers, they must be connected to the proper DS4835/36 pins. The DS4835/36 also has dedicated ADC channels and ADC configurations for the buck converter operation. Table 7-1 provides a list of the dedicated resources required for the operation of the buck controllers.

DCDC	LX	PVCC	FB	FB ADC	ADC Config for PI Loop
1	LX1	PVCC1	VCCDAC	ADC4	Config[18]
2	LX2	PVCC2	GP12	ADC10	Config[19]
3	LX3	PVCC3	GP07	ADC7	Config[20]

Table 7-1: Buck DCDC Resources

7.1 Buck Controller Operation

The DS4835/36 buck controller operates in two different modes. When first enabled, the controller is operating in startup mode. Startup mode is designed to limit the buck converter's inductor current so optical module inrush specifications are not exceeded. At the completion of the converter startup, the converter then enters normal operation.

There are four registers that are used to control the time that the PMOS and NMOS are on during each cycle. These are described in Table 7-2. The time represented by each of these settings can be calculated using the following formula. All of the DCDC timing is based off the DS4835/36's internal 128MHz clock.

Transistor On Time = Register Setting * (1/128MHz)

Register	Description
PMOS_ON	Sets the minimum time the PMOS is on during each cycle of normal operation
NMOS_ON	Sets the minimum time the NMOS is on during each cycle of normal operation
PMOS_ON_ST	Sets the time the PMOS will be on each cycle during startup operation.
NMOS_ON_ST	Sets the minimum time the NMOS is on during each cycle of startup operation

Table 7-2: Buck DCDC Timing Registers

7.1.1 Normal Operation

During the normal mode of operation, the converter operates in a continuous conduction mode of operation, which means that one of the FETs will always be conducting. Figure 7-2 shows the operation of the buck controller during normal operation.

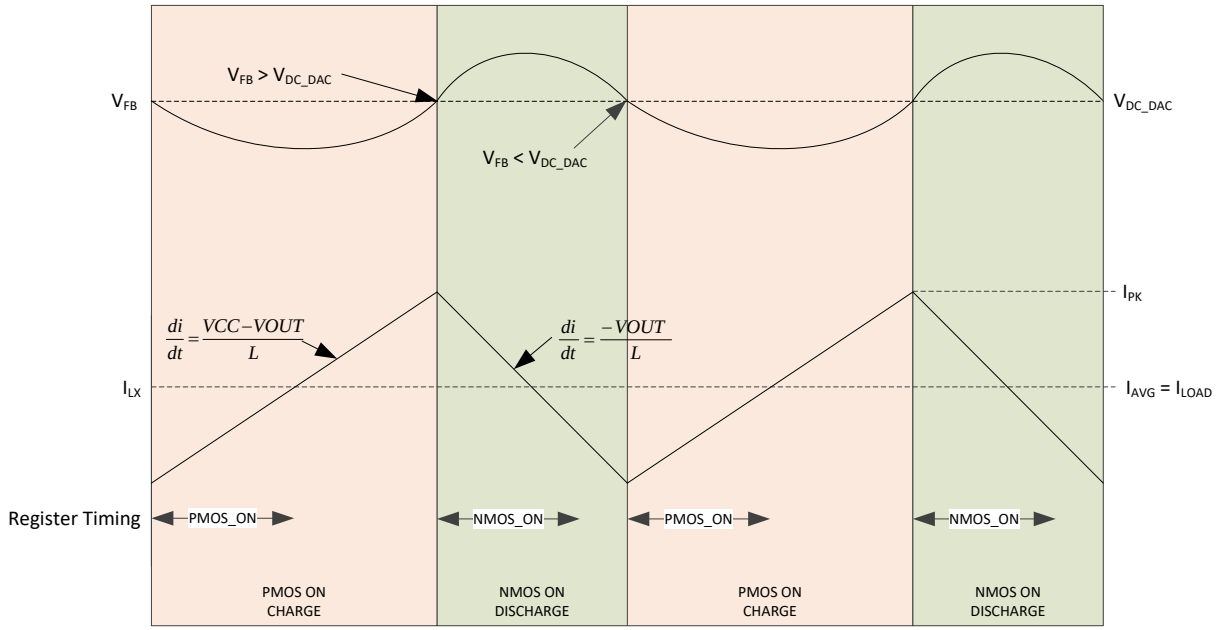


Figure 7-2: Buck Converter Operation

During the charging phase, the PMOS is turned on and the timer for PMOS_ON is started. The PMOS_ON register will set the minimum time that the PMOS will be turned on. The PMOS will remain on until V_{FB} exceeds V_{DC_DAC} and the PMOS_ON timer has expired.

At the completion of the charging phase, the controller will turn off the PMOS and turn on the NMOS, beginning the discharge phase. Current from the inductor will now flow to the output. The NMOS will remain on until the V_{FB} voltage falls back below V_{DC_DAC} and the NMOS_ON timer has expired. The NMOS_ON register is used to set a minimum time for NMOS to be on during this discharge phase.

To improve efficiency and output ripple, it is advantageous to have the fastest switching frequency possible with a buck converter. To accomplish this with the DS4835/36 controller, the PMOS_ON and NMOS_ON registers should be set to a minimal value. When these registers are kept to a minimum, the operation of the buck converter will be controlled by the feedback voltage and not the timers. This will result in the fastest operation.

7.1.2 Normal Operation with Timers Bypassed

The DS4835/36 buck controller can operate at a higher switching frequency than the normal mode of operation. This is done by entering bypass operation. In this mode of operation, the PMOS_ON and NMOS_ON timers are not used. The comparison for the expiration of these timers is not done at the completion of the charging and discharging phase. Not having to do these logic comparisons results in an even faster switching frequency. Bypass mode is useful in getting higher switching frequency, but the frequency changes with current, supply & temperature is more than the non-bypass mode.

7.1.3 Operation during Startup

When the buck converter is first enabled, it operates in startup mode. During startup mode, the converter operates in a discontinuous conduction mode of operation. This means that the inductor current falls all

the way to zero during each cycle. The operation of the buck converter during startup is shown in Figure 7-3.

During the charging phase, the PMOS is turned on for the time duration specified by PMOS_ON_ST. When the PMOS on-time has been reached, the converter then turns the NMOS on and the discharge phase begins. During startup mode, the NMOS will remain on until the inductor current reaches zero and the timer for NMOS_ON_ST has expired.

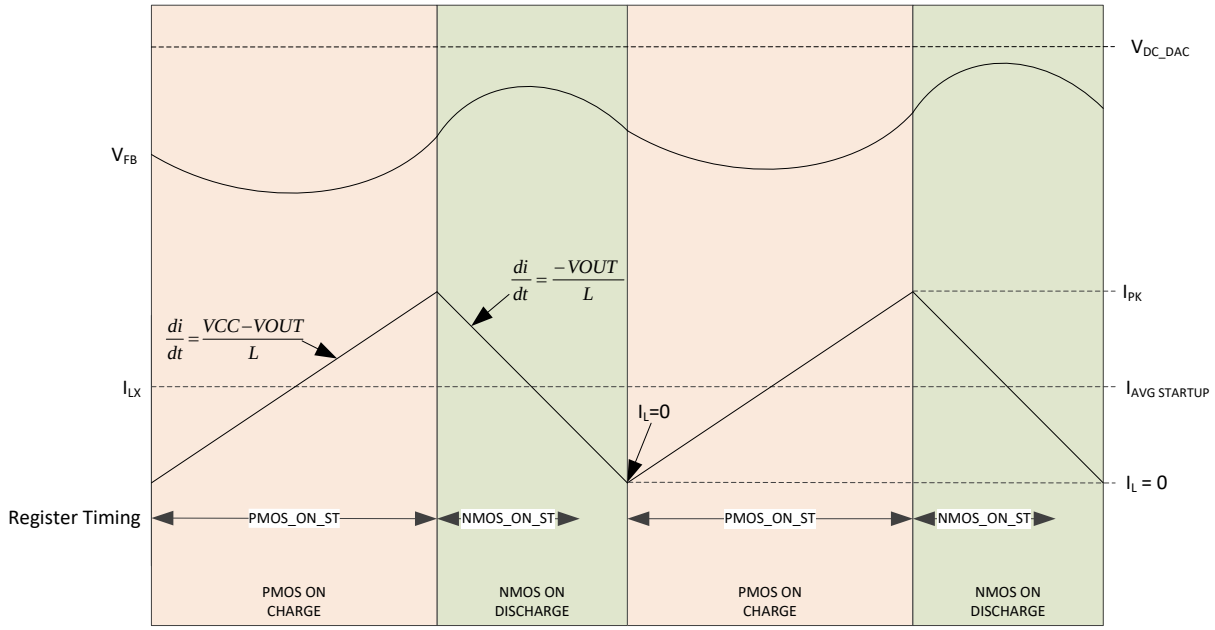


Figure 7-3: Buck Operation during Startup (NMOS_ON_ST is less than time required for current to reach zero)

The buck converter will remain in startup mode until the feedback voltage, V_{FB}, exceeds the V_{DC_DAC} setpoint voltage for the first time. Once this occurs, the controller will exit startup mode and enter normal operation.

During startup operation, the peak inductor current is limited by the PMOS_ON_ST setting. This means that the startup load current is also limited by PMOS_ON_ST. The following equations show this relationship. These formulas are true when NMOS_ON_ST is less than time required for current to reach zero. We recommend using NMOS_ON_ST=0.

$$I_{PK} = \frac{(VCC-VOUT)\times PMOS\_ON\_ST}{L} \quad \text{and} \quad I_{AVG\ STARTUP} = \frac{I_{PK}}{2}$$

When the buck converter is first started, V_{OUT} will be starting from zero volts. This will result in the first several cycles of the buck converter having a very large peak inductor current and probably violating module inrush specifications. To reduce this peak inductor current when initially turning on, the DS4835/36 buck controller will use a small PMOS on time, and ramp this time up to the time defined by PMOS_ON_ST. The first PMOS on time used will be PMOS_ON_ST / 32. Each successive cycle will increment the PMOS on time by 1 until PMOS_ON_ST is reached.

7.1.4 Setting the Output Voltage

As shown in Figure 7-1, the output voltage is determined by a comparison of V_{FB} and V_{DC_DAC}. When operating as a buck converter, an internal resistor divider with a ratio of 0.4 is placed between the FB pin and V_{FB}.

The voltage at V_{DC_DAC} is set using the 10-bit voltage DAC controlled by the VRDCDC register. This DAC has an offset voltage of approximately 150mV. Each step of the DAC corresponds to approximately 1.26mV. To calculate the setting for VRDCDC, the following equation can be used.

$$VRDCDC[9:0] = \frac{(V_{OUT} \times 0.4) - 150mV}{1.26mV}$$

The DAC offset will have some device to device variation. Because of this, the equation above should only be used to get an approximate output voltage. Calibration or closed loop control of the output voltage may be required.

The output voltage set point should not be changed in large steps once the converter is started up. If a large (>10LSBs) adjustment of output voltage is desired, VRDCDC should be increased in steps less than 10LSBs or the convertor should be disabled and re-enabled after the new voltage setting is in place.

7.1.5 Current Limiting

The DS4835/36 provides an option to limit the inductor current after startup has completed. This current limit is enabled by setting the POSLIM_EN bit in the IZDAC register. This will enable an inductor current limit of ~1.2A. If the current limit is exceeded, the controller will re-enter startup operation and attempt to power up again. Current limiting is not required during startup, as the current is inherently limited by the PMOS_ON_ST time.

7.1.6 Optional PI Control Loop

Each DCDC has an option to use a Proportional Integral (PI) control loop to improve the DC line/load regulation. During normal operation, the DCDC output voltage may change because of changes in VCC or load current. By using the PI control loop, the output DC line and load regulation can be improved.

When the PI control mode is enabled, the VRDCDC register value is not decided by the user. The PI control loop determines the appropriate VRDCDC value in a closed loop operation. The final value for VRDCDC will be:

VRDCDC (Initial Register Setting) + PI_control output

The VRDCDC register can still be written and can be used as an initial setpoint for the DCDC.

ADC configurations 18,19 and 20 must be used for feedback channels for DCDC1, 2, 3 respectively for PI loop operation. As an example, for enabling the PI loop on DCDC1, use configuration register 18. This ADC configuration must be enabled and have options such as priority and averaging setup. For this configuration, the positive channel will be ADC4 (FB1 pin) and negative channel will be internal ground. The ADC should be enabled before turning on the PI mode of the DCDC.

The ADC value from the DCDC feedback is automatically sent to the PI controller, where it is subtracted from the setpoint to calculate the error. This error is used by the PI controller to determine the VRDCDC output.

The PI controller setpoint can be calculated, as follows

Setpoint = DCDC Target Voltage * Divider Ratio * 65536 / VREF (in unipolar mode)

Setpoint = DCDC Target Voltage * Divider Ratio * 32768 / VREF (in bipolar mode and range = 1)

where Divider Ratio is 1/3 if the DIV_EN bit is set, or is equal to any external feedback network.

Once all the PI control loop constants such as setpoint, and P and I coefficients are set, the PI controller can be turned on by setting PI_EN bit in the PICNT register.

7.1.7 TX DISABLE SHUTDOWN

The DCDC converter can be shutdown automatically by a TX DISABLE event, if the TXD_EN bit is set. When this bit is set, the assertion of the Programmable Logic Array (PLA) 1 will cause the DCDC to shutdown. The DCDC enable bit will be automatically be reset. Firmware will need to re-enable the DCDC, as well as the PI control loop if being used, to start the DCDC operation again.

7.2 Buck Register Description

Following are the registers that are used to control the DCDC controller. The DS4835/36 has 3 DCDC buck controllers. The DCDC_SEL register is used to select which of the 3 DCDC controllers the following registers point to during each read or write of the register. Note that DCDC3 can also be used as a boost controller. Some of the registers for DCDC3 will have different effects when operating as a boost controller.

Note: The registers to control the buck controller are located in Module 6. Register bits in Module 6 cannot be individually written. Instead the entire register must be written.

DCDC Select (DCDC_SEL)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED														DCDC_SEL[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

The DCDC_SEL register is used to select one of the three DCDC controllers. All of the other DCDC registers are shared and the DCDC_SEL will determine which DCDC controller is being accessed by these registers.

DCDC Control (DCCN)

Bit	15	14	13:10	9	8	7:6	5	4	3	2	1	0
Name	RES	BOOST	RES	DCDC_EN	DIV_EN	RES	IZ	RES	BYP	NEG_I	CONT	RES
Reset	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

Bit	Name	Description
15	RES	Reserved. Set to 0.
14	BOOST	0: Operate in buck mode. 1: Operate in boost mode. (DCDC3 only)
13	RES	Reserved. Set to 0.
12	RES	Reserved. Set to 0.
11	RES	Reserved. Set to 0.
10	RES	Reserved. Set to 0.
9	DCDC_EN	Enable the DCDC controller.
8	DIV_EN	When set to 1, enables a divide by 3 resistor divider between the FB pin and the corresponding ADC channel.
7	RES	Reserved. Set to 0.
6	RES	Reserved. Set to 0.
5	IZ	Set to 1 for buck operation.
4	RES	Reserved. Set to 0.
3	BYP	1: Bypass operation enabled. 0: Non-bypass operation enabled.
2	NEG_I	No effect on buck mode. Set to 1.
1	CONT	When set to 1, the DCDC will operate in a continuous mode of operation and the zero-crossing comparator output will be disabled.
0	RES	Reserved. Set to 0.

VRDCDC

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED						VRDCDC[9:0]									
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

Sets the DAC voltage (V_{DC_DAC}) that is used as a comparison for V_{FB}. This is a 10-bit DAC with fullscale=1.4V and LSB=1.4mV.

PMOS_ON

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	PMOS_ON[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

Sets the minimum time that the PMOS will be turned on during normal operation.

NMOS_ON

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	NMOS_ON[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

Sets the minimum time that the NMOS will be turned on during normal operation.

PMOS_ON_ST

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	PMOS_ON_ST[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

Sets the time that the PMOS will be turned on during startup operation.

NMOS_ON_ST

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	NMOS_ON_ST[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

Sets the maximum time that the NMOS will be turned on during startup operation.

IZDAC

Bit	15:11	10:8	7:6	5	4:0
Name	RES	TONMAX	RES	POSLIM_EN	RES
Reset	0	0	0	0	0
Access	rw	rw	rw	rw	rw

Bit	Name	Description
[15:11]	RES	Reserved. Set to 0.
[10:8]	TONMAX	Limits TON time for either FET during normal operation **TONMAX** / **On Time** 000 / 64d 001 / 128d 010 / 256d 011 / 512d 1XX / disabled
7:6	RES	Reserved. Set to 0.
5	POSLIM_EN	0: Current limiting during charging phase disabled. 1: Enable current limiting during charging phase.
[4:0]	RES	Reserved. Set to 3.

DCDC OPTIONS (DCDC_OPT)

Bit	15:10	9	8:0
Name	RES	TXD_EN	RES
Reset	0	0	0
Access	rw	rw	rw

Bit	Name	Description
[15:10]	RES	Reserved. Set to 0.
9	TXD_EN	When this bit is set, the DCDC will shutdown when the output of PLA #1 is asserted.
8:0	RES	These bits should be set to 30h for proper operation.

PI Control (PICNT)

Bit	15	14	13:12	11:3	2:0
Name	PI_EN	WINDUP_EN	UPDATE_RATE	RESERVED	PIDX
Reset	0	0	0	0	0
Access	rw	rw	rw	rw	rw

Bit	Name	Description
15	PI_EN	Enables PI control loop. When this bit is set to 1, the PI loop determines the value for VRDCDC.
14	WINDUP_EN	Enables wind-up functionality for the PI controller. When enabled, the controller will stop accumulating error when the output has reached its maximum value.
13:12	UPDATE_RATE[1:0]	The Update Rate bits determine the update rate for the PI control loop. This update rate is independent of the ADC sample rate for the DCDC. **Update Rate** / **PI Loop Update Time** 0 / 16 us 1 / 32 us 2 / 64 us 3 / 128 us
11:3	RESERVED	Reserved. Set to 0.
[2:0]	PIDX[2:0]	Index bits for the PIDAT register **PIDX[2:0]** / **PI Register** 0 / PI Setpoint 1 / KP Proportional Constant 2 / KI Integral Constant 3 / Positive Clamp 4 / Shift Register

PI SETPOINT (PIDATA when PIDX = 0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	PI_SETPOINT[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

This is the setpoint for the PI controller. Results of the ADC conversion for the DCDC feedback channel are compared to this setpoint by the PI controller. If the ADC is set to run in bipolar mode, this needs to be a 16-bit, signed 2's complement value. If unipolar mode is used, this is an unsigned, 16-bit value.

KP (PIDATA when PIDX = 1)
KI (PIDATA when PIDX = 2)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	KP[15:0] or KI[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

These registers contain the PI controller's proportional and integral coefficients. These are 16-bit signed values. These values will be shifted right by KP_SHIFT or KI_SHIFT bits to provide decimal point accuracy.

PI OUTPUT CLAMP (PIDATA when PIDX = 3)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RES	OUTPUT_CLAMP[9:0]										RESERVED				
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

This register sets the maximum output value that the PI control loop can add to the intial VRDCDC register setting.

PI SHIFT REGISTER (PIDATA when PIDX = 4)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED								KI_SHIFT[3:0]				KP_SHIFT[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

The shift values provide a right-shift factor for the corresponding KP and KI coefficients. The shift values are used to provide decimal point accuracy for the coefficients.
Example, if KP = 2111 and KP_SHIFT = 11, then the effective value of KP will be 2111 / (2^{11}) = 1.0308.

7.3 Buck Converter Applications Information

7.3.1 Recommended Operation

The DS4835/36 buck converters were designed to provide flexibility to meet the needs of many applications. Maxim has tested and optimized one particular setup and this is our recommended setup. This setup uses the schematic shown in Figure 7-1, with inductors being either a TOKO 1285AS-H-2R2M or Murata LQM21PN2R2MGHL.

To get the best performance out of this circuit, Maxim recommends that the register settings shown in Table 7-3 be used.

Register	Value	Description
DCCN	032Eh	Enables buck operation in higher frequency bypass mode.
PMOS_ON	0018h	Sets PMOS on time
NMOS_ON	0000h	Set to minimum value
PMOS_ON_ST	0069h	Sets the PMOS on time during startup. This setting provides 200mA of startup current at 1.8V.
NMOS_ON_ST	0000h	Set to zero to guarantee no deadtime between consecutive inductor cycles.
IZDAC	0203h	Current limit not enabled.
DCDC_OPT	0030h	Modify to enable TXD_EN feature.

Table 7-3: Recommended Register Settings for Buck Converter

7.3.2 Buck Converter Layout Recommendations

Figure 7-4 shows a recommended layout for a buck DCDC when the application only requires the one buck converter. This is based on the schematic in Figure 7-1. When laying out the buck DCDC on the PCB, proper layout techniques must be followed to achieve good performance. Following are some layout recommendations.

- Minimize the distance from the ground node of the input capacitors (C1, C2) to the ground of the DS4835/36, which is the exposed pad.
- Minimize the distance from the ground node of the output capacitors (C3, C4) to the ground of the DS4835/36, which is the exposed pad.
- Minimize the length of the LX trace
- Minimize trace length and resistance between the input capacitors and the inductor.
- Minimize trace length and resistance between the VOUT node and the output capacitors.

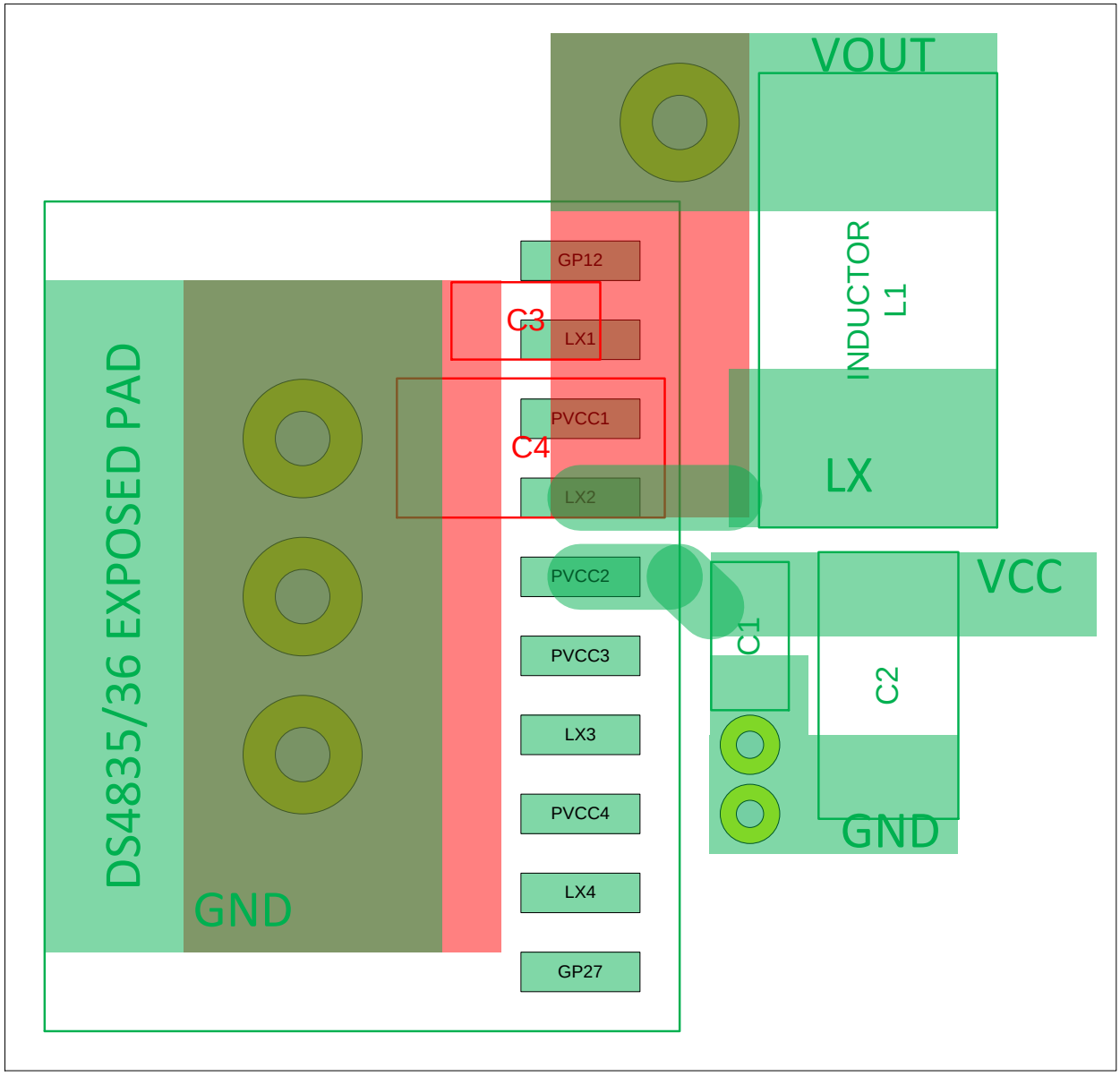


Figure 7-4: Recommended Buck Converter Layout

SECTION 8 – DCDC BOOST CONVERTER

The DS4835 (not DS4836) contains a synchronous boost DCDC converter, which includes the switching FETs and controller hardware. This can be used to generate a boosted voltage for laser diode biasing. The boost voltage can be up to 4.0V. Figure 8-1 is a diagram showing the DS4835's boost DCDC converter and the required external components. The DS4835 boost converter is available using LX3 and PVCC3/VBOOST.

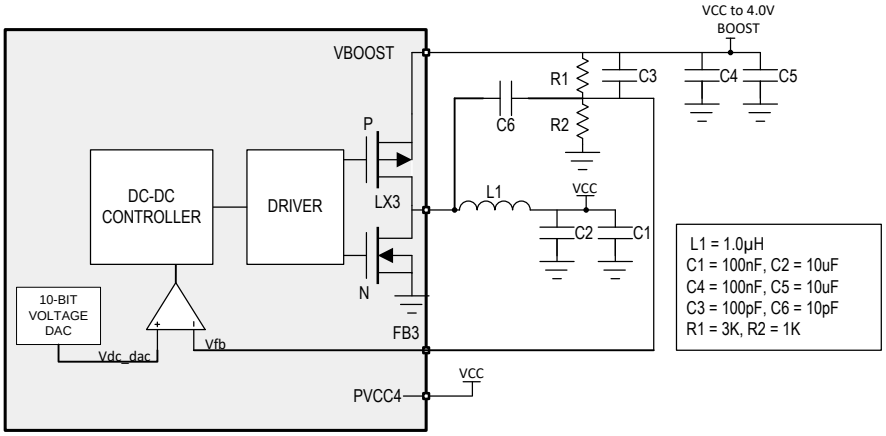


Figure 8.1: Boost DCDC Diagram

To operate the DS4835's boost controller, it must be connected to the proper DS4835 pins. The DS4835 also has a dedicated ADC channel and ADC configuration for the boost converter operation. Table 8-1 provides a list of the dedicated resources required for the operation of the boost controller.

DCDC	LX	PVCC	FB	FB ADC	ADC Config for PI Loop
3	LX3	PVCC3	GP07	ADC7	Config[20]

Table 8-1: Buck DCDC Resources

8.1- Boost Controller Operation

The DS4835 boost converter is a constant on-time controller. This means that during every cycle, the NMOS will be turned on for the same time. The time while the NMOS is turned on is known as the charging time. It is during this time that energy is developed and stored in the inductor. So, with the continuous on-time controller, the same amount of energy will be stored in the inductor during each cycle.

At the completion of the NMOS on-time and charging phase, the NMOS will be turned off and the PMOS is turned on. During this phase, known as the discharge phase, the energy that was stored in the inductor travels through the PMOS to the output caps, which are C4 and C5 in Figure 8.1, and boosts the output voltage. The PMOS on time is close loop controlled by the DCDC controller.

The DS4835 boost controller operates in two different modes. When first enabled, the controller is operating in startup mode. Startup mode is designed to limit the boost converter's inductor current so module inrush specifications are not exceeded. At the completion of the converter startup, the converter then enters normal operation, where it will remain until the converter is disabled.

There are four registers that are used to control the time that the NMOS and PMOS are on during each cycle. These are described in Table 8-2. The time represented by each of these settings can be calculated using the following formula. All of the DCDC timing is based off the DS4835's internal 128MHz clock.

Transistor On Time = Register Setting * (1/128MHz)

Register	Description
NMOS_ON	Sets the time that the NMOS is on each cycle during normal operation
PMOS_ON	Sets the minimum time the PMOS is on during each cycle of normal operation
NMOS_ON_ST	Sets the time the NMOS will be on each cycle during startup operation.
PMOS_ON_ST	Sets the maximum time the PMOS is on during each cycle of startup operation

Table 8.2: Boost DCDC Timing Registers

8.1.1 Normal Operation

During the normal mode of operation, the converter operates in a continuous mode of operation. Figure 8-2 shows the operation of the boost controller during normal operation

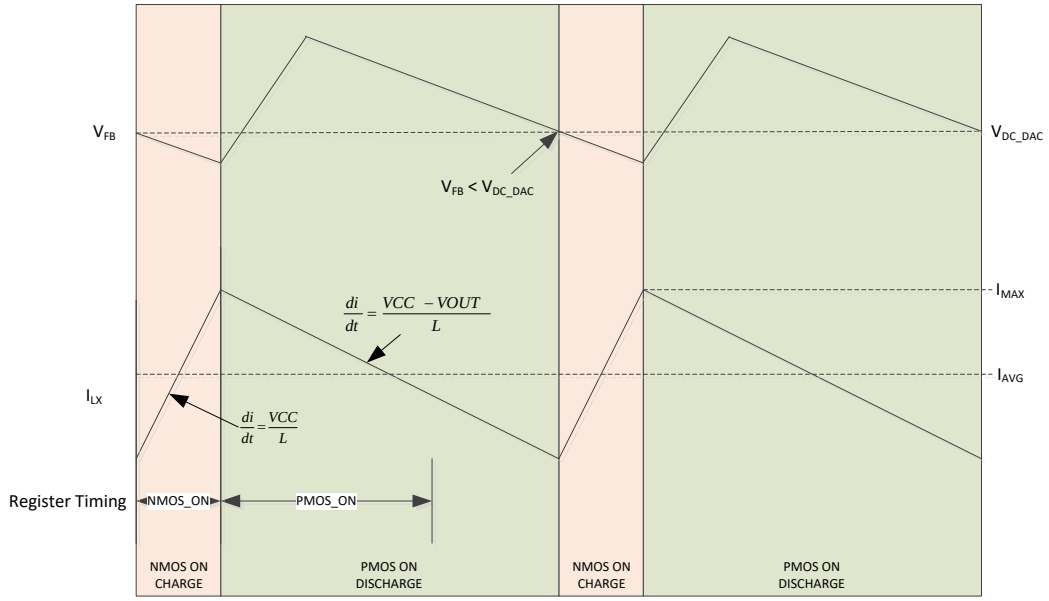


Figure 8.2: Boost Converter Operation

During normal operation, the DS4835 Boost Controller operates in continuous conduction mode, which means that one of the FETS will always be conducting. During the charging phase, the NMOS is turned on for the time duration specified by NMOS_ON. This results in a change in the inductor current. This change in inductor current is determined by the time the NMOS is on, which is set using the NMOS_ON register.

$$\Delta I_L = \frac{V_{CC}}{L} \times NMOS\_ON\_TIME$$

When the NMOS on-time has been reached, the converter then turns the PMOS on and the discharge phase begins. Current from the inductor will now flow to the output capacitor and boost the output voltage, which results in Vfb exceeding V_{DC_DAC}. The PMOS will remain on until the Vfb voltage falls back below V_{DC_DAC}. The PMOS_ON register is used to set a minimum time for PMOS to be on during this discharge phase.

To improve efficiency and output ripple, it is advantageous to have the fastest switching frequency possible with a boost converter. To accomplish this with the DS4835 controller, the NMOS_ON register is adjusted. The NMOS must be turned on long enough that enough energy is stored in the inductor to boost the output so that Vfb exceeds V_{DC_DAC}. Additional NMOS on time results in more energy being transferred to the output during the discharge phase. This causes an increase in ripple, and the switching period will be longer as it will take more time for Vfb to fall back below V_{DC_DAC}.

The PMOS_ON register sets the minimum time that the PMOS must be turned on. This is to ensure that Vfb can exceed V_{DC_DAC} prior to looking for Vfb falling below V_{DC_DAC}, which will end the discharge phase. The duty cycle of

the boost converter can be calculated using the equation below. Knowing the duty cycle and NMOS on time, the PMOS on time can then be calculated. To minimize the switching period, the PMOS on time should be set slightly lower than the calculated value.

$D = 1 - \frac{Vcc}{Vout}\eta$ where η is the efficiency of the boost converter.

8.1.2 Operation During Startup

When the boost converter is first enabled, it operates in startup mode. During startup, it is desirable to operate in discontinuous mode to limit inrush current. The operation of the boost converter during startup is shown in Figure 8-3.

During the charging phase, the NMOS is turned on for the time duration specified by NMOS_ON_ST. When the NMOS on-time has been reached, the converter then turns the PMOS on and the discharge phase begins. At the beginning of the discharge phase, the current into load capacitor is greater than load current, therefor boosting the output voltage. During startup mode, the PMOS will remain on until either of the following events occurs.

1) The DS4835's detects that the inductor current has reached zero. Or
2) The timer set by PMOS_ON_ST has expired.

During startup it is desirable to have a long PMOS_ON_ST time to allow the inductor current to reach zero, thus allowing the converter to remain in discontinuous operation.

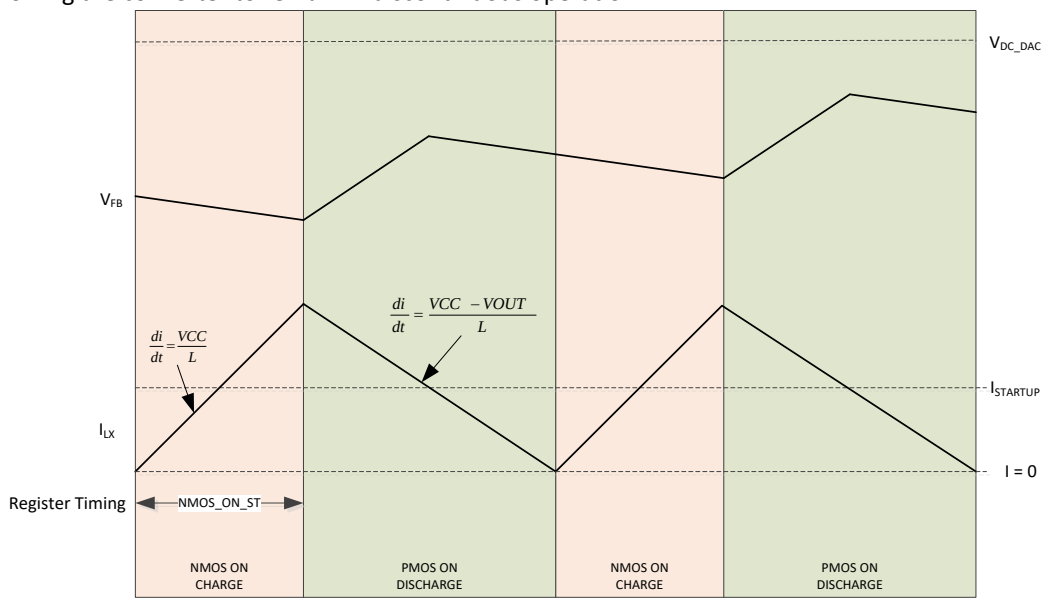


Figure 8.3: Boost Operation during Startup

The boost converter will remain in startup mode until the feedback voltage, Vfb, exceeds the V_{DC_DAC} setpoint voltage for the first time. Once this occurs, the controller will exit startup mode and enter normal operation. The architecture of this boost DCDC relies on ripple to be injected into the FB node. During startup operation, the comparator between Vfb and VDC_DAC is masked for the first 100us, so this ripple on FB will be ignored. Usually after this 100us time, the ripple injected into Vfb will cause the controller to exit startup operation. The settings for NMOS_ON_ST and PMOS_ON_ST will limit the load current that can be supplied while the boost controller is starting up.

The recommended startup procedure for the boost converter is that during startup the output voltage will charge to approximately 100mV greater than VCC. This should be accomplished by the setting of NMOS_ON_ST and PMOS_ON_ST during the 100us time prior to the comparator being enabled. At the completion of startup operation, then the output should be ramped to the final voltage either by FW increasing VRDCDC or the PI control loop bringing the output to the final value.

8.1.3 Setting the Output Voltage

As shown in Figure 8-1, the output voltage is determined by a comparison of V_{FB} and V_{DC_DAC}. The voltage at V_{DC_DAC} is set using the 10-bit voltage DAC controlled by the VRDCDC register. This DAC has an offset voltage of approximately 150mV. Each step of the DAC corresponds to approximately 1.26mV.
To calculate the setting for VRDCDC, the following equation can be used to calculate the voltage at the FB3 pin. Resistors R1 and R2 will determine the ratio between V_{FB} and V_{BOOST}.

$$VRDCDC[9:0] = \frac{V_{FB} - 150mV}{1.26mV}$$

The DAC offset will have some device to device variation. Because of this, the equation above should only be used to get an approximate output voltage. Calibration or closed loop control of the output voltage may be required.

The output voltage set point should not be changed in large steps once the converter is started up. If a large (>10LSBs) adjustment of output voltage is desired, VRDCDC should be increased in steps less than 10LSBs or the convertor should be disabled and re-enabled after the new voltage setting is in place.

8.1.4 Optional PI Control Loop

The DCDC has an option to use a Proportional Integral (PI) control loop to improve the DC line/load regulation. During normal operation, the DCDC output voltage will change because of changes in VCC or load current. It is recommended to use the PI control loop as this will significantly improve the output DC line and load regulation.

When the PI control mode is enabled, the VRDCDC register value is not decided by the user. The PI control loop determines the appropriate VRDCDC value in a closed loop operation. The final value for VRDCDC will be:

VRDCDC (Initial Register Setting) + PI_control output

The VRDCDC register can still be written and can be used as an initial setpoint for the DCDC.

ADC configuration must be used for feedback channels for DCDC3 for PI loop operation. This ADC configuration must be enabled and have options such as priority and averaging setup. The ADC should be enabled before turning on the PI mode of the DCDC. It is recommended to use the internal resistor divider between the FB3 pin and the ADC, which is enabled by DCCN.DIV_EN. This resistor divider will prevent the ADC sampling from impacting the Vboost output.

The ADC value from the DCDC feedback is automatically sent to the PI controller, where it is subtracted from the setpoint to calculate the error. This error is used by the PI controller to determine the VRDCDC output.

The PI controller setpoint can calculated, as follows

Setpoint = DCDC Target Voltage * Divider Ratio * 65536 / VREF (in unipolar mode)

Setpoint = DCDC Target Voltage * Divider Ratio * 32768 / VREF (in bipolar mode and range = 1)

where Divider Ratio is the ratio of the external resistor divider as well as the internal divider network between FB3 and the ADC.

Once all the PI control loop constants such as setpoint, and P and I coefficients are set, the PI controller can be turned on by setting PI_EN bit in the PICNT register.

8.1.5 TX Disable Shutdown

The DCDC converter can be shutdown automatically by a TX DISABLE event, if the TXD_EN bit is set. When this bit is set, the assertion of the Programmable Logic Array (PLA) 1 will cause the DCDC to shutdown. The DCDC enable bit will be automatically be reset. Firmware will need to re-enable the DCDC, as well as the PI control loop if being used, to start the DCDC operation again.

8.2 Boost Register Description

Following are the registers that are used to control the DCDC boost controller. Note that the boost controller is only available on DCDC3, which can also be used as a buck controller. The registers described below detail boost operation. Some of the registers will have different effects when operating as a buck controller.

Note: The registers to control the boost controller are located in Module 6. Register bits in Module 6 cannot be individually written. Instead the entire register must be written.

DCDC Select (DCDC_SEL)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED														DCDC_SEL[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

The DCDC_SEL register is used to select one of the three DCDC controllers. All of the other DCDC registers are shared and the DCDC_SEL will determine which DCDC controller is being accessed by these registers. To use the boost converter, set DCDC_SEL to 3.

DCDC Control (DCCN)

Bit	15	14	13:10	9	8	7:6	5	4	3	2	1	0
Name	RES	BOOST	RES	DCDC_EN	DIV_EN	RES	IZ	RES	BYP	NEG_I	CONT	RES
Reset	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

Bit	Name	Description
15	RES	Reserved. Set to 0.
14	BOOST	0: Operate in buck mode. 1: Operate in boost mode. (DCDC3 only)
13	RES	Reserved. Set to 0.
12	RES	Reserved. Set to 0.
11	RES	Reserved. Set to 0.
10	RES	Reserved. Set to 0.
9	DCDC_EN	Enable the DCDC controller.
8	DIV_EN	When set to 1, enables a divide by 3 resistor divider between the FB pin and the corresponding ADC channel. Note, this is the signal to the ADC, not the FB comparator for the DCDC converter.
7	RES	Reserved. Set to 0.
6	RES	Reserved. Set to 0.
5	IZ	Set to 0 for boost operation.
4	RES	Reserved. Set to 0.
3	BYP	Set to 0 for boost operation.
2	NEG_I	Set to 1 for boost operation.
1	CONT	When set to 1, the DCDC will operate in a continuous mode of operation and the zero-crossing comparator output will be disabled.
0	RES	Reserved. Set to 0.

VRDCDC

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED						VRDCDC[9:0]									
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

Sets the DAC voltage (V_{DC_DAC}) that is used as a comparison for V_{FB}. This is a 10-bit DAC with fullscale=1.4V and LSB=1.4mV.

NMOS_ON_ST

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	NMOS_ON_ST[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

Sets the time that the NMOS will be turned on during startup operation.

NMOS_ON

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	NMOS_ON[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

Sets the time that the NMOS will be turned on during normal operation.

PMOS_ON_ST

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	PMOS_ON_ST[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

Sets the maximum time that the PMOS will be turned on during startup operation.

PMOS_ON

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	PMOS_ON[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

Sets the minimum time that the PMOS will be turned on during normal operation.

IZDAC

Bit	15:11	10:8	7:6	5	4:0
Name	RES	TONMAX	RES	POSLIM_EN	RES
Reset	0	0	0	0	0
Access	rw	rw	rw	rw	rw

Bit	Name	Description
[15:11]	RES	Reserved. Set to 0.
[10:8]	TONMAX	Limits TON time for either FET during normal operation **TONMAX** / **On Time** 000 / 64d 001 / 128d 010 / 256d 011 / 512d 1XX / disabled
7:6	RES	Reserved. Set to 0.
5	POSLIM_EN	0: Current limiting during charging phase disabled. 1: Enable current limiting during charging phase.
[4:0]	RES	Reserved. Set to 3.

