S3FB42F

8-BIT CMOS

MICROCONTROLLER

USER'S MANUAL

Revision 1

SAMSUNG

ELECTRONICS

Important Notice

S3FB42F 8-Bit CMOS Microcontroller
User's Manual, Revision 1
Publication Number: 21-S3-FB42F-052001

Samsung Electronics' microcontroller business has been awarded full ISO-14001 certification (BVQ1 Certificate No. 9330). All semiconductor products are designed and manufactured in accordance with the highest quality standards and objectives.

Samsung Electronics Co., Ltd.
San #24 Nongseo-Lee, Kiheung-Eup
Yongin-City Kyungi-Do, Korea
C.P.O. Box #37, Suwon 449-900

TEL: (82)-(331)-209-1907
FAX: (82)-(331)-209-1889
Home-Page URL: Http://www.samsungsemi.com/

Printed in the Republic of Korea

Preface

The *S3FB42F Microcontroller User's Manual* is designed for application designers and programmers who are using the S3FB42F microcontroller for application development. It is organized in two main parts:

Part I	Programming Model	Part II	Hardware Descriptions

Part I contains software-related information to familiarize you with the microcontroller's architecture, programming model, instruction set, and interrupt structure. It has nine chapters:

Chapter 1	Product Overview	Chapter 5	Hardware Stack
Chapter 2	Address Spaces	Chapter 6	Exceptions
Chapter 3	Register	Chapter 7	Coprocessor Interface
Chapter 4	Memory Map	Chapter 8	Instruction Set

Chapter 1, "Product Overview," is a high-level introduction to S3FB42F with general product descriptions, as well as detailed information about individual pin characteristics and pin circuit types.

Chapter 2, "Address Spaces," describes program and data memory spaces. Chapter 2 also describes ROM code option.

Chapter 3, "Register," describes the special registers.

Chapter 4, "Memory Map," describes the internal register file.

Chapter 5, "Hardware Stack," describes the S3FB42F hardware stack structure in detail.

Chapter 6, "Exception," describes the S3FB42F exception structure in detail.

Chapter 7, “Coprocessor Interface,” describes the S3FB42F coprocessor interface in detail.

Chapter 8, “Instruction Set,” describes the features and conventions of the instruction set used for all S3FB-series microcontrollers.

A basic familiarity with the information in Part I will help you to understand the hardware module descriptions in Part II. If you are not yet familiar with the S3FB-series microcontroller family and are reading this manual for the first time, we recommend that you first read Chapters 1–3 carefully. Then, briefly look over the detailed information in Chapters 4, 5, 6, 7, and 8. Later, you can reference the information in Part I as necessary.

Part II "hardware Descriptions," has detailed information about specific hardware components of the S3FB42F microcontroller. Also included in Part II are electrical, mechanical. It has 19 chapters:

Chapter 9	PLL (Phase Locked Loop)	Chapter 19	Parallel Port Interface
Chapter 10	Reset and Power-Down	Chapter 20	8-bit Analog-to-Digital Converter
Chapter 11	I/O Ports	Chapter 21	I^{2}C-BUS Interface
Chapter 12	Basic Timer	Chapter 22	Random Number Generator
Chapter 13	Real Timer (Watch Timer)	Chapter 23	USB
Chapter 14	16-bit Timer (8-bit Timer A & B)	Chapter 24	Embedded Flash Memory Interface
Chapter 15	Serial I/O Interface	Chapter 25	MAC2424
Chapter 16	UART	Chapter 26	Electrical Data
Chapter 17	I^{2}S Bus (Inter-IC Sound)	Chapter 27	Mechanical Data
Chapter 18	SSFDC (Solid State Floppy Disk Card)		

Chapter 25, "MAC2424" describes the MAC2424 structure in detail, as well as instructions.

One order form is included at the back of this manual to facilitate customer order for S3FB42F microcontrollers:
the Flash Factory Writing Order Form.
You can photocopy this form, fill it out, and then forward it to your local Samsung Sales Representative.

Table of Contents

Part I — Programming Model

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

List of Figures

List of Figures (Continued)

List of Figures (Continued)

List of Figures (Continued)

List of Tables

List of Tables (Continued)

List of Programming Tips

List of Instruction Descriptions

List of Instruction Descriptions (Continued)

1 PRODUCT OVERVIEW

CALMRISC OVERVIEW

The S3FB42F single-chip CMOS microcontroller is designed for high performance using Samsung's newest 8-bit CPU core, CalmRISC.

CalmRISC is an 8-bit low power RISC microcontroller. Its basic architecture follows Harvard style, that is, it has separate program memory and data memory. Both instruction and data can be fetched simultaneously without causing a stall, using separate paths for memory access. Represented below is the top block diagram of the CalmRISC microcontroller.

FEATURES

CPU

- 8-Bit CalmRISC Core
- DSP Architecture (24 x 24-bit MAC)

Memory

- Code memory: 144K byte (72K word) half flash type memory
- Data memory: 48K byte SRAM + 69K byte flash type memory

STACK

- Size: maximum 16 (word)-level.

65 I/O Pins

- I/O: 59 pins
- Input only: 6 pins

8-Bit Serial I/O Interface

- 8-bit transmit/receive or 8-bit receive mode.
- LSB first or MSB first transmission selectable.
- Internal and external clock source.

8-Bit Basic Timer & Watchdog timer

- Programmable basic timer 8-bit counter + WDT 3-bit counter
- 8 kinds of clock source
- Overflow signal of 8-bit counter makes a basic timer interrupt. And control the oscillation warm-up time
- Overflow signal of 3-bit counter makes a system reset.

One 16-Bit Timer/Counter

- Programmable interval timer
- Two 8-bit timer counter mode and one 16-bit timer counter mode, selectable by S/W

One Real Time Clock

- Real time clock generation (0.5 or 1 second)
- Buzzer signal generation (1, 2, 4 or 8 kHz)

ROM Code Options

- Basic timer counter clock source selecting reset value

FEATURES (Continued)

I^{2}C, I^{2}S Interface

- One-Ch Multi-Master I^{2}C controller
- Two-Ch Sony/Phillips I^{2}S controller

UART Interface

- One Full-duplex UART controller

USB Specification Compliance (Ver1.0, Ver1.1)

- Built in Full Speed Transceiver
- Support 1 device address and 4 endpoints.
- 1 control endpoint and 3 data endpoints
- One 16 bytes endpoint, one 32 bytes end point, and two 64 bytes end points.
- Each data endpoint can be configurable as interrupt, bulk and isochronous.

Parallel Port Interface Controller

- Interrupt-based operation
- Support IEEE Standard 1284 communication mode (compatibility, nibble, byte and ECP mode).
- Automatic handshaking mode for any forward or reverse protocol with software enable/disable

SSFDC (Smart Media^{TM} card) Interface

- Control signals are operated by CPU instruction

Random Number Generator

- Two ring oscillators
- Linear feedback shifter register LFSR8/LFSR16

External Interrupt

- 8 source (Edge triggered 6 + Level triggered 2)

ADC

- Six 8-bit resolution channels and normal input

Two Power-down Modes

- Idle mode: only CPU clock stop.
- Stop mode: system clock and CPU clock stop.

Oscillation Sources

- Clock synthesizer (Phase-locked loop circuit) based on 32.768 kHz
- CPU clock divider circuit (Div by 1, 2, 4, 8, 16, 32, 64, 128)

Instruction Execution Times

- 33.3ns at fxx = 30 MHz when 1 cycle instruction
- 66.6ns at fxx = 30 MHz when 2 cycle instruction

Operating Temperature

- - 40 °C to 85 °C

Operating Voltage Range

- 3.0 V to 3.6 V at 30 MHz

Package Types

- 100-QFP, 100-TQFP

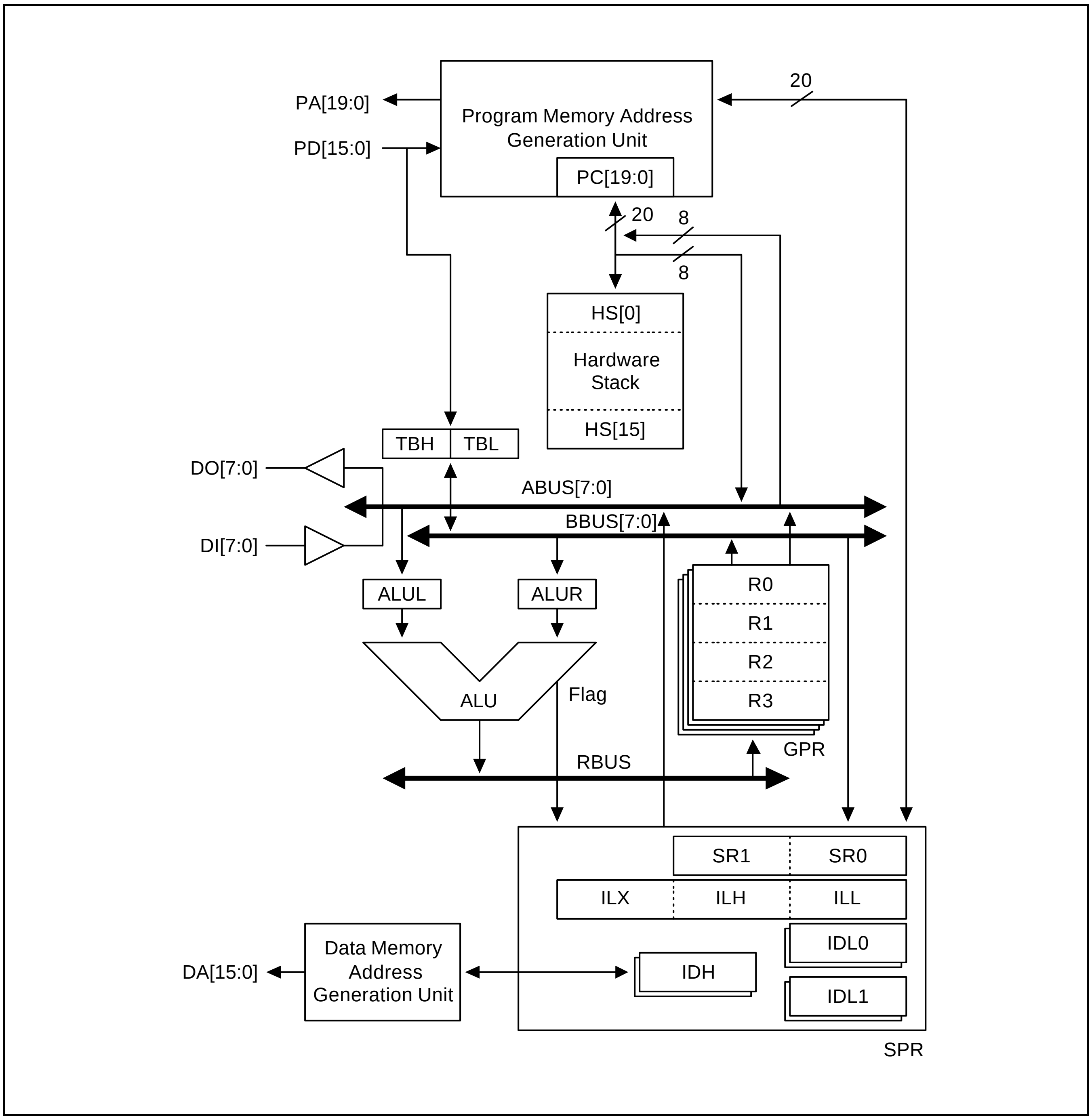


Figure 1-1. Top Block Diagram

The CalmRISC building blocks consist of:

- An 8-bit ALU
- 16 general purpose registers (GPR)
- 11 special purpose registers (SPR)
- 16-level hardware stack
- Program memory address generation unit
- Data memory address generation unit

16 GPR's are grouped into four banks (Bank0 to Bank3) and each bank has four 8-bit registers (R0, R1, R2, and R3). SPR's, designed for special purposes, include status registers, link registers for branch-link instructions, and data memory index registers. The data memory address generation unit provides the data memory address (denoted as *DA[15:0]* in the top block diagram) for a data memory access instruction. Data memory contents are accessed through *DI[7:0]* for read operations and *DO[7:0]* for write operations. The program memory address generation unit contains a program counter, PC[19:0], and supplies the program memory address through *PA[19:0]* and fetches the corresponding instruction through *PD[15:0]* as the result of the program memory access. CalmRISC has a 16-level hardware stack for low power stack operations as well as a temporary storage area.

CalmRISC has a 3-stage pipeline as discribed below:

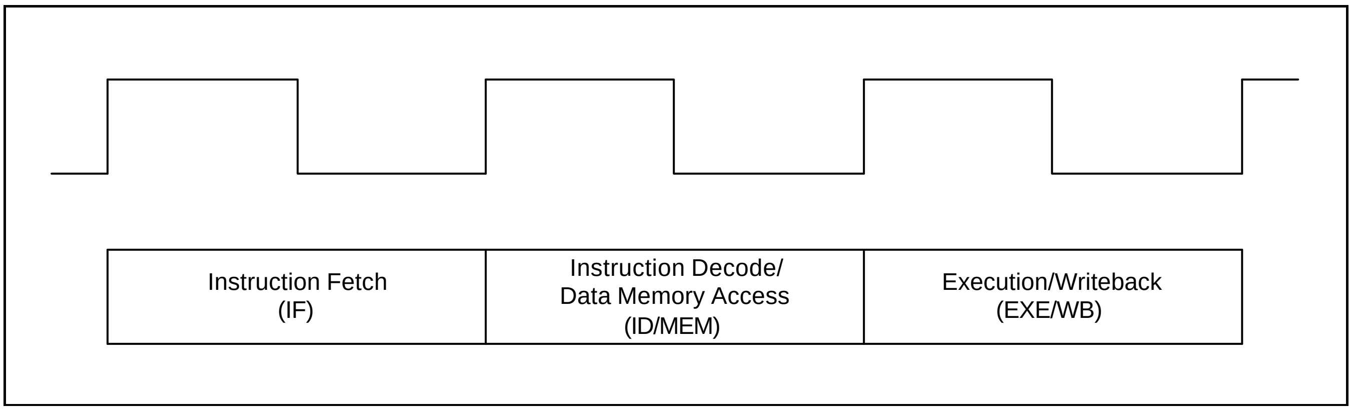


Figure 1-2. CalmRISC Pipeline Diagram

As can be seen in the pipeline scheme, CalmRISC adopts a register-memory instruction set. In other words, data memory where *R* is a GPR, can be one operand of an ALU instruction as shown below:
The first stage (or cycle) is Instruction Fetch stage (IF for short), where the instruction pointed to by the program counter, PC[19:0] , is read into the Instruction Register (IR for short). The second stage is Instruction Decode and Data Memory Access stage (ID/MEM for short), where the fetched instruction (stored in IR) is decoded and data memory access is performed, if necessary. The final stage is Execute and Write-back stage (EXE/WB), where the required ALU operation is executed and the result is written back into the destination registers.
Since CalmRISC instructions are pipelined, the next instruction fetch is not postponed until the current instruction is completely finished, but is performed immediately after the current instruction fetch is done. The pipeline stream of instructions is illustrated in the following diagram.

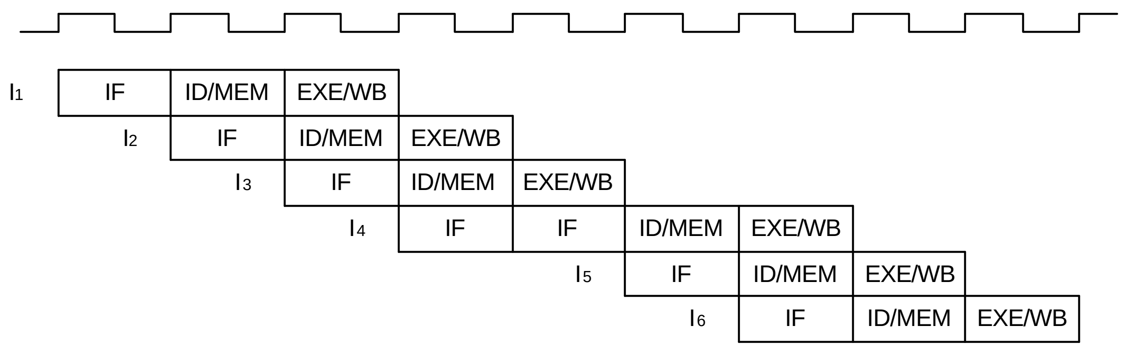


Figure 1-3. CalmRISC Pipeline Stream Diagram

Most CalmRISC instructions are 1-word instructions, while same branch instructions such as "LCALL" and "LJT" instructions are 2-word instructions. In Figure 1-3, the instruction, I_4, is a long branch instruction and it takes two clock cycles to fetch the instruction. As indicated in the pipeline stream, the number of clocks per instruction (CPI) is 1 except for long branches, which take 2 clock cycles per instruction.

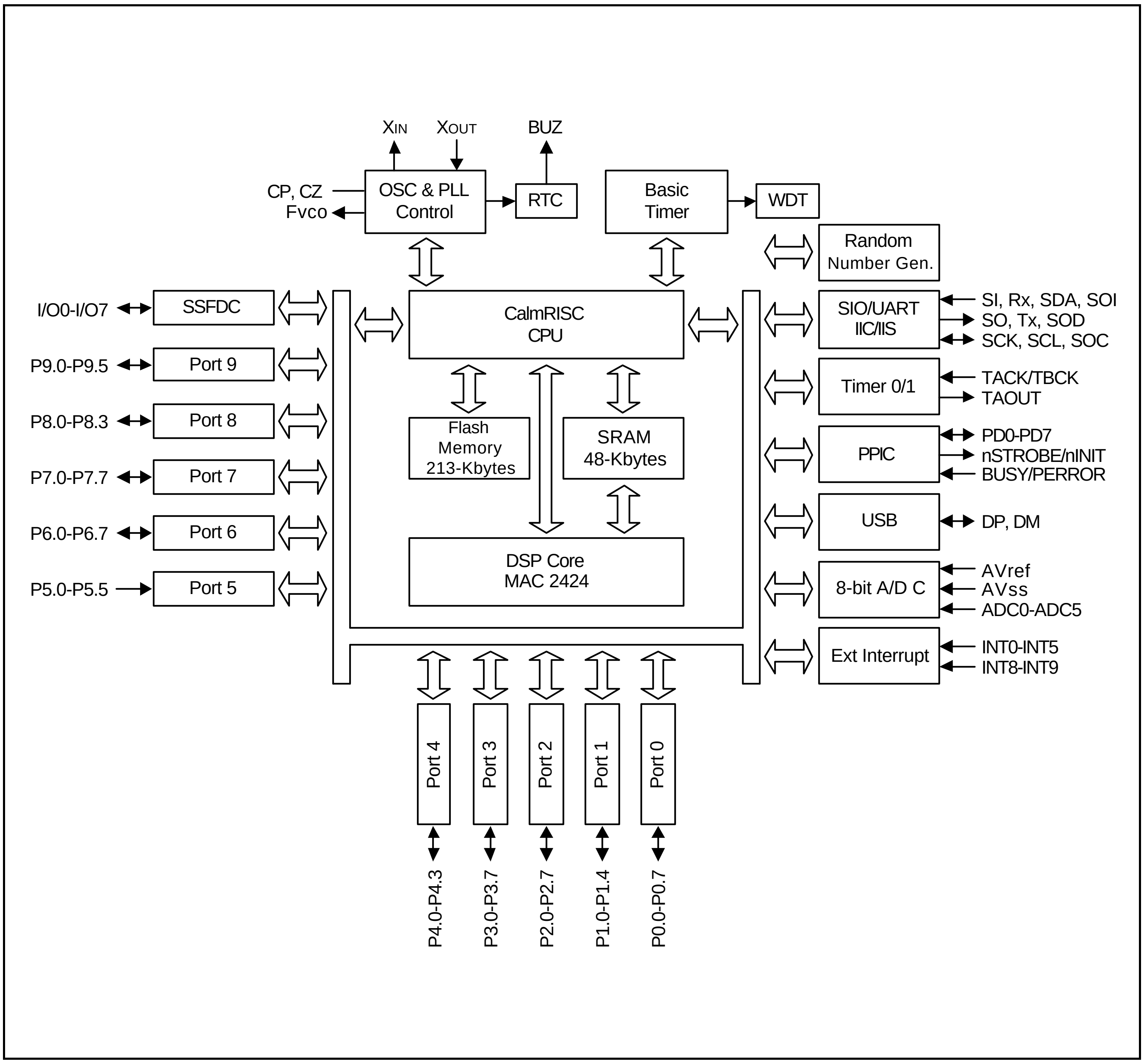
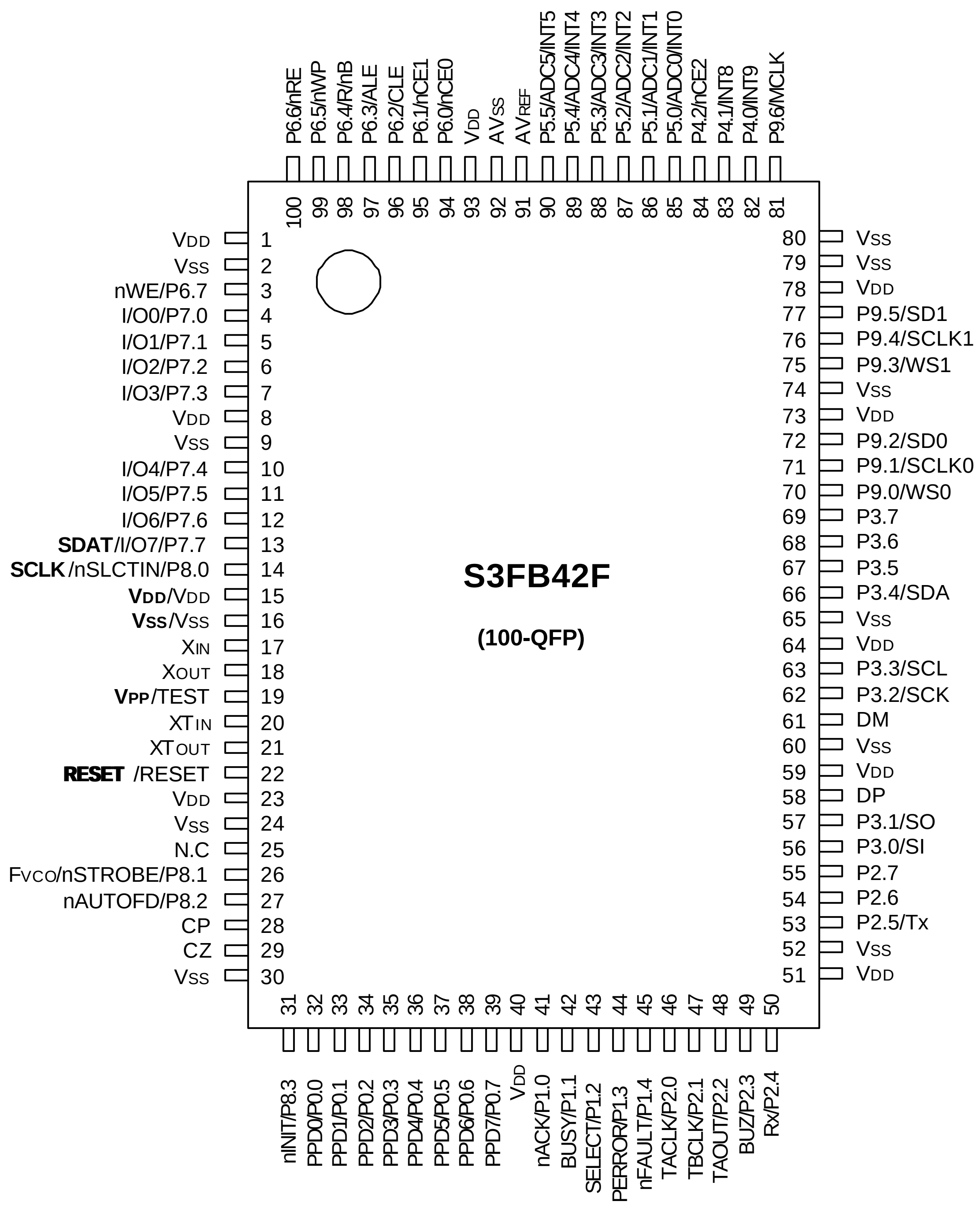
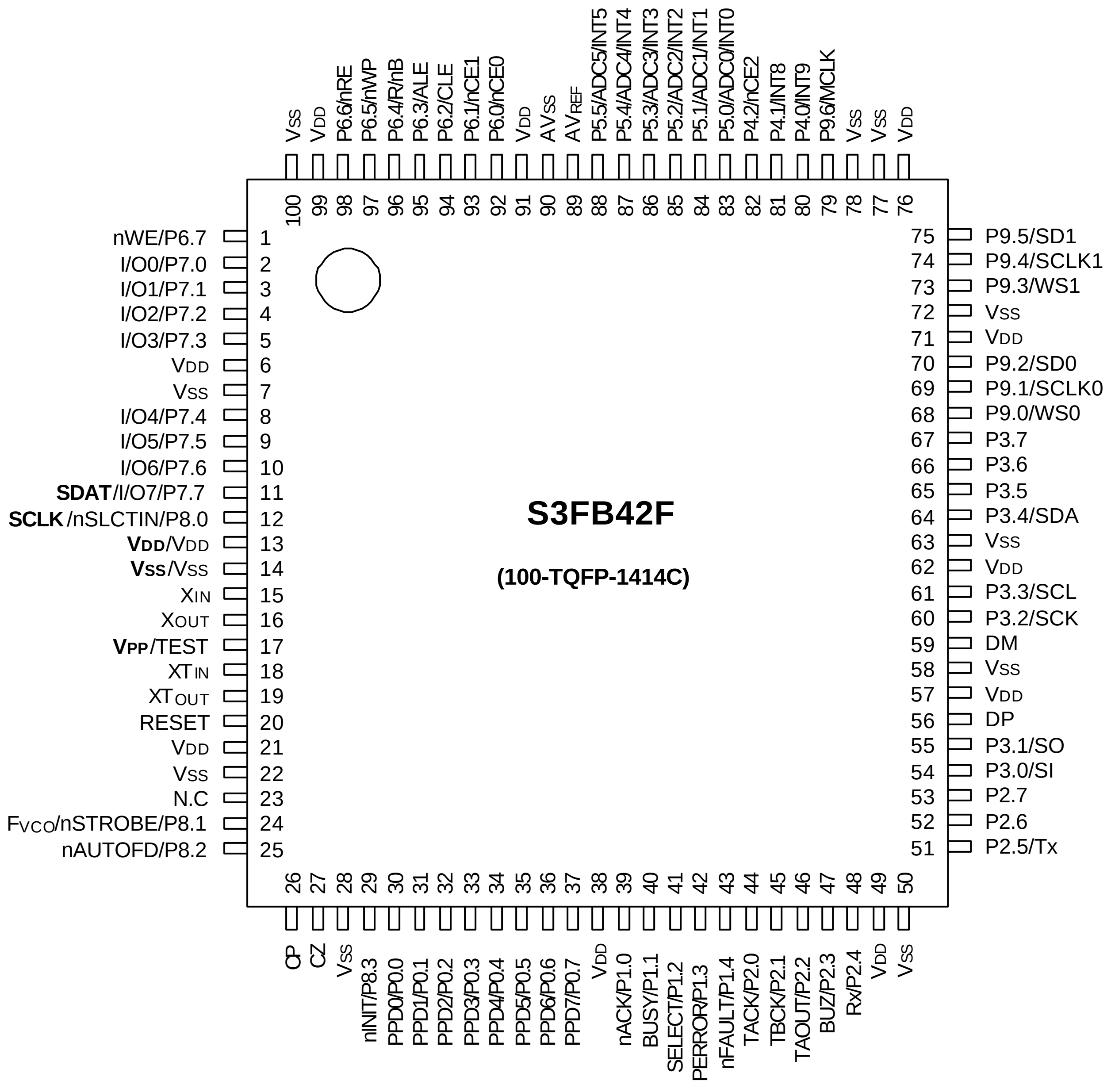


Figure 1-4. Block Diagram



NOTE: N.C means No - Connection.

Figure 1-5. 100-QFP Pin Assignment



NOTE: N.C means No - Connection

Figure 1-6. 100-TQFP Pin Assignment

PIN DESCRIPTION

Table 1-1. S3FB42F Pin Descriptions (100-TQFP)

Pin Name	Pin Type	Pin Description	Circuit Type	Pin Number	Share Pins
P0.0-P0.7	I/O	I/O port with bit programmable pins; Input or output mode selected by software; Alternately can be used as parallel port data bus pins, PPD0-PPD7. P0.0/PPD0-P0.7/PPD7: Parallel port data bus	1	30-37 (32-39)	PPD0- PPD7
P1.0-P1.4	I/O	I/O port with bit programmable pins; Push-pull output mode is selected by software; Alternately can be used as parallel port control bus pins, nACK, BUSY, SELECT, PERROR and nFAULT pin. P1.0/nACK: Not parallel port acknowledge. P1.1/BUSY: Parallel port busy. P1.2/SELECT: Parallel port select. P1.3/PERROR: Parallel port paper error P1.4/nFAULT: Not parallel port fault.	1	39-43 (41-45)	nACK- nFAULT
P2.0-P2.7	I/O	I/O port with bit programmable pins; Input and output mode are selected by software; Alternately can be used as TACLK, TBCLK, TAOUT, BUZ, Rx and Tx. P2.0/TACLK: Timer 0 clock or capture input P2.1/TBCLK: Timer 1 clock input P2.2/TAOUT: Timer 2 capture input or, PWM or toggle output P2.3/BUZ: Buzzer output P2.4/Rx: Receive input in UART P2.5/Tx: Transmit output in UART P2.6: Normal input/output pin P2.7: Normal input/output pin	4	44-48, 51-53 (46-50, 53-55)	TACLK-Tx
P3.0-P3.7	I/O	I/O port with bit programmable pins; Input or output mode selected by software; Alternately can be used as SI, SO, SCK, SCL and SDA. N-channel open drains are configurable. P3.0/SI: Serial data input pin in SIO(SPI) P3.1/SO: Serial data output pin in SIO(SPI) P3.2/SCK: Serial clock pin in SIO(SPI) P3.3/SCL: Serial clock pin in I^2C P3.4/SDA: Serial data pin in I^2C P3.5: Normal input/output pin P3.6: Normal input/output pin P3.7: Normal input/output pin	5	54-55, 60-61, 64-67 (56-57, 62-63, 66-69)	SI-SDA

NOTE: Parentheses indicate pin number for 100-QFP package.

Table 1-1. S3FB42F Pin Descriptions (100-TQFP) (Continued)

Pin Name	Pin Type	Pin Description	Circuit Type	Pin Number	Share Pins
P4.0-P4.2	I/O	I/O port with bit programmable pins; Input and output mode are selected by software; P4.0-P4.1 can be used as inputs for external interrupts INT9-INT8. (with noise filter) and assigned pull-up by software; Alternately P4.2 can be used as CE2 for SmartMedia chip select signal. P4.0/INT9: External interrupt 9 input P4.1/INT8: External interrupt 8 input P4.2/CE2: Normal in/output pin	6, 3	80-82 (82-84)	INT9-INT8 CE2
P5.0-P5.5	I	Input port with bit programmable pins; Input or ADC input mode selected by software; software assignable pull-up; Port 5 can be used as inputs for external interrupts INT0-INT5 or ADC block. P5.0/INT0/ADC0: Ext interrupt 0 or ADC0 input P5.1/INT1/ADC1: Ext interrupt 1 or ADC1 input P5.2/INT2/ADC2: Ext interrupt 2 or ADC2 input P5.3/INT3/ADC3: Ext interrupt 3 or ADC3 input P5.4/INT4/ADC4: Ext interrupt 4 or ADC4 input P5.5/INT5/ADC5: Ext interrupt 5 or ADC5 input	7	83-88 (85-90)	INT0/ADC0-INT5/ADC5
P6.0-P6.7	I/O	I/O port with bit programmable pins; Alternately Port 6 can be used as CE0, CE1, CLE, ALE, WE, WP, RE and R/B for SmartMedia control signal. P6.0/CE0: Chip Select strobe output 0 for SM. P6.1/CE1: Chip Select strobe output 1 for SM. P6.2/CLE: Command latch enable output for SM. P6.3/ALE: Address latch enable output for SM. P6.4/R/B: Ready and Busy status input for SM. P6.5/WP: Write protect output for SM. P6.6/RE: Read enable strobe output for SM. P6.7/WE: Write enable strobe output for SM.	1, 2	92-98, 1 (94-100, 3)	CE0-WE
P7.0-P7.7	I/O	I/O port with bit programmable pins; Alternately Port 7 can be used as I/O port for SmartMedia control signal. P7.0/I/O0-P7.7/I/O7: I/O port for SmartMedia control signal.	1	2-5, 8-11 (4-7, 10-13)	I/O0-I/O7

NOTE: Parentheses indicate pin number for 100-QFP package.

Table 1-1. S3FB42F Pin Descriptions (100-TQFP) (Continued)

Pin Name	Pin Type	Pin Description	Circuit Type	Pin Number	Share Pins
P8.0-P8.3	I/O	I/O port with bit programmable pins; Alternately can be used as parallel port control bus pins, nSLCTIN, NsTROBE, nAUTOFD and nINIT pin. P8.0/nSLCTIN: Not select information input. P8.1/nSTROBE/F_{VCO}: Not strobe input or F_{VCO} output P8.2/nAUTOFD: Not auto-feed input P8.3/nINIT: Not parallel port initialization.	4	12, 24, 25, 29 (14, 26, 27, 31)	nSLCTIN-nINIT
P9.0-P9.6	I/O	I/O port with bit programmable pins; Alternately can be used as serial data interface pins, WS0, SCLK0, SD0, WS1, SCLK1, SD1 and MCLK. P9.0/WS0: Word select pin in I^2S0 P9.1/SCLK0: Bit serial clock pin in I^2S0. P9.2/SD0: Serial data pin in I^2S0 P9.3/WS1: Word select pin in I^2S1. P9.4/SCLK1: Bit serial clock pin in I^2S1. P9.5/SD1: Serial data pin in I^2S1. P9.6/MCLK: Master Clock pin in I^2S0.	4	68-70, 73-76, 79 (70-72, 75-77, 79)	WS0-MCLK
DM	I/O	Only be used USB transceive/receive port	–	59 (61)	–
DP	I/O	Only be used USB transceive/receive port	–	56 (58)	–
V_{DD}, V_{DD}	–	Power supply	–	6, 13, 21, 26, 38, 49, 57, 62, 71, 76, 91, 99 (1, 8, 15, 23, 28, 40, 52, 59, 64, 73, 78, 93)	–
V_{SS}, V_{SS}	–	Ground	–	7, 14, 22, 27, 28, 50, 58, 63, 72, 77, 78, 100 (2, 9, 16, 24, 29, 30, 51, 60, 65, 74, 79, 80)	–

NOTE: Parentheses indicate pin number for 100-QFP package.

Table 1-1. S3FB42F Pin Descriptions (100-TQFP) (Continued)

Pin Name	Pin Type	Pin Description	Circuit Type	Pin Number	Share Pins
X_{IN}, X_{OUT}	–	Crystal, ceramic oscillator signal for PLL reference frequency (for external clock input, use X_{IN} and input X_{IN}'s reverse phase to X_{OUT})	–	15, 16 (17, 18)	–
XT_{IN}, XT_{OUT}	–	Crystal, ceramic oscillator (for external clock input, use XT_{IN} and input XT_{IN}'s reverse phase to XT_{OUT})	–	18, 19 (20,21)	–
CP, CZ	–	Low pass filter circuit for PLL	–	26, 27 (28, 29)	–
TEST	I	Test signal input	9	17 (19)	–
RESET	I	Reset signal	8	20 (22)	–
AV_{REF}, AV_{SS}	–	Power supply pin and ground pin for A/D converter.	–	89, 90 (91, 92)	–
SDAT	I/O	Serial data in/output pin for serial program block	1	11 (13)	P7.7
SCLK	I	Serial clock input pin for serial program block	1	12 (14)	P8.0
V_{DD}	–	Power supply pin for serial program block	–	13 (15)	–
V_{SS}	–	Ground pin for serial program block	–	14 (16)	–
V_{PP}	–	Flash Cell Power supply pin or mode selection pin for serial program block	–	17 (19)	TEST
RESET	I	Reset pin for serial program block	8	20 (22)	RESET

NOTE: Parentheses indicate pin number for 100-QFP package.

PIN CIRCUIT DIAGRAMS

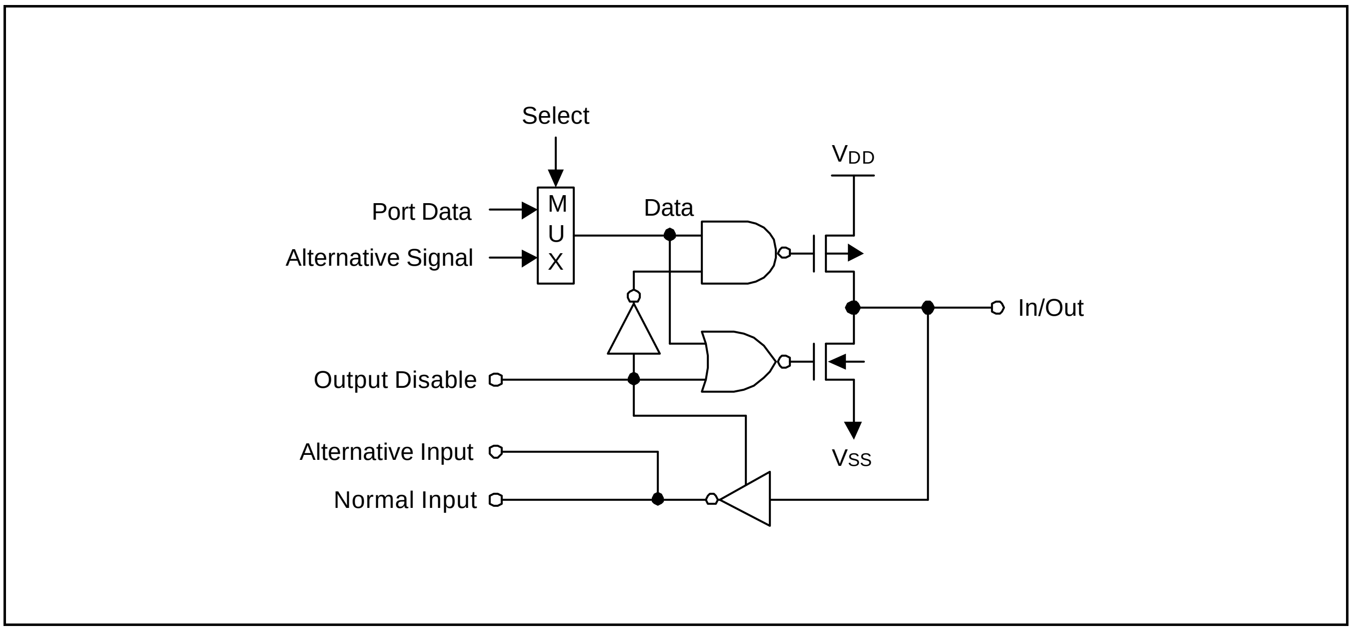


Figure 1-7. Pin Circuit Type 1 (Port 0, P1.0-P1.4, P6.0-P6.5, and Port 7)

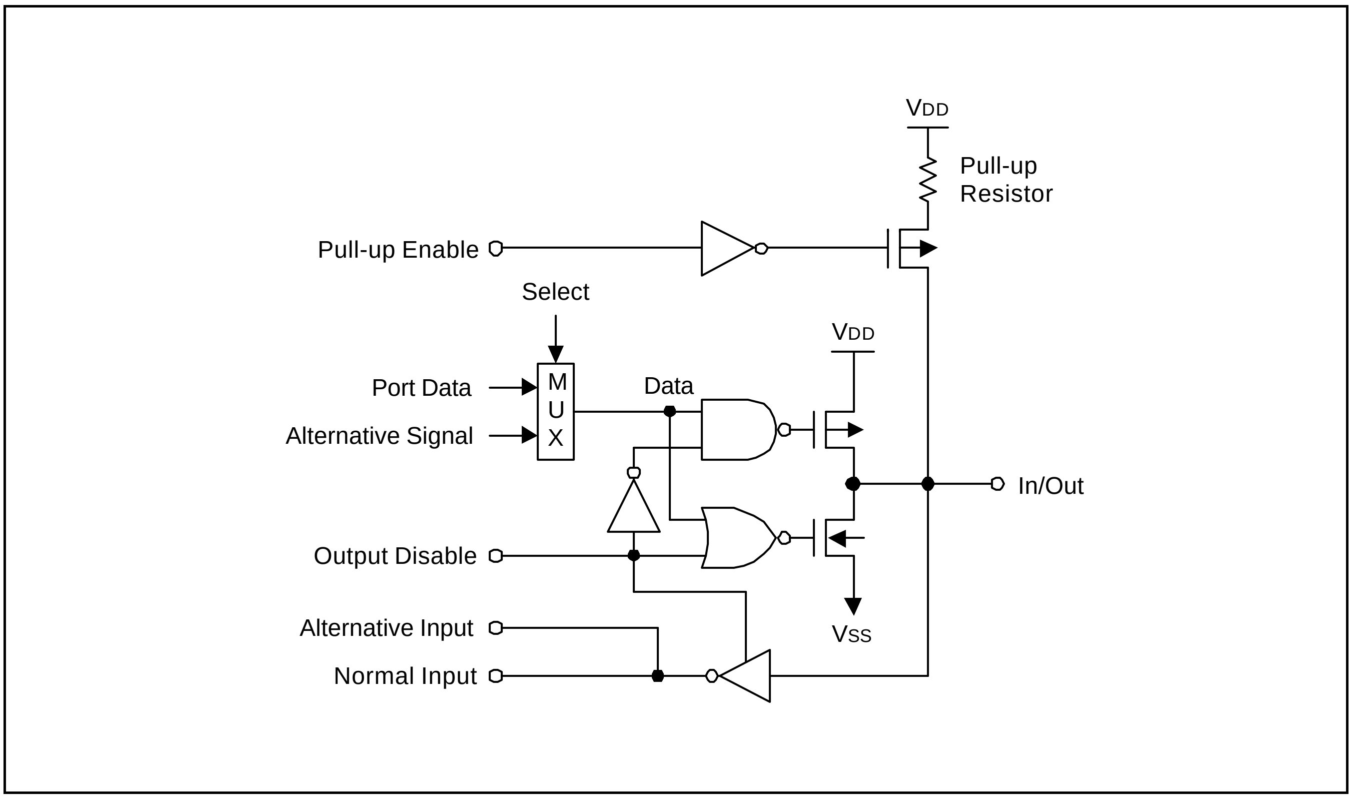


Figure 1-8. Pin Circuit Type 2 (P6.6 and P6.7)

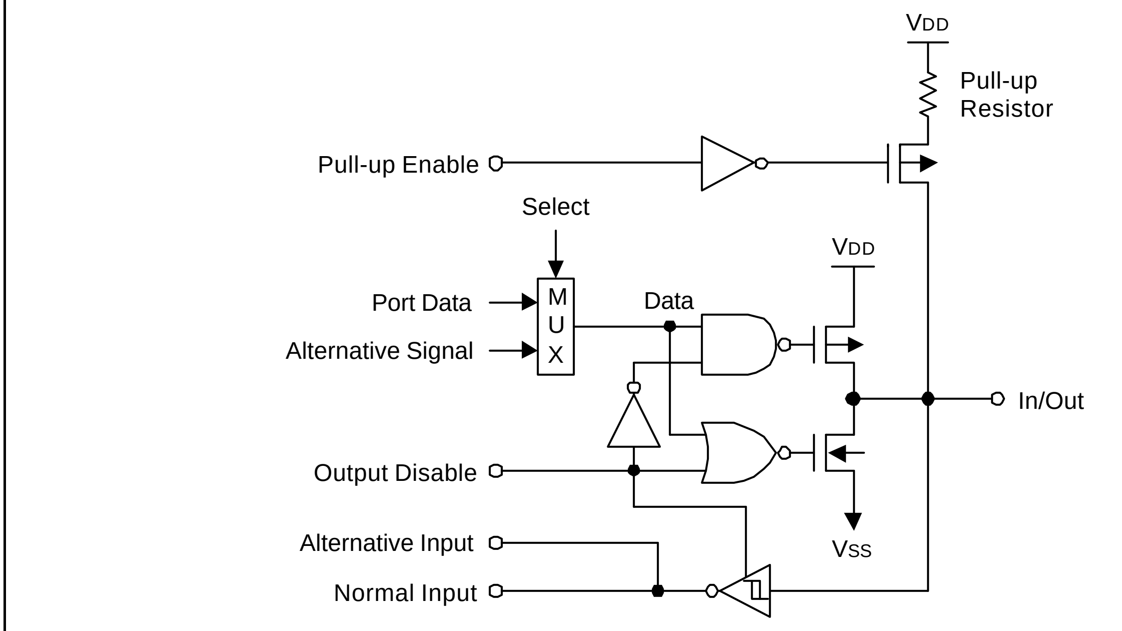


Figure 1-9. Pin Circuit Type 3 (P4.2)

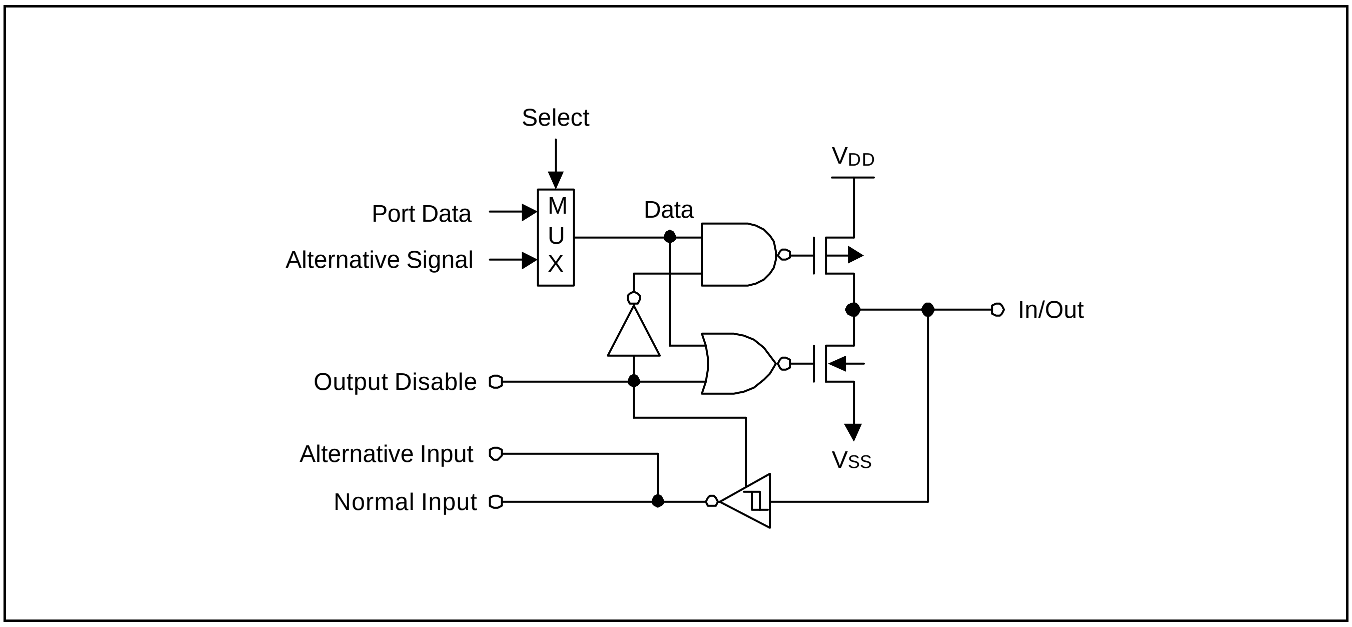


Figure 1-10. Pin Circuit Type 4 (Port 2, Port 8, and Port 9)

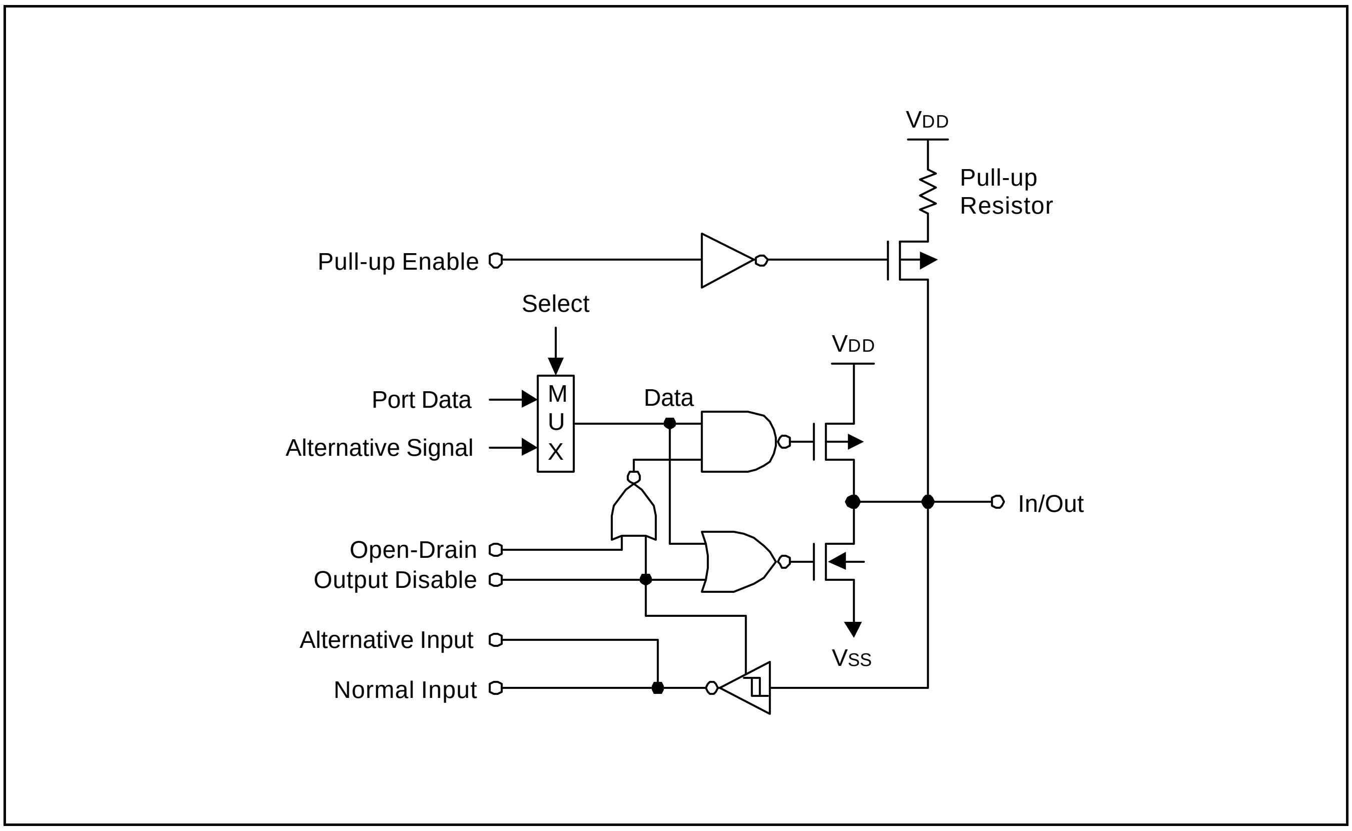


Figure 1-11. Pin Circuit Type 5 (Port 3)

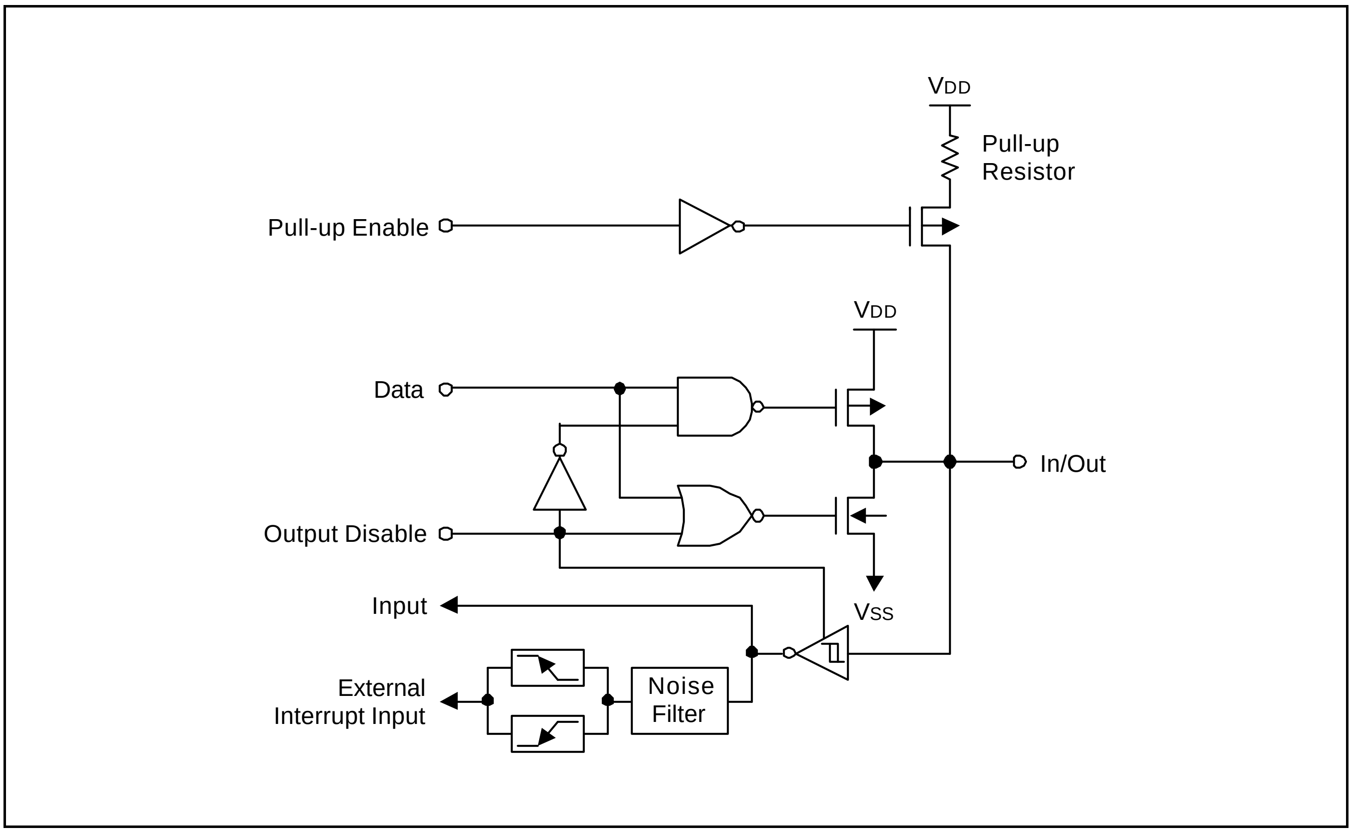


Figure 1-12. Pin Circuit Type 6 (P4.0, and P4.1)

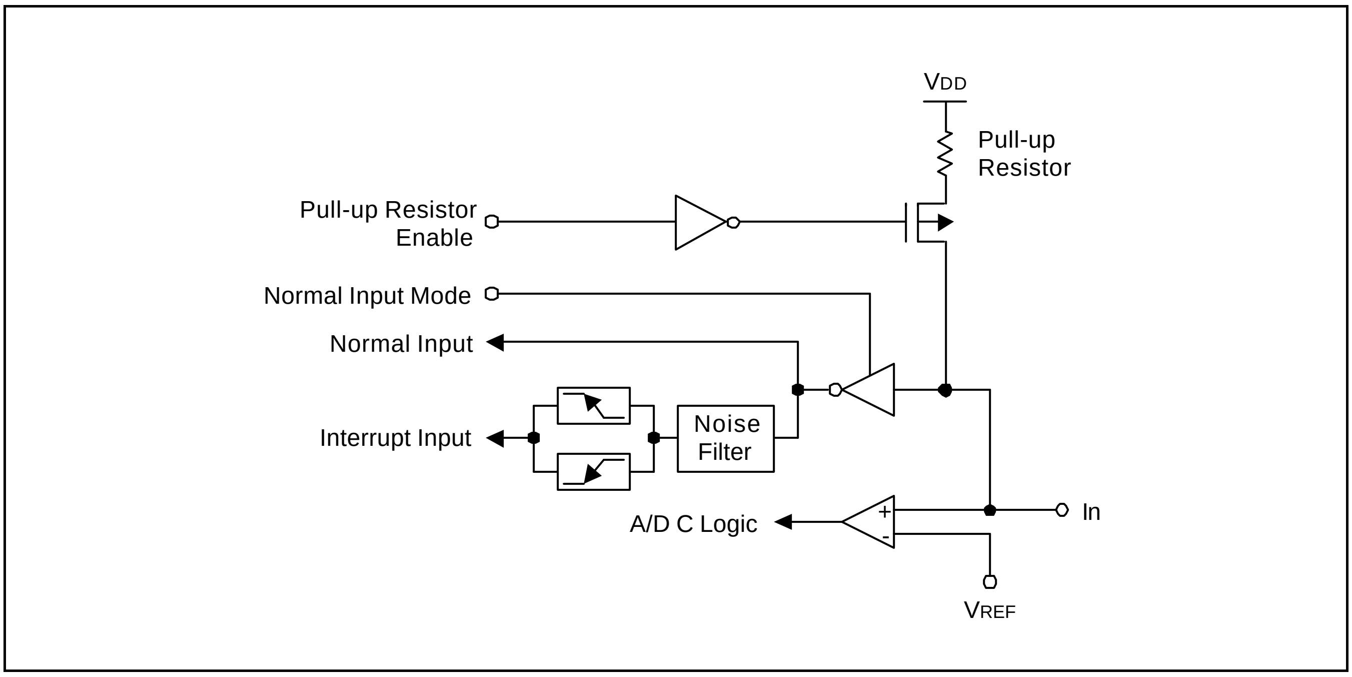


Figure 1-13. Pin Circuit Type 7 (Port 5)

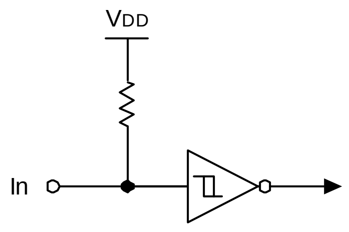


Figure 1-14. Pin Circuit Type 8 (RESET)

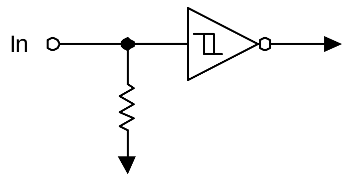


Figure 1-15. Pin Circuit Type 9 (TEST)

NOTES

2 ADDRESS SPACE

OVERVIEW

CalmRISC has 20-bit program address lines, *PA[19:0]*, which supports up to 1M-word program memory.
The 1M-word program memory space is divided into 256 pages and each page is 4K words long as shown in the next page. The upper 8 bits of the program counter, PC[19:12], points to a specific page and the lower 12 bits, PC[11:0], specify the offset address of the page.

CalmRISC also has 16-bit data memory address lines, DA[15:0], which supports up to 64K-byte data memory.
The 64K-byte data memory space is divided into 256 pages and each page has 256 bytes. The upper 8 bits of the data address, DA[15:8], points to a specific page and the lower 8 bits, DA[7:0], specify the offset address of the page.

S3FB42F has 72K-word (144K-byte) flash ROM type program memory, 34.5K-word (69K-byte) flash ROM type data memory and 48K-byte RAM type data memory.