DCDC OPTIONS (DCDC_OPT)

Bit	15:10	9	8:0
Name	RES	TXD_EN	RES
Reset	0	0	0
Access	rw	rw	rw

Bit	Name	Description
[15:10]	RES	Reserved. Set to 0.
9	TXD_EN	When this bit is set, the DCDC will shutdown when the output of PLA #1 is asserted.
8:0	RES	These bits should be set to 30h for proper operation.

PI Control (PICNT)

Bit	15	14	13:12	11:3	2:0
Name	PI_EN	WINDUP_EN	UPDATE_RATE	RESERVED	PIDX
Reset	0	0	0	0	0
Access	rw	rw	rw	rw	rw

Bit	Name	Description
15	PI_EN	Enables PI control loop. When this bit is set to 1, the PI loop determines the value for VRDCDC.
14	WINDUP_EN	Enables wind-up functionality for the PI controller. When enabled, the controller will stop accumulating error when the output has reached its maximum value.
13:12	UPDATE_RATE[1:0]	The Update Rate bits determine the update rate for the PI control loop. This update rate is independent of the ADC sample rate for the DCDC.

Update Rate	**PI Loop Update Time**
0	16 us
1	32 us
2	64 us
3	128 us

11:3	RESERVED	Reserved. Set to 0.
[2:0]	PIDX[2:0]	Index bits for the PIDAT register

PIDX[2:0]	**PI Register**
0	PI Setpoint
1	KP Proportional Constant
2	KI Integral Constant
3	Positive Clamp
4	Shift Register

PI SETPOINT (PIDATA when PIDX = 0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	PI_SETPOINT[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

This is the setpoint for the PI controller. Results of the ADC conversion for the DCDC feedback channel are compared to this setpoint by the Boost PI controller. If the ADC is set to run in bipolar mode, this needs to be a 16-bit, signed 2's complement value. If unipolar mode is used, this is an unsigned, 16-bit value.

KP (PIDATA when PIDX = 1)
KI (PIDATA when PIDX = 2)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	KP[15:0] or KI[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

These registers contain the PI controller's proportional and integral coefficients. These are 16-bit signed values. These values will be shifted right by KP_SHIFT or KI_SHIFT bits to provide decimal point accuracy.

PI OUTPUT CLAMP (PIDATA when PIDX = 3)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RES	OUTPUT_CLAMP[9:0]										RESERVED				
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

This register sets the maximum output value that the PI control loop can add to the intial VRDCDC register setting.

PI SHIFT REGISTER (PIDATA when PIDX = 4)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED								KI_SHIFT[3:0]				KP_SHIFT[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

The shift values provide a right-shift factor for the corresponding KP and KI coefficients. The shift values are used to provide decimal point accuracy for the coefficients.

Example, if KP = 2111 and KP_SHIFT = 11, then the effective value of KP will be 2111 / (2^{11}) = 1.0308.

8.3 Boost Converter Applications Information

8.3.1 Recommended Operation

The DS4835 boost converter was designed to provide flexibility to meet the needs of many applications. Maxim has tested and optimized one particular setup, and this is our recommended setup. This setup uses the schematic shown in Figure 8.1, with inductors being either a TOKO 1285AS-H-2R2M or Murata LQM21PN2R2MGHL.

To get the best performance out of this circuit, Maxim recommends that the register settings shown in Table 8.3 be used.

Register	Value	Description
DCCN	4306h	Enables DCDC for boost mode operation.
NMOS_ON	000Eh	Sets the time the NMOS is on each cycle during normal operation
PMOS_ON	0010h	Sets the minimum PMOS time for each cycle. The PMOS will turn off when Vfb < V_{DC_DAC}.
NMOS_ON_ST	0010h	Sets the NMOS on time during startup. With the recommended settings and schematic, a startup load current of 30mA can be supported.
PMOS_ON_ST	0200h	Sets the maximum time the PMOS is on during each cycle of startup operation. This long time will ensure discontinuous conduction operation.
IZDAC	0403h	This register must be set to this value to ensure proper boost operation.
DCDC_OPT	0030h	Modify to enable TXD_EN feature.

Table 8.3: Recommended Register Settings for Boost Converter

8.3.2 Connection of PVCC4

In addition to the connections for LX3, VBOOST, and FB3, the boost converter also needs to have PVCC4 (pin23) connected to a VCC supply. This pin supplies power to the boost controller. Switching will not occur if the PVCC4 pin is not connected to a power supply.

8.3.3 Parasitic Body Diode of Boost Converter PMOS

The converter's PMOS transistor has a parasitic body diode. This diode causes current to be able to flow from the boost converter's VCC supply, through the inductor and this parasitic diode, to the output. Because of this, when the boost converter is first enabled, the output should already be charged to a level approximately one diode drop below VCC. This also means that there is a connection between the boost converter's VCC and the converter's output caps and any circuitry powered by this supply. When designing the power network for an optical module, the boost converter VCC must be controlled by a current limiting inrush circuit.

8.3.4 Boost Converter Layout Recommendations

Figure 8.4 shows a recommended layout for the boost DCDC when the application only requires the one boost converter. When laying out the boost DCDC on the PCB, proper layout techniques must be followed to achieve good performance. Following are some layout recommendations.

- Minimize the distance from the ground node of the input capacitors (C1, C2) to the ground of the DS4835, which is the exposed pad.
- Minimize the distance from the ground node of the output capacitors (C4, C5) to the ground of the DS4835, which is the exposed pad.
- Minimize the length of the LX trace
- Minimize trace length and resistance between the input capacitors and the inductor.
- Minimize trace length and resistance between the VBOOST pin and the output capacitors.

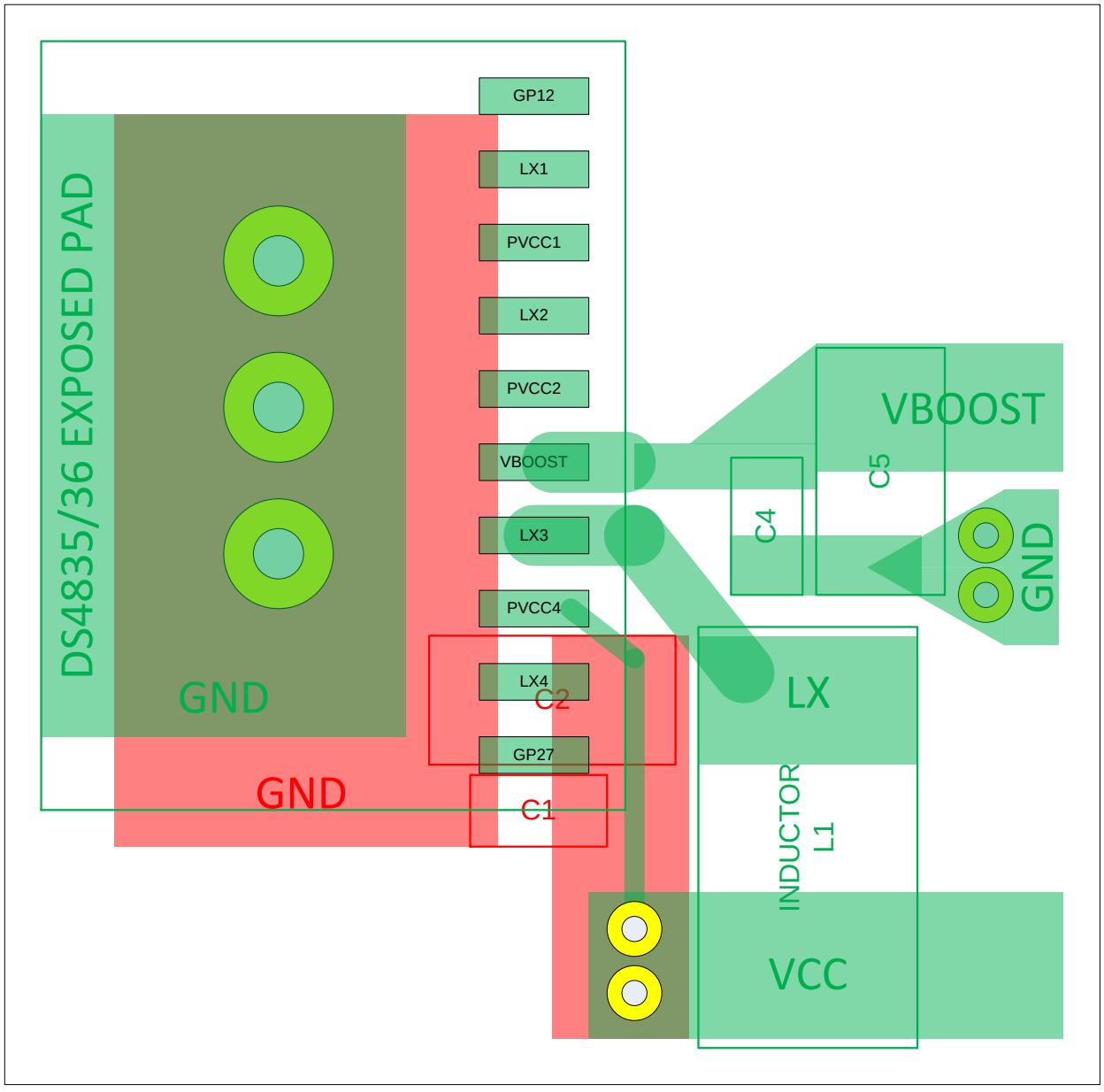


Figure 8.4: Recommended Boost Converter Layout

SECTION 9 – EXTERNAL DCDC CONTROLLER

The DS4835/36 has four independent DCDC controllers that can be used to drive external DCDC circuits. These controllers can be used for buck or boost converters or inverting DCDC converters. Figure 9.1 shows the circuit used for external boost operation. The voltage achievable with the external boost circuit is limited only by the selection of external components. Figure 9.2 shows an inverting DCDC circuit which can be used to generate a negative voltage required for EML lasers.

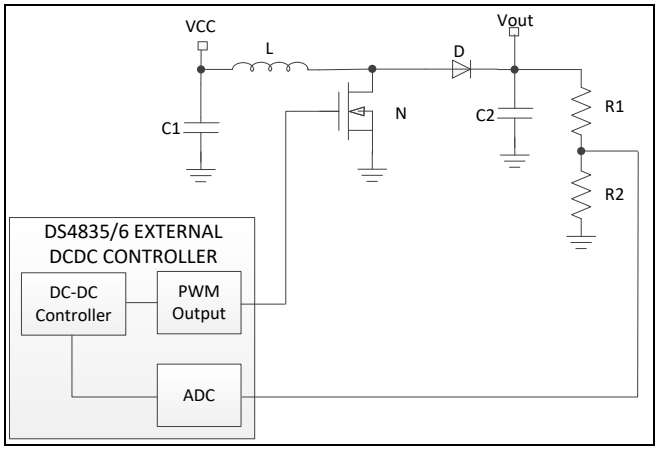


Figure 9.1: Boost DC-DC Converter

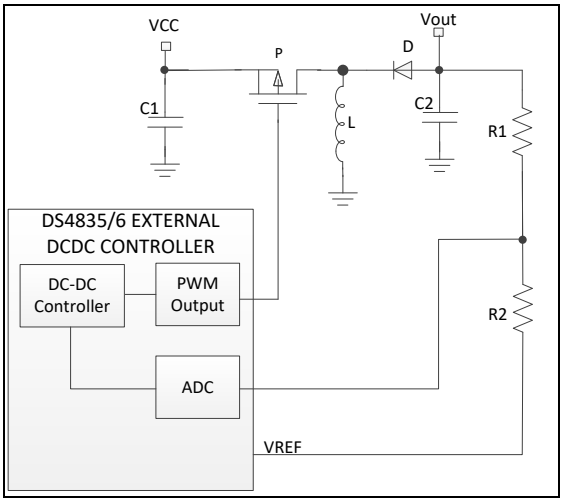


Figure 9.2: Inverting DC-DC Converter

Note: The DS4835/36 PWM outputs for external DCDC converters are switching digital signals. Precaution should be taken in layout to prevent this signal from corrupting adjacent signals.

In an optical application, these DCDC controllers may be used to create bias voltages for an APD. For this reason, the pins and registers for these DCDC controllers use an APD naming scheme. However, these controllers can be used for any type of DCDC converter. It is important that that each DCDC controller is using the correct resources. These are detailed in Table 9.1. Also refer to Table 6-2 in the ADC section for more details.

	PWM Output Pin	ADC Configuration for Feedback	Quick Trip for HLOS Shutdown	ADC Configuration for HLOS Shutdown
APD1	GP22	Config[12]	HTI[0]	Config[0]
APD2	GP23	Config[13]	HTI[1]	Config[1]
APD3	GP17	Config[14]	HTI[2]	Config[2]
APD4	GP20	Config[15]	HTI[3]	Config[3]

Table 9.1: APD Controller Resources

9.1 – Controller Operation

The DS4835/36 external DCDC controllers use a PI control loop to regulate the output voltage. The PI control algorithm runs at 128MHz. The feedback voltage from the output is input to the ADC. The DCDC controller compares this feedback voltage conversion to the target setting to create an error component for the PI control loop. The output of the controller is a duty cycle with 15-bit resolution. The DS4835/36 external DCDC controller uses a delta-sigma algorithm to increase the effective switching frequency compared to a normal duty-cycled PWM controller. The output is normally a 500kHz fixed frequency output, with options to also operate at 1MHz or 250kHz.

The PI controller has clamps that can be used to limit the incremental change of the control loop. Clamps are provided for positive and negative error, accumulated error, and absolute duty cycle.

9.1.1: DCDC Feedback

The feedback from the DCDC output can be routed to any ADC channel. It is important that the feedback channel is assigned to the correct ADC configuration. Refer to Table 11.1 for more details.

The DCDC controller calculates error using the equation:

Error = Setpoint – Feedback

If this error value is positive, then the duty cycle will be increased. Likewise, a negative error value will result in the duty cycle being decreased.

To improve the transient response of the DCDC controller, the number of LSBs of error should be maximized. This is achieved by selecting a resistor divider ratio that utilizes the full range of the ADC converter.

9.1.2: Setting the Output Voltage for Boost Operation

The DCDC converter's output voltage is set using a combination of the Vtarget register and the voltage divider created by R1 and R2. The DS4835/36's ADC maximum input voltage is VREF. So Vout must be divided down using a resistor divider to generate a feedback voltage that is less than VREF. The DCDC controller will compare the raw ADC result for this feedback channel to the Vtarget register and adjust the output accordingly. Following is the formula to calculate the desired Vtarget setting.

$Vtarget[15:0] = (Vout * FBR * 2^{16}) / VREF$

Where: Vout is the desired output voltage
FBR is the feedback ratio = R2/(R1+R2)
VREF is the ADC reference voltage
Assuming unipolar conversion and 16-bits of resolution.

To minimize the effects of ADC noise, only the upper 12 bits of Vtarget and the ADC result are compared. To best optimize this range, feedback resistors should be chosen so the feedback voltage is near the ADC full scale voltage.

9.1.3: Inverting DCDC Operation

To operate as an inverting DCDC converter requires a little more setup. Following are the steps required to configure the DS4835/36 and the PCB to work for an inverting DCDC. This is followed by an example.

1) As shown in Figure 9.2, a resistor divider is placed between the negative output voltage and the VREF pin. The resistors must be scaled so that the feedback voltage that is input to the ADC is a positive voltage.
2) Set the ADC configuration to perform a bipolar conversion for the inverting DCDC feedback. Use the feedback channel as the negative channel and internal ground as the positive channel. This will result in the conversion always being signed and negative.
3) The DCDC PI control setpoint is expecting an unsigned target. Set the DCDC setpoint to the exact expected signed ADC value. So, if B000h is the expected signed result from the ADC, set the DCDC setpoint to B000h. For more details, refer to the example below.
4) The inverting DCDC uses a PMOS switch, so the DCDC controller's output should be inverted by setting APDCN.INV_PWM.

Example of this operation assuming VREF = 2.4V, R1 = 50K and R2 = 10K.

Target Voltage = -3V.
Feedback Voltage = 2.4V – 10K(2.4V - -3V)/(10K + 50K) = 1.5V.
The ADC result will be: $-1.5/2.4*2^{15} = -20480 = B000h$

At startup, Vout = 0V.
Feedback Voltage = 2.4V – 10K(2.4V - 0V)/(10K + 50K) = 2.0V.
The ADC result will be: $-2.0/2.4*2^{15} = -27307 = 9555h$

Set the DCDC setpoint to be B000h.

At startup the output is 0V. 9555h is less than B000h because the DCDC controller does an unsigned comparison. So, the DCDC PI controller will increase the duty cycle. The PWM output is inverted, so this results in an increasing negative duty cycle which creates a more negative output voltage. The DCDC controller will continue to increase the duty cycle until the feedback reading equals B000h.

9.1.4: Startup Operation

When the controller is first started, different clamp levels can be used to provide a controlled startup. If these clamps are adjusted properly they can prevent the module from violating inrush current specifications as the DCDC converter is ramping. A different startup clamp is provided for positive error. An additional clamp is provided that limits the incremental change of the output duty cycle. The clamp of incremental duty cycle can be disabled using the DIS_STRT_CLMP bit. The controller will remain in startup operation until the feedback voltage exceeds the target setpoint for the first time.

The recommended startup procedure is to set the startup error clamp equal to the normal error clamp values. Then use the startup incremental duty cycle clamp to control the ramp time as the DCDC controller is ramping.

9.1.5: Inverting the PWM Output

Some systems may require the PWM output to be inverted, such as when a PMOS is being driven in an inverting DCDC. For these applications, the INV_PWM bit can be set. The appropriate use of this bit should allow these DCDC controllers to work for any DCDC application.

9.1.6: Fast Shutdown

These DCDC controllers have an option that allows them to be shut down immediately when an overvoltage event is detected. An example of this is if the APD voltage exceeds a certain limit, the APD voltage can be

immediately shutdown to prevent damage. This feature is enabled if the HLOS_EN bit is set. Quick trip comparisons from the ADC are used to trigger this shutdown. For APD1, the HTI[0] bit will be used to shut down this controller. Likewise, for controller 2, the HTI[1] bit will be used, and so on.

When the DCDC controller is disabled, the PWM duty cycle will be forced to 0. The controller will hold the duty cycle at 0 until the quick trip flag is cleared. Upon clearing of the quick trip flag, the output will switch back to the last calculated duty cycle or to a new duty cycle if a new feedback sample was completed during the HLOS event. This may cause very high currents and voltages. It is recommended that when a fast shutdown occurs, firmware disables the DCDC controller and re-enables later when appropriate.

9.2 – External DCDC Register Description

Each of the four converters has three independent registers for setup and configuration. These registers are APDCNn, APDIDXn, and APDDATn, where n = 1 to 4.

Note: The registers to control the external DCDC controller are located in Module 6. Register bits in Module 6 cannot be individually written. Instead the entire register must be written.

APD CONTROL (APDCNn)

Bit	15	14	13	12	11:7	6	5	4:3	2	1:0
Name	INV_PWM	HLOS_EN	X2_FREQ	HALF_FREQ	RES	DSMOD_EN	DS_STRT_CLMP	RES	EN	RES
Reset	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15	INV_PWM	If this bit is set, the PWM output is inverted. This can be used for driving different DCDC architectures.
14	HLOS_EN	When this bit is set, a HLOS quick trip will disable the DCDC controller output.
13	X2_FREQ	X2_FREQ=1; PWM output frequency is doubled. 1 MHz operation. X2_FREQ=0; PWM output frequency is normal. 500 kHz operation.
12	HALF_FREQ	HALF_FREQ=1; PWM output frequency is halved. 250 kHz operation. HALF_FREQ=0; PWM output frequency is normal. 500kHz operation.
11:7	RES	RESERVED
6	DSMOD_EN	Enable delta sigma modulation. This bit must be set to 1.
5	DIS_STRT_CLMP	This bit should be set to 0 to ensure a controlled ramp at startup.
4:3	RES	RESERVED
2	EN	Enable DCDC converter EN=1; Enables DCDC converter EN=0; Disables DCDC converter
1:0	RES	RESERVED

APD INDEX (APDIDXn)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED												APCIDX[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

The APDIDX is an index register that is used to select which register is accessed by the APDDAT register.

APDDATA (APDDATn)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	APDDATA[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

The APDDAT register is used to configure the external DCDC converter. The configuration register accessed by APDDAT is controlled by the setting of the APDIDX register. Following is a description of different APDDAT registers that can be accessed based upon the setting of APDIDX.

Note: The UROM_copyBuffer command inadvertently writes to APDDAT4. If APD4 is being used, set APDIDX4 to a value greater than or equal to 9 after initializing or making changes to APD4. This will ensure that APDDAT4 is in a non-writable area if the UROM_copyBuffer command is called.

<table>
<tr><th>APDIDX</th><th>NAME</th><th>Description</th></tr>
<tr><td>0x0h</td><td>Error Clamp Start[7:0]</td><td>Positive error clamp during the startup phase. Set this equal to the normal positive error clamp.</td></tr>
<tr><td>0x1h</td><td>High Duty Clamp[15:8]
Low Duty Clamp[7:0]</td><td>High Duty Clamp[7:0]: 8-bit clamp used to provide a clamp for the upper range of the duty cycle. This 8-bit clamp is applied to the upper 8 bits of the duty cycle.
Low Duty Clamp[7:0]: 8-bit clamp used to provide a clamp for the lower range of the duty cycle. This 8-bit clamp is applied to the upper 8 bits of the duty cycle.</td></tr>
<tr><td>0x2h</td><td>Vtarget[15:0]</td><td>This register is used to set the target value that is compared to the feedback's ADC result.</td></tr>
<tr><td>0x3h</td><td>Integral Gain[15:0]</td><td>Integral Gain Term. 1 LSB corresponds to a gain of 1/256. Example, for a gain of 1.0, set the Integral Gain to 0100h.</td></tr>
<tr><td>0x4h</td><td>Duty Start Clamp[6:0]</td><td>This register clamps the incremental change in the duty cycle. This occurs only during startup when DIS_STRT_CLMP is set to zero.</td></tr>
<tr><td>0x5h</td><td>Integrator Clamp[15:0]</td><td>This clamps the integration accumulation to prevent integral windup from occurring. This should be set to a maximum of: 65535 – (error clamp x integral gain)/256</td></tr>
<tr><td>0x6h</td><td>Negative Error Clamp[7:0]</td><td>Value used to clamp the negative error. This is an absolute value. For example, to have the negative error clamped at a value of -50, set this register to 50d.</td></tr>
<tr><td>0x7h</td><td>Positive Error Clamp[7:0]</td><td>Value used to clamp the negative error.</td></tr>
<tr><td>0x8h</td><td>Proportional Gain[15:0]</td><td>Proportional Gain Term. 1 LSB corresponds to a gain of 1/256. Example, for a gain of 1.0, set Proportional Gain to 0100h.</td></tr>
<tr><td>0x9h</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>0xAh</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>0xBh</td><td>Feedback Error</td><td>Readback of the current error value. This is a 13-bit signed value.</td></tr>
<tr><td>0xCh</td><td>Clamp Status</td><td>Status of the control loop clamps
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>15:10</td><td>Reserved</td></tr>
<tr><td>9</td><td>Set when FB_Error < Negative Error Clamp</td></tr>
<tr><td>8</td><td>Set when FB_Error > Positive Error Clamp</td></tr>
<tr><td>7:0</td><td>This is the clamped error value that will be used for the control loop.</td></tr>
</table></td></tr>
<tr><td>0xDh</td><td>Duty Cycle</td><td>The current duty cycle is stored in this register. This register is read only and should not be written.</td></tr>
</table>

9.3 – Applications Information

Component selection
The proper selection of components plays a critical role in the efficiency and overall operation of a DCDC converter. Following are some tips for selecting the proper components.

Boost Converter for APD biasing
This application may require an output voltage of up to 80V. Components must be chosen that can withstand this voltage, provide for the highest efficiency, and be the proper physical size to fit in the application.

- When selecting a transistor, the critical parameters to look for are a drain-to-source voltage rating greater than the output voltage and the lowest drain-to-source on resistance. The transistor must also be rated to handle the peak inductor current and should have a small gate capacitance.
- An inductor should be chosen with a low DC resistance and a maximum current rating that will support the application.
- The diode chosen must be rated with a reverse voltage greater than the output voltage and with a forward current rating greater than the peak inductor current. Minimizing the forward voltage drop will make the system more efficient.
- Output capacitors must be chosen with a voltage rating greater than the target output voltage.

Inverting DCDC Converter for EA Biasing

- When selecting a transistor, the critical parameters to look for are low drain-to-source on resistance, a small gate capacitance, and a drain current rated to handle the peak inductor.
- An inductor should be chosen with a low DC resistance and a maximum current rating that will support the application.
- Minimizing the forward voltage drop is critical to improve efficiency in an inverting DCDC. Most inverting DCDC circuits are targeting approximately -3.3V. Any drop across a diode is a large percentage of this output voltage, thus a large loss of efficiency.

Layout Guidelines

- To reduce the input switching loops place input capacitors as close as possible to MOSFET GND pin.
- Minimize the length of switching node, which is the drain of the transistor.
- Minimize the distance between all the ground nodes in the switching circuit.
- Minimize the trace length and resistance between the input capacitor (C1) and the inductor and/or transistor.
- Minimize trace length and resistance between the VOUT node and the output capacitor (C2).
- Additional filtering of the output may be required to reduce ripple

SECTION 10 – DIGITAL BUCK DC-DC CONVERTER

The DS4835/36 provides one additional DCDC buck converter using the PVCC4 and LX4 pins. However, this DCDC uses a different DCDC controller than what is used for the buck converters on LX1, LX2, and LX3.

10.1- Digital Buck Controller Operation

The DCDC converter on LX4 uses the controller that is designated for APD4. Doing this makes this a digital control loop, instead of the analog control loops that are used for DCDC1, DCDC2, and DCDC3. Figure 10-1 shows a block diagram of this DCDC controller.

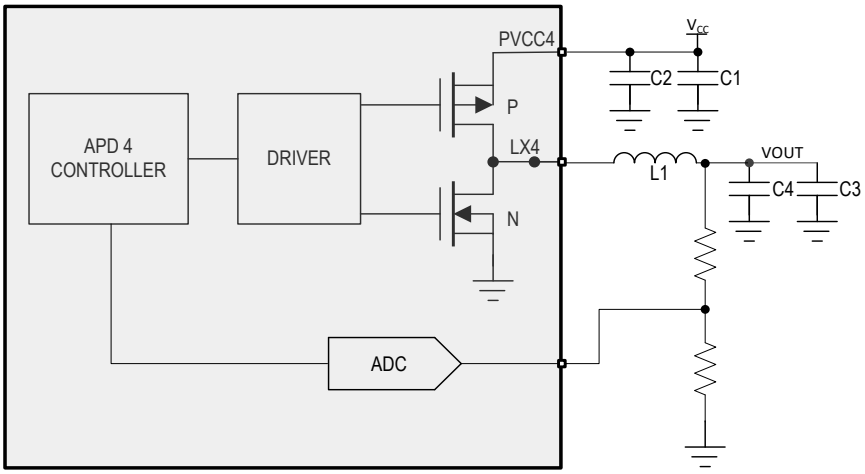


Figure 10.1: Digital Buck Controller Block Diagram

To enable this buck controller, the DCDC4 bit in MIS1 must be set. Setting this bit will allow the APD4 Controller to drive the FETs for LX4. The feedback for this DCDC converter can be routed to any ADC channel. But the feedback must use ADC Configuration 15 so that the ADC conversion will be routed to the APD controller. The DCDC setpoint and all controller configurations are then setup using the APD4 registers. Refer to the External Boost section of this user's guide for more information regarding the APD registers and setup. When the digital buck converter is operating, the APD4 pin will not be used. This pin can be used as a GPIO or any of its other functions.

Unlike the other DCDC controllers on the DS4835/36, DCDC4 will operate at a fixed switching frequency. This frequency is determined by register settings and can be 250 kHz, 500 kHz, or 1 MHz.

It must be noted that this DCDC converter will have a slow response time. The response time will be determined not only by the selection of the external components, but also by the ADC sampling time. The feedback channel must be sampled by the ADC, so ADC priority, averaging, and the total number of channels being sampled will all effect the total response time of this DCDC controller. Also, the settings of the APD controller's proportional and integral coefficients will impact the response time of this DCDC converter. This DCDC converter may not have good transient response times and should not for applications that require very small line and load transient response.

10.2 Digital Buck Controller Register Description

MIS1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	DCDC4	-	-	-	-	-	-	-	-	-	-	-	-
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
15:14	Reserved.	Reserved. Set these bits to 0.
12	DCDC4	Setting this bit to 1 will enable the APD4 controller to drive the FETs for DCDC4 located at LX4.
11:0	Reserved.	Reserved. Set these bits to 0.

SECTION 11 – THERMO-ELECTRIC COOLER CONTROLLER

The DS4835/36 contains two Thermo-Electric Cooler (TEC) Controllers. The TEC controllers use a PID control loop and are designed to use a single inductor topology to reduce PCB size and cost and improve efficiency. A block diagram of the TEC controller and the required external components is shown in Figure 11.1.

11.1 – TEC Operation

The TEC controller operates by comparing the TEC's thermistor voltage, which is captured using the ADC, to the TEC setpoint. The error between these two values is then sent to a PID controller. The PID control output is used as PWM signal to drive the DS4835/36's internal FETs designed for TEC operation.

The duty cycle is signed 16 bit number, Duty[15:0]. The MSB is the sign bit, which indicates the direction of current flow, either heating or cooling. The lower 15 bits determine the duty cycle. A pulse spreading algorithm is then used to increase the TEC's switching frequency. This pulse spreading algorithm is similar to the algorithm used by the DS4835/36 PWM blocks, refer to this section for more details. For the TEC, the 15-bit duty cycle will be spread evenly throughout 512 separate timeslots. The duration of each timeslot will be 64 of the internal 128 MHz oscillator clocks, which results in a frequency for each slot being 2 MHz. When the duty cycle is greater than 512, the switching frequency is always be 2 MHz. At lower duty cycles, the PWM starts skipping pulses. The lowest frequency occurs when the duty cycle is +/- 1LSB, which results in a 2MHz / 512 = 4KHz switching frequency.

To implement the single inductor solution, the DS4835/36 TEC controller uses one DCDC controller as a buck converter that can sink/source current connected to one side of the TEC, which is referred to as TEC+. The other side of the TEC (TEC-) connects to the switch to either VCC or ground according to heating or cooling operation.

Each TEC uses the FETs from two of the internal DCDC converters to operate. Also, each TEC requires a dedicated ADC configuration for the measurement of the thermistor voltage (Vtherm). Table 11.1 details the resources used for each of the TEC channels. Also, refer to Table 6-2 in the ADC section for more details.

TEC Channel	LX Channels Required	ADC Configuration Reserved for Thermistor Feedback
TEC1	LX1 and LX2	ADC Config 16
TEC2	LX3 and LX4	ADC Config 17

Table 11.1: TEC Resources Required

Based upon the relationship between the thermistor voltage (Vtherm) and the TEC setpoint, the TEC controller will switch between heating and cooling modes of operation. Table 11.2 describes the conditions for heating and cooling modes and how the TEC controller responds to these different modes.

Temperature	Comparison	Error Register	TEC Mode	Duty Cycle Sign	LX2/4	LX1/3
Too Hot	Vtherm < Setpoint	Positive	Cooling	Positive	Connected to GND	Sourcing Current
Too Cold	Vtherm > Setpoint	Negative	Heating	Negative	Connected to VCC	Sinking Current

Table 11.2: TEC Modes of Operation

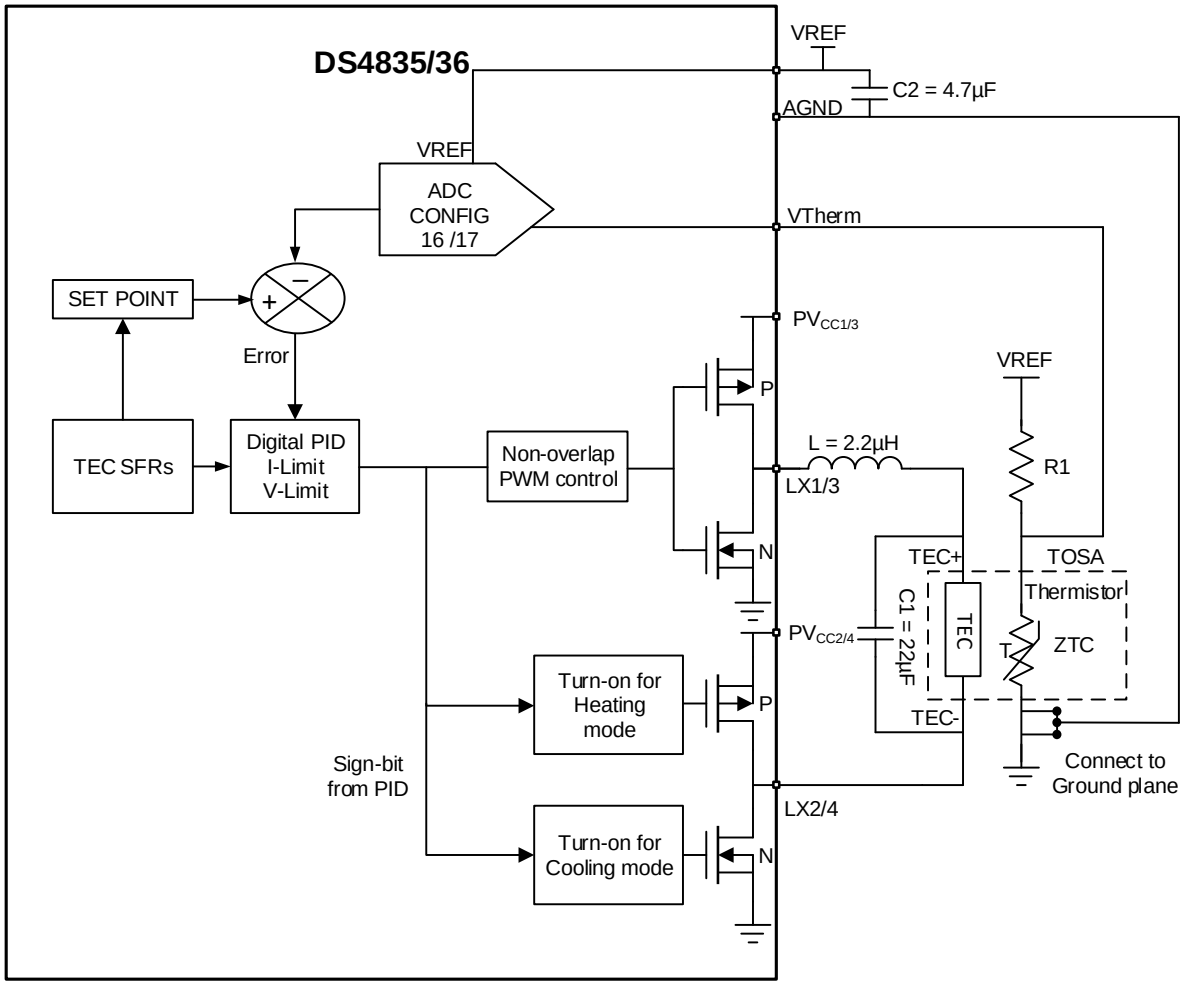


Figure 11.1: TEC Block Diagram

11.1.1 – Vtherm Ref

The ADC reference voltage available at the VREF pin is used to create a voltage-divider network with the thermistor. Using VREF for this resistor divider makes the conversion of the voltage across the thermistor ratiometric with VREF. This ensures that the TEC is not sensitive to any drift in VREF. For better performance, the AGND pin should have a Kelvin connection between TOSA ground and AGND.

11.1.2 – Zero Mode Crossing

When ambient temperature and corresponding temperature to the setpoints equals then no heating or cooling is required. At this point, TEC current should be zero (Duty Cycle = 0). The TEC controller is designed to transition between heating and cooling. As the TEC switches between heating and cooling modes, there is a possibility for slight noise in the system. This can be reduced by setting the CM_SLOW bit. When this bit is set, the TEC controller slows down the common mode switching between heating and cooling at this zero-crossing area.

11.1.3 – Compensation for Changes in VCC

The output from the PID control loop is a duty cycle for the TEC output. The voltage that is generated can be calculated by multiplying the duty cycle with VCC. This results in a change in the output voltage as VCC changes, which results in a change in the TEC current. The TEC controller can compensate for any changes in the VCC voltage by using a normalization routine, which is enabled with the VCC_COR_EN bit. To do this, the TEC controller uses the ADC's reading for VCC voltage and normalizes this against the ADC value for the minimum expected VCC voltage to create a compensated duty cycle. The equations below show the TEC voltage based on a PWM duty cycle.

Equation 1: $$TEC\ Voltage = VCC\ x\ \frac{Compensated\ Duty\ Cycle}{2\text{^}15}$$

Equation 2: $$Compensated\ Duty\ Cycle = Duty\ Cycle * \frac{2\ x\ NORMV[15:0]}{VCC\ ADC}$$

Combining these two equations results in

Equation 3: $$TEC\ Voltage = Normalized\ Voltage\ x\ \frac{Duty\ Cycle}{32768}$$

The maximum value for the normalized voltage should be the minimum expected VCC. This is called "Normalized Voltage". The NORMV register should be set half of the ADC reading for this minimum expected VCC. For example, if the minimum VCC specified is 2.9V, then NORMV[15:0] should be set to a value less than or equal to the ADC result for VCC equal to 2.9V divided by 2. For the TEC to automatically access the ADC's VCC conversion results, VCC conversions should be assigned to ADC configuration 23. Refer to Table 6-2 in the ADC section for more details.

By setting the VCC_AVG_EN bit, a 16 sample moving average can be applied to the VCC compensations. This will help to reduce the effect of transient noise on the VCC measurement and compensation.

If VCC Compensation is enabled, the TECC operation will not begin until an ADC conversion for VCC is completed following the TECC being enabled. Depending upon the ADC round robin time, this may cause a significant delay in the startup of the TECC. One method to work around this is to initially set VCC to be a priority 1 channel so this conversion will be completed quickly. Once the TEC operation has started, then the ADC conversion for VCC can be changed to a lower priority.

11.1.4 – TEC Voltage Clamp and Monitoring

The TEC controller contains clamps that can be used to limit the voltage across the TEC. These are the Positive Output Clamp and Negative Output Clamp. These can be used to clamp the output duty cycle, which acts to clamp the output voltage.

If the positive maximum voltage for the TEC is TECVmax, then using equation 3 above the value for the Positive Output Clamp (PO_CLAMP) can be calculated:

$$TECVmax = Normalized\ Voltage\ x\ \frac{PO\_CLAMP}{32768}$$

From this equation, PO_CLAMP can be calculated to be:

$$PO\_CLAMP = \frac{TECVmax}{Normalized\ Voltage}\ x\ 32768$$

Similarly, the Negative Output Clamp (NO_CLAMP) can be calculated to be:

$$NO\_CLAMP = \frac{TECVmin}{Normalized\ Voltage}\ x\ 32768$$

TECVmin will be a negative number, so, the NO_CLAMP will also negative. Store this result as signed 2's complement in the Negative Output Clamp register.

The differential voltage across the TEC can also be easily monitored by the ADC. The side of the TEC associated with LX2 or LX4 is connected internally to the ADC channels 8 and 9 respectively. By connecting the positive side of the TEC to an ADC channel, the TEC voltage can easily be monitored if the application requires this.

11.1.5 – TEC Current Limiting

Application specific current limits can be implemented with the use of either the ADC voltage monitoring or the TEC Output Clamps. These are voltage monitors, from which the current can be clamped if the resistance of the TEC is known.

11.1.6 – TEC Current Measurement

Measurements of the current through the TEC can be automatically calculated by the TEC controller. These measurements can be configured using the I_CONFIG register. This register provides bits to enable the current measurement, enable averaging of the current result, select the number of current measurements to be averaged, and select the full-scale range for the current measurement. The results of the current measurement are available in the I_MEAS register. The TEC current measurement is updated every 16us.

The actual current flowing through the TEC will depend upon external factors such as the inductor selected and the resistance of the TEC. Calibration factors need to be found, either through measurement or characterization of TEC, and then applied to the result of the current measurement to calculate the true value of current.

An alternative to using the DS4835/36's internal TEC current measurement is to estimate this current by using the ADC conversion result for the voltage across the TEC (VTEC). If the TEC has been characterized and its resistive properties are well known, then using the measured VTEC voltage divided by the TEC resistance provides an estimate of TEC current.

11.1.7 – Firmware Mode Operation

The TEC controller can be operated in mode where firmware writes directly to the duty cycle. This mode can be used for system debug and to test the efficiency of the system. This can also be used if firmware is used for the thermal loop instead of the TEC controller's PID loop. This mode of operation is enabled by setting the FMODE bit when the TEC is enabled. User firmware can now write directly to the TEC Duty Register. In this mode of operation, the TEC controller does not implement any of the output clamps.

11.1.8 – TEC Sample Time

To adjust the TEC response and performance, the TEC update rate can be adjusted using the RATE bits in the TECCN register. These bits set a sample rate from 20µs up to 2560µs. This update rate timing is independent of the ADC update for the conversion of the thermistor voltage. The ADC sample rate depends upon the ADC priority, the number of channels, and the amount of averaging applied to the thermistor conversion. It is possible for the TEC controller to oversample or undersample the ADC conversions.

11.1.9 – TEC Offset Correction for Burst Mode Systems

In a burst-mode optical system, when the laser is turned on it adds heat to the system. The heat added during this short burst time may still be enough for the laser wavelength to drift. This burst can be too short for the PID loop to respond. The TEC controller has an option to compensate for this added heat of a burst mode system. When the OFFSET_EN bit is set, an offset is added to the duty cycle whenever the GPIO selected for Burst Enable is pulsed high. This pin can be selected using the Offset Trigger Select Register. The offset to add can be set using the Duty Cycle Offset register. The offset register is a signed 16-bit register that is added to the duty cycle. If the offset is a positive value, this results in extra cooling when the offset trigger is asserted. Likewise a negative duty cycle results in extra heating when the offset trigger is asserted.

11.1.10 – Integral Windup Limit

One issue with PID controllers is integral or integrator windup. This is usually caused during startup or when there is a large change in the setpoint, which results in a significant error that is integrated. This can cause overshooting or a lag in system response. If the Integral Windup Limit is enabled by setting the WIND_EN bit, the accumulator stops integrating when the output has reached the positive or negative output clamps, which clamp the duty cycle.

11.2 – TEC Registers

Each of the TEC controllers has three dedicated registers to setup and control the TEC. These are TECCNn, TECDn, and NORMVn where n = 1 or 2.

TEC Configuration (TECCNn)

Bit	15	14	13	12	11	10:8	7	6	5	4	3:0
Name	VCC_COR_EN	CM_SLOW_EN	BIPOL_EN	RES	OFFSET_EN	RATE	TEC_EN	FMODE	WIND_EN	VCC_AVG_EN	INDEX
Reset	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	Name	Description
15	VCC_COR_EN	Setting this bit to 1 will enable the correction for VCC variation. To use this feature ADC config[23] should be used for the VCC conversion. The NORMV register needs to be set with the normalized ADC voltage
14	CM_SLOW_EN	This bit when set to 1 will enable slowing down of common mode switching at the zero crossing between heating and cooling modes of operation.
13	BIPOL_EN	Setting this bit will enable the bipolar mode for TEC operation. In this mode the Vtherm ADC conversion and TEC setpoint can be negative values. This allows use of the internal resistive divider set at VREF/2 as a reference for Vtherm.
12	Reserved	Reserved. Set to 0.
11	OFFSET_EN	Setting this bit will trigger an offset addition to the TEC duty cycle based on a selectable GPIO trigger.
10:8	RATE[2:0]	Update Rate. These bits are used to select the update rate for the PID controller. RATE[2:0] / PID Sampling Time (us) 0 / 20 1 / 40 2 / 80 3 / 160 4 / 320 5 / 640 6 / 1280 7 / 2560
7	TEC_EN	Setting this bit to '1' enables the TEC controller. Setting to '0' will disable the TEC controller.
6	FMODE	Enabled a Firmware Mode. 0: The TEC operates as a closed loop controller 1: The TEC operates in an open loop. Firmware can write directly to the duty cycle of the PWM output to the TEC. In this mode, firmware can implement a software algorithm for TEC control.
5	WIND_EN	Integrator windup control enable. When this bit is set, the integrator will not be allowed to wind up and cause lag in the system response.
4	VCC_AVG_EN	Setting this bit will enable a moving average filter (length=16) on VCC conversion results. This will help remove transient noise effect from VCC. This only has an affect when VCC_COR_EN is set to 1.
3:0	INDEX	INDEX: The index chooses the target register for the TECDn register INDEX[3:0] / TEC Register 0 / PWM Duty 1 / SETPOINT 2 / KP 3 / KI 4 / KD 5 / SHIFT 6 / Positive Error Clamp 7 / Negative Error Clamp 8 / Positive Output Clamp 9 / Negative Output Clamp 10 / I_CONFIG 11 / I_MEAS 12 / Offset Trigger Select 13 / Duty Cycle Offset

TEC DUTY (TECDn when INDEX = 0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	SIGN	TEC DUTY [14:0]														
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*

This is the TEC duty cycle. When operating in closed loop operation (FMODE = 0), the current duty cycle can be read at this register. When operating in open loop (FMODE = 1), this register can be written to change the TEC duty cycle. This is a signed 2's complement register. Bits[14:0] determine the duty cycle of the TEC controller output. The sign bit determines if the controller should be operating in heating or cooling mode, with negative corresponding to heating.
Examples:
TEC DUTY = 0001h = +1; TECN is connected to GND, TEC+ is slightly above GND with a duty cycle of 1/32768.
TEC DUTY = FFFFh = -1; TECN is connected to VCC, TEC+ is slightly lower than VCC with a duty cycle of 1/32768.
Equation 3 above can be used to calculate the actual value of TECP.

TEC SETPOINT (TECDn when INDEX = 1)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	TEC SETPOINT [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

This is the setpoint for the TEC controller. Results of the ADC conversion for Vtherm are compared to this setpoint by the TEC controller. If the TEC controller is set to run in bipolar mode, this needs to be a 16-bit, signed 2's complement value. If unipolar mode is used, this is an unsigned, 16-bit value.
Note:

1. User should write TEC SETPOINT [6:0] = 0 for DS4835/36 9-bit TEC setpoint variant.
2. User should write TEC SETPOINT [3:0] = 0 for DS4835/36 12-bit TEC setpoint variant.

KP (TECDn when INDEX = 2)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	KP[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

This register contains the PID controller's proportional coefficient. This is a 16-bit signed value. This value is shifted right by SHIFT_KP bits to provide decimal point accuracy.

KI (TECDn when INDEX = 3)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	KI[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

This register contains the PID controller's integral coefficient. This is a 16-bit signed value. This value is shifted right by SHIFT_KI bits to provide decimal point accuracy.

KD (TECDn when INDEX = 4)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	KD[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

This register contains the PID controller's derivative coefficient. This is a 16-bit signed value. This value is shifted right by SHIFT_KD bits to provide decimal point accuracy.

TEC SHIFT (TECDn when INDEX = 5)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Reserved				KD_SHIFT[3:0]				KI_SHIFT[3:0]				KP_SHIFT[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

The shift values provide a right-shift factor for the corresponding KP, KI and KD coefficients. The shift values are used to provide decimal point accuracy for the coefficients.
Example, if KP = 2111 and KP_SHIFT = 11, then the effective value of KP will be 2111 / (2^{11}) = 1.0308.

TEC POSITIVE ERROR CLAMP (TECDn when INDEX = 6)
TEC NEGATIVE ERROR CLAMP (TECDn when INDEX = 7)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	POSITIVE/NEGATIVE ERROR CLAMP [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

These registers are used to clamp the error value from the comparson of the TEC SETPOINT with the ADC Vtherm result. These are 16-bit signed values.

TEC POSITIVE OUTPUT CLAMP (TECDn when INDEX = 8)
TEC NEGATIVE OUTPUT CLAMP (TECDn when INDEX = 9)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	POSITIVE/NEGATIVE OUTPUT CLAMP [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

These registers are used to set positive and negative clamps on the duty cycle. As the TEC voltage is proportional to the TEC duty cycle, these clamps can be used to limit the TEC voltage. These clamps are 16-bit, signed 2's complement values which follow the same format as the Duty Cycle register.

I_CONFIG (TECDn when INDEX = 10)

Bit	15:11	10:8	7	6	5	4	3:2	1:0
Name	RES	RES	I_MEAS_EN	AVG_DIS	MIN_MAX	RES	MEAS_RANGE	AVG_SEL
Reset	0	011	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

Bit #	Name	Description
15:11	Reserved	Reserved. Set to 0.
10:8	Reserved	Reserved
7	I_MEAS_EN	Current measurement enable. Setting this bit to 1 will enable the current measurement functionality.
6	AVG_DIS	Average Disable, setting this bit to 1 will disable the averaging on current measurement & will show the instantaneous value.
5	MIN_MAX	When this bit is set, the instantaneous peak and valley currents are reported in the I_MEAS register. By default the current measurement shows the average value.
4	Reserved	Reserved. Set to 1.
3:2	MEAS_RANGE[1:0]	Current Range Select: These bits can be used to select the range for current measurement. Selecting the proper range will improve the resolution and accuracy of the current measurement. MEAS_RANGE[1:0] / Current Range 0 / 500 mA 1 / 500 mA 2 / 1000 mA 3 / 2000 mA
1:0	AVG_SEL[1:0]	Average Select: Selects the number of current measurements to average. AVG_SEL[3:0] / Samples 0 / 256 1 / 128 2 / 64 3 / 32

I_MEAS (TECDn when INDEX = 11)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Avg = 0	S	S	S	S	S	S	S	S	S	S	2^5	2^4	2^3	2^2	2^1	2^0
Avg = 32	S	S	S	S	S	S	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Avg = 64	S	S	S	S	S	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Avg = 128	S	S	S	S	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Avg = 256	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
MIN/MAX	0	0	0	MAX[4]	MAX[3]	MAX[2]	MAX[1]	MAX[0]	0	0	0	MIN[4]	MIN[3]	MIN[2]	MIN[1]	MIN[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 contains the measured value of TEC current. The resolution of this measurement depends upon the number of averages selected and the measurment range used. This is detailed in the bit description above and the TEC Current Measurement section.

OFFSET TRIGGER SELECT (TECDn when INDEX = 12)

These bits can be used to select which GPIO pin is used to enable offset addition to the duty cycle. When the selected pin is asserted (logic high), the offset in Duty Cycle Offset will be added to the duty cycle.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED, read returns zero											TRIG_SEL[4:0]				
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

TRIG_SEL[4:0]	GPIO		TRIG_SEL[4:0]	GPIO		TRIG_SEL[4:0]	GPIO		TRIG_SEL[4:0]	GPIO
0	NONE		1	GP00		9	GP10		17	GP20
25	NONE		2	GP01		10	GP11		18	GP21
26	NONE		3	GP02		11	GP12		19	GP22
27	NONE		4	GP03		12	GP13		20	GP23
28	NONE		5	N/A		13	GP14		21	GP24
29	NONE		6	GP05		14	GP15		22	GP25
30	NONE		7	GP06		15	GP16		23	GP26
31	NONE		8	GP07		16	GP17		24	GP27

DUTY CYCLE OFFSET (TECDn when INDEX = 13)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	DUTY CYCLE OFFSET[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

The value in this register is added to the TEC duty cycle when the selected Offset Trigger GPIO is asserted and the OFFSET_EN bit is set. This is a 16-bit signed 2's complement register and follows the same format as the Duty Cycle Register. If this register is positive, this will result in extra cooling being added to the TEC when the Offset Trigger is asserted. The final duty cycle will still be clamped by the Positive and Negative Output Clamp Registers.

NORMVn

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	NORMV[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

This register is used as a to adjust the duty cycle value to normalize the output when VCC varies. This register should be set half of the ADC conversion value for the minimum VCC expected. Refer to the Compensation for Changes in VCC section for more details on this register.

SECTION 12 – AUTOMATIC POWER CONTROL (APC)

The DS4835/36 contains 4 Automatic Power Controller (APC) blocks. This hardware will automatically compare the result of an ADC conversion of TX Power to an APC Setpoint. The APC will then generate a new value for the laser bias current. Depending upon how the module is setup, this new laser bias value can be written to one of the DS4835/36 DCDC converters, to a current DAC, or directly to the transceiver using serial communication

12.1 – APC Operation

12.1.2 – Ramping

The operation of an APC is illustrated in Figure 12.1 below. The APC will start with a bias value equal to INIT_BIAS. The APC will then determine if the TXP level is greater than APC_SETPOINT. Each conversion of the APC will result in the bias current increasing by ISTEP, until the TXP level exceeds APC_SETPOINT. When APC_SETPOINT is exceeded, the device begins a binary search to quickly reach the bias current corresponding to the proper power level. After the binary search is completed, the APC integrator is enabled and single LSB steps are used to tightly control the average power. The length of the APC integrator, or filter length, is dynamic and determines how many TXP conversions are required before an up/down decision of BIASREG is made. The operation of this filter is such that the amount of averaging is varied based on the noise on the TXP signal.

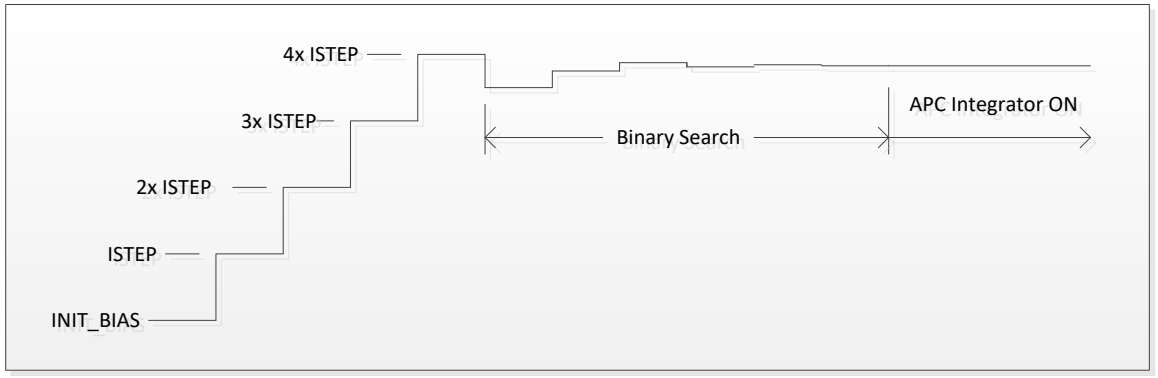


Figure 12.1: APC Ramping

To meet laser startup time requirements, some experimentation with INIT_BIAS and I_STEP may be required. A larger I_STEP value will result in a faster ramping of the optical power, but may result in some optical overshoot. INIT_BIAS can also be used to set the first bias current to a value closer to the final bias value. In some applications, it may be desirable for INIT_BIAS to be provided by a temperature indexed look up table.

12.1.2 – Sampling of TXP

An APC conversion is started when the APCCONV bit is set by firmware. When this bit is set, the APC loop will wait for the next ADC conversion of a TXP channel to complete. Each APC block has an ADC configuration that is dedicated to the APC, as shown in Table 12.1 below. The ADC configuration must be setup so that TXP is sampled as one of the configurations listed in Table 12.1. Also refer to Table 6-2 in the ADC section for more details.

ADC Configuration Register	APC Block
8	APC 1
9	APC 2
10	APC 3
11	APC 4

Table 12.1: ADC Config Registers for APC Operation

The APC uses the results of the first ADC conversion following APCCONV being set. If ADC averaging is enabled for this channel, the APC will use the averaged value. However, this is the raw ADC value. The raw ADC value used by the APC does not have ADC offset and scaling applied. The time delay from when APCCONV is set and the TXP conversion being complete will depend upon how many configurations are in the ADC sequencer, the priority given to each ADC configuration, and how many times the TXP conversion will be averaged. Refer to the ADC section for more details. Figure 12.2 shows the relationship between when ADCCONV is set, when and ADC conversion of TXP occurs, and when the ADC loop is complete. This is shown both for when the APC is ramping, as well as once the APC enters the integration phase.

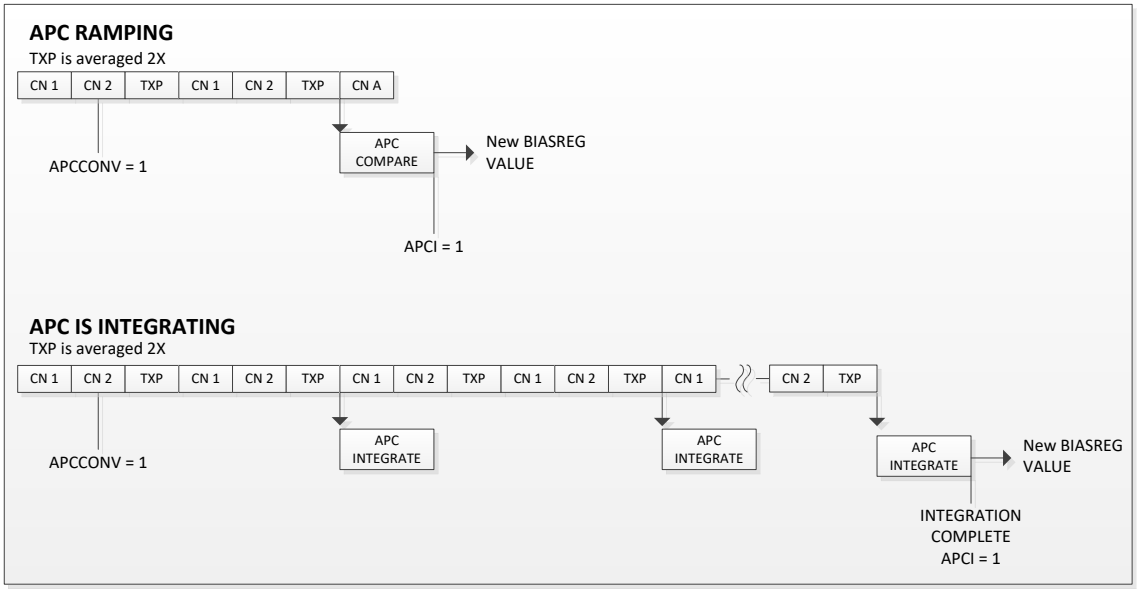


Figure 12.2: APC SAMPLING

12.1.2 – Maximum Bias

The APC will limit the bias current when the bias current exceeds MAXBIAS. When the bias level exceeds MAXBIAS, the BIAS_OVF flag will be set. Following are the actions the APC will take if the MAXBIAS is exceeded during each stage of the ramping.

- Ramping: If MAXBIAS is exceeded during the initial ramping, the BIASREG will clamp at a maximum value of MAXBIAS + I_STEP. The APC will then enter the binary search state.
- Binary Search: The binary search will complete, with the BIASREG not increasing. The binary search will always complete and not be interrupted when at MAXBIAS.
- Integration Mode: BIASREG is only allowed to change by +/- 1 LSB. There will not be any further updates until BIASREG < MAXBIAS.

12.2 – APC Registers

Each of the 4 APC peripherals has 3 dedicated registers to setup and control the APC. These are APCCNn, APCIDXn, and APDDATn where n = 1 to 4.

APC Configuration (APCCNn)

Bit	15:10	9	8	7	6:2	1	0
Name	Reserved	APCIE	APC_EN	BIAS_OVF	RESERVED	APCCONV	APCI
Reset	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw

Bit #	Name	Description
15:10	Reserved	Reserved. Set to 0.
9	APCIE	Setting this bit to '1' enables an interrupt to be generated when APCI = 1.
8	APC_EN	Setting this bit to '1' enables the APC. Setting to '0' will reset the APC controller.
7	BIAS_OVF	Flag to indicate BIASREG >= MAXBIAS
6:2	Reserved	Reserved. Set to 0.
1	APCCONV	Setting this bit to 1 will start the next APC conversion. This bit will clear when the conversion is complete. The value in BIASREG from the prior APC calculation should be read prior to setting this bit.
0	APCI	Interrupt flag that is set at the completion of an APC conversion. This bit is cleared by firmware writing a 0.

APC Index (APCIDXn)

Bit	15:3	2:0
Name	RESERVED	REGSEL[2:0]
Reset	0	0
Access	r	rw

Bit #	Name	Description
15:3	RESERVED	Reserved
2:0	REGSEL[2:0]	These bits select the source/destination for APCDATAn access.

REGSEL[2:0]	APCDATA Target Register
0	APC_TARGET
1	MAXBIAS
2	INIT_BIAS
3	I_STEP
4	BIASREG
5, 6, 7	RESERVED

APC_TARGET (APCDATAn when REGSEL = 0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	APC TARGET [15:0]															
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

This is the target level which the sampled TXD level is compared against.

MAXBIAS (APCDATAn when REGSEL = 1)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	MAXBIAS [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

This is the maximum allowed value for the bias register. The BIASREG will not exceed this value during any operation.

INIT_BIAS (APCDATAn when REGSEL = 2)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	INIT_BIAS [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

This is the initial value of BIASREG that is used by the APC when it begins its operation.

I_STEP (APCDATA when REGSEL = 3)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	I_STEP [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

I_STEP is the bias step size that the APC uses during the intial ramping. BIASREG will be incremented by I_STEP until TXP exceeds APC_SETPOINT.

BIASREG (APCDATA when REGSEL = 4)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	BIASREG [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

BIASREG is the APC output register. This is the value of bias that firmware should write to the laser driver. BIASREG is updated at the completion of an APC conversion when APCI is set. This register should be read prior to setting APCCONV and starting the next APC calculation.

APC_OPTIONS (APCOPT)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	FORCE_FILTER[7:0]								COMPARE_MASK[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	r	r	r	r	r	r	r	r	r	r	r

COMPARE_ MASK[7:0]: These bits can be used to mask bits from the APC comparison to reduce the impact of noise. The mask is applied to both the APC_TARGET and the ADC value prior to the APC comparison.

The resolution for APC filter operation anywhere between 8 and 16 bits. By default, these bits are 0x00, allowing a 16-bit comparison between APC_TARGET and the ADC results. The comparison can ignore as many bits as desired, depending on how many bits are set in this register starting from the LSB side. For example, if these bits are set to 0x0F, the lower 4 bits are ignored, and a 12-bit comparison is used to run the APC loop.

FORCE_FILTER[7:0]: These bits can be used to force the filter length of the APC loop to a minimum value. If the computed filter length is less than this value, then the filter length will be forced to this value. The filter length will be FORCE_FILTER x 4.

The settings in the APCOPT register will affect all 4 of the APC blocks.

12.3 – APC Example Code

```
void main(void)
{
 unsigned int new_bias;

 ADC_Init();                          //initialize ADC to do TXP conversions

 APCIDX1 = 1;                         //set for maxbias
 APCDAT1 = 500;                       //write maxbias

 APCIDX1 = 0;                         //set for apc target
 APCDAT1 = 0x0444;                    //write apc target

 APCIDX1 = 3;                         //set for i step
 APCDAT1 = 10;                        //set apc step size

 APCIDX1 = 2;                         //set for initial bias
 APCDAT 1= 20;                        //set initial bias
 WriteBias(20);                       //set laser driver bias to init bias

 APCCN1_bit.APC_EN = 1;               //enable APC loop
 APCCN1_bit.APCCONV = 1;              //start apc conversion

 while(1)                             //main loop
 {
  if(APCCN1_bit.APCI)                 //wait for APC conversion to complete
  {
   APCIDX1 = 4;                       //set to bias register
   new_bias = APCDAT1;                //read new bias value
   WriteBias(new_bias);               //write new bias value to laser driver

   APCCN1_bit.APCI = 0;               //clear flag
   APCCN1_bit.APCCONV = 1;            //start next conversion
  }
 }
}
```

SECTION 13 - HARDWARE PROGRAMMABLE LOGIC ARRAY

The DS4835/36 provides 4 hardware Programmable Logic Arrays (PLA). The PLAs can be used to meet optical module requirements for TXD, LOS, TXF, or similar input to output operations. This is accomplished in hardware and does not require complex software interrupts and the latency that software would cause.

Figure 13-1 shows what is implemented by each of the Programmable Logic Arrays shows the input sources for each of the PLAs.

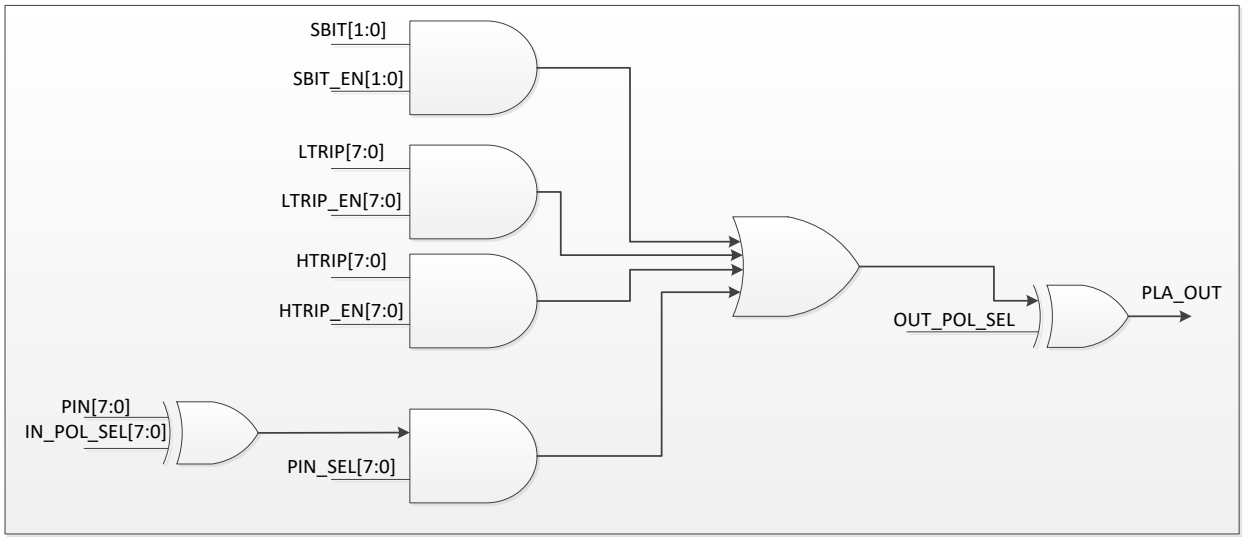


Figure 13-1: PLA Logic Functional Diagram

13.1 – Detailed Description

Each PLA controller provides a logic solution for asserting time critical signals for optical modules. There are four instances of the PLA controllers. Each PLA can use any combination of the following input sources.

- Any of the 8 Quick Trip Comparisons
- Any of the 8 PLA Input pins
- 2 Software Bits

The output from each PLA can also be routed to any of the 8 possible PLA output pins. The polarity for the PLA output pins can be configured to be either active high or active low.

13.1.1 – PLA Input Pins

Each of the PLAs can select any combination of the 8 PLA input pins using the Pin Input Select (PIS) register, which is accessed using the PLADATn. Setting a bit in the PIS register will allow the corresponding pin to be an input to the PLA. If multiple input pins are selected, the PLA output will assert when any of the selected input pins are asserted.
The polarity of the input pins can also be selected using the PIS register.

13.1.2 – PLA Quick Trip Inputs

Each PLA can also select any combination of the 8 high or low quick trip comparisons as inputs to the PLA. This is done using the Low Trip Select (LTS) and High Trip Select (HTS) registers, which are accessed with PLADATn register.

To use a quick trip source, the quick trip threshold must first be setup properly. Refer to the ADC section for more details on setting up a quick trip. If the quick trip source is enabled for a PLA, the PLA output will assert when the quick trip flag is asserted. The PLA output will remain asserted as long as the quick trip flag remains

asserted. Note, quick trip flags do not auto clear, they must be cleared by software. As an example of this operation, consider the case of implementing an RSSI based LOS. Once the low quick trip threshold is setup, the corresponding bit in LTI will assert when the received power drops below the quick trip threshold. This will automatically assert the PLA output, if this quick trip source is enabled. The PLA output will remain asserted until software clears the LTI bit. So software should not clear this bit until after the received power has increased above an acceptable level. When software clears the LTI bit, the PLA output will deassert, unless another of the enabled input sources (such as a software bit) remains asserted.

13.2 – PLACNT Register Descriptions

Each of the 4 PLAs have their own dedicated control registers, PLACNTn and PLADATn, where n = 1 to 4.

PLA Control Register (PLACNTn)

Bit	15	14	13	12:11	10:8	7	6	5	4	3	2	1:0
Name	PLAEN	PLAIE	OUT_POL	RES	OUT_SEL	SBIT2_EN	SBIT1_EN	SBIT2	SBIT1	-	PLA_OUT	REG_SEL
Reset	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	rw

BIT	NAME	DESCRIPTION
15	PLAEN	**PLA Logic Enable.** Setting this bit to '1' will enable the PLA function
14	PLAIE	**PLA Interrupt Enable**. When this bit is set to 1, then the PLA_OUT flag will generate an interrupt when asserted.
13	OUT_POL	**Selects PLA output polarity.** 0 - Active high 1 - Active low
12:11	Reserved	**Reserved**. The user should write 0 to these bits.
10:8	OUT_SEL	**Output Select.** Determines which of the 8 PLA Outputs to connect to this PLA's output. OUT_SEL[2:0] / PLA OUT Pin 000 / GP00 001 / GP01 010 / GP02 011 / GP03 100 / GP20 101 / GP21 110 / GP22 111 / GP23
7	SBEN2	**Software Bit2 Enable:** Trigger enable for software bit 2
6	SBEN1	**Software Bit1 Enable:** Trigger enable for software bit 1
5	SB2	**Software Bit 2:** Setting this bit to a '1', will assert the PLA output, if this software bit is enabled by SBEN2.
4	SB1	**Software Bit 1:** Setting this bit to a '1', will assert the PLA output, if this software bit is enabled by SBEN1.
3	Reserved	**Reserved**. The user should write 0 to this bit.
2	PLA_OUT	This bit is a read only flag showing the status of PLA. This bit is high when the PLA output is asserted. This bit is independent of output polarity select.
1:0	REG_SEL	Register Select. These bits determine which of the PLA registers the PLADAT register accesses. REG_SEL[1:0] / Register 00 / Pin Input Select (PIS) 01 / Low Trip Select (LTS) 10 / High Trip Select (HTS) 11 / Undefined

Pin Input Select Register (PLADATn when REG_SEL = 0)
When REGSEL = 0, writing to the PLADATn register writes to the Pin Input Select Register.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	IP_POL[7:0]								PIS[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
15:8	IP_POL[7:0]	**Input Pin Polarity.** These bits are used to select the polarity for the input pins enabled with the PIS[7:0] bits. 0 = active high 1 = active low
7:0	PIS[7:0]	**Pin Input Select**. Each of these 8 bits corresponds to 1 of the PLA input pins. If the bit is set, the corresponding input pin is selected as an input to the PLA.

PIS Bit	Enabled PLA Input Pin
0	GP17
1	GP05
2	GP06
3	GP07
4	GP24
5	GP25
6	GP26
7	GP27

Examples:
- If PIS[7:0] is set to 1000 0010, this allows GP27 and GP05 to be inputs to the PLA. The PLA will assert if either of these inputs are asserted.
- If PIS[7:0] is set to 1000 0010 and IP_POL[7:0] is set to 0000 0010, then GP27 will be an active high input and GP05 will be an active low input.

Low Trip Select (PLADATn when REG_SEL = 1)
When REGSEL = 1, writing to the PLADATn register writes to the Low Trip Select register.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Reserved								LTS[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

These bits select the which low quick trip comparison flags are used as inputs to the PLA.
Examples
- If LTS[7:0] is set to 0000 0101, the low quick trip comparisons for ADC configuration 0 and 2 will be used as inputs to the PLA. The PLA will assert if either of these flags are asserted.

High Trip Select (PLADATn when REG_SEL = 1)
When REGSEL = 2, writing to the PLADATn register writes to the High Trip Select register.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Reserved								HTS[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

These bits select the which high quick trip comparison flags are used as inputs to the PLA.
Examples
- If HTS[7:0] is set to 0000 0101, the high quick trip comparisons for ADC configuration 0 and 2 will be used as inputs to the PLA. The PLA will assert if either of these flags are asserted.

SECTION 14 – PWM OUTPUTS

The DS4835/36 provides 8 Pulse Width Modulated (PWM) Outputs. These PWMs have 16-bit resolution and can operate at a frequency of 2 MHz. This modulator output can be converted to a DC voltage when it's fed to an external passive low pass filter as shown in figure 14.1. For best operation, each pole of the filter should have the resistor and capacitor selected so that RC > 20.4us.

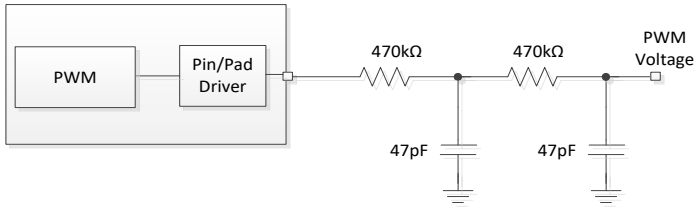


Figure 14.1: PWM Output with external low pass filter

Note: Between the DS4835/36 and the RC filter, the PWM output is a switching digital signal. Precaution should be taken in layout to prevent this signal from corrupting adjacent signals.

14.1 – Detailed Description

The DS4835/36 PWMs are clocked by the internal 128 MHz oscillator. To achieve a 2 MHz output rate, a pulse spreading algorithm is implemented. With pulse spreading, the 16-bit duty cycle is spread evenly throughout 1024 separate timeslots. The duration of the timeslots is 64 of the 128 MHz clocks, resulting in the frequency for the timeslots being 2 MHz. Figure 14.1 shows an example of how different duty cycles will be output using the DS4835/36's pulse spreading.

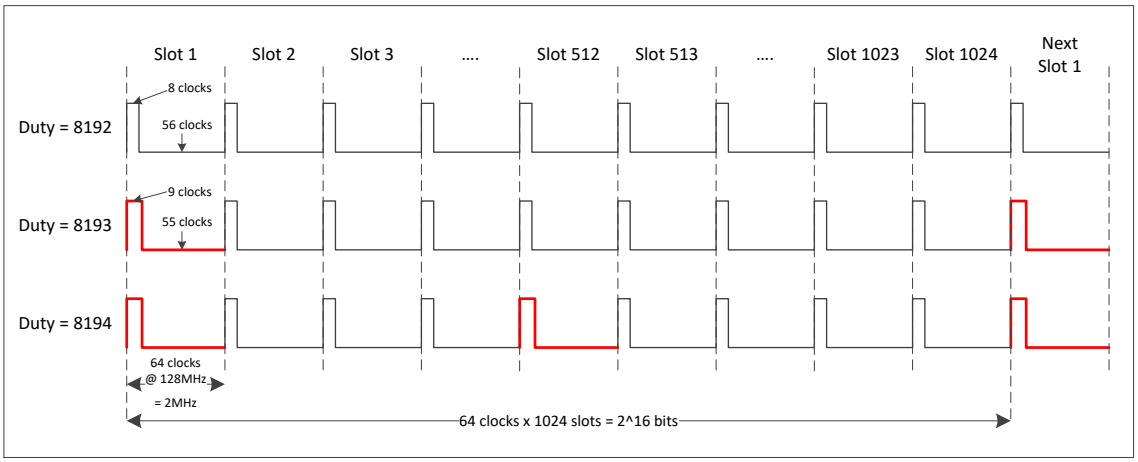


Figure 14-2 PWM Pulse Spreading

The pulse spreading algorithm first finds what is referred to as the main duty cycle of each time slot. This is found by dividing the duty cycle by 1024. The residue, or remainder of the duty cycle divided by 1024 is then found. This residue number will be evenly spread across the 1024 timeslots. Following are some examples based upon Figure 14.2

Duty Cycle = 8192: Main Duty = 8192 / 1024 = 8 cycles per slot.
Residue = 8192 % 1024 = 0. So no additional cycles are required.

Duty Cycle = 8193: Main Duty = 8193 / 1024 = 8 cycles per slot.
Residue = 8193 % 1024 = 1. One timeslot needs to have 1 additional clock added

Duty Cycle = 8194: Main Duty = 8194 / 1024 = 8 cycles per slot.
Residue = 8194 % 1024 = 2. Two timeslots needs to have 1 additional clock added

14.2 – Register description

The DS4835/36 PWMs share a set of registers with the DS4835/36 Delta Sigma DAC (DS-DAC) peripherals. For details on the DS-DAC operation, please refer to the Delta Sigma DAC Section of this user's guide. All 8 of the PWMs can be configured using only 2 peripheral registers, DPCN and DPDAT.

Note: The DPCN and DPDAT registers are located in Module 6. Register bits in Module 6 cannot be individually written. Instead the entire register must be written.

DAC PWM Control Register (DPCN)

Bit	7:6	5:4	3:0
Name	RESERVED	REGSEL	CHSEL
Reset	0	0	0
Access	r	rw	rw

BIT	NAME	DESCRIPTION
7:6	RESERVED	**Reserved**. The user should write 0 to these bits.
5:4	REGSEL	Register Select. This field selects the register for DPDATA register Read/Write. REG_SEL[1:0] – Register 00 – DAC PWM Configuration 01 – DAC PWM Duty Cycle 11 – Reserved
3:0	CHSEL	Index for DPDATA register. Selects which channel read/writes to DPDATA accesses. These bits auto increment when DPDATA is accessed.

DAC PWM Configuration (DPDATA when DPCN.REGSEL = 00)

When REGSEL = 0, writing to the DPDATA register writes to one of the 8 PWM configuration registers. The configuration register written to is selected by the CHSEL[3:0] bits.

Bit	15	14	13:7	6	5	4	3:0
Name	EN	MODE	RESERVED	VCC_COMP	TXD_ZERO	INV_DSDAC	RESERVED
Reset	0	0	0	0	0	0	0
Access	rw	rw	r	rw	rw	rw	r

BIT	NAME	DESCRIPTION
15	EN	Setting to 1 enabled the corresponding DAC/PWM
14	MODE	0: DS DAC Mode is selected 1: PWM Mode is selected
13:7	RESERVED	Reserved. The user should write 0 to these bits.
6	VCC_COMP	This bit has no effect on PWM operation
5:1	RESERVED	Reserved. The user should write 0 to these bits.
0	INV_PWM	When this bit is set, the PWM output will be inverted.

DAC PWM Duty Cycle Register (DPDUTY when DCPN.REGSEL = 1)

When REGSEL = 1, writing to the DPDATA register writes to one of the 8 PWM DPDUTY Registers. The register written to is selected by the CHSEL[3:0] bits.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	DPDUTY[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

This register is used to set the duty cycle of the PWM.

SECTION 15 – DELTA SIGMA DAC

The DS4835/36 provides 12 first order Delta-Sigma DACs (DS-DAC). The DS-DACs have 16-bit resolution and can operate at a maximum frequency of 8 MHz. This modulator output translates to a DC voltage when it's fed to an external passive low pass filter as shown in figure 15.1. For best operation, each pole of the filter should have the resistor and capacitor selected so that RC > 102.2us.

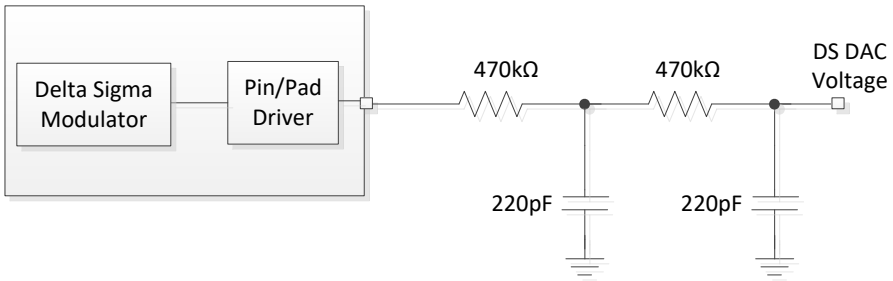


Figure 15.1: Delta Sigma DAC with external low pass filter

Note: Between the DS4835/36 and the RC filter, the DS-DAC output is a switching digital signal. Precaution should be taken in layout to prevent this signal from corrupting adjacent signals.

15.1 – Detailed Description

The DS-DAC output is a stream of pulses of equal width such that the average density of the pulses corresponds to the digital input value as shown in Figure 15.2. The output stream is then passed through a low-pass filter to produce an analog voltage.

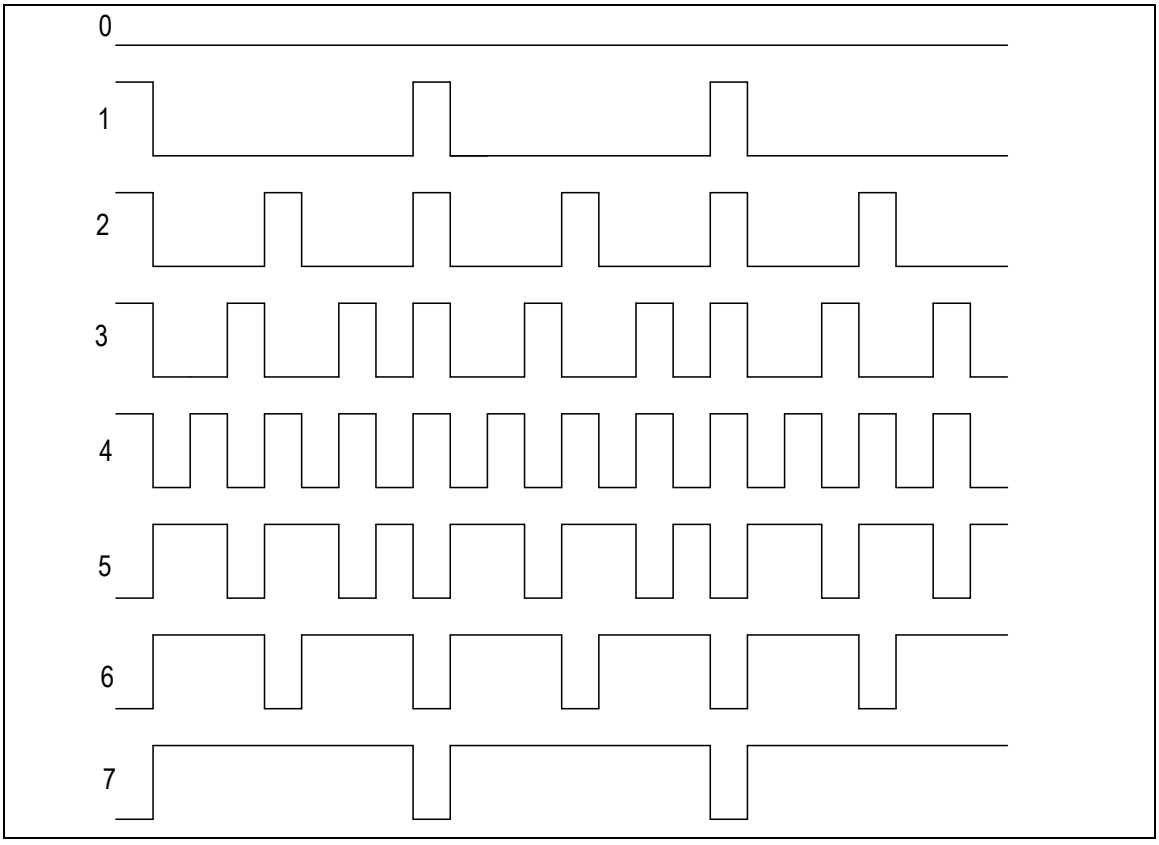


Figure 15-2 Delta Sigma DAC Outputs (example showing a 3-bit DS-DAC).

The DS_DACS are clocked by the DS4835/36's internal 16 MHz clock source. As can be seen in Figure 15.2, the fastest switching frequency will be achieved when the DS-DAC is set for a 50% duty cycle. This will result in an 8 MHz output.

The DS-DAC outputs can be set to zero when the output from PLA 1 is asserted. This provides mode to quickly shutdown the DS_DAC in the event of a TX Disable or Fault signal. This functionality is enabled by setting the TXD_ZERO bit to '1'.

The DS-DACs has two mode of the operation, normal and voltage compensated.

15.1.1 – Normal Mode

In this mode, the DS-DAC resolution can be selected by writing to DACFS[15:0] register. The DS-DAC output varies by changing the DPDUTY[15:0] register. The duty cycle is the ratio of DPDUTY[15:0] / DACFS[15:0]. Note that there are different ways to get the same duty cycle. For example, to get a 50% duty cycle, some of the possible combinations are:

DPDUTY = 4000h and DACFS = 8000h
DPDUTY = 3 and DACFS = 6
DPDUTY = 200 and DACFS = 400

To calculate the DS-DAC output voltage, the following formula can be used.