Memory configuration in CalmRISC side

Data Memory:	Total size - 117K bytes (Flash ROM type, 69K bytes and SRAM type, 48K bytes)
Code Memory:	Total size - 144K bytes (Flash ROM type, 144K bytes)

Memory configuration in MAC-2424 side

Data Memory:	X-Memory area - SRAM, 12K LWords (36K bytes) Y-Memory area - SRAM, 4K LWords (12K bytes) and Flash ROM, 23K LWords (69K bytes)
Code Memory:	Total size - 72K words (Flash ROM type, 144K byte)

Memory Type

Flash ROM: 213K bytes

SRAM: 48K bytes

PROGRAM MEMORY (ROM)

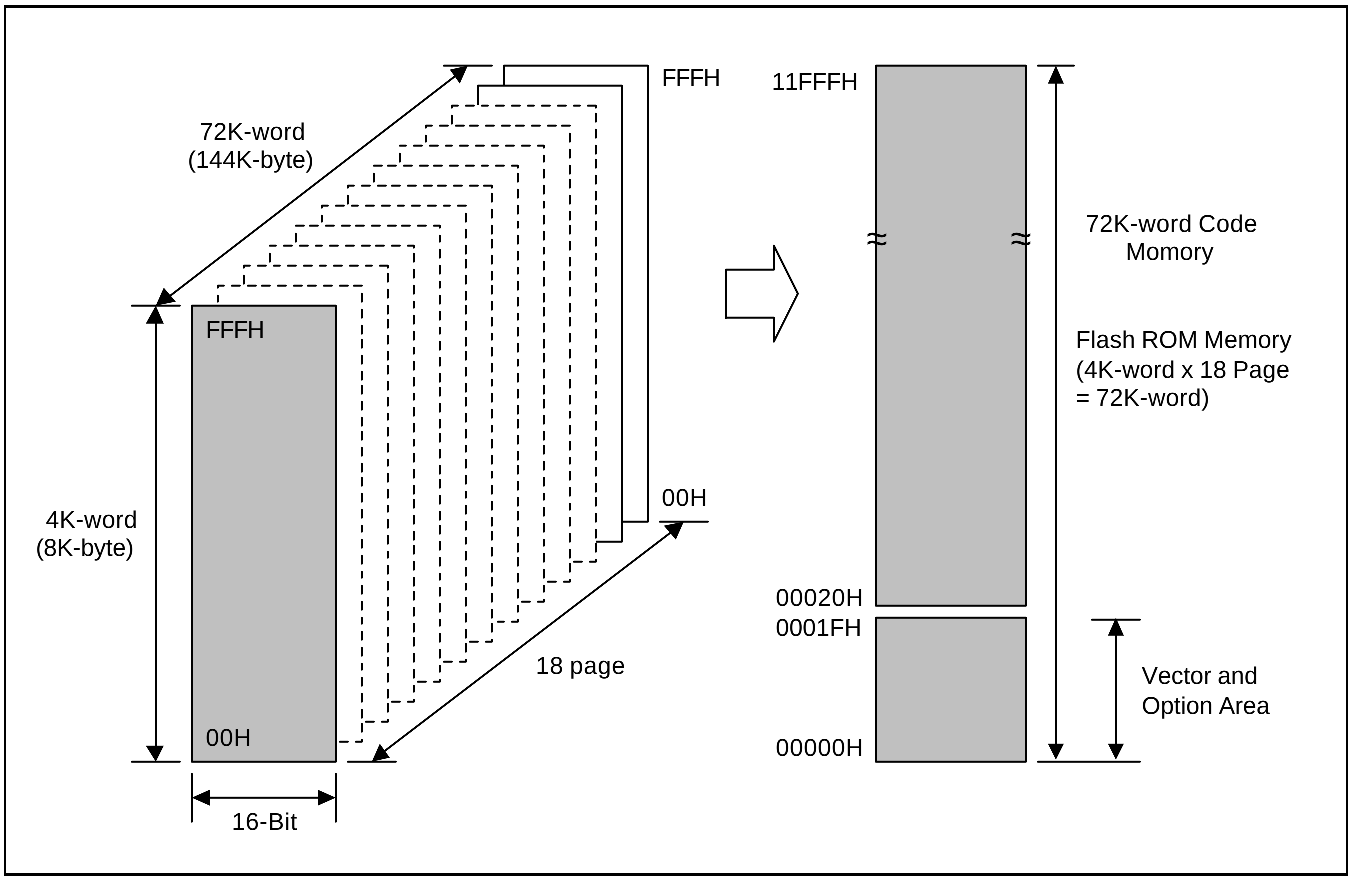


Figure 2-1. Flash Memory (Code Memory Area)

From 00000H to 00004H addresses are used for the vector address of exceptions, and 0001EH, 0001FH are used for the option only. Aside from these addresses others are reserved in the vector and option area. Program memory area from the address 00020H to 11FFH can be used for normal programs.

S3FB42F's program memory is 72K words (144K bytes).

DATA MEMORY ORGANIZATION

The total data memory bank address space is 64 K-byte, addressed by *DA[15:0],* which is also divided into 256 pages, Each page consists of 256 bytes as shown below. S3FB42F has 2 data bank memory.

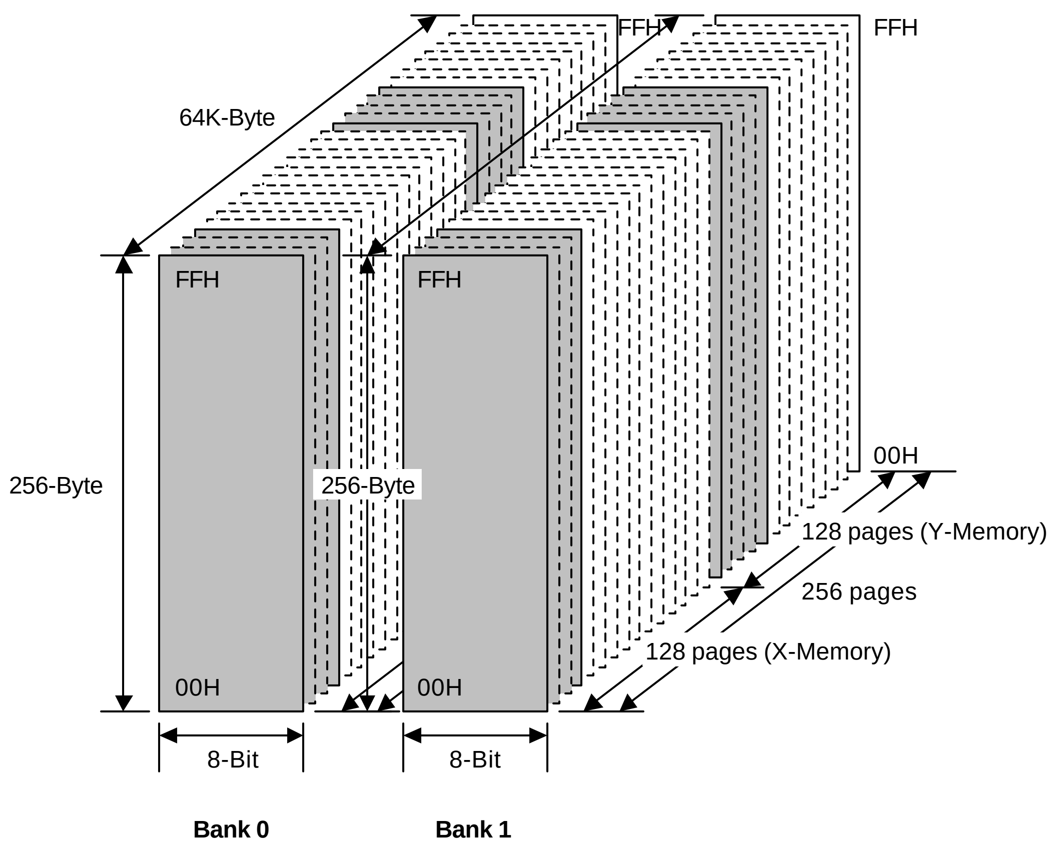
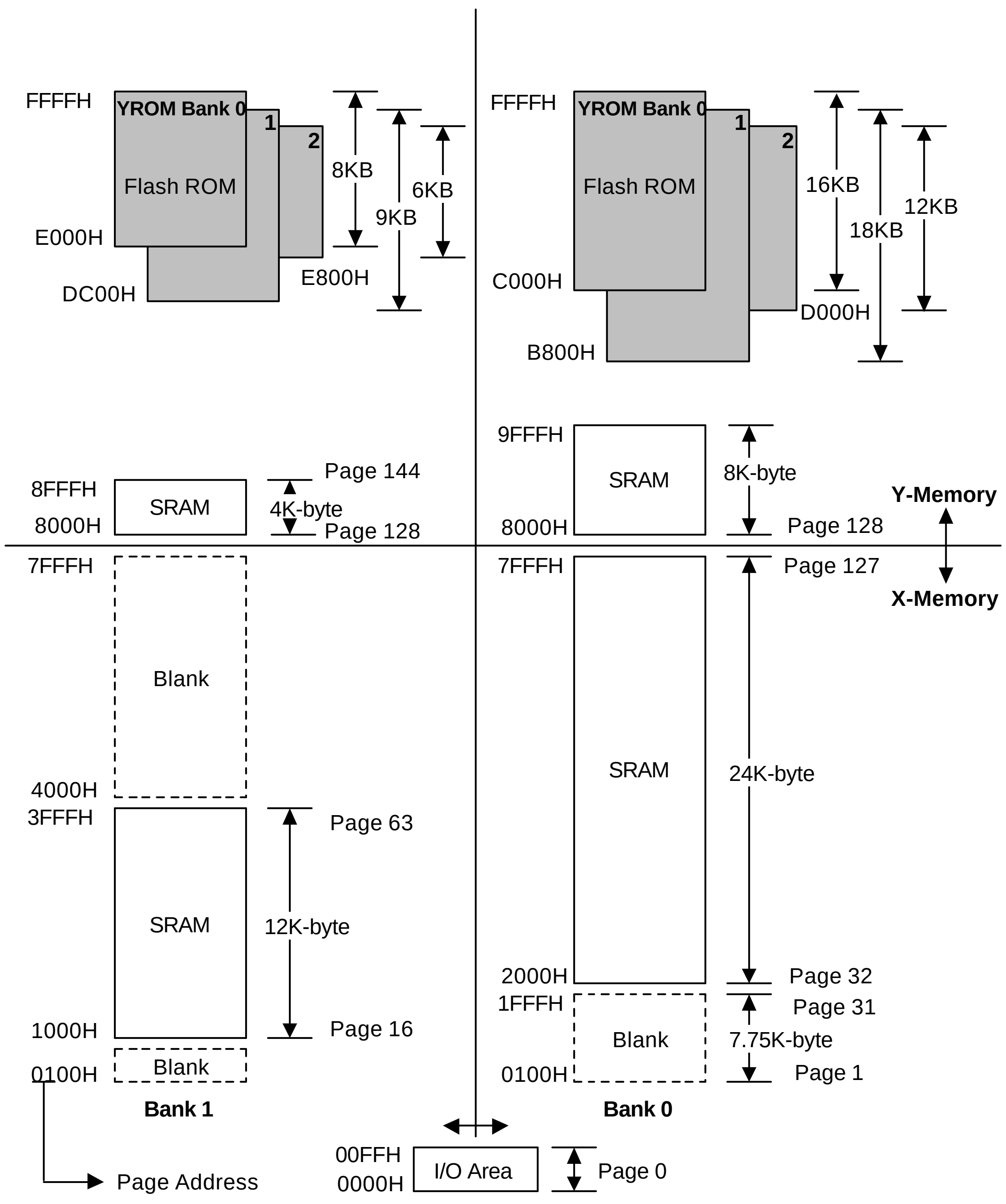
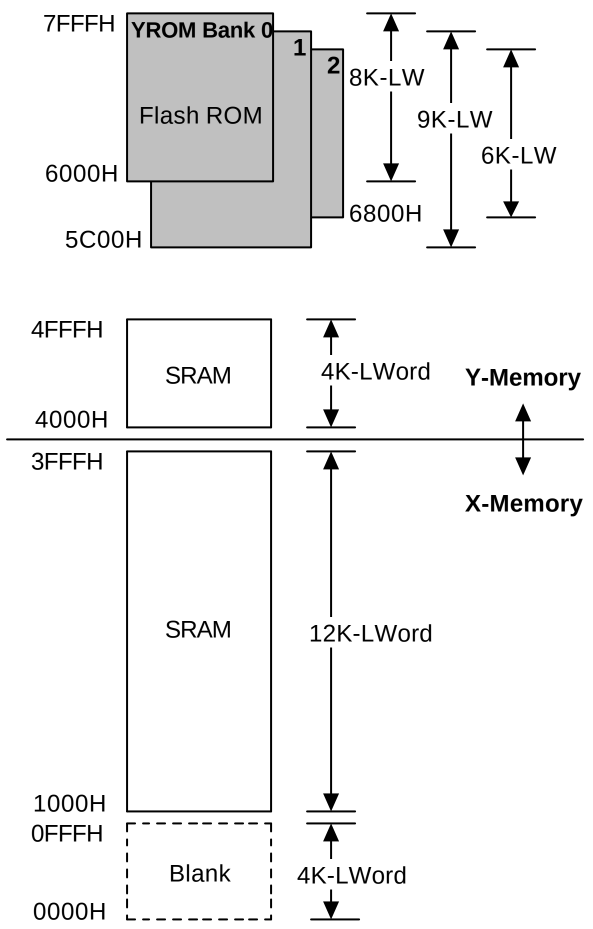


Figure 2-2. Data Memory Map



MAC access unit, LWord = 3-Byte Long.
(1-Byte to Bank 1, and 2-Byte of Bank 0)
BANK 0 XM = 2000H + MAC Offset x 2
YM = 8000H + MAC Offset x 2
BANK 1 XM = 1000H + MAC Offset
YM = 8000H + MAC Offset

Figure 2-3. Data Memory Map in CalmRISC Side



Where, LWord: 3-Byte Long.
(1-Byte of Bank 1, and 2-Byte of Bank 0)
XM MAC Offset = MAC Address - 1000H
YM MAC Offset = MAC Address - 4000H

Figure 2-4. Data Memory Map in MAC-2424 Side

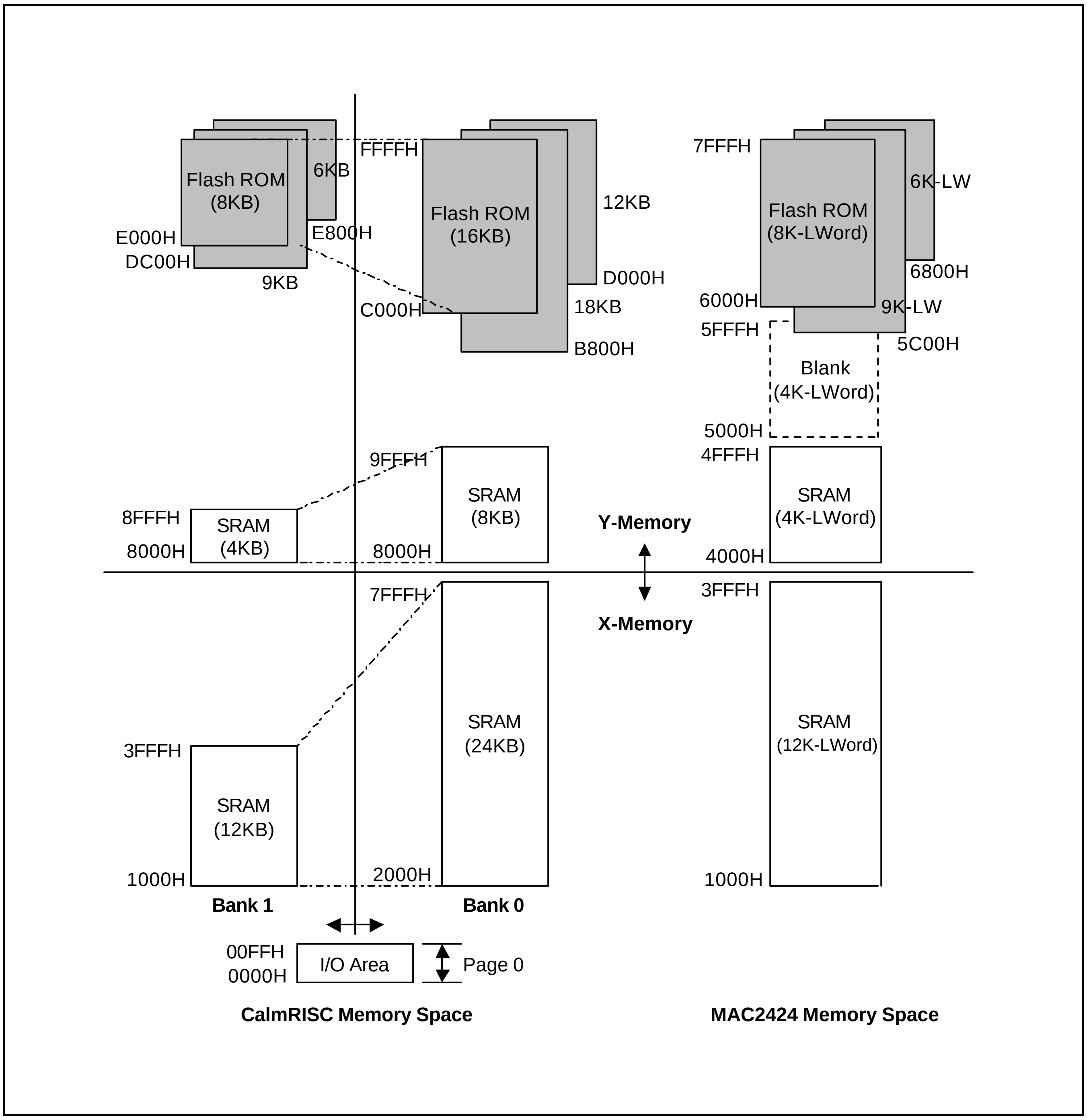


Figure 2-5. Data Memory Map

3 REGISTERS

OVERVIEW

The registers of CalmRISC are grouped into 2 parts: general purpose registers and special purpose registers.

Table 3-1. General and Special Purpose Registers

Registers		Mnemonics	Description	Reset Value
General Purpose Registers (GPR)		**R0**	General **R**egister **0**	Unknown
		R1	General **R**egister **1**	Unknown
		R2	General **R**egister **2**	Unknown
		R3	General **R**egister **3**	Unknown
Special Purpose Registers (SPR)	Group 0 (SPR0)	**IDL0**	**L**ower Byte of **I**n**d**ex Register **0**	Unknown
		IDL1	**L**ower Byte of **I**n**d**ex Register **1**	Unknown
		IDH	**H**igher Byte of **I**n**d**ex Register	Unknown
		SR0	**S**tatus **R**egister **0**	00H
	Group 1 (SPR1)	**ILX**	**I**nstruction Pointer **L**ink Register for E**x**tended Byte	Unknown
		ILH	**I**nstruction Pointer **L**ink Register for **H**igher Byte	Unknown
		ILL	**I**nstruction Pointer **L**ink Register for **L**ower Byte	Unknown
		SR1	**S**tatus **R**egister **1**	Unknown

GPR's can be used in most instructions such as ALU instructions, stack instructions, load instructions, *etc* (See the instruction set sections). From the programming standpoint, they have almost no restriction whatsoever. CalmRISC has 4 banks of GPR's and each bank has 4 registers, R0, R1, R2, and R3. Hence, 16 GPR's in total are available. The GPR bank switching can be done by setting an appropriate value in SR0[4:3] (See SR0 for details). The ALU operations between GPR's from different banks are *not* allowed.

SPR's are designed for their own dedicated purposes. They have some restrictions in terms of instructions that can access them. For example, direct ALU operations cannot be performed on SPR's. However, data transfers between a GPR and an SPR are allowed and stack operations with SPR's are also possible (See the instruction sections for details).

INDEX REGISTERS: IDH, IDL0 AND IDL1

IDH in concatenation with IDL0 (or IDL1) forms a 16-bit data memory address. Note that CalmRISC's data memory address space is 64K bytes (addressable by 16-bit addresses). Basically, IDH points to a page index and IDL0 (or IDL1) corresponds to an offset of the page. Like GPR's, the index registers are 2-way banked. There are 2 banks in total, each of which has its own index registers, IDH, IDL0 and IDL1. The banks of index registers can be switched by setting an appropriate value in SR0[2] (See SR0 for details). Normally, programmers can reserve an index register pair, IDH and IDL0 (or IDL1), for software stack operations.

LINK REGISTERS: ILX, ILH AND ILL

The link registers are specially designed for link-and-branch instructions (See LNK and LRET instructions in the instruction sections for details). When an LNK instruction is executed, the current PC[19:0] is saved into ILX, ILH and ILL registers, i.e., PC[19:16] into ILX[3:0], PC[15:8] into ILH [7:0], and PC[7:0] into ILL[7:0], respectively. When an LRET instruction is executed, the return PC value is recovered from ILX, ILH, and ILL, i.e., ILX[3:0] into PC[19:16], ILH[7:0] into PC[15:8] and ILL[7:0] into PC[7:0], respectively. These registers are used to access program memory by LDC/LDC+ instructions. When an LDC or LDC+ instruction is executed, the (code) data residing at the program address specified by ILX:ILH:ILL will be read into TBH:TBL. LDC+ also increments ILL after accessing the program memory.

There is a special core input pin signal, *nP64KW*, which is reserved for indicating that the program memory address space is only 64 K word. By grounding the signal pin to zero, the upper 4 bits of PC, PC[19:16], is deactivated and therefore the upper 4 bits , PA[19:16], of the program memory address signals from CalmRISC core are also deactivated. By doing so, power consumption due to manipulating the upper 4 bits of PC can be totally eliminated (See the core pin description section for details). From the programmer's standpoint, when *nP64KW* is tied to the ground level, then PC[19:16] is *not* saved into ILX for LNK instructions and ILX is *not* read back into PC[19:16] for LRET instructions. Therefore, ILX is totally unused in LNK and LRET instructions when *nP64KW* = 0.

STATUS REGISTER 0: SR0

SR0 is mainly reserved for system control functions and each bit of SR0 has its own dedicated function.

Table 3-2. Status Register 0: SR0

Flag Name	Bit	Description	Reset Value
eid	0	Data memory page selection in direct addressing	1
ie	1	Global interrupt enable	x
idb	2	Index register banking selection	0
grb[1:0]	4,3	GPR bank selection	00
exe	5	Stack overflow/underflow exception enable	x
ie0	6	Interrupt 0 enable	x
ie1	7	Interrupt 1 enable	x

SR0[0] (or eid) selects which page index is used in direct addressing. If eid = 0, then page 0 (page index = 0) is used. Otherwise (eid = 1), IDH of the current index register bank is used for page index. SR0[1] (or ie) is the global interrupt enable flag. As explained in the interrupt/exception section, CalmRISC has 3 interrupt sources (non-maskable interrupt, interrupt 0, and interrupt 1) and 1 stack exception. Both interrupt 0 and interrupt 1 are masked by setting SR0[1] to 0 (i.e., ie = 0). When an interrupt is serviced, the global interrupt enable flag ie is automatically cleared. The execution of an IRET instruction (return from an interrupt service routine) automatically sets ie = 1. SR0[2] (or idb) and SR0[4:3] (or grb[1:0]) selects an appropriate bank for index registers and GPR's, respectively as shown below:

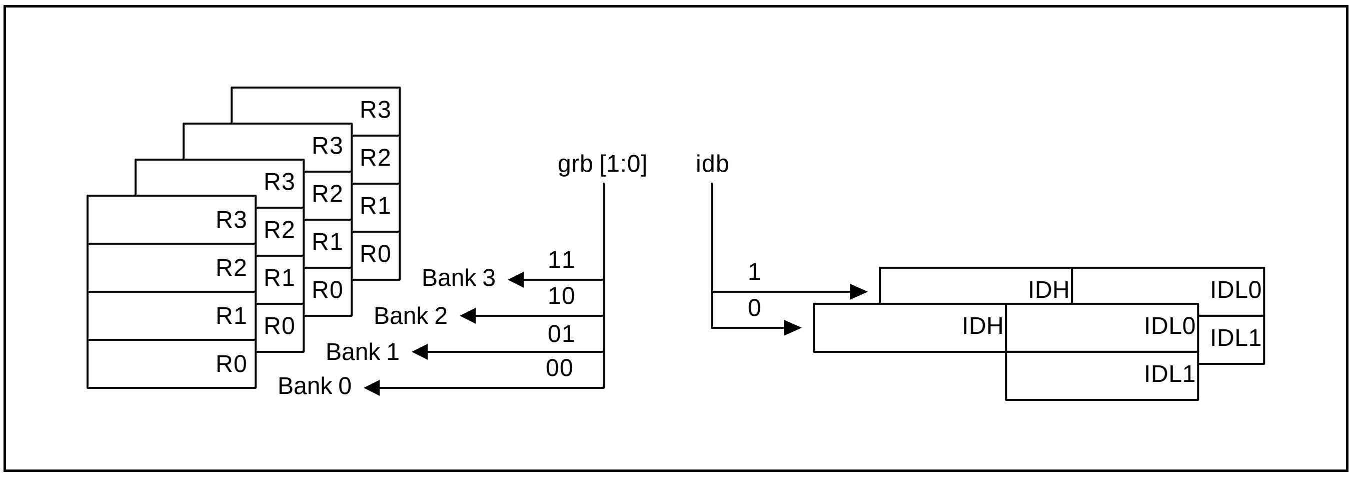


Figure 3-1. Bank Selection by Setting of GRB Bits and IDB Bit

SR0[5] (or exe) enables the stack exception, that is, the stack overflow/underflow exception. If exe = 0, the stack exception is disabled. The stack exception can be used for program debugging in the software development stage. SR0[6] (or ie0) and SR0[7] (or ie1) are enabled, by setting them to 1. Even though ie0 or ie1 are enabled, the interrupts are ignored (not serviced) if the global interrupt enable flag ie is set to 0.

STATUS REGISTER 1: SR1

SR1 is the register for status flags such as ALU execution flag and stack full flag.

Table 3-3. Status Register 1: SR1

Flag Name	Bit	Description
C	0	Carry flag
V	1	Overflow flag
Z	2	Zero flag
N	3	Negative flag
SF	4	Stack Full flag
–	5,6,7	Reserved

SR1[0] (or C) is the carry flag of ALU executions. SR1[1] (or V) is the overflow flag of ALU executions. It is set to 1 if and only if the carry-in into the 8-th bit position of addition/subtraction differs from the carry-out from the 8-th bit position. SR1[2] (or Z) is the zero flag, which is set to 1 if and only if the ALU result is zero. SR1[3] (or N) is the negative flag. Basically, the most significant bit (MSB) of ALU results becomes N flag. Note a load instruction into a GPR is considered an ALU instruction. However, if an ALU instruction touches the overflow flag (V) like ADD, SUB, CP, *etc*, N flag is updated as exclusive-OR of V and the MSB of the ALU result. This implies that even if an ALU operation results in overflow, N flag is still valid. SR1[4] (or SF) is the stack overflow flag. It is set when the hardware stack is overflowed or underflowed. Programmers can check if the hardware stack has any abnormalities by the stack exception or testing if SF is set (See the hardware stack section for great details).

NOTE: When an interrupt occur SR0 and SR1 are not saved by hardware, so the SR1 register values must be saved by software.

4 MEMORY MAP

OVERVIEW

To support the control of peripheral hardware, the address for peripheral control registers are memory-mapped to page 0 of the RAM. Memory mapping lets you use a mnemonic as the operand of an instruction in place of the specific memory location.
In this section, detailed descriptions of the S3FB42F control registers are presented in an easy-to-read format. You can use this section as a quick-reference source when writing application programs.

This memory area can be accessed with the whole method of data memory access.

— If SR0 bit 0 is "0" then the accessed register area is always page 0.

— If SR0 bit 0 is "1" then the accessed register page is controlled by the proper IDH register's value.

So if you want to access the memory map area, clear the SR0.0 and use the direct addressing mode.
This method is used for most cases.
This control register is divided into five areas. Here, the system control register area is same in every device.

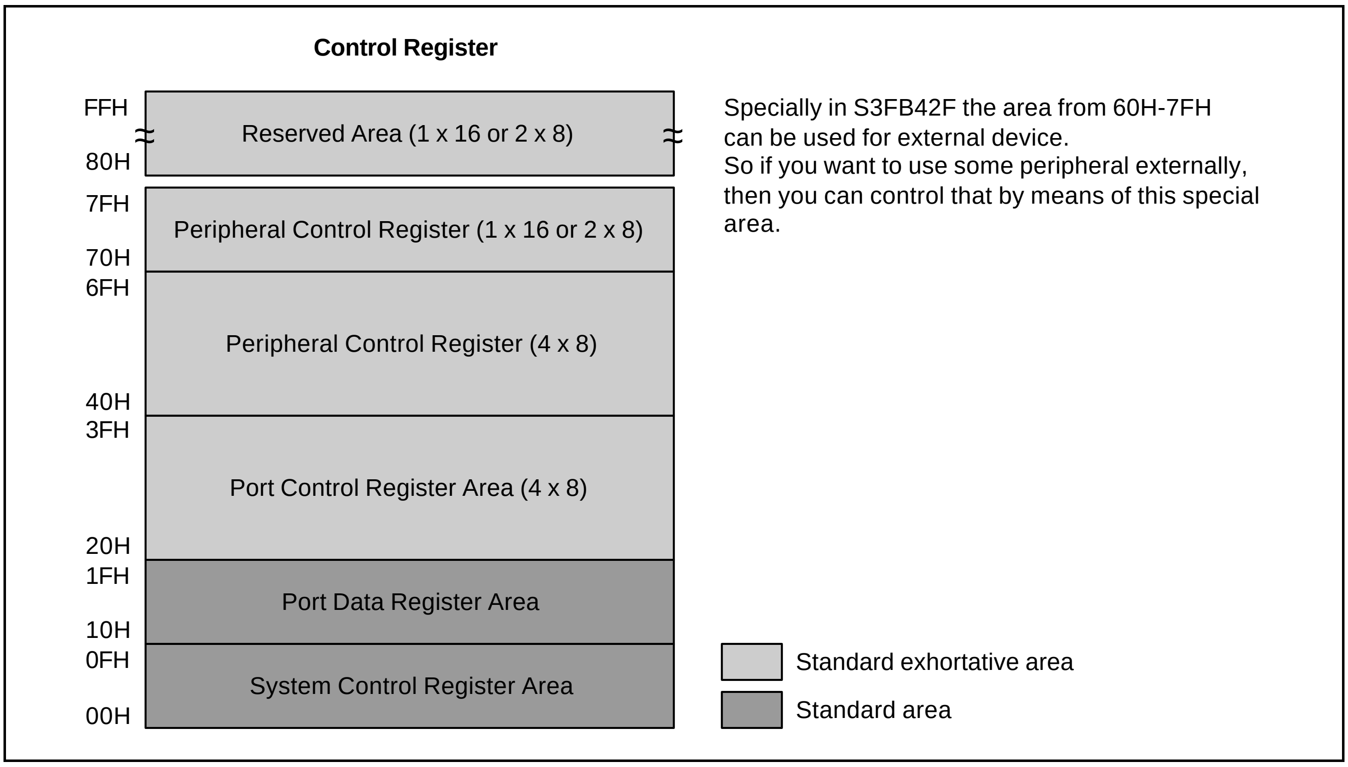

Control Register
FFH
80H
Reserved Area (1 x 16 or 2 x 8)
7FH
70H
Peripheral Control Register (1 x 16 or 2 x 8)
6FH
40H
Peripheral Control Register (4 x 8)
3FH
20H
Port Control Register Area (4 x 8)
1FH
10H
Port Data Register Area
0FH
00H
System Control Register Area
Specially in S3FB42F the area from 60H-7FH can be used for external device.
So if you want to use some peripheral externally, then you can control that by means of this special area.
Standard exhortative area
Standard area

Figure 4-1. Memory Map Area

Table 4-1. Registers

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
Location 1AH-1FH are not mapped					
Port 9 data register	P9	25	19H	00H	R/W
Port 8 data register	P8	24	18H	00H	R/W
Port 7 data register	P7	23	17H	00H	R/W
Port 6 data register	P6	22	16H	00H	R/W
Port 5 data register	P5	21	15H	00H	R
Port 4 data register	P4	20	14H	00H	R/W
Port 3 data register	P3	19	13H	00H	R/W
Port 2 data register	P2	18	12H	00H	R/W
Port 1 data register	P1	17	11H	00H	R/W
Port 0 data register	P0	16	10H	00H	R/W
Watchdog timer control register	WDTCON	15	0FH	00H	R/W
Watchdog timer enable register	WDTEN	14	0EH	00H	R/W
Basic timer counter	BTCNT	13	0DH	00H	R
Basic timer control register	BTCON	12	0CH	00H	R/W
Interrupt ID register 1	IIR1	11	0BH	–	R/W
Interrupt priority register 1	IPR1	10	0AH	00H	R/W
Interrupt mask register 1	IMR1	9	09H	00H	R/W
Interrupt request register 1	IRQ1	8	08H	–	R/W
Interrupt ID register 0	IIR0	7	07H	–	R/W
Interrupt priority register 0	IPR0	6	06H	00H	R/W
Interrupt mask register 0	IMR0	5	05H	00H	R/W
Interrupt request register 0	IRQ0	4	04H	-	R/W
Oscillator control register	OSCCON	3	03H	00H	R/W
Power control register (stop or idle mode)	PCON	2	02H	04H	R/W
Locations 00H-01H are not mapped					

NOTES:
1. '–' means underlined.
2. If you want to clear the bit of IRQx, then write the number which you want to clear to IIRx. For example, when clear IRQ0.4 then LD R0, #04H and LD IIR0, R0.

Table 4-1. Registers (Continued)

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
Port 0 control register	P0CON	32	20H	00H	R/W
Port 1 control register	P1CON	33	21H	00H	R/W
Port 2 control register low	P2CONL	34	22H	00H	R/W
Port 2 control register high	P2CONH	35	23H	30H	R/W
Port 3 control register low	P3CONL	36	24H	00H	R/W
Port 3 control register high	P3CONH	37	25H	00H	R/W
Port 3 pull-up resistor	P3PUR	38	26H	00H	R/W
Location 27H is not mapped					
Port 5 control register	P5CON	40	28H	00H	R/W
Port 5 pull-up resistor	P5PUR	41	29H	00H	R/W
Port 5 Int. control register	P5INTCON	42	2AH	00H	R/W
Port 5 Int. mode register low	P5INTMODL	43	2BH	00H	R/W
Port 5 Int. mode register High	P5INTMODH	44	2CH	00H	R/W
External Int. pending register	EINTPND	45	2DH	00H	R/W
Locations 2E-2FH are not mapped					
Port 4 control register	P4CON	48	30H	00H	R/W
Port 4 Int. control register	P4INTCON	49	31H	00H	R/W
Port 4 Int. mode register	P4INTMOD	50	32H	00H	R/W
Location 33H is not mapped					
Port 6 control register	P6CON	52	34H	00H	R/W
Port 7 control register	P7CON	53	35H	00H	R/W
Port 8 control register	P8CON	54	36H	00H	R/W
Port 9 control register	P9CON	55	37H	00H	R/W
Locations 38H-3FH are not mapped					
Timer A control register	TACON	64	40H	00H	R/W
Timer A data register	TADATA	65	41H	00H	R/W
Timer A counter	TACNT	66	42H	–	R
Location 43H is not mapped					
Timer B control register	TBCON	68	44H	00H	R/W
Timer B data register	TBDATA	69	45H	00H	R/W
Timer B counter	TBCNT	70	46H	–	R
Locations 47H-4BH are not mapped					
Watch timer control register	WTCON	76	4CH	00H	R/W
Locations 4DH-4FH are not mapped					

Table 4-1. Registers (Continued)

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
Serial I/O control register	SIOCON	80	50H	00H	R/W
Serial I/O pre-scale register	SIOPS	81	51H	00H	R/W
Serial I/O data register	SIODATA	82	52H	00H	R/W
Location 53H is not mapped					
A/D C control register	ADCON	84	54H	00H	R/W
A/D conversion result data register	ADDATA	85	55H	–	R
Locations 56H-57H are not mapped					
IIS control register 0	IISCON0	88	58H	00H	R/W
IIS mode register 0	IISMODE0	89	59H	00H	R/W
IIS buffer pointer register 0	IISPTR0	90	5AH	00H	R/W
Location 5BH is not mapped					
IIS control register 1	IISCON1	92	5CH	00H	R/W
IIS mode register 1	IISMODE1	93	5DH	00H	R/W
IIS buffer pointer register 1	IISPTR1	94	5EH	00H	R/W
Location 5FH is not mapped					
Parallel port data register	PPDATA	96	60H	00H	R/W
Parallel port command data register	PPCDATA	97	61H	00H	R/W
Parallel port status control register	PPSCON	98	62H	08H	R/W
Parallel port status register	PPSTAT	99	63H	3FH	R/W
Parallel port control register low	PPCONL	100	64H	00H	R/W
Parallel port control register high	PPCONH	101	65H	00H	R/W
Parallel port int. control register low	PPINTCONL	102	66H	00H	R/W
Parallel port int. control register high	PPINTCONH	103	67H	00H	R/W
Parallel port int. pending register low	PPINTPNDL	104	68H	00H	R/W
Parallel port int. pending register high	PPINTPNDH	105	69H	00H	R/W
Parallel port ack. width data register	PPACKD	106	6AH	xxH	R/W
Locations 6BH-6FH are not mapped					
SmartMedia control register	SMCON	112	70H	00H	R/W
ECC counter	ECCNT	113	71H	00H	R/W
ECC data register low	ECCL	114	72H	00H	R/W
ECC data register high	ECCH	115	73H	00H	R/W
ECC data register extension	ECCX	116	74H	00H	R/W
ECC result register low	ECCRSTL	117	75H	00H	R/W

Table 4-1. Registers (Continued)

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
ECC result register high	ECCRSTH	118	76H	00H	R/W
ECC clear register	ECCCLR	119	77H	–	W
Flash memory control register	FMCON	120	78H	00H	R/W
Location 79H is not mapped					
Flash user programming serial clock register	FSCLK	122	7AH	00H	R/W
Flash user programming serial data register	FSDAT	123	7BH	00H	R/W
Locations 7CH-7FH are not mapped					
Function address register	FUNADDR	128	80H	00H	R
Power management register	PWRMAN	129	81H	00H	R
Frame number LO register	FRAMELO	130	82H	00H	R
Frame number HI register	FRAMEHI	131	83H	00H	R
Interrupt pending register	INTREG	132	84H	00H	R/W
Interrupt enable register	INTENA	133	85H	00H	R/W
Endpoint index register	EPINDEX	134	86H	00H	R/W
Locations 87H-88H are not mapped					
Endpoint direction register	EPDIR	137	89H	00H	W
IN control status register	INCSR	138	8AH	00H	R/W
OUT control status register	OUTCSR	139	8BH	00H	R/W
IN MAX packet register	INMAXP	140	8CH	00H	R/W
OUT MAX packet register	OUTMAXP	141	8DH	00H	R/W
Write counter LO register	WRTCNTLO	142	8EH	00H	R/W
Write counter HI register	WRTCNTHI	143	8FH	00H	R/W
Endpoint 0 FIFO register	EP0FIFO	144	90H	00H	R/W
Endpoint 1 FIFO register	EP1FIFO	145	91H	00H	R/W
Endpoint 2 FIFO register	EP2FIFO	146	92H	00H	R/W
Endpoint 3 FIFO register	EP3FIFO	147	93H	00H	R/W

Table 4-1. Registers (Continued)

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
Control register for random number generator	RANCON	168	A8H	–	R/W
8-bit linear feedback shift register	LFSR8	169	A9H	–	R/W
16-bit linear feedback shift register lower	LFSR16L	170	AAH	–	R/W
16-bit linear feedback shift register higher	LFSR16H	171	ABH	–	R/W
PLL data register lower	PLLDATAL	172	ACH	–	R/W
PLL data register higher	PLLDATAH	173	ADH	–	R/W
PLL control register	PLLCON	174	AEH	0	R/W
Location AFH is not mapped					
UART line control register	LCON	176	B0H	00H	R/W
UART control register	UCON	177	B1H	00H	R/W
UART status register	USSR	178	B2H	C0H	R
UART transmit buffer register	TBR	179	B3H	–	W
UART receive buffer register	RBR	180	B4H	–	R
UART band rate divisor register	UBRDR	181	B5H	00H	R/W
UART interrupt pending register	UPEND	182	B6H	00	R/W
Location BFH is not mapped					
IIC control register	IICCON	184	B8H	00H	R/W
IIC status register	IICSR	185	B9H	00H	R/W
IIC data register	IICDATA	186	BAH	–	R/W
IIC address register	IICADDR	187	BBH	–	R/W
IIC pre-scaler register	IICPS	188	BCH	FFH	R/W
IIC pre-scaler count register for test	IICCNT	189	BDH	–	R
Locations BEH-BFH is not mapped					
64-byte IIS I/O buffer	BUF64		C0H FFH	–	R/W

SAMSUNG ELECTRONICS

5 HARDWARE STACK

OVERVIEW

The hardware stack in CalmRISC has two usages:

— To save and restore the return PC[19:0] on LCALL, CALLS, RET, and IRET instructions.

— Temporary storage space for registers on PUSH and POP instructions.

When PC[19:0] is saved into or restored from the hardware stack, the access should be 20 bits wide. On the other hand, when a register is pushed into or popped from the hardware stack, the access should be 8 bits wide. Hence, to maximize the efficiency of the stack usage, the hardware stack is divided into 3 parts: the extended stack bank (XSTACK, 4-bits wide), the odd bank (8-bits wide), and the even bank (8-bits wide).

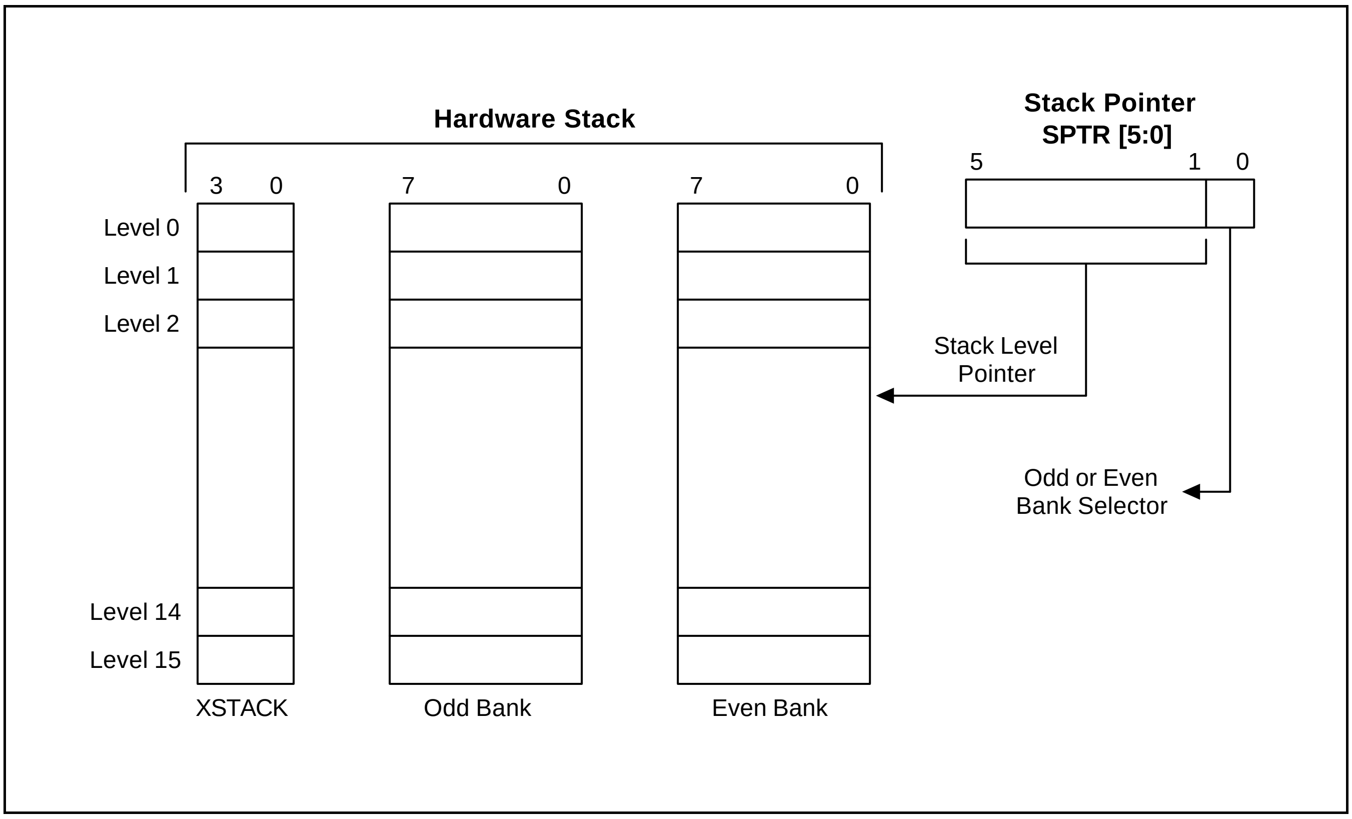


Figure 5-1. Hardware Stack

The top of the stack (TOS) is pointed to by a stack pointer, called **sptr[5:0]**. The upper 5 bits of the stack pointer, sptr[5:1], points to the stack level into which either PC[19:0] or a register is saved. For example, if sptr[5:1] is 5H or TOS is 5, then level 5 of XSTACK is empty and either level 5 of the odd bank or level 5 of the even bank is empty. In fact, sptr[0], the stack bank selection bit, indicates which bank(s) is empty. If sptr[0] = 0, both level 5 of the even and the odd banks are empty. On the other hand, if sptr[0] = 1, level 5 of the odd bank is empty, but level 5 of the even bank is occupied. This situation is well illustrated in the figure below.

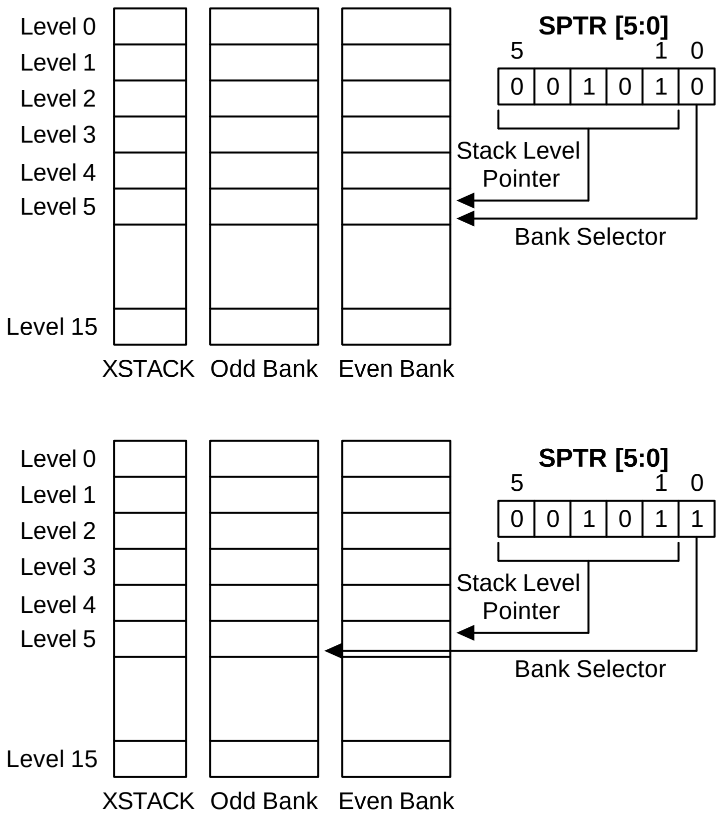


Figure 5-2. Even and Odd Bank Selection Example

As can be seen in the above example, sptr[5:1] is used as the hardware stack pointer when PC[19:0] is pushed or popped and sptr[5:0] as the hardware stack pointer when a register is pushed or popped. Note that XSTACK is used only for storing and retrieving PC[19:16]. Let us consider the cases where PC[19:0] is pushed into the hardware stack (by executing LCALL/CALLS instructions or by interrupts/exceptions being served) or is retrieved from the hardware stack (by executing RET/IRET instructions). Regardless of the stack bank selection bit (sptr[0]), TOS of the even bank and the odd bank store or return PC[7:0] or PC[15:8], respectively. This is illustrated in the following figures.

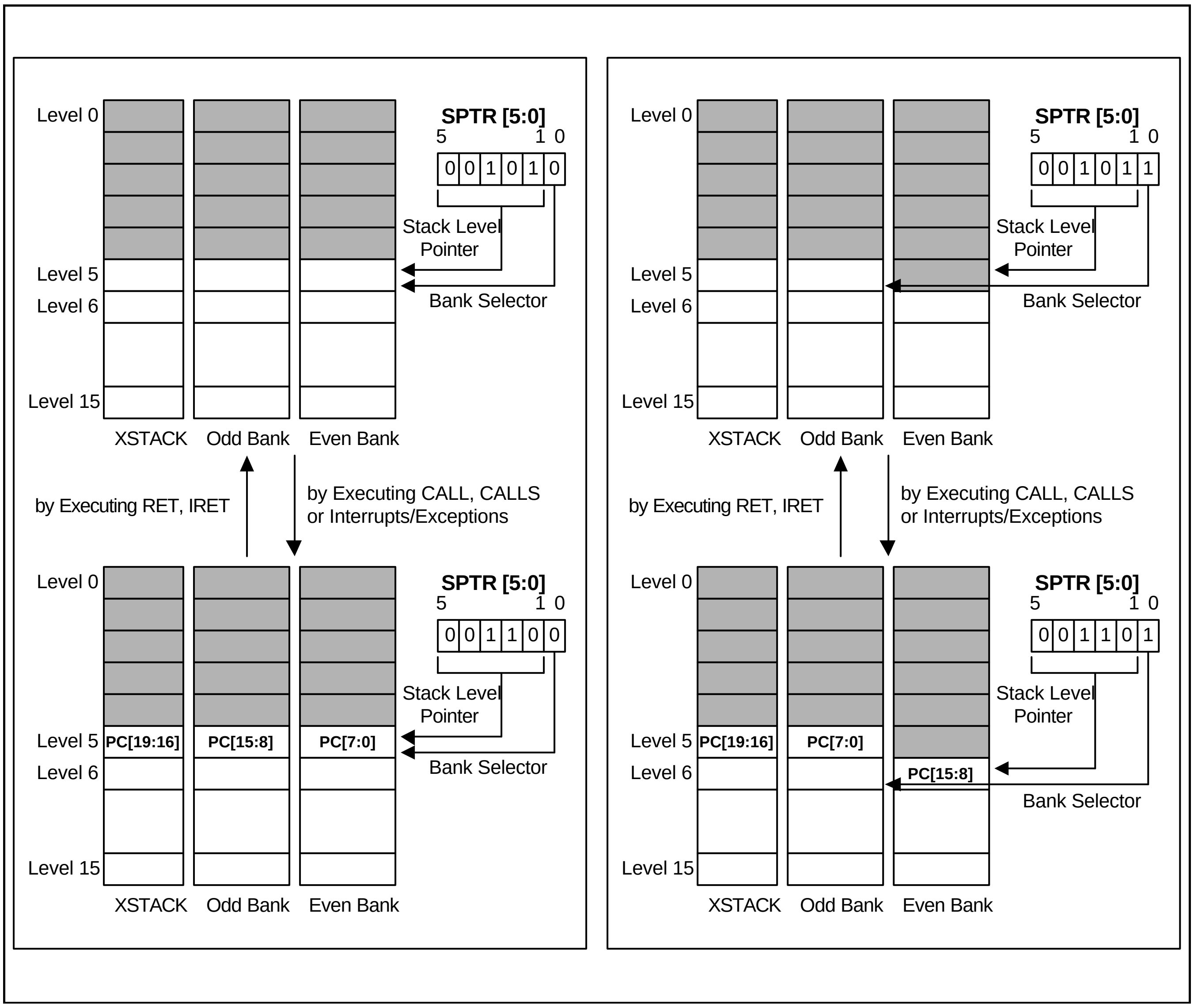


Figure 5-3. Stack Operation with PC [19:0]

As can be seen in the figures, when stack operations with PC[19:0] are performed, the stack level pointer sptr[5:1] (*not* sptr[5:0]) is either incremented by 1 (when PC[19:0] is pushed into the stack) or decremented by 1 (when PC[19:0] is popped from the stack). The stack bank selection bit (sptr[0]) is unchanged. If a CalmRISC core input signal *nP64KW* is 0, which signifies that only PC[15:0] is meaningful, then any access to XSTACK is totally deactivated from the stack operations with PC. Therefore, XSTACK has no meaning when the input pin signal, *nP64KW*, is tied to 0. In that case, XSTACK doesn't have to even exist. As a matter of fact, XSTACK is not included in CalmRISC core itself and it is interfaced through some specially reserved core pin signals (*nPUSH, nSTACK*, *XHSI[3:0]*, *XSHO[3:0]*), if the program address space is more than 64K words (See the core pin signal section for details).

With regards to stack operations with registers, a similar argument can be made. The only difference is that the data written into or read from the stack are a byte. Hence, the even bank and the odd bank are accessed alternately as shown below.

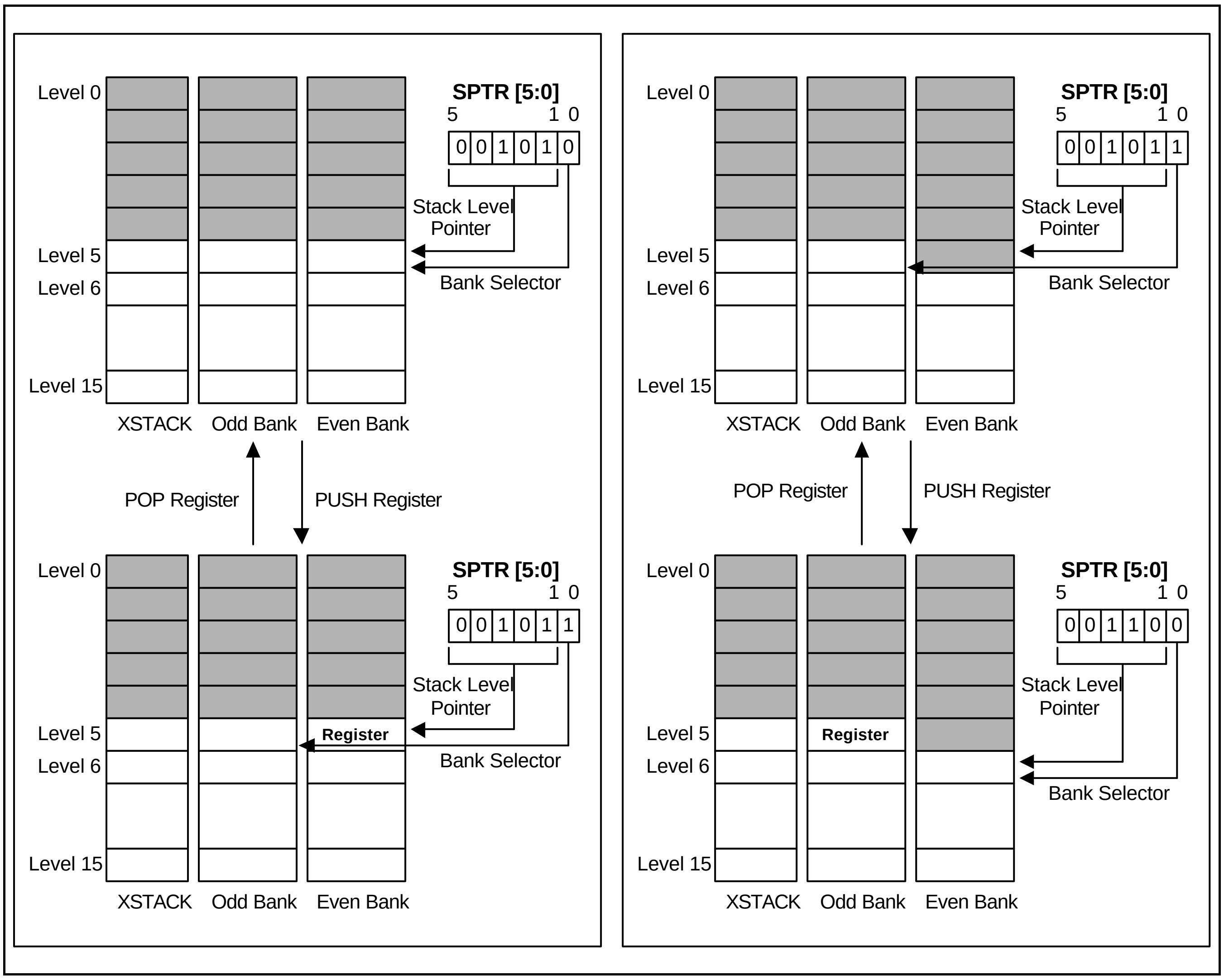


Figure 5-4. Stack Operation with Registers

When the bank selection bit (sptr[0]) is 0, then the register is pushed into the even bank and the bank selection bit is set to 1. In this case, the stack level pointer is unchanged. When the bank selection bit (sptr[0]) is 1, then the register is pushed into the odd bank, the bank selection bit is set to 0, and the stack level pointer is incremented by 1. Unlike the push operations of PC[19:0], any data are not written into XSTACK in the register push operations. This is illustrated in the example figures. When a register is pushed into the stack, sptr[5:0] is incremented by 1 (*not* the stack level pointer sptr[5:1]). The register pop operations are the reverse processes of the register push operations. When a register is popped out of the stack, sptr[5:0] is decremented by 1 (*not* the stack level pointer sptr[5:1]).

Hardware stack overflow/underflow happens when the MSB of the stack level pointer, sptr[5], is 1. This is obvious from the fact that the hardware stack has only 16 levels and the following relationship holds for the stack level pointer in a normal case.

Suppose the stack level pointer sptr[5:1] = 15 (or 01111B in binary format) and the bank selection bit sptr[0] = 1. Here if either PC[19:0] or a register is pushed, the stack level pointer is incremented by 1. Therefore, sptr[5:1] = 16 (or 10000B in binary format) and sptr[5] = 1, which implies that the stack is overflowed. The situation is depicted in the following.

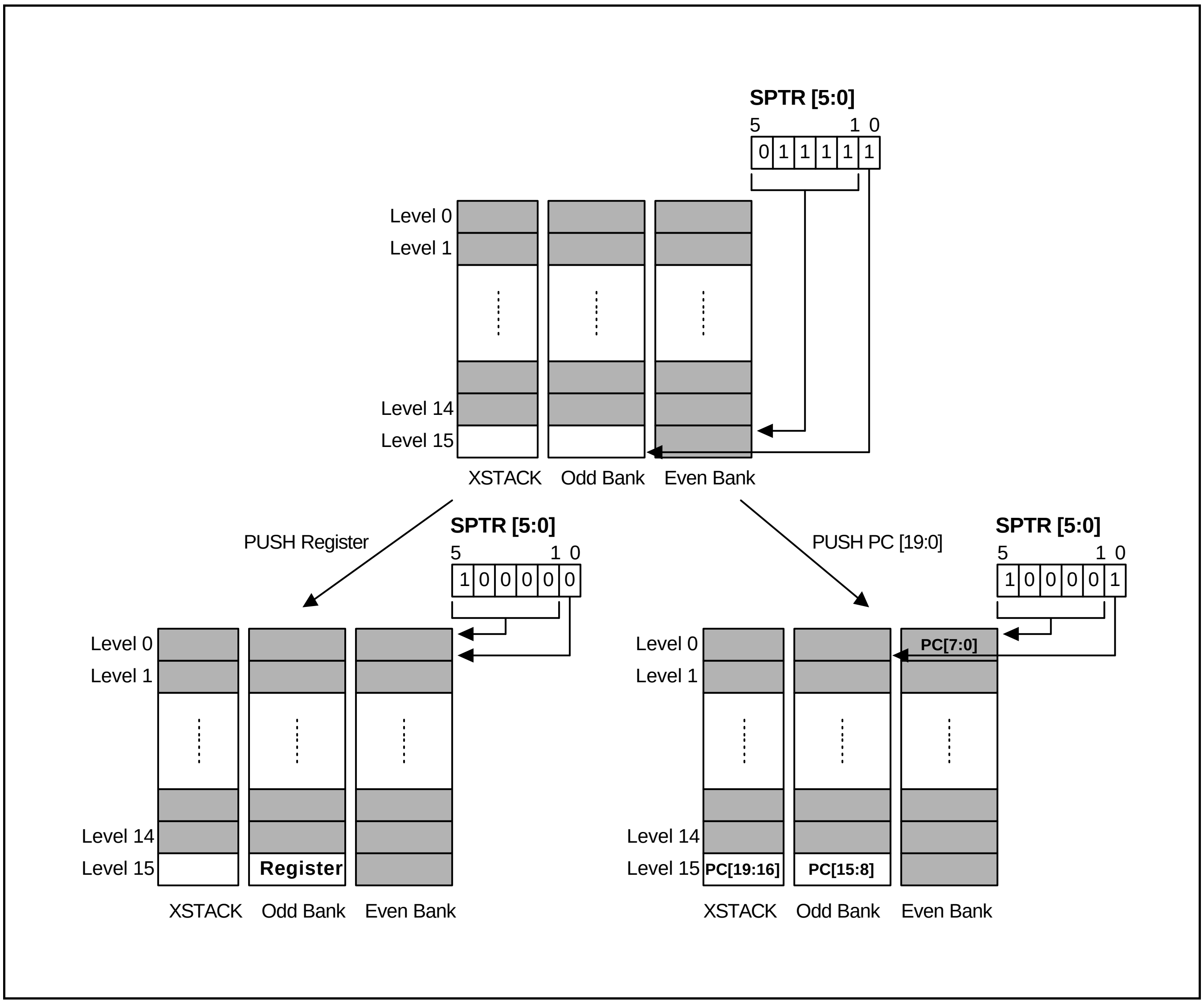


Figure 5-5. Stack Overflow

The first overflow happens due to a register push operation. As explained earlier, a register push operation increments sptr[5:0] (not sptr[5:1]) , which results in sptr[5] = 1, sptr[4:1] = 0 and sptr[0] = 0. As indicated by sptr[5] = 1, an overflow happens. Note that this overflow doesn't overwrite any data in the stack. On the other hand, when PC[19:0] is pushed, sptr[5:1] is incremented by 1 instead of sptr[5:0], and as expected, an overflow results. Unlike the first overflow, PC[7:0] is pushed into level 0 of the even bank and the data that has been there before the push operation is *overwritten*. A similar argument can be made about stack underflows. Note that any stack operation, which causes the stack to overflow or underflow, doesn't necessarily mean that any data in the stack are lost, as is observed in the first example.