$Vout = Vcc \times {}^{DPDUTY}/_{DACFS}$ where DPDUTY <= DACFS

15.1.2 – Vcc Compensation

During normal operation of a delta-sigma DAC, the output voltage will vary as the VCC voltage changes. This is because the output voltage is determined by the following equation

$$Vout = Vcc\ x\ DutyCycle$$

The DS4835/36 DS-DAC controller was designed with an option to compensate for changes in the VCC supply. This allows for a stable output voltage even when the supply voltage changes. To accomplish this, the DS-DAC controller takes the results from ADC readings of VCC and dynamically adjusts the duty cycle. VCC conversions must be performed using ADC Config[23]. Refer to Table 6-2 in the ADC section for more details.

To set the desired output voltage, the DPDUTY register must be set using the following formula:

$$DPDUTY = \frac{Vtarget\ x\ 2^{16}}{4\ x\ VREF}$$

Where

- Vtarget is the target voltage. Example 1.5V.
- VREF is the ADC Reference voltage, for example 2.4V.
- And 2^{16} is the ADC resolution. In this example, unipolar conversions are being done for 16-bit resolution.
- The 4 in the denominator is from the divide by 4 resistor divider implemented internally for the VCC measurement.

To guarantee this VCC Compensation operation will work as the input voltage changes, the target voltage (Vtarget) must be less than the minimum possible VCC supply voltage. Note, in this mode of operation, the user should not write to the DACFS register.

15.2 – Register description

The DS4835/36 DS-DACs share a set of registers with the DS4835/36 PWM peripherals. For details on the PWM operation, please refer to the PWM section of this user's guide. All 12 of the DS-DACs can be configured using only 2 peripheral registers, DPCN and DPDAT.

Note: The DPCN and DPDAT registers are located in Module 6. Register bits in Module 6 cannot be individually written. Instead the entire register must be written.

DAC PWM Control Register (DPCN)

Bit	7:6	5:4	3:0
Name	RESERVED	REGSEL	CHSEL
Reset	0	0	0
Access	r	rw	rw

BIT	NAME	DESCRIPTION
7:6	RESERVED	**Reserved**. The user should write 0 to these bits.
5:4	REGSEL	Register Select. This field selects the register for DATA register Read/Write. REG_SEL[1:0] – Register 00 – DAC PWM Configuration 01 – DAC PWM Duty Cycle 10 – DAC Full Scale 11 – Reserved
3:0	CHSEL	Index for DPDATA register. Selects which channel read/writes to DPDATA accesses. These bits auto increment when DPDATA is accessed.

DAC PWM Configuration (DPDATA when DPCN.REGSEL = 00)

When REGSEL = 0, writing to the DPDATA register writes to one of the 12 DS-DAC configuration registers. The configuration register written to is selected by the CHSEL[3:0] bits.

Bit	15	14	13:7	6	5	4	3:0
Name	EN	MODE	RESERVED	VCC_COMP	TXD_ZERO	INV_DSDAC	RESERVED
Reset	0	0	0	0	0	0	0
Access	rw	rw	r	rw	rw	rw	r

BIT	NAME	DESCRIPTION
15	EN	Setting to 1 enabled the corresponding DAC/PWM
14	MODE	0: DS DAC Mode is selected 1: PWM Mode is selected
13:7	RESERVED	Reserved. The user should write 0 to these bits.
6	VCC_COMP	1: VCC Compensated mode of operation. 0: Normal mode of operation This bit only effects DS-DAC operation, not PWM.
5	TXD_ZERO	When this bit is set, the DS-DAC output will go to 0 when PLA 1 is asserted.
4	INV_DSDAC	When this bit is set, the DS-DAC output will be inverted.
3:0	RESERVED	Reserved. The user should write 0 to these bits.

DAC PWM Duty Cycle Register (DPDUTY when DCPN.REGSEL = 1)

When REGSEL = 1, writing to the DPDATA register writes to one of the 12 DS-DAC DPDUTY registers. The register written to is selected by the CHSEL[3:0] bits.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	DPDUTY[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

This register is used to set the duty cycle of the DS-DAC when operating in normal mode. In VCC compensated mode, this register is used to set the desired target voltage.

DAC Full-Scale register (DACFS when DCPN.REGSEL = 2)

When REGSEL = 2, writing to the DPDATA register writes to one of the 12 DS-DAC DACFS registers. The register written to is selected by the CHSEL[3:0] bits.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	DACFS[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

This register is used to set the full-scale value for the DS-DAC counter. This is written by the user when operating in normal mode. When operating in VCC Compensated mode, this register should not be written because it will be dynamically adjusted by the DS-DAC controller.

SECTION 16 – CURRENT DAC

The DS4835/36 provides 4 current DACs (IDAC) with 12-bits of resolution. These are designed for biasing lasers, including high-power tunable lasers.

16.1- Detailed Description

Each IDAC can be configured to have a full-scale range of 2mA to 200mA. The IDAC output pins can be connected in parallel to obtain additional current or higher resolution. A block diagram for an IDAC is shown in Figure 16.1.

VCCDAC must be applied for the IDACs to work properly. Refer to the DS4835/36's Datasheet for information regarding the range of VCCDAC. It should be noted that the feedback for DCDC1 is shared with the VCCDAC pin. This was designed so DCDC1 can be used as a buck converter to supply power to VCCDAC.

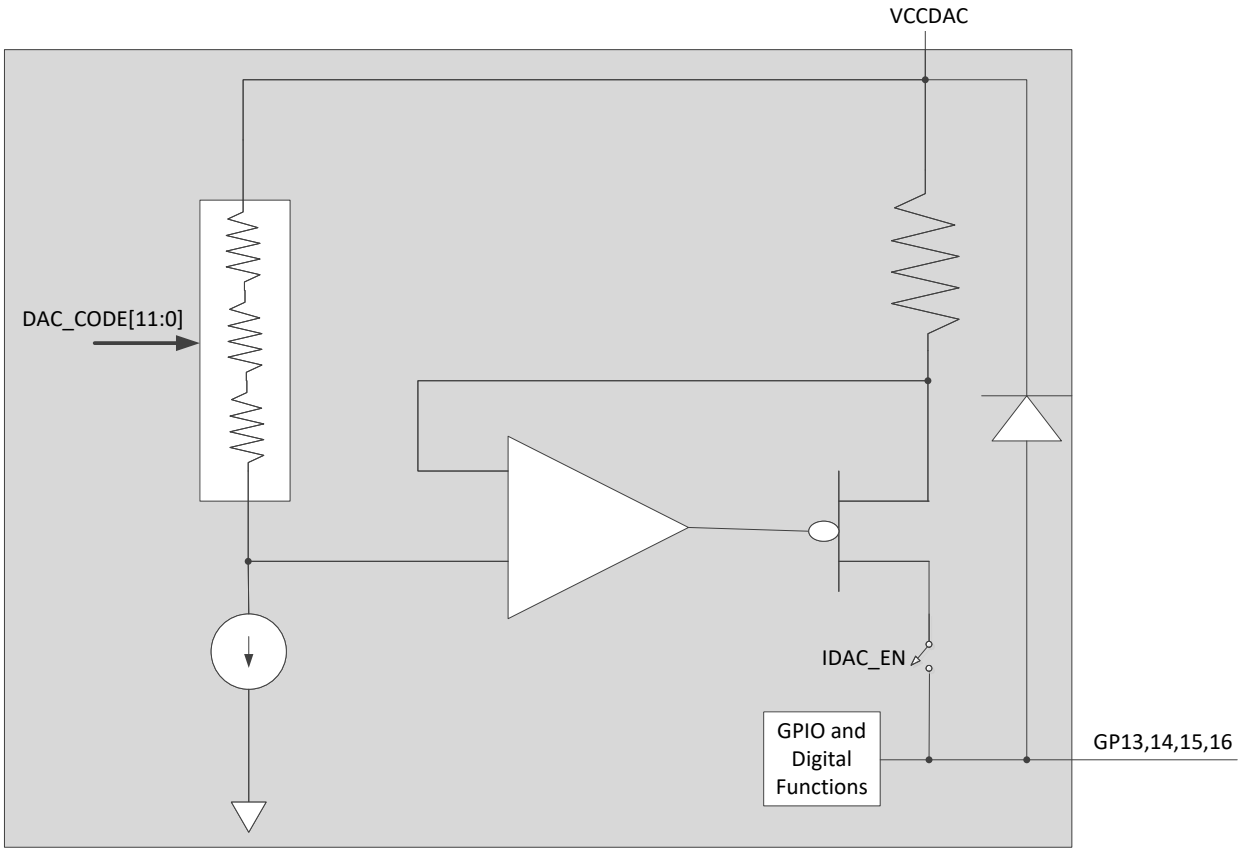


Figure 16.1: IDAC Block Diagram

Note: There is a diode connected internally between each IDAC output pin and VCCDAC. This diode will not have any impact on the operation of IDAC. But, this diode may impact the GPIO and digital functionality of these pins if they are not being used as IDACs. This diode will clamp the input and output level of digital functions to a diode drop above VCCDAC. If VCCDAC is less then VCC, this will prevent proper operation of digital functions and GPIO on these pins. Following are some recommendations.

- If VCCDAC = VCC, these pins can be used for IDAC, GPIO, or any other digital function.
- If VCCDAC < VCC, these pins should be used only as IDAC or ADC pins that have inputs that will not exceed VCCDAC.
- If used for ADC, these pins have a higher leakage current than ADC channels on other pins. These pins should only be used to monitor ADC channels with a low source impedance.

16.2- IDAC Register Descriptions

Each of the 4 IDACs has two dedicated registers, IDCNn and IDCDn, where n = 1 to 4. These registers are used to configure the IDAC and change the output current.

IDAC Control Register (IDCNn)

Bit	15:4	3:2	1	0
Name	RESERVED	RANGE_SEL[1:0]	EN_2MA	IDAC_EN
Reset	0	0	0	0
Access	r	rw	rw	rw

BIT	NAME	DESCRIPTION
15:4	RESERVED	**Reserved**. The user should write 0 to these bits.
3:2	RANGE_SEL	**IDAC Range Select.** Selects the IDAC full scale range when EN_2MA is set to 0. RANGE_SEL[1:0] / Full Scale Current (mA) 00 / 25 01 / 50 10 / 100 11 / 200
1	EN_2MA	**Enable 2mA Range:** Setting this bit will enable the 2mA current range of the IDAC.
0	IDAC_EN	**IDAC ENABLE.** Setting this bit will enable the current DAC function.

IDAC Data Register (IDCDn)

This register is used to change the IDAC output.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED				IDAC DATA[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

SECTION 17 – SAMPLE AND HOLD

The DS4835/36 has two independent, but identical, Sample and Hold (S/H) differential channels. Sample and Hold 0 (S/H0) is on GP20-GP21 and Sample and Hold 1 (S/H1) is on GP26-GP27. The trigger for each S/H can be selected from a list of available pins.

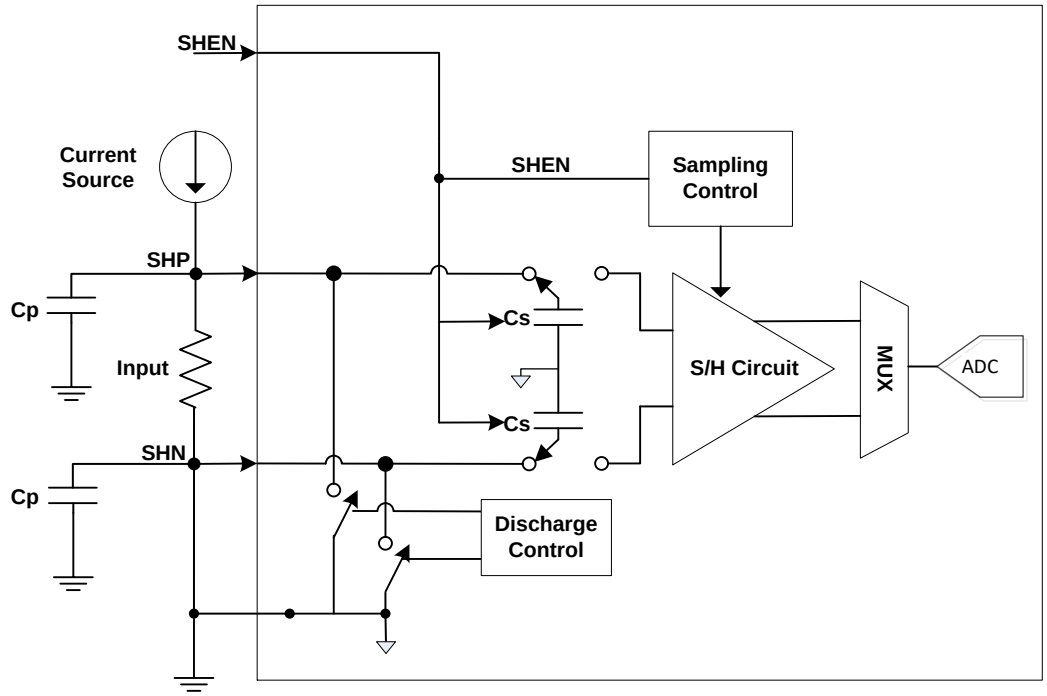


Figure 17-1: Sample and Hold Functional Block Diagram

17.1 – Detailed Description

As shown in figure 17-1, 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 discharge circuit to discharge parasitic capacitance on the input node and the sample capacitor before it starts sampling. The input voltage is sampled using approximately 3pF capacitors on the positive and negative inputs. The negative input pin is used to reduce ground offset and noise. The capacitors are connected to the input pins when the sample trigger signal SHEN is high. During the high period of sample pulse, the S/H performs sampling which ends at the negative edge of the SHEN sample pulse.

17.1.1 – 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 the voltage stored on the sampling capacitors are converted by the ADC controller. See Figure 17-2 for Sample and Hold Timings. Each S/H can be independently enabled by setting their respective enable bit in the Sample and Hold Control Register (SHCN). The sample and hold have two modes of operation "Single Mode" and "Dual Mode".

17.1.2 – Pin Capacitance Discharge

Before the S/H circuitry starts sampling, there is an option to discharge the 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 will discharge the pin or PCB capacitance for the sample and hold channels. The discharge 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. Pin discharge timing is shown in Figure 17-2.

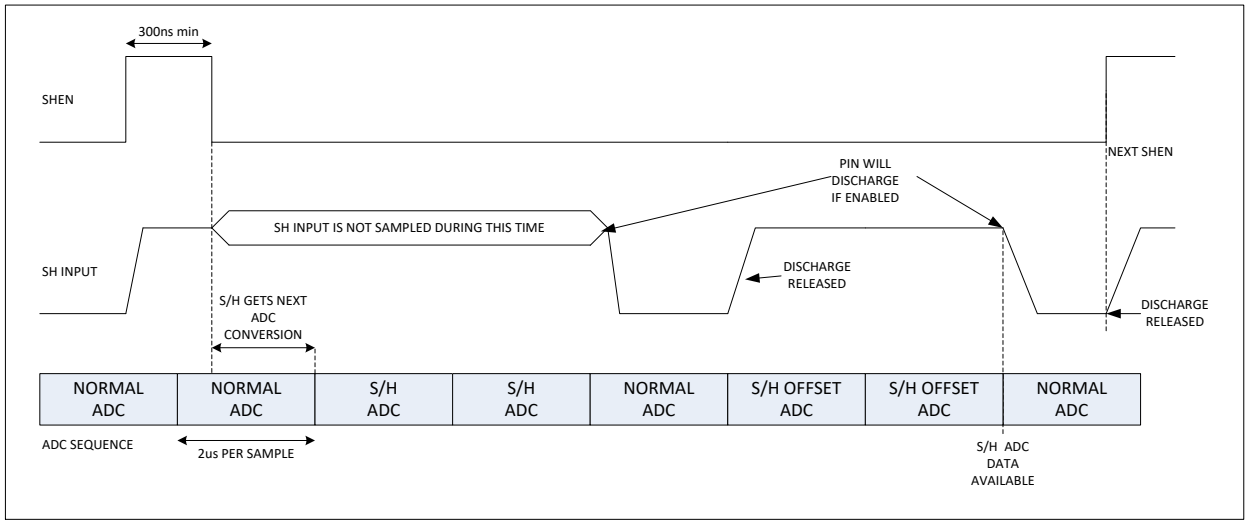


Figure 17-2 – Sample and Hold Conversion Timings with Average = 2, 0us of Extended Acquisition, and Discharge Enabled

17.1.3 – ADC Sampling and Averaging

As described in the ADC section, when a S/H channel is ready for an ADC conversion, the conversion will be given top priority by the ADC and will occur after the current conversion in progress has completed. If both S/H channels need to be converted, they will be given these conversions will be done back-to back.

Averaging of the ADC results can also be implemented for the S/H channels. When averaging is selected, the ADC will make multiple conversions of the same ADC sample and average the results. When averaging is selected, all S/H conversions will be completed back-to-back, there will be no normal ADC sequenced conversions occurring during this time.

At the completion of an S/H ADC conversion, a flag bit will be set in the ADCST register. The SH0I bit will be set at the completion of the ADC conversion of S/H 0, and SH1I will be set at the completion of S/H 1. The setting of these flags can cause an interrupt if enabled.

17.1.4 – Sample and Hold Offset Cancelation

Sample and hold conversions may have an offset value that is large enough to corrupt the conversion of low-level signals. The DS4835/36 sample and hold block can cancel out this offset. Each S/H conversion will automatically be followed by a conversion of S/H offset with the S/H inputs internally grounded. If enabled with the SH_OFF_CANC bit, this offset measurement will be subtracted from the S/H result. It is recommended that offset cancellation be enabled to improve the measurement accuracy of S/H conversions. There are no disadvantages to using this feature.

17.1.5 – Sample and Hold Extended Acquisition

The sample acquisition time for a sample and hold can be selected using the TEMP_ACQ bits in the TEMPCN register. If extended acquisition is selected with these bits, this acquisition time will apply to both sample and hold channels as well as internal and external temperature conversions. Also, if extended acquisition is enabled and the internal ADC buffer is enabled with the MUXBUF_EN bit, this buffer will be applied to the sample and hold and temperature channels. It is recommended to use at least 0.5us of extended acquisition for S/H channels to improve the S/H performance.

17.1.6 – Single and Dual Mode Operation

During the single mode operation, a single SHEN signal is used to trigger both S/H circuits. When operating in single mode, the SHEN0 signal will be used.

Dual mode operation is selected when SH_DUAL bit in the SHCN register is set to '1'. In this mode of operation, both the S/H circuits work independently. Each sample and hold will have separate triggers. The SHEN0 and SHEN1 provide sample pulses to S/H0 and S/H1 respectively.

17.1.7 – Sample and Hold Data Reading

Each S/H has defined data buffer locations where the ADC controller stores the S/H results following the ADC conversion. These are SH0_DATA and SH1_DATA. The S/H data is sampled in bipolar mode, so the results will be signed.

17.2 – Sample and Hold Register Descriptions

Both S/H circuits are configured by the Sample and Hold Config (SHCN) register. The ADCST register contains interrupt flags for the two S/H circuits, and S/H data is available in the SH0_DATA and SH1_DATA registers.

Sample and Hold Data (SH0_DATA and SH1_DATA)

These two registers are used to readback the values of an internal or external temperature conversion. These results need to have ADC scale and offset applied, refer to the Applying Scale and Offset to ADC Results section for more details. The following table shows the bit weighting for the sample and hold result registers.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S/H Value	S	2^{14}	2^{13}	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}

Sample and Hold Interrupt flag

See the ADST register description for Sample and Hold interrupts flags and interrupt enable bits.

Sample and Hold Control Register (SHCN)

Bit	15:14	13:12	11	10	9	8	7:6	5	4	3:2	1	0
Name	SHEN1_SEL	SHEN0_SEL	EN_CLIP	SH_DUAL	PIN_DIS1	PIN_DIS0	SH1_AVG	-	SH1_EN	SH0_AVG	SH_OFF_CANC	SH0_EN
Reset	0	0	0	0	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:14	SHEN1_SEL	**SHEN1 Pin Select.** Selects which pin will be used for SHEN1. **SHEN1_SEL[1:0]** / **SHEN1 PIN** 00 / GP00 01 / GP01 10 / GP02 11 / GP03
13:12	SHEN0_SEL	**SHEN0 Pin Select.** Selects which pin will be used for SHEN0. **SHEN0_SEL[1:0]** / **SHEN0 PIN** 00 / GP05 01 / GP15 10 / GP16 11 / GP17
11	EN_CLIP	**SHEN Clip Enable**. Setting this bit will allow clipping of the SHEN pulse internally. When this option is enabled the SH circuit uses only the first 280ns of SHEN pulse internally for sampling and then goes into hold mode. This is useful in burst mode system where the falling edge of SHEN may not be well defined. The conversion will start only after the external SHEN pulse goes low. For example, if the external SHEN pulse is 500ns and this bit is enabled, then the S/H will sample the input for 280ns from the rising edge. The S/H will then be in a hold mode until SHEN goes low.
10	SH_DUAL	**Sample and Hold Dual Mode.** Setting this bit to '1' configures "Dual Mode" Sample and Hold operation. In dual mode, both sample and holds act independently and use independent sample trigger input signals. SHEN0 is the sample trigger for S/H 0. SHEN1 is the sample trigger for S/H 1. When this bit is set to '0', single mode operation is selected. In single mode operation both sample and hold circuits are triggered by the SHEN0 signal.
9	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 the capacitance at the SHP1 and SHN1 pins by driving these pins to GND after the completion of the S/H 1 ADC conversion.
8	PIN_DIS0	**Pin Discharge Enable 0.** Setting this bit to '1' enables the pin discharge function for Sample and Hold 0. The discharge function discharges the capacitance at the SHP0 and SHN0 pins by driving these pins to GND after the completion of the S/H 0 ADC conversion.
7:6	SH1_AVG	**SHEN1 Average.** Selects how many S/H1 conversions to average. **SH1_AVG[1:0]** / **Samples** 00 / 1 01 / 2 10 / 4 11 / 8
5	Reserved	**Reserved**. The user should write 0 to this bit.
4	SH1_EN	**Sample and Hold 1 Enable.** Setting this bit to '1' enables Sample and Hold 1 operation.
3:2	SH0_AVG	**SHEN0 Average.** Selects how many S/H0 conversions to average. **SH0_AVG[1:0]** / **Samples** 00 / 1 01 / 2 10 / 4 11 / 8
1	SH_OFF_CANC	**Sample and Hold Offset Cancelation.** Setting this bit to 1 will enable offset cancellation for the sample and hold channels. In this mode, the ADC cancels the internal offset of the sample and hold block by doing an additional conversion internally.
0	SH0_EN	**Sample and Hold 0 Enable.** Setting this bit to '1' enables Sample and Hold 0 operation.

SECTION 18 – I²C-COMPATIBLE SLAVE INTERFACE

The DS4835/36 provides an I²C compatible slave controller that allows communication with a host device and supports 4 user programmable slave addresses. The DS4835/36 I²C slave interface also has a dedicated 8-byte transmit page for each slave address and an 8-byte receive FIFO which is shared between all 4 slave addresses. This can support 400 kHz I²C communication without clock stretching and reduces software overhead. The DS4835/36 I²C slave controller can bootload new firmware into flash using the I²C slave interface. This interface can be setup to provide system interrupts after each I²C event. Figure 18-1 shows the basic operation of the I²C slave controller. The blocks in Figure 18-1 that are shaded are shown in more detail in Figure 18-3.

SDA and SCL are open-drain only, +5V tolerant pins.

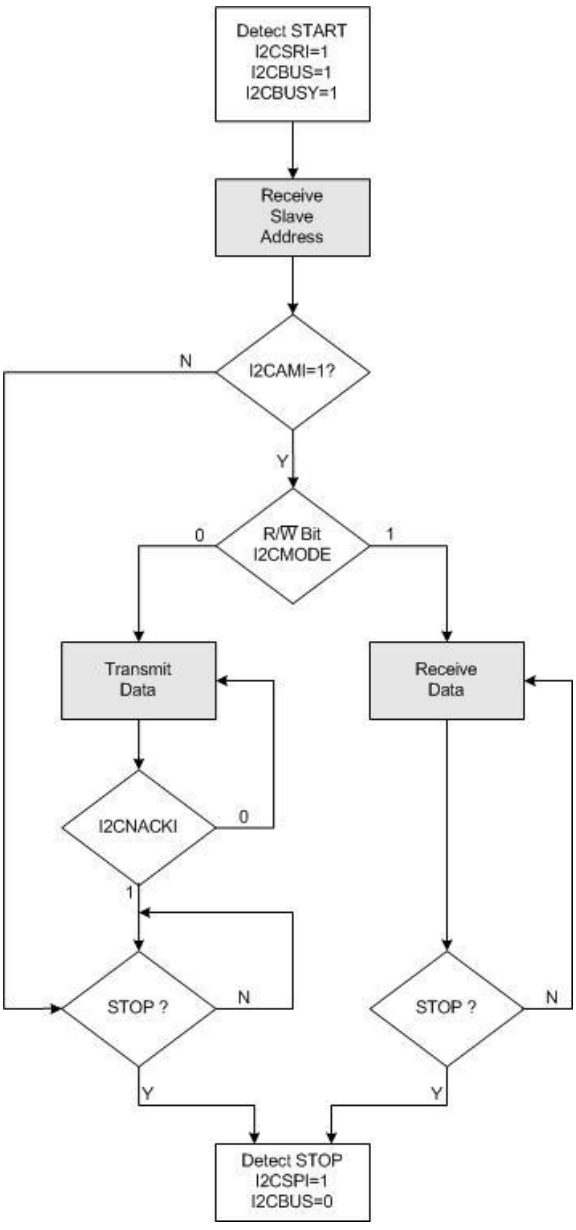


Figure 18-1 Slave I²C Flow

18.1 Detailed Description

18.1.1 – Default Operation

The I²C slave controller is enabled (I2CCN_S.I2CEN=1) by default. If the I²C slave controller is enabled, the DS4835/36 I²C bootloader can operate. This allows bootloading of blank devices without any setup of the I²C slave controller. Prior to the I²C slave controller being used for normal data communication, the I²C SFRs should be configured for necessary I²C communication. These configurations include setting an I²C slave address and enabling the slave controller to generate interrupts during I²C events. This controller can also operate as an SMBUS slave.

18.1.2 – RX FIFO and TX Page

The DS4835/36 I²C controller has 4 user programmable slave addresses. The user can program slave addresses by writing to I2CSLA_S, I2CSLA2_S, I2CSLA3_S, and I2CSLA4_S registers. The I2CSLA_S is by default enabled and other slave addresses are by default disabled. These other slave addresses can be enabled by writing '1' in the ADDR2EN, ADDR3EN, and ADDR4EN bits defined in I2CCN_S register.

The DS4835/36 I²C controller has an 8-byte receive FIFO which is shared among the enabled slave addresses. The receive FIFO is controlled using the I2CRXFIE (I2C Receive FIFO Interrupt Enable) and I2CRXFST (I2C Receive FIFO Status Flags) registers and is read from the I2CBUF_S register. Refer to the individual bit description in the I2C Slave Controller Register Description section.

The I²C controller has 4 Transmit (TX) pages, one dedicated to each slave address. Each TX page size is 4 words (16-bits). The controller selects one TX page according to 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 written by writing data to the I2CBUF_S register. The I²C transmission is controlled using the I2CTXFIE (I2C Transmit FIFO Interrupt Enable) and I2CTXFST (I2C Transmit FIFO Status) registers. Refer to the individual bit description in the I²C Slave Controller Register Description section.

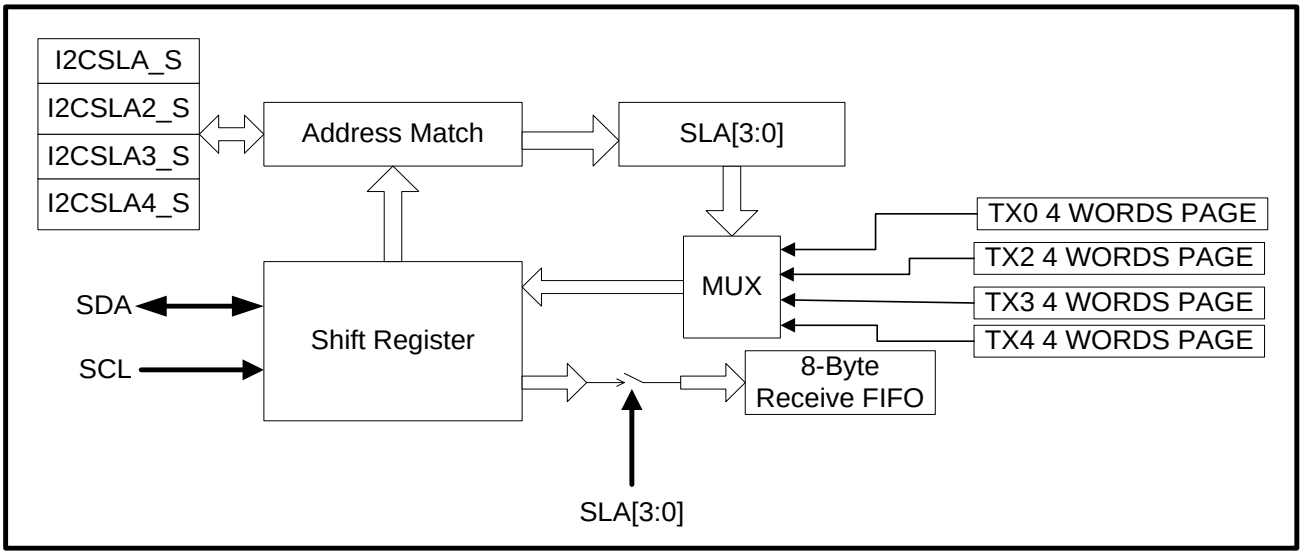


Figure 18-2 Slave I²C Block Diagram with RX FIFO and TX Pages

18.1.3 – Slave Addresses

Prior to communication, an I²C slave address may need to be selected. By default, the I²C slave controller normally responds to two slave addresses. The I²C 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 a different slave address to the I2CSLA_S register. The address contained in the I2CSLA_S register is the address with the R/W\ bit. If an address other than 36h is desired, the I2CSLA_S register can be programmed with this new address.

The DS4835/36 has 3 more user programmable slave addresses which can be programmed 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 ADDR2EN, ADDR3EN, ADDR4EN bits respectively which are located in the I2CCN_S register.

The I²C slave controller supports the General Call Address, which is 00h with I2CSLA_S slave register. This feature can be enabled by setting the I2CCN_S. I2CGCEN bit to a 1.

18.1.4 – I²C Start Detection

The I²C Slave Controller always monitors the I²C bus for an I²C start, which is a high to low transition on SDA while SCL is held high. If an I²C start (or restart) condition is detected, the I²C slave controller sets the I2CSRI bit in the I2CST_S register, which can cause an interrupt if enabled. The detection of a start brings the I²C controller out of its idle state. Following a start, the I²C controller begins to monitor data on the I²C bus and the I2CBUSY bit is set to a 1. The I2CBUS bit is also set to a 1 to indicate that the I²C bus is currently busy.

18.1.5 – I²C Stop Detection

The I²C Slave Controller also always monitors the I²C bus for an I²C stop, which is a low to high transition on SDA while SCL is held high. If an I²C stop condition is detected, the I²C 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 I²C bus is no longer busy.

18.1.6 – Slave Address Matching

Following an I²C start or restart, the I²C slave controller knows that the next byte of data to be transmitted by the host should be a slave address. The I²C slave automatically monitors for the slave address without any software interaction. The I²C slave controller compares the first 7 bits received to the slave address programmed in the I2CSLA_S register, and the I2CSLA2_S, I2CSLA3_S, and I2CSLA4_S if they are enabled. If the received slave address matches an enabled I²C Slave addresses, the I²C slave controller does the following steps. This is illustrated in Figure 18-3 (without RX FIFO and TX Pages) and Figure 18-4 (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 transfer mode bit (I2CMODE) with the value of the received R/W\ bit. This bit can be used by software to determine if the I²C 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.
- Sets the appropriate SLA[3:0] bits in the CUR_SLA register. These bits are set according to the following table.

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//W\ bit + ACK/NACK), the I²C slave controller enters one of the following three states.

- **Data Transmit**: The slave address matched and the R//W\ bit is '1'. The host is now expecting data from the DS4835/36. The host can now begin clocking data from the DS4835/36. The I²C slave controller retains control of the SDA line so data can be transmitted to the host.
- **Data Receive**: The slave address matched and the R//W\ bit is '0'. The host wants to write data to the I²C slave. After ACK/NACK bit, the DS4835/36 releases SDA and is ready to receive a byte of data.
- **Wait for Start/Stop**: The received slave address did not match any slave addresses. The I²C controller enters idle state and waits for the next start or stop condition.

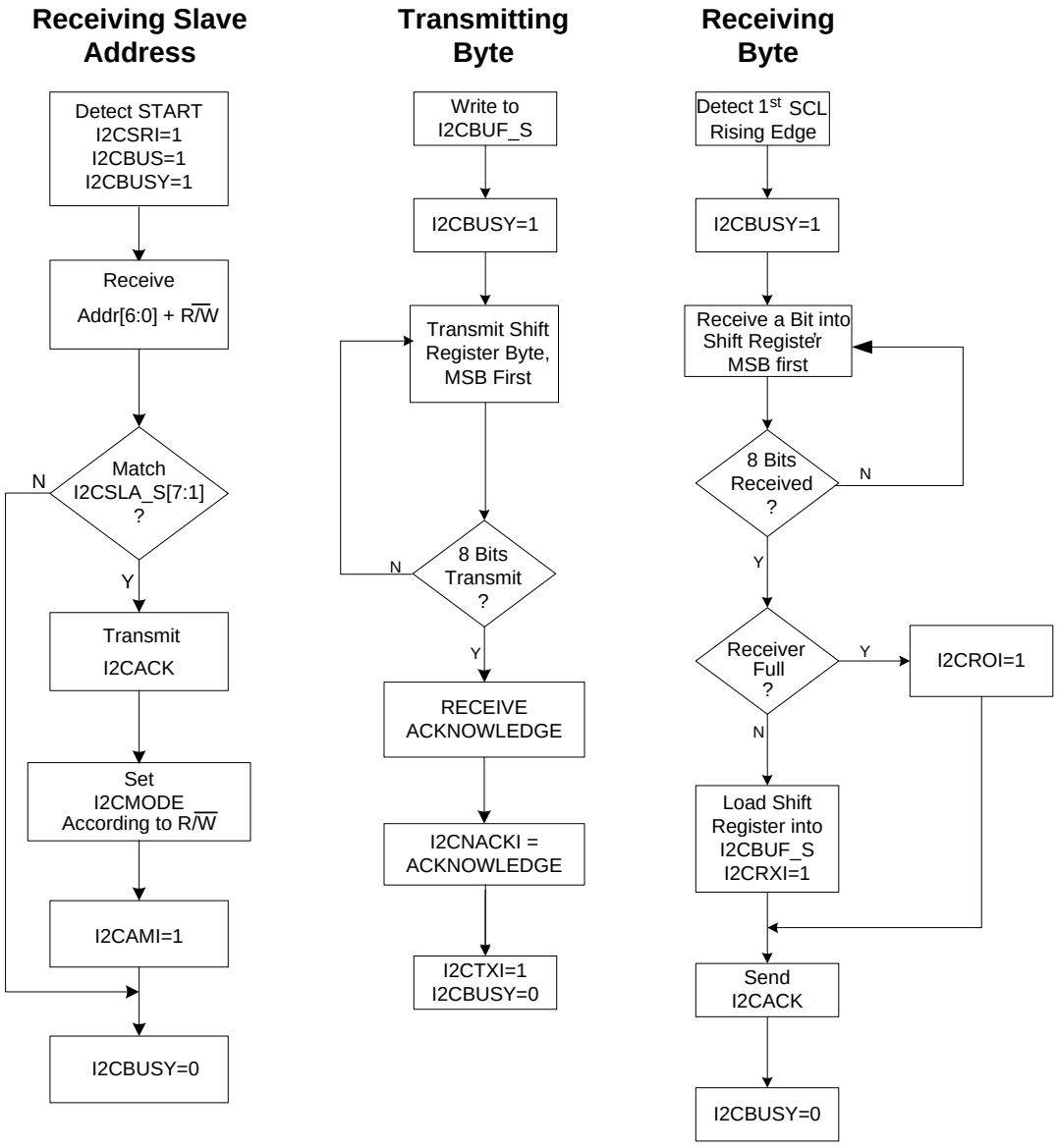


Figure 18-3 Slave I²C Data Flow

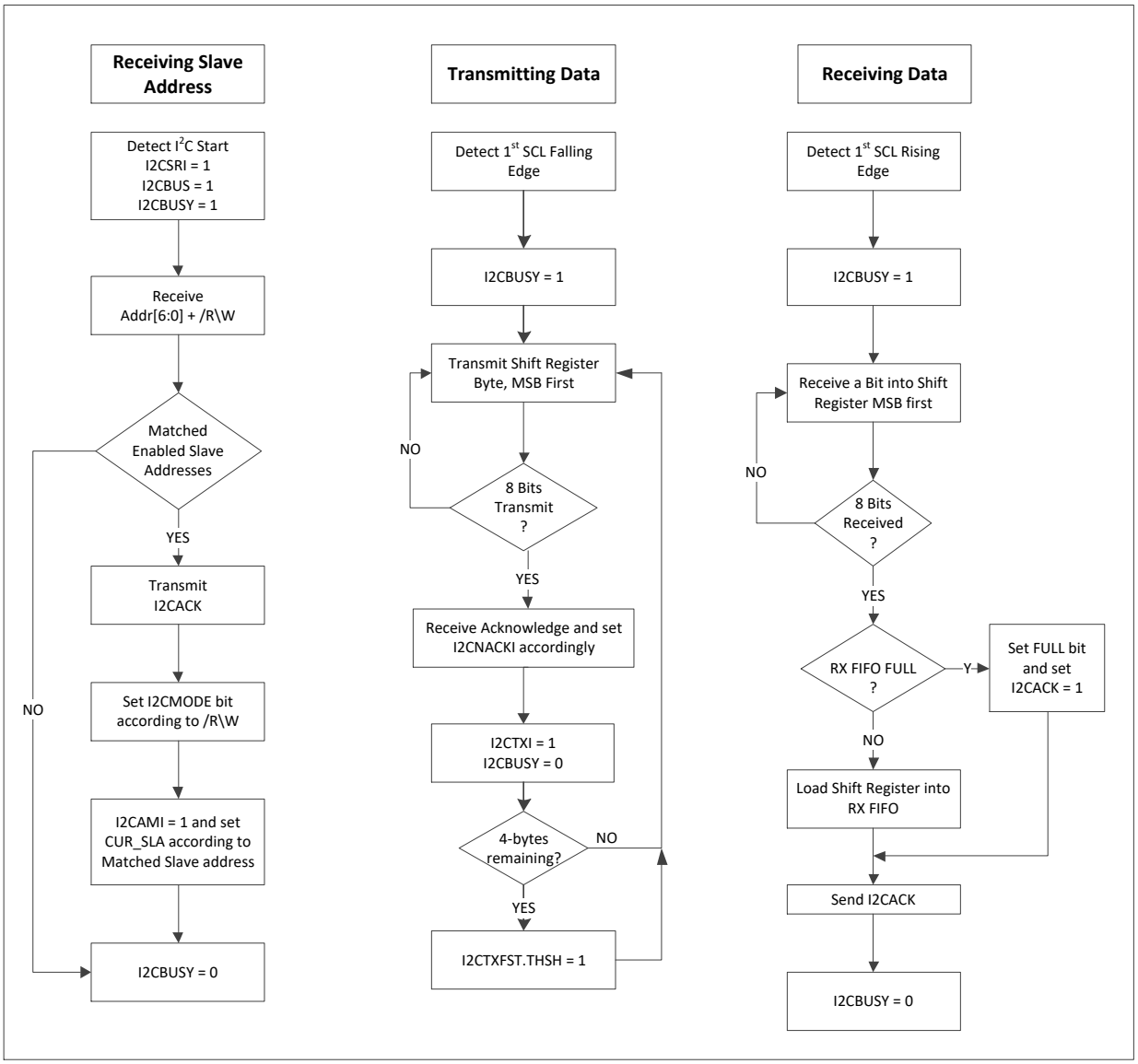


Figure 18-4 Slave I²C Data Flow using 8-Byte Transmit Page and 8-Byte Receive FIFO

18.1.7 – Transmitting Data

The DS4835/36 I²C Slave Controller enters into data transmission mode after receiving a matching slave address with the R//W\ bit set to '1. The steps of data transmission are shown in Figure 18-3. 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, any writes to I2CBUF_S will be 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.

The DS4835/36 I²C Controller has a dedicated 4-word (8-byte) TX Page for each slave. The I²C controller internally selects one of the four pages based upon the SLA[3:0] bits which are set during the slave address match event. The TXPG_EN bit in the I2CTXFIE register enables the TX PAGEs for all enabled slave addresses.

The I²C controller allows defining memory map structures in the SRAM for individual slave addresses using the MADDR, MADDR2, MADDR3, and MADDR4 registers. These register bits 11:0 are used to define start

address of the memory map structure and bit 12 is used to define memory rollover boundary between 128 and 256 bytes. The I²C controller maintains the memory address for each individual slave address in the read memory address pointer RPNTR register. Each slave address has a 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 byte boundary rollovers internally on the read memory address.

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

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

MOVE  RPNTR, #0000h           //Initialize RPNTR to current read address

MOVE  DP[0],   RPNTR          //Copy current memory address in the data pointer
MOVE  M2[0],   @DP[0]         //Write data from @DP[0] to TX PAGE via I2CBUF_S register
                              // RPNTR = RPNTR + 1 and rollover handled internally

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

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

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

Following the 8th data bit (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 I²C 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 I²C slave controller is not actively participating in the transfer of data.

The detection of an ACK by the I²C slave controller indicates that the host wants to receive another byte of data. The I²C 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 I²C slave controller releases control of SDA following the reception of NACK bit. After the NACK, the slave controller enters idle state and monitors the I²C bus for a start or stop condition.

When TX Pages are enabled, the SLA[3:0] bits in the CUR_SLA register selects one of the TX pages to use as shown in Figure 18-4. The I²C controller reads data from the selected TX page and writes this to the shift register. After the I²C controller transmits 4 bytes, it sets the threshold interrupt flag (THSH) in the I2CTXST register which can generate an interrupt, if enabled. This can be used as an indication that additional data needs to be loaded into the TX Page.

18.1.8 – Receiving Data

The I²C Slave Controller enters data reception mode after receiving a matching slave address with the R//W\ bit set to '0'. The steps of data reception are shown in Figure 18-3 and Figure 18-4. 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.

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 register.
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 a 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 the 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_S, 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 1st 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.

As shown in figure 18-4, when receive FIFO is enabled, the incoming data is copied into the FIFO and THSH bit is set when 4 bytes are copied into the FIFO by the I²C controller which can generate interrupt if enabled. The receive FIFO is read using the I2CBUF_S register.

18.1.9 – Memory Address Detection

The I²C Slave Controller provides options to detect the memory address without software intervention. The MADDR_ENx bits in CUR_SLA[7:4] has individual control bits to enable memory address detection for each slave address. 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 MPNTR (Memory address pointer) register and sets MADI bit in the I2CST2_S register, which can generate an interrupt if enabled. The MPNTR shows the current memory address of the active slave address. When memory address detection is not enabled with the FIFO, this receive byte (memory address) is copied to the receive FIFO.

18.1.10 – Clock Stretching

If the 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 can hold 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 18-5 shows the I²C slave controller clock stretching after receiving the 9th clock of a byte.

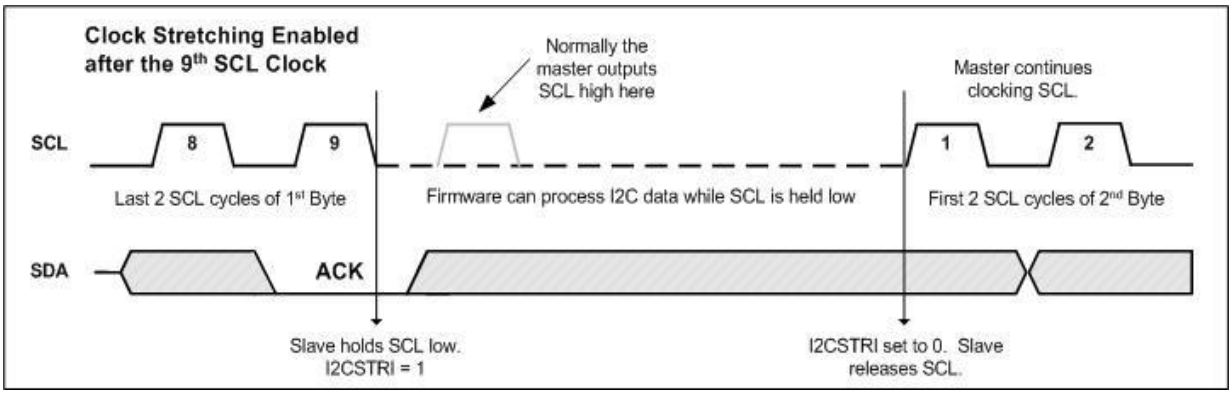


Figure 18-5 Slave I²C Clock Stretching

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.

Applications should use clock stretching if the I²C slave interrupts are not assigned the highest priority. Generally optical applications are set to respond only to interrupts from the I²C slave controller, not continuous polling of the slave I²C controller flags. 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/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 1st 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 1st 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.

18.1.11 – Resetting the I²C Slave Controller

The I²C Slave Controller can be reset by disabling the I²C Slave controller by writing '0' at I2CEN bit in the I2CCN_S register. A reset forces the I²C slave controller to release both SDA and SCL if they are being held low by the I²C slave controller. Following a reset, the I²C slave controller must be re-initialized.

Note1: When the I²C slave interface is disabled, the I²C bootloader is not available.

18.2 DS4836 I²C Slave Fast-Mode Plus (1MHz)

The DS4836 supports 1MHz (Fast-Mode Plus) I²C slave interface. The I²C slave controller has a new register I2CCN2 which has bits to control the Fast-Mode plus. This register is required for the Fast-Mode Plus operation.

To enable 1MHz operation, the user should perform the following changes in the existing DS4835/36 based code

- Set FSP_SEL to '1'. This enables the fast-mode plus operation.
- Set THD_SEL to 11b. Setting these bits, changes data transmit hold time to 3 system clocks from the 6 system clocks time (default data transmit hold time).
- Setting bit NACK_ST_EN to '1'. Setting this bit enables NACK on I2C stop after write cycle.
- Optionally, set RPU_SEL to '1'. Setting this bit allows the RPNTR register update at the I2C Start.
- Optionally, user can set IV2_SRI_EN, IV2_TXTH_EN and IV2_MADI_EN bits to '1'. This enables IV2 interrupt using which user can process above interrupt faster than other interrupts. If user enables these bits, IV2 register should be update with the start address of IV2 interrupt handler.

Refer to the I2CCN2 register bit description for more information.

18.3 I²C Slave Controller Register Description

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

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. Refer to 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 controller to respond to a general call address (address = 0000 0000). Clearing this bit to '0' disables response to the 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//W\ bit value received in the current slave address. This bit is updated by the I²C controller when a slave address match occurs.
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.

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 bit 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 any of the enabled slave address registers. 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.

I²C Slave Interrupt Enable Register (I2CIE_S)

Bit	15:10	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
Access	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 (I2CROI=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 any of the enabled slave addresses (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 the 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.

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 cycle 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 it into 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. Setting this bit to '1' by software causes an interrupt if enabled.
0	Reserved	**Reserved.** The user should not write to this bit.

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	r	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 an I²C start + two master clocks is received (SADI = 1).
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 (MADI = 1). Clearing this bit to '0' disables the memory address detection interrupt.
2:0	Reserved	**Reserved.** The user should not write to these bits.

I2C Slave Address Registers (I2CSLA_S, 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	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 I^2CSLA_S is 36h.

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 I^2C slave interface. When a match to this address is detected, the I^2C 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//W\ bit value in current value in I^2C transfer and set by hardware.

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 I^2C transfer is read from or written to this register. The I^2C 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.

Note: The I2CBUF_S register is not read back by the debugger. When viewing this register in the debugger, it will always return as 0000h.

Memory Map Address Registers (MADDR, MADDR2, MADDR3, 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 a boundary rollover to occur at memory address 256. Setting this bit to '0', enables a boundary rollover to occur at memory address 128. This is for the corresponding I2CSLAn_S slave address.
11	Reserved	**Reserved.** The user should not write to this bit.
10:8	PAGE	**PAGE:** These bits define the page of the SRAM memory map structure for I2CSLAn_S slave address.
7:0	MEM_ADDR	**Memory Address.** These bits define the start address of the SRAM memory map structure for I2CSLAn_S slave address.

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 for the I2CSLA 4_S register.
6	MADDR_EN3	**Memory Address Detection Enable 3:** Setting this bit to '1', enables the memory address detection for the I2CSLA 3_S register.
5	MADDR_EN2	**Memory Address Detection Enable 2:** Setting this bit to '1', enables the memory address detection for the I2CSLA 2_S register.
4	MADDR_EN1	**Memory Address Det1ction Enable 1:** Setting this bit to '1' enables the memory address detection for the 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.

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_PNTR	**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.

Note: The MPNTR register is not read back by the debugger. When viewing this register in the debugger, it will always return as 0000h.

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_PNTR	**Memory Address.** These bits maintain current read address of memory map structure for the current active slave address and is used in word mode.

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 PAGEs 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.

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 I^2C controller sets this bit to '1' when only 4 bytes remain in the TX page for the current active slave address.
0	Reserved	**Reserved.** The user should not write to this bit.

I2C Receive FIFO Interrupt Enable (I2CRXFIE)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	-	-	-	RXFIFO_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	RXFIFO_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 the 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 the FIFO receives 4 bytes.
0	EMPTY	**FIFO EMPTY:** Setting this bit to '1', generates an interrupt when the FIFO is empty

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.

I²C Slave Control2 Register (I2CCN2_S)

Bit	15	14	13	12	11	10	9-8	7	6	5	4	3	2	1	0
Name	-	NACK_ST_EN	RPU_SEL	-	-	FSP_SEL	THD_SEL[1:0]	-	-	-	-	-	IV2_MADI_EN	IV2_THSH_EN	IV2_SRI_EN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Access	r	rw	rw	r	r	rw	rw	r	r	r	r	r	rw	rw	rw

BIT	NAME	DESCRIPTION
15	Reserved	**Reserved.** The user should not write to this bit.
14	NACK_ST_EN	**NACK on I²C Stop Enable:** Setting this bit to '1', enables NACK on I²C stop after write operation. This sets the I2CACK bit to '1' on detecting I²C stop after write operation which NACKs the slave address until user firmware clears the I2CACK bit.
13	RPU_SEL	**RPNTR Update Select:** Setting this bit to '1', enables updating RPNTR on I²C start. Default is RPNTR update on SADI interrupt.
12:11	Reserved	**Reserved.** The user should not write to these bits.
10	FSP_SEL	**Fast Mode Plus Select.** When this bit is set to a '1', the I²C Start and Stop setup/hold timing is configured for the Fast-Mode Plus I²C operation. Default is 400KHz timing.
9-8	THD_SEL[1:0]	**Transmit Data Hold Select.** Setting these bits to 11b select the transmit data hold select timing for the Fast-Mode plus operation. Default is the Fast-Mode operation (400kHz).
7:3	Reserved	**Reserved.** The user should not write to these bits.
2	IV2_MADI_EN	**IV2 I2C Memory Address Interrupt Enable.** This bit enables the I²C memory address interrupt (MADI) for the interrupt vector2. This bit must be cleared to '0' by software once set.
1	IV2_THSH_EN	**IV2 I2C TX Threshold Interrupt Enable.** This bit enables the I²C TX Threshold interrupt (THSH) for the interrupt vector2. This bit must be cleared to '0' by software once set.
0	IV2_SRI_EN	**IV2 I2C Start Interrupt Enable.** This bit enables the I²C start interrupt for the interrupt vector2. This bit must be cleared to '0' by software once set.

Interrupt Vector2 Register (IV2)

The Interrupt Vector2 (IV2) register provides the location of the interrupt service routine which can be used for fast processing of some of I2C interrupt. It may be set to any location within program memory. The IV2 register defaults to 0000h on reset or power-up. User should define the interrupt vector2 address to this register.

SECTION 19 – I²C-COMPATIBLE MASTER INTERFACE

The DS4835/36 provides two I²C-compatible master controllers that allows the DS4835/36 to communicate with slave devices. Each master has an 8-byte buffer which reduces software overheads during multiple byte communication.

19.1 – Detailed Description

19.1.1 – 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.

Unless explicitly stated, all references to SDA and SCL in this section refer to the SDA and SCL lines of the master I²C bus, not the DS4835/36's I²C slave interface SDA and SCL pins.

The DS4835/36 master I²C controller is not intended to be used on an I²C bus that has multiple masters connected to the bus.

Each of the master I²C instances has its own dedicated MSDAn and MSCLn pins, where n = 1 or 2. Table 19.1 shows the default pins for each of the master I²C blocks.

Function	Pin
MSDA1	GP10
MSCL1	GP11
MSDA2	GP02
MSCL2	GP03

Table 19-1: Default Master I²C Pins

19.1.2 – Default Operation

The I²C master controller is disabled by default. The I²C master controller is enabled by setting the TW2WEN bits in the CNT2W register to a 1. Prior to the I²C master controller being used for communication, some software setup is required.

19.1.3 – I²C Timing and Clock Generation

In an I²C system, the master is responsible for generating the SCL signal. The DS4835/36 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 (CCK2W).

The setup and hold times and t_{BUF} timing of the master I²C controller can also be adjusted to meet the timing requirements of target slave devices. This is accomplished using the MTSPEC Register.

The CC2W and MTSPEC registers provide a flexible way to program the required timing values for I²C timing. It is up to the user to make sure I²C timing specifications are not violated. Refer to the target slave device's timing requirements prior to setting up the master I2C timing. Refer to Table 19-2 for more information on setting up the master I²C timing.

19.1.4 – Data Buffer

The DS4835/36 Master I²C Controller has an 8-byte data buffer that allows up to 8 bytes to be read or written during one transaction. This data buffer is accessed by reading or writing to the DAD2W register when the ADSEL2W bit in CNT2W is set to 0. The MADR_2W[2:0] bits are used to index each of the 8 data buffers. MADR_2W will automatically increment on each access of DAD2W.
When the master I²C is writing data, the data buffer must be loaded with all data prior to the transaction starting. When reading data, all received bytes of data are available in the data buffer at the completion of the transaction.

19.1.5 – Starting a Transaction

All I²C transactions are started by writing a slave address to the slave address register, SLA2W. When this write occurs, the master I²C controller will begin the transaction by first sending an I²C start command. This will be followed by the master I²C controller sending all 8 bits that were written to SLA2W.

Figure 19-1 shows the format used by the SLA2W register for slave address 36h in write mode. Bit 0 of SLA2W is the read/write (R/W) bit of the slave address. When bit 0 is '1', the I²C master will operate in receiver mode (data read from slave) at the completion of the slave address. When bit 0 is '0', the I²C master will operate in transmitter mode (data write to slave) at the completion of the slave address.

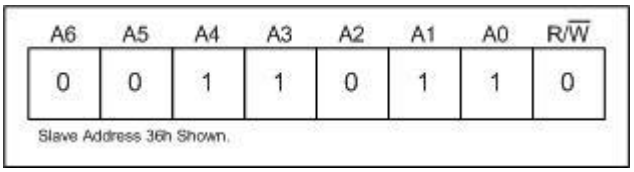


Figure 19-1: Slave Address Format

After sending the 8 bits of the slave address, the master will release SDA and allow the slave device to issue an ACK during the 9th clock cycle. If an ACK is received, then:

- If bit 0 of SLA2W is '1', the I²C master will operate in receiver mode and will begin reading data from the slave.
- If bit 0 of SLA2W is '0', the I²C master will operate in transmitter mode and will begin transmitting data write to the slave.

If the master receives a NACK, the FAIL_STAT bits will be set to 1 and the XFR_FAIL flag will be set. The master will not try to read or write any data if a NACK is received following the transmission of the slave address.

19.1.6 – Transmitting Data

The DS4835/36 I²C Master Controller enters data transmission mode after transmitting a slave address with the R/W bit set to a 0. The steps of data transmission are shown in Figure 19-2. The steps to setup a data transmission are detailed below. These must be done before writing to SLA2W and starting the transaction

If the first byte to be transmit will be an address byte, this needs to be setup by doing the following.

1) Set ADSEL2W to 1 so DAD2W will point to the MEM_ADD register.
2) Write the desired memory address to DAD2W, which will be stored to MEM_ADD
3) Set WMA to 1, which tell the controller to transmit the MEM_ADD before transmitting data bytes.

To load the data to transmit

1) Set ADSEL2W to 0 so DAD2W will point to the DataBuffer registers.
2) Set MADR_2W to 0 to index DataBuffer[0], which will be the first data byte transmitted
3) Write the first data byte to DAD2W.
4) Repeat step 3 until up to 8 bytes are loaded. MADR_2W will automatically increment after each write to DAD2W.
5) Set NB_2W to the number of data bytes to transmit – 1. So to transmit 1 byte, NB_2W will be written to 0.
6) Set EWS to 1 if the master should terminate this transaction with an I²C STOP. Clear EWS if a STOP should not be sent at the end of the transaction.
7) Clear the TWI2W bit so software will know when the transaction is complete.

Following the 8th bit of data (least significant bit) for each byte being shifted to SDA, the SDA line will be released by the DS4835/36 master controller. This allows the slave to signal an ACK or NACK during the 9th clock cycle. The DS4835/36 I²C master controller samples the acknowledge bit following the 9th SCL rising edge. If an ACK is received, the DS4835/36 I²C master controller will continue transmitting the next byte of data, or if all data has been transmit, end the transaction based upon the setting of EWS. If a NACK is received, no further bytes will be transmit, the TWI2W bit will be set, and the XFR_FAIL bit will be set with the appropriate error code in FAIL_STAT.

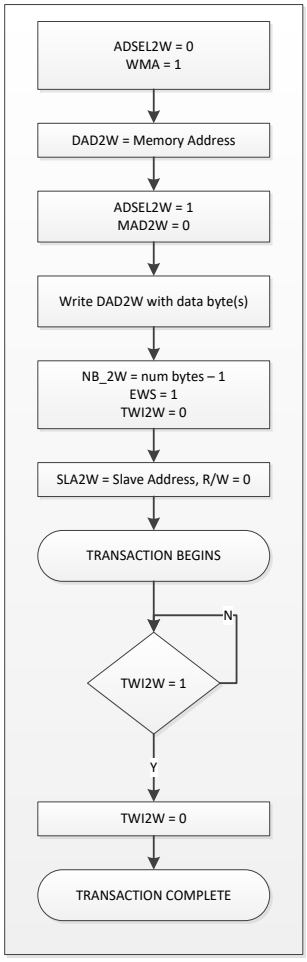


Figure 19-2: Master I²C Data Flow for Writing

19.1.7 – Direct Read

The DS4835/36 I²C Master Controller enters data reception mode after transmitting a slave address with the R/W bit set to a 1. The steps to setup for receiving data are shown below. These must be done before writing to SLA2W with the R/W bit set to 1, which starts the read transaction.

1) Set MADR_2W to 0 so DAD2W points to DataBuffer[0]. This is where the first byte of received data will be stored.
2) Set NB_2W to the number of data bytes to receive – 1. So to receive 1 byte, NB_2W will be written to 0.
3) Set EWS to 1 if the master should terminate this transaction with an I²C STOP. Clear EWS if a STOP should not be sent at the end of the transaction.
4) Clear the TWI2W bit so software will know when the transaction is complete.

After all the bytes are received, the TWI2W bit will set. The data can then be read by doing the following steps.

1) Set ADSEL2W to 0 so DAD2W points to the DataBuffers.
2) Set MADR_2W to 0 so DAD2W points to DataBuffer[0]. This is where the first byte of received data is stored.
3) Read the received byte(s) from DAD2W.

19.1.8 – I²C Random Read

To do an I²C random read, the master must first perform a memory address write to set the memory address before reading back data from the slave. The DS4835/36 can do the dummy write and readback in one transaction. The details for this are shown in Figure 19-3 and detailed below.

1) Set ADSEL2W to 1 so DAD2W will point to the MEM_ADD register.
2) Write the desired memory address to DAD2W, which will be stored to MEM_ADD
3) Set WMA to 1, which tell the controller to transmit the MEM_ADD before reading data. This is also known as a dummy write.
4) Set MADR_2W to 0 so DAD2W points to DataBuffer[0]. This is where the first byte of received data will be stored.
5) Set NB_2W to the number of data bytes to receive – 1. So, to receive 1 byte, NB_2W will be written to 0.
6) Set EWS to 1 if the master should terminate this transaction with an I²C STOP. Clear EWS if a STOP should not be sent at the end of the transaction.
7) Clear the TWI2W bit so software will know when the transaction is complete.

At the completion of the dummy write (TWI2W=1), then the steps detailed in the Receiving Data section can be followed to read the desired number of bytes from the slave.

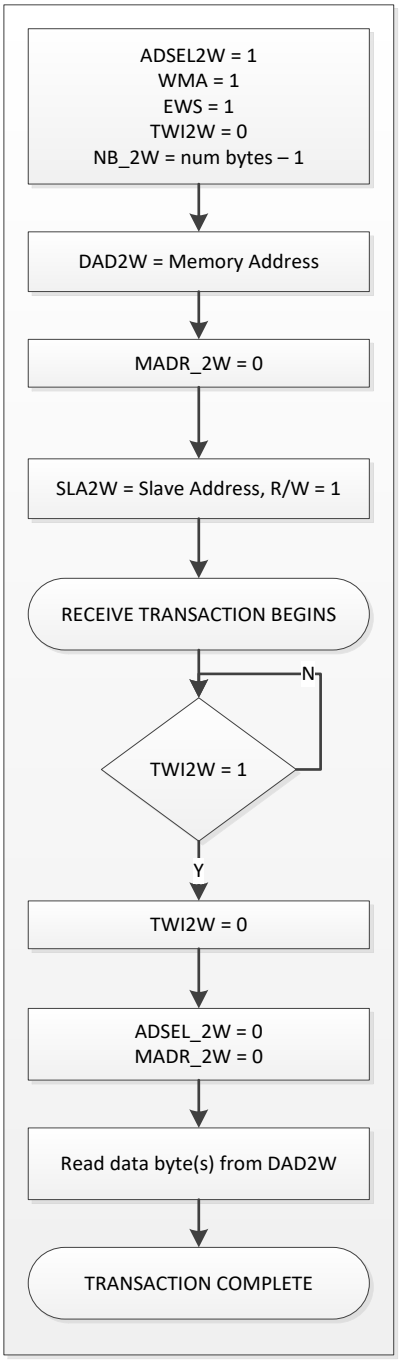


Figure 19-3: Master I²C Data Flow for a Random Read

19.1.9 – Resetting the I²C Master Controller

The I²C master controller can be reset by disabling the I²C master controller by writing '0' at TW2WEN = 0 in the CNT2W 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. Following a reset, the I²C master controller must be re-initialized before it can be used again.

19.1.10 – Error Handling

The DS4835/36 I²C controller monitors the acknowledge bits received from the slave. If a NACK is received, FAIL_STAT bits will be set that will detail if the NACK occurred during the transmit of the slave address, memory address, or a data byte. Also, when a NACK is received, the XFR_FAIL bit will be set.

Some applications may benefit by implementing a software timeout for master I²C transactions. This timeout can be used to reset the master I²C controller if it takes longer than expected for a transaction to complete.

19.1.11 – Alternate Location

Some applications may require two slave devices with the same slave address. To support such applications, the DS4835/36 can provide the MSDA1 functionality at alternate pin locations. This is done by setting the appropriate MSDA_SEL[3:0] bits in the MIS2 register. The alternate pins for MSDA1are shown in Table 19.2.

MSDA_SEL[3:0]	Pin
0x00	GP10
0x01	GP10
0x02	GP12
0x04	GP13
0x08	GP14

Table 19-1: Default Master I²C Pins

19.2 – 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. The bit descriptions below detail how to use these registers.

Each of the I²C interfaces has its own dedicated registers. In the register names below, n = 1 or 2 differentiates the registers for the two independent interfaces.

While an I²C transfer is occurring, the software should not attempt to write to any registers. All register configuration must be done while there is not an ongoing I²C transaction.

19.2.1 – I²C Master Control Register (CNT2Wn)

Bit	15	14:12	11	10	9:8	7	6:4	3	2	1	0
Name	TW2WEN	MADR_2W	TWI2WIE	SADDR_ONLY	FAIL_STAT	ADSEL2W	NB_2W	XFR_FAIL	EWS	TWI2W	WMA
Reset	0	0	0	0	0	0	0	0	0	1	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15	TW2WEN	**I²C Master Enable:** Setting this bit to '1' will enable the I²C Master.
14:12	MADR_2W	**I²C Master Data Buffer Index.** Index to write to 8 data buffers. Auto increments on r/w to data buffers.
11	TWI2WIE	**I²C Master Interrupt Enable.** Setting this bit to '1' will enable an interrupt to be generated when the TWI2W bit is set.
10	SADDR_ONLY	**I²C Slave address only.** Setting this bit to '1' will terminate the current transaction after sending only the slave address. This is useful for polling slave addresses on the I²C bus.
9:8	FAIL_STAT	**I²C Fail Status.** 00 – No Fail 01 – Fail on slave address transfer 10 – Fail on mem address transfer 11 – Fail on data byte transfer in write cycle
7	ADSEL2W	**Address/Data Buffer Select:** Setting this bit to '0' will point DAD2W to the 8 data buffers. Setting this bit to '1' will point DAD2W to the MEM_ADDR register.
6:4	NB_2W	**Number of Data Bytes:** Number of bytes to read or write in the current transaction. Set this to the desired number of bytes – 1
3	XFR_FAIL	**I²C Master Fault Flag.** This bit is set to '1' to indicate that the last transfer has failed because a NACK was received
2	EWS	**End With Stop:** Setting this bit to '1' will issue a STOP at end of transaction. If this bit is cleared, the transaction will end without a STOP.
1	TWI2W	**Transmit Complete Interrupt Flag:** This bit is set to '1' to indicate that an I²C transaction has completed. 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	WMA	**With Memory Address:** Write Operation: If this bit is set to '1' the master I²C will transmit MEM_ADDR prior to sending any data bytes. If this bit is set to 0, the master I²C will not transmit MEM_ADD at the beginning of the transaction. Read Operation: If this bit is set to '1', master performs a write cycle first to send the memory address (dummy write), followed by a read cycle. If this bit is set to 0, the read cycle is a direct read with no dummy write.

19.2.2 – I²C Master Data Buffer Register (DAD2Wn when ADSEL2W = 0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED								DATA[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	r	r	r	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*

BIT	NAME	DESCRIPTION
15:8	Reserved	**Reserved**. The user should write 0 to these bits.
7:0	D[7:0]	Data buffer for received data, or data to be transmit.

The Data Buffer is 8-bytes deep. The MADR_2W[2:0] bits index which buffer is pointed to when accessing DAD2W.

Note: The DAD2Wn register is not read back by the debugger. When viewing this register in the debugger, it will always return as 0000h.

19.2.3 – I²C Master MEM_ADDR (DAD2Wn when ADSEL2W = 1)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED								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	r	r	r	r	r	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*

19.2.4 – I²C Slave Address Register (SLA2Wn)

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	R/W
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

This is the slave address that will be used for an I²C transaction. Writing to this register will start a transaction. All registers must be configured properly prior to writing to this register.

19.2.5 – I²C Master Clock Control Register (CCK2Wn)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	SCL_HI_CNT[7:0]								SCL_LO_CNT[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	SCL_HI_CNT[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. Refer to Table 19-2 to calculate the master I²C timing. SCL_HI_CNT [7:0] must be set to a minimum value of 2 to ensure proper operation.
7:0	SCL_LO_CNT[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. Refer to Table 19-2 to calculate the master I²C timing. SCL_LO_CNT[7:0] must be set to a minimum value of 4 to ensure proper operation.

19.2.6 – I²C Master Timing Specification Register (MTSPECn)

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

BIT	NAME	DESCRIPTION
15:10	TBUF[5:0]	**TBUF Time:** These bits provide the tbuf timing spec to create buffer time between start/start or stop/start. Refer to Table 19-2 to calculate the master I²C timing. The tbuf time will probably be dominated by the firmware delay in setting up a new transaction, not the time programmed by these bits.
9:4	TSTA[5:0]	**Start Hold and Stop Setup Time.** These bits define the start and stop setup timing. Refer to Table 19-2 to calculate the master I²C timing.
3	Reserved	**Reserved.** This bit should be set to 0.
2:0	THDD[2:0]	**Data Hold Time.** These bits define the data hold time. Refer to Table 19-2 to calculate the master I²C timing.

Timing Parameter	Value (ns)
t_{LOW}	(SCL_LO_CNT + 1) x 62.5
t_{HIGH}	(SCL_HI_CNT + 1) x 62.5
f_{SCL}	1 / [(SCL_LO_CNT + SCL_HI_CNT + 2) x 62.5]
$t_{SU:STA}$	(TSTA + 1) x 62.5
$t_{HD:DAT}$	(THDD + 1) x 62.5
$t_{SU:DAT}$	$t_{LOW} - t_{HD:DAT}$
$t_{SU:STO}$	(TSTA + 1) x 62.5
t_{BUF}	(TBUF + TSTA + 2) x 2 x 62.5
$t_{HD:STA}$	(SCL_HI_CNT + 1) x 62.5

Table 19-2: Master I²C Timing

SECTION 20 –SERIAL PERIPHERAL INTERFACE (SPI)

The DS4835/36 has a Serial Peripheral Interface (SPI) that can be used as a master or slave. As a master, this can be used to interface to any slave devices that use a SPI interface. As a slave, this can be used as a way of communication with a host system. The SPI interface allows access to a four-wire full-duplex serial bus. 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.

The four external interface signals used by the SPI module are MOSI (Master Out, Slave In), MISO (Master In, Slave Out), SPICK (SPI Clock), SSEL (Slave Select). The following table shows the location of these pins for master and slave operation.

Functional	Master Mode	Slave Mode
Output from Serial Shift Register	MSPI_MOSI: GP10	SSPI_MISO: GP13
Input to Serial Shift Register	MSPI_MISO: GP13	SSPI_MOSI: GP10
Serial Shift Clock	MSPI_CK: GP11	SSPI_CK: GP11
Slave Select	Any GPIO	SSPI_SSEL: GP12

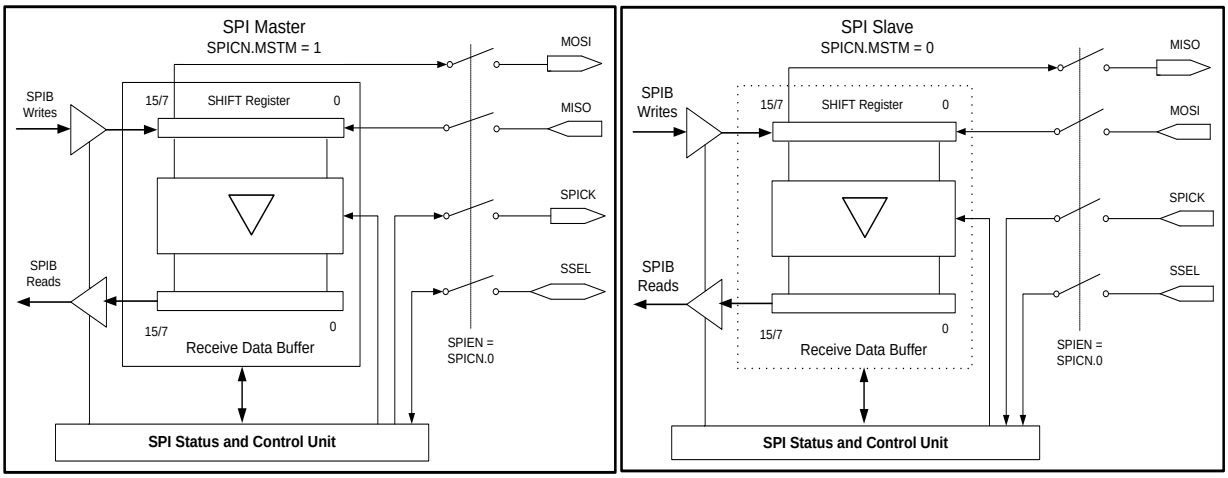


Figure 20-1: SPI Master and Slave Block Diagram

20.1 – Serial Peripheral Interface (SPI) Detailed Description

The block diagram Figure 20-1 shows the SPI external interface signals, control unit, read buffer, and single shift register common to the transmit and receive data path. This is shown for both master and slave operation. SPI can be viewed as a synchronous serial I/O port that shifts a 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 a 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 a 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.

The 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 a slave SPI device on the network.

20.1.1 – SPI Transfer Formats

During a 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 20-2 and Figure 20-3. The Slave Select signal can remain asserted between successive transfers. Table 20-1 illustrates the SPI modes.

Table 20-1 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

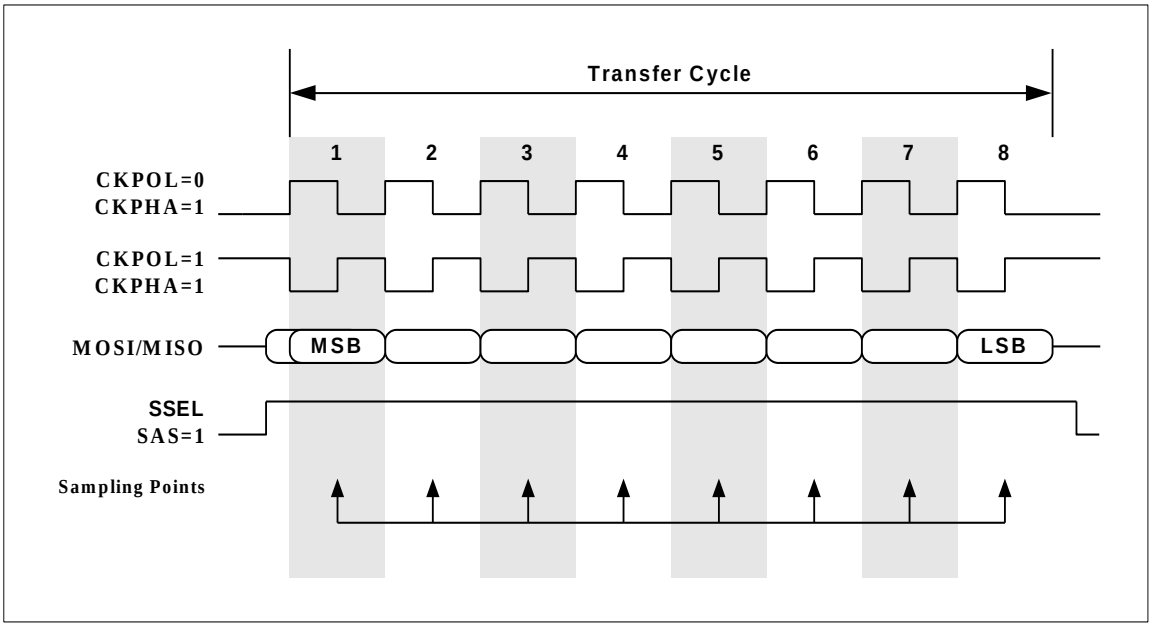


Figure 20-2: SPI Transfer Formats (CKPHA=1)

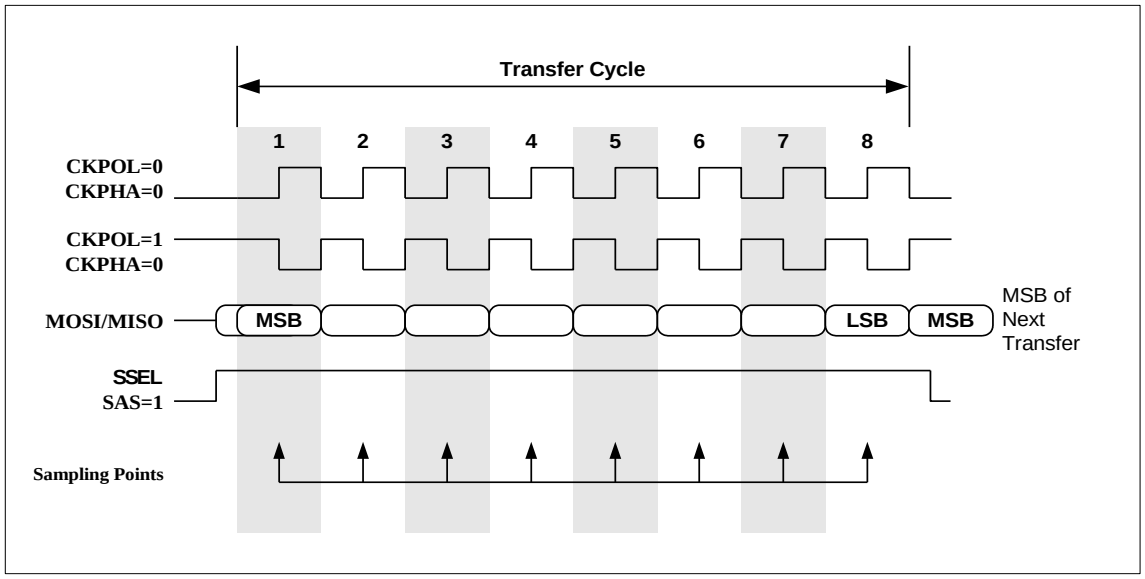


Figure 20-3: SPI Transfer Formats (CKPHA=0)

20.1.2 – 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. 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.

20.1.3 – Receive Overrun

Since the receive direction of SPI is double buffered, there is no overrun condition if 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.

20.1.4 – 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.

20.1.5 – SPI Interrupts

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

- Write Collision (WCOL)
- Receive overrun
- SPI Transfer Complete

These three 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 three SPI interrupt sources will cause an interrupt.

20.2 – 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 a SPI master device can initiate a data transfer. The master is responsible for manually selecting/deselecting slave(s) via any GPIO pin used as a Slave Select pin. Writing a data character to the SPI shift register (SPIB) starts a data transfer. The SPI master immediately shifts out the data serially on the MSPI_MOSI pin, most significant bit first, while providing the serial clock on MSPI_CK output. New data is simultaneously received on the MSPI_MISO pin and placed 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. During the transfer, the SPI Transfer Busy (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 (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 networks with different word length requirements. 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.

20.2.1 – SPI Transfer Baud Rates

When operating as a SPI master, the SPI serial clock (MSPI_CK) 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:

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

20.2.2 – SPI Slave Select

When operating in master mode, GPIO pins must be used for Slave Select signals. Using GPIO allows for multiple slave select pins to be used, allowing interfacing to multiple slave devices. User firmware must assert the appropriate Slave Select GPIO prior to beginning an SPI transaction, and deassert the Slave Select GPIO at the completion of the transaction.

20.3 – 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 clock (SSPI_CK) sourced from the master to control the data transfer.

The Slave Select SSPI_SSEL input must be externally asserted by a master before data exchange can take place. SSPI_SSEL must be asserted before data transaction begins 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 SSPI_SSEL edge, dependent on the data transfer format.

The SPI slave receives data from a master at the SSPI_MOSI pin, most significant bit first, while simultaneously transferring the contents of its shift register to the master on the SSPI_MISO pin, also most significant bit first. Data received from the master replaces data in the internal shift register until the transfer completes. The 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 8th shift clock edge.

When SSPI_SSEL is not asserted, the slave device ignores the SSPI_CK clock and the shift register is disabled. Under this condition, the device is basically idle, data will not be shifted out of or into the shift register. The SSPI_MISO 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 SSPI_SSEL 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.

The transfer format (CKPOL, CKPHA settings) and the character length selection (CHR) for the slave device, should match the master for a proper communication.

20.3.1 – SPI Slave Select

The SPI Slave Select (SSPI_SSEL) 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. When SAS is cleared to 0, SSPI_SSEL is configured to be active low. When SAS is set to 1, SSPI_SSEL is configured to be active high.

20.3.2 – 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.

20.4 – SPI Register Descriptions

The SPI Module has four SFR registers. These are SPICN, SPICF, SPICK and SPIB. The SPI control register SPICN and SPI configuration register SPICF controls and configures the Serial Peripheral Interface respectively. The SPI Clock Register SPICK configures SPI Baud rate in Master mode. The SPI Buffer SPIB is used in SPI data transfer.

SPI Control Register (SPICN)

Bit	7	6	5	4	3	2	1	0
Name	STBY	SPIC	ROVR	WCOL	MODF	RES	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 a 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 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	RESERVED	**Reserved.** Set to 0.
2	RESERVED	**Reserved.** Set to 0.
1	MSTM	**Master Mode Enable.** When set to '1', the SPI module will operate in Master mode. When set to '0', the SPI module will operate in Slave mode.
0	SPIEN	**SPI Enable.** Setting this bit to '1', enables the SPI Module. Setting this bit to '0', disables the SPI module.

SPI Data Buffer Register (SPIB)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	SPIB[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.

Note: The SPIB register is not read back by the debugger. When viewing this register in the debugger, it will always return as 0000h.

BIT	NAME	DESCRIPTION
15:0	SPIB[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.

SPI Configuration Register (SPICF)

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 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 SSPI_SSEL active state. When the SAS is cleared to '0', the SSPI_SSEL is active low and will respond to an external low signal. When the SAS is set to '1', the SSPI_SSEL is active high. This bit has no affect in master mode.
5:3	Reserved	**Reserved.** The user should write 0 to these bits.
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 the 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.

SPI Clock Register (SPICK)

Bit	7	6	5	4	3	2	1	0
Name	SPICK[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[7:0]	**Clock Divide Ratio Bits.** This register is used to set the baud rate when operating in master mode. The register has no function when operation in slave mode. 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)

SECTION 21 – MASTER 3-WIRE INTERFACE

The DS4835/36 has a proprietary 3-Wire master interface for communication with Maxim 3-wire laser drivers. The 3-wire communication mode operates similar to a SPI bus. However, in the 3-wire mode, there is one bi-directional I/O signal instead of separate data in and data out signals. The chip select is used to initiate and terminate a communication. The 3-Wire Master interface operates using a MSB first protocol and reads data on the falling edge of MSCL. During 3-Wire write operation the 3-Wire master outputs the data on the falling edge of MSCL.

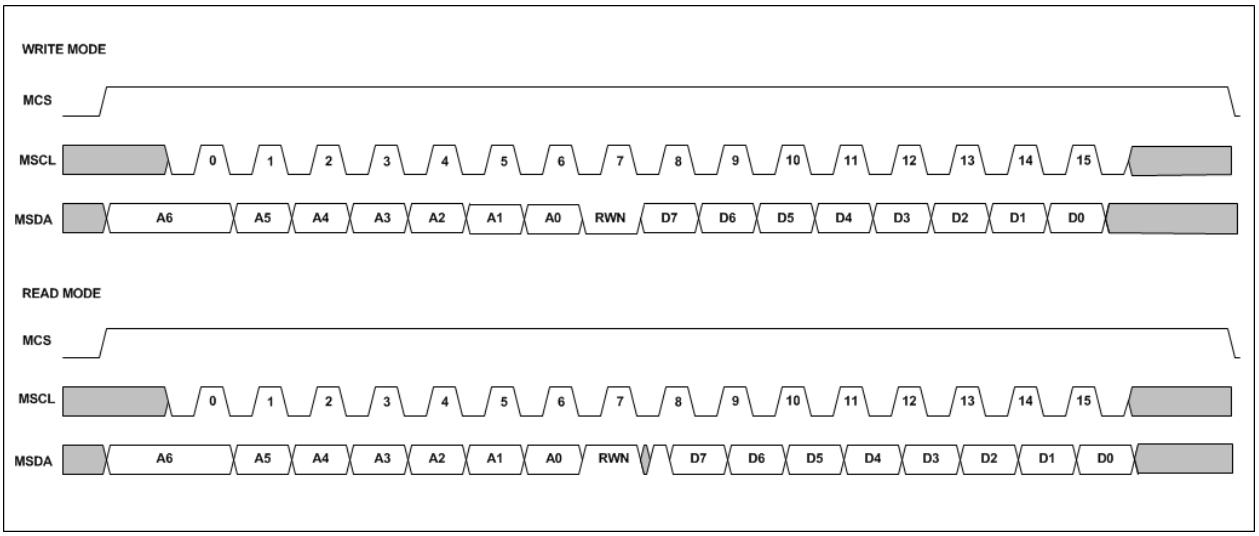


Figure 21-1: 3-Wire 1-byte Write and Read Operations

21.1 – Detailed Description

The DS4835/36 has a proprietary 3-Wire digital serial interface that is designed to interface with Maxim 3-wire slave devices (laser drivers). The DS4835/36 acts as the 3-Wire master. It is a 3-pin interface consisting of:

- MSDA: A bidirectional data line
- MSCL: Clock signal
- MCS: Chip select output. Chip select is active high.

The 3-Wire master initiates data transfer by generating clock. The MSCL pin is used to synchronize data movement between the DS4835/36 and the slave device.

The DS4835/36 3-Wire interface supports multiple-byte data transfers of up to 8 bytes. The 3-Wire interface provides 8 user selectable MSCL clock frequencies. The 3-Wire Control Register (TWR) is used to control and configure the 3-Wire interface.