In SR1, there is a status flag, SF (Stack Full Flag), which is exactly the same as sptr[5]. In other words, the value of sptr[5] can be checked by reading SF (or SR1[4]). SF is not a sticky flag in the sense that if there was a stack overflow/underflow but any following stack access instructions clear sptr[5] to 0, then SF = 0 and programmers cannot tell whether there was a stack overflow/underflow by reading SF. For example, if a program pushes a register 64 times in a row, sptr[5:0] is exactly the same as sptr[5:0] before the push sequence. Therefore, special attention should be paid.

Another mechanism to detect a stack overflow/underflow is through a stack exception. A stack exception happens only when the execution of any stack access instruction results in SF = 1 (or sptr[5] = 1). Suppose a register push operation makes SF = 1 (the SF value before the push operation doesn’t matter). Then the stack exception due to the push operation is immediately generated and served If the stack exception enable flag (exe of SR0) is 1. If the stack exception enable flag is 0, then the generated interrupt is not served but pending. Sometime later when the stack exception enable flag is set to 1, the pending exception request is served even if SF = 0. More details are available in the stack exception section.

6 EXCEPTIONS

OVERVIEW

Exceptions in CalmRISC are listed in the table below. Exception handling routines, residing at the given addresses in the table, are invoked when the corresponding exception occurs. The starting address of each exception routine is specified by concatenating 0H (leading 4 bits of 0) and the 16-bit data in the exception vector listed in the table. For example, the interrupt service routine for NMI starts from 0H:PM[00001H]. Note that ":" means concatenation and PM[*] stands for the 16-bit content at the address * of the program memory. Aside from the exception due to reset release, the current PC is pushed in the stack on an exception. When an exception is executed due to NMI/IRQ[1:0]/IEXP, the global interrupt enable flag, ie bit (SR0[1]), is set to 0, whereas ie is set to 1 when IRET or an instruction that explicitly sets ie is executed.

Table 6-1. Exceptions

Name	Address	Priority	Description
Reset	00000H	1 st	Exception due to reset release.
NMI	00001H	2 nd	Exception due to *nNMI* signal. Non-maskable.
IRQ[0]	00002H	4 th	Exception due to *nIRQ[0]* signal. Maskable by setting ie/ie0.
IRQ[1]	00003H	5 th	Exception due to *nIRQ[1]* signal. Maskable by setting ie/ie1.
IEXP	00004H	3 rd	Exception due to stack full. Maskable by setting exe.
–	00005H	–	Reserved.
–	00006H	–	Reserved.
–	00007H	–	Reserved.

NOTE: Break mode due to BKREQ has a higher priority than all the exceptions above. That is, when BKREQ is active, even the exception due to reset release is not executed.

HARDWARE RESET

When Hardware Reset is active (the reset input signal pin *nRES* = 0), the control pins in the CalmRISC core are initialized to be disabled, and SR0 and sptr (the hardware stack pointer) are initialized to be 0. Additionally, the interrupt sensing block is cleared. When Hardware Reset is released (nRES = 1), the reset exception is executed by loading the JP instruction in IR (Instruction Register) and 0h:0000h in PC. Therefore, when Hardware Reset is released, the "JP {0h:PM[00000h]}" instruction is executed. When the reset exception is executed, a core output signal *nEXPACK* is generated to acknowledge the exception.

NMI EXCEPTION (EDGE SENSITIVE)

On the falling edge of a core input signal *nNMI*, the NMI exception is executed by loading the CALL instruction in IR and 0h:0001h in PC. Therefore, when NMI exception is activated, the "CALL {0h:PM[00001h]}" instruction is executed. When the NMI exception is executed, the ie bit (SR0[1]) becomes 0 and a core output signal *nEXPACK* is generated to acknowledge the exception.

IRQ[0] EXCEPTION (LEVEL-SENSITIVE)

When a core input signal *nIRQ[0]* is low, SR0[6] (ie0) is high, and SR0[1] (ie) is high, IRQ[0] exception is generated, and this will load the CALL instruction in IR (Instruction Register) and 0h:0002h in PC. Therefore, on an IRQ[0] exception, the "CALL {0h:PM[00002h]}" instruction is executed. When the IRQ[0] exception is executed, SR0[1] (ie) is set to 0 and a core output signal *nEXPACK* is generated to acknowledge the exception.

IRQ[1] EXCEPTION (LEVEL-SENSITIVE)

When a core input signal *nIRQ[1]* is low, SR0[7] (ie1) is high, and SR0[1] (ie) is high, IRQ[1] exception is generated, and this will load the CALL instruction in IR (Instruction Register) and 0h:0003h in PC. Therefore, on an IRQ[1] exception, the "CALL {0h:PM[00003h]}" instruction is executed. When the IRQ[1] exception is executed, SR0[1] (ie) is set to 0 and a core output signal *nEXPACK* is generated to acknowledge the exception.

HARDWARE STACK FULL EXCEPTION

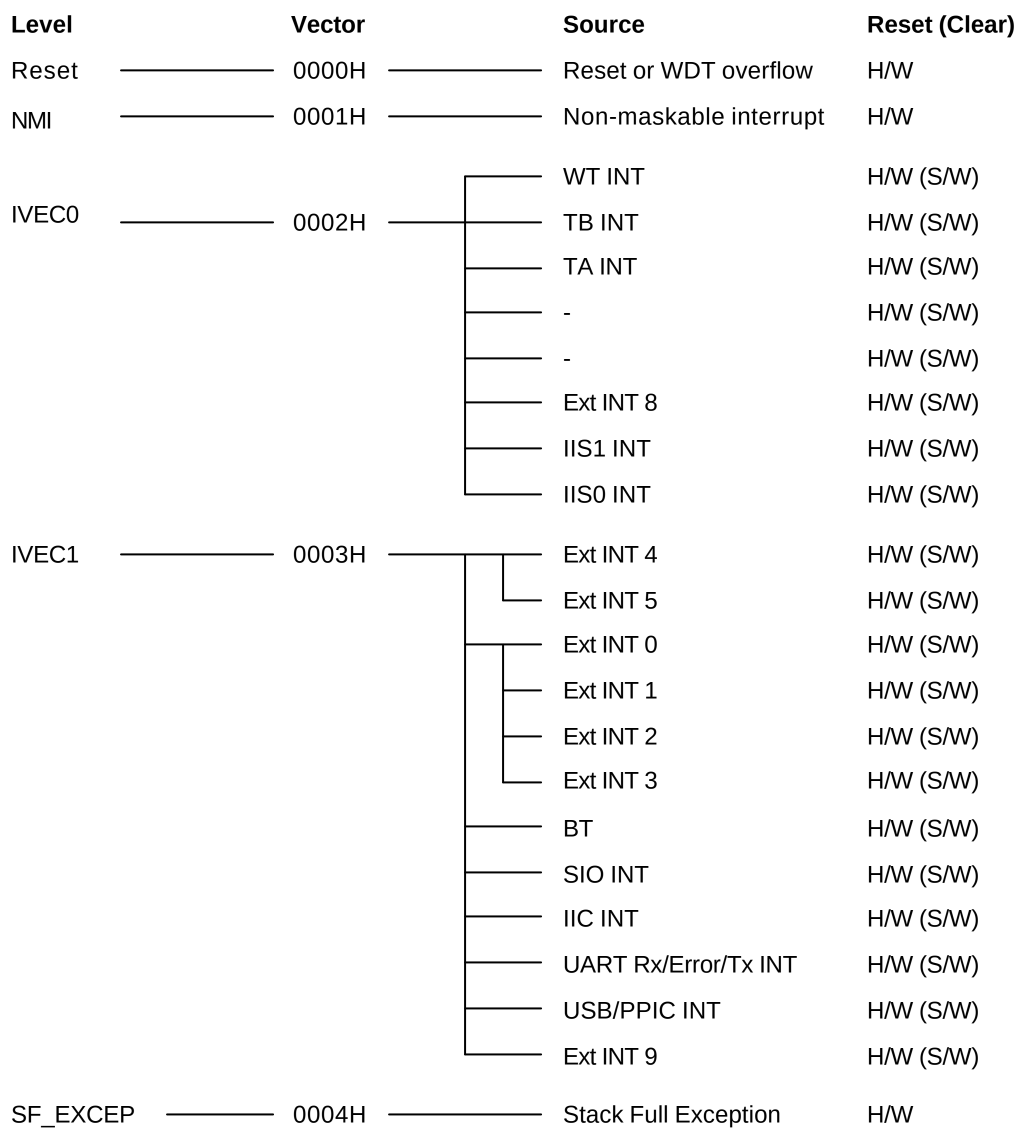
A Stack Full exception occurs when a stack operation is performed and as a result of the stack operation sptr[5] (SF) is set to 1. If the stack exception enable bit, exe (SR0[5]), is 1, the Stack Full exception is served. One exception to this rule is when nNMI causes a stack operation that sets sptr[5] (SF), since it has higher priority.

Handling a Stack Full exception may cause another Stack Full exception. In this case, the new exception is ignored. On a Stack Full exception, the CALL instruction is loaded in IR (Instruction Register) and 0h:0004h in PC. Therefore, when the Stack Full exception is activated, the "CALL {0h:PM[00004h]}" instruction is executed. When the exception is executed, SR0[1] (ie) is set to 0, and a core output signal *nEXPACK* is generated to acknowledge the exception.

BREAK EXCEPTION

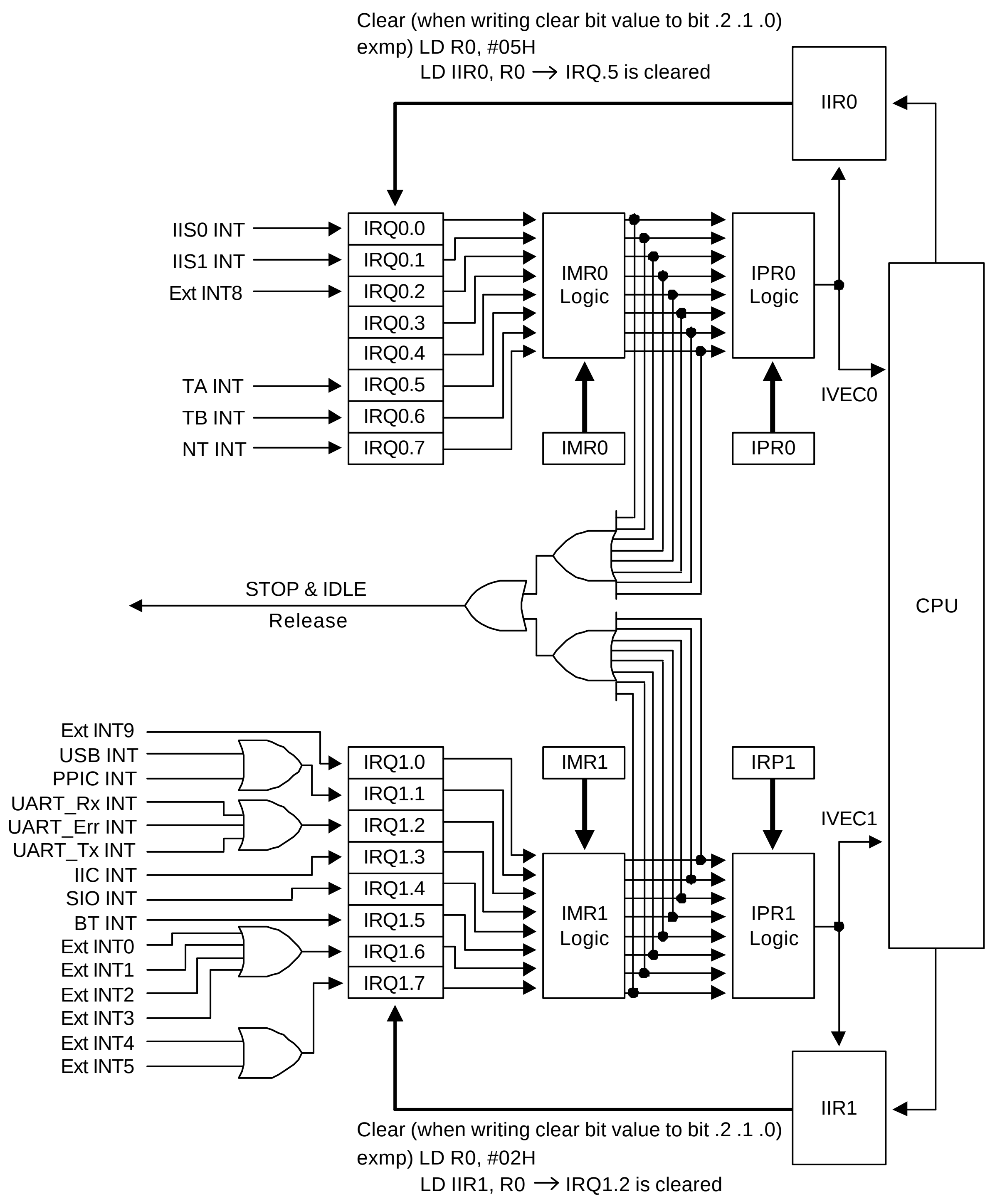
Break exception is reserved only for an in-circuit debugger. When a core input signal, *BKREQ*, is high, the CalmRISC core is halted or in the break mode, until *BKREQ* is deactivated. Another way to drive the CalmRISC core into the break mode is by executing a break instruction, BREAK. When BREAK is fetched, it is decoded in the fetch cycle (IF stage) and the CalmRISC core output signal *nBKACK* is generated in the second cycle (ID/MEM stage). An in-circuit debugger generates *BKREQ* active by monitoring *nBKACK* to be active. BREAK instruction is exactly the same as the NOP (no operation) instruction except that it does not increase the program counter and activates nBKACK in the second cycle (or ID/MEM stage of the pipeline). There, once BREAK is encountered in the program execution, it falls into a deadlock. BREAK instruction is reserved for in-circuit debuggers only, so it should not be used in user programs.

EXCEPTIONS (or INTERRUPTS)



NOTES:
1. NMI has the highest priority for an interrupt level, followed by SF_EXCEP, IVEC0 and IVEC1.
2. In the case of IVEC0 and IVEC1, one interrupt vector has several interrupt sources. The priority of the sources is controlled by setting the IPR register.
3. External interrupts are triggered by a rising or falling edge, depending on the corresponding control register setting, Ext INT0-Ext INT5 have no interrupt pending bit but have an enable bit.
4. After system reset, IIS0 INT has the highest priority in the IVEC0 level, followed by IIS1 INT and other interrupt sources.
5. The interrupt priority can be changed by setting of IPR register.
6. The pending bit is cleared by hardware when CPU reads the IIR register value.

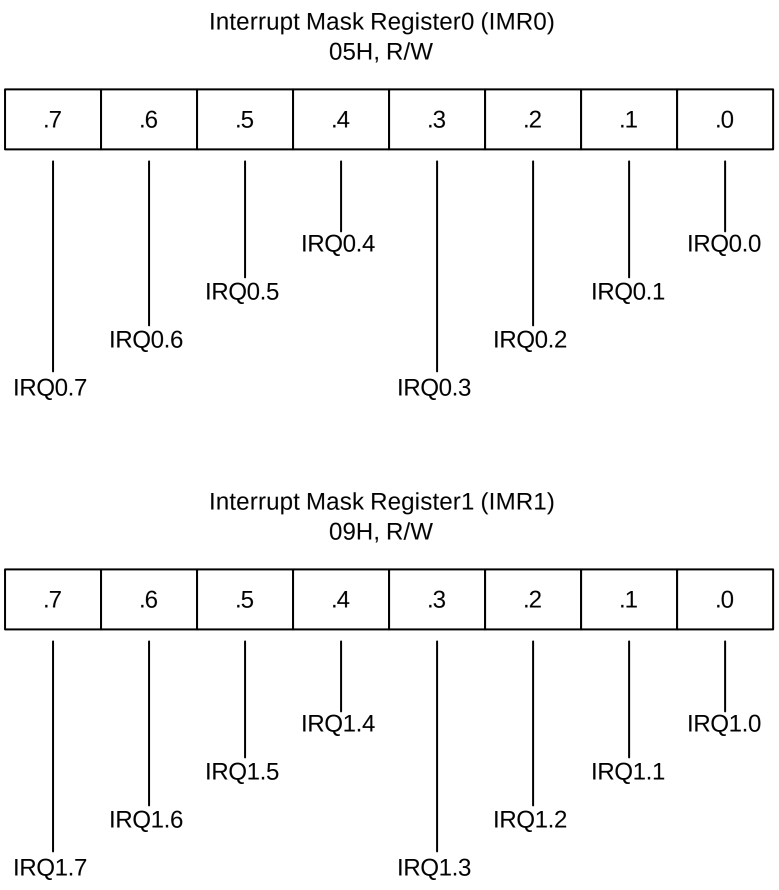
Figure 6-1. Interrupt Structure



NOTE: The IRQ register value is cleared by H/W when the IIR register is read by the programmer in an interrupt service routine. However, if you want to clear by S/W, then write the proper value to the IIR register like above examples. For clear all the bits of IRQx register at one time write "#08h" to the IIRx register.

Figure 6-2. Interrupt Structure

INTERRUPT MASK REGISTERS

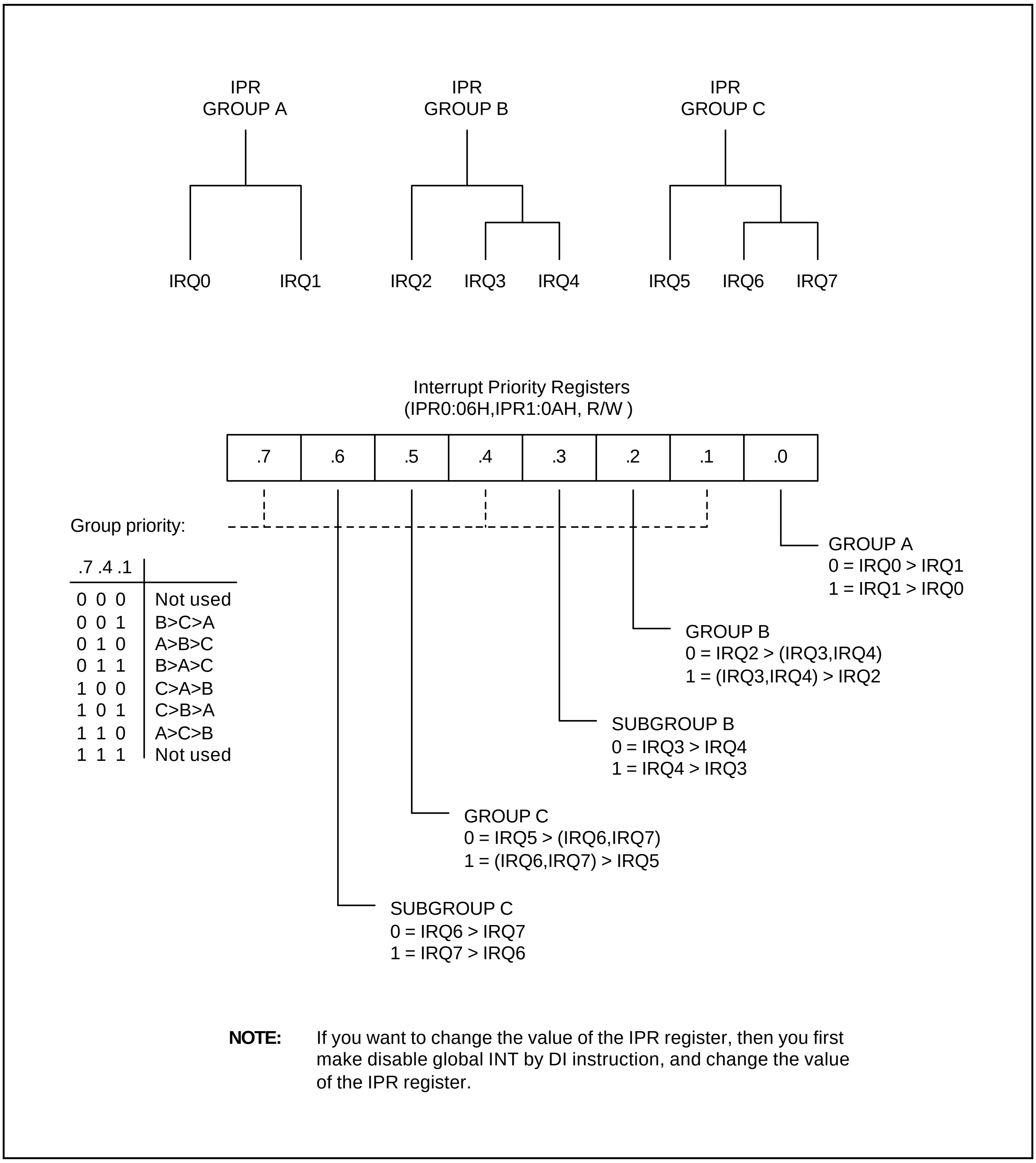


Interrupt request enable bits:
0 = Disable interrupt request
1 = Enable interrupt request

NOTE: If you want to change the value of the IMR register, then you first make disable global INT by DI instruction, and change the value of the IMR register.

Figure 6-3. Interrupt Mask Register

INTERRUPT PRIORITY REGISTER



.7 .4 .1	
0 0 0	Not used
0 0 1	B>C>A
0 1 0	A>B>C
0 1 1	B>A>C
1 0 0	C>A>B
1 0 1	C>B>A
1 1 0	A>C>B
1 1 1	Not used

NOTE: If you want to change the value of the IPR register, then you first make disable global INT by DI instruction, and change the value of the IPR register.

Figure 6-4. Interrupt Priority Register

☞ PROGRAMMING TIP — Interrupt Programming Tip 1

Jumped from vector 2

```
            PUSH        SR1
            PUSH        R0
            AND         SR0, #0FEh
            LD          R0, IIR0
            CP          R0, #03h
            JR          ULE, LTE03
            CP          R0, #05h
            JR          ULE, LTE05
            CP          R0, #06h
            JP          EQ, IRQ6_srv
            JP          IRQ7_srv
LTE05       CP          R0, #04h
            JP          EQ, IRQ4_srv
            JP          IRQ5_srv
LTE03       CP          R0, #01h
            JR          ULE, LTE01
            CP          R0, #02h
            JP          EQ, IRQ2_srv
            JP          IRQ3_srv
LTE01       CP          R0, #00h
            JP          EQ, IRQ0_srv
            JP          IRQ1_srv
IRQ0_srv    →    service for IRQ0
            •
            •
            POP         R0
            POP         SR1
            IRET
IRQ1_srv    →    service for IRQ1
            •
            •
            POP         R0
            POP         SR1
            IRET
            •
            •
IRQ7_srv    →    service for IRQ7
            •
            •
            POP         R0
            POP         SR1
            IRET
```

NOTE: If the SR0 register is changed in the interrupt service routine, then the SR0 register must be pushed and poped in the interrupt service routine.

☞ PROGRAMMING TIP — Interrupt Programming Tip 2

Jumped from vector 2

```
            PUSH      SR1
            PUSH      R0
            PUSH      R1
            LD        R0, IIR0
            SL        R0
            LD        R1, # < TBL_INTx
            ADD       R0, # > TBL_INTx
            PUSH      R0
            PUSH      R1
            RET
TBL_INTx    LJP       IRQ0_svr
            LJP       IRQ1_svr
            LJP       IRQ2_svr
            LJP       IRQ3_svr
            LJP       IRQ4_svr
            LJP       IRQ5_svr
            LJP       IRQ6_svr
            LJP       IRQ7_svr
IRQ0_srv    →  service for IRQ0
            •
            •
            POP       R1
            POP       R0
            POP       SR1
            IRET
IRQ1_srv    →  service for IRQ1
            •
            •
            POP       R1
            POP       R0
            POP       SR1
            IRET
            •
            •
IRQ7_srv    →  service for IRQ7
            •
            •
            POP       R1
            POP       R0
            POP       SR1
            IRET
```

NOTES:
1. If the SR0 register is changed in the interrupt service routine, then the SR0 register must be pushed and poped in the interrupt service routine.
2. Above example is assumed that the ROM size is less than 64Kword and all the LJP instructions which is in the jump table (TBL-INTx) is in the same page.

7 COPROCESSOR INTERFACE

OVERVIEW

CalmRISC supports an efficient and seamless interface with coprocessors. By integrating a MAC (multiply and accumulate) DSP coprocessor engine with the CalmRISC core, not only the microcontroller functions but also complex digital signal processing algorithms can be implemented in a single development platform (or MDS). CalmRISC has a set of dedicated signal pins, through which data/command/status are exchanged to and from a coprocessor. Figure 7-1 depicts the coprocessor signal pins and the interface between two processors.

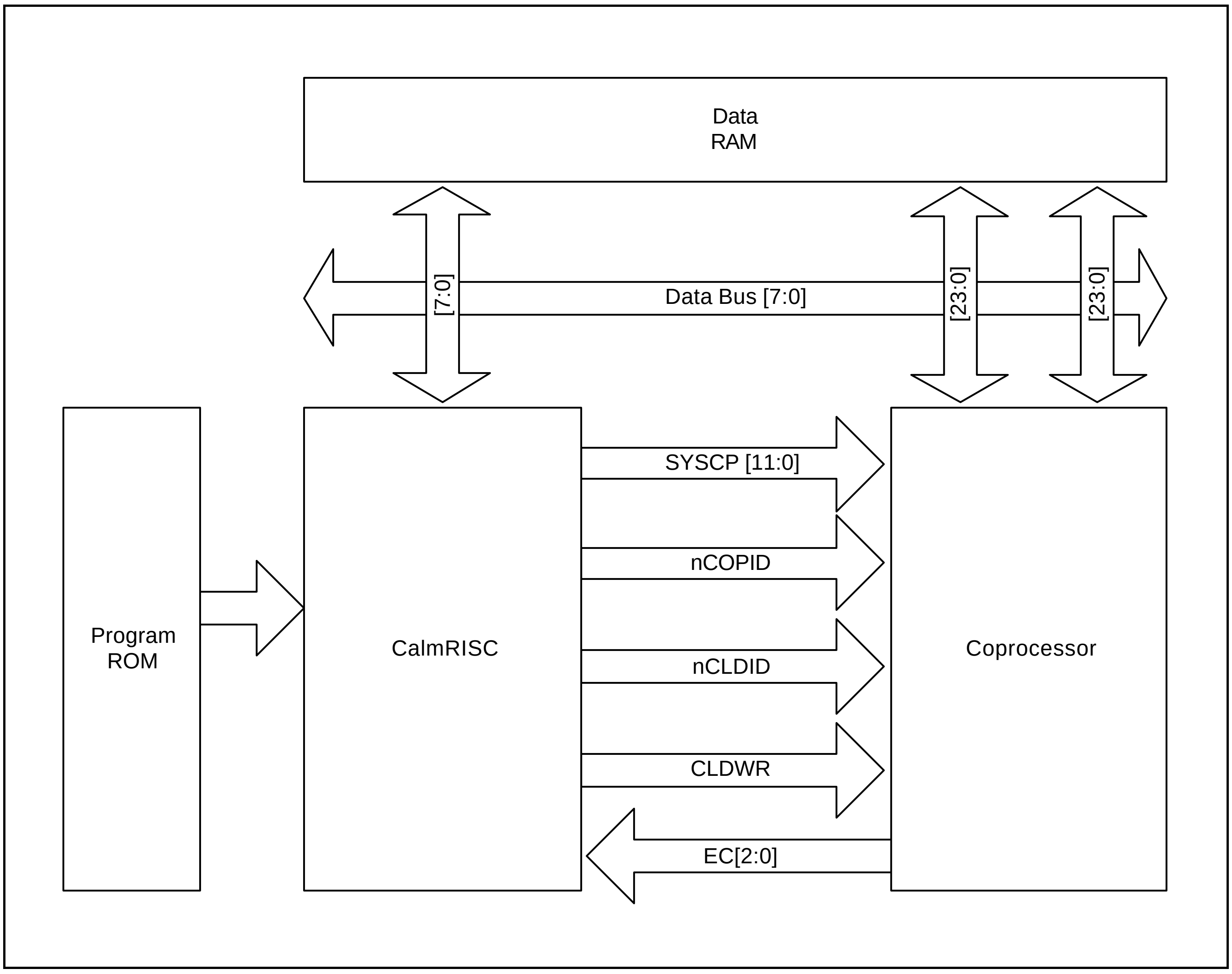


Figure 7-1. Coprocessor Interface Diagram

As shown in the coprocessor interface diagram above, the coprocessor interface signals of CalmRISC are: *SYSCP[11:0]*, *nCOPID*, *nCLDID*, *nCLDWR*, and *EC[2:0]*. The data are exchanged through data buses, *DI[7:0]* and *DO[7:0]*. A command is issued from CalmRISC to a coprocessor through *SYSCP[11:0]* in COP instructions. The status of a coprocessor can be sent back to CalmRISC through EC[2:0] and these flags can be checked in the condition codes of branch instructions. The coprocessor instructions are listed in the following table

Table 7-1. Coprocessor instructions

Mnemonic	Op 1	Op 2	Description
COP	#imm:12	–	Coprocessor operation
CLD	GPR	imm:8	Data transfer from coprocessor into GPR
CLD	imm:8	GPR	Data transfer of GPR to coprocessor
JP(or JR) CALL LNK	EC2–EC0	label	Conditional branch with coprocessor status flags

The coprocessor of CalmRISC does not have its own program memory (i.e., it is a passive coprocessor) as shown in Figure 7 -1. In fact, the coprocessor instructions are fetched and decoded by CalmRISC, and CalmRISC issues the command to the coprocessor through the interface signals. For example, if “COP #imm:12” instruction is fetched, then the 12-bit immediate value (imm:12) is loaded on *SYSCP[11:0]* signal with *nCOPID* active in ID/MEM stage, to request the coprocessor to perform the designated operation. The interpretation of the 12-bit immediate value is totally up to the coprocessor. By arranging the 12-bit immediate field, the instruction set of the coprocessor is determined. In other words, CalmRISC only provides a set of generic coprocessor instructions, and its installation to a specific coprocessor instruction set can differ from one coprocessor to another. CLD Write instructions (“CLD imm:8, GPR”) put the content of a GPR register of CalmRISC on the data bus (*DO[7:0]*) and issue the address(imm:8) of the coprocessor internal register on *SYSCP[7:0]* with *nCLDID* active and *CLDWR* active. CLD Read instructions (“CLD GPR, imm:8” in Table 7-1) work similarly, except that the content of the coprocessor internal register addressed by the 8-bit immediate value is read into a GPR register through *DI[7:0]* with *nCLDID* active and *CLDWR* deactivated.

The timing diagram given below is a coprocessor instruction pipeline and shows when the coprocessor performs the required operations. Suppose I_2 is a coprocessor instruction. First, it is fetched and decoded by CalmRISC (at t = T(i-1)). Once it is identified as a coprocessor instruction, CalmRISC indicates to the coprocessor the appropriate command through the coprocessor interface signals (at t = T(i)). Then the coprocessor performs the designated tasks at t = T(i) and t = T(i+1). Hence IF from CalmRISC and then ID/MEM and EX from the coprocessor constitute the pipeline for I_2. Similarly, if I_3 is a coprocessor instruction, the coprocessor’s ID/MEM and EX stages replace the corresponding stages of CalmRISC.

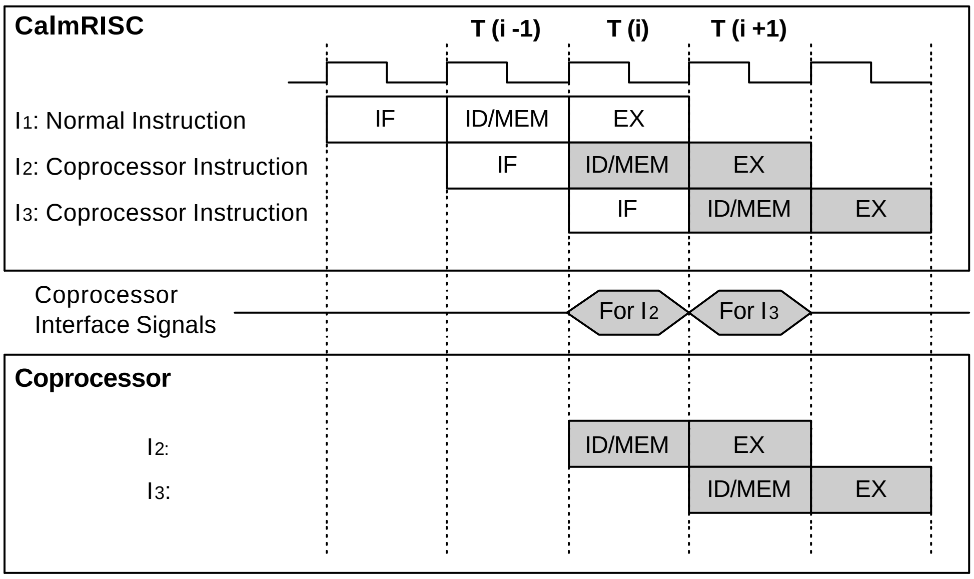


Figure 7-2. Coprocessor Instruction Pipeline

In a multi-processor system, the data transfer between processors is an important factor to determine the efficiency of the overall system. Suppose an input data stream is accepted by a processor, in order for the data to be shared by another processors. There should be some efficient mechanism to transfer the data to the processors. In CalmRISC, data transfers are accomplished through a single shared data memory. The shared data memory in a multi-processor has some inherent problems such as data hazards and deadlocks. However, the coprocessor in CalmRISC accesses the shared data memory only at the designated time by CalmRISC at which time CalmRISC is guaranteed not to access the data memory, and therefore there is no contention over the shared data memory. Another advantage of the scheme is that the coprocessor can access the data memory in its own bandwidth.

NOTES

8 INSTRUCTION SET

OVERVIEW

GLOSSARY

This chapter describes the CalmRISC instruction set and the details of each instruction are listed in alphabetical order. The following notations are used for the description.

Table 8-1. Instruction Notation Conventions

Notation	Interpretation
<opN>	Operand N. N can be omitted if there is only one operand. Typically, <op1> is the destination (and source) operand and <op2> is a source operand.
GPR	General Purpose Register
SPR	Special Purpose Register (IDL0, IDL1, IDH, SR0, ILX, ILH, ILL, SR1)
adr:N	N-bit address specifier
@idm	Content of memory location pointed by ID0 or ID1
(adr:N)	Content of memory location specified by adr:N
cc:4	4-bit condition code. Table 8-6 describes cc:4.
imm:N	N-bit immediate number
&	Bit-wise AND
|	Bit-wise OR
~	Bit-wise NOT
^	Bit-wise XOR
N**M	Mth power of N
$(N)_M$	M-based number N

As additional note, only the affected flags are described in the tables in this section. That is, if a flag is not affected by an operation, it is NOT specified.

INSTRUCTION SET MAP

Table 8-2. Overall Instruction Set Map

IR	[12:10]000	001	010	011	100	101	110	111
[15:13,7:2] 000 xxxxxx	ADD GPR, #imm:8	SUB GPR, #imm:8	CP GPR, #imm8	LD GPR, #imm:8	TM GPR, #imm:8	AND GPR, #imm:8	OR GPR, #imm:8	XOR GPR, #imm:8
001 xxxxxx	ADD GPR, @idm	SUB GPR, @idm	CP GPR, @idm	LD GPR, @idm	LD @idm, GPR	AND GPR, @idm	OR GPR, @idm	XOR GPR, @idm
010 xxxxxx	ADD GPR, adr:8	SUB GPR, adr:8	CP GPR, adr:8	LD GPR, adr:8	BITT adr:8.bs		BITS adr:8.bs	
011 xxxxxx	ADC GPR, adr:8	SBC GPR, adr:8	CPC GPR, adr:8	LD adr:8, GPR	BITR adr:8.bs		BITC adr:8.bs	
100 000000	ADD GPR, GPR	SUB GPR, GPR	CP GPR, GPR	BMS/BMC	LD SPR0, #imm:8	AND GPR, adr:8	OR GPR, adr:8	XOR GPR, adr:8
100 000001	ADC GPR, GPR	SBC GPR, GPR	CPC GPR, GPR	*invalid*				
100 000010	*invalid*	*invalid*	*invalid*	*invalid*				
100 000011	AND GPR, GPR	OR GPR, GPR	XOR GPR, GPR	*invalid*				
100 00010x	SLA/SL/ RLC/RL/ SRA/SR/ RRC/RR/ GPR	INC/INCC/ DEC/ DECC/ COM/ COM2/ COMC GPR	*invalid*	*invalid*				
100 00011x	LD SPR, GPR	LD GPR, SPR	SWAP GPR, SPR	LD TBH/TBL, GPR				
100 00100x	PUSH SPR	POP SPR	*invalid*	*invalid*				
100 001010	PUSH GPR	POP GPR	LD GPR, GPR	LD GPR, TBH/TBL				

Table 8-2. Overall Instruction Set Map (Continued)

IR	[12:10]000	001	010	011	100	101	110	111
100 001011	POP	*invalid*	LDC	*invalid*	LD SPR0, #imm:8	AND GPR, adr:8	OR GPR, adr:8	XOR GPR, adr:8
100 00110x	RET/LRET/IRET/NOP/ BREAK	*invalid*	*invalid*	*invalid*				
100 00111x	*invalid*	*invalid*	*invalid*	*invalid*				
100 01xxxx	LD GPR:bank, GPR:bank	AND SR0, #imm:8	OR SR0, #imm:8	BANK #imm:2				
100 100000 100 110011	*invalid*	*invalid*	*invalid*	*invalid*				
100 1101xx	LCALL cc:4, imm:20 (2-word instruction)							
100 1110xx	LLNK cc:4, imm:20 (2-word instruction)							
100 1111xx	LJP cc:4, imm:20 (2-word instruction)							
[15:10] 101 xxx	JR cc:4, imm:9							
110 0xx	CALLS imm:12							
110 1xx	LNKS imm:12							
111 xxx	CLD GPR, imm:8 / CLD imm:8, GPR / JNZD GPR, imm:8 / SYS #imm:8 / COP #imm:12							

NOTE: "*invalid*" - invalid instruction.

Table 8-3. Instruction Encoding

Instruction	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD GPR, #imm:8	000			000			GPR		imm[7:0]							
SUB GPR, #imm:8				001												
CP GPR, #imm:8				010												
LD GPR, #imm:8				011												
TM GPR, #imm:8				100												
AND GPR, #imm:8				101												
OR GPR, #imm:8				110												
XOR GPR, #imm:8				111												
ADD GPR, @idm	001			000			GPR		idx	mod		offset[4:0]				
SUB GPR, @idm				001												
CP GPR, @idm				010												
LD GPR, @idm				011												
LD @idm, GPR				100												
AND GPR, @idm				101												
OR GPR, @idm				110												
XOR GPR, @idm				111												
ADD GPR, adr:8	010			000			GPR		adr[7:0]							
SUB GPR, adr:8				001												
CP GPR, adr:8				010												
LD GPR, adr:8				011												
BITT adr:8.bs				10		bs										
BITS adr:8.bs				11												
ADC GPR, adr:8	011			000			GPR		adr[7:0]							
SBC GPR, adr:8				001												
CPC GPR, adr:8				010												
LD adr:8, GPR				011												
BITR adr:8.bs				10		bs										
BITC adr:8.bs				11												

Table 8-3. Instruction Encoding (Continued)

Instruction	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD GPRd, GPRs	100			000			GPRd		000000						GPRs	
SUB GPRd, GPRs				001												
CP GPRd, GPRs				010												
BMS/BMC				011												
ADC GPRd, GPRs				000					000001							
SBC GPRd, GPRs				001												
CPC GPRd, GPRs				010												
invalid				011												
invalid				ddd					000010							
AND GPRd, GPRs				000					000011							
OR GPRd, GPRs				001												
XOR GPRd, GPRs				010												
invalid				011												
ALUop1				000			GPR		00010					ALUop1		
ALUop2				001			GPR							ALUop2		
invalid				010–011			xx							xxx		
LD SPR, GPR				000			GPR		00011					SPR		
LD GPR, SPR				001			GPR							SPR		
SWAP GPR, SPR				010			GPR							SPR		
LD TBL, GPR				011			GPR							x	0	x
LD TBH, GPR														x	1	x
PUSH SPR				000			xx		00100					SPR		
POP SPR				001			xx							SPR		
invalid				010–011			xx							xxx		
PUSH GPR				000			GPR		001010						GPR	
POP GPR				001			GPR								GPR	
LD GPRd, GPRs				010			GPRd								GPRs	
LD GPR, TBL				011			GPR								0	x
LD GPR, TBH															1	x
POP				000			xx		001011						xx	
LDC @IL				010											0	x
LDC @IL+															1	x
Invalid				001, 011											xx	

NOTE: "x" means not applicable.

Table 8-3. Instruction Encoding (Concluded)

Instruction	15-13	12	11	10	9	8	7	6	5	4	3	2	1	0	2nd word
MODop1	100	000			xx		00110					MODop1			–
Invalid		001–011			xx							xxx			
Invalid		000			xx		01		xxxxxx						
AND SR0, #imm:8		001			imm[7:6]				imm[5:0]						
OR SR0, #imm:8		010			imm[7:6]										
BANK #imm:2		011			xx				x	imm [1:0]		xxx			
Invalid		0	xxxx				10000000-11001111								
LCALL cc, imm:20			cc				1101				imm[19:16]				imm[15:0]
LLNK cc, imm:20															
LJP cc, imm:20															
LD SPR0, #imm:8		1	00		SPR0		IMM[7:0]								–
AND GPR, adr:8			01		GPR		ADR[7:0]								
OR GPR, adr:8			10												
XOR GPR, adr:8			11												
JR cc, imm:9	101	imm [8]	cc				imm[7:0]								
CALLS imm:12	110	0	imm[11:0]												
LNKS imm:12		1													
CLD GPR, imm:8	111	0	00		GPR		imm[7:0]								
CLD imm:8, GPR			01		GPR										
JNZD GPR, imm:8			10		GPR										
SYS #imm:8			11		xx										
COP #imm:12		1	imm[11:0]												

NOTES:
1. "x" means not applicable.
2. There are several MODop1 codes that can be used, as described in table 8-9.
3. The operand 1(GPR) of the instruction JNZD is Bank 3's register.

Table 8-4. Index Code Information (“idx”)

Symbol	Code	Description
ID0	0	Index 0 IDH:IDL0
ID1	1	Index 1 IDH:IDL1

Table 8-5. Index Modification Code Information (“mod”)

Symbol	Code	Function
@IDx + offset:5	00	DM[IDx], IDx ← IDx + offset
@[IDx - offset:5]	01	DM[IDx + (2’s complement of offset:5)], IDx ← IDx + (2’s complement of offset:5)
@[IDx + offset:5]!	10	DM[IDx + offset], IDx ← IDx
@[IDx - offset:5]!	11	DM[IDx + (2’s complement of offset:5)], IDx ← IDx

NOTE: Carry from IDL is propagated to IDH. In case of @[IDx - offset:5] or @[IDx - offset:5]!, the assembler should convert offset:5 to the 2’s complement format to fill the operand field (offset[4:0]).
Furthermore, @[IDx - 0] and @[IDx - 0]! are converted to @[IDx + 0] and @[IDx + 0]!, respectively.

Table 8-6. Condition Code Information (“cc”)

Symbol (cc:4)	Code	Function
Blank	0000	always
NC or ULT	0001	C = 0, unsigned less than
C or UGE	0010	C = 1, unsigned greater than or equal to
Z or EQ	0011	Z = 1, equal to
NZ or NE	0100	Z = 0, not equal to
OV	0101	V = 1, overflow - signed value
ULE	0110	~C | Z, unsigned less than or equal to
UGT	0111	C & ~Z, unsigned greater than
ZP	1000	N = 0, signed zero or positive
MI	1001	N = 1, signed negative
PL	1010	~N & ~Z, signed positive
ZN	1011	Z | N, signed zero or negative
SF	1100	Stack Full
EC0-EC2	1101-1111	EC[0] = 1/EC[1] = 1/EC[2] = 1

NOTE: EC[2:0] is an external input (CalmRISC core’s point of view) and used as a condition.

Table 8-7. “ALUop1” Code Information

Symbol	Code	Function
SLA	000	arithmetic shift left
SL	001	shift left
RLC	010	rotate left with carry
RL	011	rotate left
SRA	100	arithmetic shift right
SR	101	shift right
RRC	110	rotate right with carry
RR	111	rotate right

Table 8-8. “ALUop2” Code Information

Symbol	Code	Function
INC	000	increment
INCC	001	increment with carry
DEC	010	decrement
DECC	011	decrement with carry
COM	100	1’s complement
COM2	101	2’s complement
COMC	110	1’s complement with carry
–	111	reserved

Table 8-9. “MODop1” Code Information

Symbol	Code	Function
LRET	000	return by IL
RET	001	return by HS
IRET	010	return from interrupt (by HS)
NOP	011	no operation
BREAK	100	reserved for debugger use only
–	101	reserved
–	110	reserved
–	111	reserved

QUICK REFERENCE

Operation	op1	op2	Function	Flag	# of word / cycle
AND OR XOR ADD SUB CP	GPR	adr:8 #imm:8 GPR @idm	op1 ← op1 & op2 op1 ← op1 | op2 op1 ← op1 ^ op2 op1 ← op1 + op2 op1 ← op1 + ~op2 + 1 op1 + ~op2 + 1	z,n z,n z,n c,z,v,n c,z,v,n c,z,v,n	1W1C
ADC SBC CPC	GPR	GPR adr:8	op1 ← op1 + op2 + c op1 ← op1 + ~op2 + c op1 + ~op2 + c	c,z,v,n c,z,v,n c,z,v,n	
TM	GPR	#imm:8	op1 & op2	z,n	
BITS BITR BITC BITT	R3	adr:8.bs	op1 ← (op2[bit] ← 1) op1 ← (op2[bit] ← 0) op1 ← ~(op2[bit]) z ← ~(op2[bit])	z z z z	
BMS/BMC	–	–	TF ← 1 / 0	–	
PUSH POP	GPR	–	HS[sptr] ← GPR, (sptr ← sptr + 1) GPR ← HS[sptr - 1], (sptr ← sptr - 1)	– z,n	
PUSH POP	SPR	–	HS[sptr] ← SPR, (sptr ← sptr + 1) SPR ← HS[sptr - 1], (sptr ← sptr - 1)	–	
POP	–	–	sptr ← sptr – 2	–	

SLA
SL
RLC
RL
SRA
SR
RRC
RR
INC
INCC
DEC
DECC
COM
COM2
COMC
	GPR	–	c ← op1[7], op1 ← {op1[6:0], 0}
			c ← op1[7], op1 ← {op1[6:0], 0}
			c ← op1[7], op1 ← {op1[6:0], c}
			c ← op[7], op1 ← {op1[6:0], op1[7]}
			c ← op[0], op1 ← {op1[7],op1[7:1]}
			c ← op1[0], op1 ← {0, op1[7:1]}
			c ← op1[0], op1 ← {c, op1[7:1]}
			c ← op1[0], op1 ← {op1[0], op1[7:1]}
			p1 ← op1 + 1
			op1 ← op1 + c
			op1 ← op1 + 0FFh
			op1 ← op1 + 0FFh + c
			op1 ← ~op1
			op1 ← ~op1 + 1
			op1 ← ~op1 + c
				c,z,v,n
				c,z,n
				c,z,n
				c,z,n
				c,z,n
				c,z,n
				c,z,n
				c,z,n
				c,z,v,n
				c,z,v,n
				c,z,v,n
				c,z,v,n
				z,n
				c,z,v,n
				c,z,v,n	

QUICK REFERENCE (Continued)

Operation	op1	op2	Function	Flag	# of word / cycle
LD	GPR :bank	GPR :bank	op1 ← op2	z,n	1W1C
LD	SPR0	#imm:8	op1 ← op2	–	
LD	GPR	GPR SPR adr:8 @idm #imm:8 TBH/TBL	op1 ← op2	z,n	
LD	SPR TBH/TBL	GPR	op1 ← op2	–	
LD	adr:8	GPR	op1 ← op2	–	
LD	@idm	GPR	op1 ← op2	–	
LDC	@IL @IL+	–	(TBH:TBL) ← PM[(ILX:ILH:ILL)], ILL++ if @IL+	–	1W2C
AND OR	SR0	#imm:8	SR0 ← SR0 & op2 SR0 ← SR0 | op2	–	1W1C
BANK	#imm:2	–	SR0[4:3] ← op2	–	
SWAP	GPR	SPR	op1 ← op2, op2 ← op1 (excluding SR0/SR1)	–	
LCALL cc	imm:20	–	If branch taken, push XSTACK, HS[15:0] ← {PC[15:12],PC[11:0] + 2} and PC ← op1 else PC[11:0] ← PC[11:0] + 2	–	2W2C
LLNK cc	imm:20	–	If branch taken, IL[19:0] ← {PC[19:12], PC[11:0] + 2} and PC ← op1 else PC[11:0] ← PC[11:0] + 2	–	
CALLS	imm:12	–	push XSTACK, HS[15:0] ← {PC[15:12], PC[11:0] + 1} and PC[11:0] ← op1	–	1W2C
LNKS	imm:12	–	IL[19:0] ← {PC[19:12], PC[11:0] + 1} and PC[11:0] ← op1	–	

JNZD	Rn	imm:8	if (Rn == 0) PC ← PC[delay slot] - 2's
			complement of imm:8, Rn--
			else PC ← PC[delay slot]++, Rn--
				–	

LJP cc	imm:20	–	If branch taken, PC ← op1 else PC[11:0] < PC[11:0] + 2	–	2W2C
JR cc	imm:9	–	If branch taken, PC[11:0] ← PC[11:0] + op1 else PC[11:0] ← PC[11:0] + 1	–	1W2C

NOTE: op1 - operand1, op2 - operand2, 1W1C - 1-Word 1-Cycle instruction, 1W2C - 1-Word 2-Cycle instruction, 2W2C - 2-Word 2-Cycle instruction. The Rn of instruction JNZD is Bank 3's GPR.

QUICK REFERENCE (Concluded)

Operation	op1	op2	Function	Flag	# of word / cycle
LRET RET IRET NOP BREAK	–	–	PC ← IL[19:0] PC ← HS[sptr - 2], (sptr ← sptr - 2) PC ← HS[sptr - 2], (sptr ← sptr - 2) no operation no operation and hold PC	–	1W2C 1W2C 1W2C 1W1C 1W1C
SYS	#imm:8	–	no operation but generates SYSCP[7:0] and nSYSID	–	1W1C
CLD	imm:8	GPR	op1 ← op2, generates SYSCP[7:0], nCLDID, and CLDWR	–	
CLD	GPR	imm:8	op1 ← op2, generates SYSCP[7:0], nCLDID, and CLDWR	z,n	
COP	#imm:12	–	generates SYSCP[11:0] and nCOPID	–	

NOTES:

1. op1 - operand1, op2 - operand2, sptr - stack pointer register, 1W1C - 1-Word 1-Cycle instruction, 1W2C - 1-Word 2-Cycle instruction
2. Pseudo instructions
   - SCF/RCF
     Carry flag set or reset instruction
   - STOP/IDLE
     MCU power saving instructions
   - EI/DI
     Exception enable and disable instructions
   - JP/LNK/CALL
     If JR/LNKS/CALLS commands (1 word instructions) can access the target address, there is no conditional code in the case of CALL/LNK, and the JP/LNK/CALL commands are assembled to JR/LNKS/CALLS in linking time, or else the JP/LNK/CALL commands are assembled to LJP/LLNK/LCALL (2 word instructions) instructions.

INSTRUCTION GROUP SUMMARY

ALU INSTRUCTIONS

“ALU instructions” refer to the operations that use ALU to generate results. ALU instructions update the values in Status Register 1 (SR1), namely carry (C), zero (Z), overflow (V), and negative (N), depending on the operation type and the result.

ALUop GPR, adr:8

Performs an ALU operation on the value in GPR and the value in DM[adr:8] and stores the result into GPR.
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPR+(not DM[adr:8])+1 is performed.
adr:8 is the offset in a specific data memory page.

The data memory page is 0 or the value of IDH (Index of Data Memory Higher Byte Register), depending on the value of eid in Status Register 0 (SR0).

Operation

GPR ← GPR ALUop DM[00h:adr:8] if eid = 0
GPR ← GPR ALUop DM[IDH:adr8] if eid = 1
Note that this is an 8-bit operation.

Example

```
ADD R0, 80h             // Assume eid = 1 and IDH = 01H
                        // R0 ← R0 + DM[0180h]
```

ALUop GPR, #imm:8

Stores the result of an ALU operation on GPR and an 8-bit immediate value into GPR.
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPR+(not #imm:8)+1 is performed.
#imm:8 is an 8-bit immediate value.

Operation

GPR ← GPR ALUop #imm:8

Example

```
ADD R0, #7Ah            // R0 ← R0 + 7Ah
```

ALUop GPRd, GPRs

Store the result of ALUop on GPRs and GPRd into GPRd.
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPRd + (not GPRs) + 1 is performed.
GPRs and GPRd need not be distinct.

Operation

GPRd ← GPRd ALUop GPRs
GPRd - GPRs when ALUop = CP (comparison only)

Example

```
ADD R0, R1              // R0 ← R0 + R1
```

ALUop GPR, @idm

Performs ALUop on the value in GPR and DM[ID] and stores the result into GPR. Index register ID is IDH:IDL (IDH:IDL0 or IDH:IDL1).
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPR+(not DM[idm])+1 is performed.
idm = IDx+off:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]!
(IDx = ID0 or ID1)

Operation

GPR - DM[idm] when ALUop = CP (comparison only)
GPR ← GPR ALUop DM[IDx], IDx ← IDx + offset:5 when idm = IDx + offset:5
GPR ← GPR ALUop DM[IDx - offset:5], IDx ← IDx - offset:5 when idm = [IDx - offset:5]
GPR ← GPR ALUop DM[IDx + offset:5] when idm = [IDx + offset:5]!
GPR ← GPR ALUop DM[IDx - offset:5] when idm = [IDx - offset:5]!

When carry is generated from IDL (on a post-increment or pre-decrement), it is propagated to IDH.

Example

```
ADD R0, @ID0+2          // assume ID0 = 02FFh
                        // R0 ← R0 + DM[02FFh], IDH ← 03h and IDL0 ← 01h
ADD R0, @[ID0-2]        // assume ID0 = 0201h
                        // R0 ← R0 + DM[01FFh], IDH ← 01h and IDL0 ← FFh
ADD R0, @[ID1+2]!       // assume ID1 = 02FFh
                        // R0 ← R0 + DM[0301], IDH ← 02h and IDL1 ← FFh
ADD R0, @[ID1-2]!       // assume ID1 = 0200h
                        // R0 ← R0 + DM[01FEh], IDH ← 02h and IDL1 ← 00h
```

ALUopc GPRd, GPRs

Performs ALUop with carry on GPRd and GPRs and stores the result into GPRd.
ALUopc = ADC, SBC, CPC
GPRd and GPRs need not be distinct.

Operation

GPRd ← GPRd + GPRs + C when ALUopc = ADC
GPRd ← GPRd + (not GPRs) + C when ALUopc = SBC
GPRd + (not GPRs) + C when ALUopc = CPC (comparison only)

Example

```
ADD R0, R2          // assume R1:R0 and R3:R2 are 16-bit signed or unsigned numbers.
ADC R1, R3          // to add two 16-bit numbers, use ADD and ADC.

SUB R0, R2          // assume R1:R0 and R3:R2 are 16-bit signed or unsigned numbers.
SBC R1, R3          // to subtract two 16-bit numbers, use SUB and SBC.

CP R0, R2           // assume both R1:R0 and R3:R2 are 16-bit unsigned numbers.
CPC R1, R3          // to compare two 16-bit unsigned numbers, use CP and CPC.
```

ALUopc GPR, adr:8

Performs ALUop with carry on GPR and DM[adr:8].

Operation

GPR ← GPR + DM[adr:8] + C when ALUopc = ADC
GPR ← GPR + (not DM[adr:8]) + C when ALUopc = SBC
GPR + (not DM[adr:8]) + C when ALUopc = CPC (comparison only)

CPLop GPR (Complement Operations)

CPLop = COM, COM2, COMC

Operation

COM GPR	not GPR (logical complement)
COM2 GPR	not GPR + 1 (2's complement of GPR)
COMC GPR	not GPR + C (logical complement of GPR with carry)

Example

```
COM2 R0             // assume R1:R0 is a 16-bit signed number.
COMC R1             // COM2 and COMC can be used to get the 2's complement of it.
```

IncDec GPR (Increment/Decrement Operations)

IncDec = INC, INCC, DEC, DECC

Operation

INC GPR	Increase GPR, i.e., GPR ← GPR + 1
INCC GPR	Increase GPR if carry = 1, i.e., GPR ← GPR + C
DEC GPR	Decrease GPR, i.e., GPR ← GPR + FFh
DECC GPR	Decrease GPR if carry = 0, i.e., GPR ← GPR + FFh + C

Example

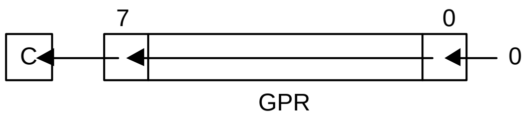
```
INC R0          // assume R1:R0 is a 16-bit number
INCC R1         // to increase R1:R0, use INC and INCC.

DEC R0          // assume R1:R0 is a 16-bit number
DECC R1         // to decrease R1:R0, use DEC and DECC.
```

SHIFT/ROTATE INSTRUCTIONS

Shift (Rotate) instructions shift (rotate) the given operand by 1 bit. Depending on the operation performed, a number of Status Register 1 (SR1) bits, namely Carry (C), Zero (Z), Overflow (V), and Negative (N), are set.

SL GPR

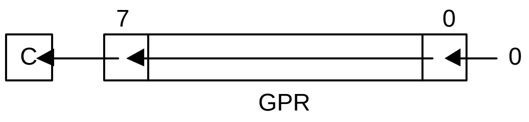
Operation



Carry (C) is the MSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting.
Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

SLA GPR

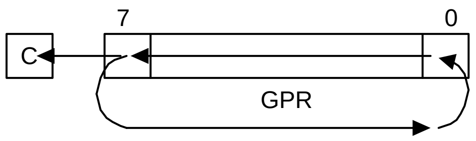
Operation



Carry (C) is the MSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting.
Overflow (V) will be 1 if the MSB of the result is different from C. Z will be 1 if the result is 0.

RL GPR

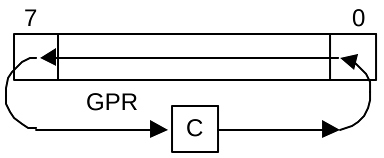
Operation



Carry (C) is the MSB of GPR before rotating. Negative (N) is the MSB of GPR after rotatin/g.
Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

RLC GPR

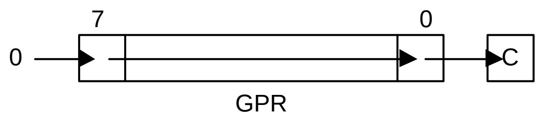
Operation



Carry (C) is the MSB of GPR before rotating, Negative (N) is the MSB of GPR after rotating.
Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

SR GPR

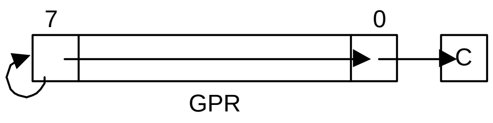
Operation



Carry (C) is the LSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting.
Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

SRA GPR

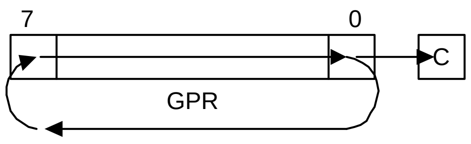
Operation



Carry (C) is the LSB of GPR before shifting, Negative (N) is the MSB of GPR after shifting.
Overflow (V) is not affected. Z will be 1 if the result is 0.

RR GPR

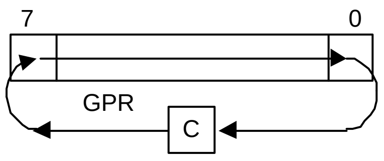
Operation



Carry (C) is the LSB of GPR before rotating. Negative (N) is the MSB of GPR after rotating.
Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

RRC GPR

Operation



Carry (C) is the LSB of GPR before rotating, Negative (N) is the MSB of GPR after rotating.
Overflow (V) is not affected. Zero (Z) will be 1 if the result is 0.