21.1.1 - Starting a Transaction

All 3-wire transactions are started by writing a memory address to the MEM_ADD register (DADDR when AD_SEL = 1). When this write occurs, the 3-Wire master will begin the transaction by first sending the memory address.

As shown in Figure 21.1, the 8th bit sent is a read/write bit. When this bit is a '1', the 3-wire master will operate in receiver mode (data is read from slave) at the completion of the memory address. When bit 0 is '0', the 3-wire master will operate in transmitter mode (data is written to slave) at the completion of the memory address.

21.1.2 – Chip Select Options

The DS4835/36 3-Wire master will automatically assert the MCS pin (logic high) at the beginning of a

transaction, and release MCS (logic low) at the end of a transaction. This is shown in Figure 21-1. If there are multiple slaves in a system, the DS4835/36 provides an option to disable the assertion of MCS. When this is disabled, any GPIO can be configured to function as chip select and the application program should control chip select during the 3-Wire communication.

21.1.3 – Data Buffer

The DS4835/36 3-Wire master has an 8-byte data buffer that allows up to 8 bytes to be read or written during one transaction. This data buffer is accessed by reading or writing to the DADDR register when the AD_SEL bit in TWR is set to 0. The MADR_3W[2:0] bits are used to index each of the 8 data buffers. MADR_3W will automatically increment on each access of DADDR.
When the 3-wire master is writing data, the data buffer must be loaded with all data prior to the transaction starting. When reading data, all received bytes of data are available in the data buffer at the completion of the transaction.

21.1.4 – Transmitting Data

The DS4835/36 3-Wire master enters data transmission mode after transmitting a memory address with the R/W bit set to a 0. The steps to perform a data transmission are detailed below. These must be done before writing the memory address and starting the transaction

1) Clear the TWI bit so software will know when the transaction is complete.
2) Set AD_SEL to 0 so DADDR will point to the DataBuffer registers.
3) Set MADR_3W to 0 to index DataBuffer[0], which will be the first data byte transmitted
4) Write the first data byte to DADDR.
5) Repeat step 3 until up to 8 bytes are loaded. MADR_3W will automatically increment after each write to DADDR.
6) Set NB_3W to the number of data bytes to transmit – 1. So, to transmit 1 byte, write NB_3W to 0.
7) Set AD_SEL to 1 so DADDR will point to the MEM_ADD register.
8) Write the desired memory address with the read/write bit set to 0 to DADDR. This ill be stored to MEM_ADD. This write will start the 3-wire transaction
9) Wait for TWI to be set to signal the transaction is complete.

21.1.5 – Receiving Data

The DS4835/36 3-Wire master enters into data transmission mode after transmitting a memory address with the R/W bit set to a 0. The steps to perform a data read are detailed below. These must be done before writing the memory address and starting the transaction.

1) Clear the TWI bit so software will know when the transaction is complete.
2) Set MADR_3W to 0 so DADDR points to DataBuffer[0]. This is where the first byte of received data will be stored.
3) Set NB_3W to the number of data bytes to receive – 1. So, to receive 1 byte, write NB_3W to 0.
4) Set AD_SEL to 1 so DADDR will point to the MEM_ADD register.
5) Write the desired memory address with the read/write bit set to 1 to DADDR. This will be stored to MEM_ADD. This write will start the 3-wire transaction
6) Wait for TWI to be set to signal the transaction is complete.

After all the bytes are received, the TWI bit will set. The data can then be read by doing the following steps.

1) Set AD_SEL to 0 so DADDR points to the DataBuffers.
2) Set MADR_3W to 0 so DADDR points to DataBuffer[0]. This is where the first byte of received data is stored.
3) Read the received byte(s) from DADDR.

21.2 – 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.

3-Wire Control Register (TWR)

Bit	15	14:12	11	10:8	7	6:4	3	2	1	0
Name	RES	NB_3W	AD_SEL	MADR_3W	TWEN	CLK_EXT	TWIE	DIS_CS	TWI	TWB
Reset	0	0	0	0	0	0	0	0	1	0
Access	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15	RESERVED	**Reserved**. The user should not write to this bit.
14:12	NB_3W	**Number of Data Bytes:** Number of bytes to read or write in the current transaction. Set this to the desired number of bytes – 1
11	AD_SEL	**Address/Data Buffer Select:** Setting this bit to '0' will point DADDR to the 8 data buffers. Setting this bit to '1' will point DADDR to the MEM_ADDR register.
10:8	MADR_3W	**3-Wire Data Buffer Index.** Index to write to 8-byte data buffer. This auto increments on r/w to data buffer.
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	CLK_EXT [2:0]	**3-Wire Clock Period.** These bits are used for setting the 3-Wire MSCL 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	DIS_CS	**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, software can select and use any GPIO as a chip select. Clearing this bit will enable MCS as active chip select and it is set to high (See Figure 21-1) 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 the data transfer is completed. This bit can generate interrupt if TWIE bit is enabled. Once set, this bit should be cleared by software.
0	TWB	**3-Wire Busy.** This bit is set to '1' at the start of a transaction and will be cleared to '0' at the end of the transaction. This bit cannot be written by the user.

CLK_EXT[2:0]	MSCL Clock Period
000	1000 ns
001	1250 ns
010	1500 ns
011	1750 ns
100	2000 ns
101	2250 ns
110	2500 ns
111	2750 ns

3-Wire Master Data Buffer Register (DADDR when AD_SEL = 0)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED								DATA[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	r	r	r	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*

BIT	NAME	DESCRIPTION
15:8	Reserved	Reserved. The user should write 0 to these bits.
7:0	D[7:0]	Data buffer for received data, or data to be transmit.

The Data Buffer is 8-bytes deep. The MADR_3W[2:0] bits index which buffer is pointed to when accessing DADDR.

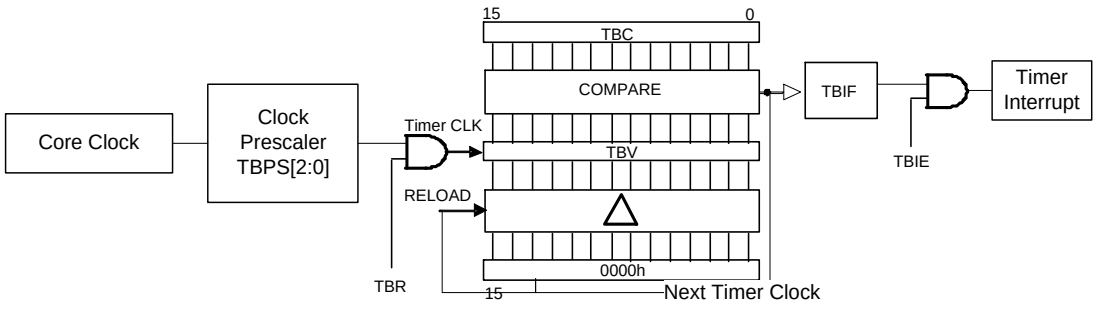
3-Wire Master MEM_ADDR (DADDR when AD_SEL = 1)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED								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	r	r	r	r	r	rw*	rw*	rw*	rw*	rw*	rw*	rw*	rw*

SECTION 22 – GENERAL-PURPOSE TIMERS

The DS4835/36 has three identical 16-bit general-purpose timers. Each timer has the following,

- Two modes of operation: Free synchronous counter and Compare Mode
- Interrupt options for both modes of operation
- 6 selectable clock prescalers



Compare Mode Block Diagram

Notes:

1. Free synchronous mode is a special compare mode in which the TBV is compared with 0xFFFF.

2. The TBV register is reset to 0x0000 when the TBR bit is set to '0' (Timer off).

Figure 22-1: Timer Functional Block Diagram

22.1 – Detailed Description

The DS4835/36 has three 16-bit programmable timer modules. The core clock is the input source for each timer. Each timer has two modes of operation i.e. free synchronous timer and compare mode. The timer is enabled using the Time Base Run (TBR) bit in the Time Base Control Register (TBCN). When this bit is set to '1', it enables the timer, which starts counting up. When this bit is set to '0', the timer is stopped. Each timer has six clock prescalers that can be selected. Using various prescalers and compare modes, various timing loops can be generated.

22.1.1 – Timer Modes

Each timer has two modes of operation i.e. free synchronous timer and compare mode. The MODE bit in the TBCN 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 Time Base Value Register (TBV) value reaches to 0xFFFF, the TBIF interrupt flag is set to '1' which generates an interrupt if enabled, and the timer reloads the TBV register with 0x0000 at the next timer clock.

In compare mode, the timer module begins counting from 0x0000 and when the value in the TBV register matches the value in the Time Base Compare Register (TBC), the TBIF interrupt flag is set to '1' which generates an interrupt if enabled. When the match occurs, the timer reloads the TBV register with 0x0000 at the next timer clock. In compare mode, the TBC register should be written first before setting the TBR bit.

22.2 – Timer Register Descriptions

Each timer module has three independent SFR registers. These are TBCN, TBV and TBC. The Time Base Control Register TBCN controls the timer operation. The Time Base Value Register TBV is the Timer Value register and is incremented every timer clock when enabled. The Time Base Compare Register TBCx is used in the timer compare mode only.

22.2.1 – Time Base Control Register (TBCN1, TBCN2, TBCN3)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	TBR	MODE	-	-	TBIE	-	-	-	TBIF	-	TBPS[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	TBR	**Timer Run Control.** Setting this bit to '1' will enable the timer. Clearing this bit to '0' will stop the timer and clear the TBV 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 TBV register starts counting from 0x0000. When the TBV register reaches 0xFFFF, TBIF is set to '1' and the TBV 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 TBV register starts counting from 0x0000. When the TBV register matches the value in the TBC register, TBIF is set to '1' and the TBV register reloads to 0x0000 at the next timer clock. **NOTE**: In the compare mode, the TBC register value should be set prior to write to the 'MODE' bit.
10:9	Reserved	**Reserved**. The user should write 0 to these bits.
8	TBIE	**Timer Interrupt Enable.** Setting the TBIE bit to '1' causes an interrupt to be generated to the CPU when TBIF=1. Clearing this bit to '0' will not cause an interrupt when TBIF=1.
7:5	Reserved	**Reserved**. The user should write 0 to these bits.
4	TBIF	**Timer Matched Interrupt Flag.** This bit is set to '1' when 1. In free synchronous mode, the TBV register value reaches 0xFFFF. 2. In compare mode, the TBV register value matches the value in the TBC register. This flag generates an interrupt if the TBIE 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	TBPS[2:0]	**Timer Prescaler Select.** These bits configure the prescaler from the core 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

22.2.2 – Time Base Value Register (TBV1, TBV2 and TBV3)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	TBV(1,2,3)															
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

The TBV register is a read only register and it resets to 0x0000 when

- The timer is stopped (TBR = 0).
- TBV = 0xFFFF when operating in free synchronous mode.
- TBV = TBC when operating in compare mode.

22.2.3 – Time Base Compare Register (TBC1, TBC2 and TBC3)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	TBC(1,2,3)															
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

The TBC register is used as a comparison for the TBV register when operating on compare mode. This register has no impact when operating in free synchronous mode.

SECTION 23 – HARDWARE MULTIPLIER MODULE

The hardware multiplier module can be used by the DS4835/36 to support high-speed multiplication. 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)

23.1 – 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 23-1 shows a block diagram of the hardware multiplier.

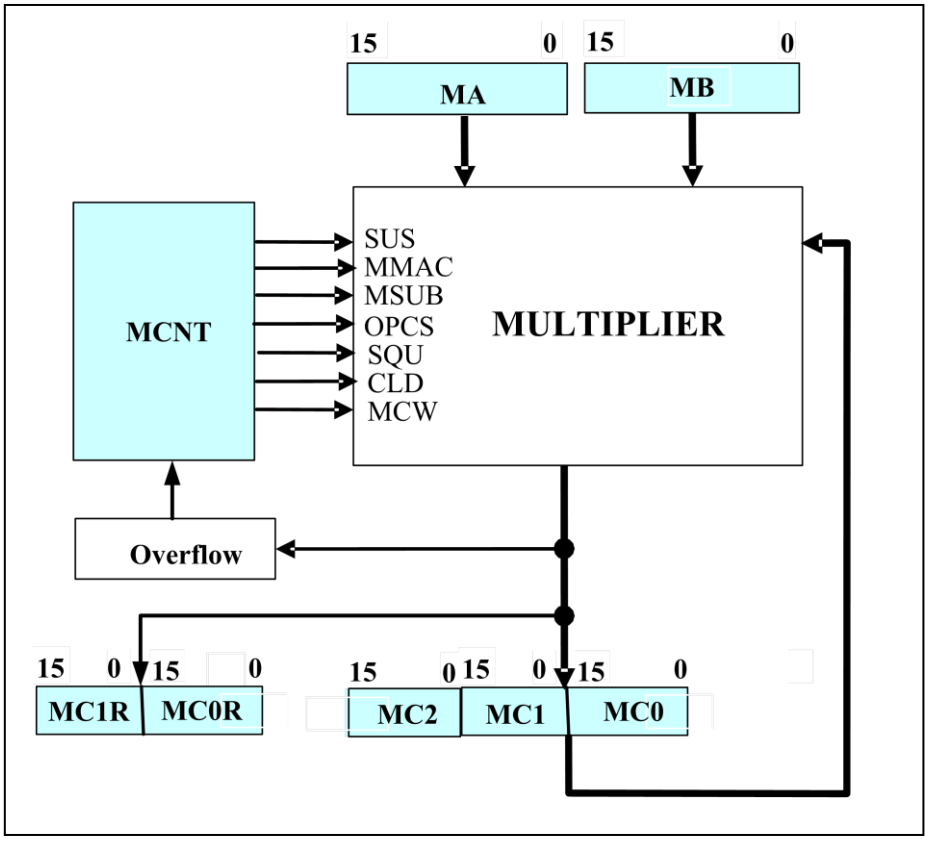


Figure 23-1: Multiplier Organization

23.2 – 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.

23.3 – 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 23-1 for details.

23.3.1 – 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.

23.3.2 – 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 writes to both registers.

23.4 – 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 23-1 shows the operations supported by the multiplier and associated MCNT control bit settings.

23.4.1 – 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 the 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 23-1 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 DS4835/36 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.

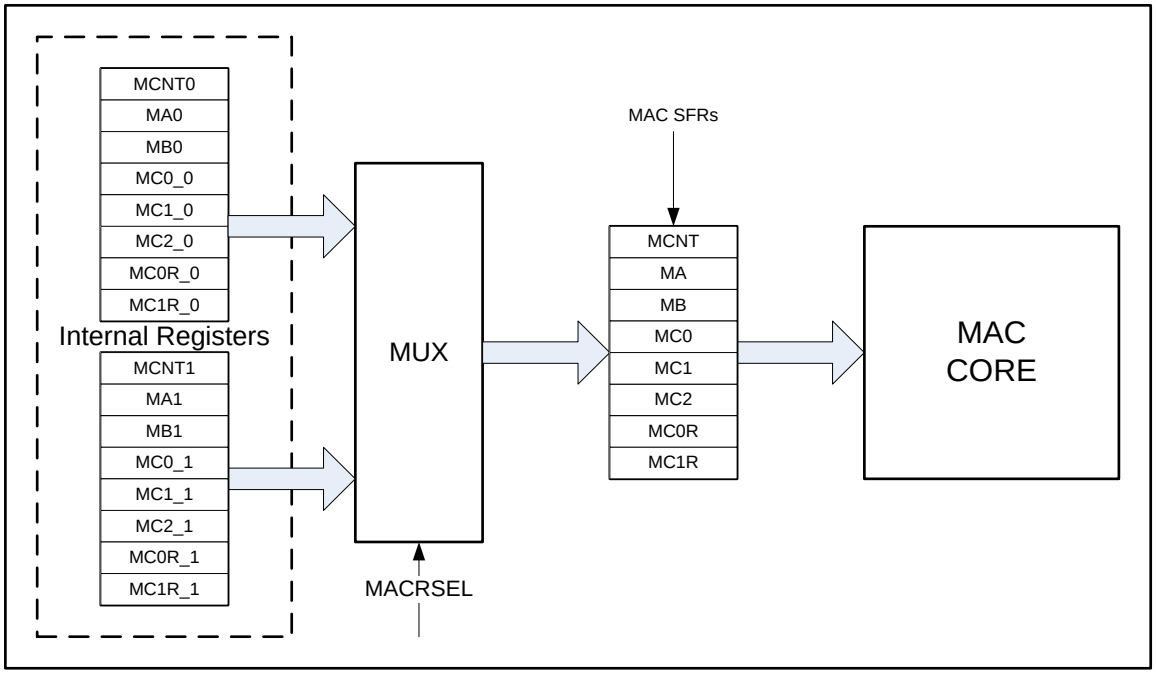


Figure 23-2: **Dual MAC registers Organization**

23.5 – 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 23-2 Hardware Multiplier Registers

Register	Function
MCNT	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	Multiplier Operand A Register. Used by the user software to load one of the 16-bit values for a hardware multiplier operation.
MB	Multiplier Operand B Register. Used by the user software to load one of the 16-bit values for a hardware multiplier operation.
MC2	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	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	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	Multiplier Read Register 1. 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 until an SFR related to the multiplier has been changed.
MC0R	Multiplier Read Register 0. 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 unchanged until an SFR related to the multiplier has been changed.
SHFT	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	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.

23.5.1 – 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

23.5.2 – 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.

23.5.3 – 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.

23.5.4 – 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.

23.5.5 – 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.

23.5.6 – 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.

23.5.7 – 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.

23.5.8 – 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.

23.5.9 – 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.

23.5.10 – MAC Shift Register (SHFT)

Bit	7	6	5	4	3	2	1	0
Name	CSHFT	RSHFT	LSHFT	SHVAL				
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7	CSHFT	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 to a 0.
6	RSHFT	Shift Right: Writing a 1 to this bit will cause the MC2-M0 registers to shift right the number of bits defined by SHVAL. This bit auto clears itself, so a read on SHFT register will always return 0 for this bit position.
5	LSHFT	Shift Left: Writing a 1 to this bit will cause the MC2-M0 registers to shift left the number of bits defined by SHVAL. This bit auto clears itself, so a read on SHFT register will always return 0 for this bit position.
4:0	SHVAL	Shift Value: These bits specify number of positions to be shifted left or right depending on which bit is set LSHFT or RSHFT. A maximum of 31 shift operations can be performed in one write operation to SHFT register.

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:

1:
MA = 2; MB = 2;
    ⇨ MC2 MC1 MC0 = 2*2 = 0000 0000 0100
SHVAL = 1; // one bit shift operation
RSHFT =1; //Shift right by one bit
    ⇨ MC2 MC1 MC0 = 0000 0000 0010 = 2
2:
MA = 8; MB = 8;
    ⇨ MC2 MC1 MC0 = 8*8 = 0000 0100 0000
SHVAL = 6; // Six bits shift operation
RSHFT =1; //Shift right by 6 bits
    ⇨ MC2 MC1 MC0 = 0000 0000 0001 = 1

Left Shift Operation:

1:
MA = 2; MB = 2;
    ⇨ MC2 MC1 MC0 = 2*2 = 0000 0000 0100
SHVAL = 1; // one bit shift operation
LSHFT =1; //Shift left by one bit
    ⇨ MC2 MC1 MC0 = 0000 0000 1000 = 8
2:
MA = 1; MB = 2;
    ⇨ MC2 MC1 MC0 = 1*2 = 0000 0000 0010
SHVAL = 6; // Six bits shift operation
LSHFT =1; //Shift left by 6 bits
    ⇨ MC2 MC1 MC0 = 0000 1000 0000 = 128

23.6 – 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 24 – WATCHDOG TIMER

24.1 - 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 24-1 below, 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 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.

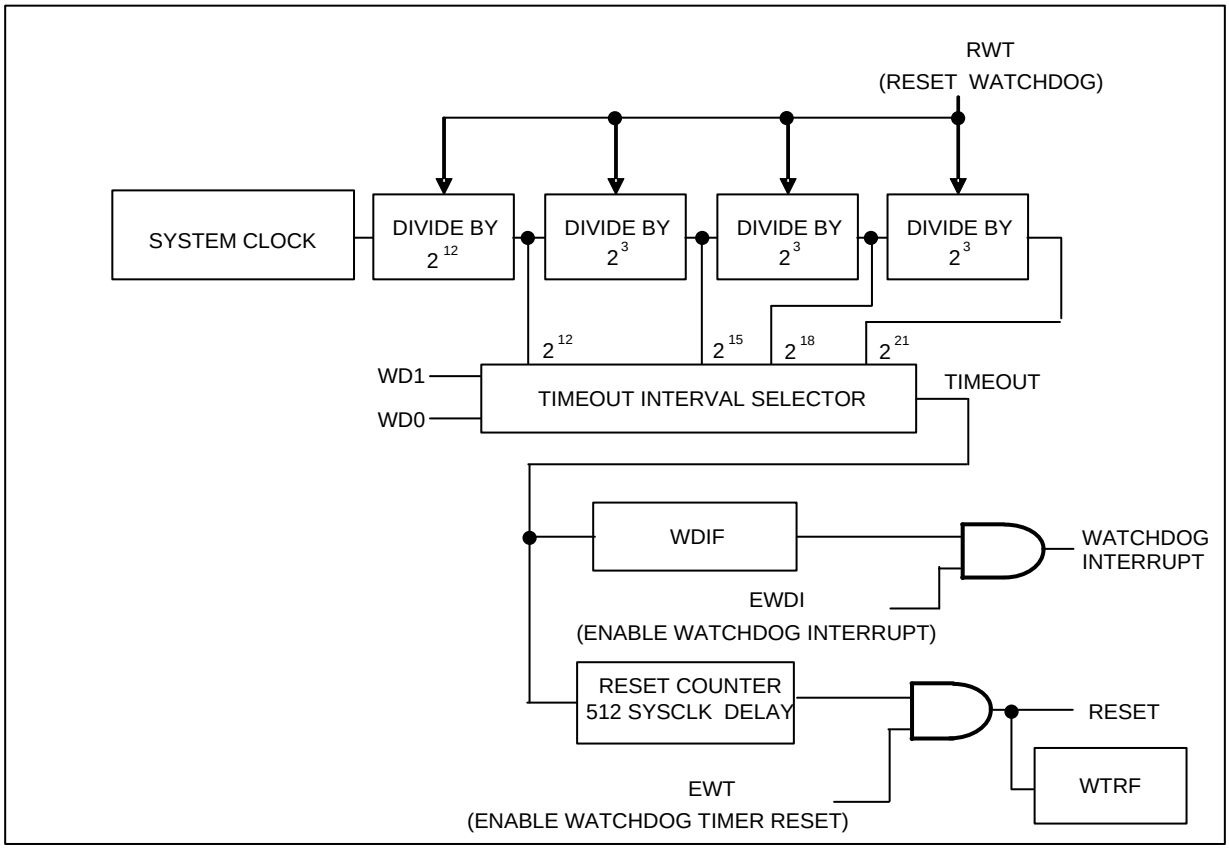


Figure 24-1: **Watchdog Timer Block Diagram**

24.2 – 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 DS483X. Table 24-1 describes the possible states of the watchdog timer.

Table 24-1. 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	Watchdog timeout has occurred, but 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 timeout has occurred, and an interrupt has been generated. Watchdog reset will occur in 512 system clock cycles if RWT is not set or WDIF not cleared.

24.2.1 – 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.

24.2.2 – Watchdog Timer Reset Operation

In order to reset the DS483X, 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 DS483X 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.

24.2.3 – 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 a watchdog interrupt being generated. At this point the watchdog timer reset feature 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 24-2.

Table 24-2. 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 DS483X 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 25 – SUPPLY VOLTAGE MONITOR (SVM)

The DS4835/36 provides a feature to allow monitoring of its power supply. The Supply Voltage Monitor (SVM) monitors the V_{CC} power supply and can alert the processor through an interrupt if V_{CC} falls below a programmable threshold. Note, V_{CC} for the DS4835/36 is the same level as VCCO, which is the output from the inrush circuit.

The DS4835/36 provides the following power monitoring features:

- SVM compares V_{CC} 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.

Note: The SVM thresholds listed below are approximate and there will be device to device variation that can exceed 100mV. Across temperature, these thresholds will also vary by +/-50mV.

The Supply Voltage Monitor is controlled by the peripheral register SVM.

Supply Voltage Monitor Register (SVM)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	SVTH[2:0]			-	-	-	-	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	r	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.
10:8	SVTH[3:0]	**Supply Voltage Threshold Bits [3:0]**: These bits are used to select a user defined supply voltage threshold. If V_{CC} is below this threshold SVMI bit will be set and an interrupt can be generated if enabled. The threshold level can be adjusted from 2.65V to 3.10V. The default value is 00h for 3.10V. **SVTH[2:0]** / **SVM Level** 0 / 3.10 1 / 3.05 2 / 3.0 3 / 2.9 4 / 2.85 5 / 2.75 6 / 2.7 7 / 2.65 SVM Level - Voltage level at which SVMI trip occurs. Note: The SVTH[2:0] bits can only be modified when SVMEN = 0. Writing to these bits is ignored if SVMEN = 1.
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_{CC} supply voltage falls below the threshold defined by SVTH[2:0]. 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_{CC} 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_{CC} against the programmed (SVTH[2:0]) threshold. After SVMEN is set, SVMRDY will be set in approximately 20 us. Clearing this bit to 0 disables the SVM.

SECTION 26 – MISCELLANEOUS

Some other miscellaneous features of DS4835/36 are:

- CRC8
- Miscellaneous Registers

26.1 - CRC8

The DS4835/36 has a built-in hardware CRC8 calculator. The registers used for CRC8 are CRCIN and CRCOUT. The CRC algorithm that is calculated is x^8+x^2+x+1.

CRC Data in (CRCIN)

Bit	7	6	5	4	3	2	1	0
Name	CRCIN_7	CRCIN_6	CRCIN_5	CRCIN_4	CRCIN_3	CRCIN_2	CRCIN_1	CRCIN_0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7:0	CRCIN[7:0]	**CRC Data in.** The user program writes data to this register for which the CRC8 calculation should be performed upon.

CRC Data out (CRCOUT)

Bit	7	6	5	4	3	2	1	0
Name	CRCOUT_7	CRCOUT_6	CRCOUT_5	CRCOUT_4	CRCOUT_3	CRCOUT_2	CRCOUT_1	CRCOUT_0
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
7:0	CRCOUT[7:0]	**CRC Data out.** The user program reads CRC8 result from this register for all the data that was written to CRCIN. NOTE: This register has to be cleared to 0x00 by software before starting a CRC8 calculation.

26.1.1 Example

```
unsigned char Calculate_CRC8(unsigned char* data, int length)
{
        unsigned int i = 0;
        unsigned char CRC_result;
        CRCOUT = 0x00;
        for( ; i<length ; i++)
        {
                CRCIN = data[i];
                //Incrementing i in the loop takes a cycle atleast. So CRC should have been completed in this time.
        }
        CRC_result = CRCOUT;
        return CRC_result;
}
```

26.2 - Miscellaneous Registers

The DS4835/36 does contains registers that are used for a few miscellaneous functions. These registers are described below.

MIS1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	-	-	-	DCDC4	-	-	-	-	-	-	-	-	-	-	-	-
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
15:14	Reserved.	Reserved. Set these bits to 0.
12	DCDC4	Setting this bit to 1 will enable the APD4 controller to drive the FETs for DCDC4 located at LX4.
11:0	Reserved.	Reserved. Set these bits to 0.

MIS2

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3:0
Name	-	-	-	-	-	-	-	-	-	-	-	-	MSDA_SEL[3:0]
Reset	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

BIT	NAME	Description
15:14	Reserved.	Reserved. Set these bits to 0.
3:0	MSDA_SEL[3:0]	These bits determine which GP pin will be used for the MSDA1 functionality.

MSDA_SEL[3:0]	Pin
00h	GP10
01h	GP10
02h	GP12
04h	GP13
08h	GP14

DEV_NUM

Bit	7	6	5	4	3	2	1	0
Name	DIS_I2C_LOAD	DEV_NUM[6:0]						
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	Description
7	DIS_I2C_LOAD	When this bit is set to 1, the I2C bootloader slave address 34h will be disabled. Refer to the In-System Programming section for more details.
6:0	DEV_NUM[6:0]	General Purpose bits that can be used to number this device, or anything else required by the application.

GP_REG

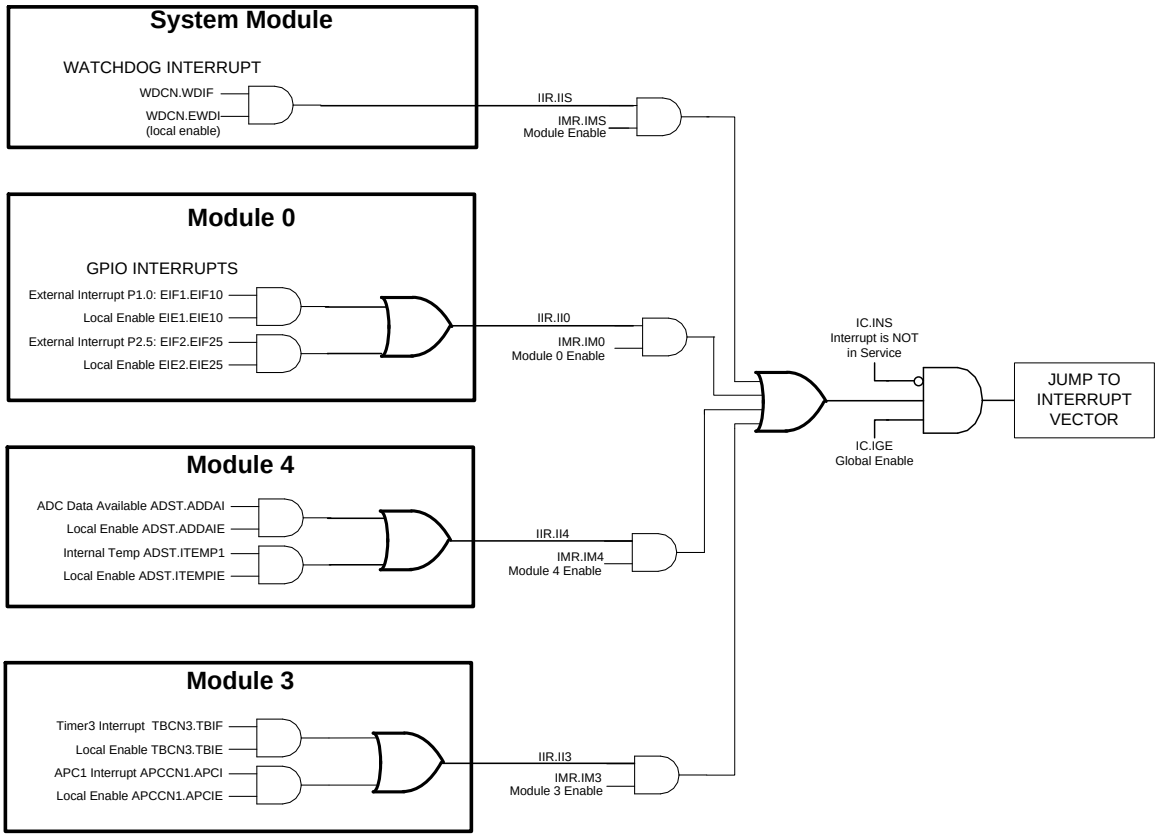
Bit	7	6	5	4	3	2	1	0
Name	GP_REG[7:0]							
Reset	0	0	0	0	0	0	0	0
Access	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	Description
7:0	GP_REG[7:0]	8-bit General Purpose Register.

The DS4836 has one General Purpose Registers and the DS4835 has two General Purpose Registers. These registers are commonly used to store the value of the IMR register while processing interrupts.

SECTION 27 – INTERRUPTS

The DS4835/36 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 such as the ADC. 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. Figure 27-1 shows a diagram of the interrupt hierarchy.



Note: Only a few of the DS4835 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 27-1: Interrupt Hierarchy

27.1 – Servicing Interrupts

For the DS4835/36 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 (IMR). 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 27-1 as well as Table 27-1.

Table 27-1. Interrupt Sources and Control Bits

Interrupt	Interrupt Flag	Local Enable Bit	Interrupt Identification Bit	Module Enable Bit
Port 0 GPIO	EIF0.EIF0n	EIE1.EIE1n	IIR.II0	IMR.IM0
Port 1 GPIO	EIF1.EIF1n	EIE2.EIE2n		
Port 2 GPIO	EIF2.EIF2n	EIE3.EIE3n		
Timer 1 Flag	TBCN1.TBIF	TBCN1.TBIE		
Timer 2 Flag	TBCN2.TBIF	TBCN2.TBIE		
PLA[4:1] Flag	PLACNTn.PLA_OUT	PLACNTn.PLAIE		
Port 6 GPIO	EIF6.EIF6n	EIE6.EIE6n	IIR.II1	IMR.IM1
SVM	SVM.SVMI	SVM.SVMIE		
I2C Slave Start	I2CST_S.ICSRI	I2CIE_S.ICSRIE	IIR.II2	IMR.IM2
I2C Slave Transmit	I2CST_S. I2CTXI	I2CIE_S. I2CTXIE		
I2C Slave Receive	I2CST_S. I2CRXI	I2CIE_S. I2CRXIE		
I2C Slave Clock Stretch	I2CST_S. I2CSTRI	I2CIE_S. I2CSTRIE		
I2C Slave Timeout	I2CST_S. I2CTOI	I2CIE_S. I2CTOIE		
I2C Slave Address Match	I2CST_S. I2CAMI	I2CIE_S. I2CAMIE		
I2C Slave NACK	I2CST_S. I2CNACKI	I2CIE_S. I2CNACKIE		
I2C Slave General Call	I2CST_S. I2CGCI	I2CIE_S. I2CGCIE		
I2C Slave Overrun	I2CST_S. I2CROI	I2CIE_S. I2CROIE		
I2C Slave Stop	I2CST2_S. I2CSPI	I2CIE2_ I2CSPIE		
I2C Slave Start Address	I2CST2_S.SADI	I2CIE2_S.SADIE		
I2C Slave Memory Address	I2CST2_S.MADI	I2CIE2_S.MADIE		
I2C Slave Page Threshold	I2CTXFST.THSH	I2CTXFIE.THSH		
I2C Slave FIFO Threshold	I2CRXFST.THSH	I2CRXFIE.THSH		
Timer 3 Flag	TBCN3.TBIF	TBCN3.TBIE	IIR.II3	IMR.IM3
APC[4:1] Flag	APCCNn.APCI	APCCNn.APCIE		
ADC Data	ADST.ADDAI	ADST.ADDAIE	IIR.II4	IMR.IM4
ADC P1 Data	ADST.PR1I	ADST.PR1IE		
ADC P2 Data	ADST.PR2I	ADST.PR2IE		
dInternal Temp Data	ADST.ITEMPI	ADST.ITEMPIE		
External Temp Data	ADST.ETEMP1	ADST.ETEMPIE		
Sample Hold Data 1	ADST.SH1I	ADST.SH1IE		
Sample Hold Data 2	ADST.SH2I	ADST.SH2IE		
QT Low Threshold[7:0]	LTI.IFn	LTI.IEn		
QT High Threshold[7:0]	HTI.IFn	HTI.IEn		
QT High Threshold[7:0]	HTI1.HTIIn	HTI1.HTIEn		
QT High Threshold[12:8]	HTI2.HTIIn	HTI2.HTIEn		
APC Data Available	APCN.APCI	APCN.APCIE		
Master I2C Complete[1:0]	CNT2Wn.TWI2W	CNT2Wn.TWI2WIE	IIR.II5	IMR.IM5
SPI Interrupts	SPICN.SPIC, ROVR, WCOL	SPICN.SPIEN		
3-Wire Complete	TWR.TWI	TWR.TWIE		
Watchdog Timeout	WDCN.WDIF	WDCN.EWDI	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.

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.

27.2 – 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 DS4835/36 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.

27.2.1 – 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. 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.

27.2.2 – Interrupt Prioritization by Software

All interrupt sources of the DS4835/36 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.

27.2.3 – Interrupt Exception Window

An interrupt exception window is a non-interruptible 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. There is a single condition in the DS4835/36 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 28 – 1-WIRE TEST ACCESS PORT

The Test Access Port (TAP) allows an external device to communicate with the onboard debugger and bootloader interfaces which are implemented by a combination of hardware and utility ROM support routines. The TAP takes the form of a 1-Wire interface which consists of a single bidirectional data line multiplexed with the reset function.

Using the 1-Wire interface, the external master can read and write dedicated data buffers and status registers without processor intervention. The 1-Wire slave interface supports overdrive communications only.

For more information on the Maxim 1-Wire protocol and supporting devices, refer to the following sources:

- AN937: The Book of iButton Standards (http://www.maximintegrated.com/app-notes/index.mvp/id/937)
- AN1796: Overview of 1-Wire Technology and Its Use (http://www.maximintegrated.com/app-notes/index.mvp/id/1796)
- AN187: 1-Wire Search Algorithm (http://www.maximintegrated.com/app-notes/index.mvp/id/187)

While the TAP's 1-Wire slave interface is generally compatible with the 1-Wire standards as defined by the above sources, there are a few differences that must be considered when connected to a 1-Wire network.

- The 1-Wire slave interface does not support standard speed communications – all communications take place at overdrive speed only. This also means that the standard speed versions of commands such as Skip ROM and Match ROM are not supported.
- The Read ROM command is not supported. If only one device is on the 1-Wire network, the Overdrive Skip ROM command can be used to access the device, or if necessary the Search ROM command can be used to obtain the ROM ID.
- If the 1-Wire bus is shared with other 1-Wire devices, a standard-speed reset pulse will trigger a full device reset (external reset), not just a 1-Wire reset.

28.1 – OWL/RST Pin Functionality

In addition to providing a reset to the DS4835/36, the RESET pin is also dedicated for use by the 1-Wire slave interface. For clarity, this document will refer to this pin as the One-Wire Loader/RESET (OWL/RST) pin.

Combining the standard reset function with a 1-Wire communications interface requires only one pin (OWL/RST) to operate (plus ground), compared with the four communications lines (TCK, TMS, TDI, TDO) plus reset that are required for a standard JTAG TAP interface. Additionally, since an external reset pin would be included on the device in any case, adding the 1-Wire TAP port does not increase the pin count or occupy any GPIO pins or other resources during debugging.

Because the 1-Wire interface and reset function are multiplexed on the OWL/RST the minimum pulse width for an external reset is extended (from the usual 4 system clock cycle width) to approximately 150µs (refer to the datasheet for more information). This means that low pulses shorter than 150µs will be ignored by the device or will be treated as 1-Wire communications if they follow the proper format.

Note that the 1-Wire interface is a slave interface only and is not a master interface (which would enable communication with other slave devices). Instead, this interface is designed to allow 1-Wire master devices (or microcontrollers implementing the 1-Wire protocol) to communicate with the bootloader and debug functions. It is possible to implement a 1-Wire master function in firmware, as with any MAXQ microcontroller, but this would have to be done using a different pin as the OWL/RESET pin is dedicated for use by the 1-Wire slave interface.

Note that the 1-Wire slave interface on the OWL/RESET pin is dedicated for use by the bootloader and debugger only and cannot be used for general-purpose 1-Wire communications by application code.

28.2 – Disabling the 1-Wire/TAP Interface

It is possible for an application to disable the 1-Wire/TAP interface entirely by setting the SC.TAP bit in the System Control register to 0. Setting this bit to 0 has the following effects.

- The minimum low pulse length needed to trigger an external reset is reduced from the 1-Wire-compatible 150µs value to the standard 4-system-clock-cycle length. This allows the device to respond to an external reset condition more quickly than when the 1-Wire/TAP interface is active.
- A low pulse will be output on the OWL/RST pin in the standard manner whenever a reset condition internal to the device occurs (such as a watchdog reset).
- Most importantly, the 1-Wire interface is disabled, which means that all access to the bootloader and/or debugger is blocked until SC.TAP is set back to 1, or the device is reset (which will reset SC.TAP to 1 automatically).

Note: Special care must be taken when using this feature to disable the 1-Wire/TAP. Unlike with a JTAG interface, it is not possible to hold the device in external reset while communicating with the registers in the TAP. Since the reset function and 1-Wire communications are multiplexed on the same pin, this means that the external reset must be released (allowing the device to begin executing code) before 1-Wire communications can begin. Since it takes a certain number of 1-Wire communications transactions to even attempt to enter bootloader or debugger mode, it is inevitable that a certain number of instructions at the beginning of the application will execute before the 1-Wire master has a chance to assert control of the device.

If the application code sets SC.TAP to 0 (disabling the 1-Wire TAP) at the beginning of the application, it is possible for the following sequence of events to occur.

1. External reset is held low, holding the device in reset.
2. External reset is released, allowing the device to run.
3. Device begins executing application code from program flash memory.
4. The external bus master begins 1-Wire communications to attempt to enter the loader/debugger.
5. The application code sets SC.TAP to 0, disabling the 1-Wire.
6. 1-Wire communications are interrupted; the master can no longer communicate with the 1-Wire/TAP.

This means that if the application disables the 1-Wire/TAP quickly enough, it may no longer be possible to 'recover' the device to erase it, debug code or load new application code, as the application has locked out this facility. In this sense, it has a similar effect to that of the Permanent Loader Lockout function, but in this case the effect may be unintended.

To avoid this possible sequence of events, the application should delay for a minimum amount of time (at least 5 seconds) to allow an external PC-based 1-Wire master time to enter bootloader mode as needed, before it disables the 1-Wire/TAP. The application can continue other operations, but should not set SC.TAP to 0 until this minimum timeout has elapsed. If future access to the device over the 1-Wire is not a concern (the application loaded into the part will never be altered), then this guideline may be disregarded once application development has completed.

The hardware- and ROM-based 1-Wire bootloader allows the internal flash program memory to be loaded by an external master over the 1-Wire/TAP port. This bootloader can be used in conjunction with the 1-Wire/TAP-based debugger facilities to allow IDE software (such as MAX-IDE or IAR Embedded Workbench) to erase the device, load new application code, and then perform debugging operations as part of the application development cycle.

28.3 – 1-Wire Interface Layers

The general timing and communications protocols used by the 1-Wire slave interface are those standardized for the 1-Wire network with exceptions as noted in that section.
Since the 1-Wire interface is a slave interface, all communications with it are initiated and timed by the 1-Wire master.

As discussed in the Book of iButton Standards, the 1-Wire communications protocol can be described in terms of the standard ISO Open System Interconnection (OSI) layered network model. Network layers which apply to this description are the Physical, Link, Network and Transport layers. The Presentation layer would correspond to higher-level application software functions (such as library layers) which implement communications protocols using the 1-Wire layers as a basic foundation.