LOAD INSTRUCTIONS

Load instructions transfer data from data memory to a register or from a register to data memory, or assigns an immediate value into a register. As a side effect, a load instruction placing a value into a register sets the Zero (Z) and Negative (N) bits in Status Register 1 (SR1), if the placed data is 00h and the MSB of the data is 1, respectively.

LD GPR, adr:8

Loads the value of DM[adr:8] into GPR. Adr:8 is offset in the page specified by the value of eid in Status Register 0 (SR0).

Operation

```
GPR ← DM[00h:adr:8]   if eid = 0
GPR ← DM[IDH:adr:8]   if eid = 1
```

Note that this is an 8-bit operation.

Example

```
LD R0, 80h               // assume eid = 1 and IDH= 01H
                         // R0 ← DM[0180h]
```

LD GPR, @idm

Loads a value from the data memory location specified by @idm into GPR.
idm = IDx+off:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]!
(IDx = ID0 or ID1)

Operation

```
GPR ← DM[IDx], IDx ← IDx + offset:5 when idm = IDx + offset:5
GPR ← DM[IDx - offset:5], IDx ← IDx - offset:5 when idm = [IDx - offset:5]
GPR ← DM[IDx + offset:5] when idm = [IDx + offset:5]!
GPR ← DM[IDx - offset:5] when idm = [IDx - offset:5]!
```

When carry is generated from IDL (on a post-increment or pre-decrement), it is propagated to IDH.

Example

```
LD R0, @[ID0 + 03h]!     // assume IDH:IDL0 = 0270h
                         // R0 ← DM[0273h], IDH:IDL0 ← 0270h
```

LD REG, #imm:8

Loads an 8-bit immediate value into REG. REG can be either GPR or an SPR0 group register - IDH (Index of Data Memory Higher Byte Register), IDL0 (Index of Data Memory Lower Byte Register)/ IDL1, and Status Register 0 (SR0). #imm:8 is an 8-bit immediate value.

Operation

```
REG ← #imm:8
```

Example

```
LD R0 #7Ah        // R0 ← 7Ah
LD IDH, #03h                // IDH ← 03h
```

LD GPR:bs:2, GPR:bs:2

Loads a value of a register from a specified bank into another register in a specified bank.

Example

```
LD R0:1, R2:3               // R0 in bank 1, R2 in bank 3
```

LD GPR, TBH/TBL

Loads the value of TBH or TBL into GPR. TBH and TBL are 8-bit long registers used exclusively for LDC instructions that access program memory. Therefore, after an LDC instruction, LD GPR, TBH/TBL instruction will usually move the data into GPRs, to be used for other operations.

Operation

```
GPR ← TBH (or TBL)
```

Example

```
LDC @IL                     // gets a program memory item residing @ ILX:ILH:ILL
LD R0, TBH
LD R1, TBL
```

LD TBH/TBL, GPR

Loads the value of GPR into TBH or TBL. These instructions are used in pair in interrupt service routines to save and restore the values in TBH/TBL as needed.

Operation

```
TBH (or TBL) ← GPR
```

LD GPR, SPR

Loads the value of SPR into GPR.

Operation

```
GPR ← SPR
```

Example

```
LD R0, IDH                  // R0 ← IDH
```

LD SPR, GPR

Loads the value of GPR into SPR.

Operation

SPR ← GPR

Example

```
LD IDH, R0                // IDH ← R0
```

LD adr:8, GPR

Stores the value of GPR into data memory (DM). adr:8 is offset in the page specified by the value of eid in Status Register 0 (SR0).

Operation

DM[00h:adr:8] ← GPR if eid = 0
DM[IDH:adr:8] ← GPR if eid = 1

Note that this is an 8-bit operation.

Example

```
LD 7Ah, R0                // assume eid = 1 and IDH = 02h.
                          // DM[027Ah] ← R0
```

LD @idm, GPR

Loads a value into the data memory location specified by @idm from GPR.
idm = IDx+off:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]!
(IDx = ID0 or ID1)

Operation

DM[IDx] ← GPR, IDx ← IDx + offset:5 when idm = IDx + offset:5
DM[IDx - offset:5] ← GPR, IDx ← IDx - offset:5 when idm = [IDx - offset:5]
DM[IDx + offset:5] ← GPR when idm = [IDx + offset:5]!
DM[IDx - offset:5] ← GPR when idm = [IDx - offset:5]!

When carry is generated from IDL (on a post-increment or pre-decrement), it is propagated to IDH.

Example

```
LD @[ID0 + 03h]!, R0      // assume IDH:IDL0 = 0170h
                          // DM[0173h] ← R0, IDH:IDL0 ← 0170h
```

BRANCH INSTRUCTIONS

Branch instructions can be categorized into jump instruction, link instruction, and call instruction. A jump instruction does not save the current PC, whereas a call instruction saves (“pushes”) the current PC onto the stack and a link instruction saves the PC in the link register IL. Status registers are not affected. Each instruction type has a 2-word format that supports a 20-bit long jump.

JR cc:4, imm:9

imm:9 is a signed number (2’s complement), an offset to be added to the current PC to compute the target (PC[19:12]:(PC[11:0] + imm:9)).

Operation

```
PC[11:0] ← PC[11:0] + imm:9          if branch taken (i.e., cc:4 resolves to be true)
PC[11:0] ← PC[11:0] + 1              otherwise
```

Example

```
L18411:                          // assume current PC = 18411h.
        JR Z, 107h               // next PC is 18518 (18411h + 107h) if Zero (Z) bit is set.
```

LJP cc:4, imm:20

Jumps to the program address specified by imm:20. If program size is less than 64K word, PC[19:16] is not affected.

Operation

```
PC[15:0] ← imm[15:0]    if branch taken and program size is less than 64K word
PC[19:0] ← imm[19:0]    if branch taken and program size is equal to 64K word or more
PC [11:0] ← PC[11:0] + 1 otherwise
```

Example

```
L18411:                          // assume current PC = 18411h.
        LJP Z, 10107h            // next instruction’s PC is 10107h If Zero (Z) bit is set
```

JNZD Rn, imm:8

Jumps to the program address specified by imm:8 if the value of the bank 3 register Rn is not zero. JNZD performs only backward jumps, with the value of Rn automatically decreased. There is one delay slot following the JNZD instruction that is always executed, regardless of whether JNZD is taken or not.

Operation

```
If (Rn == 0) PC ← PC[delay slot] (-) 2’s complement of imm:8, Rn ← Rn - 1
else PC ← PC[delay slot] + 1, Rn ← Rn - 1.
```

Example

```
LOOP_A:                               // start of loop body
          •
          •
          •
          JNZD R0, LOOP_A          // jump back to LOOP_A if R0 is not zero
          ADD R1, #2                // delay slot, always executed (you must use one cycle instruction only)
```

CALLS imm:12

Saves the current PC on the stack (“pushes” PC) and jumps to the program address specified by imm:12. The current page number PC[19:12] is not changed. Since this is a 1-word instruction, the return address pushed onto the stack is (PC + 1). If nP64KW is low when PC is saved, PC[19:16] is not saved in the stack.

Operation

```
HS[sptr][15:0] ← current PC + 1 and sptr ← sptr + 2 (push stack)          if nP64KW = 0
HS[sptr][19:0] ← current PC + 1 and sptr ← sptr + 2 (push stack)          if nP64KW = 1
PC[11:0] ← imm:12
```

Example

```
L18411:                               // assume current PC = 18411h.
          CALLS 107h                  // call the subroutine at 18107h, with the current PC pushed
                                      // onto the stack (HS ← 18412h) if nP64KW = 1.
```

LCALL cc:4, imm:20

Saves the current PC onto the stack (pushes PC) and jumps to the program address specified by imm:20. Since this is a 2-word instruction, the return address saved in the stack is (PC + 2). If nP64KW, a core input signal is low when PC is saved, 0000111111PC[19:16] is not saved in the stack and PC[19:16] is not set to imm[19:16].

Operation

```
HS[sptr][15:0] ← current PC + 2 and sptr + 2 (push stack)   if branch taken and nP64KW = 0
HS[sptr][19:0] ← current PC + 2 and sptr + 2 (push stack)   if branch taken and nP64KW = 1
PC[15:0] ← imm[15:0]    if branch taken and nP64KW = 0
PC[19:0] ← imm[19:0]    if branch taken and nP64KW = 1
PC[11:0] ← PC[11:0] + 2   otherwise
```

Example

```
L18411:                               // assume current PC = 18411h.
          LCALL NZ, 10h:107h          // call the subroutine at 10107h with the current PC pushed
                                      // onto the stack (HS ← 18413h)
```

LNKS imm:12

Saves the current PC in IL and jumps to the program address specified by imm:12. The current page number PC[19:12] is not changed. Since this is a 1-word instruction, the return address saved in IL is (PC + 1). If the program size is less than 64K word when PC is saved, PC[19:16] is not saved in ILX.

Operation

```
IL[15:0] ← current PC + 1          if program size is less than 64K word
IL[19:0] ← current PC + 1          if program size is equal to 64K word or more
PC[11:0] ← imm:12
```

Example

```
L18411:                           // assume current PC = 18411h.
        LNKS 107h                 // call the subroutine at 18107h, with the current PC saved
                                  // in IL (IL[19:0] ← 18412h) if program size is 64K word or more.
```

LLNK cc:4, imm:20

Saves the current PC in IL and jumps to the program address specified by imm:20. Since this is a 2-word instruction, the return address saved in IL is (PC + 2). If the program size is less than 64K word when PC is saved, PC[19:16] is not saved in ILX.

Operation

```
IL[15:0] ← current PC + 2   if branch taken and program size is less than 64K word
IL[19:0] ← current PC + 2   if branch taken and program size is 64K word or more
PC[15:0] ← imm[15:0]    if branch taken and program size is less than 64K word
PC[19:0] ← imm[19:0]    if branch taken and program size is 64K word or more
PC[11:0] ← PC[11:0] + 2   otherwise
```

Example

```
L18411:                           // assume current PC = 18411h.
        LLNK NZ, 10h:107h         // call the subroutine at 10107h with the current PC saved
                                  // in IL (IL[19:0] ← 18413h) if program size is 64K word or more
```

RET, IRET

Returns from the current subroutine. IRET sets ie (SR0[1]) in addition. If the program size is less than 64K word, PC[19:16] is not loaded from HS[19:16].

Operation

```
PC[15:0] ← HS[sptr - 2] and sptr ← sptr - 2 (pop stack) if program size is less than 64K word
PC[19:0] ← HS[sptr - 2] and sptr ← sptr - 2 (pop stack) if program size is 64K word or more
```

Example

```
        RET                       // assume sptr = 3h and HS[1] = 18407h.
                                  // the next PC will be 18407h and sptr is set to 1h
```

LRET

Returns from the current subroutine, using the link register IL. If the program size is less than 64K word, PC[19:16] is not loaded from ILX.

Operation

PC[15:0] ← IL[15:0]	if program size is less than 64K word
PC[19:0] ← IL[19:0]	if program size is 64K word or more

Example

```
LRET                // assume IL = 18407h.
                    // the next instruction to execute is at PC = 18407h
                    // if program size is 64K word or more
```

JP/LNK/CALL

JP/LNK/CALL instructions are pseudo instructions. If JR/LNKS/CALLS commands (1 word instructions) can access the target address, there is no conditional code in the case of CALL/LNK and the JP/LNK/CALL commands are assembled to JR/LNKS/CALLS in linking time or else the JP/LNK/CALL commands are assembled to LJP/LLNK/LCALL (2 word instructions) instructions.

BIT MANIPULATION INSTRUCTIONS

BITop adr:8.bs

Performs a bit operation specified by op on the value in the data memory pointed by adr:8 and stores the result into R3 of current GPR bank or back into memory depending on the value of TF bit.

BITop = BITS, BITR, BITC, BITT
BITS: bit set
BITR: bit reset
BITC: bit complement
BITT: bit test (R3 is not touched in this case)
bs: bit location specifier, 0 - 7.

Operation

R3 ← DM[00h:adr:8] BITop bs if eid = 0
R3 ← DM[IDH:adr:8] BITop bs if eid = 1 (no register transfer for BITT)
Set the Zero (Z) bit if the result is 0.

Example

```
BITS 25h.3          // assume eid = 0. set bit 3 of DM[00h:25h] and store the result in R3.
BITT 25h.3          // check bit 3 of DM[00h:25h] if eid = 0.
```

BMC/BMS

Clears or sets the TF bit, which is used to determine the destination of BITop instructions. When TF bit is clear, the result of BITop instructions will be stored into R3 (fixed); if the TF bit is set, the result will be written back to memory.

Operation

TF ← 0	(BMC)
TF ← 1	(BMS)

TM GPR, #imm:8

Performs AND operation on GPR and imm:8 and sets the Zero (Z) and Negative (N) bits. No change in GPR.

Operation

Z, N flag ← GPR & #imm:8

BITop GPR.bs

Performs a bit operation on GPR and stores the result in GPR.
Since the equivalent functionality can be achieved using OR GPR, #imm:8, AND GPR, #imm:8, and XOR GPR, #imm:8, this instruction type doesn’t have separate op codes.

AND SR0, #imm:8/OR SR0, #imm:8

Sets/resets bits in SR0 and stores the result back into SR0.

Operation

```
SR0 ← SR0 & #imm:8
SR0 ← SR0 | #imm:8
```

BANK #imm:2

Loads SR0[4:3] with #imm[1:0].

Operation

```
SR0[4:3] ← #imm[1:0]
```

MISCELLANEOUS INSTRUCTION

SWAP GPR, SPR

Swaps the values in GPR and SPR. SR0 and SR1 can NOT be used for this instruction.
No flag is updated, even though the destination is GPR.

Operation

```
temp ← SPR
SPR ← GPR
GPR ← temp
```

Example

```
SWAP R0, IDH            // assume IDH = 00h and R0 = 08h.
                        // after this, IDH = 08h and R0 = 00h.
```

PUSH REG

Saves REG in the stack (Pushes REG into stack).
REG = GPR, SPR

Operation

```
HS[sptr][7:0] ← REG and sptr ← sptr + 1
```

Example

```
PUSH R0                 // assume R0 = 08h and sptr = 2h
                        // then HS[2][7:0] ← 08h and sptr ← 3h
```

POP REG

Pops stack into REG.
REG = GPR, SPR

Operation

REG ← HS[sptr-1][7:0] and sptr ← sptr – 1

Example

```
        POP R0              // assume sptr = 3h and HS[2] = 18407h
                            // R0 ← 07h and sptr ← 2h
```

POP

Pops 2 bytes from the stack and discards the popped data.

NOP

Does no work but increase PC by 1.

BREAK

Does nothing and does NOT increment PC. This instruction is for the debugger only. When this instruction is executed, the processor is locked since PC is not incremented. Therefore, this instruction should not be used under any mode other than the debug mode.

SYS #imm:8

Does nothing but increase PC by 1 and generates SYSCP[7:0] and nSYSID signals.

CLD GPR, imm:8

GPR ← (imm:8) and generates SYSCP[7:0], nCLDID, and nCLDWR signals.

CLD imm:8, GPR

(imm:8) ← GPR and generates SYSCP[7:0], nCLDID, and nCLDWR signals.

COP #imm:12

Generates SYSCP[11:0] and nCOPID signals.

LDC

Loads program memory item into register.

Operation

[TBH:TBL] ← PM[ILX:ILH:ILL]	(LDC @IL)
[TBH:TBL] ← PM[ILX:ILH:ILL], ILL++	(LDC @IL+)

TBH and TBL are temporary registers to hold the transferred program memory items. These can be accessed only by LD GPR and TBL/TBH instruction.

Example

```
LD ILX, R1          // assume R1:R2:R3 has the program address to access
LD ILH, R2
LD ILL, R3
LDC @IL             // get the program data @(ILX:ILH:ILL) into TBH:TBL
```

PSEUDO INSTRUCTIONS

EI/DI

Exceptions enable and disable instruction.

Operation

```
SR0 ← OR   SR0,#00000010b    (EI)
SR0 ← AND SR0,#11111101b    (DI)
```

Exceptions are enabled or disabled through this instruction. If there is an EI instruction, the SR0.1 is set and reset, when DI instruction.

Example

```
DI
•
•
•
EI
```

SCF/RCF

Carry flag set and reset instruction.

Operation

```
CP R0,R0         (SCF)
AND R0,R0        (RCF)
```

Carry flag is set or reset through this instruction. If there is an SCF instruction, the SR1.0 is set and reset, when RCF instruction.

Example

```
SCF
RCF
```

STOP/IDLE

MCU power saving instruction.

Operation

```
SYS #0Ah        (STOP)
SYS #05h        (IDLE)
```

The STOP instruction stops the both CPU clock and system clock and causes the microcontroller to enter STOP mode. The IDLE instruction stops the CPU clock while allowing system clock oscillation to continue.

Example

```
STOP(or IDLE)
NOP
NOP
NOP
•
•
```

ADC — Add with Carry

Format: ADC <op1>, <op2>
<op1>: GPR
<op2>: adr:8, GPR

Operation: <op1> ← <op1> + <op2> + C
ADC adds the values of <op1> and <op2> and carry (C) and stores the result back into <op1>

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
. **N:** exclusive OR of V and MSB of result.

Example:

```
ADC     R0, 80h             // If eid = 0, R0 ← R0 + DM[0080h] + C
                            // If eid = 1, R0 ← R0 + DM[IDH:80h] + C

ADC     R0, R1              // R0 ← R0 + R1 + C

ADD     R0, R2
ADC     R1, R3
```

In the last two instructions, assuming that register pair R1:R0 and R3:R2 are 16-bit signed or unsigned numbers. Even if the result of “ADD R0, R2” is not zero, Z flag can be set to ‘1’ if the result of “ADC R1,R3” is zero. Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit addition, take care of the change of Z flag.

ADD — Add

Format: ADD <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> + <op2>

ADD adds the values of <op1> and <op2> and stores the result back into <op1>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
. **N:** exclusive OR of V and MSB of result.

Example: Given: IDH:IDL0 = 80FFh, eid = 1

```
ADD      R0, 80h              // R0 ← R0 + DM[8080h]

ADD      R0, #12h             // R0 ← R0 + 12h

ADD      R1, R2               // R1 ← R1 + R2

ADD      R0, @ID0 + 2         // R0 ← R0 + DM[80FFh], IDH ← 81h, IDL0 ← 01h
ADD      R0, @[ID0 – 3]       // R0 ← R0 + DM[80FCh], IDH ← 80h, IDL0 ← FCh
ADD      R0, @[ID0 + 2]!      // R0 ← R0 + DM[8101h], IDH ← 80h, IDL0 ← FFh
ADD      R0, @[ID0 – 2]!      // R0 ← R0 + DM[80FDh], IDH ← 80h, IDL0 ← FFh
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

AND — Bit-wise AND

Format: AND <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> & <op2>

AND performs bit-wise AND on the values in <op1> and <op2> and stores the result in <op1>.

Flags: **Z:** set if result is zero. Reset if not.
N: set if the MSB of result is 1. Reset if not.

Example: Given: IDH:IDL0 = 01FFh, eid = 1

```
AND     R0, 7Ah             // R0 ← R0 & DM[017Ah]

AND     R1, #40h            // R1 ← R1 & 40h

AND     R0, R1              // R0 ← R0 & R1

AND     R1, @ID0 + 3        // R1 ← R1 & DM[01FFh], IDH:IDL0 ← 0202h
AND     R1, @[ID0 – 5]      // R1 ← R1 & DM[01FAh], IDH:IDL0 ← 01FAh
AND     R1, @[ID0 + 7]!     // R1 ← R1 & DM[0206h], IDH:IDL0 ← 01FFh
AND     R1, @[ID0 – 2]!     // R1 ← R1 & DM[01FDh], IDH:IDL0 ← 01FFh
```

In the first instruction, if eid bit in SR0 is zero, register R0 has garbage value because data memory DM[0051h-007Fh] are not mapped in S3CB018/S3FB018. In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

AND SR0 — Bit-wise AND with SR0

Format: AND SR0, #imm:8

Operation: SR0 ← SR0 & imm:8

AND SR0 performs the bit-wise AND operation on the value of SR0 and imm:8 and stores the result in SR0.

Flags: –

Example: Given: SR0 = 11000010b

```
nIE     EQU     ~02h
nIE0    EQU     ~40h
nIE1    EQU     ~80h

        AND     SR0, #nIE | nIE0 | nIE1

        AND     SR0, #11111101b
```

In the first example, the statement “AND SR0, #nIE|nIE0|nIE1” clear all of bits of the global interrupt, interrupt 0 and interrupt 1. On the contrary, cleared bits can be set to ‘1’ by instruction “OR SR0, #imm:8”. Refer to instruction OR SR0 for more detailed explanation about enabling bit.

In the second example, the statement “AND SR0, #11111101b” is equal to instruction DI, which is disabling interrupt globally.

BANK — GPR Bank selection

Format: BANK #imm:2

Operation: SR0[4:3] ← imm:2

Flags: –

NOTE: For explanation of the CalmRISC banked register file and its usage, please refer to chapter 3.

Example:

```
BANK     #1                    // Select register bank 1
LD       R0, #11h              // Bank1's R0 ← 11h

BANK     #2                    // Select register bank 2
LD       R1, #22h              // Bank2's R1 ← 22h
```

BITC — Bit Complement

Format: BITC adr:8.bs

bs: 3-digit bit specifier

Operation: R3 ← ((adr:8) ^ (2**bs)) if (TF == 0)

(adr:8) ← ((adr:8) ^ (2**bs)) if (TF == 1)

BITC complements the specified bit of a value read from memory and stores the result in R3 or back into memory, depending on the value of TF. TF is set or clear by BMS/BMC instruction.

Flags: **Z:** set if result is zero. Reset if not.

NOTE: Since the destination register R3 is fixed, it is not specified explicitly.

Example: Given: IDH = 01, DM[0180h] = FFh, eid = 1

```
BMC                         // TF ← 0
BITC     80h.0              // R3 ← FEh, DM[0180h] = FFh

BMS                         // TF ← 1
BITC     80h.1              // DM[0180h] ← FDh
```

BITR — Bit Reset

Format:	BITR adr:8.bs
	bs: 3-digit bit specifier
Operation:	R3 ← ((adr:8) & ((11111111)$_2$ - (2**bs))) if (TF == 0)
	(adr:8) ← ((adr:8) & ((11111111)$_2$ - (2**bs))) if (TF == 1)
	BITR resets the specified bit of a value read from memory and stores the result in R3 or back into memory, depending on the value of TF. TF is set or clear by BMS/BMC instruction.
Flags:	**Z:** set if result is zero. Reset if not.
NOTE:	Since the destination register R3 is fixed, it is not specified explicitly.
Example:	Given: IDH = 01, DM[0180h] = FFh, eid = 1

```
BMC                       // TF ← 0
BITR     80h.1            // R3 ← FDh, DM[0180h] = FFh

BMS                       // TF ← 1
BITR     80h.2            // DM[0180h] ← FBh
```

BITS — Bit Set

Format: BITS adr:8.bs

bs: 3-digit bit specifier.

Operation: R3 ← ((adr:8) | (2**bs))   if (TF == 0)

(adr:8) ← ((adr:8) | (2**bs))   if (TF == 1)

BITS sets the specified bit of a value read from memory and stores the result in R3 or back into memory, depending on the value of TF. TF is set or clear by BMS/BMC instruction.

Flags: **Z:** set if result is zero. Reset if not.

NOTE: Since the destination register R3 is fixed, it is not specified explicitly.

Example: Given: IDH = 01, DM[0180h] = F0h, eid = 1

```
BMC                     // TF ← 0
BITS     80h.1          // R3 ← 0F2h, DM[0180h] = F0h

BMS                     // TF ← 1
BITS     80h.2          // DM[0180h] ← F4h
```

BITT — Bit Test

Format: BITT adr:8.bs

bs: 3-digit bit specifier.

Operation: Z ← ~((adr:8) & (2**bs))

BITT tests the specified bit of a value read from memory.

Flags: **Z:** set if result is zero. Reset if not.

Example: Given: DM[0080h] = F7h, eid = 0

```
        BITT    80h.3               // Z flag is set to '1'
        JR      Z, %1               // Jump to label %1 because condition is true.
        •
        •
        •
%1      BITS    80h.3
        NOP
        •
        •
        •
```

BMC/BMS – TF bit clear/set

Format: BMS/BMC

Operation: BMC/BMS clears (sets) the TF bit.

TF ← 0 if BMC

TF ← 1 if BMS

TF is a single bit flag which determines the destination of bit operations, such as BITC, BITR, and BITS.

Flags: –

NOTE: BMC/BMS are the only instructions that modify the content of the TF bit.

Example:

```
BMS                          // TF ← 1
BITS     81h.1

BMC                          // TF ← 0
BITR     81h.2
LD       R0, R3
```

CALL — Conditional Subroutine Call (Pseudo Instruction)

Format: CALL cc:4, imm:20
CALL imm:12

Operation: If CALLS can access the target address and there is no conditional code (cc:4), CALL command is assembled to CALLS (1-word instruction) in linking time, else the CALL is assembled to LCALL (2-word instruction).

Example:

```
        CALL    C, Wait             // HS[sptr][15:0] ← current PC + 2, sptr ← sptr + 2
        •                           // 2-word instruction
        •
        •
        CALL    0088h               // HS[sptr][15:0] ← current PC + 1, sptr ← sptr + 2
        •                           // 1-word instruction
        •
        •
Wait:   NOP                         // Address at 0088h
        NOP
        NOP
        NOP
        NOP
        RET
```

CALLS — Call Subroutine

Format: CALLS imm:12

Operation: HS[sptr][15:0] ← current PC + 1, sptr ← sptr + 2 if the program size is less than 64K word.

HS[sptr][19:0] ← current PC + 1, sptr ← sptr + 2 if the program size is equal to or over 64K word.

PC[11:0] ← imm:12
CALLS unconditionally calls a subroutine residing at the address specified by imm:12.

Flags: –

Example:

```
        CALLS    Wait
        •
        •
        •
Wait:   NOP
        NOP
        NOP
        RET
```

Because this is a 1-word instruction, the saved returning address on stack is (PC + 1).

CLD — Load into Coprocessor

Format: CLD imm:8, <op>

<op>: GPR

Operation: (imm:8) ← <op>

CLD loads the value of <op> into (imm:8), where imm:8 is used to access the external coprocessor's address space.

Flags: –

Example:

```
AH      EQU     00h
AL      EQU     01h
BH      EQU     02h
BL      EQU     03h
        •
        •
        •
        CLD     AH, R0              // A[15:8] ← R0
        CLD     AL, R1              // A[7:0] ← R1

        CLD     BH, R2              // B[15:8] ← R2
        CLD     BL, R3              // B[7:0] ← R3
```

The registers A[15:0] and B[15:0] are Arithmetic Unit (AU) registers of MAC816.
Above instructions generate SYSCP[7:0], nCLDID and CLDWR signals to access MAC816.

CLD — Load from Coprocessor

Format: CLD <op>, imm:8

<op>: GPR

Operation: <op> ← (imm:8)

CLD loads a value from the coprocessor, whose address is specified by imm:8.

Flags:
Z: set if the loaded value in <op1> is zero. Reset if not.
N: set if the MSB of the loaded value in <op1> is 1. Reset if not.

Example:

```
AH      EQU    00h
AL      EQU    01h
BH      EQU    02h
BL      EQU    03h
        •
        •
        •
        CLD    R0, AH           // R0 ← A[15:8]
        CLD    R1, AL           // R1 ← A[7:0]

        CLD    R2, BH           // R2 ← B[15:8]
        CLD    R3, BL           // R3 ← B[7:0]
```

The registers A[15:0] and B[15:0] are Arithmetic Unit (AU) registers of MAC816.
Above instructions generate SYSCP[7:0], nCLDID and CLDWR signals to access MAC816.

COM — 1's or Bit-wise Complement

Format: COM <op>

<op>: GPR

Operation: <op> ← ~<op>

COM takes the bit-wise complement operation on <op> and stores the result in <op>.

Flags: **Z:** set if result is zero. Reset if not.
N: set if the MSB of result is 1. Reset if not.

Example: Given: R1 = 5Ah

```
COM     R1                        // R1 ← A5h, N flag is set to '1'
```

COM2 — 2's Complement

Format: COM2 <op>

<op>: GPR

Operation: <op> ← ~<op> + 1

COM2 computes the 2's complement of <op> and stores the result in <op>.

Flags:

C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative.

Example: Given: R0 = 00h, R1 = 5Ah

```
COM2    R0                    // R0 ← 00h, Z and C flags are set to ‘1’.

COM2    R1                    // R1 ← A6h, N flag is set to ‘1’.
```

COMC — Bit-wise Complement with Carry

Format: COMC <op>

<op>: GPR

Operation: <op> ← ~<op> + C

COMC takes the bit-wise complement of <op>, adds carry and stores the result in <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: If register pair R1:R0 is a 16-bit number, then the 2's complement of R1:R0 can be obtained by COM2 and COMC as following.

```
COM2            R0
COMC            R1
```

Note that Z flag do not exactly reflect result of 16-bit operation. For example, if 16-bit register pair R1: R0 has value of FF01h, then 2's complement of R1: R0 is made of 00FFh by COM2 and COMC. At this time, by instruction COMC, zero (Z) flag is set to '1' as if the result of 2's complement for 16-bit number is zero. Therefore when programming 16-bit comparison, take care of the change of Z flag.

COP — Coprocessor

Format: COP #imm:12

Operation: COP passes imm:12 to the coprocessor by generating SYSCP[11:0] and nCOPID signals.

Flags: –

Example:

```
COP     #0D01h                  // generate 1 word instruction code(FD01h)
COP     #0234h                  // generate 1 word instruction code(F234h)
```

The above two instructions are equal to statement “ELD A, #1234h” for MAC816 operation. The microcode of MAC instruction “ELD A, #1234h” is “FD01F234”, 2-word instruction. In this, code ‘F’ indicates ‘COP’ instruction.

CP — Compare

Format: CP <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> + ~<op2> + 1

CP compares the values of <op1> and <op2> by subtracting <op2> from <op1>. Contents of <op1> and <op2> are not changed.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero (i.e., <op1> and <op2> are same). Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: R0 = 73h, R1 = A5h, IDH:IDL0 = 0123h, DM[0123h] = A5, eid = 1

```
CP      R0, 80h            // C flag is set to '1'

CP      R0, #73h           // Z and C flags are set to '1'

CP      R0, R1             // V flag is set to '1'

CP      R1, @ID0           // Z and C flags are set to '1'
CP      R1, @[ID0 – 5]
CP      R2, @[ID0 + 7]!
CP      R2, @[ID0 – 2]!
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

CPC — Compare with Carry

Format: CPC <op1>, <op2>

<op1>: GPR
<op2>: adr:8, GPR

Operation: <op1> ← <op1> + ~<op2> + C

CPC compares <op1> and <op2> by subtracting <op2> from <op1>. Unlike CP, however, CPC adds (C - 1) to the result. Contents of <op1> and <op2> are not changed.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: If register pair R1:R0 and R3:R2 are 16-bit signed or unsigned numbers, then use CP and CPC to compare two 16-bit numbers as follows.

```
CP          R0, R1
CPC         R2, R3
```

Because CPC considers C when comparing <op1> and <op2>, CP and CPC can be used in pair to compare 16-bit operands. But note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit comparison, take care of the change of Z flag.

DEC — Decrement

Format: DEC <op>

<op>: GPR

Operation: <op> ← <op> + 0FFh

DEC decrease the value in <op> by adding 0FFh to <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: R0 = 80h, R1 = 00h

```
DEC     R0                  // R0 ← 7Fh, C, V and N flags are set to '1'

DEC     R1                  // R1 ← FFh, N flags is set to '1'
```

DECC — Decrement with Carry

Format: DECC <op>

<op>: GPR

Operation: <op> ← <op> + 0FFh + C

DECC decrease the value in <op> when carry is not set. When there is a carry, there is no change in the value of <op>.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: If register pair R1:R0 is 16-bit signed or unsigned number, then use DEC and DECC to decrement 16-bit number as follows.

```
DEC     R0
DECC    R1
```

Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit decrement, take care of the change of Z flag.

DI — Disable Interrupt (Pseudo Instruction)

Format: DI

Operation: Disables interrupt globally. It is same as “AND SR0, #0FDh” .
DI instruction sets bit1 (ie: global interrupt enable) of SR0 register to “0”

Flags: –

Example: Given: SR0 = 03h

```
DI                                    // SR0 ← SR0 & 11111101b
```

DI instruction clears SR0[1] to ‘0’, disabling interrupt processing.

EI — Enable Interrupt (Pseudo Instruction)

Format:	EI
Operation:	Enables interrupt globally. It is same as “OR SR0, #02h” . EI instruction sets the bit1 (ie: global interrupt enable) of SR0 register to “1”
Flags:	–
Example:	Given: SR0 = 01h

```
EI                                   // SR0 ← SR0 | 00000010b
```

The statement “EI” sets the SR0[1] to ‘1’, enabling all interrupts.

IDLE — Idle Operation (Pseudo Instruction)

Format: IDLE

Operation: The IDLE instruction stops the CPU clock while allowing system clock oscillation to continue. Idle mode can be released by an interrupt or reset operation.
The IDLE instruction is a pseudo instruction. It is assembled as "SYS #05H", and this generates the SYSCP[7-0] signals. Then these signals are decoded and the decoded signals execute the idle operation.

Flags: –

NOTE: The next instruction of IDLE instruction is executed, so please use the NOP instruction after the IDLE instruction.

Example:

```
IDLE
NOP
NOP
NOP
•
•
•
```

The IDLE instruction stops the CPU clock but not the system clock.

INC — Increment

Format:	INC <op>
	<op>: GPR
Operation:	<op> ← <op> + 1
	INC increase the value in <op>.
Flags:	**C:** set if carry is generated. Reset if not. **Z:** set if result is zero. Reset if not. **V:** set if overflow is generated. Reset if not. **N:** set if result is negative. Reset if not.
Example:	Given: R0 = 7Fh, R1 = FFh

```
INC     R0              // R0 ← 80h, V flag is set to '1'

INC     R1              // R1 ← 00h, Z and C flags are set to '1'
```

INCC — Increment with Carry

Format: INCC <op>

<op>: GPR

Operation: <op> ← <op> + C

INCC increase the value of <op> only if there is carry. When there is no carry, the value of <op> is not changed.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: exclusive OR of V and MSB of result.

Example: If register pair R1:R0 is 16-bit signed or unsigned number, then use INC and INCC to increment 16-bit number as following.

```
INC         R0
INCC        R1
```

Assume R1:R0 is 0010h, statement “INC R0” increase R0 by one without carry and statement “INCC R1” set zero (Z) flag to ‘1’ as if the result of 16-bit increment is zero. Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit increment, take care of the change of Z flag.

IRET — Return from Interrupt Handling

Format: IRET

Operation: PC ← HS[sptr - 2], sptr ← sptr - 2

IRET pops the return address (after interrupt handling) from the hardware stack and assigns it to PC. The ie (i.e., SR0[1]) bit is set to allow further interrupt generation.

Flags: –

NOTE: The program size (indicated by the nP64KW signal) determines which portion of PC is updated. When the program size is less than 64K word, only the lower 16 bits of PC are updated (i.e., PC[15:0] ← HS[sptr – 2]).
When the program size is 64K word or more, the action taken is PC[19:0] ← HS[sptr - 2].

Example:

```
SF_EXCEP:    NOP                  // Stack full exception service routine
             •
             •
             •
             IRET
```

JNZD — Jump Not Zero with Delay slot

Format: JNZD <op>, imm:8

<op>: GPR (bank 3's GPR only)

imm:8 is an signed number

Operation: PC ← PC[delay slot] - 2's complement of imm:8

<op> ← <op> - 1

JNZD performs a backward PC-relative jump if <op> evaluates to be non-zero. Furthermore, JNZD decrease the value of <op>. The instruction immediately following JNZD (i.e., in delay slot) is always executed, and this instruction must be 1 cycle instruction.

Flags: –

NOTE: Typically, the delay slot will be filled with an instruction from the loop body. It is noted, however, that the chosen instruction should be "dead" outside the loop for it executes even when the loop is exited (i.e., JNZD is not taken).

Example: Given: IDH = 03h, eid = 1

```
        BANK    #3
        LD      R0, #0FFh               // R0 is used to loop counter
        LD      R1, #0
%1      LD      IDL0, R0
        JNZD    R0, %B1                 // If R0 of bank3 is not zero, jump to %1.
        LD      @ID0, R1                // Clear register pointed by ID0
        •
        •
        •
```

This example can be used for RAM clear routine. The last instruction is executed even if the loop is exited.

JP — Conditional Jump (Pseudo Instruction)

Format: JP cc:4 imm:20
JP cc:4 imm:9

Operation: If JR can access the target address, JP command is assembled to JR (1 word instruction) in linking time, else the JP is assembled to LJP (2 word instruction) instruction.
There are 16 different conditions that can be used, as described in table 8-6.

Example:

```
%1    LD     R0, #10h          // Assume address of label %1 is 020Dh
      •
      •
      •
      JP     Z, %B1            // Address at 0264h

      JP     C, %F2            // Address at 0265h
      •
      •
      •
%2    LD     R1, #20h          // Assume address of label %2 is 089Ch
      •
      •
      •
```

In the above example, the statement “JP Z, %B1” is assembled to JR instruction. Assuming that current PC is 0264h and condition is true, next PC is made by PC[11:0] ← PC[11:0] + offset, offset value is “64h + A9h” without carry. ‘A9’ means 2’s complement of offset value to jump backward. Therefore next PC is 020Dh. On the other hand, statement “JP C, %F2” is assembled to LJP instruction because offset address exceeds the range of imm:9.

JR — Conditional Jump Relative

Format: JR cc:4 imm:9

cc:4: 4-bit condition code

Operation: PC[11:0] ← PC[11:0] + imm:9 if condition is true. imm:9 is a signed number, which is sign-extended to 12 bits when added to PC.
There are 16 different conditions that can be used, as described in table 8-6.

Flags: –

NOTE: Unlike LJP, the target address of JR is PC-relative. In the case of JR, imm:9 is added to PC to compute the actual jump address, while LJP directly jumps to imm:20, the target.

Example:

```
        JR      Z, %1           // Assume current PC = 1000h
        •
        •
        •
%1      LD      R0, R1          // Address at 10A5h
        •
        •
        •
```

After the first instruction is executed, next PC has become 10A5h if Z flag bit is set to ‘1’. The range of the relative address is from +255 to –256 because imm:9 is signed number.

LCALL — Conditional Subroutine Call

Format: LCALL cc:4, imm:20

Operation: HS[sptr][15:0] ← current PC + 2, sptr ← sptr + 2, PC[15:0] ← imm[15:0] if the condition holds and the program size is less than 64K word.

HS[sptr][19:0] ← current PC + 2, sptr ← sptr + 2, PC[19:0] ← imm:20 if the condition holds and the program size is equal to or over 64K word.

PC[11:0] ← PC[11:0] + 2 otherwise.
LCALL instruction is used to call a subroutine whose starting address is specified by imm:20.

Flags: –

Example:

```
LCALL     L1

LCALL     C, L2
```

Label L1 and L2 can be allocated to the same or other section. Because this is a 2-word instruction, the saved returning address on stack is (PC + 2).

LD adr:8 — Load into Memory

Format:	LD adr:8, <op>
	<op>: GPR
Operation:	DM[00h:adr:8] ← <op> if eid = 0 DM[IDH:adr:8] ← <op> if eid = 1
	LD adr:8 loads the value of <op> into a memory location. The memory location is determined by the eid bit and adr:8.
Flags:	–
Example:	Given: IDH = 01h

```
LD        80h, R0
```

If eid bit of SR0 is zero, the statement “LD 80h, R0” load value of R0 into DM[0080h], else eid bit was set to ‘1’, the statement “LD 80h, R0” load value of R0 into DM[0180h]

LD @idm — Load into Memory Indexed

Format: LD @idm, <op>

<op>: GPR

Operation: (@idm) ← <op>

LD @idm loads the value of <op> into the memory location determined by @idm. Details of the @idm format and how the actual address is calculated can be found in chapter 2.

Flags: –

Example: Given R0 = 5Ah, IDH:IDL0 = 8023h, eid = 1

```
LD          @ID0, R0                  // DM[8023h] ← 5Ah
LD          @ID0 + 3, R0              // DM[8023h] ← 5Ah, IDL0 ← 26h
LD          @[ID0-5], R0              // DM[801Eh] ← 5Ah, IDL0 ← 1Eh
LD          @[ID0+4]!, R0             // DM[8027h] ← 5Ah, IDL0 ← 23h
LD          @[ID0-2]!, R0             // DM[8021h] ← 5Ah, IDL0 ← 23h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

LD — Load Register

Format: LD <op1>, <op2>

<op1>: GPR
<op2>: GPR, SPR, adr:8, @idm, #imm:8

Operation: <op1> ← <op2>

LD loads a value specified by <op2> into the register designated by <op1>.

Flags: **Z:** set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example: Given: R0 = 5Ah, R1 = AAh, IDH:IDL0 = 8023h, eid = 1

```
LD      R0, R1              // R0 ← AAh

LD      R1, IDH             // R1 ← 80h

LD      R2, 80h             // R2 ← DM[8080h]

LD      R0, #11h            // R0 ← 11h

LD      R0, @ID0+1          // R0 ← DM[8023h], IDL0 ← 24h
LD      R1, @[ID0-2]        // R1 ← DM[8021h], IDL0 ← 21h
LD      R2, @[ID0+3]!       // R2 ← DM[8026h], IDL0 ← 23h
LD      R3, @[ID0-5]!       // R3 ← DM[801Eh], IDL0 ← 23h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

LD — Load GPR:bankd, GPR:banks

Format: LD <op1>, <op2>

<op1>: GPR: bankd
<op2>: GPR: banks

Operation: <op1> ← <op2>

LD loads a value of a register in a specified bank (banks) into another register in a specified bank (bankd).

Flags: **Z:** set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example:

```
LD      R2:1, R0:3            // Bank1's R2 ← bank3's R0

LD      R0:0, R0:2            // Bank0's R0 ← bank2's R0
```

LD — Load GPR, TBH/TBL

Format: LD <op1>, <op2>

<op1>: GPR
<op2>: TBH/TBL

Operation: <op1> ← <op2>

LD loads a value specified by <op2> into the register designated by <op1>.

Flags:
Z: set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example: Given: register pair R1:R0 is 16-bit unsigned data.

```
LDC       @IL                      // TBH:TBL ← PM[ILX:ILH:ILL]
LD        R1, TBH                  // R1 ← TBH
LD        R0, TBL                  // R0 ← TBL
```

LD — Load TBH/TBL, GPR

Format: LD <op1>, <op2>

<op1>: TBH/TBL
<op2>: GPR

Operation: <op1> ← <op2>

LD loads a value specified by <op2> into the register designated by <op1>.

Flags: –

Example: Given: register pair R1:R0 is 16-bit unsigned data.

```
LD        TBH, R1               // TBH ← R1
LD        TBL, R0               // TBL ← R0
```

LD SPR — Load SPR

Format: LD <op1>, <op2>

<op1>: SPR
<op2>: GPR

Operation: <op1> ← <op2>

LD SPR loads the value of a GPR into an SPR.
Refer to Table 3-1 for more detailed explanation about kind of SPR.

Flags: –

Example: Given: register pair R1:R0 = 1020h

```
LD          ILH, R1                   // ILH ← 10h
LD          ILL, R0                   // ILL ← 20h
```

LD SPR0 — Load SPR0 Immediate

Format: LD SPR0, #imm:8

Operation: SPR0 ← imm:8

LD SPR0 loads an 8-bit immediate value into SPR0.

Flags: –

Example: Given: eid = 1, idb = 0 (index register bank 0 selection)

```
LD          IDH, #80h               // IDH point to page 80h
LD          IDL1, #44h
LD          IDL0, #55h
LD          SR0, #02h
```

The last instruction set ie (global interrupt enable) bit to ‘1’.
Special register group 1 (SPR1) registers are not supported in this addressing mode.

LDC — Load Code

Format: LDC <op1>

<op1>: @IL, @IL+

Operation: TBH:TBL ← PM[ILX:ILH:ILL]

ILL ← ILL + 1 (@IL+ only)

LDC loads a data item from program memory and stores it in the TBH:TBL register pair.

@IL+ increase the value of ILL, efficiently implementing table lookup operations.

Flags: –

Example:

```
LD      ILX, R1
LD      ILH, R2
LD      ILL, R3
LDC     @IL                 // Loads value of PM[ILX:ILH:ILL] into TBH:TBL

LD      R1, TBH             // Move data in TBH:TBL to GPRs for further processing
LD      R0, TBL
```

The statement “LDC @IL” do not increase, but if you use statement “LDC @IL+”, ILL register is increased by one after instruction execution.

LJP — Conditional Jump

Format: LJP cc:4, imm:20

cc:4: 4-bit condition code

Operation: PC[15:0] ← imm[15:0] if condition is true and the program size is less than 64K word. If the program is equal to or larger than 64K word, PC[19:0] ← imm[19:0] as long as the condition is true. There are 16 different conditions that can be used, as described in table 8-6.

Flags: –

NOTE: LJP cc:4 imm:20 is a 2-word instruction whose immediate field directly specifies the target address of the jump.

Example:

```
        LJP    C, %1          // Assume current PC = 0812h
        •
        •
        •
  %1    LD     R0, R1         // Address at 10A5h
        •
        •
        •
```

After the first instruction is executed, LJP directly jumps to address 10A5h if condition is true.

LLNK — Linked Subroutine Call Conditional

Format: LLNK cc:4, imm:20

cc:4: 4-bit condition code

Operation: If condition is true, IL[19:0] ← {PC[19:12], PC[11:0] + 2}.

Further, when the program is equal to or larger than 64K word, PC[19:0] ← imm[19:0] as long as the condition is true. If the program is smaller than 64K word, PC[15:0] ← imm[15:0].
There are 16 different conditions that can be used, as described in table 8-6.

Flags: –

NOTE: LLNK is used to conditionally to call a subroutine with the return address saved in the link register (IL) without stack operation. This is a 2-word instruction.

Example:

```
        LLNK     Z, %1                  // Address at 005Ch, ILX:ILH:ILL ← 00:00:5Eh
        NOP                             // Address at 005Eh
        •
        •
        •
%1      LD       R0, R1
        •
        •
        •
        LRET
```

LNK — Linked Subroutine Call (Pseudo Instruction)

Format: LNK cc:4, imm:20
LNK imm:12

Operation: If LNKS can access the target address and there is no conditional code (cc:4), LNK command is assembled to LNKS (1 word instruction) in linking time, else the LNK is assembled to LLNK (2 word instruction).

Example:

```
            LNK      Z, Link1           // Equal to "LLNK Z, Link1"
            LNK      Link2              // Equal to "LNKS Link2"
            NOP
            •
            •
            •
Link2:      NOP
            •
            •
            •
            LRET

Subroutines          section CODE, ABS 0A00h
            Subroutines
Link1:      NOP
            •
            •
            •
            LRET
```

LNKS — Linked Subroutine Call

Format: LNKS imm:12

Operation: IL[19:0] ← {PC[19:12], PC[11:0] + 1} and PC[11:0] ← imm:12
LNKS saves the current PC in the link register and jumps to the address specified by imm:12.

Flags: –

NOTE: LNKS is used to call a subroutine with the return address saved in the link register (IL) without stack operation.

Example:

```
        LNKS    Link1                   // Address at 005Ch, ILX:ILH:ILL ← 00:00:5Dh
        NOP                             // Address at 005Dh
        •
        •
        •

Link1:  NOP
        •
        •
        •
        LRET
```

LRET — Return from Linked Subroutine Call

Format: LRET

Operation: PC ← IL[19:0]
LRET returns from a subroutine by assigning the saved return address in IL to PC.

Flags: –

Example:

```
        LNK     Link1
Link1:  NOP
        •
        •
        •
        LRET                            ;   PC[19:0] ← ILX:ILH:ILL
```

NOP — No Operation

Format: NOP

Operation: No operation.

When the instruction NOP is executed in a program, no operation occurs. Instead, the instruction time is delayed by approximately one machine cycle per each NOP instruction encountered.

Flags: –

Example:

```
NOP
```

OR — Bit-wise OR

Format: OR <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> | <op2>
OR performs the bit-wise OR operation on <op1> and <op2> and stores the result in <op1>.

Flags: **Z:** set if result is zero. Reset if not.
N: exclusive OR of V and MSB of result.

Example: Given: IDH:IDL0 = 031Eh, eid = 1

```
OR      R0, 80h           // R0 ← R0 | DM[0380h]

OR      R1, #40h          // Mask bit6 of R1

OR      R1, R0            // R1 ← R1 | R0

OR      R0, @ID0          // R0 ← R0 | DM[031Eh], IDL0 ← 1Eh
OR      R1, @[ID0-1]      // R1 ← R1 | DM[031Dh], IDL0 ← 1Dh
OR      R2, @[ID0+1]!     // R2 ← R2 | DM[031Fh], IDL0 ← 1Eh
OR      R3, @[ID0-1]!     // R3 ← R3 | DM[031Dh], IDL0 ← 1Eh
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

OR SR0 — Bit-wise OR with SR0

Format: OR SR0, #imm:8

Operation: SR0 ← SR0 | imm:8

OR SR0 performs the bit-wise OR operation on SR0 and imm:8 and stores the result in SR0.

Flags: –

Example: Given: SR0 = 00000000b

```
EID     EQU     01h
IE      EQU     02h
IDB1    EQU     04h
IE0     EQU     40h
IE1     EQU     80h

        OR      SR0, #IE | IE0 | IE1

        OR      SR0, #00000010b
```

In the first example, the statement "OR SR0, #EID|IE|IE0" set global interrupt(ie), interrupt 0(ie0) and interrupt 1(ie1) to '1' in SR0. On the contrary, enabled bits can be cleared with instruction "AND SR0, #imm:8". Refer to instruction AND SR0 for more detailed explanation about disabling bit.

In the second example, the statement "OR SR0, #00000010b" is equal to instruction EI, which is enabling interrupt globally.

POP — POP

Format:	POP
Operation:	sptr ← sptr – 2
	POP decrease sptr by 2. The top two bytes of the hardware stack are therefore invalidated.
Flags:	–
Example:	Given: sptr[5:0] = 001010b
	POP
	This POP instruction decrease sptr[5:0] by 2. Therefore sptr[5:0] is 001000b.

POP — POP to Register

Format: POP <op>

<op>: GPR, SPR

Operation: <op> ← HS[sptr - 1], sptr ← sptr - 1

POP copies the value on top of the stack to <op> and decrease sptr by 1.

Flags: **Z:** set if the value copied to <op> is zero. Reset if not.
N: set if the value copied to <op> is negative. Reset if not.
When <op> is SPR, no flags are affected, including Z and N.

Example:

```
POP     R0          // R0 ← HS[sptr-1], sptr  ← sptr-1

POP     IDH         // IDH ← HS[sptr-1], sptr  ← sptr-1
```

In the first instruction, value of HS[sptr-1] is loaded to R0 and the second instruction “POP IDH” load value of HS[sptr-1] to register IDH. Refer to chapter 5 for more detailed explanation about POP operations for hardware stack.

PUSH — Push Register

Format: PUSH <op>

<op>: GPR, SPR

Operation: HS[sptr] ← <op>, sptr ← sptr + 1

PUSH stores the value of <op> on top of the stack and increase sptr by 1.

Flags: –

Example:

```
PUSH      R0                        // HS[sptr] ← R0, sptr ← sptr + 1

PUSH      IDH                       // HS[sptr] ← IDH, sptr ← sptr + 1
```

In the first instruction, value of register R0 is loaded to HS[sptr-1] and the second instruction "PUSH IDH" load value of register IDH to HS[sptr-1]. Current HS pointed by stack point sptr[5:0] be emptied. Refer to chapter 5 for more detailed explanation about PUSH operations for hardware stack.

RET — Return from Subroutine

Format: RET

Operation: PC ← HS[sptr - 2], sptr ← sptr – 2

RET pops an address on the hardware stack into PC so that control returns to the subroutine call site.

Flags: –

Example: Given: sptr[5:0] = 001010b

```
       CALLS    Wait                  // Address at 00120h
       •
       •
       •
Wait:  NOP                            // Address at 01000h
       NOP
       NOP
       NOP
       NOP
       RET
```

After the first instruction CALLS execution, “PC+1”, 0121h is loaded to HS[5] and hardware stack pointer sptr[5:0] have 001100b and next PC became 01000h. The instruction RET pops value 0121h on the hardware stack HS[sptr-2] and load to PC then stack pointer sptr[[5:0] became 001010b.

RL — Rotate Left

Format:	RL <op>
	<op>: GPR
Operation:	C ← <op>[7], <op> ← {<op>[6:0], <op>[7]}
	RL rotates the value of <op> to the left and stores the result back into <op>. The original MSB of <op> is copied into carry (C).
Flags:	**C:** set if the MSB of <op> (before rotating) is 1. Reset if not. **Z:** set if result is zero. Reset if not. **N:** set if the MSB of <op> (after rotating) is 1. Reset if not.
Example:	Given: R0 = 01001010b, R1 = 10100101b

```
RL    R0              // N flag is set to '1', R0 ← 10010100b

RL    R1              // C flag is set to '1', R1 ← 01001011b
```

RLC — Rotate Left with Carry

Format: RLC <op>

<op>: GPR

Operation: C ← <op>[7], <op> ← {<op>[6:0], C}

RLC rotates the value of <op> to the left and stores the result back into <op>.
The original MSB of <op> is copied into carry (C), and the original C bit is copied into <op>[0].

Flags:
C: set if the MSB of <op> (before rotating) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after rotating) is 1. Reset if not.

Example: Given: R2 = A5h, if C = 0

```
RLC      R2                    // R2 ← 4Ah, C flag is set to '1'

RL       R0
RLC      R1
```

In the second example, assuming that register pair R1:R0 is 16-bit number, then RL and RLC are used for 16-bit rotate left operation. But note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit decrement, take care of the change of Z flag.

RR — Rotate Right

Format: RR <op>

<op>: GPR

Operation: C ← <op>[0], <op> ← {<op>[0], <op>[7:1]}

RR rotates the value of <op> to the right and stores the result back into <op>. The original LSB of <op> is copied into carry (C).

Flags:
C: set if the LSB of <op> (before rotating) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after rotating) is 1. Reset if not.

Example: Given: R0 = 01011010b, R1 = 10100101b

RR	R0	// No change of flag, R0 ← 00101101b
RR	R1	// C and N flags are set to ‘1’, R1 ← 11010010b

RRC — Rotate Right with Carry

Format: RRC <op>

<op>: GPR

Operation: C ← <op>[0], <op> ← {C, <op>[7:1]}

RRC rotates the value of <op> to the right and stores the result back into <op>. The original LSB of <op> is copied into carry (C), and C is copied to the MSB.

Flags:
C: set if the LSB of <op> (before rotating) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after rotating) is 1. Reset if not.

Example: Given: R2 = A5h, if C = 0

```
RRC     R2                          // R2 ← 52h, C flag is set to '1'

RR      R0
RRC     R1
```

In the second example, assuming that register pair R1:R0 is 16-bit number, then RR and RRC are used for 16-bit rotate right operation. But note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit decrement, take care of the change of Z flag.

SBC — Subtract with Carry

Format: SBC <op1>, <op2>

<op1>: GPR
<op2>: adr:8, GPR

Operation: <op1> ← <op1> + ~<op2> + C

SBC computes (<op1> - <op2>) when there is carry and (<op1> - <op2> - 1) when there is no carry.

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated.
N: set if result is negative. Reset if not.

Example:

```
SBC      R0, 80h              // If eid = 0, R0 ← R0 + ~DM[0080h] + C
                              // If eid = 1, R0 ← R0 + ~DM[IDH:80h] + C

SBC      R0, R1               // R0 ← R0 + ~R1 + C

SUB      R0, R2
SBC      R1, R3
```

In the last two instructions, assuming that register pair R1:R0 and R3:R2 are 16-bit signed or unsigned numbers. Even if the result of “ADD R0, R2” is not zero, zero (Z) flag can be set to ‘1’ if the result of “SBC R1,R3” is zero. Note that zero (Z) flag do not exactly reflect result of 16-bit operation. Therefore when programming 16-bit addition, take care of the change of Z flag.

SL — Shift Left

Format: SL <op>

<op>: GPR

Operation: C ← <op>[7], <op> ← {<op>[6:0], 0}

SL shifts <op> to the left by 1 bit. The MSB of the original <op> is copied into carry (C).

Flags: **C:** set if the MSB of <op> (before shifting) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after shifting) is 1. Reset if not.

Example: Given: R0 = 01001010b, R1 = 10100101b

```
SL          R0                          // N flag is set to ‘1’, R0 ← 10010100b

SL          R1                          // C flag is set to ‘1’, R1 ← 01001010b
```

SLA — Shift Left Arithmetic

Format: SLA <op>

<op>: GPR

Operation: C ← <op>[7], <op> ← {<op>[6:0], 0}

SLA shifts <op> to the left by 1 bit. The MSB of the original <op> is copied into carry (C).

Flags:
C: set if the MSB of <op> (before shifting) is 1. Reset if not.
Z: set if result is zero. Reset if not.
V: set if the MSB of the result is different from C. Reset if not.
N: set if the MSB of <op> (after shifting) is 1. Reset if not.

Example: Given: R0 = AAh

```
SLA     R0                  // C, V, N flags are set to ‘1’, R0 ← 54h
```

SR — Shift Right

Format:	SR <op>
	<op>: GPR
Operation:	C ← <op>[0], <op> ← {0, <op>[7:1]}
	SR shifts <op> to the right by 1 bit. The LSB of the original <op> (i.e., <op>[0]) is copied into carry (C).
Flags:	**C:** set if the LSB of <op> (before shifting) is 1. Reset if not. **Z:** set if result is zero. Reset if not. **N:** set if the MSB of <op> (after shifting) is 1. Reset if not.
Example:	Given: R0 = 01011010b, R1 = 10100101b

```
SR      R0                          // No change of flags, R0 ← 00101101b

SR      R1                          // C flag is set to ‘1’, R1 ← 01010010b
```

SRA — Shift Right Arithmetic

Format: SRA <op>

<op>: GPR

Operation: C ← <op>[0], <op> ← {<op>[7], <op>[7:1]}

SRA shifts <op> to the right by 1 bit while keeping the sign of <op>. The LSB of the original <op> (i.e., <op>[0]) is copied into carry (C).

Flags:
C: set if the LSB of <op> (before shifting) is 1. Reset if not.
Z: set if result is zero. Reset if not.
N: set if the MSB of <op> (after shifting) is 1. Reset if not.

NOTE: SRA keeps the sign bit or the MSB (<op>[7]) in its original position. If SRA is executed 'N' times, N significant bits will be set, followed by the shifted bits.

Example: Given: R0 = 10100101b

```
SRA     R0                  // C, N flags are set to '1', R0 ← 11010010b
SRA     R0                  // N flag is set to '1', R0 ← 11101001b
SRA     R0                  // C, N flags are set to '1', R0 ← 11110100b
SRA     R0                  // N flags are set to '1', R0 ← 11111010b
```

STOP — Stop Operation (pseudo instruction)

Format: STOP

Operation: The STOP instruction stops the both the CPU clock and system clock and causes the microcontroller to enter the STOP mode. In the STOP mode, the contents of the on-chip CPU registers, peripheral registers, and I/O port control and data register are retained. A reset operation or external or internal interrupts can release stop mode. The STOP instruction is a pseudo instruction. It is assembled as “SYS #0Ah”, which generates the SYSCP[7-0] signals. These signals are decoded and stop the operation.

NOTE: The next instruction of STOP instruction is executed, so please use the NOP instruction after the STOP instruction.

Example:

```
STOP
NOP
NOP
NOP
•
•
•
```

In this example, the NOP instructions provide the necessary timing delay for oscillation stabilization before the next instruction in the program sequence is executed. Refer to the timing diagrams of oscillation stabilization, as described in Figure 18-3, 18-4

SUB — Subtract

Format: SUB <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> + ~<op2> + 1

SUB adds the value of <op1> with the 2's complement of <op2> to perform subtraction on <op1> and <op2>

Flags:
C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
V: set if overflow is generated. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: IDH:IDL0 = 0150h, DM[0143h] = 26h, R0 = 52h, R1 = 14h, eid = 1

```
SUB     R0, 43h            // R0 ← R0 + ~DM[0143h] + 1 = 2Ch

SUB     R1, #16h           // R1 ← FEh, N flag is set to ‘1’

SUB     R0, R1             // R0 ← R0 + ~R1 + 1 = 3Eh

SUB     R0, @ID0+1         // R0 ← R0 + ~DM[0150h] + 1, IDL0 ← 51h
SUB     R0, @[ID0-2]       // R0 ← R0 + ~DM[014Eh] + 1, IDL0 ← 4Eh
SUB     R0, @[ID0+3]!      // R0 ← R0 + ~DM[0153h] + 1, IDL0 ← 50h
SUB     R0, @[ID0-2]!      // R0 ← R0 + ~DM[014Eh] + 1, IDL0 ← 50h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode. The example in the SBC description shows how SUB and SBC can be used in pair to subtract a 16-bit number from another.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

SWAP — Swap

Format:	SWAP <op1>, <op2>
	<op1>: GPR <op2>: SPR
Operation:	<op1> ← <op2>, <op2> ← <op1>
	SWAP swaps the values of the two operands.
Flags:	–
NOTE:	Among the SPRs, SR0 and SR1 can not be used as <op2>.
Example:	Given: IDH:IDL0 = 8023h, R0 = 56h, R1 = 01h

```
SWAP      R1, IDH                        // R1 ← 80h, IDH ← 01h
SWAP      R0, IDL0                       // R0 ← 23h, IDL0 ← 56h
```

After execution of instructions, index registers IDH:IDL0 (ID0) have address 0156h.

SYS — System

Format: SYS #imm:8

Operation: SYS generates SYSCP[7:0] and nSYSID signals.

Flags: –

NOTE: Mainly used for system peripheral interfacing.

Example:

```
SYS        #0Ah

SYS        #05h
```

In the first example, statement “SYS #0Ah” is equal to STOP instruction and second example “SYS #05h” is equal to IDLE instruction. This instruction does nothing but increase PC by one and generates SYSCP[7:0] and nSYSID signals.

TM — Test Multiple Bits

Format: TM <op>, #imm:8

<op>: GPR

Operation: TM performs the bit-wise AND operation on <op> and imm:8 and sets the flags. The content of <op> is not changed.

Flags: **Z:** set if result is zero. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: R0 = 01001101b

```
TM          R0, #00100010b          // Z flag is set to '1'
```

XOR — Exclusive OR

Format: XOR <op1>, <op2>

<op1>: GPR
<op2>: adr:8, #imm:8, GPR, @idm

Operation: <op1> ← <op1> ^ <op2>

XOR performs the bit-wise exclusive-OR operation on <op1> and <op2> and stores the result in <op1>.

Flags: **Z:** set if result is zero. Reset if not.
N: set if result is negative. Reset if not.

Example: Given: IDH:IDL0 = 8080h, DM[8043h] = 26h, R0 = 52h, R1 = 14h, eid = 1

```
XOR     R0, 43h              // R0 ← 74h

XOR     R1, #00101100b       // R1 ← 38h

XOR     R0, R1               // R0 ← 46h

XOR     R0, @ID0             // R0 ← R0 ^ DM[8080h], IDL0 ← 81h
XOR     R0, @[ID0-2]         // R0 ← R0 ^ DM[807Eh], IDL0 ← 7Eh
XOR     R0, @[ID0+3]!        // R0 ← R0 ^ DM[8083h], IDL0 ← 80h
XOR     R0, @[ID0-5]!        // R0 ← R0 ^ DM[807Bh], IDL0 ← 80h
```

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 8-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

NOTES

9 PLL (PHASE LOCKED LOOP)

OVERVIEW

S3FB42F builds clock synthesizer for system clock generation, which can operate external crystal (32.768 kHz) for reference, using internal phase-locked loop (PLL) and voltage-controlled oscillator (VCO). For real-time clock, 32.768 kHz crystal is recommended to use.

System clock circuit

The system clock circuit has the following component:

- External crystal oscillator, 32.768 kHz.
- Phase comparator, noise filter and frequency divider.
- PLL control circuit: Control register, PLLCON and PLL frequency divider data register.

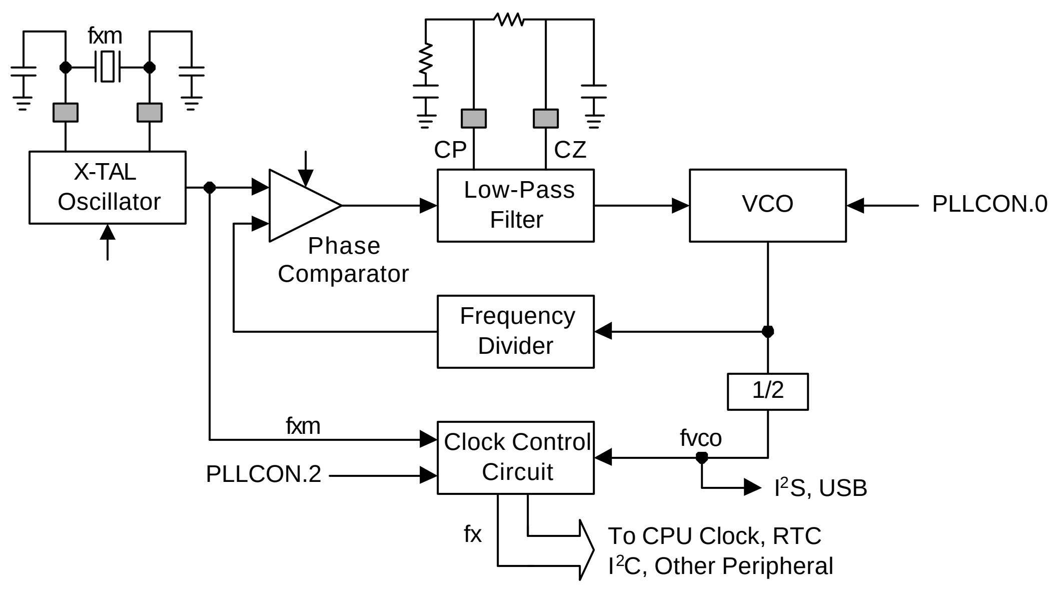


Figure 9-1. Simple Circuit Diagram

PLL REGISTER

Table 9-1. PLL Register Description

Register	Address	R/W/C	Description
PLLCON	AEH	R/W	PLL control register
PLLDATA, H	AC, ADH	R/W	PLL frequency divider data register

PLL CONTROL REGISTER (PLLCON)

Register	Address	R/W	Description	Reset Value
PLLCON	0xAE	R/W	PLL control register	00h

Bit	Bit Name	Description
[0]	Enable	This bit control the operation of PLL block. When this bit is set as "1", phase comparater, filter and VCO are activated.
[1]	f_{VCO} output	This bit enable or disable the f_{VCO} output through P8.1 pin.
[2]	Clock source selection	This bit control the selection of f_{xm} or f_{VCO} clock. When this bit is set as "1", f_{VCO}, PLL output frequency is selected as main clock oscillator.
[7:3]	–	–

PLL FREQUENCY DIVIDER DATA REGISTER (PLLDATA)

Register	Address	R/W	Description	Reset Value
PLLDATAH,L	0xAD, 0xAC	R/W	PLL frequency divider data register	–

Bit	Bit Name	Description
[15:14]	Postscaler div	Post-scaler divider value
[13:12]	Data	PLL Frequency Divider Data (bit 11 to bit 10) These bits have the bit 11 to bit10 of the frequency divider data register setting value.
[11:10]	–	Always "11"
[9:0]	Data	PLL Frequency Divider Data (bit 9 to bit 0) These bits have the bit 11 to bit10 of the frequency divider data register setting value. This frequency divider circuit divide the VCO frequency, f_{VCO}, down to reference frequency for phase comparator. The frequency divider data register setting value is like below. 0x1D60: f_{PLL}, f_{USB} = 45.158 MHz for 44.1 kHz 0x1DB6: f_{PLL}, f_{USB} = 48 MHz 0x1DDA: f_{PLL}, f_{USB} = 49.152 MHz for 48 kHz

The PLL frequency divider data is

$$N = \frac{f_{VCO}}{f_{xm}} - 2$$

where f_{VCO} is the frequency that user wants to obtain and f_{xm} is the main oscillation frequency (Typ. 32.768 KHz).

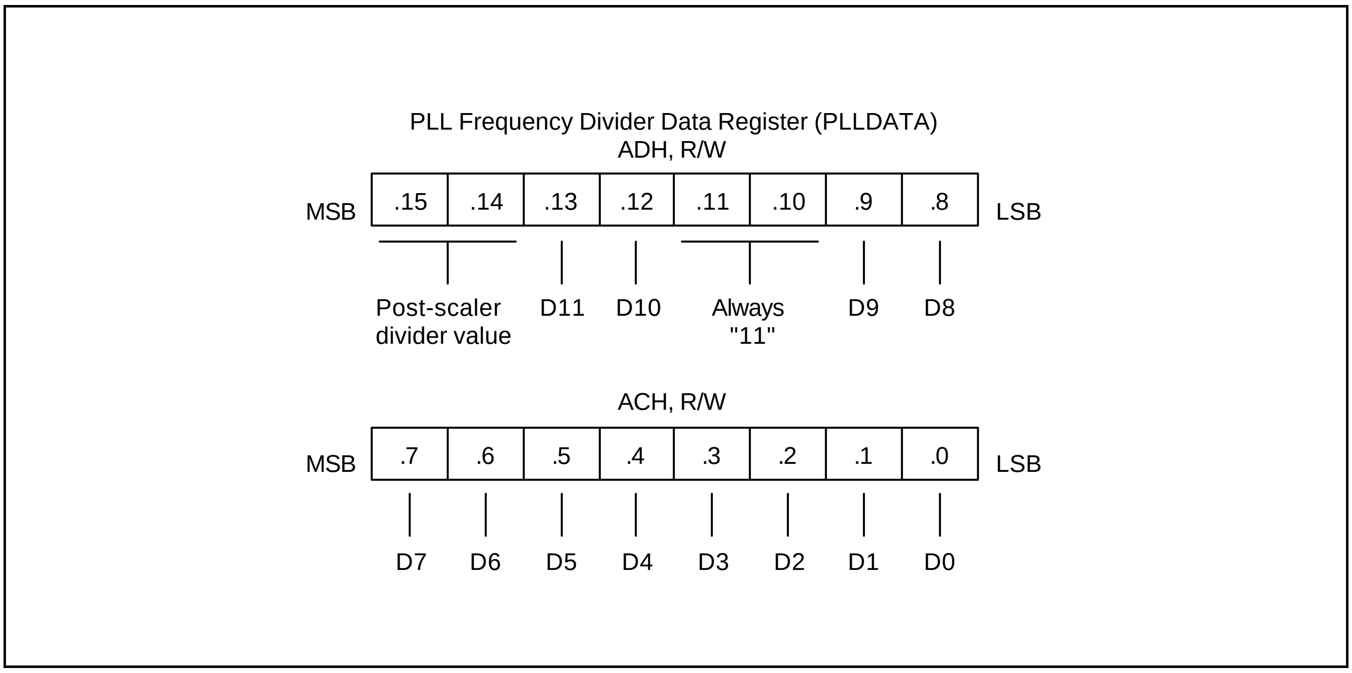


Figure 9-2. PLL Frequency Divider Data Register (PLLDATA)

SYSTEM CONTROL CIRCUIT

Table 9-2. System Control Circuit Register Description

Register	Address	R/W	Description
OSCCON	03H	R/W	Oscillator control register
PCON	02H	R/W	Power control register

OSCILLATOR CONTROL REGISTER (OSCCON)

Register	Address	R/W	Description	Reset Value
OSCCON	0x03	R/W	Oscillator control register	00h

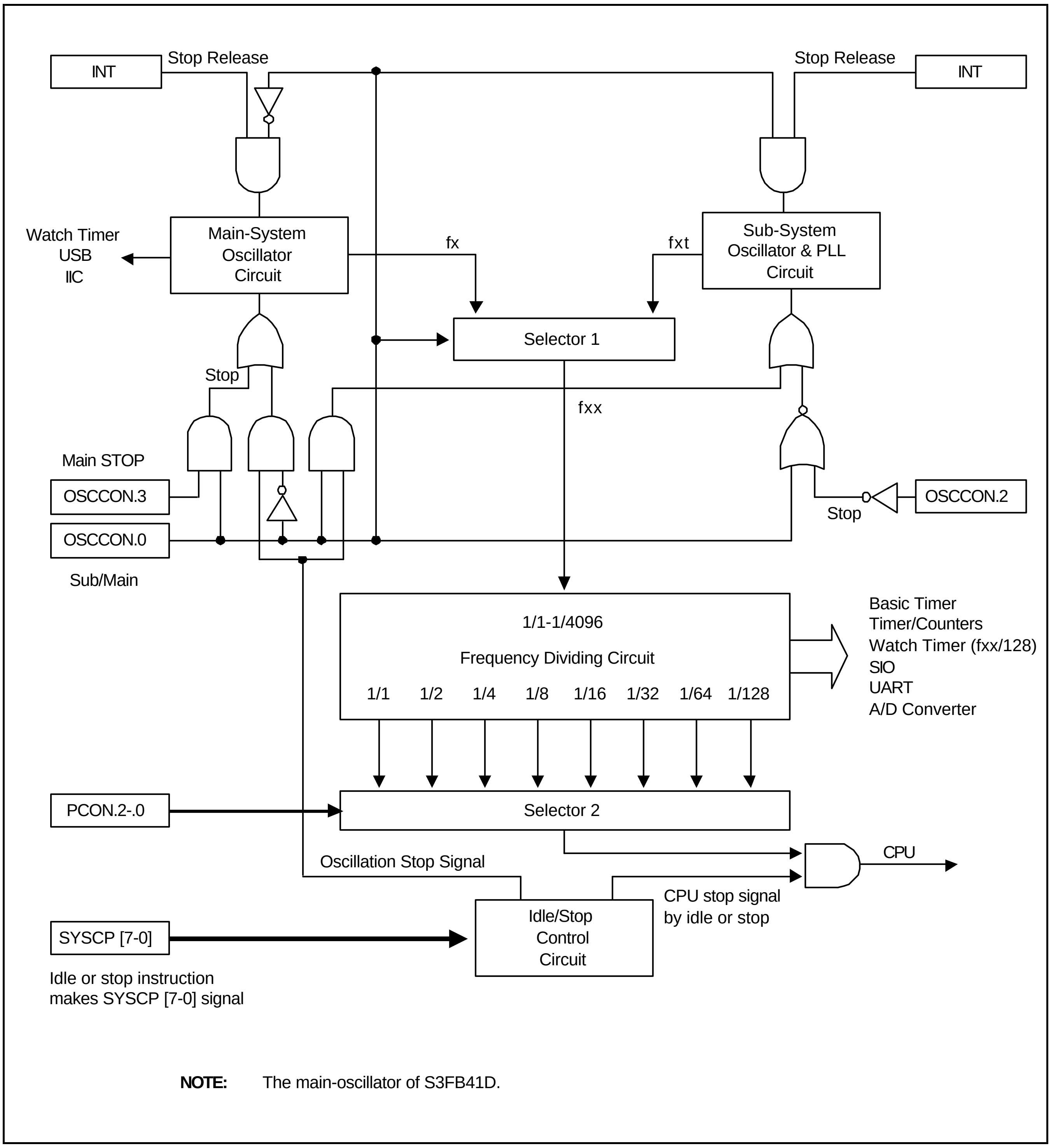
Bit	Bit Name	Description
[0]	System clock source selection	System (fxx) clock source selection bit: 0 = main system clock oscillator (fx) select (PLL system oscillator or Xout) 1 = subsystem clock oscillator (fxt or fxm) select.
[1]	–	–
[2]	Sub-clock control	Sub-clock control bit: 0 = Sub oscillator RUN. (fxt) 1 = sub oscillator STOP.
[3]	Main-clock control	Main-clock control bit: 0 = Main-clock oscillator RUN.(fxm) 1 = Main-clock oscillator STOP.
[7:4]	–	–

NOTE: After setting wanted clock selection, FMCON must be set to proper value.

POWER CONTROL REGISTER (PCON)

Register	Address	R/W	Description	Reset Value
PCON	0x02	R/W	Power control register	04h

Bit	Bit Name	Description
[7:6]	USB wait selection	USB stretch cycle selection bits: 00 = 15 cycle stretch 10 = 14 cycle stretch 01 = 13 cycle stretch 11 = 12 cycle stretch
[5]	USB High-Low selection	USB High or Low width stretch select. 0 = Low width stretch 1 = High width stretch
[4:3]	–	–
[2:0]	System clock selection	System clock selection bits: 000 = fxx/128 001 = fxx/64 010 = fxx/32 011 = fxx/16 100 = fxx/8 101 = fxx/4 110 = fxx/2 111 = fxx/1



NOTE: The main-oscillator of S3FB41D.

Figure 9-3. System Clock Circuit Diagram

10 RESET AND POWER-DOWN

OVERVIEW

During a power-on reset, the voltage at V_{DD} goes to High level and the RESET pin is forced to Low level. The RESET signal is input through a Schmitt trigger circuit where it is then synchronized with the CPU clock. This procedure brings S3FB42F into a known operating status.

For the time for CPU clock oscillation to stabilize, the RESET pin must be held to low level for a minimum time interval after the power supply comes within tolerance. For the minimum time interval, see the electrical characteristic.

In summary, the following sequence of events occurs during a reset operation:

- All interrupts are disabled.
- The watchdog function (basic timer) is enabled.
- Ports are set to input mode except port 1 which is set to output mode.
- Peripheral control and data registers are disabled and reset to their default hardware values.
- The program counter (PC) is loaded with the program reset address in the ROM, 00000H.
- When the programmed oscillation stabilization time interval has elapsed, the instruction stored in ROM location 00000H is fetched and executed.

NOTE

To program the duration of the oscillation stabilization interval, you make the appropriate settings to the basic timer control register, BTCON, before entering STOP mode. Also, if you do not want to use the basic timer watchdog function (which causes a system reset if a basic timer counter overflow occurs), you can disable it by writing '1010 0101b' to the WDTEN register.

NOTES

11 I/O PORTS

PORT DATA REGISTERS

All thirteen port data registers have the identical structure shown in Figure 11-1 below.:

Table 11-1. Port Data Register Summary

Register Name	Mnemonic	Address	Reset Value	R/W
Port 0 Data Register	P0	10h	00h	R/W
Port 1 Data Register	P1	11h	00h	R/W
Port 2 Data Register	P2	12h	00h	R/W
Port 3 Data Register	P3	13h	00h	R/W
Port 4 Data Register	P4	14h	00h	R/W
Port 5 Data Register	P5	15h	xxh	R
Port 6 Data Register	P6	16h	00h	R/W
Port 7 Data Register	P7	17h	00h	R/W
Port 8 Data Register	P8	18h	00h	R/W
Port 9 Data Register	P9	19h	00h	R/W

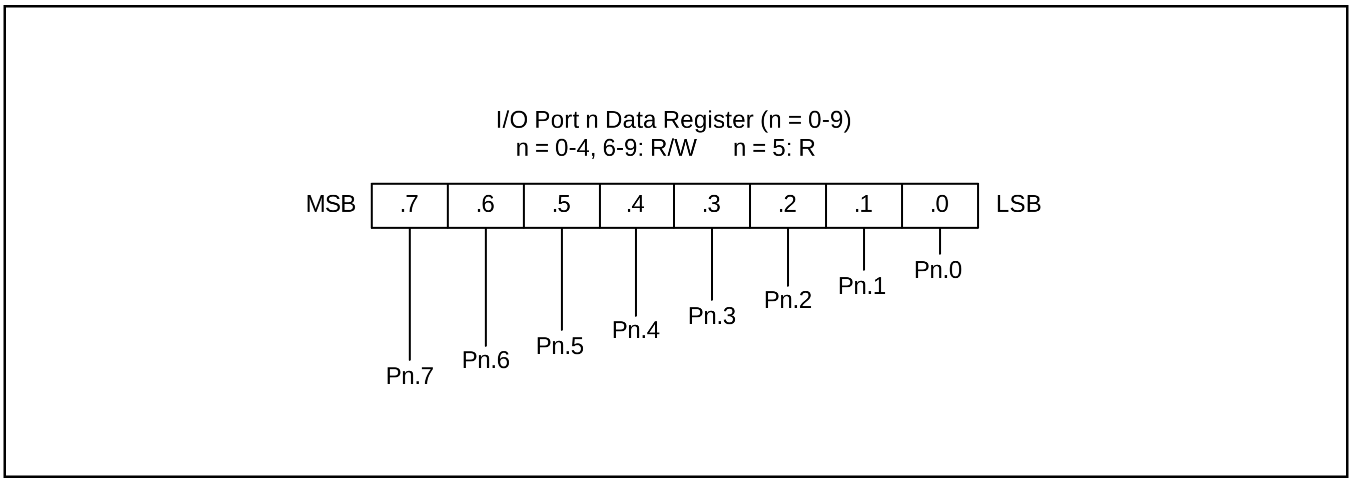

I/O Port n Data Register (n = 0-9)
n = 0-4, 6-9: R/W n = 5: R
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Pn.7
Pn.6
Pn.5
Pn.4
Pn.3
Pn.2
Pn.1
Pn.0

Figure 11-1. Port Data Register Structure

PORT CONTROL REGISTERS

PORT 0 CONTROL REGISTER (P0CON)

Register	Address	R/W	Description	Reset Value
P0CON	0x20	R/W	Port 0 control register	00h

Bit	Setting	Description
[7:0]	0 or 1	**Port 0 Setting** 0: Normal C-MOS input mode 1: Normal C-MOS output mode

NOTE: The parallel port control (PPCONL.1) register can assign port 0 to parallel printer port's data bus mode, which is not effected by P0CON setting.

PORT 1 CONTROL REGISTER (P1CON)

Register	Address	R/W	Description	Reset Value
P1CON	0x21	R/W	Port 1 control register	00h

Bit	Setting	Description
[4:0]	0 or 1	**P1.0, P1.1, P1.2, P1.3 or P1.4 Setting** 0: Normal C-MOS input mode 1: Normal C-MOS output mode

NOTE: The parallel port control (PPCONL.1) register can assign port 1 to parallel printer port's control bus mode, which is not effected by P1CON.0-5 setting.

PORT 2 CONTROL LOW REGISTER (P2CONL)

Register	Address	R/W	Description	Reset Value
P2CONL	0x22	R/W	Port 2 control low register	00h

Bit	Setting	Description
[0]	0 or 1	**P2.0 Setting** 0: Schmitt trigger input mode or TACLK input mode 1: Normal C-MOS output mode
[1]	0 or 1	**P2.1 Setting** 0: Schmitt trigger input mode or TBCLK input mode 1: Normal C-MOS output mode
[3:2]	0 or 1	**P2.2 Setting** 00: Schmitt trigger input mode 01: Normal C-MOS output mode 10: Serial data output (SO) for SIO (SP1) (C-MOS output mode) 11: Serial data output (SO) for SIO (SP1) (N-channel open drain output mode)
[5:4]	0 or 1	**P2.3 Setting** 00: Schmitt trigger input mode 01: Normal C-MOS output mode 10: Serial data output (SCK) for SIO (SP1) (C-MOS output mode) 11: Serial data output (SCK) for SIO (SP1) (N-channel open drain output mode)
[6]	0 or 1	**P2.4 Setting** 0: Schmitt trigger input mode or Rx input mode in UART 1: Normal C-MOS output mode
[7]	–	–

PORT 2 CONTROL HIGH REGISTER (P2CONH)

Register	Address	R/W	Description	Reset Value
P2CONH	0x23	R/W	Port 2 control high register	30h

Bit	Setting	Description
[1:0]	0 or 1	**P2.5 Setting** 00: Schmitt trigger input mode 01: Normal C-MOS output mode 10: Tx output mode in UART 11: Invalid
[2]	0 or 1	**P2.6 Setting** 0: Schmitt trigger input mode 1: Normal C-MOS output mode
[3]	0 or 1	**P2.7 Setting** 0: Schmitt trigger input mode 1: Normal C-MOS output mode
[4]	0 or 1	**P6.6 pull-up resistor Setting** 0: Disable Pull-up resistor 1: Enable Pull-up resistor (Reset value)
[5]	0 or 1	**P6.7 pull-up resistor Setting** 0: Disable Pull-up resistor 1: Enable Pull-up resistor (Reset value)

NOTE: The pull-up resistors of P6.6 and P6.7can be assigned by P2CONH.4, 5.

PORT 3 CONTROL LOW REGISTER (P3CONL)

Register	Address	R/W	Description	Reset Value
P3CONL	0x24	R/W	Port 3 control low register	00h

Bit	Setting	Description
[1:0]	0 or 1	**P3.0 Setting** 00: Schmitt trigger input mode, serial data input (SI) for SIO (SPI) 01: Normal C-MOS output mode 10: N-Ch Open drain output mode 11: N-Ch Open drain output mode
[3:2]	0 or 1	**P3.1 Setting** 00: Schmitt trigger input mode 01: Normal C-MOS output mode 10: Serial data output (SO) for SIO(SP1) (CMOS output mode) 11: Serial data output (SO) for SIO(SP1) (N-channel open-drain output mode)
[5:4]	0 or 1	**P3.2 Setting** 00: Schmitt trigger input mode, serial clock input mode (SCK) for SIO (SPI) 01: Normal C-MOS output mode 10: Serial data output (SCK) for SIO(SP1) (CMOS output mode) 11: Serial data output (SCK) for SIO(SP1) (N-channel open-drain output mode)
[7:6]	0 or 1	**P3.3 Setting** 00: Schmitt trigger input mode 01: Normal C-MOS output mode 10: Serial clock port (SCL) for I^2C (schmitt trigger input or output mode) 11: Serial clock port (SCL) for I^2C (schmitt trigger input or N-ch open drain output mode)

PORT 3 CONTROL HIGH REGISTER (P3CONH)

Register	Address	R/W	Description	Reset Value
P3CONH	0x25	R/W	Port 3 control high register	00h

Bit	Setting	Description
[1:0]	0 or 1	**P3.4 Setting** 00: Schmitt trigger input mode 01: Normal C-MOS output mode 10: Serial data port (SDA) for I^2C (Schmitt trigger input/ C-MOS output mode) 11: Serial data port (SDA) for I^2C (Schmitt trigger input and N-ch Open drain output mode)
[3:2]	0 or 1	**P3.5 Setting** 00: Schmitt trigger input mode 01: Normal C-MOS output mode 10: N-Ch Open drain output mode 11: N-Ch Open drain output mode
[5:4]	0 or 1	**P3.6 Setting** 00: Schmitt trigger input mode 01: Normal C-MOS output mode 10: N-Ch Open drain output mode 11: N-Ch Open drain output mode
[7:6]	0 or 1	**P3.7 Setting** 00: Schmitt trigger input mode 01: Normal C-MOS output mode 10: N-Ch Open drain output mode 11: N-Ch Open drain output mode

PORT 3 PULL-UP REGISTER (P3PUR)

Register	Address	R/W	Description	Reset Value
P3PUR	0x26	R/W	Port 3 pull-up resistor enable register	00h

Bit	Setting	Description
[7:0]	0 or 1	P3.0-3.7 Pull-up Resistor Setting 0: Disable pull-up resistor 1: Enable pull-up resistor

PORT 4 CONTROL REGISTER (P4CON)

Register	Address	R/W	Description	Reset Value
P4CON	0x30	R/W	Port 4 control register	00h

Bit	Setting	Description
[1:0]	0 or 1	**P4.0 Setting** 00: Schmitt trigger input mode or external interrupt 9 input 01: Schmitt trigger input mode or external interrupt 9 input with pull-up resistor 10: Normal C-MOS output mode 11: Normal C-MOS output mode
[3:2]	0 or 1	**P4.1 Setting** 00: Schmitt trigger input mode or external interrupt 8 input 01: Schmitt trigger input mode or external interrupt 8 input with pull-up resistor 10: Normal C-MOS output mode 11: Normal C-MOS output mode
[5:4]	0 or 1	**P4.2 Setting** 00: Schmitt trigger input mode 01: Schmitt trigger input mode with pull-up resistor 10: Normal C-MOS output mode 11: Normal C-MOS output mode

PORT 4 INTERRUPT CONTROL REGISTER (P4INTCON)

Register	Address	R/W	Description	Reset Value
P4INTCON	0x31	R/W	Port 4 interrupt control register	00h

Bit	Setting	Description
[1:0]	0 or 1	**Setting the external interrupt enable of P4.1, P4.0 (INT8-9)** 0: Disable external interrupt 1: Enable external interrupt

PORT 4 INTERRUPT MODE REGISTER (P4INTMOD)

Register	Address	R/W	Description	Reset Value
P4INTMOD	0x32	R/W	Port 4 interrupt mode register	00h

Bit	Setting	Description
[1:0] [3:2]	0 or 1	**Setting the external interrupt mode of P4.0 (INT9) and P4.1 (INT8)** 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: High level interrupt enable 11: Low level interrupt enable

PORT 5 CONTROL REGISTER (P5CON)

Register	Address	R/W	Description	Reset Value
P5CON	0x28	R/W	Port 5 control register	00h

Bit	Setting	Description
[0]	0 or 1	**P5.0 Setting** 0: Normal C-MOS input mode or external interrupt 0 input 1: ADC0 input mode
[1]	0 or 1	**P5.1 Setting** 0: Normal C-MOS input mode or external interrupt 1 input 1: ADC1 input mode
[2]	0 or 1	**P5.2 Setting** 0: Normal C-MOS input mode or external interrupt 2 input 1: ADC2 input mode
[3]	0 or 1	**P5.3 Setting** 0: Normal C-MOS input mode or external interrupt 3 input 1: ADC3 input mode
[4]	0 or 1	**P5.4 Setting** 0: Normal C-MOS input mode or external interrupt 4 input 1: ADC4 input mode
[5]	0 or 1	**P5.5 Setting** 0: Normal C-MOS input mode or external interrupt 5 input 1: ADC5 input mode

PORT 5 PULL-UP REGISTER (P5PUR)

Register	Address	R/W	Description	Reset Value
P5PUR	0x29	R/W	Port 5 pull-up resistor enable register	00h

Bit	Setting	Description
[5:0]	0 or 1	**P5.0-5.5 Pull-up Resistor Setting** 0: Disable pull-up resistor 1: Enable pull-up resistor

PORT 5 INTERRUPT CONTROL REGISTER (P5INTCON)

Register	Address	R/W	Description	Reset Value
P5INTCON	0x2A	R/W	Port 5 interrupt control register	00h

Bit	Setting	Description
[5:0]	0 or 1	**Setting the external interrupt enable of P5.0-P5.5 (INT0-5)** 0: Disable External Interrupt 1: Enable External Interrupt

PORT 5 EXTERNAL INTERRUPT PENDING REGISTER (EINTPND)

Register	Address	R/W	Description	Reset Value
EINTPND	0x2D	R/W	Port 5 external interrupt pending register	00h

Bit	Setting	Description
[5:0]	0 or 1	**Setting the external interrupt pending bit of P5.0-P5.5 (INT0-5)** 0: Interrupt is not pending when read. (*When write, pending bit is clear*) 1: Interrupt is pending when read. (*When write, pending bit is not effected*)

PORT 5 INTERRUPT MODE LOW REGISTER (P5INTMODL)

Register	Address	R/W	Description	Reset Value
P5INTMODL	0x2B	R/W	Port 5 interrupt mode low register	00h

Bit	Setting	Description
[1:0] [3:2] [5:4] [7:6]	0 or 1	**Setting the external interrupt mode of P5.0(INT0)/P5.1(INT1)/P5.2(INT2)/P5.3(INT3)** 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: Falling or rising edge interrupt enable 11: Invalid value

PORT 5 INTERRUPT MODE HIGH REGISTER (P5INTMODH)

Register	Address	R/W	Description	Reset Value
P5INTMODH	0x2C	R/W	Port 5 interrupt mode high register	00h

Bit	Setting	Description
[1:0] [3:2]	0 or 1	**Setting the external interrupt mode of P5.4(INT4)/P5.5(INT5)** 00: Falling edge interrupt enable 01: Rising edge interrupt enable 10: Falling or rising edge interrupt enable 11: Invalid value

PORT 6 CONTROL REGISTER (P6CON)

Register	Address	R/W	Description	Reset Value
P6CON	0x34	R/W	Port 6 control register	00h

Bit	Setting	Description
[0]	0 or 1	**P6.0 Setting** 0: Normal C-MOS input mode 1: Normal C-MOS output mode; Chip enable 1 (CE1) for SmartMedia
[1]	0 or 1	**P6.1 Setting** 0: Normal C-MOS input mode 1: Normal C-MOS output mode; Chip enable 0 (CE0) for SmartMedia
[2]	0 or 1	**P6.2 Setting** 0: Normal C-MOS input mode 1: Normal C-MOS output mode; Command latch enable (CLE) for SmartMedia
[3]	0 or 1	**P6.3 Setting** 0: Normal C-MOS input mode 1: Normal C-MOS output mode; Address latch enable (ALE) for SmartMedia
[4]	0 or 1	**P6.4 Setting** 0: Normal C-MOS input mode; Ready/Busy (R/B) for SmartMedia 1: Normal C-MOS output mode
[5]	0 or 1	**P6.5 Setting** 0: Normal C-MOS input mode 1: Normal C-MOS output mode; Write protect (WP) for SmartMedia
[6]	0 or 1	**P6.6 Setting** 0: Normal C-MOS input mode 1: Normal C-MOS output mode; Read enable (RE) for SmartMedia
[7]	0 or 1	**P6.7 Setting** 0: Normal C-MOS input mode 1: Normal C-MOS output mode; Write enable (WE) for SmartMedia

NOTES:
1. When the SmartMedia control (SMCON) register is enabled, the access of port 7 generate the read or write strobe signal to the SmartMedia memory. However, other pins for SmartMeida interface should set interface condition and generate interface signal by CPU instruction. This provide the customer with the high speed memory access time, small chip size and small power consumption together.
2. The pull-up resistors of P6.6 and P6.7can be assigned by P2CONH.4, 5.

PORT 2 CONTROL HIGH REGISTER OR P6PUR (P2CONH)

Register	Address	R/W	Description	Reset Value
P2CONH	0x23	R/W	Port 2 control high register	30h

Bit	Setting	Description
[3:0]	0 or 1	**P2.5, 6,7 Setting (release see the P2CONH register)**
[4]	0 or 1	**P6.6 Pull-up Resistor Setting** 0: Disable Pull-up resistor 1: Enable Pull-up resistor (Reset value)
[5]	0 or 1	**P6.7 Pull-up Resistor Setting** 0: Disable Pull-up resistor 1: Enable Pull-up resistor (Reset value)

PORT 7 CONTROL REGISTER (P7CON)

Register	Address	R/W	Description	Reset Value
P7CON	0x35	R/W	Port 7 control register	00h

Bit	Setting	Description
[7:0]	0 or 1	**Port 7 Setting** 0: Normal C-MOS input mode 1: Normal C-MOS output mode

NOTE: When the SmartMedia control (SMCON) register is enabled, the read or write operation for port 7 activate the ECC block. The ECC block capture the data on port 7 access and execute ECC operation.

PORT 8 CONTROL REGISTER (P8CON)

Register	Address	R/W	Description	Reset Value
P8CON	0x36	R/W	Port 8 control register	00h

Bit	Setting	Description
[0]	0 or 1	**P8.0 Setting** 0: Schmitt trigger level input mode 1: Normal C-MOS output mode
[1]	0 or 1	**P8.1 Setting** 0: Schmitt trigger level input mode 1: Normal C-MOS output mode
[2]	0 or 1	**P8.2 Setting** 0: Schmitt trigger level input mode 1: Normal C-MOS output mode
[3]	0 or 1	**P8.3 Setting** 0: Schmitt trigger level input mode 1: Normal C-MOS output mode

NOTE: The parallel port control (PPCONH.1) register can assign port 8 to parallel printer port's control bus mode, which is not effected by P8CON.0-3 setting.

PORT 9 CONTROL REGISTER (P9CON)

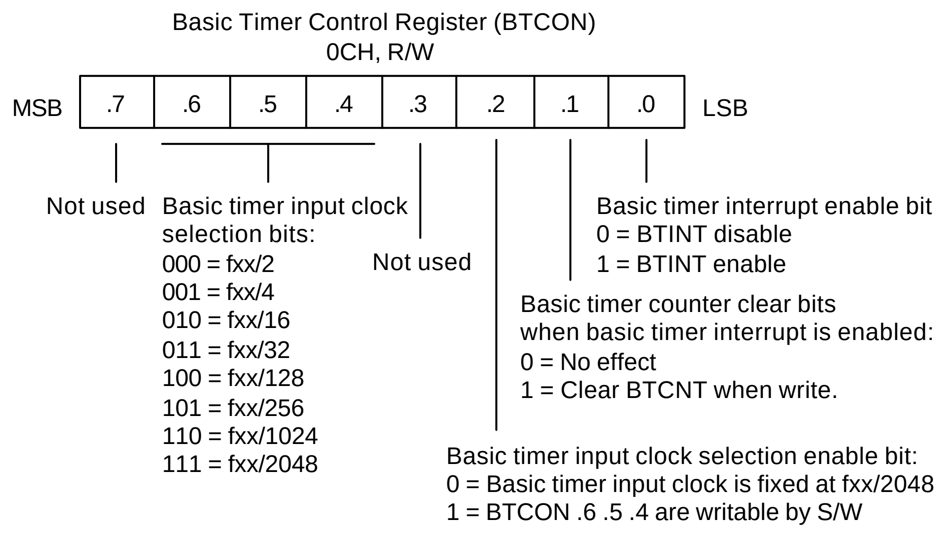
Register	Address	R/W	Description	Reset Value
P9CON	0x37	R/W	Port 9 control register	00h

Bit	Setting	Description
[0]	0 or 1	**P9.0 Setting** 0: Schmitt trigger input mode, word selection input mode (WS0) 1: Normal C-MOS output mode, word selection output mode (WS0)
[1]	0 or 1	**P9.1 Setting** 0: Schmitt trigger input mode, bit shift clock input mode (SCLK0) 1: Normal C-MOS output mode, bit shift clock output mode (SCLK0)
[2]	0 or 1	**P9.2 Setting** 0: Schmitt trigger input mode, shift data input mode (SD0) 1: Normal C-MOS output mode, shift data output mode (SD0)
[3]	0 or 1	**P9.3 Setting** 0: Schmitt trigger input mode, word selection input mode (WS1) 1: Normal C-MOS output mode, word selection output mode (WS1)
[4]	0 or 1	**P9.4 Setting** 0: Schmitt trigger input mode, bit shift clock input mode (SCLK1) 1: Normal C-MOS output mode, bit shift clock output mode (SCLK1)
[5]	0 or 1	**P9.5 Setting** 0: Schmitt trigger input mode, shift data input mode (SD1) 1: Normal C-MOS output mode, shift data output mode (SD1)
[6]	0 or 1	**P9.6 Setting** 0: Schmitt trigger input mode 1: Normal C-MOS output mode, Master clock output mode (MCLK) for IIS0

NOTE: The direction of WS and SCLK port is decided by IISCON.3, *MASTER,* where is the output mode in the master mode or the input mode in the slave mode. Also, the direction of SD port is decided by IISCON.2, *TRANS*, where is the output mode in the transmitter mode or the input mode in the receive mode.

12 BASIC TIMER

OVERVIEW



Basic Timer Control Register (BTCON)
0CH, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Not used
Basic timer input clock selection bits:
000 = fxx/2
001 = fxx/4
010 = fxx/16
011 = fxx/32
100 = fxx/128
101 = fxx/256
110 = fxx/1024
111 = fxx/2048
Not used
Basic timer interrupt enable bit
0 = BTINT disable
1 = BTINT enable
Basic timer counter clear bits when basic timer interrupt is enabled:
0 = No effect
1 = Clear BTCNT when write.
Basic timer input clock selection enable bit:
0 = Basic timer input clock is fixed at fxx/2048
1 = BTCON .6 .5 .4 are writable by S/W

NOTE: After the reset, BTCON.2 is set to "0" and basic timer input clock is fixed at fxx/2048. If you want to change the basic timer input clock, you should set the BTCON.2 to "1", and then the BTCON .6 .5 .4 are writable by S/W.

Figure 12-1. Basic Timer Control Register (BTCON)

WATCHDOG TIMER

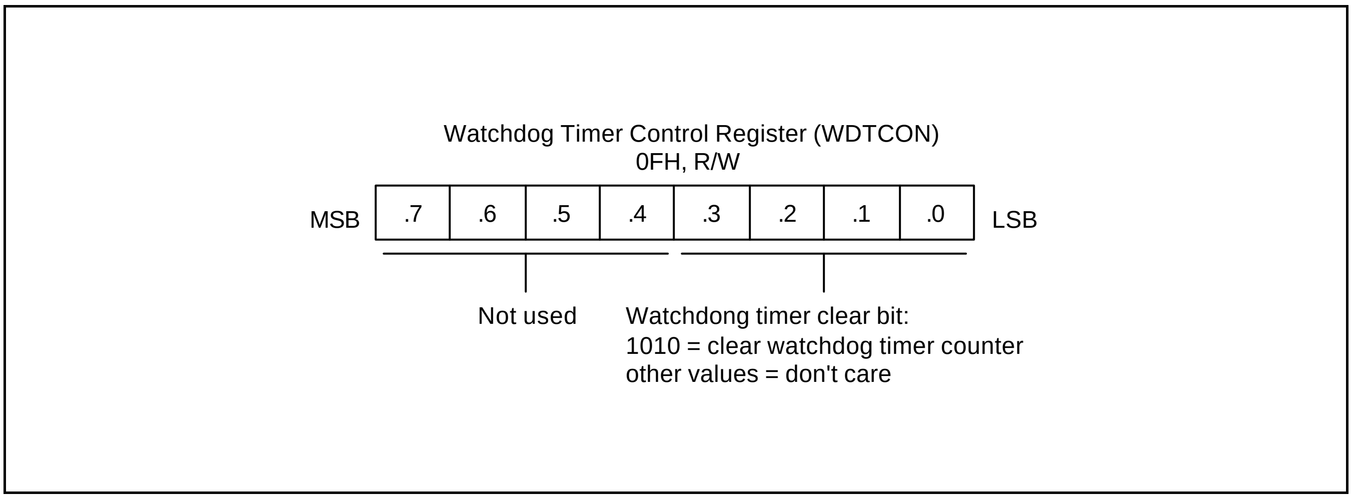

Watchdog Timer Control Register (WDTCON)
0FH, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Not used
Watchdong timer clear bit:
1010 = clear watchdog timer counter
other values = don't care

Figure 12-2. Watchdog Timer Control Register (WDTCON)

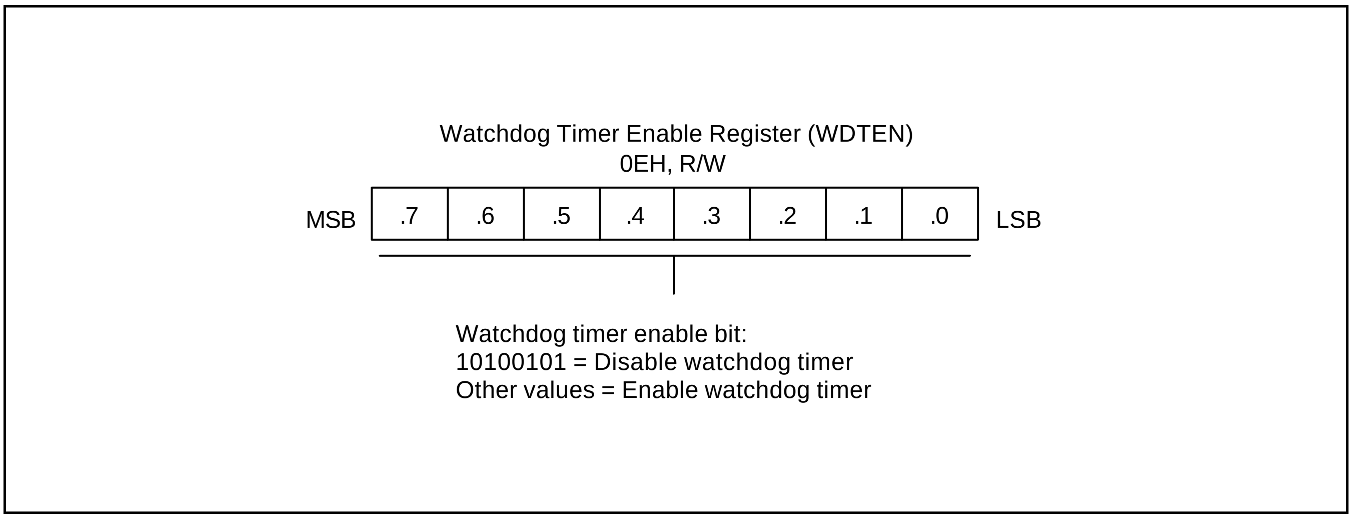

Watchdog Timer Enable Register (WDTEN)
0EH, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Watchdog timer enable bit:
10100101 = Disable watchdog timer
Other values = Enable watchdog timer

Figure 12-3. Watchdog Timer Enable Register (WDTEN)

BLOCK DIAGRAM

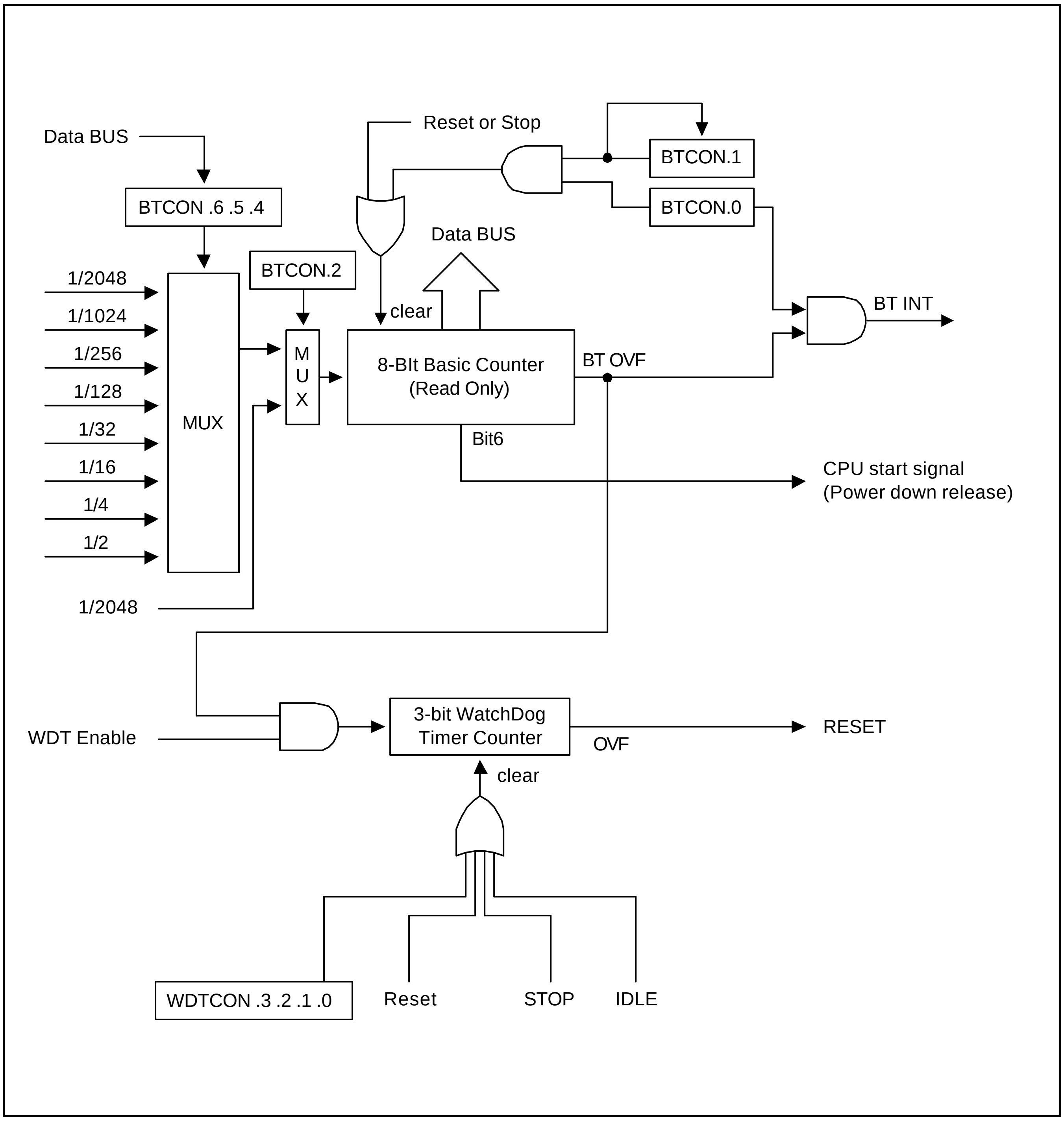


Figure 12-4. Basic Timer & Watchdog Timer Functional Block Diagram

NOTES

13 REAL TIMER (WATCH TIMER)

OVERVIEW

Real time clock functions include real-time and watch-time measurement and interval timing for the system clock. To start real time clock operation, set bit 1 of the real time clock(watch timer) control register, WTCON.1 to "1". After the real time clock starts and elapses a time, the real time clock interrupt is automatically set to "1", and interrupt requests commence in 3.91ms, or, 0.5 and 1-second intervals.

The watch timer can generate a steady 0.5 kHz, 1 kHz, 2 khz or 4 kHz signal to the BUZZER output. By setting WTCON.3 and WTCON.2 to "11b", the real time clock will function in high-speed mode, generating an interrupt every 3.91 ms. High-speed mode is useful for timing events for program debugging sequences.

— Real-Time and Watch-Time Measurement

— Using a Main Oscillator or Sub Oscillator Clock Source

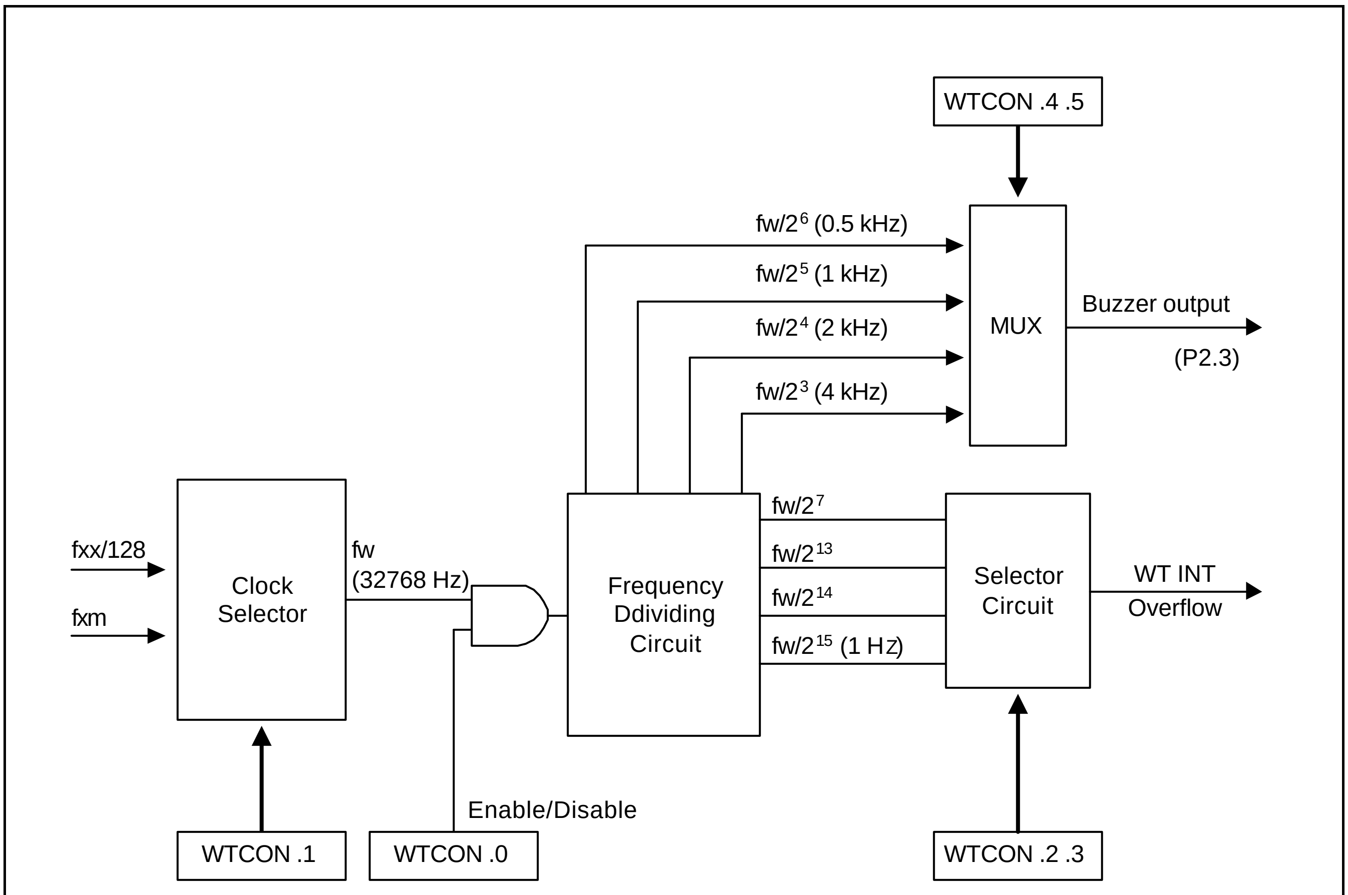
— Buzzer Output Frequency Generator

— Timing Tests in High-Speed Mode

Table 13-1. Watch Timer Control Register (WTCON): 8-Bit R/W

Bit Name	Values		Function	Address
WTCON.7	–		Not used	4CH
WTCON.6	–		Not used	
WTCON .5-.4	0	0	0.5 kHz buzzer (BUZ) signal output (when WTCON.1 = "1")	
	0	1	1 kHz buzzer (BUZ) signal output (when WTCON.1 = "1")	
	1	0	2 kHz buzzer (BUZ) signal output (when WTCON.1 = "1")	
	1	1	4 kHz buzzer (BUZ) signal output (when WTCON.1 = "1")	
WTCON .3-.2	0	0	Set watch timer interrupt to 1S (when WTCON.1 = "1")	
	0	1	Set watch timer interrupt to 0.5S (when WTCON.1 = "1")	
	1	0	Set watch timer interrupt to 0.25S (when WTCON.1 = "1")	
	1	1	Set watch timer interrupt to 3.91mS (when WTCON.1 = "1")	
WTCON.1	0		Select fxx/128 as the watch timer clock	
	1		Select X_{OUT} as the watch timer clock	
WTCON.0	0		Stop watch timer counter; clear frequency dividing circuits	
	1		Run watch timer counter	

WATCH TIMER CIRCUIT DIAGRAM



fx = Main system clock (f VCO or fxm)
fxx = Selected system clock (fx or fxt)
fxt = Sub system clock
fxm = Main osillator output
fvco = PLL Vco output

Figure 13-1. Watch Timer Circuit Diagram

14 16-BIT TIMER (8-BIT TIMER A & B)

OVERVIEW

This 16-bit timer has two modes. One is 16-bit timer mode and the other is two 8-bit timer mode. When Bit 2 of TBCON is "1", it operates with the 16-bit timer. When it is "0", it operates with two 8-bit timers. When it operates with the 16-bit timer, the TBCNT's clock source can be selected by setting TBCON.3. If TBCON.3 is "0", the timer A's overflow would be TBCNT's clock source. If it is "1", the timer A's interval out would be TBCNT's clock source. The timer clock source can be selected by S/W.

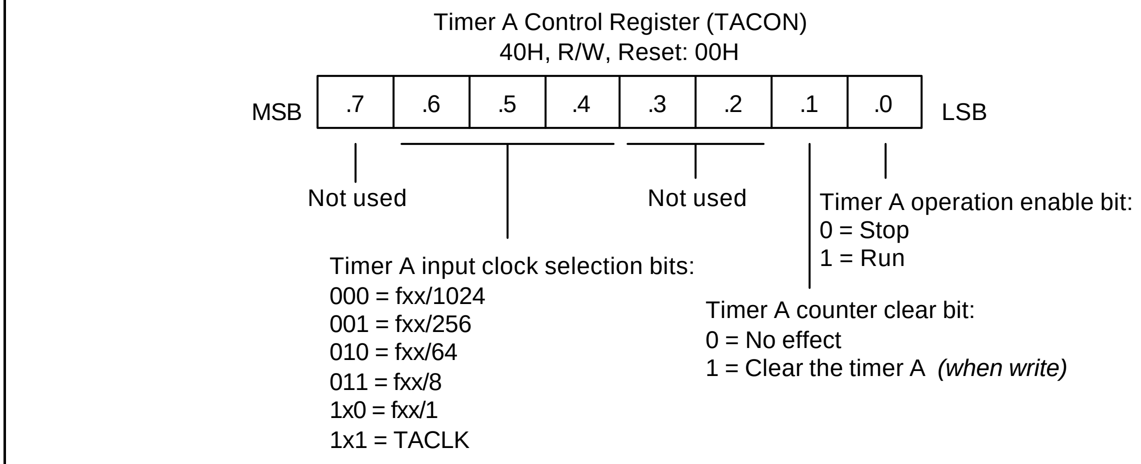
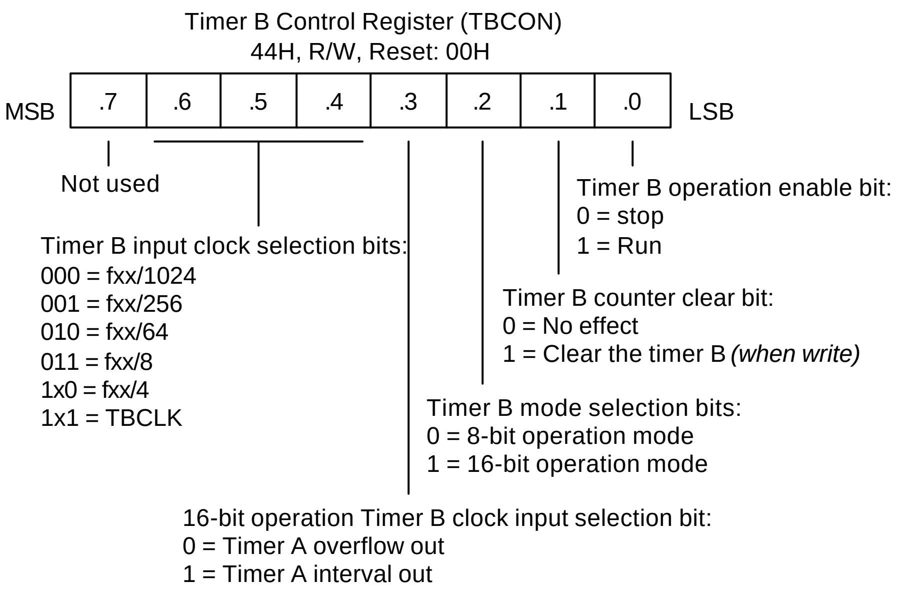


Figure 14-1. Timer A Control Register (TACON)


Timer B Control Register (TBCON)
44H, R/W, Reset: 00H
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Not used
Timer B input clock selection bits:
000 = fxx/1024
001 = fxx/256
010 = fxx/64
011 = fxx/8
1x0 = fxx/4
1x1 = TBCLK
Timer B operation enable bit:
0 = stop
1 = Run
Timer B counter clear bit:
0 = No effect
1 = Clear the timer B *(when write)*
Timer B mode selection bits:
0 = 8-bit operation mode
1 = 16-bit operation mode
16-bit operation Timer B clock input selection bit:
0 = Timer A overflow out
1 = Timer A interval out

NOTE: At 16-bit operation mode 16-bit counter clock input is selected by TACON .6, .5, .4

Figure 14-2. Timer B Control Register (TBCON)

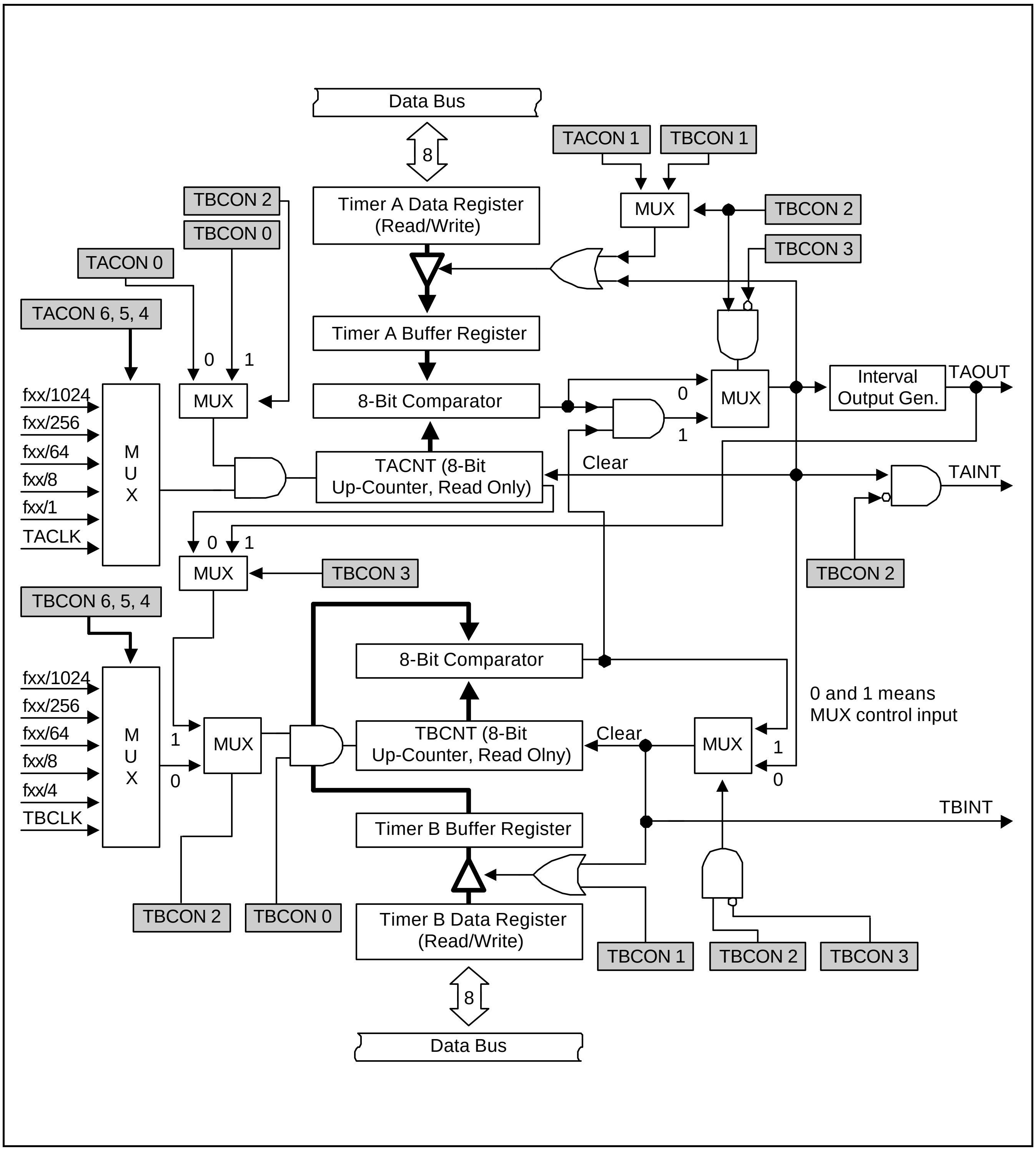
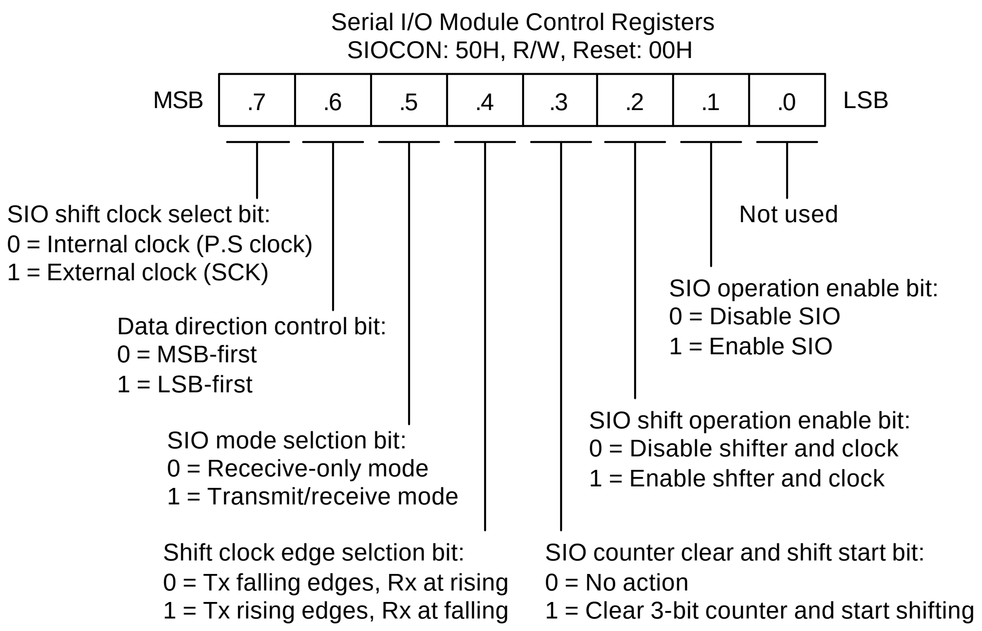


Figure 14-3. Timer A, B Function Block Diagram

NOTES

15 SERIAL I/O INTERFACE

OVERVIEW


Serial I/O Module Control Registers
SIOCON: 50H, R/W, Reset: 00H
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
SIO shift clock select bit:
0 = Internal clock (P.S clock)
1 = External clock (SCK)
Not used
SIO operation enable bit:
0 = Disable SIO
1 = Enable SIO
Data direction control bit:
0 = MSB-first
1 = LSB-first
SIO shift operation enable bit:
0 = Disable shifter and clock
1 = Enable shfter and clock
SIO mode selction bit:
0 = Rececive-only mode
1 = Transmit/receive mode
Shift clock edge selction bit:
0 = Tx falling edges, Rx at rising
1 = Tx rising edges, Rx at falling
SIO counter clear and shift start bit:
0 = No action
1 = Clear 3-bit counter and start shifting

NOTES:
1. SIOCON.2 and SIOCON.3 should not be set simultaneously. If it is done, the data can be lost.
2. SIOCON.3 must be set separately when starting communication.