28.4 – Bus Interface (Physical Layer)

As described in the Book of iButton Standards, the 1-Wire communications bus consists of a single data/power line plus ground. Devices (either master or slave) which interface to the 1-Wire communications bus do so using an open drain (active low) connection, which means that the 1-Wire bus normally idles in a high state. An external pullup resistor pulls the 1-Wire line high when no master or slave device is driving the line, which means that 1-Wire devices do not actively drive the 1-Wire line high; instead, they either drive the line low or release it (set outputs to high impedance) to allow the external resistor to pull the line high. This allows the 1-Wire bus to operate in a wired-OR manner and avoids bus contention if more than one device attempts to drive the 1-Wire bus at the same time.

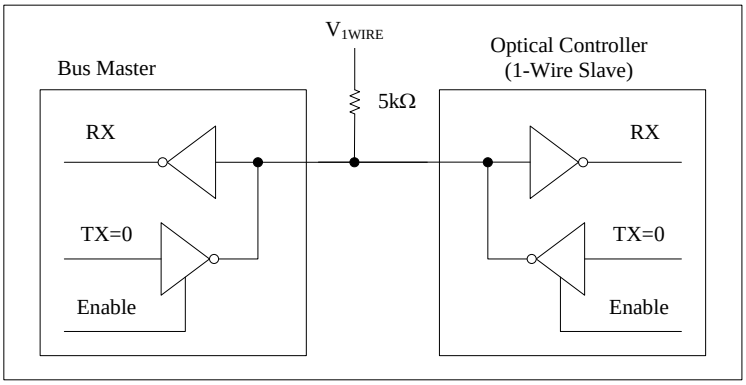


Figure 28-1. 1-Wire Bus Driver Interface

28.4.1 - Reset, Presence Detect and Data Transfer (Link Layer)

The 1-Wire Bus supports a single master and one or more slave devices (multidrop). Slave devices may connect to and disconnect from the 1-Wire Bus dynamically (as is typically the case with iButtons that operate using an intermittent touch contact interface), which means that it is the master's responsibility to poll the bus as needed to determine the number and types of 1-Wire devices that are connected to the bus.

All communications sequences on the 1-Wire bus are initiated by the master device. The master determines when a 1-Wire time slot begins, as well as the overall communications speed that will be used. There are two different communications speeds supported by the official 1-Wire specifications – standard speed and overdrive speed. However, only overdrive speed is supported by this 1-Wire slave interface.

28.4.2 - Reset and Presence Detect

When a 1-Wire device is first connected to the bus, it generates a presence pulse to let the bus master know that it is present on the line and waiting for communication from the master.

The 1-Wire master begins each communications sequence by sending a reset pulse. This pulse resets all 1-Wire slave devices on the line to their initial states and causes them all to begin monitoring the line for a ROM-layer command from the 1-Wire master. Each 1-Wire device on the line responds to the reset pulse by sending out a presence pulse; these pulses from multiple 1-Wire slave devices are combined in wired-OR fashion, resulting in a pulse whose length is determined by the slowest 1-Wire slave device on the bus.

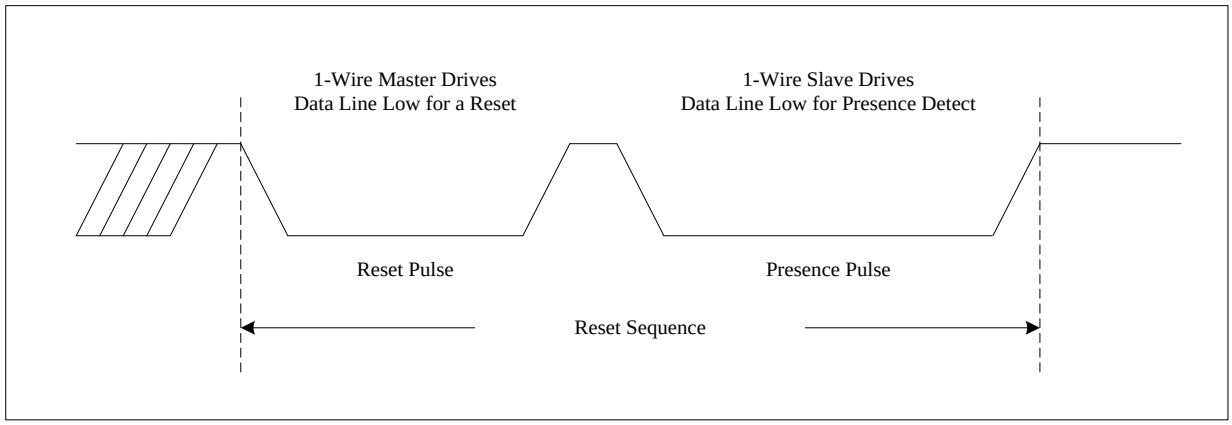


Figure 28-2. 1-Wire Reset Sequence

In general, the 1-Wire line must idle in a high state when communications are not taking place. It is possible for the master to pause communications in between time slots; there is no overall 'time-out' period that will cause a slave to revert to the reset state if the master takes too long between one time slot and the next.

The 1-Wire communications protocol relies on the fact that the maximum allowable length for a bit transfer (write 0/1 or read bit) time slot is less than the minimum length for a 1-Wire reset. At any time, if the 1-Wire line is held low (by the master or by any slave device) for more than the minimum reset pulse time, all slave devices on the line will interpret this as a 1-Wire reset pulse.

28.4.3 - Writing Bits (From Master, to Slave)

All 1-Wire bit time slots are initiated by the 1-Wire bus master and begin with a single falling edge. There is no indication given by the beginning of a time slot whether a read bit or write bit operation is intended, as the time slots all begin in the same manner. Rather, the 1-Wire command protocol enforces agreement between the 1-Wire master and slave as to which time slots are used for bit writes and which time slots are used for bit reads.

When multiple bits of a value are transmitted (or read) in sequence, the least significant bit of the value is always sent or received first. The 1-Wire bus is a half-duplex bus, so data may be transmitted in only one direction (from master to slave or from slave to master) at any given time.

As can be seen below in Figure 28-3, the time slots for writing a zero bit and writing a one bit begin identically, with the falling edge and a minimum-width low pulse sent by the master. To write a one bit, the master releases the line after the minimum low pulse, allowing it to be pulled high. To write a zero bit, the master continues to hold the line low until the end of the time slot.

From the slave's point of view, the initial falling edge of the time slot triggers the start of an internal timer, and when the proper amount of time has passed, the slave samples the 1-Wire line which is being driven by the master. This sampling point is in between the end of the minimum-width low pulse and the end of the time slot.

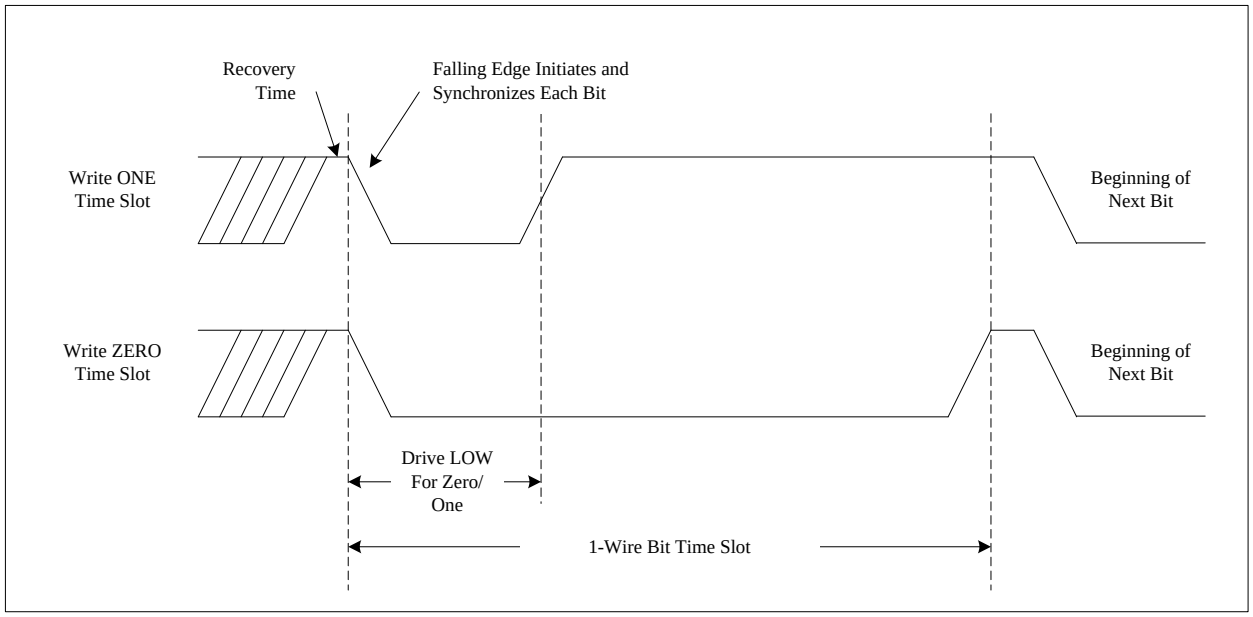


Figure 28-3. 1-Wire Write Bit Sequence

28.4.4 - Reading Bits (From Slave, to Master)

As with all 1-Wire transactions, the master initiates all bit read time slots. Like the bit write time slots, the time slot for a bit read begins with a falling edge. From the master's point of view, this time slot is transmitted identically to the "Write 1 Bit" time slot shown above. The master begins by transmitting a falling edge, holds the line low for a minimum-width period, and then releases the line.

The difference here is that instead of the slave sampling the line, the slave begins transmitting either a 0 (by holding the line low) or a 1 (by leaving the line to float high) after the initial falling edge. The master then samples the line to read the bit value that is being transmitted by the slave device.

As an example, Figure 28-4 shows a read sequence in which the slave devices transmit data back to the 1-Wire bus master upon request. Note that to transmit a 1 bit, the slave device does not need to do anything; it simply leaves the line alone (to float high) and waits for the next time slot. To transmit a 0 bit, the slave device holds the line low until the end of the time slot.

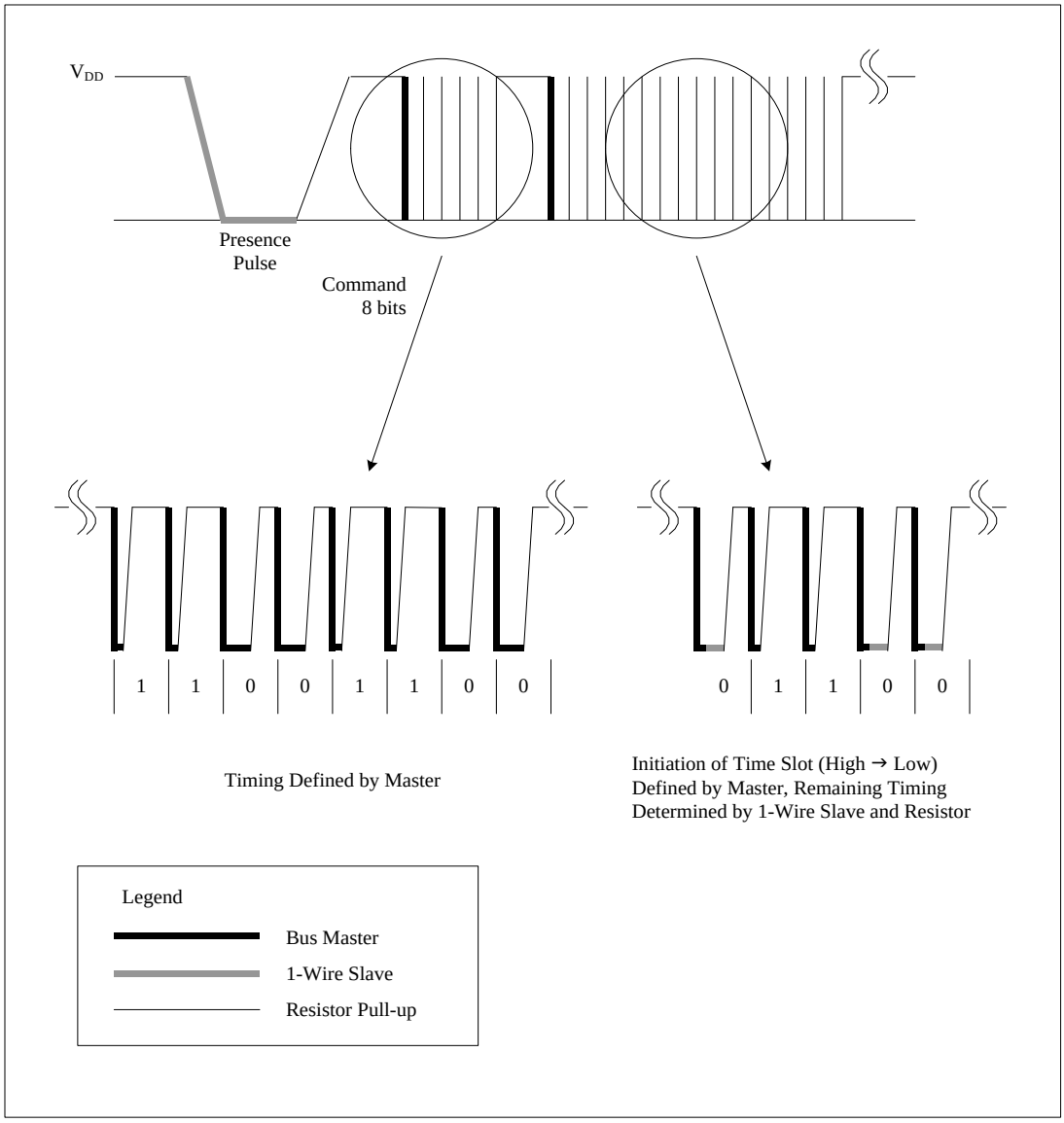


Figure 28-4. 1-Wire Read Sequence

28.4.5 - Standard Speed and Overdrive Speed

Unlike standard 1-Wire devices, the DS4835/36 1-Wire slave interface does not support standard speed communications and operates at overdrive speed only. This means that communications with the 1-Wire slave should begin with an overdrive-speed reset pulse, followed by one of the ROM layer commands (Overdrive Skip ROM, Overdrive Match ROM, or Search ROM) sent entirely at overdrive speed. Figures 28-5 through 28-8 show the timing required for the 1-Wire overdrive transactions.

Even if a standard speed reset pulse is sent (which will trigger a full chip reset if the pulse is longer than the minimum external reset pulse width), the communications speed will remain at overdrive.

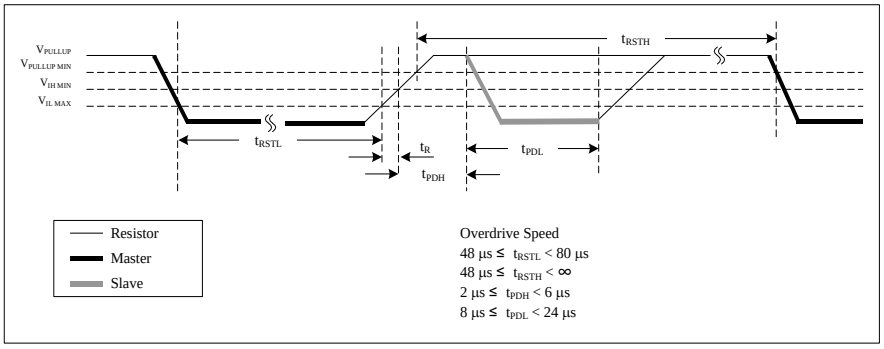


Figure 28-5. 1-Wire Reset Sequence Timing

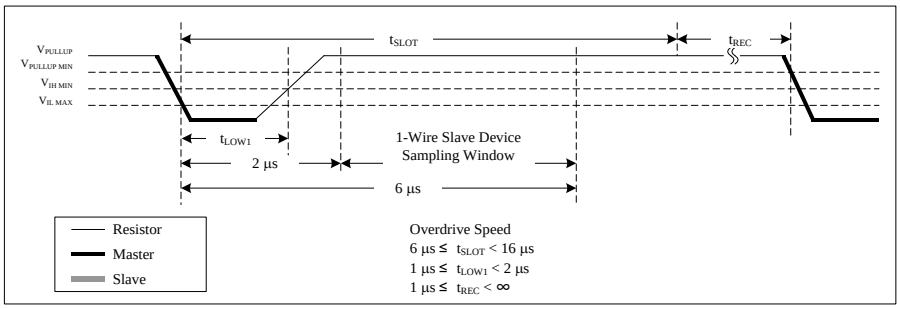


Figure 28-6. 1-Wire Write One Sequence Timing

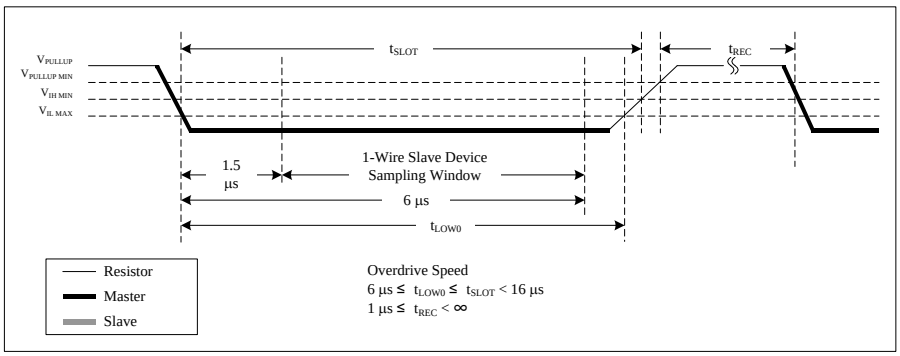


Figure 28-7. 1-Wire Write Zero Sequence Timing

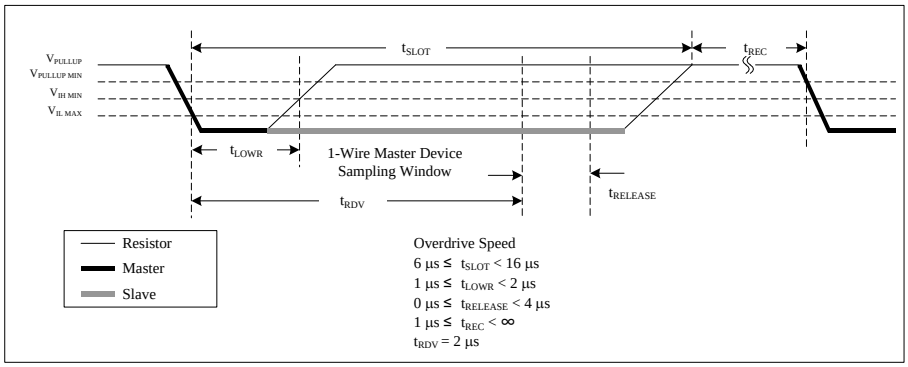


Figure 28-8. 1-Wire Read One/Zero Sequence Timing

28.5 – ROM Commands (Network Layer)

Following the initial 1-Wire reset pulse on the bus, all slave 1-Wire devices are active, which means that they are monitoring the bus for commands. Since a 1-Wire bus may have multiple slave devices present on the bus at any time, the 1-Wire master must go through a process (defined by the 1-Wire command protocol) to activate only the 1-Wire slave device that it intends to communicate with and deactivate all others. This is accomplished using the device's unique ROM ID. This is the purpose of the commands described in the network layer of the 1-Wire protocol, also known as the ROM layer commands.

In an optical system, it is assumed that there will be only one DS4835/36 on the 1-Wire bus, and no other 1-Wire slave devices will be attached. The DS4835/36 does not implement a unique ROM ID for this reason. To access the DS4835/36 1-Wire, a ROM command must still be used, but the only command supported by the DS4835/36 is the Skip ROM command.

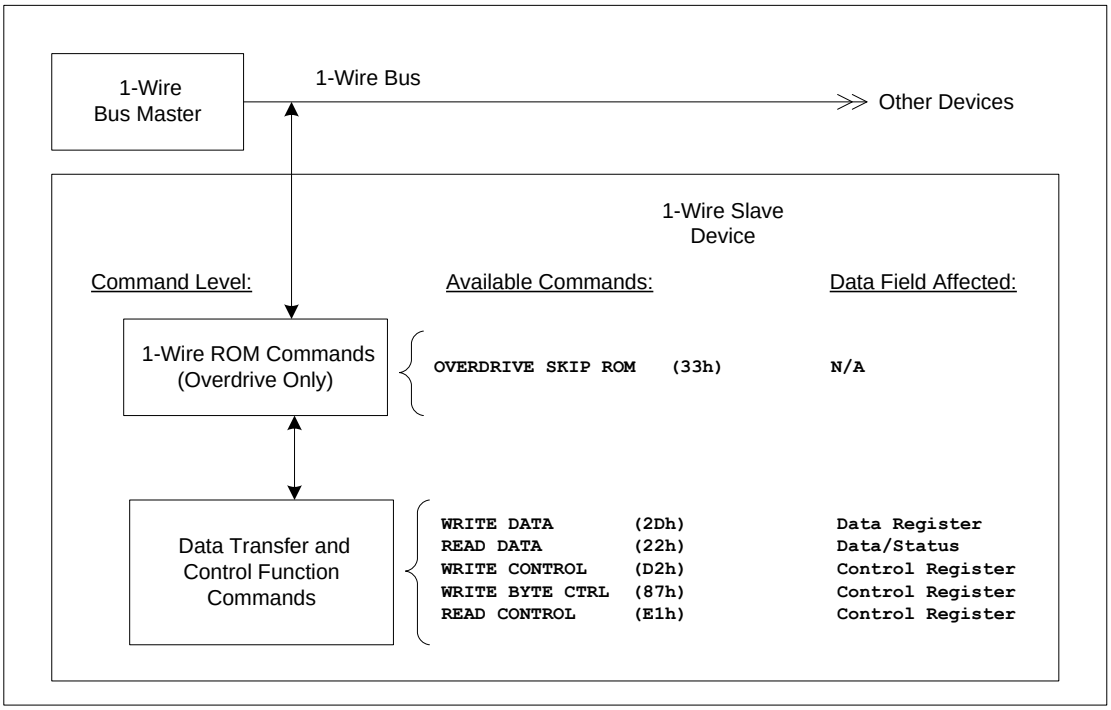


Figure 28-9. 1-Wire Command Set

28.5.1 - Overdrive Skip ROM Command [3Ch]

The Overdrive Skip ROM command activates all 1-Wire slave devices present on the 1-Wire bus, regardless of their ROM ID. Normally, this command is used when only a single 1-Wire slave device is connected to the bus. After the Overdrive Skip ROM command has completed, all slave devices on the 1-Wire bus are selected or "active", and communications will proceed to the Transport layer.

28.6 – Data Transfer Commands (Transport Layer)

While the commands in the ROM layer are (by design) generally shared across all devices that support the 1-Wire protocol, the command set supported by the Transport layer is much more device specific. For the DS4835/36 1-Wire slave interface, the Transport layer command set is designed to allow data to be transferred back and forth between the 1-Wire slave interface and the 1-Wire bus master from three registers: Control, Status and Data. This is shown in Figure 28.10. Figure 28.11 shows additional detail on the commands used by the 1-Wire interface to access these three registers.

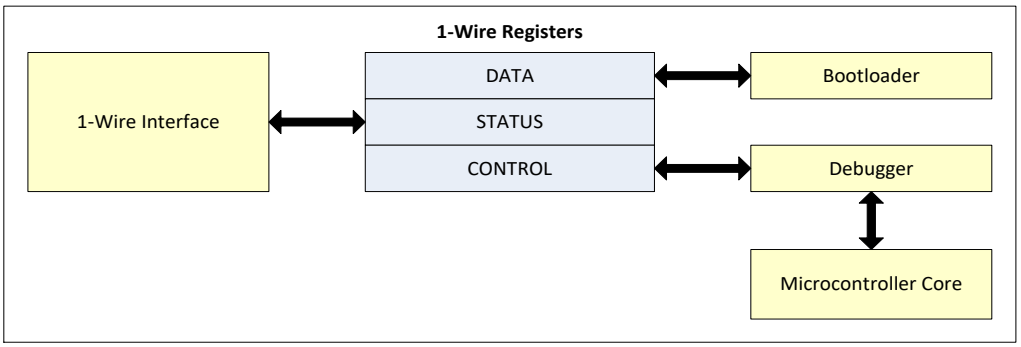


Figure 28-10. 1-Wire Registers

The Control register allows the 1-Wire bus master to set certain operating mode bits through the TAP. For example, this register controls whether the part goes into normal mode or bootloader mode following reset. This register is also used to control entry into debug mode.

Note that unlike most standard 1-Wire parts, completing a transaction at the Transport Layer does not automatically send the 1-Wire engine back to the starting state (where it would wait for a 1-Wire reset pulse to begin a new transaction). Instead, the part remains at the Transport layer if a 1-Wire bus reset does not occur. What this means is that multiple Transport layer commands can be performed back to back, without the requirement to reset and reselect the part each time (by Skip ROM command). This improves overall throughput when communicating with the 1-Wire TAP.

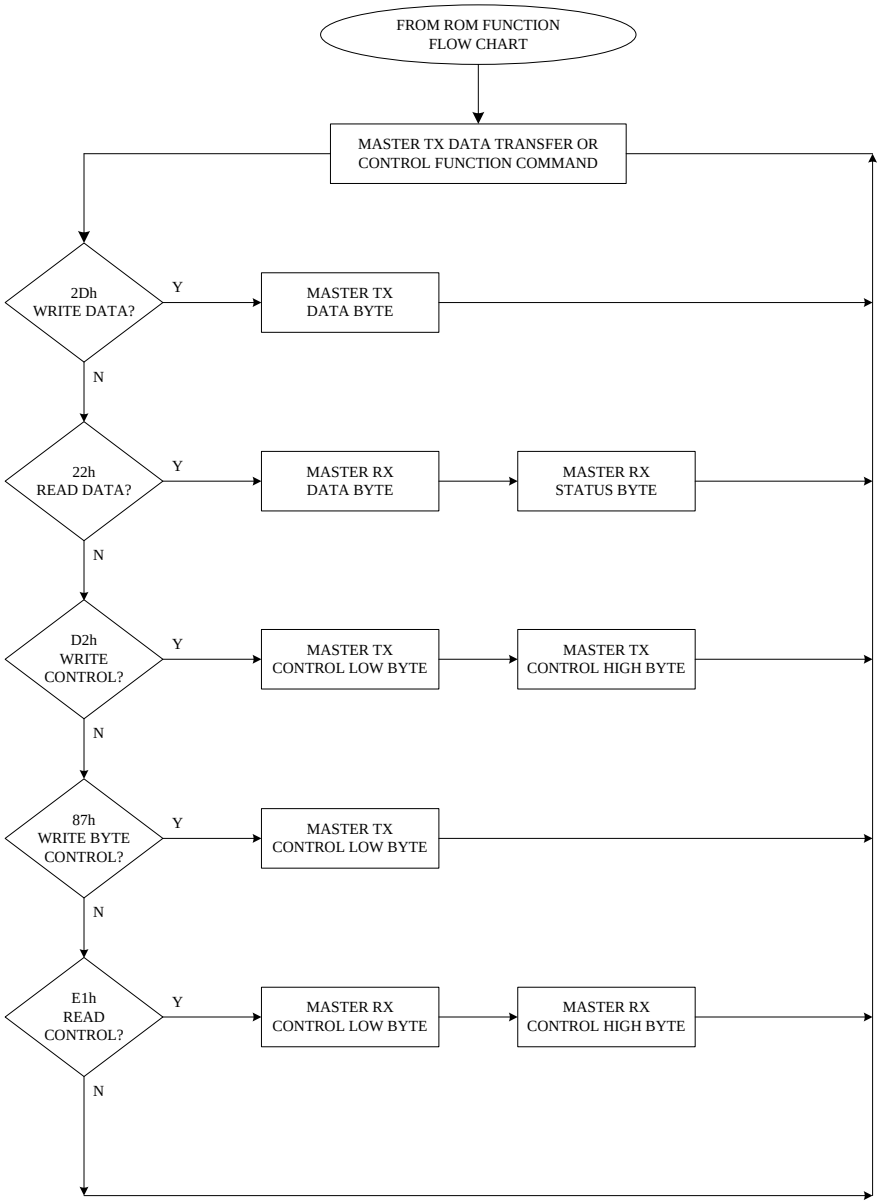


Figure 28-10. Transport Layer Commands

28.6.1 - Write Data Command [2Dh]

This command sends data from the 1-Wire bus master to the 1-Wire bootloader or debugger, once one of these modes has been successfully entered. Data is transmitted one byte at a time. The Read Data command must be used to determine whether the bootloader is ready to accept the next data byte from the 1-Wire master.

Write Data Command Details

Byte	Written by Master (Read by Slave)	Written by Slave (Read by Master)
0	2Dh (command code)	--
1	Data byte to write to bootloader	

28.6.2 - Read Data Command [22h]

This command is used by the master to read data from the 1-Wire bootloader, once bootloader or debugger mode has been successfully entered. Two bytes are returned by this command, which represent the current values of the Data and Status registers. The Status register must be examined by the 1-Wire master to determine if the Data register byte contains valid data, or if the bootloader is still busy processing a command or reading a previously sent byte.

Read Data Command Details

Byte	Written by Master (Read by Slave)	Written by Slave (Read by Master)
0	22h (command code)	--
1	--	Data byte read from bootloader
2	--	Status register value

28.6.3 - Write Control Command [D2h]

This command is used by the 1-Wire master to write a new value to the Control register in the 1-Wire TAP. This command writes new values to both the high byte and the low byte of the Control register.

Write Control Command Details

Byte	Written by Master (Read by Slave)	Written by Slave (Read by Master)
0	D2h (command code)	--
1	New value for Control register – low byte	--
2	New value for Control register – high byte	--

28.6.4 - Write Byte Control Command [87h]

This command is used by the 1-Wire master to write a new low byte value to the Control register in the 1-Wire TAP. This command modifies the low byte only of the Control register, leaving the high 8 bits unchanged.

Write Byte Control Command Details

Byte	Written by Master (Read by Slave)	Written by Slave (Read by Master)
0	D2h (command code)	--
1	New value for Control register – low byte	--

28.6.5 - Read Control Command [E1h]

This command is used by the master to read back the current value of the Control register (low byte and high byte).

Read Control Command Details

Byte	Written by Master (Read by Slave)	Written by Slave (Read by Master)
0	E1h (command code)	--
1	--	Control register – low byte
2	--	Control register – high byte

28.7 – TAP Interface Layer

This section covers the functionality of the TAP interface layer registers which bring the device in and out of bootloader mode and communicate with the 1-Wire bootloader.

Note: These registers are not accessible directly from the CPU – they can be read and written through the 1-Wire interface only. As such, they do not have module[index]-style register addresses.

1-Wire TAP Data Register

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED								DATA[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	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

BIT	NAME	DESCRIPTION
15:8	RESERVED	Reserved. The bits return 0.
7:0	DATA[7:0]	1-Wire TAP Data. Reading from and writing to this register through the 1-Wire interface have different effects. In either case, bootloader or debug mode must be active for valid data to be written or read. Reading from this register returns the most recent byte (if any) that was sent by the 1-Wire bootloader or debugger. Since the bootloader or debugger do not initiate commands, this will always be in response to a command sent by the master. The status register (which is always read when this register is read using the Read Data command) can be examined to determine if the value returned for this register is valid. Writing to this register sends a byte of data to the 1-Wire bootloader or debugger. The RXC bit in the Status register must be monitored to determine whether the 1-Wire bootloader is ready to accept the next data byte.

1-Wire TAP Status Register

Bit	15:5	4	3:2	1:0
Name	RESERVED	RXC	RESERVED	DB STAT
Reset	0	0	0	0
Access	r	r	r	r

BIT	NAME	DESCRIPTION
15:4	RESERVED	Reserved. The bits return 0.
4	RXC	RXC Status Bit. This bit is set to 1 by hardware when the 1-Wire master writes a data byte to the bootloader using the "Write Data" command. When the bootloader reads the data byte, this bit is automatically cleared, indicating that the 1-Wire master may now send another byte.
3:2	RESERVED	Reserved. The bits return 0.
1:0	DB STAT[1:0]	**Debug Engine Status.** These bits indicate the current status of the debug or bootloader block which transfers data to and from the 1-Wire bootloader. These bits can be examined in conjunction with the RXC bit to determine whether the bootloader or debugger is ready to transmit or receive the next data byte. These bits have slightly different meaning based upon if operating in bootloader or debug mode. Refer to these sections of the user's guide for more details on the meaning of these bits.

1-Wire TAP Control Register

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	RESERVED							RST	RES	TLR	RES		DEN	RES		SPE
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Access	r	r	r	r	r	r	r	rw	r	rw	r	r	rw	r	r	rw

BIT	NAME	DESCRIPTION
15:9	RESERVED	Reserved. These bits should be set to 0.
8	RST	Internal Reset. Since the reset function and the OWL line are shared, this bit provides a way to force the part into the equivalent of an external reset without affecting the state of the 1-Wire line. Setting this bit to 1 will hold the part in reset, just as if OWL/RST were being held low. Clearing this bit to 0 will release the part from reset.
7	RESERVED	Reserved. This bit should be set to 0.
6	TLR	Test Logic Reset. Setting this bit to 1 resets the TAP and the debug engine block. It has no effect on other 1-Wire logic/registers, or any system or peripheral registers. The TAP will remain in reset if this bit is set to 1. Note that since this bit defaults to 1 following reset, it must be cleared to 0 before bootloader mode can be entered.
5:4	RESERVED	Reserved. These bits should be set to 0.
3	DEN	Debug Engine Enable. Setting this bit to 1 enables communications through the debug engine block by read/write operations to the Data register. This bit must be set to 1 before bootloader or debug mode can be entered.
2:1	RESERVED	Reserved. These bits should be set to 0.
0	SPE	System Programming Enable. This bit controls entry of the part into bootloader mode. When SPE=0 (default mode), the CPU will vector to the beginning of application code (at flash address 0000h) following a reset. When SPE=1, the CPU will vector to the bootloader routine in the utility ROM following reset.

SECTION 29 – IN-CIRCUIT DEBUG MODE

The DS4835/36 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 1-Wire/RST as its means of communication between the host and the DS4835/36. Figure 29-1 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.

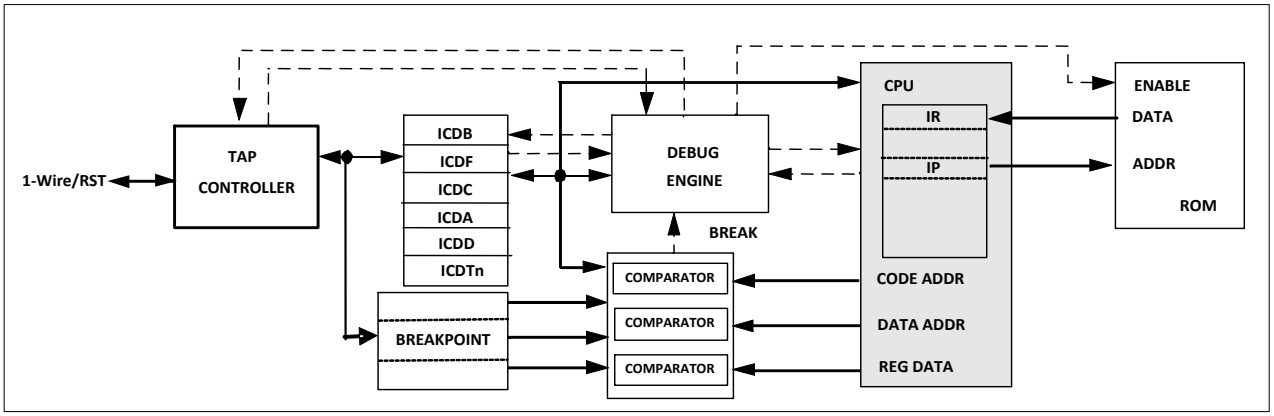


Figure 29-1: In-Circuit Debugger

The embedded hardware debug engine is implemented as a stand-alone hardware block in the DS4835/36. 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.

The debug engine is disabled after power-on reset and will remain inactive until the CONTROL register is written with the "WRITE CONTROL" RAM command to set the DEN bit and clear the TLR bit. The debug engine defaults to background mode if the debug engine is enabled. The debug engine is a slave device, only the host can initiate a transfer cycle via the 1-Wire Loader. To provide handshaking between the host and the debug engine, the two status bits and RXC bit are used to signify the following conditions.

RXC	DBSTAT[1:0]	Status	Description
X	00	Non-Debug	Default condition, Background mode, or debug engine inactive
X	11	Debug-Idle	Debug engine is ready to receive data from the host (command, data, or follow-on data bytes)
X	10	Debug-Busy	Debug engine is busy without valid data (i.e. ROM code execution or trace operation)
X	11	Debug-Valid	Debug engine is busy with valid data
1	XX	Debug-Wait	The Debug engine has received a new data byte but has not yet consumed it. Host must wait until RXC has cleared before writing another data byte. This bit is only valid in debug mode when using the password unlock command.

These status bits are controlled directly by the debug engine. A non-debug state will be shifted out when the debug engine is disabled or is in background mode. The debug engine outputs the debug-idle if a Debug command is received in background mode or a match occurs for a breakpoint. The status reflects whether the last command (or data) was accepted by the debug engine. To ensure proper communication, the host should check the debug-idle before issuing any debug mode commands or new data. The debug engine issues a debug-busy when it is executing a debug operation after receiving a valid command and/or correct data stream. The debug-valid state is used when presenting the valid data. Command or data can be written using the "WRITE DATA" command during debug-busy or debug-valid, but the host must ensure the return to debug-idle to ensure the last transfer was successful.

29.1 – Background Mode Operation

When the DS4835/36 in operating in background mode, the host can communicate with the TAP without disturbing CPU operation. Note, however, that 1-Wire in-system programming also requires use of the 1-Wire controller and, if enabled (SPE=1, PSS1:0= 0), in-system programming 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 DS4835/36 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 29-2 shows the background mode commands supported by the DS4835/36. Encodings not listed in this table are not supported in background mode and are treated as no operations.

Table 29-2. Background Mode Commands

Op Code	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.

29.1.1 – Breakpoint Registers

The DS4835/36 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.

29.1.1.1 – 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.

29.1.1.2 – 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.

29.1.1.3 – 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.

29.1.1.4 – 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.

29.1.1.5 – 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

29.1.1.6 – 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

29.1.2 – 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 DS4835/36 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 DS4835/36 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.

29.2 – 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 as the interrupt handler is also halted because of IGE = 0.

29.2.1 – 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 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 29-3 shows the debug mode commands supported by the DS4835/36. 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 29-3. 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.

Opcode	Command	Operation
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 DS4835/36. Writing to the IP register alters the address that execution resumes from when the debugging engine exits.
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).

29.2.2 – 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 M6[32h], 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.
- Nonimplemented and reserved peripheral registers in the range are represented as empty word values in Table 29-4. These values should be ignored.

The first byte output by this command is the value 224d, which represents the number of words output for peripheral registers. There are a total of 256 words that are output by this command. Table 29-4 lists all the registers output and the order in which they are output.

Important Notes:

- Registers I2CBUF_S, MPTNR, HTI, DAD2W1, SPIB and IDCD3 are not read by this routine. The values returned by these registers is 0000h.
- The IC.IGE bit will always read back 0. When in debug mode, this bit is set to 0 to prevent interrupts from interfering with the debugger. This bit will be set back to the prior value when exiting the debugger.
- The ADDATA register will be read by this routine. This will result in ADCN.INDEX incrementing at every debug mode breakpoint when the registers are read or when code is stepped.

Table 29-4. Output from Read Register Map Command

WORD	REGISTER	WORD	REGISTER	WORD	REGISTER	WORD	REGISTER	WORD	REGISTER	WORD	REGISTER	WORD	REGISTER	WORD	REGISTER	
0	PO2	32		64	I2CBUF_S	96	MCNT	128	ADCN	160	DAD2W1	192		224	AP	APC
1	PO1	33	TECD1	65	I2CST_S	97	MA	129	ADST	161	DAD2W2	193	PICNT	225	PSF	IC
2	PO0	34	TECD2	66	MPNTR	98	MB	130	ADDATA	162	IDCD1	194	PIDATA	226	IMR	SC
3	EIF2	35	PO6	67	I2CTXFST	99	MC2	131		163	IDCD2	195	DPDATA	227	IIR	CKCN
4	EIF1	36	CRCIN	68	I2CTXFIE	100	MC1	132	HTI	164	SPIB	196	APDDAT1	228	WDCN	00h
5	EIF0	37		69	I2CRXFST	101	MC0	133		165	IDCD3	197	APDDAT2	229	A[0]	
6	TBV1	38	EIF6	70	I2CRXFIE	102	TBCN3	134	LTI	166	IDCD4	198	APDDAT3	230	A[1]	
7	TBCN1	39	EIE6	71	I2CST2_S	103	SHFT	135		167	DADDR	199	APDDAT4	231	A[2]	
8	PI2	40	PI6	72	RPNTR	104	MC1R	136	TEMPCN	168		200	DCDC_SEL	231	A[3]	
9	PI1	41	SVM	73	I2CCN_S	105	MC0R	137	SH0_DATA	169	MTSPEC1	201	DCCN	233	A[4]	
10	PI0	42	FCNTL	74	I2CCN2_S	106	TBC3	138		170	CNT2W1	202	VRDCDC	234	A[5]	
11	TBC1	43	FDATA	75	GP_REG	107	TBV3	139		171	CNT2W1	203	IZDAC	235	A6[]	
12	TBV2	44		76	I2CSLA_S	108	APCDAT2	140	SHCN	172	MTSPEC2	204	PMOS_ON	236	A[7]	
13	EIE2	45	TECCN1	77	I2CSLA2_S	109	APCDAT3	141	SH1_DATA	173		205	NMOS_ON	237	A[8]	
14	EIE1	46	TECCN2	78	I2CSLA3_S	110	MACSEL	142	ITEMP_DATA	174	TWR	206	PMOS_ON_ST	238	A[9]	
15	EIE0	47		79	I2CSLA4_S	111	APCDAT1	143		175		207	NMOS_ON_ST	239	A[10]	
16	PD2	48	EIES6	80	I2CIE2_S	112	APCCN1	144		176	SLA2W1	208		240	A[11]	
17	PD1	49	PD6	81	MADDR	113	APCIDX1	145		177	SLA2W2	209		241	A[12]	
18	PD0	50		82	MADDR2	114	APCDAT4	146		178	IDCN1	210	DCDC_OPT	242	A[13]	
19	EIES2	51		83	MADDR3	115		147	SPB	179	IDCN2	211		243	A[14]	
20	EIES1	52		84	MADDR4	116	APCCN2	148	DEV_NUM	180	SPICK	212	APDCN1	244	A[15]	
21	EIES0	53	CRCOUT	85	CUR_SLA	117	APCIDX2	149		181	SPICN	213	APDIDX1	245	IP	
22	PLACNT1	54	GP_REG	86	I2CIE_S	118	APCCN3	150	IEN0	182	SPICF	214		246	SP	
23	PLADAT1	55		87	BRKPNT	119	APCIDX3	151	IEN1	183	IDCN3	215	APDCN2	247	IV	
24	PLACNT2	56		88	ICDT0	120	APCCN4	152	IEN2	184	IDCN4	216	APDIDX2	248	LC[0]	
25	PLADAT2	57		89	ICDT1	121	APCIDX4	153	IEN6	185		217		249	LC[1]	
26	PLACNT3	58		90	ICDC	122		154	INRSH	186	MIS1	218	APDCN3	250	OFFS	
27	PLADAT3	59		91	ICDF	123		155		187	MIS2	219	APDIDX3	251	DPC	
28	PLACNT4	60		92	ICDB	124		156		188	CCK2W1	220		252	GR	
29	PLADAT4	61		93	ICDA	125		157		189	CCK2W2	221	APDCN4	253	BP	
30	TBCN2	62	NORMV	94	ICDD	126	APCOPT	158	ETEMP_DATA	190		222	APDIDX4	254	DP[0]	
31	TBC2	63		95	I2CBUF_S	127	MCNT	159	IV2	191		223	DPCN	255	DP[1]	

29.2.3 – 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.