Figure 15-1. Serial I/O Module Control Registers (SIOCON)

NOTES:

1. Tx: 1) Set the bit 1 and 2 of SIOCON in advance.
   2) Push data into SIODATA.
   3) Set the bit 3 of SIOCON to start the transmission.

2. Rx: 1) Set the bit 1 and 2 of SIOCON in advance.
   2) Set the bit 3 of SIOCON to receive the data.
   3) Read the data from SIODATA.

SIO PRE-SCALER REGISTER (SIOPS)

The control register for serial I/O interface module, SIOPS, is located at 49H. The value stored in the SIO pre-scaler registers, SIOPS, lets you determine the SIO clock rate (baud rate) as follows:

Baud rate = Input clock (fxx/2) / (Pre-scaler value + 1), or, SCLK input clock

where the input clock is fxx.

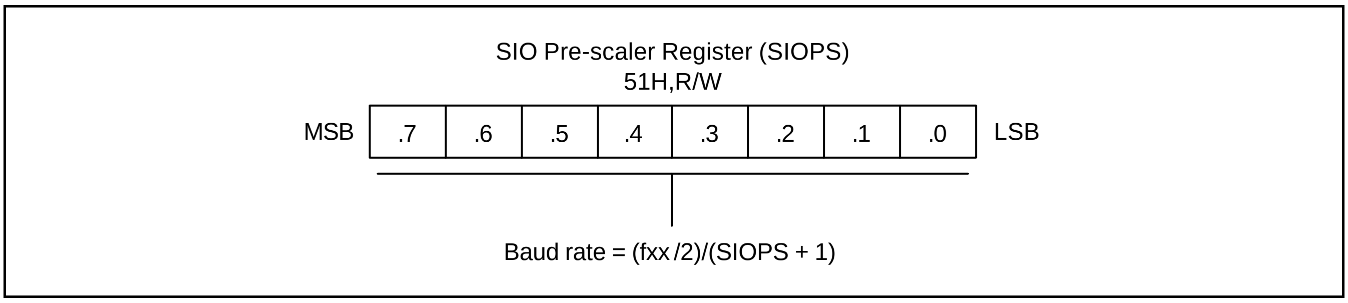

SIO Pre-scaler Register (SIOPS)
51H,R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Baud rate = (fxx /2)/(SIOPS + 1)

Figure 15-2. SIO Pre-scaler Register (SIOPS)

BLOCK DIAGRAM

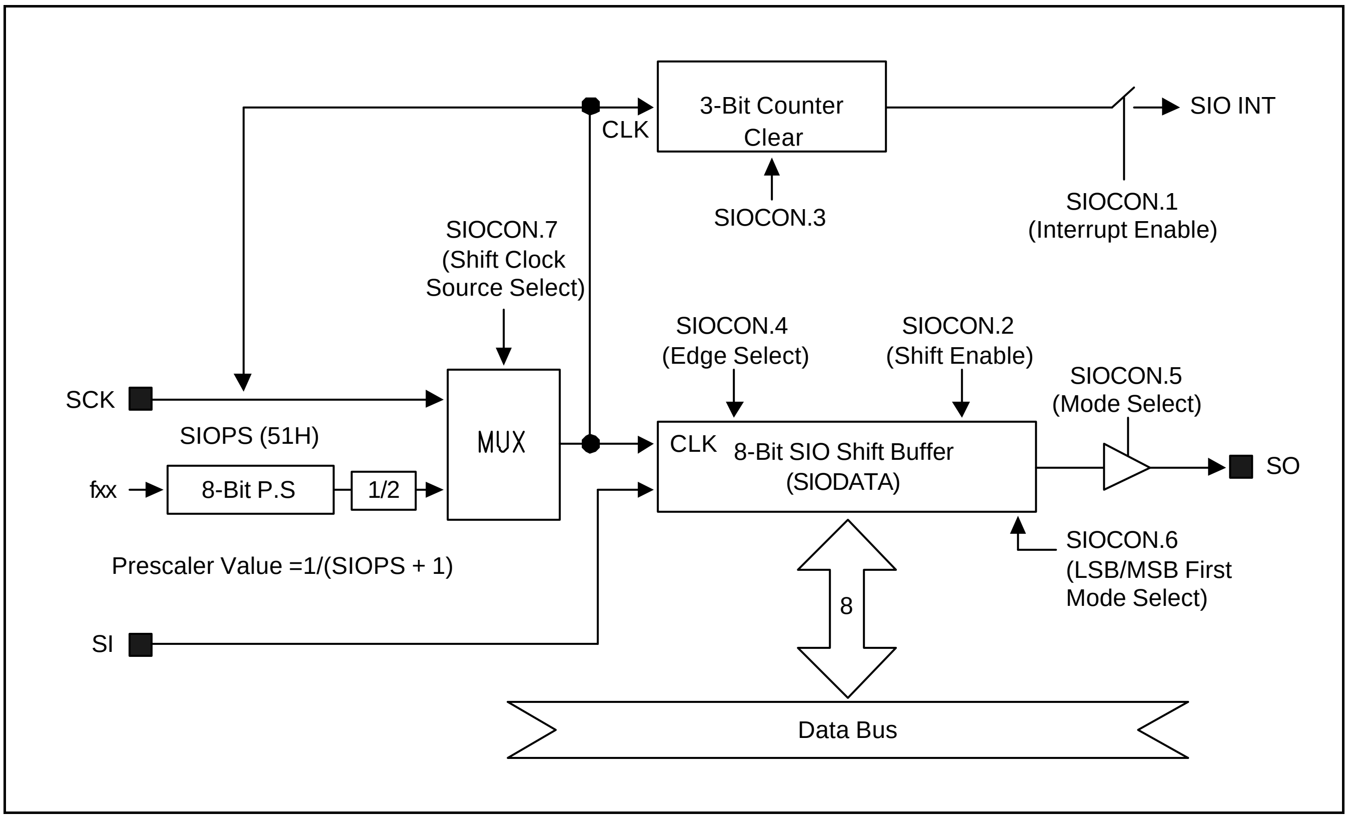

3-Bit Counter Clear
CLK
SIO INT
SIOCON.3
SIOCON.1 (Interrupt Enable)
SIOCON.7 (Shift Clock Source Select)
SIOCON.4 (Edge Select)
SIOCON.2 (Shift Enable)
SIOCON.5 (Mode Select)
SCK
SIOPS (51H)
MUX
CLK
8-Bit SIO Shift Buffer (SIODATA)
SO
fxx
8-Bit P.S
1/2
SIOCON.6 (LSB/MSB First Mode Select)
Prescaler Value =1/(SIOPS + 1)
8
SI
Data Bus

Figure 15-3. SIO Function Block Diagram

SERIAL I/O TIMING DIAGRAM

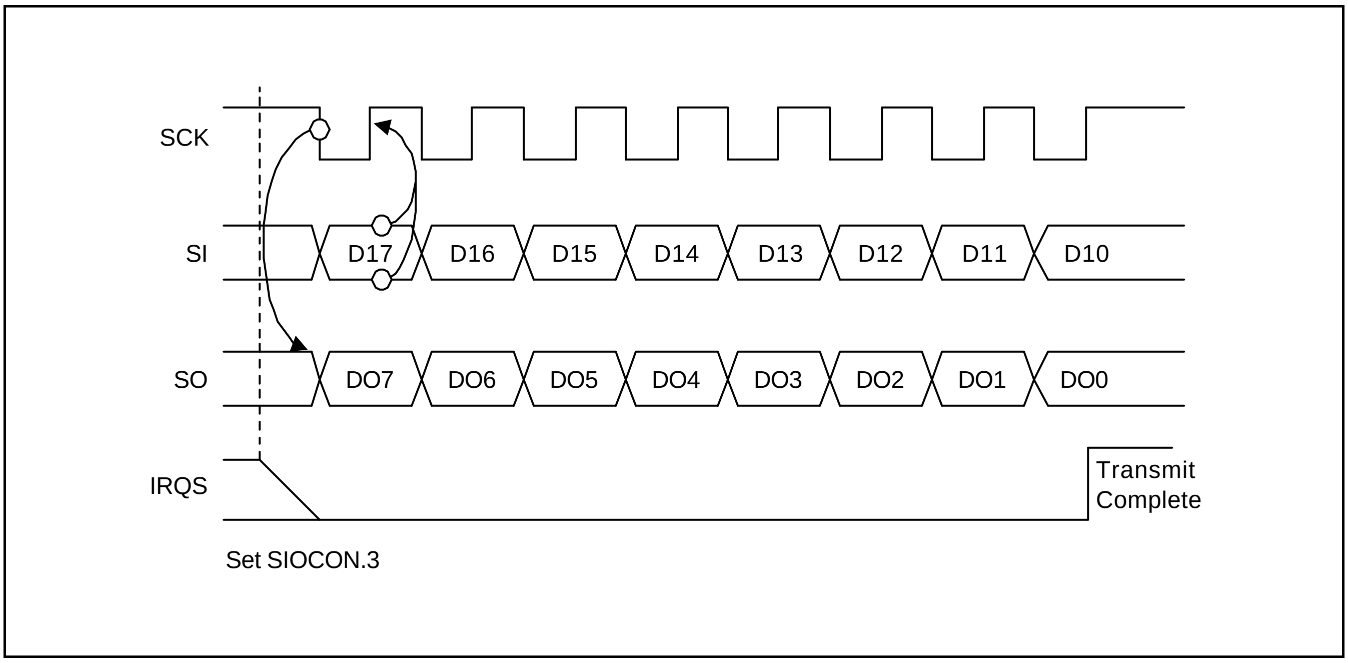


Figure 15-4. Serial I/O Timing in Transmit/Receive Mode (Tx at falling, SIOCON.4=0)

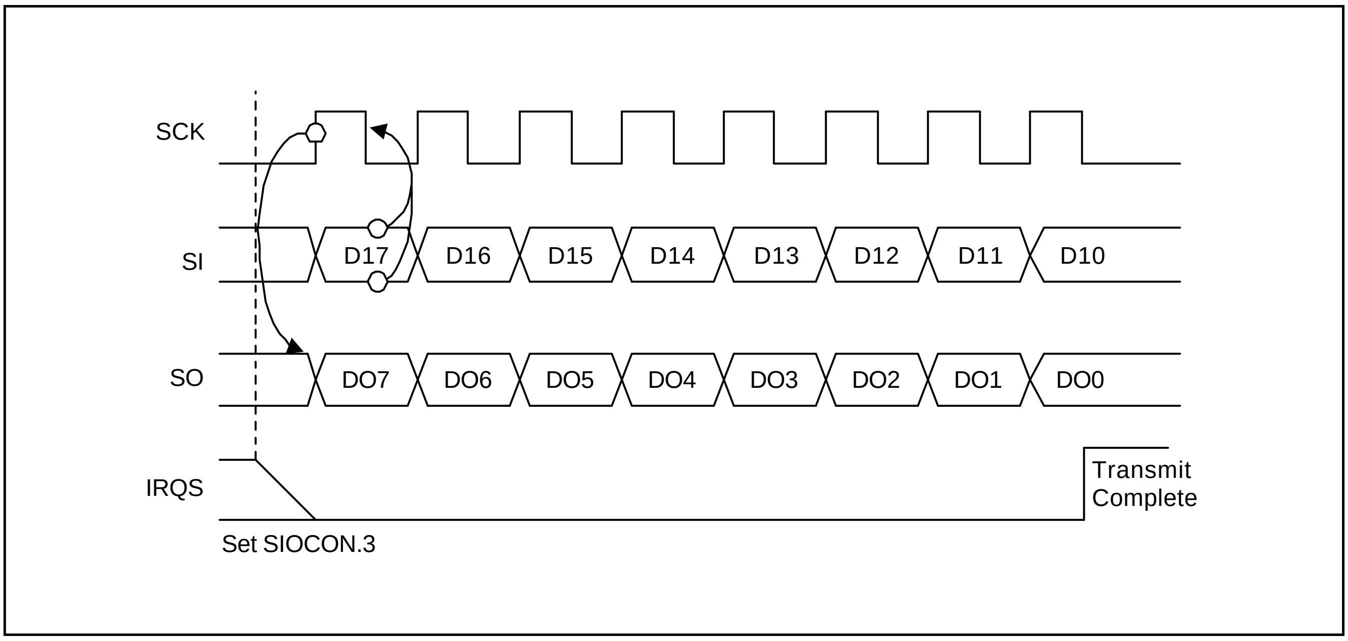


Figure 15-5. Serial I/O Timing in Transmit/Receive Mode (Tx at rising, SIOCON.4=1)

NOTES

16 UART

OVERVIEW

An UART contains a programmable baud rate generator, Rx and Tx port for UART communication, Tx and Rx shift registers, Tx and Rx buffer registers, Tx and Rx control blocks and control registers. Important features of the UART block include programmable baud rates, transmit/receive (full duplex mode), one or two stop bit insertion, 5-bit, 6-bit, 7-bit, or 8-bit data transmit/receive, and parity checking.

The baud rate generator can be clocked by the internal oscillation clock. The transmitter cotains a Tx data buffer register and a Tx shift register. Similary the receiver cotains a Rx data buffer register and a Rx shift register. Data to be transmitted is written to the Tx buffer register, then copied to the Tx shift register, and shift out by the transmit data pin (Tx). Data received is shifted in by the receive data pin (Rx), then copied from shift register to the Rx buffer register whenever one data byte is received. The control unit provides control for mode selection and status/interrupt generation.

UART Baud rate = fxx/(16 x (Divisor Value + 1))

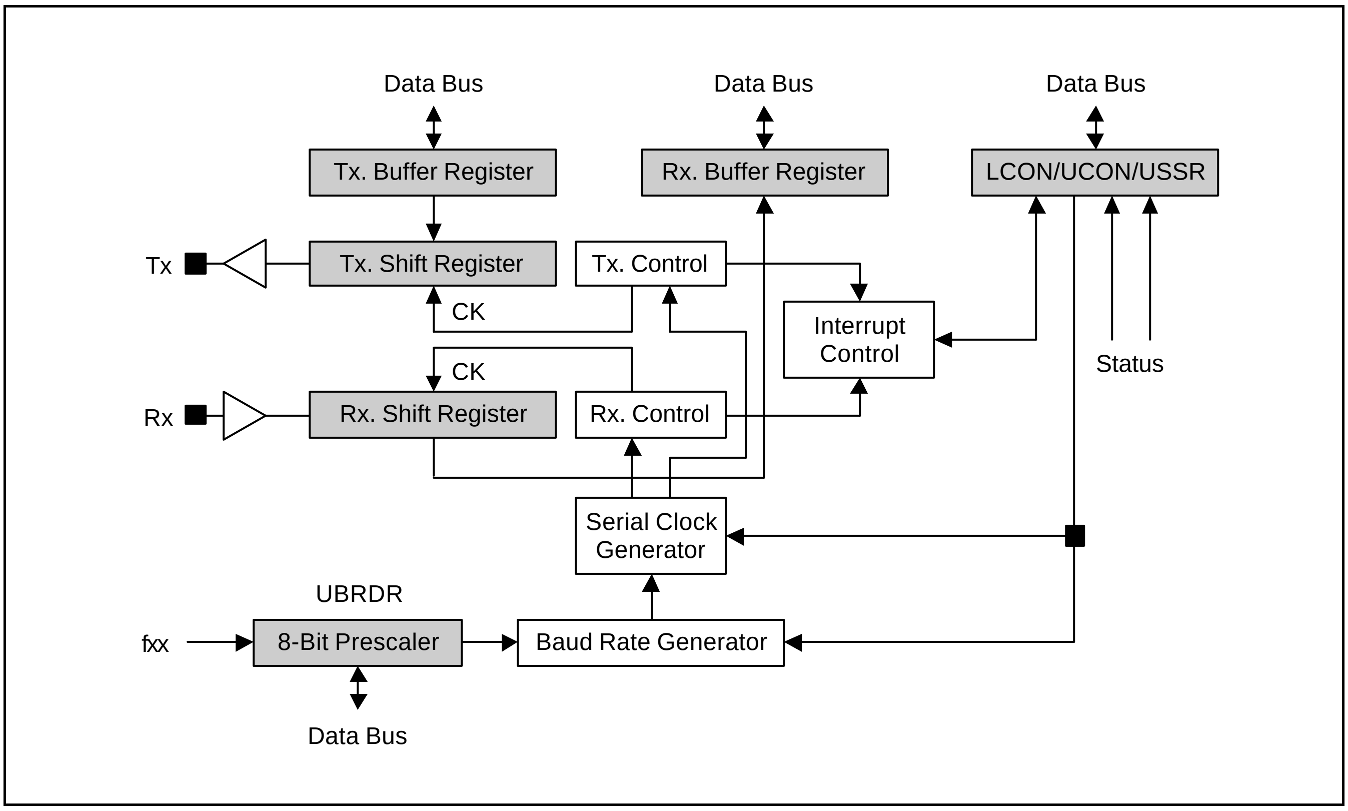


Figure 16-1. UART Block Diagram

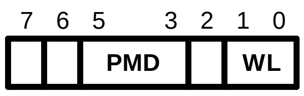
UART SPECIAL REGISTERS

UART LINE CONTROL REGISTER

The UART line control register, LCON, is used to control the UART.

Register	Address	R/W	Description	Reset Value
LCON	0xB0	R/W	UART line control register	00h

[1:0]	Word length (WL).	The two-bit word length value indicates the number of data bits to be transmitted or received per frame. The options are 5-bit, 6-bit, 7-bit, and 8-bit.
[2]	Number of stop bits	LCON[2] specifies how many stop bits are used to signal end-of-frame (EOF). When it is 0, one bit signals the EOF; when it is 1, two bits signal EOF.
[5:3]	Parity mode (PMD)	The 3-bit parity mode value specifies how parity generation and checking are to be performed during UART transmit and receive operations. There are five options (see Figure 16-2).
[6]	–	
[7]	–	



[1:0] Word-length per frame (WL)
00 = 5-bit
01 = 6-bit
10 = 7-bit
11 = 8-bit

[2] Number of stop bit at end of frame
0 = One stop bit per frame
1 = Two stop bits per frame

[5:3] Parity mode (PMD)
0xx = No parity bit in frame
100 = Odd parity
101 = Even parity
110 = Parity forced/checked as 1
111 = Parity forced/checked as 0

Figure 16-2. UART Line Control Register (LCON)

UART CONTROL REGISTER

The UART control register, UCON, is used to control the single-channel UART.

Register	Address	R/W	Description	Reset Value
UCON	0xB1	R/W	UART control register	00h

[0]	Rx interrupt enable	UART Rx interrupt control: 0 = Disable, 1 = Enable
[1]	Rx enable	UART Rx operation control: 0 = Disable, 1 = Enable
[2]	Rx status interrupt enable	This bit enables the UART to generate an interrupt if an exception (break, frame error, parity error, or overrun error) occurs during a receive operation. When UCON[2] is set to 1, a receive status interrupt will be generated each time a Rx exception occurs. When UCON[2] is 0, no receive status interrupt will be generated.
[3]	Tx interrupt enable	UART Tx interrupt control: 0 = Disable, 1 = Enable
[4]	Tx enable	
[5]	-	
[6]	Send break	Setting UCON[6] causes the UART to send a break. Break is defined as a continuous Low level signal on the transmit data output with a duration of more than one frame transmission time. By setting this bit when the transmitter is empty (transmitter empty bit, SSR[7] = 1), you can use the transmitter to time the frame. When SSR[7] is 1, write the transmit buffer register, TBR, with the data to be transmitted. Then poll the SSR[7] value. When it returns to 1, clear (reset) the send break bit, UCON[6].
[7]	Loopback bit	Setting UCON[7] causes the UART to enter loopback mode. In loopback mode, the transmit data output is sent High level and the transmit buffer register (TBR) is internally connected to the receive buffer register (RBR). This mode is provided for test purposes only.

UART STATUS REGISTER

The UART status register, USSR, is a read-only register that is used to monitor the status of serial I/O operations in the single-channel UART.

Register	Address	R/W	Description	Reset Value
USSR	0xB2	R	UART status register	c0h

[0]	Overrun error	USSR[0] is automatically set to 1 whenever an overrun error occurs during a serial data receive operation. If the receive status interrupt enable bit, UCON[2] is 1, a receive status interrupt will be generated if an overrun error occurs. This bit is automatically cleared to 0 whenever the UART status register (USSR) is read.
[1]	Parity error	USSR[1] is automatically set to 1 whenever a parity error occurs during a serial data receive operation. If the receive status interrupt enable bit, UCON[2] is 1, a receive status interrupt will be generated if a parity error occurs. This bit is automatically cleared to 0 whenever the UART status register (USSR) is read.
[2]	Frame error	USSR[2] is automatically set to 1 whenever a frame error occurs during a serial data receive operation. If the receive status interrupt enable bit, UCON[2] is 1, a receive status interrupt will be generated if a frame error occurs. The frame error bit is automatically cleared to 0 whenever the UART status register (USSR) is read.
[3]	Break interrupt	USSR[3] is automatically set to 1 to indicate that a break signal has been received. If the receive status interrupt enable bit, UCON[2], is 1, a receive status interrupt will be generated if a break occurs. The break interrupt bit is automatically cleared to 0 when you read the UART status register.
[4]	–	
[5]	Receive data ready	USSR[5] is automatically set to 1 whenever the receive data buffer register (RBR) contains valid data received over the serial port. The receive data can then be read from the RBR. When this bit is 0, the RBR does not contain valid data. Depending on the current setting of the SIO receive mode bits, UCON[1:0], an interrupt or a DMA request is generated when USSR[5] is 1.
[6]	Tx buffer register empty	USSR[6] is automatically set to 1 when the transmit buffer register (TBR) does not contain valid data. In this case, the TBR can be written with the data to be transmitted. When this bit is 0, the TBR contains valid Tx data that has not yet been copied to the transmit shift register. In this case, the TBR cannot be written with new Tx data. Depending on the current setting of the UART transmit mode bits, UCON[4:3], an interrupt or a DMA request will be generated whenever USSR[6] is 1.
[7]	Transmitter empty (T)	USSR[7] is automatically set to 1 when the transmit buffer register has no valid data to transmit and when the Tx shift register is empty. When the transmitter empty bit is 1, it indicates to software that it can now disable the transmitter function block.

UART TRANSMIT BUFFER REGISTER

The UART transmit holding register, TBR, contains an 8-bit data value to be transmitted over the single-channel UART.

Register	Address	R/W	Description	Reset Value
TBR	0xB3	W	Serial transmit buffer register	xxh

[7:0]	Transmit data	This field contains the data to be transmitted over the single-channel UART. When this register is written, the transmit buffer register empty bit in the status register, USSR[6], should be 1. This prevents overwriting transmit data that may already be present in the TBR. Whenever the TBR is written with a new value, the transmit register empty bit, SSR[6], is automatically cleared to 0.

UART RECEIVE BUFFER REGISTER

The receive buffer register, RBR, contains an 8-bit field for received serial data.

Register	Address	R/W	Description	Reset Value
RBR	0xB4	R	Serial receive buffer register	xxh

[7:0]	Receive data	This field contains the data received over the single-channel UART. When this register is read, the receive data ready bit in the UART status register, USSR[5], should be 1. This prevents reading invalid receive data that may already be present in the RBR. Whenever the RBR is written with a new value, the receive data ready bit, USSR[5], is automatically cleared to 0.

UART BAUD RATE PRESCALER REGISTERS

The value stored in the baud rate divisor register, UBRDR, is used to determine the serial Tx/Rx clock rate (baud rate) as follows:

Baud rate = fxx/((Divisor value + 1) x 16)

Register	Address	R/W	Description	Reset Value
UBRDR	0xB5	R/W	Baud rate divisor register	0000h

UART INTERRUPT PENDING REGISTER (UPEND)

Register	Address	R/W	Description	Reset Value
UPEND	0xB6	R/W	UART interrupt pending register	00h

Bit	Setting	Description
[0]	0 or 1	UART Rx interrupt pending bit 0: When read, interrupt is not pending. (When write, pending bit is clear) 1: When read, interrupt is pending. (When write, pending bit is not affected)
[1]	0 or 1	UART Error interrupt pending bit 0: When read, interrupt is not pending. (When write, pending bit is clear) 1: When read, interrupt is pending. (When write, pending bit is not affected)
[2]	0 or 1	UART Tx interrupt pending bit 0: When read, interrupt is not pending. (When write, pending bit is clear) 1: When read, interrupt is pending. (When write, pending bit is not affected)
[7:3]	–	–

17 I^2S BUS (INTER-IC SOUND)

OVERVIEW

Many digital audio systems are being introduced into the consumer audio market, including compact disc, digital audio tape, digital sound processors, and digital TV-sound. The digital audio signals in these systems are being processed by a number of (V) LSI ICs, such as:

- A/D and D/A converters;
- Digital signal processors;
- Error correction for compact disc and digital recording;
- Digital filters;
- Digital input/output interfaces.

Standardized communication structures are vital for both the equipment and the IC manufacturer, because they increase system flexibility. To this end, we have used the inter-IC sound (I^2S) bus-a serial link especially for digital audio.

The bus has only to handle audio data, while the other signals, such as sub-coding and control, are transferred separately. To minimize the number of pins required and to keep wiring simple, a 3-line serial bus is used consisting of a line for two time-multiplexed data channels, a word select line and a clock line. Since the transmitter and receiver have the same clock signal for data transmission, the transmitter as the master, has to generate the bit clock, word-select signal and data. In complex systems however, there may be several transmitters and receivers, which makes it difficult to define the master. In such systems, there is usually a system master controlling digital audio data-flow between the various ICs. Transmitters then, have to generate data under the control of an external clock, and so act as a slave.

Figure 17-1 illustrates some simple system configurations and the basic interface timing. Note that the system master can be combined with a transmitter or receiver, and it may be enabled or disabled under software control or by pin programming.

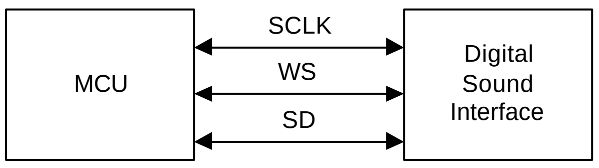


Figure 17-1. Simple System Configuration

THE I^{2}S BUS

As shown in Figure 17-1, the bus has three lines:

- Continuous serial clock (SCLK);
- Word select (WS);
- Serial data (SD);

and the device generating SCLK and WS is the master.

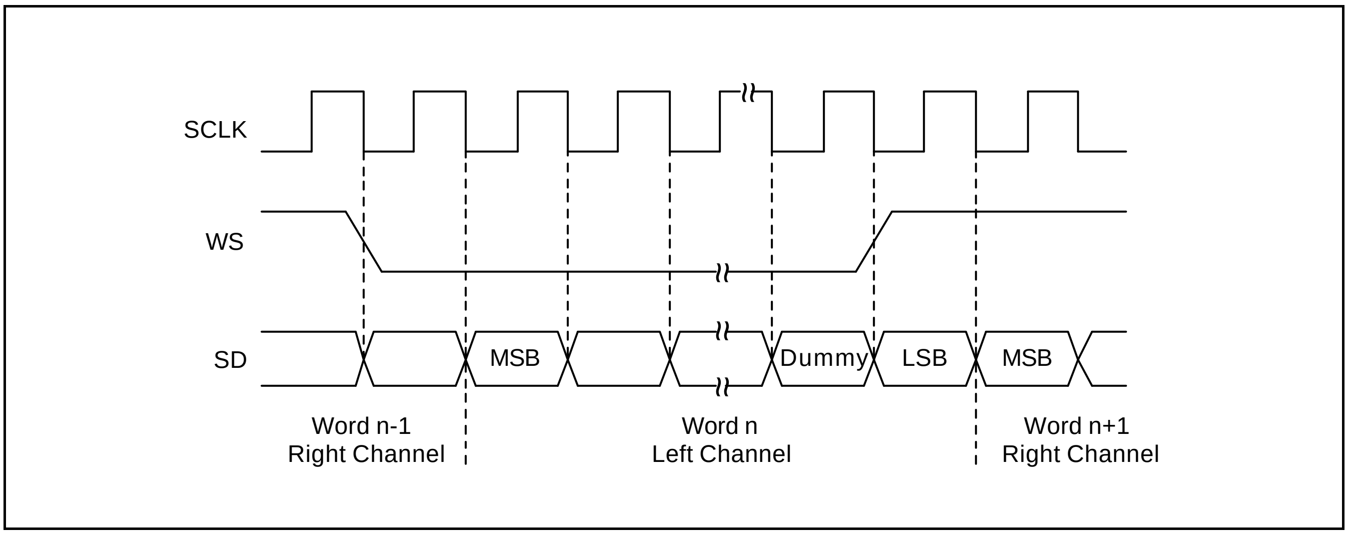


Figure 17-2. I^{2}S Basic Interface Format (Phillips)

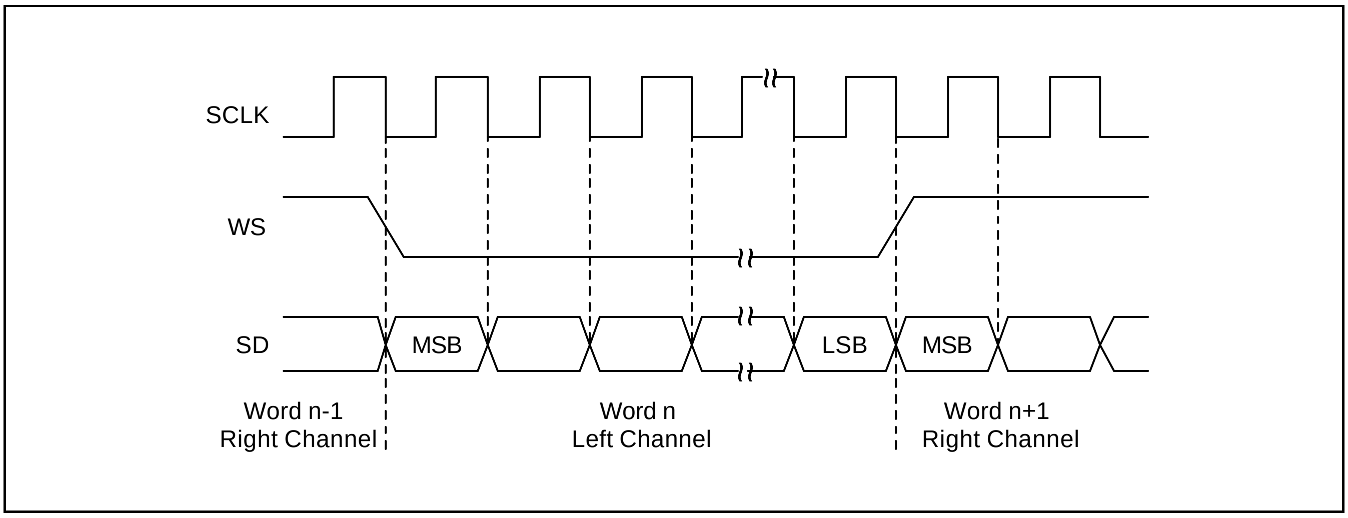


Figure 17-3. LSI Interface Format (Sony)

Serial Data

Serial data is transmitted in two's complement with the MSB first. The MSB is transmitted first because the transmitter and receiver may have different word lengths. It isn't necessary for the transmitter to know how many bits the receiver can handle, nor does the receiver need to know how many bits are being transmitted.

When the system word length is greater than the transmitter word length, the word is truncated (least significant data bits are set to 0) for data transmission. If the receiver is sent more bits than its word length, the bits after the LSB are ignored. On the other hand, if the receiver is sent fewer bits than its word length, the missing bits are set to zero internally. And so, the MSB has a fixed position, whereas the position of the LSB depends on the word length. The transmitter always sends the MSB of the next word one clock period after the WS changes.

Serial data sent by the transmitter may be synchronized with either the trailing (HIGH-to-LOW) or the leading (LOW-to-HIGH) edge of the clock signal. However, the serial data must be latched into the receiver on the leading edge of the serial clock signal, and so there are some restrictions when transmitting data that is synchronized with the leading edge (see Figure 17-4 and Table 17-1).

Word Select

The word select line indicates the channel being transmitted:

- WS = 0; channel 1 (left);
- WS = 1; channel 2 (right).

WS may change either on a trailing or leading edge of the serial clock, but it does not need to be symmetrical. In the slave, this signal is latched on the leading edge of the clock signal. The WS line changes one clock period before the MSB is transmitted. This allows the slave transmitter to derive synchronous timing of the serial data that will be set up for transmission. Furthermore, it enables the receiver to store the previous word and clear the input for the next word (see Figure 17-2).

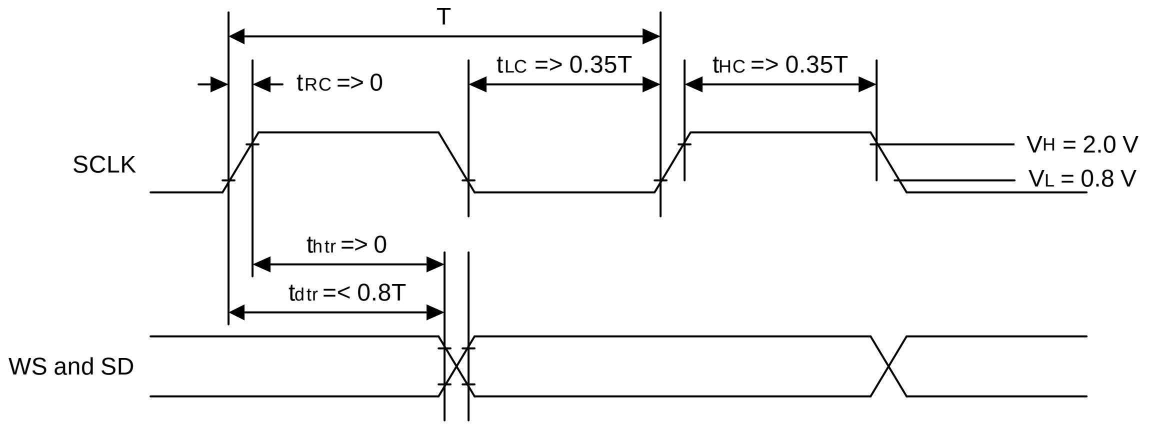
TIMING

In the I^2S format, any device can act as the system master by providing the necessary clock signals. A slave will usually derive its internal clock signal from an external clock input. This means, taking into account the propagation delays between master clock and the data and/or word-select signals, that the total delay is simply the sum of:

- The delay between the external (master) clock and the slave's internal clock; and
- The delay between the internal clock and the data and/or word-select signals.

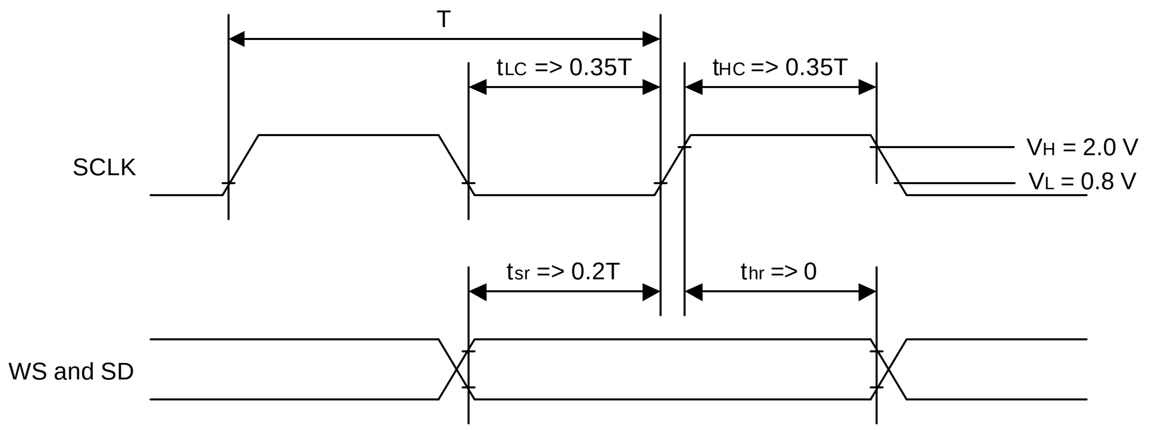
For data and word-select inputs, the external to internal clock delay is of no consequence because it only lengthens the effective set-up time (see Figure 17-3). The major part of the time margin is to accommodate the difference between the propagation delay of the transmitter, and the time required to set up the receiver.

All timing requirements are specified relative to the clock period or to the minimum allowed clock period of a device. This means that higher data rates can be used in the future.



T = clock period
T_{tr} = minimum allowed clock period for transmitter
$T > T_{tr}$
t_{RC} is only relevant for transmitters in slave mode.

Figure 17-4. Timing for I²S Transmitter



T = clock period
T_r = minimum allowed clock period for transmitter
$T > T_{tr}$

Figure 17-5. Timing for I²S Receiver

Table 17-1. Master Transmitter with Data Rate of 2.5 MHz (10%) (Unit: ns)

	Min	Typ	Max	Condition
Clock period T	360	400	440	Ttr = 360
Clock HIGH tHC	160			min > 0.35T = 140 (at typical data rate)
Clock LOW tLC	160			min > 0.35T = 140 (at typical data rate)
Delay tdtr			300	max < 0.80T = 320 (at typical data rate)
Hold time thtr	100			min > 0
Clock rise-time tRC			60	max > 0.15T = 54 (atrelevent in slave mode)

Table 17-2. Slave Receiver with Data Rate of 2.5 MHz (10%) (Unit: ns)

	Min	Typ	Max	Condition
Clock period T	360	400	440	Ttr = 360
Clock HIGH tHC	110			min < 0.35T = 126
Clock LOW tLC	110			min < 0.35T = 126
Set-up time tsr	60			min < 0.20T = 72
Hold time thtr	0			min < 0

I^{2}S SPECIAL REGISTER DESCRIPTION

I^{2}S CONTROL REGISTERS

Table 17-3. Function Register Description

Register	Address	R/W/C	Description
IISCON0	0x58	R/W	I^{2}S0 Control register
IISCON1	0x5C	R/W	I^{2}S1 Control register
IISMODE0	0x59	R/W	I^{2}S0 Mode register
IISMODE1	0x5D	R/W	I^{2}S1 Mode register
IISPTR0	0x5A	R/W	I^{2}S0 buffer pointer register
IISPTR1	0x5E	R/W	I^{2}S1 buffer pointer register
IISBUF	0xC0-0xFF	R/W	I^{2}S Buffer registers

I^{2}S CONTROL REGISTERS (IISCON)

Register	Address	R/W	Description	Reset Value
IISCON0	0x58	R/W	I^{2}S control register 0	00h
IISCON1	0x5C	R/W	I^{2}S control register 1	00h

IIS control register0 has the following control bit settings:

[0]	I^{2}S0	Enable I^{2}S0 block when this bit is set as 1. 0: I^{2}S0 block disable. 1: I^{2}S0 block enable.
[1]	MCLK	Select normal or MCLK output mode 0: Normal output 1: MCLK output
[2]	SD0_OUT	Select the input or output mode for each I^{2}S0 Serial Data pin. Set SD0 pin as an input or an output. 0: Input 1: Output
[3]	SLAVE	MCU generate the SCLK0 and WS0 signal to transmit or receive the serial data. 0: Master mode, SCLK0 and WS0 is output mode. 1: Slave mode, SCK0 and WS0 is input mode.
[4]	SCKPOL	Select the serial clock0 polarity. 0: Active low 1: Active high
[5]	CHPOL	Select the Left/Right channel polarity. 0: Left High 1: Left Low
[6]	LSBFIRST	Select MSB("0") first at or LSB("1") first in serial interface

[7]	PSMODE.	Select the Phillips IIS0 interface format or Sony LSI interface format. 0: I^{2}S, 1: LSI

IIS control register1 has the following control bit settings:

[0]	I^{2}S1 Enable	Enable I^{2}S1 block when this bit is set as 1. 0: I^{2}S1 block is disabled. 1: I^{2}S1 block is enabled.
[1]	–	–
[2]	SD1_OUT	Select the input or output mode for each I^{2}S1 Serial Data pin. Set SD1 pin as an input or an output. 0: Input 1: Output
[3]	SLAVE	MCU generate the SCLK1 and WS1 signal to transmit or receive the serial data. 0: Master mode, SCLK1 and WS1 is output mode. 1: Slave mode, SCK1 and WS1 is input mode.
[4]	SCKPOL	Select the serial clock1 polarity. 0: Active low 1: Active high
[5]	CHPOL	Select the Left/Right channel polarity. 0: Left High 1: Left Low
[6]	LSBFIRST	Select MSB("0") first at or LSB("1") first in serial interface
[7]	PSMODE.	Select the Phillips IIS0 interface format or Sony LSI interface format. 0: I^{2}S, 1: LSI

I^{2}S MODE REGISTERS (IISMODE)

Register	Address	R/W	Description	Reset Value
IISMODE0	0x59	R/W	I^{2}S mode register 0	00h
IISMODE1	0x5D	R/W	I^{2}S model register 1	00h

I^{2}S control register has the following control bit settings:

[1-0]	BitPSlot:	Set the bit number per slot 00: 8-bit 01: 16-bit 10: 24-bit 11: 32-bit
[5-2]	SFREQ	Set the Sampling frequency for Left/Right channel output 0100: Select an 11.025 kHz as audio sampling frequency 0101: Select an 22.05 kHz as audio sampling frequency 0110: Select an 44.1 kHz as audio sampling frequency 0111: Select an 88.2 kHz as audio sampling frequency 1000: Select an 8 kHz as audio sampling frequency 1001: Select an 16 kHz as audio sampling frequency 1010: Select an 32 kHz as audio sampling frequency 1100: Select an 12 kHz as audio sampling frequency 1101: Select an 24 kHz as audio sampling frequency 1110: Select an 48 kHz as audio sampling frequency 1111: Select an 96 kHz as audio sampling frequency
[7-6]	BitPFs	Set the bit number per sampling frequency 01: 32-bit 10: 48-bit (slave only) 11: 64-bit

When MCLK (Master Clock) is enabled, MCLK frequency is Fs x 256.

I^2S POINTER REGISTERS (IISPTR)

Register	Address	R/W	Description	Reset Value
IISPTR0	0x5A	R/W	I^2S buffer pointer register 0	00h
IISPTR1	0x5E	R/W	I^2S buffer pointer register 1	00h

[5-0]	Pointer	Buffer pointer register. The bit 5 is not incremented but bit4-0 are automatically incremented whenever buffer operation is done. After pointer value, IISPTR[4:0] reached to 0x1F, interrupt request flag is active. IISPTR will increment from the initial value to IISPTR[4:0] = 0x1f, IISPTR[4:0] is cleared to 0x00. However IISPTR[5] is not changed. If IISPTR[5] = 1, second half buffer (0xE0–0xEF) is accessible. Otherwise, first half buffer (0xC0–0xDF) is accessible.

I^2S BUFFER REGISTERS (IISBUF)

Register	Address	R/W	Description	Reset Value
IISBUF	0xC0-0xFF	R/W	I^2S buffer registers	00h

[7-0]	DATA	I^2S buffer registers hold the audio data for transmitting data to audio DAC or receiving data from external I.C.

NOTES

18 SSFDC (SOLID STATE FLOPPY DISK CARD)

OVERVIEW

S3FB42F build a interface logic for SmartMedia™ card, called as SSFDC, solid state floppy disk card.
The SSFDC interface include the use of simple hardware together with software to generate a basic control signal or ECC for SmartMedia™.

The built-in SSFDC interface logic consists of ECC block and the read/write strobe signal generation block.
The high speed RISC CPU core, CalmRisc support high speed control for other strobe signal generation and detection. Therefore, ALE, CLE, CE and etc signal should be operated by CPU instruction. This mechanism provide the balanced cost and power consumption without the de-graduation of SSFDC access speed.

Physical format is necessary to maintain wide compatibility. SmartMedia™ has a standard physical format. System makers and controller manufacturers are requested to conform their products to such specifications.
For logical format, SmartMedia™ employs a DOS format on top of physical format. See PC Card Standard Vol.7 and other references for more information. With all SmartMedia™ products, physical and logical formatting has been completed at time of shipment.

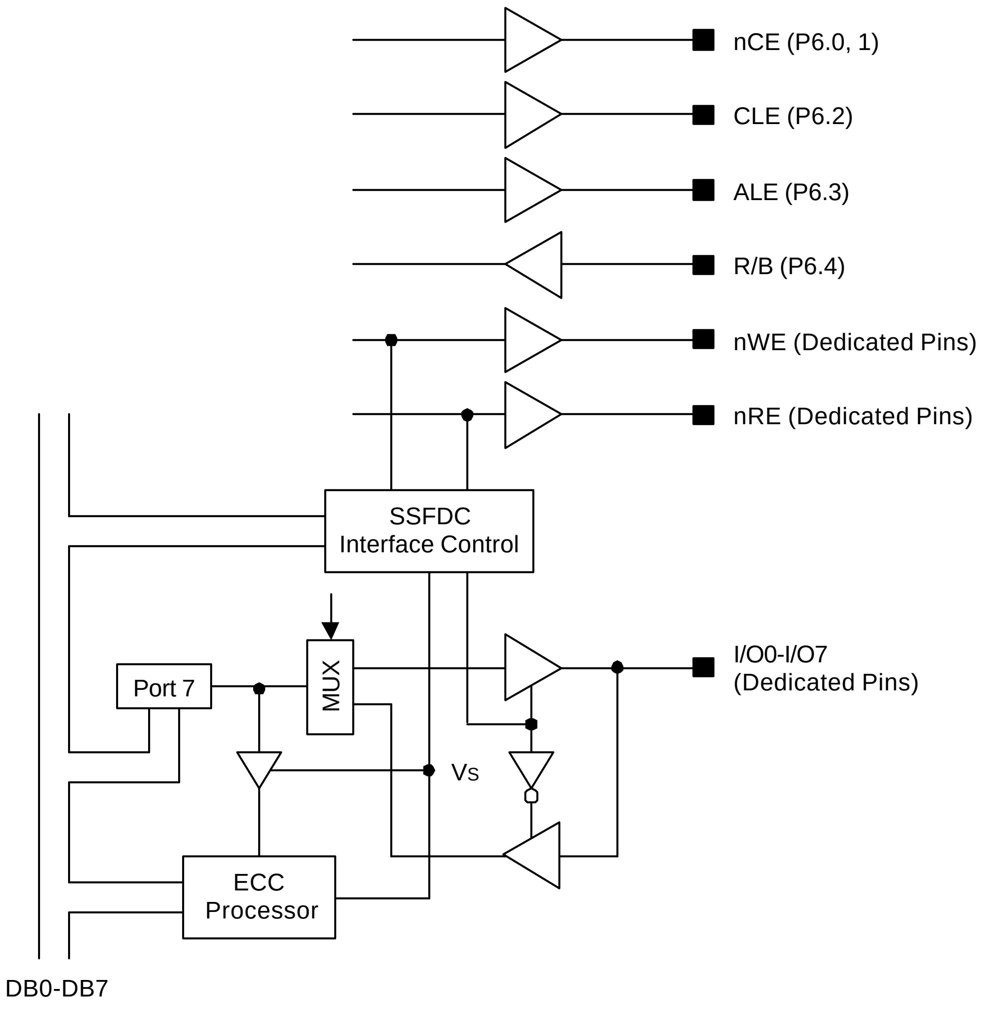


Figure 18-1. Simple System Configuration

SSFDC REGISTER DESCRIPTION

Description of the register in the SSFDC, SmartMedia interface is listed the below table.

Table 18-1. Control Register Description

Register	Address	R/W/C	Description
SMCON	70H	R/W	SmartMedia control register
ECCNT	71H	R/W	ECC count register
ECCL/H/X	72/73/74H	R/W	ECC data register low/high/extension
ECCRSTL/H	75/76H	R/W	ECC result data register low/high
ECCCLR	77H	W	ECC clear register

SMARTMEDIA CONTROL REGISTER (SMCON)

Register	Address	R/W	Description	Reset Value
SMCON	0x70	R/W	SmartMedia control register	00h

[0]	ECC Enable	This bit enable or disable the ECC operation in the SmartMedia block. When this bit is set as "1", ECC block is activated and ECC operation is done whenever accessing the Port 7. *"1" : Enable "0" : Disable*.
[1]	Enable SmartMedia interface	This bit control the operation of SmartMedia block. When this bit is set as "1", Port 7 is activated as I/O data bus of SmartMedia interface. SmartMedia control signal is generated whenever accessing the Port 7.
[3:2]	Wait cycle control	These bit control the wait cycle insertion when access to SmartMedia card. 00: No wait in nWE or nRD signal 01: 1 wait in nWE or nRD signal 10: 3 wait in nWE or nRD signal 11: Invalid setting.

SMARTMEDIA ECC COUNT REGISTER (ECCNT)

Register	Address	R/W	Description	Reset Value
ECCNT	0x71	R/W	SmartMedia ECC count register	00h

[7:0]	Count	This field acts as the up-counter. You can know the ECC count number by reading this register. This register is cleared by setting the SMCON.0, *Start* bit or overflow of counter.

SMARTMEDIA ECC DATA REGISTER (ECCDATA)

Register	Address	R/W	Description	Reset Value
ECCX	0x74	R/W	SmartMedia ECC data extension register	00h
ECCH	0x73	R/W	SmartMedia ECC data high register	00h
ECCL	0x72	R/W	SmartMedia ECC data low register	00h

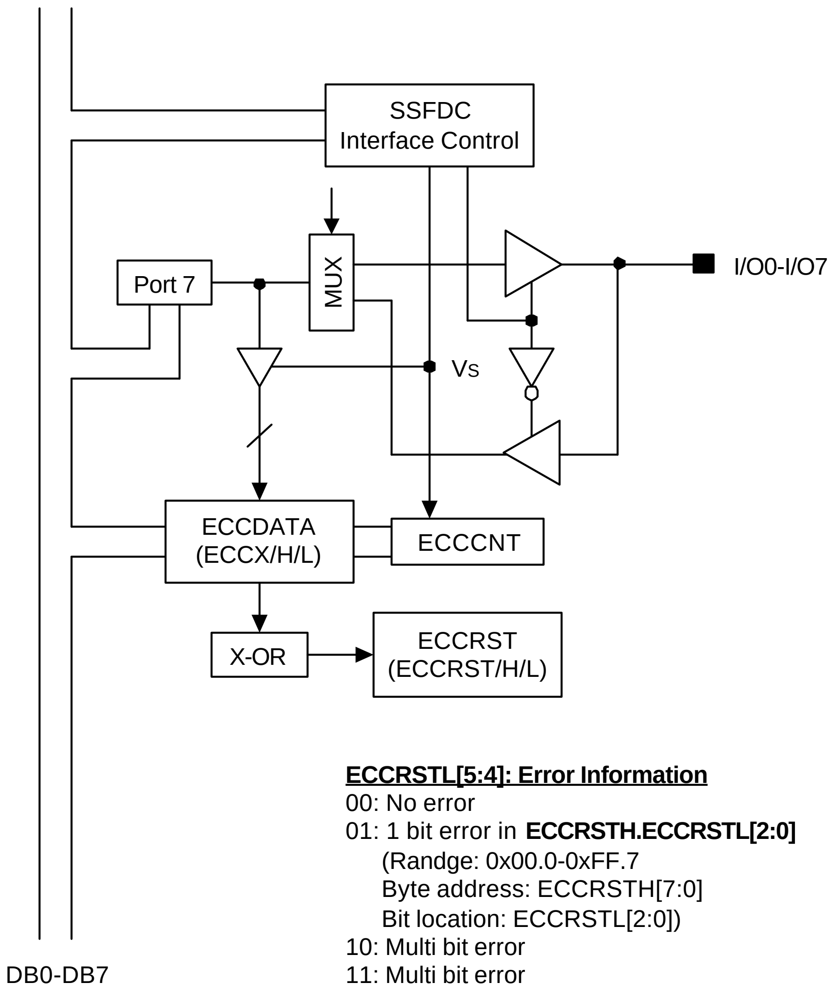
[7:0]	Data	Data field acts as ECC data register when SMCON.0, e*nable* bit is set. The access instruction to Port 7 execute an 1byte ECC operation. The writing to ECCCLR register have all ECC data registers clear to zero

SMARTMEDIA ECC RESULT DATA REGISTER (ECCRST)

Register	Address	R/W	Description	Reset Value
ECCRSTL	0x75	R/W	SmartMedia ECC result data register low	00h
ECCRSTH	0x76	R/W	SmartMedia ECC result data register high	00h

[7:0] Data

After ECC compare operation is executed, ECC result out to ECC result data register, ECCRST.
ECCRSTH[7:0] have the byte location with correctable error bit.
ECCRSTL[2:0] have the bit location where is correctable error bit.
ECCRSTL[5:4] have the error information.
00: No error occurred.
01: detect 1 bit error but recoverable
10: detect the multiple bit error.
11: detect the multiple bit error.



<u>**ECCRSTL[5:4]: Error Information**</u>
00: No error
01: 1 bit error in **ECCRSTH.ECCRSTL[2:0]**
(Randge: 0x00.0-0xFF.7
Byte address: ECCRSTH[7:0]
Bit location: ECCRSTL[2:0])
10: Multi bit error
11: Multi bit error

Figure 18-2. ECC Processor Block Diagram

NOTES

19 PARALLEL PORT INTERFACE

OVERVIEW

The S3FB42F's parallel port interface controller (PPIC) supports four IEEE Standard 1284 communication modes:

— Compatibility mode (Centronics^{TM})

— Nibble mode

— Byte mode

— Enhanced Capabilities Port (ECP) mode

The PPIC also supports all variants of these communication modes, including device ID requests. The PPIC contains specific hardware to support the following operations:

— Automatic hardware handshaking between host and peripheral in Compatibility and ECP modes

These features can substantially improve data transfer rates when S3FB42F operates the parallel port in the Compatibility or ECP mode.

In addition, hardware handshaking over the parallel port can be enabled or disabled by software. This gives you the direct control of PPIC signals as well as the eventual use of future protocols. Other operations defined in the IEEE Standard 1284, such as negotiation, Nibble mode and Byte mode data transfers, and termination cycles, must be carried out by software. The IEEE 1284 EPP communications mode is not supported.

NOTE

Here we assume that you are familiar with the parallel port communication protocols specified in the IEEE 1284 Parallel Port Standard. If you are not, we strongly recommend for you to read this standard beforehand. It would be helpful for you in understanding the contents described in this section.

PPIC OPERATING MODES

The S3FB42F PPIC supports four kinds of handshaking modes for data transfers:

— Software handshaking mode to forward and reverse data transfers

— Compatibility hardware handshaking mode to forward data transfers

— ECP hardware handshaking mode to forward and reverse data transfers

Mode selection is specified in the PPIC control low register (PPCONL). By setting the PPCONL[6:4], one of these four modes is enabled.

Software Handshaking Mode

This mode is enabled by setting the PPCONL's mode-selection bits, PPCONL[5:4], to "00."

In this mode, you can use PPIC interrupt event registers (PPINTCON and PPINTPND) and the read/write PPIC status register (PPSTAT and PPSCON) to detect and control the logic levels on all parallel port signal pins. Software can control all parallel port operations, including all four kinds of parallel port communications protocols supported by the S3FB42F (refer to IEEE 1284 standard for operation control). In addition, it also gives software the flexibility of adopting new and revised protocols.

Compatibility Hardware Handshaking Mode

Compatibility hardware handshaking mode is enabled by setting the PPCONL's mode-selection bits as "01", i.e. PPCONL[5:4] = 01. In this mode, hardware generates all handshaking signals needed to implement compatibility mode of the parallel port communication protocol.

When this mode is enabled, the PPIC automatically generates a BUSY signal to receive the leading edge of nSTROBE from the host, and latches the logic levels on PPD7-PPD0 pins into the PPDATA register. The PPIC then waits for nSTROBE to negate it and for the PPDATA's data field to be read. After the PPDATA is read, the PPIC asserts nACK for the duration specified in the ACK Width Data Register (PPACKD), and then negates the nACK and BUSY signal to conclude the data transfer, as shown in Figure 19-1.

NOTE

The BUSY-control bit initial value in the PPSCON register, PPSCON[3], which is "1" after a system reset, results in the high logic level on BUSY output and handshaking disable. To enable hardware handshaking in this mode, the BUSY-control bit PPSCON[3] must be cleared to "0" by software beforehand.

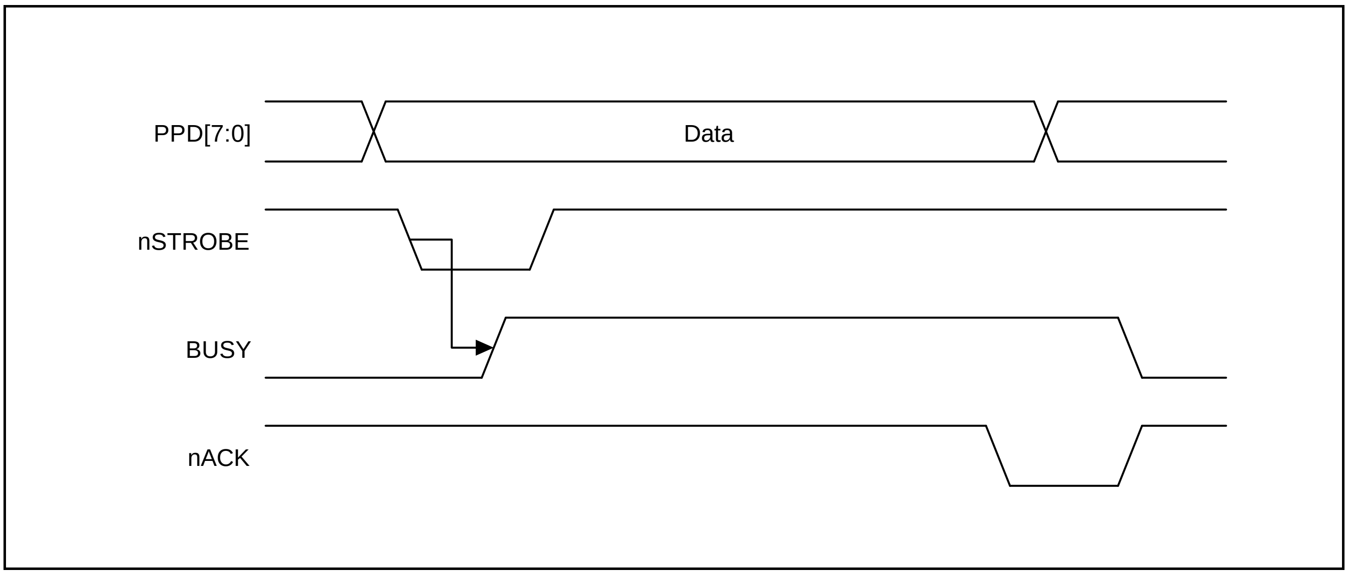


Figure 19-1. Compatibility Hardware Handshaking Timing

ECP Mode

ECP hardware handshaking mode is enabled by setting the PPCONL's mode-selection bits to "10", i.e. PPCONL[5:4] = 10. In this mode, hardware generates handshaking signals needed to implement ECP mode of the parallel port communication protocol.

When receiving data from the host, the PPIC automatically responds to the high-to-low transition on the nSTROBE by latching the logic levels on the PPD7-PPD0 to PPDATA(and PPCDATA). The nAUTOFD logic level, which is latched to the PPINTPNDH[1] or [0], command or data received flags, indicates whether the current data on the PPDATA[7:0] is a data-byte or a command-byte. When the PPDATA is read, the PPIC drives BUSY to high level and waits for nSTROBE to go high level. It then drives BUSY to low level to conclude one forward data transfer operation, as shown in Figure 19-2.

The reception of a command byte causes the command received-bit in the PPIC interrupt pending register, PPINTPNDH[1], to be set to "1". By examining the PPDATA/PPCDATA[7], software will interpret the command byte as a channel address if it is "1" and carry out the corresponding operation, or interpret the command byte as a run-length count if it is "0" and then perform data decompression.

During reverse data transfers, software is responsible for data compression, and writing data or command byte in PPDATA or PPCDATA to define the logic levels on PPD7-PPD0 and BUSY pins. The write to PPDATA or PPCDATA indicates whether the current data on the PPD[7:0] is a data-byte or a command-byte. When some value is written to PPDATA, that means data-byte type and is output through the BUSY pin to high. When some value is written to PPCDATA, that means command-byte type and is output through the BUSY pin to low. In response to writing the PPDATA or PPCDATA, the PPIC automatically drives the nACK to low level and waits for the nAUTOFD to go to high level. It then drives nACK to high level to conclude one reverse data transfer operation, as shown in Figure 19-3.

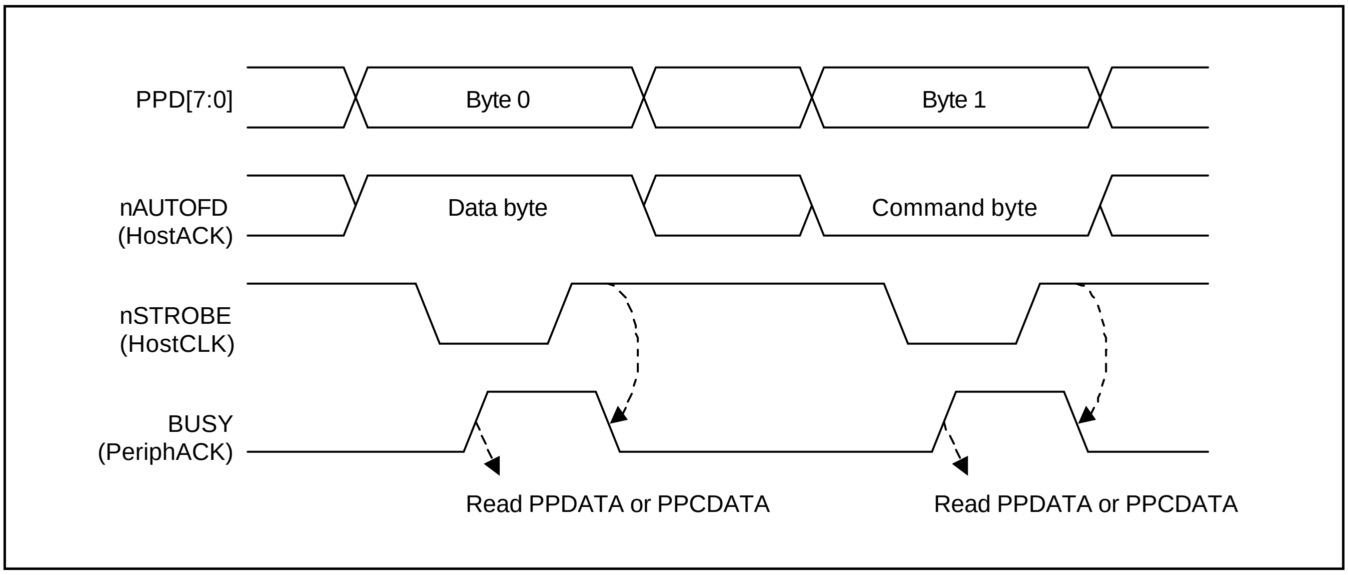


Figure 19-2. ECP Hardware Handshaking Timing (Forward)

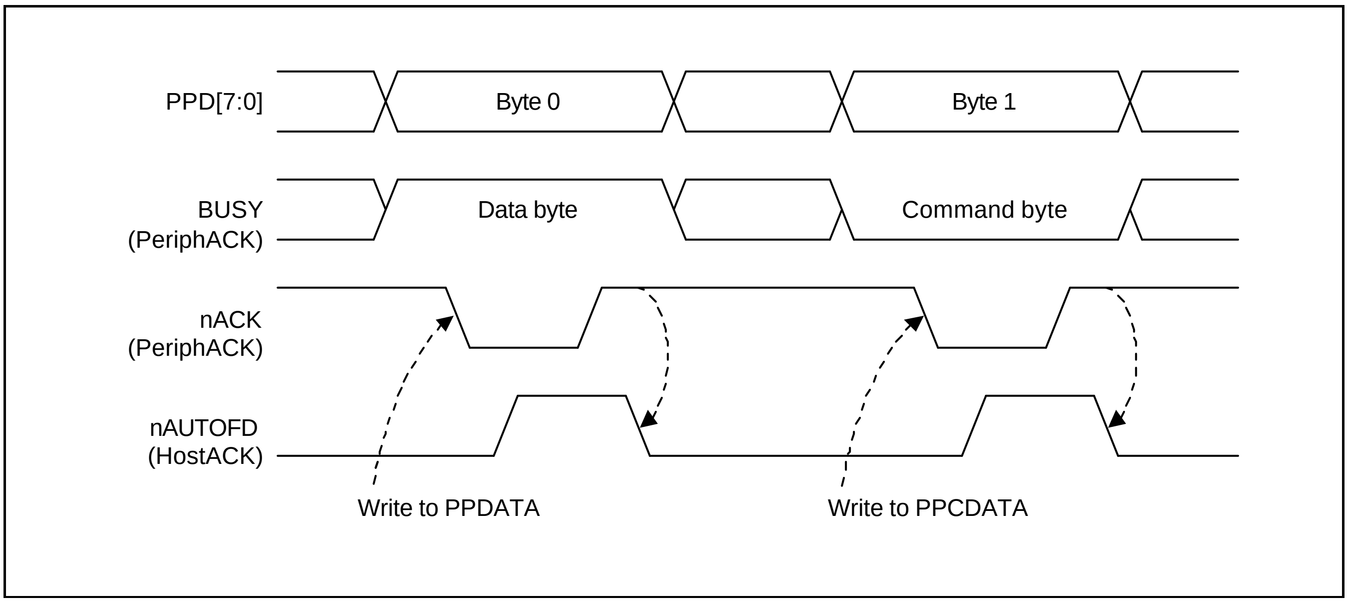


Figure 19-3. ECP Hardware Handshaking Timing (Reverse)

Digital Filtering

The S3FB42F provides digital filtering function on host control signal inputs, nSELECTIN (P1284), nSTROBE (HostCLK), nAUTOFD (H0TACK) and nINIT (nReverseRequest), to improve noise immunity and make the PPIC more impervious to the inductive switching noise. The digital filtering function can be enabled regardless of hardware handshaking or software handshaking.

If this function is enabled, the host control signal can be detected only when its input level keeps stable during three sampling periods.

Digital filtering can be disabled to avoid signal missing in some specialized applications with high bandwidth requirement. Otherwise, it is recommended that digital filtering be enabled.

PPIC SPECIAL REGISTERS

PARALLEL PORT DATA/COMMAND DATA REGISTER

The parallel port data/command data register, PPDATA/PPCDATA, contains an 8-bit data field, PPDATA/PPCDATA[7:0], that defines the logic level on the parallel port data pins, PPD[7:0].

Register	Address	R/W	Description	Reset Value
PPDATA	0x60	R/W	Parallel port data register	00h
PPCDATA	0x61	R/W	Parallel port command data register	00h

[7:0] This is an 8-bit read/write field. When PPCONL[7] is zero and this field (PPDATA or PPCDATA) is read, this field provides the logic level on the PPD[7:0], which is latched when the strobe input from the host (nSTROBE) transits from high to low level. (The PPCONL[7] bit determines the forward or reverse dataflow direction of the parallel port.) When PPCONL[7] is one and this field(PPDATA or PPCDATA) is written, the value of this field determines the logic level on the PPD[7:0].

During the ECP forward data transfers, the logic level of the nAUTOFD is read from PPINTPNDH[1] or [0], command-byte received or data-byte received. The nAUTOFD indicates whether the data in the PPDATA/PPCDATA is a data-byte or a command-byte.

When read PPDATA or PPCDATA,
command-byte: PPINTPND[1:0] = '10b'
data-byte: PPINTPND[1:0] = '01b'

To read the nAUTOFD from the PPINTPNDH[1] or [0] the following two conditions are required:
1) nSTROBE has transited from high level to low level.
2) The data bus output enable bit in the PPCONL[7] is 0.

When the ECP data transfers are in reverse and the data bus output enable bit in the parallel port control register, PPCONL[7] is 1, the logic level of BUSY pin is written from PPDATA or PPCDATA. The BUSY pin indicates that the data written in the PPDATA is a data-byte, or the data written in the PPCDATA is a command-byte.

BUSY pin
0 = Command-byte in the PPCDATA[7:0]
1 = Data-byte in the PPDATA[7:0]

PARALLEL PORT STATUS CONTROL AND STATUS REGISTER

The parallel port status control and status register, PPSCON, PPSTAT, contain eleven bits to control the parallel port interface signals. These eleven bits consist of four read-only bits to read the logic level of the host input pins, two read-only bits to read the logic level on the BUSY and nACK output pins, and five read/write bits to control the logic levels on the printer output pins by software for handshaking control

Register	Address	R/W	Description	Reset Value
PPSCON	0x62	R/W	Parallel port status control register	08h

[0]	nFAULT control (nPeriphRequest)	Setting this bit drives the nFAULT output to low level; clearing it drives the signal high level on the external nFAULT pin. The nFAULT informs the host of a fault condition in the printer engine.
[1]	SELECT control (Xflag)	Setting this bit to one drives the SELECT output to High level; clearing it to zero drives the signal low on the external SELECT pin. The SELECT informs the host of a response from the printer engine.
[2]	PERROR control (nACKReverse)	Setting this bit drives PERROR output to high level; clearing it drives the signal low level on the external PERROR pin. The PERROR informs the host that a paper error has occurred in the engine.
[3]	BUSY control (PeriACK)	Setting this bit drives the external BUSY output to high level by force. This disables hardware handshaking. When this bit is zero, the external BUSY output is the internal BUSY signal.
[4]	nACK control (PeriphCLK)	Setting this bit drives the external nACK output to low level by force. This is generally done to disable hardware handshaking. When this bit is zero, the external nACK is the internal nACK signal.

Register	Address	R/W	Description	Reset Value
PPSTAT	0x63	R/W	Parallel port status register	3Fh

[0]	BUSY status (PeriphACK)	This read-only bit reflects the logic level on the external BUSY output pin. After a system reset, the PPSCON[3] is "1", which results in one, the value of PPSTAT[0] being "1". So, for compatibility mode operation, you must clear the PPSCON[3] by software beforehand so as to enable the hardware handshaking.
[1]	nACK status (PeriphCLK)	This read-only bit reflects the level read on the external nACK output pin. After a system reset, PPSTAT[1] is "1". When the PPSCON[4] is set to be high, this bit is forced to be low and then the internal nACK is ignored.
[2]	nSLCTIN status (P1284)	This read-only bit reflects the level read on the nSLCTIN input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCONL[2:3], are not set to zero.
[3]	nSTROBE status (HostCLK)	This read-only bit reflects the level read on the nSTROBE input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCONL[2:3], are not set to zero.
[4]	nAUTOFD status (HostACK)	This read-only bit reflects the level read on the nAUTOFD input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCONL[2:3], are not set to zero.
[5]	nINIT status (nReverseRequest)	This read-only bit reflects the level read on the nINIT input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCONL[2:3], are not set to zero.

PARALLEL PORT CONTROL REGISTER

The parallel port control register, PPCON, is used to configure the PPI operations, such as handshaking, digital filtering, operating mode, data bus output and abort operations. The PPCONH[6:4] bits are read-only.

Register	Address	R/W	Description	Reset Value
PPCONL	0x64	R/W	Parallel port control low register	00h

[0]	Software reset	Setting the software reset bit causes the PPIC's handshaking control c to immediately terminate the current operation and return to software Idle state. When PPCONL[0] is set to "1", the full status bit, PPCONH[5], is automatically cleared to "0".
[1]	PPIC enable	Setting this bit enables PPIC mode. Clearing this bit disable PPIC operation and enter power saving mode.
[3:2]	Digital filter enable	Setting this bit enables digital filtering on all four host control signal inputs: nSELECTIN, nSTROBE, nAUTOFD, and nINIT. 00: Disable 01: 2 Step filtering 10: 3 Step filtering 11: 3 Step filtering
[6:4]	Mode selection	This three-bit value selects the current operating mode of the parallel port interface x00:Software mode x01:Compatibility mode 010: Forward ECP mode 110: Reverse ECP mode **Software mode**: disables all hardware handshaking so that handshaking can be performed by software. **Compatibility mode**: Compatibility mode hardware handshaking can be enabled during a forward data transfer. You can change the mode selection at any time, but if a Compatibility mode operation is currently in-progress, it will be completed as a normal operation. Mode should be changed from Compatibility mode to another mode only when BUSY is high level. This ensures that there is no parallel port activity while the parallel port is being re-configured. **ECP mode**: ECP mode hardware handshaking support can be enabled during forward or reverse data transfers. You can change the mode selection at any time, but if an ECP cycle is currently in progress, it will be completed as a normal operation.

[7]	Data bus output enable	The parallel port data bus output enable bit performs two functions: 1) It controls the state of the tri-state output drivers. 2) It qualifies the data latching from the output drivers into the parallel port data register's data field, PPDATA[7:0]. When PPCONL[7] is "0", the parallel port data bus lines, PPD[7:0] are disabled. This allows data to be latched onto the PPDATA or PPCDATA's data field. When PPCONL[7] is "1", the PPD[7:0] is enabled and data is prevented from being latched onto the PPDATA or PPCDATA's data field. In this frozen state, the data field is unaffected by the transition of nSTROBE. The setting of the abort bit, PPCONH[3], affects the operation of the data bus output enable bit, PPCONL[7]. If PPCONH[3] is "1", the nSELECTIN must remain high to allow PPCONL[7] to be set, or to remain set. If PPCONL[7] is "1" and nSELECTIN goes low, the PPCONL[7] is cleared and setting this bit will have no effect.

Register	Address	R/W	Description	Reset Value
PPCONH	0x65	R/W	Parallel port control high register	00h

[0]	–	
[1]	–	
[2]	Abort	The abort bit causes the parallel port interface controller to use nSELECTIN to detect the time when the host suddenly aborts a reverse transfer and returns to compatibility mode; If PPCONH[2] is "1", the low level on nSELECTIN causes the parallel port data bus output enable bit PPCONL[7] to be cleared, and the output drivers for the data bus lines PPD[7:0] to be tri-stated.
[3]	Error cycle	The error cycle bit is used to execute an error cycle in compatibility mode. When PPCONH[3] is set to "1", the BUSY status bit in the parallel port status register, PPSTAT[0], is set to "1". This immediately causes the S3FB42F to drive the BUSY to high level. If you set the error cycle bit while a compatibility mode handshaking sequence is in progress, the PPSTAT[0] will remain to be set to one beyond the end of the current cycle. The error cycle bit does not affect the nACK pulse if it is already active, but it will delay an nACK pulse if it is about to be generated. When PPCONH[3] is "1", software can set or clear the parallel port status register control bits: PPSCON[0] (nFAULT control), PPSCON[1] (SELECT control), and PPSCON[2] (PERROR control). When PPCONH[3] is cleared to "0", the parallel port interface controller generates a delayed nACK pulse and makes BUSY low active to finish the error cycle.
[4]	–	
[5]	Data latch status	If a data is latched to PPDATA, then this bit is set to '1'. It is automatically cleared to zero when the PPDATA is read in software, compatibility and forward ECP mode.
[6]	Data empty	In reverse ECP mode, this bit specifies the PPDATA is empty. It is automatically cleared to zero while the PPDATA is written with a new data.

PARALLEL PORT INTERRUPT EVENT REGISTERS

The two parallel port interrupt event registers, PPINTCON and PPINTPND, control interrupt-related events for the input signal originating from the host, as well as data reception, command reception, and invalid events. The parallel port interrupt control register, PPINTCON, contains the interrupt enable bits for each interrupt event that is indicated by the PPINTPND status bits. If the PPINTCON enable bit is "1", the corresponding event causes the S3FB42F CPU to generate an interrupt request. Otherwise, no interrupt request is issued.

NOTE

To clear the corresponding pending bit to zero after a interrupt service routine, write the pending bit to zero. The value of the pending bit is changed from one to zero automatically.

Register	Address	R/W	Description	Reset Value
PPINTCONL	0x66	R/W	Parallel port interrupt control low register	00h
PPINTPNDL	0x68	R/W	Parallel port interrupt pending low register	00h

[0]	nSLCTIN Low-to-High (P1284)	The bit of PPINTPND is set when a Low-to-High transition on Nslctin is detected. If the corresponding enable bit is set in the PPINTCON register, an interrupt request is generated.
[1]	nSLCTIN High-to-Low	The bit of PPINTPND is set when a High-to-Low transition on nSLCTIN is detected. If the corresponding enable bit is set in the PPINTCON register, an interrupt request is generated.
[2]	nSTROBE Low-to-High (HostCLK)	The bit of PPINTPND is set when a Low-to-High transition in the nSTROBE is detected. If the corresponding enable bit is set in the PPINTCON register, an interrupt request is generated.
[3]	nSTROBE High-to-Low	The bit of PPINTPND is set when a High-to-Low transition in the nSTROBE is detected. If the corresponding enable bit is set in the PPINTCON register, an interrupt request is generated.
[4]	nAUTOFD Low-to-High (HostACK)	The bit of PPINTPND is set when a Low-to-High transition in the nAUTOFD is detected. If the corresponding enable bit is set in the PPINTCON register, an interrupt request is generated.
[5]	nAUTOFD High-to-Low	The bit of PPINTPND is set when a High-to-Low transition in the nAUTOFD is detected. If the corresponding enable bit is set in the PPINTCON register, an interrupt request is generated.
[6]	nINIT Low-to-High (nReverseRequest)	The bit of PPINTPND is set when a Low-to-High transition in the nINIT is detected. If the corresponding enable bit is set in the PPINTCON register, an interrupt request is generated.
[7]	nINIT High-to-Low	The bit of PPINTPND is set when a High-to-Low transition in the nINIT is detected. If the corresponding enable bit is set in the PPINTCON register, an interrupt request is generated.

Register	Address	R/W	Description	Reset Value
PPINTCONH	0x67	R/W	Parallel port interrupt control high register	00h
PPINTPNDH	0x69	R/W	Parallel port interrupt pending high register	00h

[0]	Data received	The bit of PPINTPND is set when data is latched into the PPDATA register's data field. This occurs during every High-to-Low transition of nSTROBE when the parallel port data bus enable bit, PPCONL[7], is "0". An interrupt is also generated if the ECP-with-RLE mode is enabled, and if a data decompression is in progress.
[1]	Command received	The bit of PPINTPND is set when a command byte is latched into the PPDATA register data field. If ECP-without-RLE mode is enabled, the command received interrupt is issued whenever a run-length or channel address is received. If ECP-with-RLE mode is enabled, the command received interrupt is issued only when a channel address is received. This event can be posted only when ECP mode is enabled. The corresponding enable bit in the PPINTCON register determines whether or not an interrupt request will be generated when a command byte is received.
[2]	Invalid transition	The bit of PPINTPND is set when nSLCTIN transitions high-to-low in the middle of an ECP forward data transfer handshaking sequence. This interrupt is issued if nSLCTIN is low when nSTROBE is low or when BUSY is high. This event can be detected only when ECP mode is enabled and should return to compatibility mode.
[3]	Transmit Data Empty	The bit of PPINTPND is set to one when the transmit data register (=PPDATA) can be written during an ECP reverse data transfers

PARALLEL PORT ACK WIDTH REGISTER

This register contains the 8-bit nACK pulse width field. This value defines the nACK pulse width whenever the parallel port interface controller enters Compatibility mode, that is, when the parallel port control register mode bits, PPCONL[5:4], are set to "01". The nACK pulse width is selectable from 0 to 255 X_{IN} periods.

The nACK pulse width can be modified at any time and with any PPIC operation mode selection, but it can only be used during a compatibility handshaking cycle. If you change the nACK width near the end of a data transfer (when nACK is already low), the new pulse width value does not affect the current cycle. The new pulse width value would be used at the start of the next cycle.

Register	Address	R/W	Description	Reset Value
PPACKD	0x6A	R/W	Parallel port acknowledge width data register	xxh

The value in the 8-bit field defines the nACK pulse width when Compatibility mode is enabled (PPCONL[5:4]=01). The period of the nACK pulse can range from 0 to 255 X_{IN} with 2 X_{IN} steps.

20 8-BIT ANALOG-TO-DIGITAL CONVERTER

OVERVIEW

The S3FB42F has six 8-bit resolution A/D converter input (ADC0 to ADC5). The 8-bit A/D converter (ADC) module uses successive approximation logic to convert analog levels entering at one of the six input channels to equivalent 8-bit digital values (ADDATAH, ADDATAL). The analog input level must lie between the AV_{REF} and AV_{SS} values. The A/D converter has the following components:

- Analog comparator with successive approximation logic
- D/A converter logic (resistor string type)
- ADC control register (ADCON)
- Six multiplexed analog data input pins (ADC0-ADC5)
- 8-bit A/D conversion data output register (ADDATAH, ADDATAL)
- 6-bit digital input port
- AV_{REF} and AV_{SS} pins

FUNCTION DESCRIPTION

To initiate an analog-to-digital conversion procedure, you write the channel selection data in the A/D converter control register ADCON to select one of the six analog input pins (ADCn, n = 0-5) and set the conversion start or enable bit, ADCON.0. The conversion result data load to ADDATA register.

During a normal conversion, A/D C logic initially sets the successive approximation register to 80H (the approximate half-way point of an 8-bit register). This register is then updated automatically during each conversion step. The successive approximation block performs 8-bit conversions for one input channel at a time. You can dynamically select different channels by manipulating the channel selection bit value (ADCON.6-4) in the ADCON register. To start the A/D conversion, you should set a the enable bit, ADCON.0. When a conversion is completed, ADCON.3, the end-of-conversion (EOC) bit is automatically set to 1 and the result is dumped into the ADDATA register where it can be read. The A/D converter then enters an idle state. Remember to read the contents of ADDATA before another conversion starts. Otherwise, the previous result will be overwritten by the next conversion result.

NOTE

Because the A/D converter has no sample-and-hold circuitry, it is very important that fluctuation in the analog level at the ADC0-ADC5 input pins during a conversion procedure be kept to an absolute minimum. Any change in the input level, perhaps due to noise, will invalidate the result. If the chip enters to STOP or IDLE mode in conversion process, there will be a leakage current path in A/D block. You must use STOP or IDLE mode after A/D C operation is finished.

CONVERSION TIMING

The A/D conversion process requires 4 steps (4, 16, 32 or 64 clock edges) to convert each bit and 2 steps to setup the A/D Converter block. Therefore, a total of 14 steps are required to complete an 8-bit conversion. One step can be 1 clock, 4 clocks, 8 clocks or 16 clocks by software.

With an 20 MHz CPU clock frequency, one clock cycle is 50 ns. The conversion rate is calculated as follows:

$$\boxed{\text{Start 2 step} + (\text{1 step/bit} \times \text{10 bits}) + \text{EOC 2 step} = \text{56 clocks, where 1 step} = \text{4 clocks}}$$

To get the correct A/D conversion result data, A/D conversion time should be longer than 20μs whatever oscillation frequency is used.

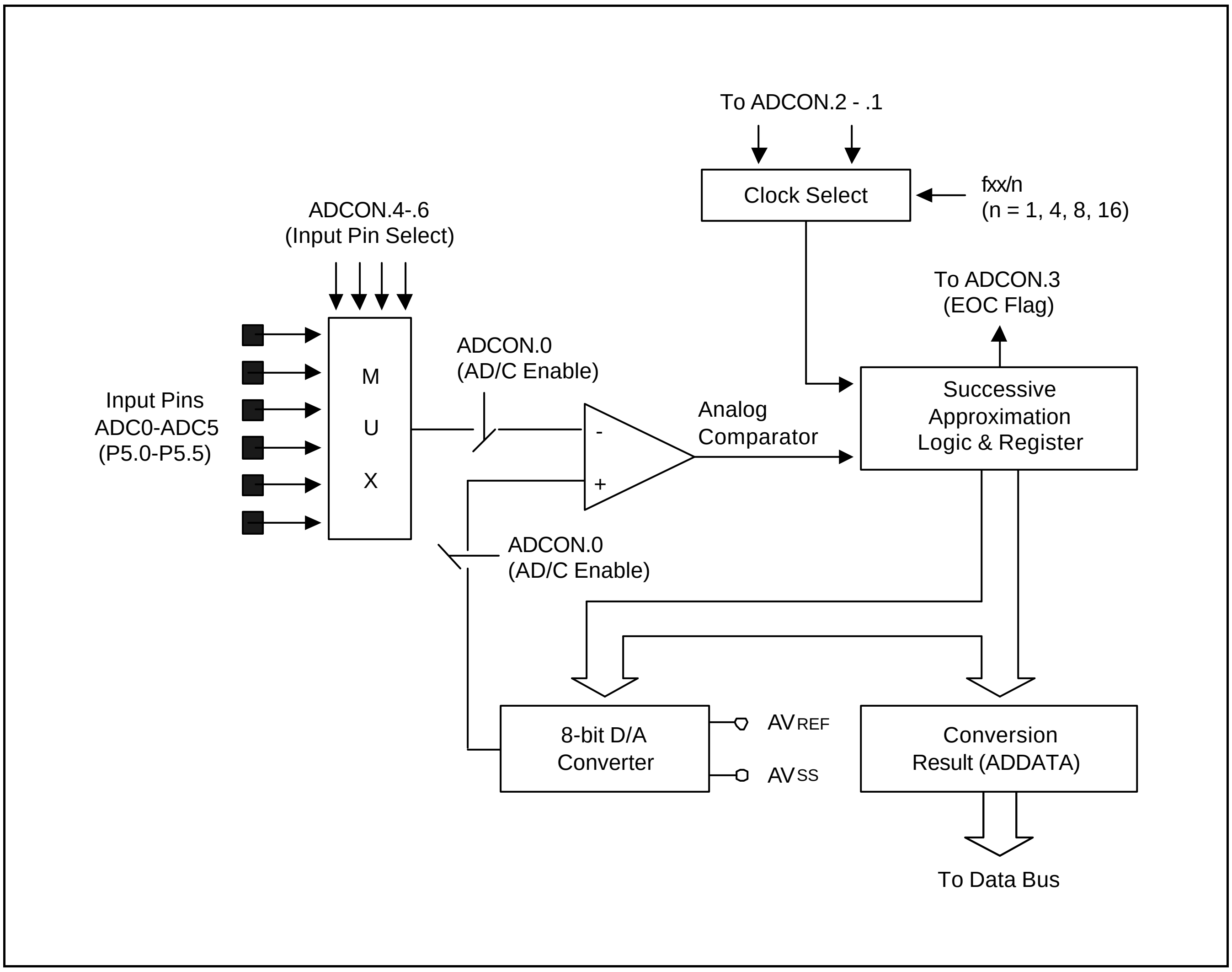


Figure 20-1. A/D C Block Diagram

A/D C SPECIAL REGISTERS

A/D C CONTROL REGISTERS

The A/D C control registers, ADCON is used to control the operation of the six 8-bit A/D C channel.

Register	Address	R/W	Description	Reset Value
ADCON	0x54	R/W	A/D C control register	00h

A/D Converter control register has the following control bit settings:

[0]	ADSTR	0: A/D conversion is disabled. 1: A/D conversion begins and is cleared after conversion.
[2:1]	Select the Conversion Speed	00: Step clock = fxx/16. 01: Step clock = fxx/8. 10: Step clock = fxx/4. 11: Step clock = fxx/1.
[3]	EOC (read-only)	0: Conversion is not completed. 1: This flag is set after conversion.
[6:4]	A/D C input select	000: select a ADC0 001: select a ADC1 010: select a ADC2 011: select a ADC3 100: select a ADC4 101: select a ADC5

[7] Not used.

A/D CONVERTER DATA REGISTERS

The A/D Conversion data high register, ADDATA, contains a conversion result value that specify analog input channel.

Register	Address	R/W	Description	Reset Value
ADDATA	0x55	R	A/DC Conversion Result data register	xxh

A/D Converter data register has the following bits:

[7:0] A/D C Data This register has the bit 7 to bit 0 of an A/D conversion result value.

NOTES

21 I^2C-BUS INTERFACE

OVERVIEW

The S3FB42F internal IIC bus (I^2C-bus) controller has the following important features:

— It requires only two bus lines, a serial data line (SDA) and a serial clock line (SCL). When the I^2C-bus is free, both lines are High level.

— Each device that is connected to the bus is software-addressable by a multi master using a unique address. Slave relationships on the bus are constant. The bus master can be either a master-transmitter or a master-receiver. The I^2C bus controller supports multi master mode.

— It supports 8-bit, bi-directional, serial data transfers.

— The number of ICs that you can connect to the same I^2C-bus is limited only by the maximum bus capacitance of 400 pF.

Figure 21-1 shows a block diagram of the S3FB42F I^2C-bus controller.

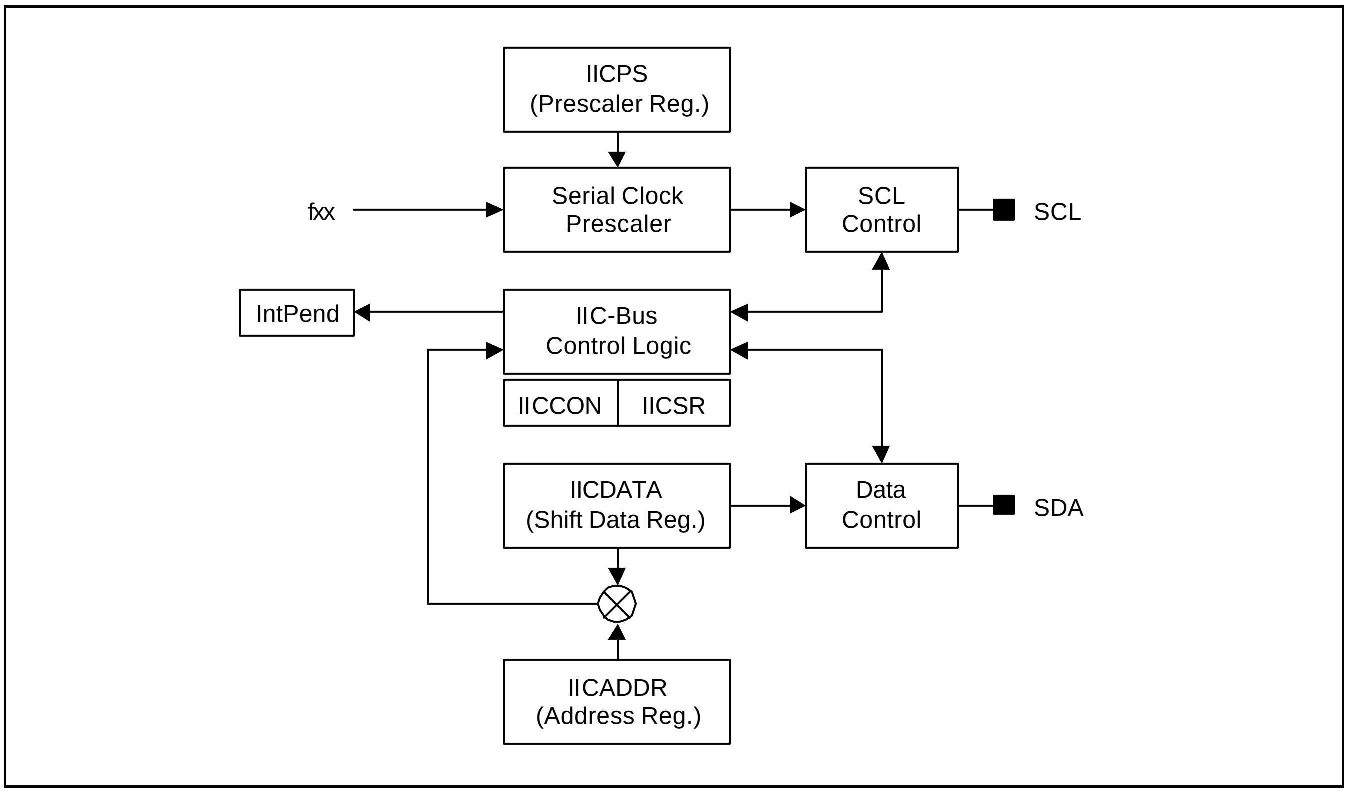


Figure 21-1. I^2C-Bus Block Diagram

FUNCTIONAL DESCRIPTION

The S3FB42F I^2C bus controller is the master or slave of the serial I^2C-bus. Using a prescaler register, you can program the serial clock frequency that is supplied to the I^2C bus controller. The serial clock frequency is calculated as follows:

fxx/(4 $\times$ (prescaler register (IICPS) value + 1)): IICPS must not be 00h.

In master Tx mode, to start a I2C-bus arbitration, the programmer writes a slave address to the data register, IICDATA and “0x3D” to the control register, IICCON. The bus controller then generates Start condition and shifts the 7-bit slave address.

The receiver sends an acknowledge by pulling the SDA line from High to Low during a master SCL pulse. After acknowledg3e cycle, the status register, IICSR, is updated corresponding arbitration result and interrupt request is activated if interrupt is enable.

After sensing interrupt or polling the status register, the programmer can continue the data shift operation. For the data arbitration, the programmer writes the data to the data register, IICDATA and writes “0x3E” to the control register. For the consecutive read/write operations, you must set the ACK bit in the control status register.

For read operations, you can read the data after you have confirmed the pending bit in the interrupt pending register. To signal the end of the read operation, you can reset the ACK bit to inform the receive/transmitter when the last byte is to be written/read.

Following a read/write operation, you set IICCON[1:0] to “3” to generate a Stop code. If you want to complete another data transfer before issuing the Stop code, you can send the Start code using the Repeat Start command (with IICCON[1:0] = “1”). When the slave address and read/write control bit have been sent, and when the receive acknowledge ahs been issued to control SCL timing, the data transfer is initiated.

I^2C SPECIAL REGISTERS

MULTI-MASTER I^2C-BUS CONTROL REGISTER

The I^2C-bus control register, IIICCON, is used to control the I^2C module.

Register	Address	R/W	Description	Reset Value
IICCON	0xB8	R/W	I^2C-Bus control register	00h

[1:0]	I^2C-bus arbitration control	This two-bit value controls I^2C operations. 00: No interrupt, pending 01: Generate start condition and shift address byte 10: Shift data byte 11: Generate Stop condition
[3:2]	I^2C-bus Tx/Rx mode selection	This two-bit value determines which mode is currently able to read/write data from/to IICDATA. 00: Slave Rx mode (default) 01: Slave Tx mode 10: Master Rx mode 11: Master Tx mode
[4]	I^2C-bus acknowledge (ACK) enable bit	This bit value determines whether I^2C-bus enables or disables the ACK signal generation.
[5]	I^2C bus enable bit	This bit specifies whether I^2C-bus is enabled or disabled. 0: Disable serial Tx/Rx 1: Enable serial Tx/Rx
[6]	–	–
[7]	Reset	If '1' is written to this bit, the I^2C bus controller is reset to its initial state (It is not automatically cleared).

MULTI-MASTER I^{2}C-BUS CONTROL/STATUS REGISTER (IICSR)

The multi-master I^{2}C-bus control/status register, ICCSR, four bits, ICCSR.3–ICCSR.0, are read-only status flags. ICCSR register settings are used to control or monitor the following I^{2}C-bus functions (see Figure):

— I^{2}C-bus busy status flag

— Failed bus arbitration procedure status flag

— Slave address/address register match or general call received status flag

— Slave address 00000000B (general call) received status flag

— Last received bit status flag (not ACK = "1", ACK = "0")

Register	Address	R/W	Description	Reset Value
IICSR	0xB9	R/W	I^{2}C-bus status register	00h

[0]	Last-received bit (LRB) status flag (read only)	IICSR[0] is automatically set to 1 whenever an ACK signal is not received during a last bit receive operation. When the last receive bit is zero, an ACK signal is detected and the last-received bit status flag is cleared.
[1]	General call status flag (read only)	IICSR[1] is automatically set to 1 whenever '00000000B', general call value is issued by the received slave address. When the Start/stop condition was occurred, IICSR[1] is cleared.
[2]	Master address call status flag (read only)	IICSR[2] is automatically set to 1 whenever the received slave address matches the address value in IICADDR register. This bit is cleared after Start/stop condition is occurred.
[3]	Arbitration status flag (read only)	IICSR[3] is automatically set to 1 to indicate that a bus arbitration has been failed during I^{2}C-bus interface. The zero of IICSR[3] means okay status for the current I^{2}C-bus interface.
[4]	IIC operation status flag (read)	IICSR[4] is automatically set to 1 to indicate that the end of shifting for byte or stop condition is occurred. This bit is cleared when IIC operations are activated by writing IICCON.
	IIC interrupt source enable (write)	In write operation for this bit, this bit value determines that interrupt is enable or not to indicate the end of shifting for byte or stop condition is occurred. 0: No interrupt, pending 1: IIC interrupt
[5]	IIC-bus busy status (read-only)	IICSR[5] indicates that IIC-bus is not busy and the '1' status means IIC-bus is busy. This bit is set after start condition is detected and cleared after stop condition is occurred.
[7:6]	SCL/SDA digital filter selection	Setting this bit enables digital filtering on all two signal inputs: SCL and SDA. 00: Disable 01: 1 clock period 10: 2 clock period filtering 11: 3 clock period filtering

MULTI-MASTER I^2C-BUS TRANSMIT/RECEIVE DATA REGISTER (IICDATA)

In a transmit operation, data that is written to the IICDATA is transmitted serially, MSB first. (For receive operations, the input data is written into the IICDATA register LSB first.)

The IICCON.5 setting enables or disables serial transmit/receive operations. When IICCON.5 = "1", data can be written to the an I^2C data register. The I^2C-bus data register can, however, be read at any time, regardless of the current IICCON.5 setting.

Register	Address	R/W	Description	Reset Value
IICDATA	0xBA	R/W	I^2C-bus data register	xxh

[7:0] Data This data field acts as serial shift register and read buffer for interfacing to the I^2C-bus. All read and write operations to/from the I^2C-bus are done via this register. The IICDATA register is a combination of a shift register and a data buffer. 8-bit parallel data is always written to the shift register, and read from the data buffer. I^2C-bus data is always shifted in or out of the shift register.

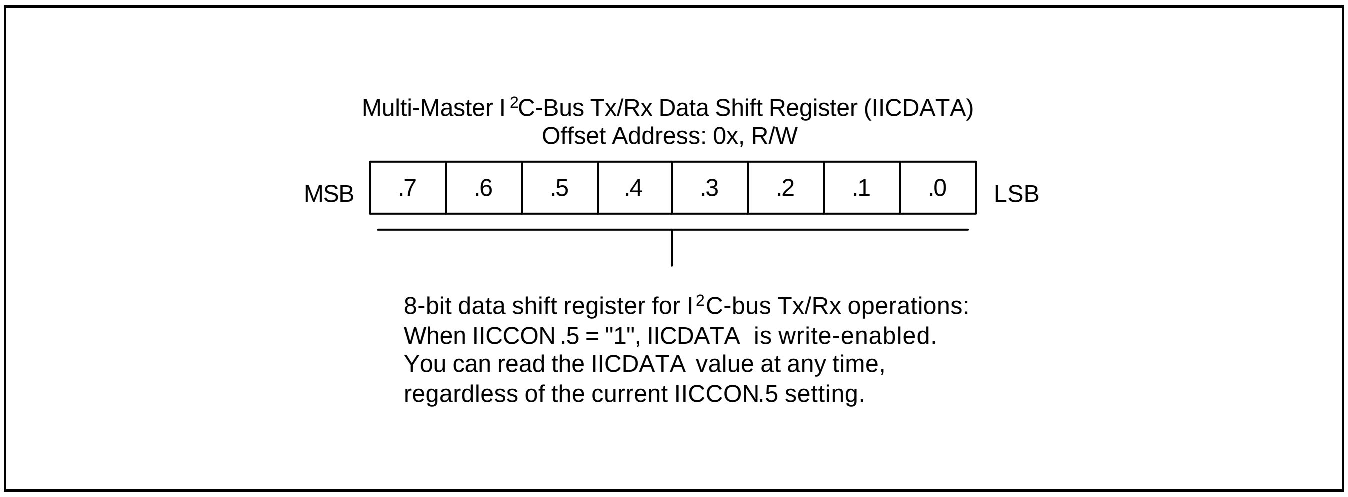

Multi-Master I^2C-Bus Tx/Rx Data Shift Register (IICDATA)
Offset Address: 0x, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
8-bit data shift register for I^2C-bus Tx/Rx operations:
When IICCON .5 = "1", IICDATA is write-enabled.
You can read the IICDATA value at any time,
regardless of the current IICCON.5 setting.

Figure 21-2. Multi-Master I^2C-Bus Tx/Rx Data Register (IICDATA)

MULTI-MASTER I^2C-BUS ADDRESS REGISTER (IICADDR)

The address register for the I^2C-bus interface, IICADDR, is located at address 0xBB. It is used to store a latched 7-bit slave address. This address is mapped to IICADDR.7–IICADDR.1; bit 0 is not used (see Figure 21-3).

The latched slave address is compared to the next received slave address. If a match condition is detected, and if the latched value is 00000000B, a general call status is detected.

Register	Address	R/W	Description	Reset Value
IICADDR	0xBB	R/W	I^2C-bus address register	xxh

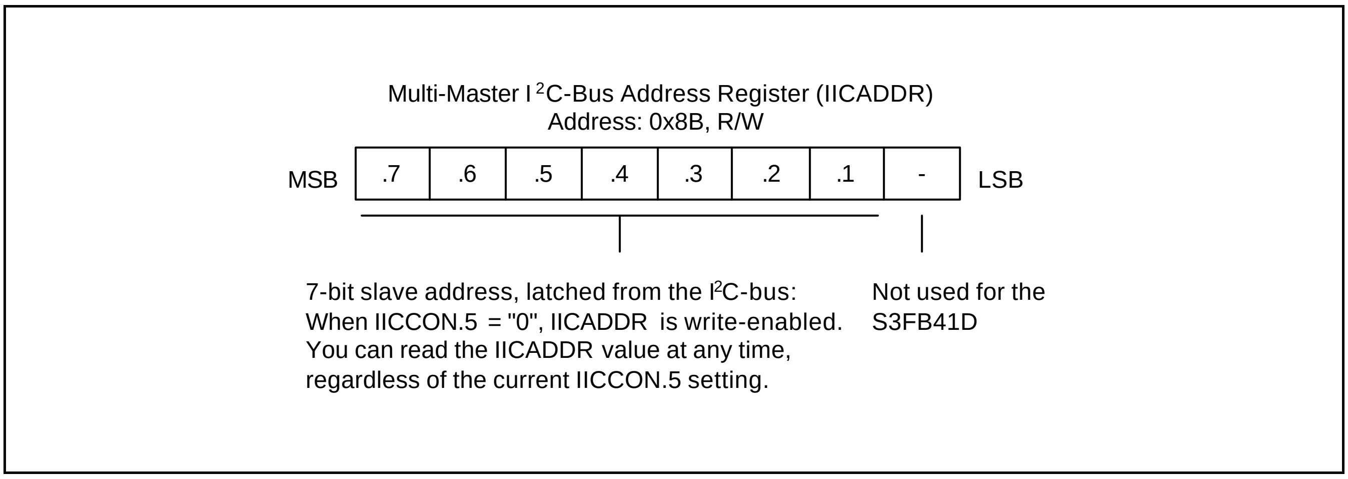

Multi-Master I²C-Bus Address Register (IICADDR)
Address: 0x8B, R/W
MSB
.7
.6
.5
.4
.3
.2
.1
-
LSB
7-bit slave address, latched from the I²C-bus:
When IICCON.5 = "0", IICADDR is write-enabled.
You can read the IICADDR value at any time,
regardless of the current IICCON.5 setting.
Not used for the
S3FB41D

Figure 21-3. Multi-Master I²C-Bus Address Register (IICADDR)

Prescaler Register (IICPS)

The prescaler register for the I²C-bus is described in the following table.

Register	Address	R/W	Description	Reset Value
IICPS	0xBC	R/W	I²C-bus Prescaler register	xxh

[7:0]	Prescaler value	This prescaler value is used to generate the serial I2C-bus clock. The system clock is divided by (4 × (prescaler value + 1)) to make the serial I²C clock. If the prescaler value is zero, I²C operation may be worked incorrectly.

PRESCALER COUNTER REGISTER (IICCNT)

The prescaler counter register for the I²C-bus is described in the following table.

Register	Address	R/W	Description	Reset Value
IICCNT	0xBD	R	I²C-bus Prescaler counter register	xxh

[7:0]	Prescaler counter value	This 8-bit value is the value of the prescaler counter. It is read (in test mode only) to check the counter's current value.

22 RANDOM NUMBER GENERATOR

OVERVIEW

The S3FB42F internal random number generator block has the following features:

— 2 ring oscillators, which run at –33MHz and –8MHz. They operate in a fully asynchronous manner with the CPU clock and are used as a clock to LFSR8.

— An 8-bit register LFSR8 is a linear feedback shift register, which changes its state with the clock from the ring oscillators. LFSR8 can serve as a source of the random number generation.

— A 16-bit register LFSR16 is a 16-bit linear feedback shift register, whose coefficients are provided by LFSR8. It changes its state when the lower byte is read.

— For the maximum flexibility, the programmers can use either LFSR8 or LFSR16 as the random number generator of their choice. If 16-bit or longer random numbers are required, LFSR16 can be preferred. Otherwise, LFSR8 can be a better choice.

Figure 22-1 shows a block diagram of the S3FB42F random number generator.

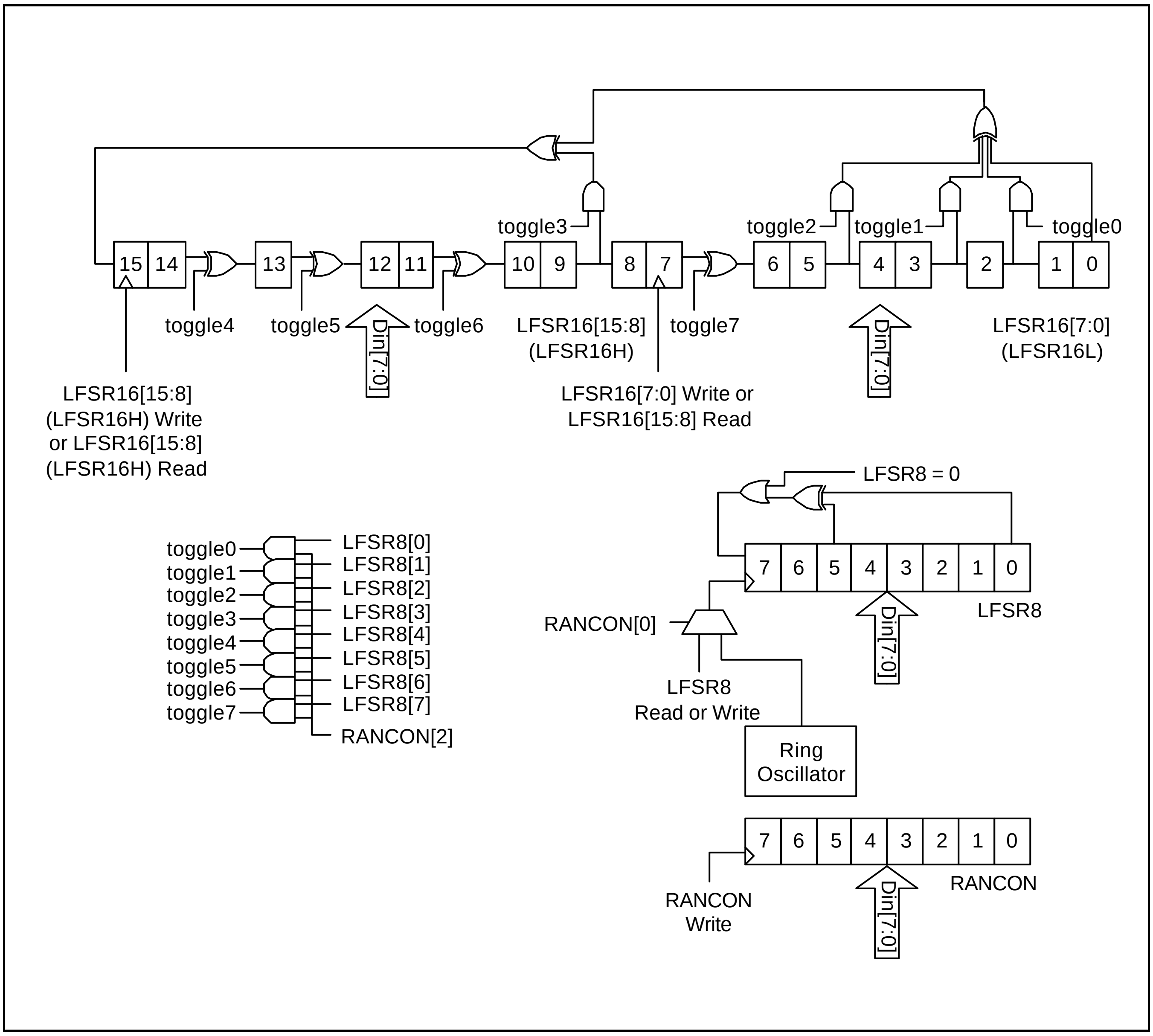


Figure 22 -1. Top Block Diagram of Random Number Generator

FUNCTIONAL DESCRIPTION

The S3FB42F random number generator has 4 registers, LFSR16[15:8] (LFSR16H), LFSR16[7:0] (LFSR16L), LFSR8, and RANCON, which are addressed by ABH, AAH, A9H, and A8H, respectively. For better randomness, it has two ring oscillators, the fast ring oscillator and the slow ring oscillator, which run at ~33MHz and ~8MHz, respectively.

RANDOM NUMBER CONTROL REGISTER

The random number control register, RANCON, is used to control the random number generator module.

Register	Address	R/W	Description	Reset Value
RANCON	0xA8	R/W	Control Register for Random Number Generation	xxh

[0]	LFSR8 Clock Selection or Ring Oscillator Disable	This bit is used to select the clock source for LFSR8. LFSR8 can be clocked either by the ring oscillator output or by the access signals from the core. When this bit is set, the ring oscillator output signal is tied to high. See the detailed description for LFSR8 0: Core R/W Signals 1: Ring Oscillator
[1]	Ring Oscillator Selection	The ring oscillator block consists of 2 ring oscillators, which run at ~33MHz and ~8MHz, respectively. This bit multiplexes the two ring oscillators to the ring oscillator output. 0: Fast Ring Oscillator 1: Slow Ring Oscillator
[2]	Polynomial Switch	If this bit is clear, the polynomial coefficients of LFSR16 is not affected by the value of LFSR8. Otherwise, the polynomial coefficients are determined by the value of LFSR8. Referring to the top block diagram, this serves as the mask bit for toggle0, toggle1, …, toggle7. 0: Toggle Bits Mask 1: Toggle Bits On
[3]	Test Bit	This bit is for Test Purpose. When RANCON[0] is set and the ring oscillator makes a rising transition, then this is set.
[4]	Test Bit	This bit is for Test Purpose. When RANCON[0] is set and the ring oscillator makes a falling transition, then this is set.
[5]	Slow Ring Off	If this bit value is set, the slow ring oscillator stops. 0: Slow Ring Oscillator Run 1: Slow Ring Oscillator Stop.
[6]	Fast Ring Off	If this bit value is set, the fast ring oscillator stops. 0: Fast Ring Oscillator Run 1: Fast Ring Oscillator Stop.
[7]	–	–

RING OSCILLATOR

The ring oscillator block consist of 2 ring oscillators, one of which runs at ~33MHz, which is called “fast ring oscillator” and the other runs at ~8MHz, which is called “slow ring oscillator”. The frequencies of the ring oscillators are determined by the gate size as well as the number of gates in the oscillator loop. Depending on a specific application, programmers can select the fast or the slow ring oscillators, which can serve the application best. Since the ring oscillators run totally asynchronously with the master clock of the chip, they can clock LFSR8m which, in turn, can be used as an 8-bit random number generator. The ring oscillators are laid out to be sensitive to fabrication conditions, the exact frequencies of the ring oscillators can vary from a chip to another. Each ring oscillator can be stopped by setting the corresponding control bit, Fast Ring Off or Slow Ring Off, in order to save power consumption.

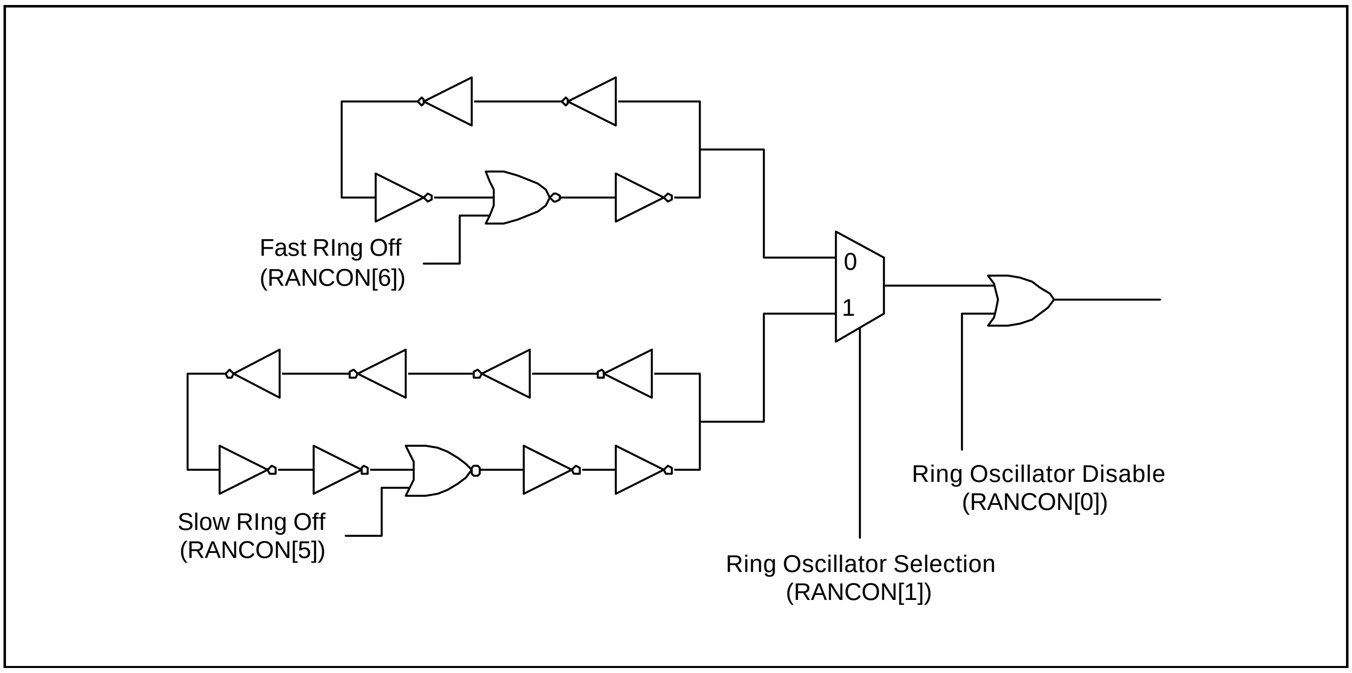


Figure 22-2. Ring Oscillator Block

LINEAR FEEDBACK SHIFT REGISTER 8 (LFSR8)

LFSR8 is a register for generating 8-bit random numbers. When RANCON[0] (LFSR8 Clock Selection) is set, LFSR8 is linear feedback shifted at the rising edge of the ring oscillator output. When RANCON[0] is clear, the core (CalmRISC) can parallel load the data through the input data bus (Din[7:0]) by writing the data to the address 0x92h. Also the core can read the contents of LFSR8 by reading the address 0xA9, regardless of the value of RANCON[0]. Note that when the core reads in the contents of LFSR8, a single linear feedback shift operation is performed right after the read operation if RANCON[0] = 0.

Register	Address	R/W	Description	Reset Value
LFSR8	0xA9	R/W	8-bit linear feedback shift register	xxh

LFSR8[7:0]	When RANCON[0] = 1, a linear feedback shift operation is performed at the rising edge of the ring oscillator output. In this case, the core (CalmRISC core) cannot write data into LFSR8.

When RANCON[0] = 0, the core can write data into LFSR8 by a load instruction to the address 0xA9. A read operation on LFSR8 is automatically followed by a linear feedback shift operation.

NOTE:
When RANCON[0] = 1, a write operation by the core has no effect and a linear feedback operation does not automatically ensue after a core read operation.

LINEAR FEEDBACK SHIFT REGISTER 16 (LFSR16)

LFSR16 is a 16-bit linear feedback shift register, which can be parallel loaded through a core write operation or linear feedback shifted by a core read operation on LFSR[15:8]. The polynomial coefficients of LFSR is determined by the value of LFSR8, only when RANCON[2](Polynomial Switch) is set. Otherwise, the polynomial coefficient is fixed such that LFSR16 performs a simple rotate operation.