29.2.4 – 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.

29.2.5 – 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. If the write register command is used and invokes a reset of the device, the debug engine will 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.

29.3 – In-Circuit Debug Peripheral Registers

The following peripheral registers are used to control the in-circuit debug mode of the DS4835/36. 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.

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.

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.

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.

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

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.

In-Circuit Debug Flag Register (ICDF, M2[1Bh])

Bit	7	6	5	4	3	2	1	0
Name	-	-	RXC	-	PSS1	PSS0	SPE	TXC
Reset	0	0	0	0	0	0	0	0
Access	r	r	rw	r	rw	rw	rw	rw

r = read, s = special

BIT	NAME	DESCRIPTION
7:6	Reserved	Reserved. Do not write to these bits.
5	RXC	Receive Complete: This bit is set when the 1-Wire host sends a byte into the DATA register. This data will then be copied to the ICDB register. This bit will be cleared when the device ROM reads the ICDB register. This bit will be copied to the 1-Wire Status register when the master reads status.
4	Reserved	Reserved. Do not write to this bit..
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 SPE is set to 1, otherwise, the logic values of these bits have no meaning:
1	SPE	System Program Enable. The 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 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 1-Wire). 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 SPE bit will be cleared by hardware when the ROD bit is set. CPU writes to the SPE bit (0 or 1) will result in clearing of the PSS[1:0] bits.
0	TXC	Transmit Complete: 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.

PSS1	PSS0	Interface/Action
0	0	1-Wire
0	1	I^2C Bootloader
1	x	Exit Loader

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 30 – IN-SYSTEM PROGRAMMING

The DS4835/36 contains an internal bootstrap loader utilizing the 1-Wire 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, DS4835/36 ROM code is executed which determines if bootloader operation is desired. Figure 30-1 provides information on how the DS4835/36 enters bootloader operation.

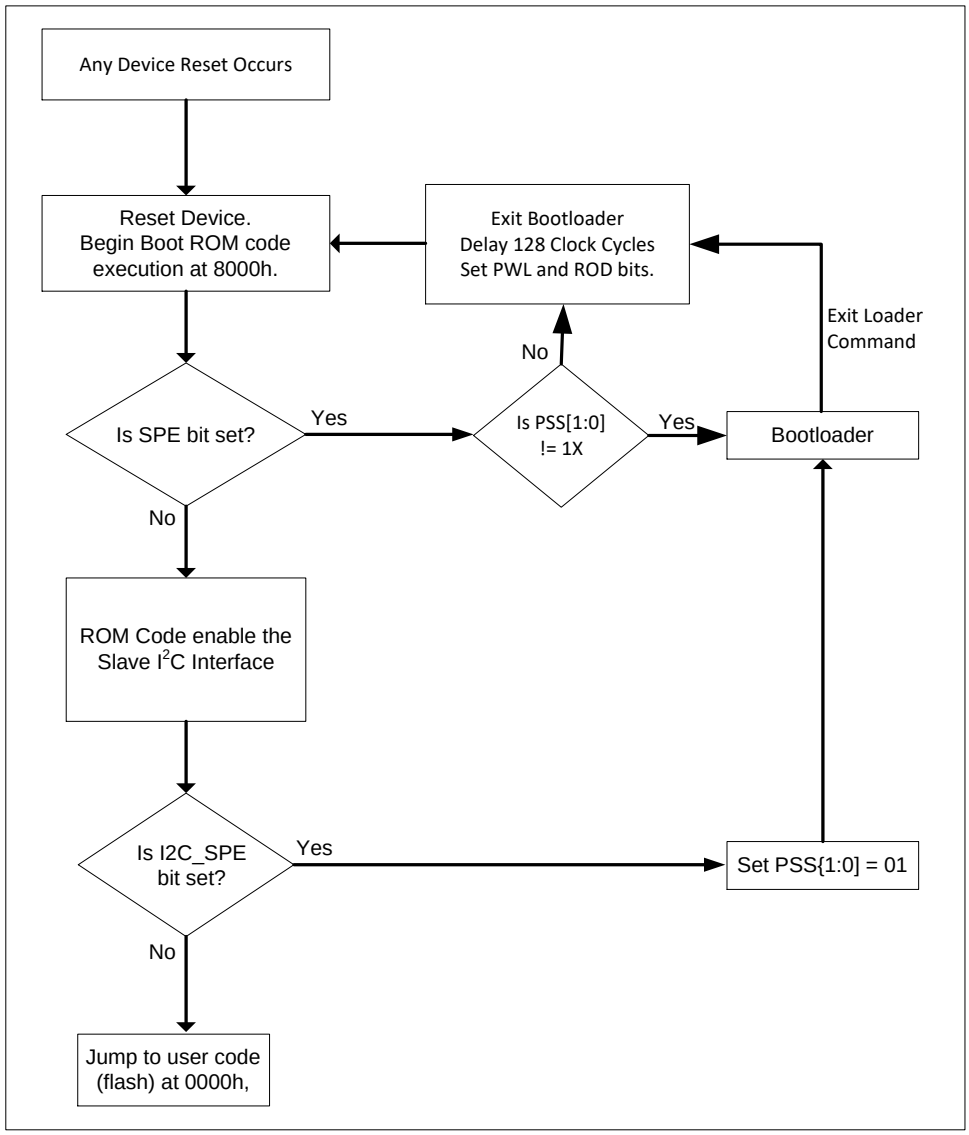


Figure 30-1: Entering Bootloader Operation

30.1 – Detailed Description

Following every reset, device ROM code is executed which determines if the DS4835/36 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 1-Wire Bootloader section for more details on setting the SPE bit. If SPE is set, the DS4835/36 enters bootloader operation.

If SPE is not set, the DS4835/36 then enables the slave I^2C interface. The I2C_SPE bit in the SPB register is read to determine if I^2C bootloader operation is desired. The SPB register is not cleared by a reset. See the Entering I^2C Bootloader section for more details on setting the I2C_SPE bit. If I2C_SPE is set, the DS4835/36 will set the PSS[1:0] bits to 01, which designates I^2C bootloader, and enter bootloader operation.

If none of the preceding conditions have been met, the DS4835/36 ROM code will be complete. The DS4835/36 will then jump to program memory location 0000h and begin normal program execution.

30.1.1 - Password Protection

The DS4835/36 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 DS4835/36 contains a valid password.

To protect the program memory, the DS4835/36 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 1-Wire or I^2C interface.

The bootloader master erase command can be used if the PWL bit is set. When this command is sent, the entire flash memory will be erased. At the completion of this, the password will be all FFFFh and the PWL bit will be cleared to 0.

30.1.1 – Entering 1-Wire Bootloader Mode

The procedure to enter bootloader mode through the 1-Wire interface is as follows.

1. Send a 1-Wire overdrive speed reset pulse. The device should respond with a presence pulse.
2. Transmit the Overdrive Skip ROM command (3Ch). (This assumes that this microcontroller is the only device on the 1-Wire bus).
3. Use the Write Control command to set the Control register to 0149h (transmit D2h 49h 01h). This sets the RST bit, holding the part in internal reset, and sets SPE and DEN to 1 to enable communication in loader mode.
4. Delay for a brief period (perhaps 1ms) to allow the reset to take effect.
5. Use the Write Control command to set the Control register to 0009h (transmit D2h 09h 00h). This clears the internal reset, activates the debug engine block and TAP, and releases the part to begin executing the bootloader routine in the utility ROM.
6. Delay for a slightly longer period (perhaps 10ms) to give the bootloader time to initialize.
7. Begin transmitting/receiving data to and from the bootloader using the Write Data and Read Data/Status commands.

30.1.3 – Entering I²C Bootloader

The DS4835/36 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 1-Wire 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 DS4835/36 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 30-1 details the special functions that can be performed using slave 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 DS4835/36 when an I²C Stop is received.
All other bytes	The I2C_SPE bit in I2C_SPB is cleared. The DS4835/36 will NACK this byte.

Table 30-1. Special Functions of Address 34h

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 DS4835/36 must then be reset. This can be done using either the 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.

30.1.4 –I²C Bootloader disable

The DS4835/36 provides options to disable the bootloader slave address 34h. The device has DEV_NUM register which is cleared only on POR. Bit 7 of the DEV_NUM controls bootloader slave address. Setting DEV_NUM[7] disables the slave address. Applications in which bootloader address is not required should set the DEV_NUM[7] to diable this slave address.

30.2 – Bootloader Operation

Once in bootloader mode, the 1-Wire and I²C interfaces both use the same commands. How these commands are implemented will be different between the two interfaces. Following shows and example command and data bytes required. The next two sections will detail how to implement these commands using either the 1-Wire or I²C interface.

Byte(s)	Command	Data In	Data Out	Return
Input	Command	Data in	00h	00h
Output	X	X	Data Ojut	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.
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 1-Wire bootload mode, status bits will tell when ROM loader is sending valid 3Eh.

30.2.1 – 1-Wire Bootloader Protocol

When using the 1-Wire Boot Loader option (SPE=1, PSS[1:0] = 2'b00), the sole purpose of the debug hardware is to simultaneously transfer the data byte transferred from the host with the "WRITE DATA" RAM command to the ICDB register. The ROM Loader can then transfer a data byte to the host with the "READ DATA" RAM command. This command returns the STATUS byte as well as the DATA byte from the ROM Loader. Table 30-2 provides details on the different values returned by the status byte. The status and data byte are transferred from the emulation engine to the 1-Wire shift register upon reception of the "READ DATA" RAM command. The debug hardware additionally clears the TXC bit at this point to signify to the ROM Loader that the data byte has been consumed.

RXC	DBSTAT[1:0]	Status	Description
X	00	Reserved	Invalid Condition
X	11	Reserved	Invalid Condition
X	10	Loader-Busy	The bootloader is busy executing code or processing the current command. Disregard the value received for the Data register.
X	11	Loader-Valid	The Data register contained a valid byte sent by the bootloader.
1	XX	Loader-Wait	The bootloader has received a new data byte (sent by a Write Data command) but has not yet unloaded it from the register. The 1-Wire master must wait for RXC to clear to 0 before sending another byte.

Table 30-2. Status Bits for Loader Data Transfer

The procedure for transferring data in 1-Wire loader mode is slightly different depending on whether you are intending to transmit a byte and receive one in return, or if you are intending to transmit a byte without expecting a reply. For certain commands, the bootloader will transmit a reply byte for each byte it receives, and the following procedure can be used.

1. Use the Write Data command to send the transmit byte.
2. Use the Read Data command to read the receive byte (data) and status.
3. If the DBSTAT bits are anything other than 11b, ignore the received data byte and return to step 2 for another attempt at reading valid data.
4. If the DBSTAT bits are 11b, then the received data byte is valid.

For other commands, the bootloader expects to receive multiple bytes from the master and does not transmit any reply bytes until it has received and processed the entire command. For these situations, the following procedure can be used.

1. Use the Write Data command to send the transmit byte.
2. Use the Read Status command to read the receive byte and status.
3. If the RXC bit is 1, the loader has not yet processed the byte. Repeat step 2 until the RXC bit has been cleared. Disregard the byte of returned data from this step.
4. Use the Write Data command to send the next transmit byte.
5. Repeat steps 2 and 3 to until this byte has been processed.

30.2.2 – 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 DS4835/36, slave address 36h (R/W bit = 0) is used. To read data from the DS4835/36 I²C bootloader, slave address 37h (R/W bit = 1) is used. The I²C bootloader does not return the status bits that are available from the 1-Wire bootloader. The following I²C steps are required to send each command.

1) Send an I²C start, followed by writing slave address 36h (R/W bit set to write).
2) Write command byte.
3) Write any Data In bytes.
4) If readback is necessary, an I²C Restart needs to be issued, followed by writing slave address 37h (R/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 30-2. After receiving and acknowledging 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).

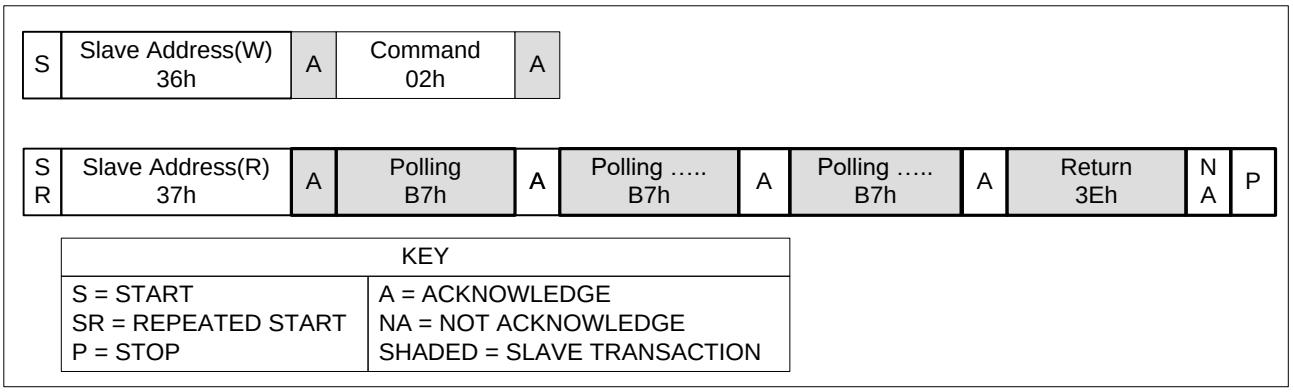


Figure 30-2: I²C Bootloader Polling

Refer to Application Note 5602: In-System Programming Using I²C Bootloader Commands for ISP using the I²C bootloader. This application note is defined for DS4830 however, this is applicable to DS4835/36 also**.**

30.3 – Bootloader Commands

Commands for the DS4835/36 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 30-3.

Table 30-3 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 I²C 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 DS4835/36 bootloader.

30.3.1 - 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.

30.3.2 - 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 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.

30.3.3 - Command 02h – Master Erase

	Byte 1	Byte 2
	Command	Return
Input	02h	X
Output	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.

30.3.4 - Command 03h – Password Match

	Byte 1	Bytes 2 to 33	Byte 34
	Command	Data In	Return
Input	03h	32-Byte Password	X
Output	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.

30.3.5 - Command 04h – Get Status

	Byte 1	Byte 2	Byte 3	Byte 4
	Command	**Data Out**	**Data Out**	**Return**
Input	04h	X	X	X
Output	X	Flags	Status Code	3Eh

The Status Flags and Status Code returned by the Get Status command are defined in Tables 30-4 and 30-5. 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.

Flag Bit	Description
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 DS4835/36 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 DS4835/36 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 30-4. Bootloader Status Flags

Status Value	Description
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.

Table 30-5. Bootloader Status Codes

30.3.6 - Command 05h – Get Supported Commands

	Byte 1	Byte 3	Byte 3	Byte 4	Byte 5	Byte6
	Command	**Data Out**	**Data Out**	**Data Out**	**Data Out**	**Return**
Input	05h	X	X	X	X	X
Output	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 DS4835/36 is 403Fh, indicating that command families 0, 1, 2, 3, 4, 5 and E are supported. This command is not password protected.

30.3.7 - Command 06h – Get Code Size

	Byte 1	Byte 2	Byte 3	Byte 4
	Command	**Data Out**	**Data Out**	**Return**
Input	06h	X	X	X
Output	X	SizeL	SizeH	3Eh

This command returns SizeH:SizeL, which represents the size of available code memory in words minus 1. The DS4835/36 will return a value of 7FFFh, which indicates 32k words of program memory are available. This command is not password protected.

30.3.8 - Command 07h – Get Data Size

	Byte 1	Byte 2	Byte 3	Byte 4
	Command	**Data Out**	**Data Out**	**Return**
Input	07h	X	X	X
Output	X	SizeL	SizeH	3Eh

This command returns SizeH:SizeL, which represents the size of available data memory in words minus 1. The DS4835/36 will return a value of 07FFh, which indicates 2k words of data memory are available. This command is not password protected.

30.3.9 - Command 08h – Get Loader Version

	Byte 1	Byte 2	Byte 3	Byte 4
	Command	**Data Out**	**Data Out**	**Return**
Input	08h	X	X	X
Output	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.

30.3.10 - Command 09h – Get Utility ROM Version

	Byte 1	Byte 2	Byte 3	Byte 4
	Command	**Data Out**	**Data Out**	**Return**
Input	09h	X	X	X
Output	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.

30.3.11 - Command 0Dh – ID Banner command

	Byte 1	Byte 2	Length Bytes	Byte n+1
	Command	**Data Out**	**Data Out**	**Return**
Input	0Dh	X	X	X
Output	X	ID Banner (1)	ID Banner (n)	3Eh

This command returns the device's specific ID Banner. The ID Banner command can be used to differentiate various MAXIM MCU's. The ID banner for DS4835/36 is "DS4835/36 Loader 1.01 08-24-2016"

30.3.12 - Command 10h – Load Code

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Return**
Input	10h	Length	AddressL	AddressH	Data to load	X
Output	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 DS4835/36 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 one word of data to flash is approximately 40us. 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 100 kHz. 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 1-Wire 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. During this time, a read of the Status Byte will return the status bits set to 10, which is loader busy. The host must wait for the loader to complete the flash write before sending the next byte.

30.3.13 - Command 11h – Load Data

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Return**
Input	11h	Length	AddressL	AddressH	Data to load	X
Output	X	X	X	X	X	3Eh

This command writes (Length) bytes of data into the data SRAM starting at byte address (AddressH:AddressL). The DS4835/36 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.

30.3.14 - Command 20h – Dump Code

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Length Bytes	Byte Length+7
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Data In**	**Data Out**	**Return**
Input	20h	2	AddrL	AddrH	LengthL	LengthH	X	X
Output	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.

30.3.15 - Command 21h – Dump Data

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 5	Length Bytes	Byte Length+7
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Data In**	**Data Out**	**Return**
Input	21h	2	AddrL	AddrH	LengthL	LengthH	X	X
Output	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.

30.3.16 - Command 30h – CRC Code

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Data In**	**Data Out**	**Data Out**	**Return**
Input	30h	2	AddrL	AddrH	LengthL	LengthH	X	X	X
Output	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 to complete. 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 1-Wire loader, the 1-Wire status bits can be used to determine when the CRC calculation is complete.

30.3.17 - Command 31h – CRC Data

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Data In**	**Data Out**	**Data Out**	**Return**
Input	31h	2	AddrL	AddrH	LengthL	LengthH	X	X	X
Output	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 to complete. 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 1-Wire loader, the 1-Wire status bits can be used to determine when the CRC calculation is complete.

30.3.18 - Command 40h – Verify Code

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Return**
Input	40h	Length	AddrL	AddrH	Data to Verify	X
Output	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.

30.3.19 - Command 41h – Verify Data

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Return**
Input	41h	Length	AddrL	AddrH	Data to Verify	X
Output	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.

30.3.20 - Command 50h – Load and Verify Code

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Return**
Input	50h	Length	AddrL	AddrH	Data to load and verify	X
Output	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 of 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.

30.3.21 - Command 51h – Load and Verify Data

	Byte 1	Byte 2	Byte 3	Byte 4	(Length) Bytes	Byte Length+5
	Command	**Data In**	**Data In**	**Data In**	**Data In**	**Return**
Input	51h	Length	AddrL	AddrH	Data to load and verify	X
Output	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.

30.3.22 - Command E0h – Code Page Erase

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
	Command	**Data In**	**Data In**	**Data In**	**Return**
Input	E0h	0	PageNum	0	X
Output	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 DS4835/36 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

SECTION 31 – SYSTEM REGISTER DESCRIPTIONS

Most functions of the DS4835/36 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 DS4835/36's system registers. Table 31-1 shows the DS4835/36 system register map. Table 31-2 explains system register bit functions. This is followed by a detailed bit description.

Table 31-1. 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 31-2. 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)															

31.1 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.

31.2 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)

31.3 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.

31.4 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.

31.5 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.

31.6 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.

Bit	Name	Function
7	TAP	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.
6:5	Reserved	Reserved. All reads return 0.
4	CDA0	**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.
3	Reserved	Reserved. All reads return 0.
2	ROD	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.
1	PWL	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.
0	Reserved	Reserved. All reads return 0.

CDA0	Byte Mode Active Page	Word Mode Active Page
0	P0	P0 and P1
1	P1	P0 and P1

31.7 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

31.8 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.

Refer watchdog section for WDCN register description and further detail.

31.9 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.

31.10 Prefix Register (PFX[n], 0Bh[n])

Initialization: This register is cleared to 0000h 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>15:0</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>

31.11 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.

31.12 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.

31.13 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.

31.14 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.

31.15 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.

31.16 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).

31.17 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>

31.18 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.

31.19 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.

31.20 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.

31.21 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.

31.22 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.

31.23 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.

31.24 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]).

31.25 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.

31.26 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 32 – INSTRUCTION SET

Table 32-1. 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 with Carry

Add

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
```

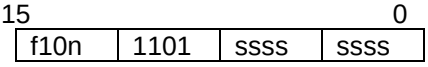
{L/S}DJNZ LC*[n], src* Decrement Counter,
{Long/Short} Jump Not Zero

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:



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* Conditional {Long/Short}
Jump on Status Flag
{L/S} JUMP Z / {L/S} JUMP NZ, *src*
{L/S} JUMP E / {L/S} JUMP NE, *src*
{L/S} JUMP S, *src*

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 1st 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 1st 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>*, C — 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 Conditional
Return on Status Flag
RET Z / RET NZ
RET S

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 Conditional Return from Interrupt on Status Flag
RETI Z / RETI NZ
RETI S

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

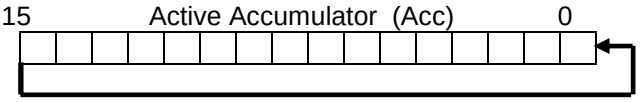
Example(s):
```
RETI S                  ; S=0, return from interrupt (RETI) does not occur
```

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.

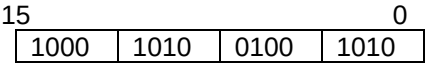
Status Flags: C (for RLC only), S, Z (for RLC only)

RL Operation:



Acc.[15:1]← Acc.[14:0]; Acc.0 ← Acc.15

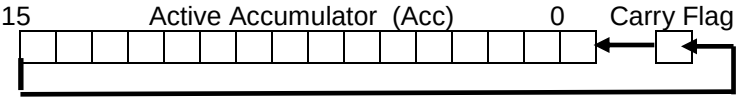
Encoding:



Example(s):

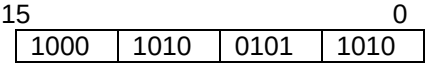
```
                    ; Acc = A345h, S=1, Z=0
RL                  ; Acc = 468Bh, S=0, Z=0
RL                  ; Acc = 8D16h, S=1, Z=0
```

RLC Operation:



Acc.[15:1]← Acc.[14:0]; Acc.0 ← C; C ← Acc.15

Encoding:



Example(s):

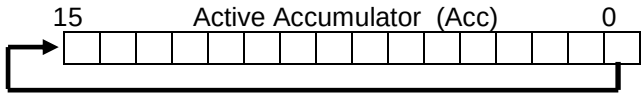
```
                    ; Acc = A345h, C=1, S=1, Z=0
RLC                 ; Acc = 468Bh, C=1, S=0, Z=0
RLC                 ; Acc = 8D17h, C=0, S=1, Z=0
```

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.

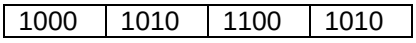
Status Flags: C (for RRC only), S, Z (for RRC only)

RR Operation:



Acc.[14:0]← Acc.[15:1]; Acc.15 ← Acc.0

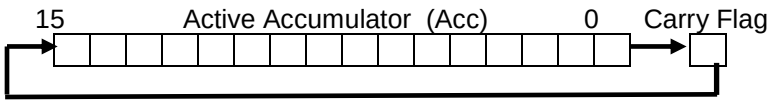
Encoding:



Example(s):

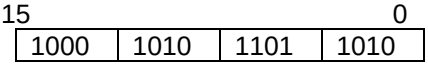
```
                    ; Acc = A345h, S=1, Z=0
RR                  ; Acc = D1A2h, S=1, Z=0
RR                  ; Acc = 68D1h, S=0, Z=0
```

RRC Operation:



Acc.[14:0]← Acc.[15:1]; Acc.15 ← C; C ← Acc.0

Encoding:



Example(s):

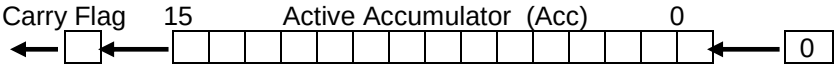
```
                    ; Acc = A345h, C=1, S=1, Z=0
RRC                 ; Acc = D1A2h, C=1, S=1, Z=0
RRC                 ; Acc = E8D1h, C=0, S=1, Z=0
```

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.

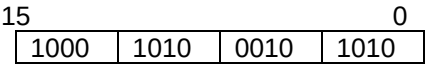
Status Flags: C, S, Z

SLA Operation:



C ← Acc.15; Acc.[15:1]← Acc.[14:0]; Acc.0 ← 0

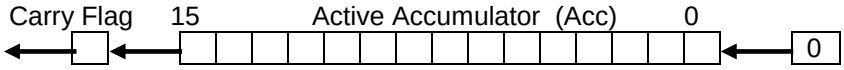
Encoding:



Example(s):

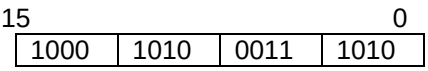
```
                    ; Acc = E345h, C=0, S=1, Z=0
SLA                 ; Acc = C68Ah, C=1, S=1, Z=0
SLA                 ; Acc = 8D14h, C=1, S=1, Z=0
```

SLA2 Operation:



C ← Acc.14; Acc.[15:2]← Acc.[13:0]; Acc.[1:0] ← 0

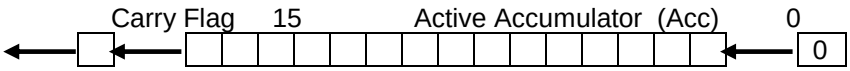
Encoding:



Example(s):

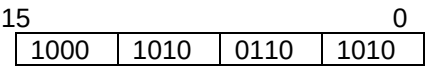
```
                    ; Acc = E345h, C=0, S=1, Z=0
SLA2                ; Acc = 8D14h, C=1, S=1, Z=0
```

SLA4 Operation:



C ← Acc.12; Acc.[15:4]← Acc.[11:0]; Acc.[3:0] ← 0

Encoding:



Example(s):

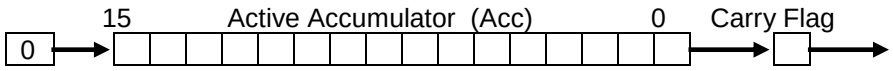
```
                    ; Acc = E345h, C=0, S=1, Z=0
SLA4                ; Acc = 3450h, C=0, S=0, Z=0
```

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.

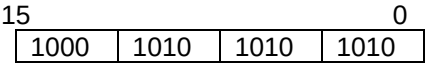
Status Flags: C, S (changes for SR only), Z

SR Operation:



Acc.15 ← 0; Acc.[14:0]← Acc.[15:1]; C ← Acc.0

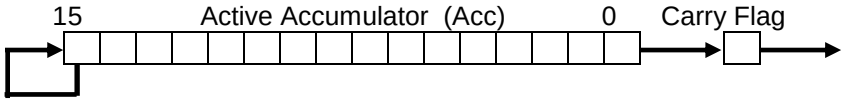
Encoding:



Example(s):

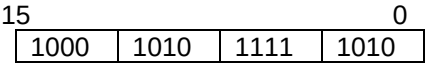
```
                    ; Acc = A345h, C=1, S=1, Z=0
SR                  ; Acc = 51A2h, C=1, S=0, Z=0
SR                  ; Acc = 28D1h, C=0, S=0, Z=0
```

SRA Operation:



Acc.[14:0]← Acc.[15:1]
Acc.15 ← Acc.15
C ← Acc.0

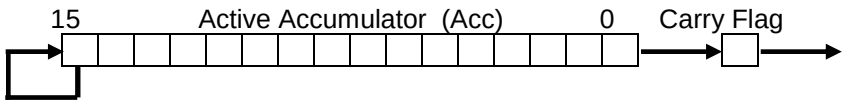
Encoding:



Example(s):

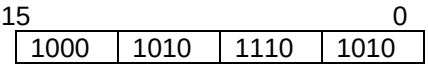
```
                    ; Acc = 0003h, C=0, Z=0
SRA                 ; Acc = 0001h, C=1, Z=0
SRA                 ; Acc = 0000h, C=1, Z=1
```

SRA2 Operation:



Acc.[13:0]← Acc.[15:2]
Acc.[15:14] ← Acc.15
C ← Acc.1

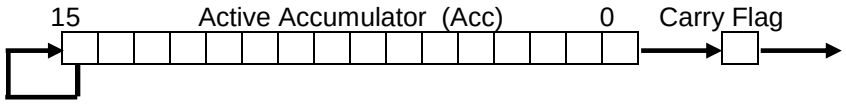
Encoding:



Example(s):

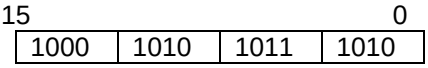
```
                    ; Acc = 0003h, C=0, Z=0
SRA2                ; Acc = 0000h, C=1, Z=1
```

SRA4 Operation:



Acc.[11:0]← Acc.[15:4]
Acc.[15:12] ← Acc.15
C ← Acc.3

Encoding:



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 33 – PROGRAMMING

The following section provides a programming overview of the DS4835/36. 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.

33.1 – Addressing Modes

The instruction set for the DS4835/36 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
```

33.2 – Prefixing Operations

All instructions on the DS4835/36 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.

33.3 – Reading and Writing Registers

All functions in the DS4835/36 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.

33.3.1 – 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
```

33.3.2 – 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
```

33.3.3 – 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]
```

33.3.4 – 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
```

33.4 – 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.

33.5 – Using the Arithmetic and Logic Unit

The DS4835/36 provides a 16-bit Arithmetic and Logic Unit (ALU) which allows operations to be performed between the active accumulator and any other register. The DS4835/36 is equipped with sixteen 16-bit working accumulators.

33.5.1 – 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
```

33.5.2 – 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 33-1.

Table 33-1. 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
```

33.5.3 – 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]
```

33.5.4 – 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
```

33.5.5 – 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.

33.5.6 – Example: Adding two four-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
```

33.6 - 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.

33.6.1 - 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
```

33.6.2 - 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
```

33.6.3 - 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. Please 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.

33.6.4 - 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)

33.6.5 - 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)

33.7 - Controlling Program Flow

The DS4835/36 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.

33.7.1 - 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.

33.7.2 - 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.

33.7.3 - 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.

33.7.4 - 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
```

33.7.5 - 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
        ...
```

33.7.6 - Conditional returns

Similar to the conditional jumps, the DS4835/36 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
```

33.8 - Handling Interrupts

Handling interrupts in the DS4835/36 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
```

Please 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.

33.8.1 - Conditional return from interrupt

Similar to the conditional returns, the DS4835/36 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
```

33.9 - 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 DS4835/36 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 DS4835/36, 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
```

33.10 - 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--]
```

SECTION 34 – UTILITY ROM

34.1 – Overview

The DS4835/36 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 some of the 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 + 3]

It is also possible to call utility ROM functions directly, using the entry points given in Table 34-1. Calling a function directly will provide faster code execution.

Table 34-1 DS4835/36 Utility ROM Functions

INDEX	FUNCTION NAME	ENTRY POINT	SUMMARY
1	UROM_flashWrite	8473h	Programs a single word of flash memory.
2	UROM_flashErasePage	8496h	Erases (programs to FFFFh) a 256-word page of flash.
3	UROM_flashEraseAll	84ACh	Erases (programs to FFFFh) all flash memory.
4	UROM_moveDP0	84BBh	Reads a byte/word at DP[0].
5	UROM_moveDP0inc	84BEh	Reads a byte/word at DP[0], then increments DP[0].
6	UROM_moveDP0dec	84C1h	Reads a byte/word at DP[0], then decrements DP[0].
7	UROM_moveDP1	84C4h	Reads a byte/word at DP[1].
8	UROM_moveDP1inc	84C7h	Reads a byte/word at DP[1], then increments DP[0].
9	UROM_moveDP1dec	84CAh	Reads a byte/word at DP[1], then decrements DP[0].
10	UROM_moveBP	84CDh	Reads a byte/word at BP[OFFS].
11	UROM_moveBPinc	84D0h	Reads a byte/word at BP[OFFS], then increments OFFS.
12	UROM_moveBPdec	84D3h	Reads a byte/word at BP[OFFS], then decrements OFFS.
13	UROM_copyBuffer	84D6h	Copies LC[0] bytes/words (up to 255) from DP[0] to BP[OFFS].
15	UROM_infoRead	850Eh	Reads ADC scale and offset values from factory info memory.
16	UROM_sramRead	851Ch	Read data from SRAM
17	UROM_sramWrite	8519h	Write data to SRAM
	soft_reset	88C5h	Performs a reset of the DS4835/36.
	UpdateTxBuffer	88CFh	Loads TX Buffer for currently selected slave address
	UpdateTxBuffer1_2	88DFh	Loads TX Buffer for slave address 1 and 2
	UpdateAllTxBuffers	88FDh	Loads TX Buffer for all 4 slave addresses
	ReadI2CRxBuffer	8933h	Reads data from slave I2C RX Buffer
	ReadWordFromFlash	88CCh	Reads word from flash.

34.2 – In-Application Programming Functions

34.2.1 – 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]],A[10],A[11]

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.

34.2.2 – 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]],A[10],A[11]

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.

34.2.3 – 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],A[10],A[11]

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.

34.3 – Data Transfer Functions

The DS4835/36 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.

Notes for all Data Transfer Functions:

- Before calling these functions, DPC should be set appropriately to configure 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 2-4. When a byte mode address is used, CDA0 must be set appropriately to access either the upper or lower half of program flash memory.

34.3.1 – 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:

- This function automatically selects DP[0] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[0]

34.3.2 – 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:

- This function automatically selects DP[0] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[0]++

34.3.3 – 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:

- This function automatically selects DP[0] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[0]—

34.3.4 – 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:

- This function automatically selects DP[1] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[1]

34.3.5 – 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:

- This function automatically selects DP[1] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[1]++

34.3.6 – 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:

- This function automatically selects DP[1] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @DP[1]—

34.3.7 – 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:

- This function automatically selects BP[OFFS] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @BP[OFFS]

34.3.8 – 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:

- This function automatically selects BP[OFFS] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @BP[OFFS++]

34.3.9 – 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:

- This function automatically selects BP[OFFS] as the data pointer before reading the byte/word value.
- Implemented as: move GR, @BP[OFFS--]

34.3.10 – 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], APDDAT4

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.
- Both DP[0] and BP[OFFS] should be configured to the same mode (byte or word) for correct buffer copying.
- This function automatically selects the data pointers before reading the byte/word values.

Note: The UROM_copyBuffer command inadvertently writes to APDDAT4. If APD4 is being used, set APDIDX4 to a value greater than or equal to 9 after initializing or making changes to APD4. This will ensure that APDDAT4 is in a non-writable area if the UROM_copyBuffer command is called.

34.3.11 – UROM_sramWrite

Function	UROM_sramWrite
Summary	Writes data at A[1] to SRAM address given by A[0]
Inputs	A[1]: Data to write to SRAM A[0]: Address in SRAM to write.
Outputs	None
Destroys	DP[0]

Notes:

- This function can be used to quickly copy data to SRAM.

34.3.12 – UROM_sramRead

Function	UROM_sramRead
Summary	Reads data from SRAM address given by A[1]/
Inputs	A[1]: SRAM address to read from.
Outputs	A[0]: Data read from SRAM
Destroys	DP[0]

Notes:

- This function can be used to quickly read data from SRAM.

34.4 – Special Functions

The DS4835/36 also provides some special UROM functions that are designed to be called directly by firmware in an application.

34.4.1 SOFT_RESET

Function	soft_reset
Summary	Resets the DS4835/36
Inputs	None
Outputs	None
Destroys	All

UROM has the code necessary at this location to generate an internal reset when application software jumps to this location.

34.4.2 UROM_infoRead

Function	UROM_infoRead
Summary	Returns values from info memory.
Inputs	A[1]: Location to read from
Outputs	A[0]
Destroys	A[0], A[1]

Notes:

- This function will return values from info memory.
- Because this function has A[1] as an input, this should be interrupt protected.
- Prior to calling this function, load A[1] with the correct value for the desired parameter.
    - 0019h: ADC scale for internal temperature conversions
    - 001Ah: ADC offset for internal temperature conversions
    - 0035h: ASCII characters for die revision

34.4.3 UpdateTxBuffer

Function	UpdateTxBuffer
Summary	Loads data into currently selected slave I2C TX Buffer.
Inputs	None
Outputs	None
Destroys	None

This routine will copy 4 words of data from the SRAM location currently pointed to by RPNTR to the slave I2C TX buffer selected by CUR_SLA. Prior to calling this routine, the CUR_SLA register must be set for the proper slave address.

34.4.4 UpdateTxBuffer1_2

Function	UpdateTxBuffer1_2
Summary	Loads data into currently the slave I2C TX Buffers for addresses 1 and 2.
Inputs	None
Outputs	None
Destroys	None

This routine will copy 4 words of data from the SRAM location currently pointed to by RPNTR for I2C slave address 1 into this slave address 1's TX buffer. This is followed by copying 4 words of data from the SRAM location currently pointed to by RPNTR for I2C slave address 2 into this slave address 2's TX buffer.

34.4.5 UpdateTxBufferAll

Function	UpdateTxBuffer1_2
Summary	Loads data into all of the slave I2C TX Buffers.
Inputs	None
Outputs	None
Destroys	None

For each of the 4 slave addresses, this routine will copy 4 words of data from the SRAM location currently pointed to by the slave's RPNTR into the slave's TX buffer.

34.4.6 ReadI2CRxBuffer

Function	ReadI2CRxBuffer
Summary	Loads data into all of the slave I2C TX Buffers.
Inputs	A[0]: Address for RX Count A[1]: Address in which to store the received data.
Outputs	A[0]
Destroys	None

This routine will copy all data received from the I2C Slave RX FIFO into the SRAM location pointed to by the combination of A[0] and A[1]. A[1] is the location of the beginning of the data array to store the received data to. A[0] is an offset (rx count) to account for any data bytes that have already been stored to the data array. At the completion of this routine, A[0] (rx count) will be updated to reflect the offset for the new data that was just copied.

34.4.7 ReadWordFromFlash

Function	ReadWordFromFlash
Summary	Reads one word from flash
Inputs	DP[0] Address for RX Count
Outputs	A[0]
Destroys	None

This routine will read one word of data from the address given by DP[0] and return this data at A[0].

34.5 – Utility ROM Examples

34.5.1 – Reading Constant Word Data from Flash

```
UROM_moveDP0inc  equ  08511h

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
```

34.5.2 – 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
```

REVISION HISTORY

Changes made between Revision 0.3 and 0.4

The DS4836 was included into the DS4835 User's Guide.

Section 1: Overview
- Added DS4836 block diagram.

Section 2: Architecture
- Changed "When VDD is below the POR level, the state of all the DS4835/36 pins is high impedance" to "When V_{DD} is greater than the V_{BO} level, the default state of all the DS4835/36 pins is high-impedance."
- Added "When VDD is less than the V_{BO} level, the state of the DS4835/36 pins is undetermined."

Section 3: Peripheral Registers
- Added I2CCN2, IV2 for DS4836.
- Added M1[22] as a GP_REG.
- Changed GP_REG at M2[11] to be DS4835 only.

Section 4: GPIO
- Updated GPIO drawings to show switch between pin and ADC for DS4836
- Added text details about this switch on DS4836.

Section 5: Inrush
- Added note that if VCCO current is greater than inrush limit, the device will not power up.

Section 6: ADC
- Updated Section 6.1.11. Factory calibrated scale and offset values for voltage conversions was removed.
- Added DS4836 note about switch between pin and ADC.

Section 9: External DCDC Controller
- Added note regarding layout of digital switching signals to prevent corruption of adjacent signals.
- Separated component selection into APD boost and inverting DCDC recommendations.
- Added formula for integrator clamp.
- Added recommendation to optimize the feedback network for better transient response.

Section 11: TEC
- Added mention of Vtherm conversion being ratiometric and the TEC will not be sensitive to any drift in VREF.
- Added section on estimating TEC current based upon VTEC and TEC resistance based upon characterization of the TEC.
- Updated information for NORMV needing to be set to ½ of the ADC value for the minimum expected VCC.
- Updated TEC setpoint register for variants.
- Added note about startup delay if VCC compensation is enabled.

Section 14: PWM
- Added note regarding layout of digital switching signals to prevent corruption of adjacent signals.

Section 15: Delta Sigma DAC
- Added note regarding layout of digital switching signals to prevent corruption of adjacent signals.

Section 16: Current DAC
- Added note regarding higher leakage on IDAC pins when using for ADC.

Section 18: Slave I2C
- Changed Figure 18-4 for transmitting data to start on first falling edge of SCL.
- Added DS4836 I2C Slave Fast-Mode Plus 1MHz Section

Section 22: General Purpose Timers
- Added Timer3 registers.

Section 24: Watchdog
- Included in this version of user's guide.

Section 26: Misc
- Added General Purpose Register (GP_REG) description.

Section 29: Debug
- Added I2CCN2, IV2, and GP_REG to peripheral register map.

Section 30: In System Programming
- Changed timer to load one word of flash from 80us to 40us. 40us is the datasheet specification.

Section 34: Utility ROM
- Removed ADC scale and offset for voltage conversions from infoRead command.
- Added die revision to infoRead command.
- Added note that infoRead should be interrupt protected.