Register	Address	R/W	Description	Reset Value
LFSR16[15:8] (LFSR16H)	0xAB	R/W	16-bit linear feedback shift register	xxh
LFSR16[7:0] (LFSR16L)	0xAA	R/W		xxh

LFSR16[15:8] and LFSR16[7:0]	LFSR16H[15:8] and LFSR16L[7:0] can be individually read and loaded. A linear feedback shift operation is performed on LFSR16[15:0] right after LFSR16H[15:8] is read. The linear feedback shift operations follow the rule below:

RANCON[2]=0	A simple rotate operation is executed.
	LFSR16[15] = LFSR16[0]
	LFSR16[14] = LFSR16[15]
	• • •
	LFSR16[0] = LFSR16[1]

RANCON[2]=1	A linear feedback operation is executed.
	LFSR16[15] = (toggle0&LFSR16[2])^(toggle1&LFSR16[3])^
	(toggle1&LFSR16[5])^(toggle1&LFSR16[9])^
	LFSR16[0]
	LFSR16[14] = LFSR16[15]
	LFSR16[13] = toggle4^ LFSR16[14]
	LFSR16[12] = toggle5^ LFSR16[13]
	LFSR16[11] = LFSR16[12]
	LFSR16[10] = toggle6^ LFSR16[11]
	LFSR16[9] = LFSR16[10]
	LFSR16[8] = LFSR16[9]
	LFSR16[7] = LFSR16[8]
	LFSR16[6] = toggle7^ LFSR16[7]
	LFSR16[5] = LFSR16[6]
	LFSR16[4] = LFSR16[5]
	LFSR16[3] = LFSR16[4]
	LFSR16[2] = LFSR16[3]
	LFSR16[1] = LFSR16[2]
	LFSR16[0] = LFSR16[1]
	Where
	toggle0 = LFSR8[0]&RANCON[2]
	:
	:
	toggle7 = LFSR8[7]&RANCON[2]

23 USB

USB PERIPHERAL FEATURES

Table 23-1. General USB Features

Complete USB Specification	yes
On-chip USB transceivers	yes
Automatic transmit/receive FIFO management	yes
Suspend/resume	yes
USB rate (full speed)	12 Mbps
USB interrupt vectors	yes

Table 23-2. General Function Features

Control Endpoint	1
Data endpoints	3
FIFO sizes Endpoint 0 Endpoint 1 Endpoint 2 Endpoint 3	 16 bytes 32 bytes 64 bytes 64 bytes
Direction of data endpoints	In/Out
Supported transfer of data endpoints	Interrupt/Bulk/Isochronous transfer

FUNCTIONAL SPECIFICATION

— Power Management

— General purpose Full Speed Controller

— Each data endpoints support interrupt, bulk and isochronous transfer

— Protocol handling in hardware

— Built-in Full-Speed tranceiver

The basic blocks are the Serial Interface Engine (SIE), MCU Interface Unit (MIU), Function Interface Unit (FIU) and SIE Interface Unit (SIU).

USB MODULE BLOCK DIAGRAM

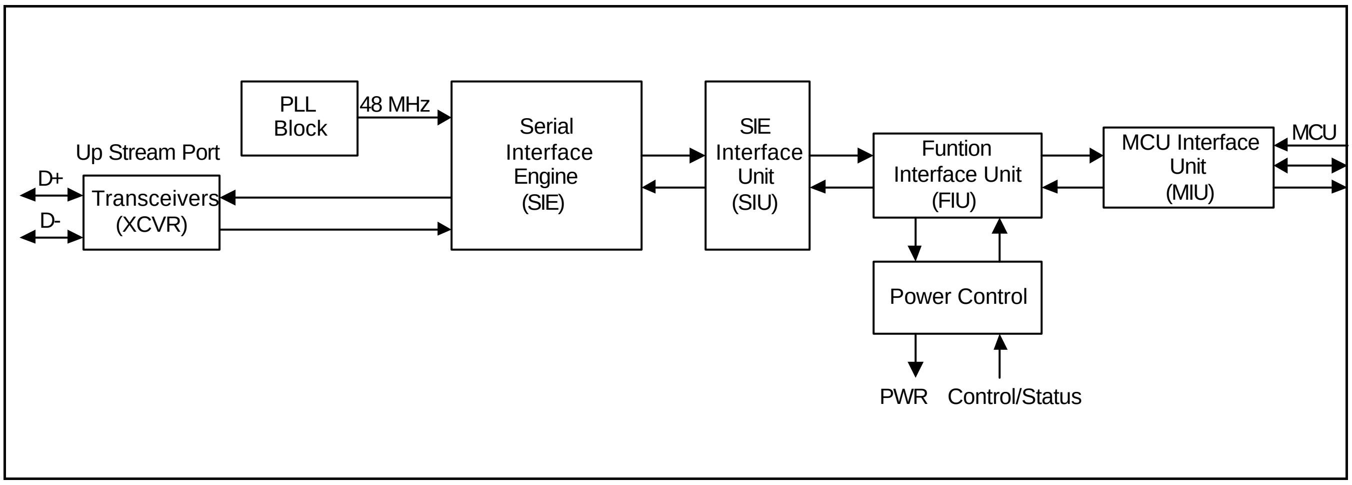


Figure 23-1. USB Module Block Diagram

FUNCTION DESCRIPTION

Transceivers (XCVR)

The transceiver consists of a differential receiver, two single ended receivers and two drivers. That is capable of transmitting and receiving data at 12 Mbit/sec and 1.5 Mbit/sec meeting the USB requirements.

Serial Interface Engine (SIE)

The Serial Interface Engine implements the protocol layer of the USB. It does the clock recovery, error checking, data conversion between serial and parallel data, do the handshake on the USB bus if the packet was directed to it, bus timeout if response from the host is late, and all other USB protocol related functions.

It consists of Phase Locked Loop (PLL) for clock recovery from the incoming data, CRC checker and generator, bit stuff and bit removal logic, NRZI encoder/decoder, shift register for serial/parallel conversion, PID decoder, data toggler and sync detect logic.

SIE Interface Unit (SIU)

The SIE Interface Unit interfaces with SIE to get the parallel data and pass on to the FIU. Other important function of the SIU is to compare the device and endpoint address in the token packet with the valid device and endpoint addresses from the embedded function, and generate a address valid signal to the SIE so it can complete the handshake to FIU so they can get started waiting for the data phase.

Function Interface Unit (FIU)

Function Interface Unit consists of Endpoint0 and three additional endpoints for the embedded function. The Endpoint0 logic consists of 16 byte bi-directional FIFO and all the control logic necessary to interface with the SIU on one side and with the MCU interface logic on the other side. The control logic keeps track of data toggle bit in a multiple packet transaction and resend of the data when the request is retried by the host. It handles the setting and clearing of the endpoint stall bit. The OUT/SETUP data from the FIFO is read by the MCU interface and data for the IN is loaded into the FIFO by MCU interface.

The three additional endpoints are programmable as In or Out endpoint, and they can be interrupt, bulk or isochronous types. Each endpoint consists of 32, and 64 byte bi-directional FIFOs used in one direction only with direction programmed via a control bit in their respective CSR register. The data transfers between the MCU and the FIFOs are controlled by setting/clearing bits in the CSR. Interrupt may be generated on occurrence of some significant events and this interrupt can be disabled by the firmware.

MCU Interface Unit (MIU)

This block of logic will allow the MCU to interface to the FIU units. This block will handle the MCU timing, address decoding and data multiplexing.

Suspend/Resume:

The suspend timer is used to detect inactivity on the upstream port. If no SOF is received for more than 3 ms device enters a suspend state and SUSPEND signal is asserted. On detecting SUSPEND, STOP_CLK signal can be asserted by the MCU (or external hardware) to stop the clock in the USB block.

When resume is detected by the upstream port control logic the SUSPEND signal is removed and suspend state is reset at the end of resume.

MCU can also do a remote wakeup by asserting RESUME_IN to the USB block.

MCU Programming:

MCU firmware need to support the Function Unit completely, all the traffic related to the embedded port will be relayed to the MCU by the USB block.

The Host commands supported by the Function Unit will depend on the device firmware is implementing e.g. in monitor application HID class besides the required standard commands need to be supported.

The USB block presents number of registers to the MCU for controlling, monitoring and data transfers.

USB FUNCTION REGISTERS DESCRIPTION

Table 23-3. USB Function Registers Description

Register Name	ADDR	R/W/C	Description
FUNADDR	80H	R/W	Function Address Register
PWRMAN	81H	R/W	Power Management Register
FRAMELO	82H	R	Frame Number LO Register
FRAMEHI	83H	R	Frame Number HI Register
INTREG	84H	R/W	Interrupt Pending Register
INTENA	85H	R/W	Interrupt Enable Register
EPINDEX	86H	R/W	Endpoint Index Register
EPDIR	89H	W	Endpoint Direction Register
INCSR	8AH	R/W	IN Control Status Register
OUTCSR	8BH	R/W	OUT Control Status Register
INMAXP	8CH	R/W	IN MAX Packet Register
OUTMAXP	8DH	R/W	OUT MAX Packet Register
WRTCNTLO	8EH	R/W	Write Counter LO Register
WRTCNTHI	8FH	R/W	Write Counter HI Register
EP0FIFO	90H	R/W	Endpoint 0 FIFO Register
EP1FIFO	91H	R/W	Endpoint 1 FIFO Register
EP2FIFO	92H	R/W	Endpoint 2 FIFO Register
EP3FIFO	93H	R/W	Endpoint 3 FIFO Register
USBENA	9EH	R/W	USB Enable Register

USB RELEATED REGISTERS

Some of the registers in the USB function unit are similar, specially pertaining to the endpoints. Hence the description of those registers will be presented only once here in the USB Related Registers to avoid duplication and avoid keep both sets updated.

Function Address Register

Register	Address	R/W	Description	Reset Value
FUNADDR	0x80	R/W	Function address register	00h

At reset the address is 00h. After the SET_ADDRESS is received by the MCU, it should load the address received into this register. This register is enabled for address comparison after the "Status" phase of the SET_ADDRESS control transfer. This is so that the status IN packet which will still have "0" address can to be recognized for this embedded function by the hardware in the SIU. This register should be loaded before setting DATAEND and clearing OUTPKTRDY in the EP0 CSR.

This register is cleared by the core when port reset is received from the host for the embedded port or when USB_RESET has been received.

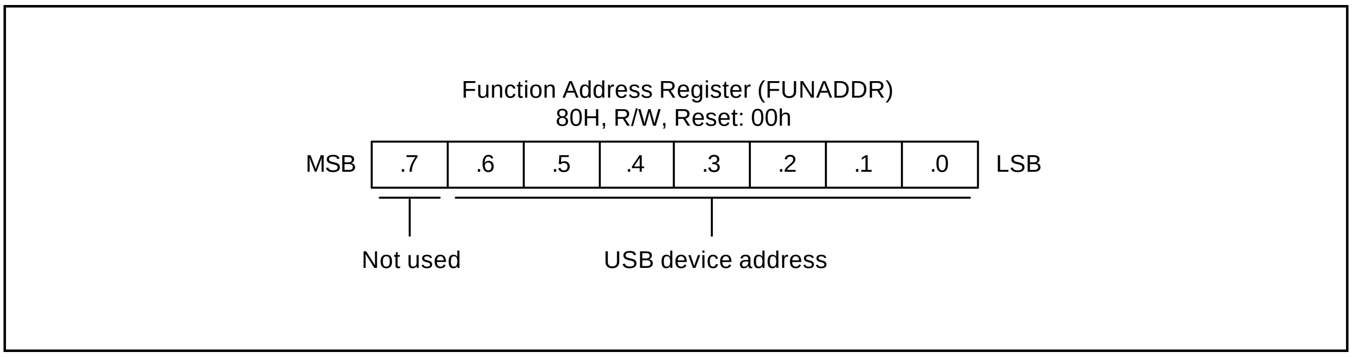


Figure 23-2. Function Address Register

Power Management Register

This Register is used for power management in the function controller core.

Register	Address	R/W	Description	Reset Value
PWRMAN	0x81	R/W	Power Management register	00h

SUSPEND: When the function receives a suspend signaling, the function controller core sets this bit. This also generates an interrupt to the microcontroller. Upon seeing this bit set, the microcontroller can store its internal register and enter suspend mode, disabling the clock of the fucntion controller core.

UC_RESUME: When the microcontroller is awakend by keyboard stroke or mouse movement, it starts its wakeup sequence and sets this bit. While this bit is set and the function is in suspend mode, the function generate a resume signaling as long as this bit for a 10 to 15ms duration to start the resume signaling, After the resume signaling, the microcontroller can clear both the SUSPEND and SEND_RESUME bits.

USB_RESUME: When the function controller core is in suspend mode and recieves resume signaling this bit is set and an interrupt is generated. The microcontroller, se this bit set, can start wake-up sequence

USB_RESTN: The function contoller core sets this bit, if reset signaling is received from the host.

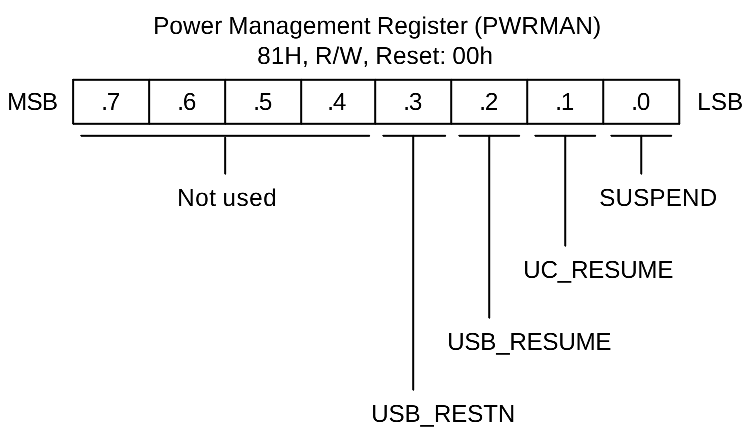


Figure 23-3. Power Management Register

Frame Number Register

Register	Address	R/W	Description	Reset Value
FRAMELO	0x82	R	Frame number low register	00h
FRAMEHI	0x83	R	Frame number high register	00h

On detection of SOF from the host, this register is updated with the frame number received with the SOF packet.

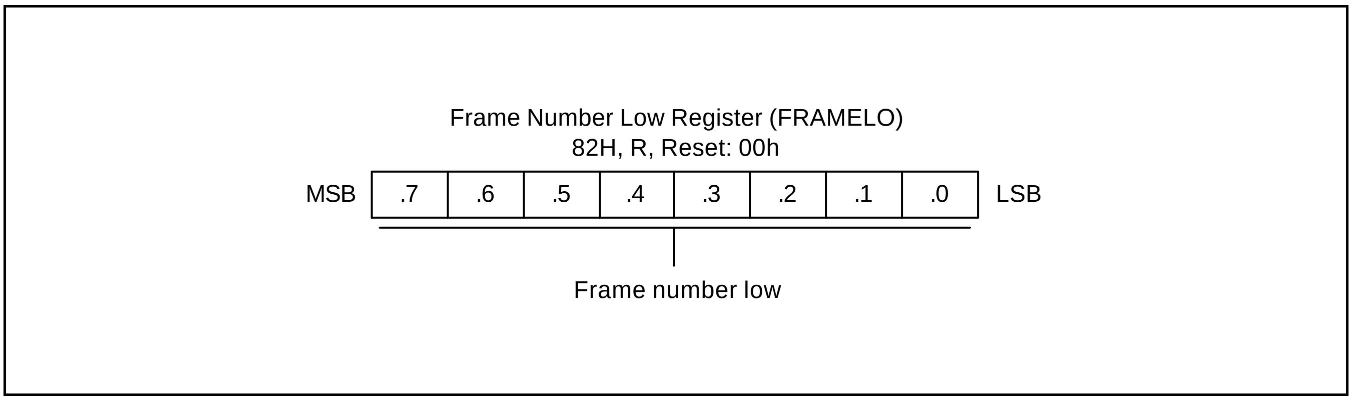


Figure 23-4. Frame Number Low Register

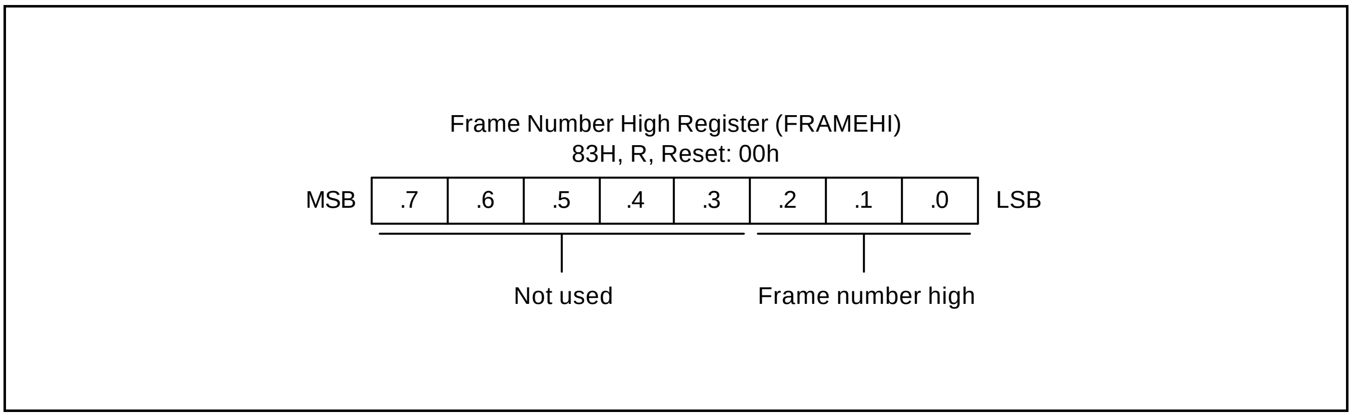


Figure 23-5. Frame Number High Register

Interrupt Pending Register

Register	Address	R/W	Description	Reset Value
INTREG	0x84	R/W	Interrupt pending register	00h

This register is used to indicate the condition that sent and interrupt to the microcontroller.

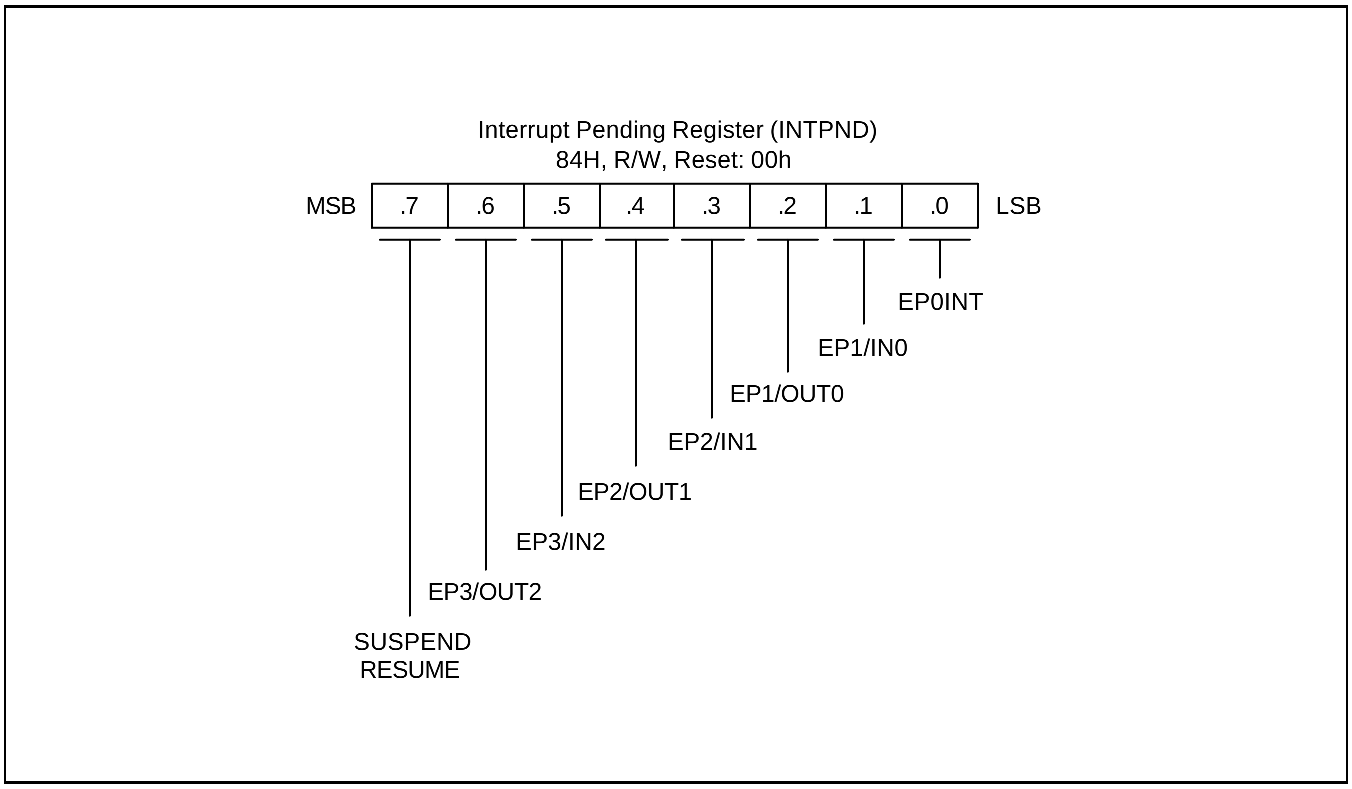


Figure 23-6. Interrupt Pending Register

Table 23-4. Interrupt Pending Register

Bit Description	USB	MCU	Condition for Interrupt
EP0 (CONTROL)	W	R/C	The USB sets this bit under the following conditions. 1.OUT_PKT_RDY is set 2.IN_PKT_RDY is set 3.SENT_STALL is cleared 4.IN_PKT_RDY is cleared
EP1/IN0	W	R/C	The USB sets this bit under the following conditions. IN_PKT_RDY is cleared
EP1/OUT0	W	R/C	The USB sets this bit under the following conditions. 1.Set OUT_PKT 2.Set FORCE_STALL
ENDPT2/IN1	W	R/C	The USB sets this bit under the following conditions. IN_PKT_RDY is cleared
EP2/OUT1	W	R/C	The USB sets this bit under the following conditions. 1.Set OUT_PKT 2.Set FORCE_STALL
EP3/IN2	W	R/C	The USB sets this bit under the following conditions. IN_PKT_RDY is cleared
EP3/OUT2	W	R/C	The USB sets this bit under the following conditions. 1.Set OUT_PKT 2.Set FORCE_STALL

Interrupt Enable Register

Register	Address	R/W	Description	Reset Value
INTENA	0x85	R/W	Interrupt enable register	00h

This register serves as interrupt mask register. If the corresponding bit = 0 then the respective interrupt is disabled, and when = 1 interrupt is enabled. By default upon reset, all the interrupts are disabled. If an interrupt is being serviced firmware may want to mask the interrupt by masking the corresponding bit(s) or when certain interrupt status bits are going to be polled.

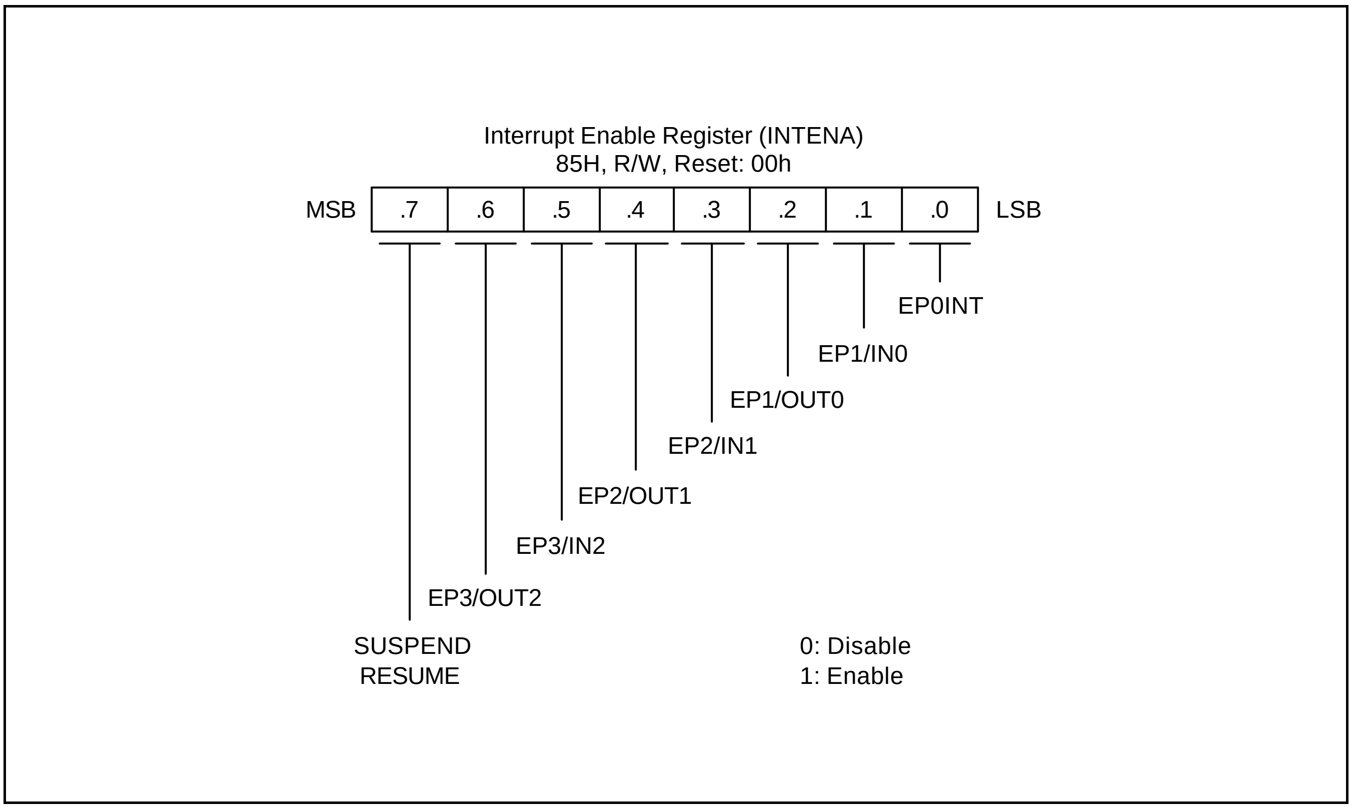


Figure 23-7. Interrupt Enable Register

Endpoint Index Register

Register	Address	R/W	Description	Reset Value
EPINDEX	0x86	R/W	Endpoint index register	00h

If ISO_UPDATE bits is set, Isocronous transaction is enabled in endpoint 1-6. Endpoint 0-3 registers (IN_CSR,OUT_CSR, CNT, MAXP) share the same address space. To select between them, Endpoint Register is provided and MCU can load the register. The buffer data is available for each endpoint at unique addresses and are independent of the FUNC_EP_SEL bits.

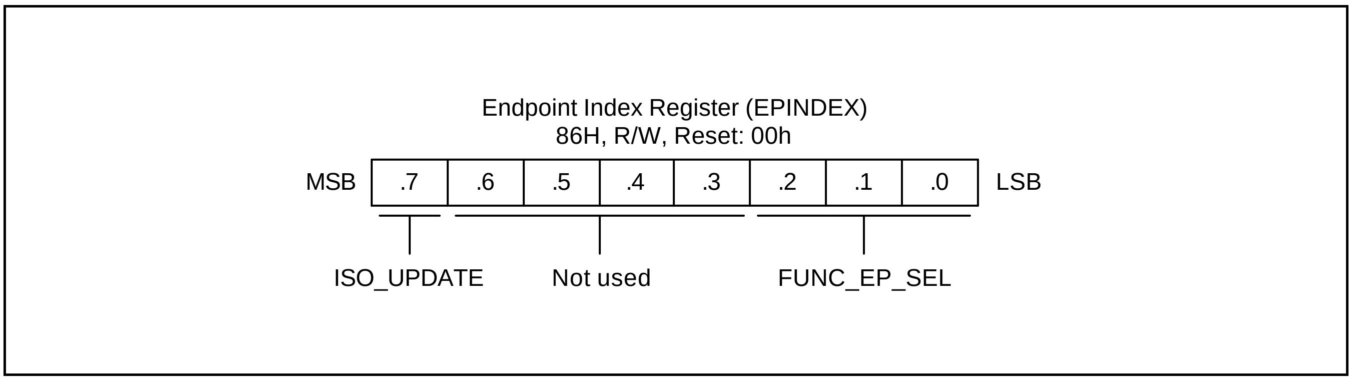


Figure 23-8. Endpoint Index Register

Endpoint Direction Register

Register	Address	R/W	Description	Reset Value
EPDIR	0x89	W	Endpoint direction register	00h

If ENDPOINTX DIRECTION Bit is 1, ENDPOINTX is for IN Transaction else ENDPOINTX is for OUT Transaction.

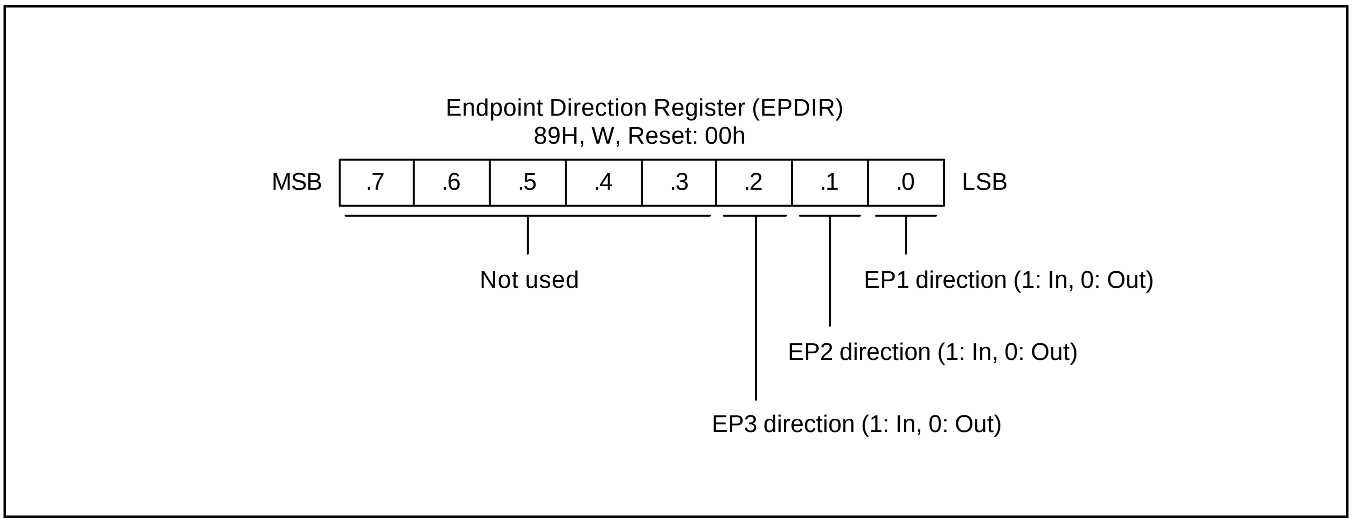

Endpoint Direction Register (EPDIR)
89H, W, Reset: 00h
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Not used
EP1 direction (1: In, 0: Out)
EP2 direction (1: In, 0: Out)
EP3 direction (1: In, 0: Out)

Figure 23-9. Endpoint Direction Register

ENDP POINT 0 Control Status Register

When EP Select register is equal to zero, Endpoint Out CSR and Endpoint In CSR select the same Endpoint0 CSR. Reading and writing to either address accesses the same register. This Register has Control and status bits for EP0, Since control transactions involve both IN and OUT token.

Register	Address	R/W	Description	Reset Value
EP0CSR	0x8A, 0x8B	R/W	EP0 CSR register	00h

EP0	USB	MCU	Description
OUT_PKT_RDY	W	R/C	Packet received from the Host is ready in the FIFO
IN_PKT_RDY	R/C	R/W	Packet to be sent to the Host is ready in the FIFO
SENT_STALL	W	R/C	USB sent a stall handshake to the Host.
DATA_END	R/C	R/W	set by MCU when last data is loaded in FIFO or no Data is needed by the command
SETUP_END	W	R/C	Set when current control transaction need to be aborted.
FORCE_STALL	R/C	R/W	Force a stall handshake to the Host(Write Only?)
CLR_OUT_PKT_RDY	R	W	Clear the OUT PKT RDY bit.
CLR_SETUP_END	R	W	Clear the SETUPEND bit.

OUT_PKT_RDY: The GFI sets this bits, whenever it has a valid token packet in the endpt0 FIFO. The micro controller seeing this bit set, unloads the FIFO and clears this bit. If it is the SETUP phase, then the micro controller also decodes the SETUP token, and checks to see if it is a valid command and, then clears this bit by doing CLR_EP0_OUTPKTRDY. At the time of clearing this bit, the micro controller will also set FORCE_STALL if it is a invalid command, and DATA END if the length of data transfer during data phase is zero (no DATA phase, viz., SET_ADDRESS).

IN_PKT_RDY: The micro controller after filling the FIFO with a IN data, set this bit. MCU should wait for this bit to be cleared by the GFI before loading next IN token. If the function receives a valid IN token, while IN PKT RDY is not set by the micro controller then the endpt0 state machine issues a NAK and shake.

SENT_STALL: When the hardware decodes an illegal sequence from the host, it may send a STALL on its own to the USB host. This bit is set to inform the MCU that such an event has happened. This is informational only and does not cause an interrupt, and it needs to be cleared by the MCU after it has seen it.

DATA_END: During the DATA phase of a control transfer, after the micro controller has finished loading/unloading the exact number of bytes as specified in the SETUP phase, it sets this bit.

FORCE_STALL: When an illegal or unsupported command is decoded by the firmware it needs to set this bit. When set, this bit causes the hardware to return a STALL handshake to the host and is reset by the hardware when handshake has been sent. This bit should not be set when host does a SET_FEATURE STALL, as this will cause all transfers to/from endpoint 0 to return STALL. This behavior is different for other endpoints.

Once the micro controller sees this bit set, it should end the SETUP phase, stop loading/unloading the FIFO (NOTE: Micro Controller does not set DATA_END in this case). The GFI before setting this bit flushes the FIFO, and prevents micro controller accesses to the FIFO.

CLR_OUTPKT_RDY: Write a *1* to this bit to clear the out packet ready. This bit not sticky, It can be set in conjunction with other bits.

CLR_SETUP_END: Write a *1* to this bit to clear SETUP_END bit. This again is not sticky and can be set together with other bits

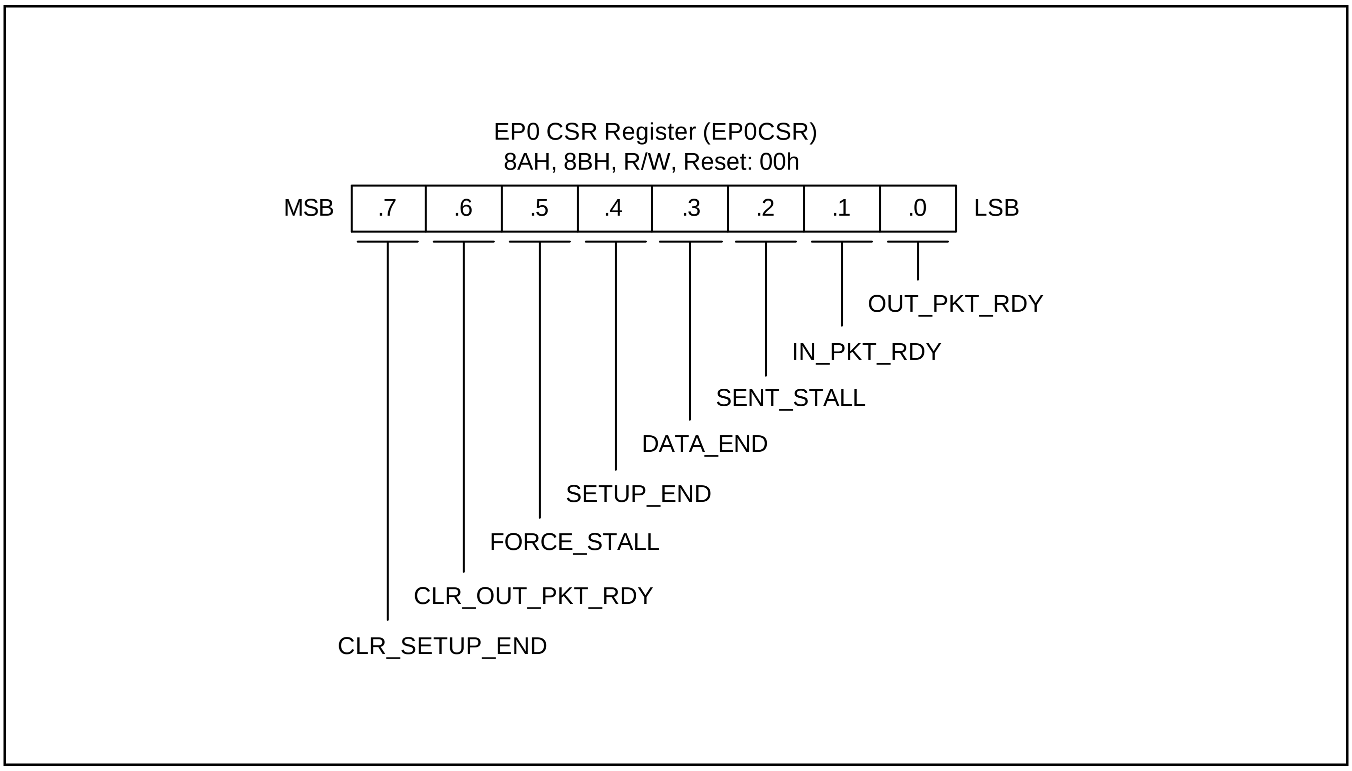


Figure 23-10. EP0 CSR Register (EP0CSR)

IN Control Status Register

For Endpoints other than 0, separate in and out registers are available and micro-code should read the register the endpoint has been programmed for.

Register	Address	R/W	Description	Reset Value
INCSR	0x8A	R/W	IN control status register	00h

ENDPT1CSR BIT	USB	MCU	Description
IN_PKT_RDY	R/C	R/W	When packet has been load into FIFO by MCU and ready for transfer to the host, write '1' to this bit.
UNDERRUN	W	R/C	USB under-run error during ISO.
FORCE STALL	R/C	R/W	Force a stall handshake to the host.
ISO	R	R/W	if set, indicates an isocronous endpoint.
INTPT_ENDPT	R	R/W	if set, USB sends packet whatever data is there in the FIFO.
IN_PKT_RDY2	R/C	R	When MCU writes a '1' to bit 0,this bit is always set. It is cleared by the USB when the data has been transferred to the host.
FIFO_FLUSH	R/C	W	The MCU sets this bit if it intends to flush the IN FIFO. This bit is cleared by the USB when the FIFO is flushed. The MCU is interrupted when this happens. If a token is in progress, the USB waits until the transmissions in complete before the FIFO is flushed.
CLR_DATA_TOGGLE	R	W	When the MCU writes a 1 to this bit, the data toggle bit is cleared. This is a write-only.

IN_PKT_RDY: The micro controller after filling the FIFO with a IN data, set this bit. MCU should wait for this bit to be cleared by the USB before loading next IN token. If the function receives a valid IN token, while IN PKT RDY is not set by the micro controller then the endpt state machine issues a NAK handshake.

UNDER RUN: This bit is used to isocronous endpoints. It is set if the function times out to an IN token.

ISO: if this bit is set, the endpoint behaves as an isocronous endpoint.

FORCE_STALL: This bit is set by the micro controller. Whenever this bit is set, the function controller issues a STALL handshake to the host. This bit may be set by the MCU for any fault condition within the function or when host does a SET_FEATURE (ENDPOINT_STALL). It is cleared by the micro controller when it receives a CLEAR_FEATURE (ENDPOINT_STALL) command from the host.

IN_PKT_RDY2: When MCU writes a *1* to bit 0 position, this bit always gets set and is cleared by the hardware when all the packets have been transferred to the host.

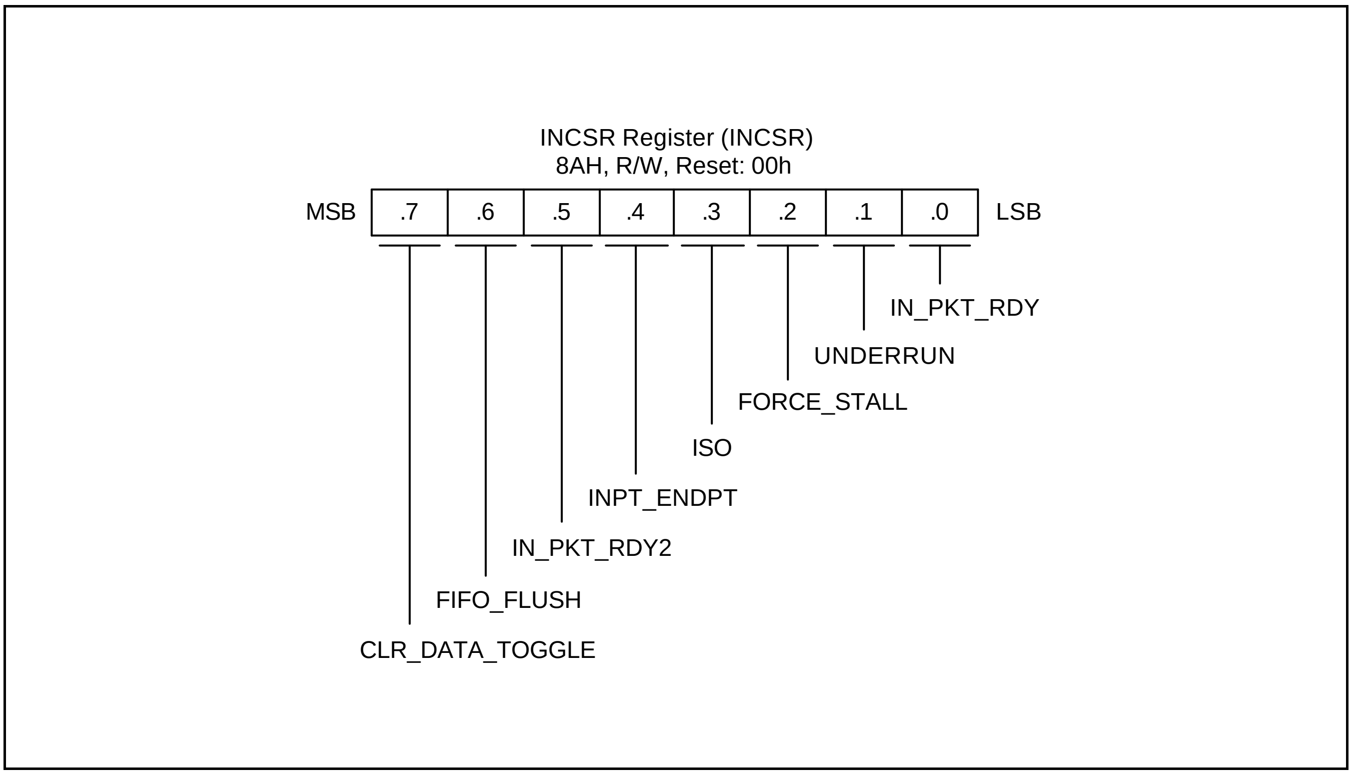


Figure 23-11. INCSR Register

Out Control Status Register

Register	Address	R/W	Description	Reset Value
OUTCSR	0x8B	R/W	OUT control status register	00h

OUT CSR BIT	USB	MCU	Description
OUT_PKT_RDY	W	R/C	Packet received from the host is ready in the FIFO
OVERRUN	W	R/C	This is set only in ISO mode. USB sets this bit when overrun is detected.
SEND_STALL	R/C	R/W	MCU forces a stall handshake to the host.
FORCE_STALL	W	R/C	USB sets this bit when OUT transaction ended with STALL handshake. This happens when: 1. Host sends more then MAXP data. 2. USB detected protocol violation.
ISO	R	R/W	if set, indicates an isocronous endpoint.
DATA_ERR	W	R/C	This is set only in ISO mode. USB sets this bit if at the time of setting OUTPKTRDY if an error has occurred.

OUT_PKT_RDY: The GFI sets this bits, whenever it has a valid token packet in the endpt1 FIFO. The micro controller seeing this bit set, unloads the FIFO and clears this bit by doing writing a *1* to this bit. At the time of clearing this bit, the micro controller should also set SEND_STALL if a stall condition exists.

OVERRUN: This is used for isochronous endpoints only, if an OUT token packet is receved and the out_pkt_rdy from the pervious transactions is not cleared, the USB discard the data and set this bit to indicate to the micro controller that an OUT packet was lost.

SEND_STALL: This bit is set by the micro controller. Whenever this bit is set, the function controller issues a STALL handshake to the host.

This bit may be set by the MCU for any fault condition within the function or when host does a SET_FEATURE(ENDPOINT_STALL). It is cleared by the micro controller when it receives a CLEAR_FEATURE(ENDPOINT_STALL) command from the host.

ISO: if this bit is set, the endpoint behaves as an isocronous endpoint.
If this bit is dear, the endpoint be haves as a bulk or interrupt endpoint.

FORCE_STALL: USB sets this bit when OUT transaction ended with STALL handshake.
This happens when:
1. Host sends more than MAXP data.
2. USB detected protocol violation.

DATA_ERR: For ISO endpoint , HW sets OUT PKT RDY even if the core has a CRC/bit stuffing error. But DATA_ERR bit is also set in this case. If the microcode is capable of error recovery it can unload the packet, else it can flush the FIFO, which will clear out the FIFO and reset OUT PKT RDY.

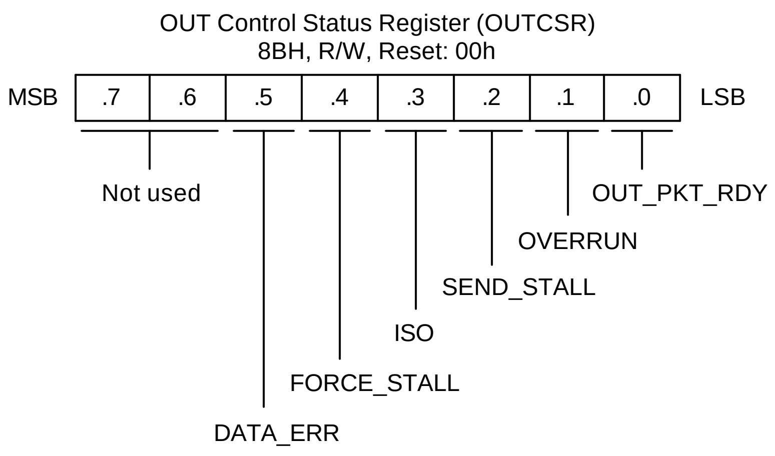


Figure 23-12. OUT Control Status Register

IN MAX Packet Register

Register	Address	R/W	Description	Reset Value
INMAXP	0x8C	R/W	IN MAX packet register	00h

NAME	USB	MCU	Description
MAXP	R	R/W	0000 MAXP = 0 0001 MAXP = 8 0010 MAXP = 16 0011 MAXP = 24 0100 MAXP = 32 0101 MAXP = 40 0110 MAXP = 48 0111 MAXP = 56 1000 MAXP = 64

This register has Maximum packet size for the IN endpoint. The packet is selectable in multiple of 8 byte.

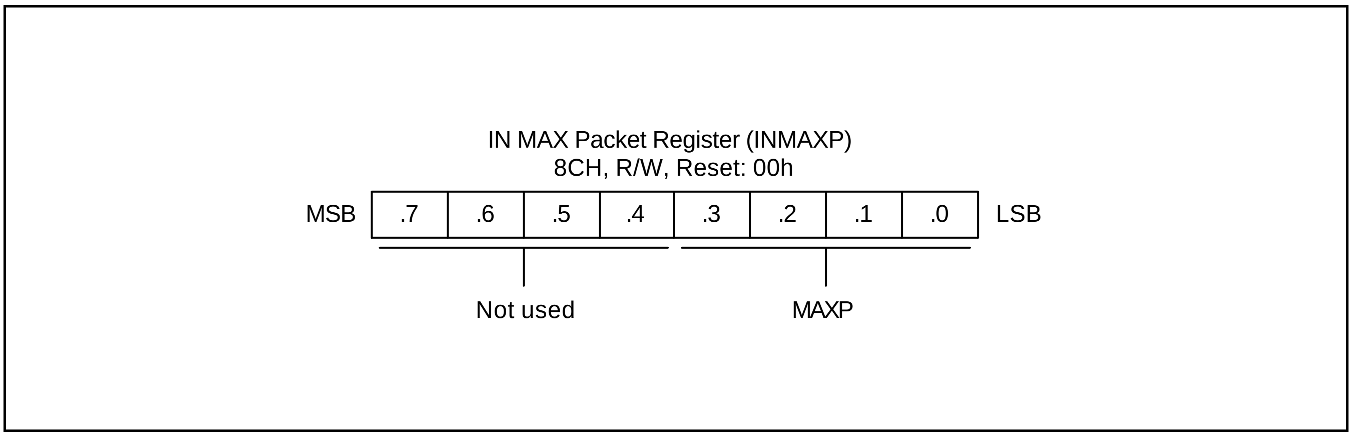


Figure 23-13. IN MAX Packet Register (INMAXP)

OUT MAX Packet Register

Register	Address	R/W	Description	Reset Value
OUTMAXP	0x8D	R/W	OUT MAX pocket register	00h

NAME	USB	MCU	Description
MAXP	R	R/W	0000 MAXP = 0 0001 MAXP = 8 0010 MAXP = 16 0011 MAXP = 24 0100 MAXP = 32 0101 MAXP = 40 0110 MAXP = 48 0111 MAXP = 56 1000 MAXP = 64

This register has Maximum packet size for the OUT endpoint. The packet is selectable in multiple of 8byte.

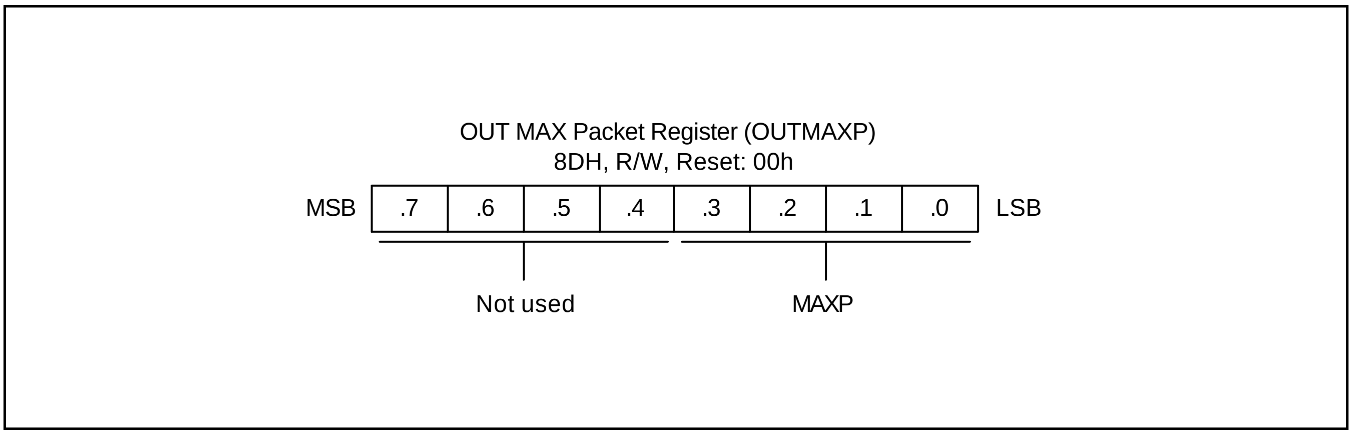


Figure 23-14. OUT MAX Packet Register

EP0 MAX Packet Register

Register	Address	R/W	Description	Reset Value
EP0MAXP	0x8C, 0x8D	R/W	EP0 MAX packet regsiter	00h

NAME	USB	MCU	Description
MAXP	R	R/W	00 MAXP = 8 01 MAXP = 16

This register has Maximum packet size for the Endpoint0. The packet is selected as either 8 or 16 bytes.

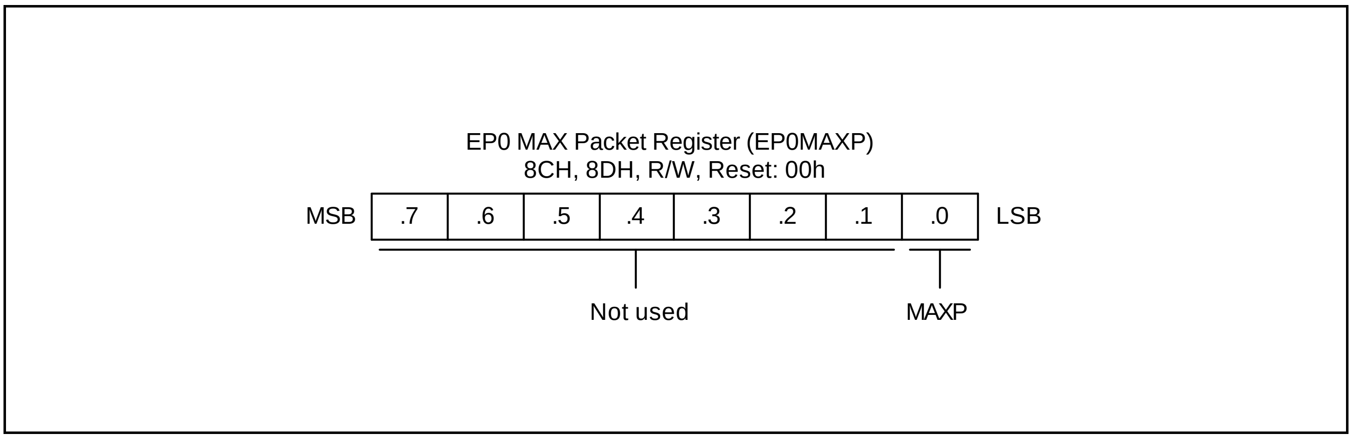


Figure 23-15. EP0 MAX Packet Register

Write Counter Register

Register	Address	R/W	Description	Reset Value
WRTCNTLO	0x8E	R/W	Write counter low register	00h
WRTCNTHI	0x8F	R/W	Write counter high register	00h

when OUT_PKT_RDY is set for EPX, this register maintains the number of bytes in the EPX_OUT_FIFO.

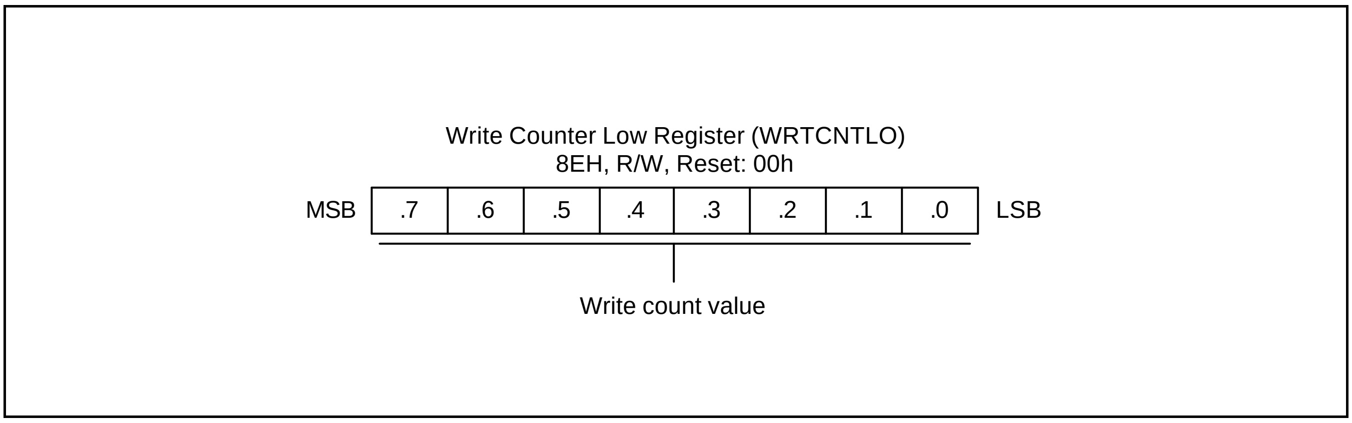


Figure 23-16. Write Counter LO Regsiter

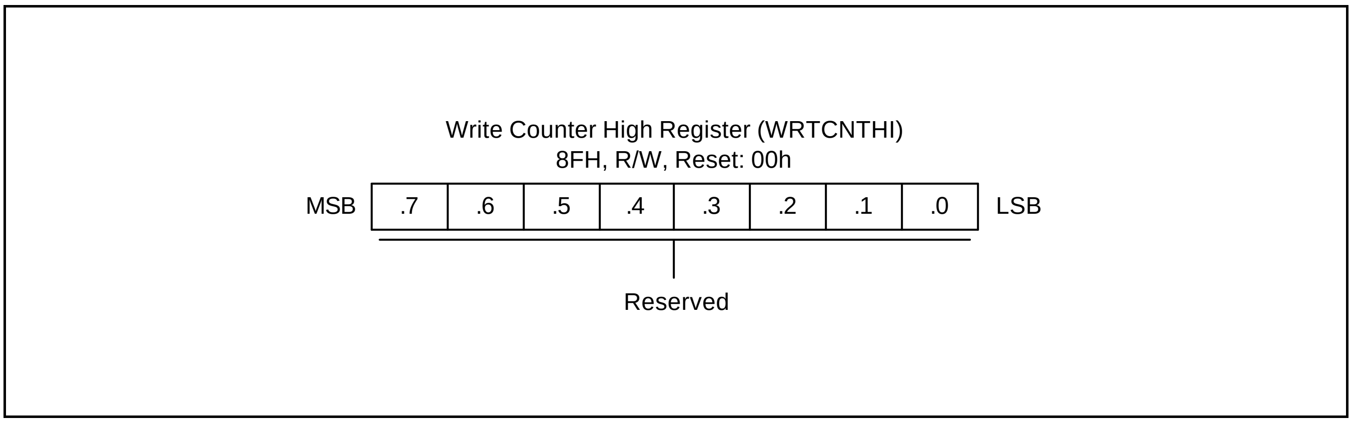


Figure 23-17. Write Counter HI Register

ENDPOINT0 FIFO Register

Register	Address	R/W	Description	Reset Value
EP0FIFO	0x90	R/W	Endpoint0 FIFO register	00h

This register is used to read the Endpoint0 FIFO. The Endpoint0 is bidirectional and can be accessed either by USB or the microcontroller. The default direction is from the GFI to the microcontroller. However, oncethe Endpoint0 receives a SETUP token, and it has decoded the direction of the DATA phase of the control transfer to be IN, the direction of the FIFO is changed.

ENDPOINTX FIFO Register

Register	Address	R/W	Description	Reset Value
EP1FIFO	0x91	R/W	Endpoint1 FIFO register	00h
EP2FIFO	0x92	R/W	Endpoint2 FIFO register	00h
EP3FIFO	0x93	R/W	Endpoint3 FIFO register	00h

This register is used to access ENDPOINTX whith the microcontroller.

USB ENABLE Register

Register	Address	R/W	Description	Reset value
USBENA	0x9E	R/W	USB Enable register	00h

If MCU is reseted by power on reset or external reset, you must manuplate this register to enable USB function.

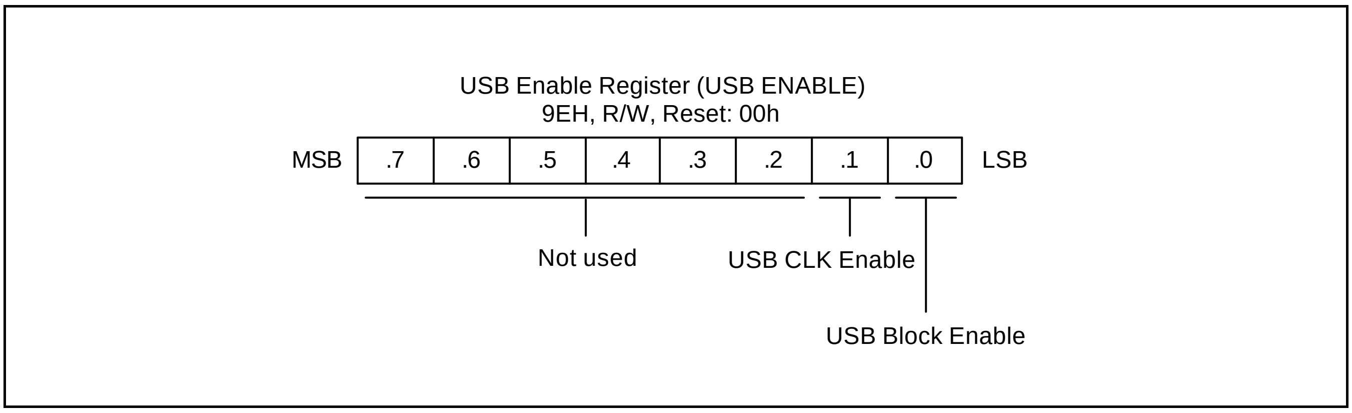


Figure 23-18 USB Enable Register

24 EMBEDDED FLASH MEMORY INTERFACE

OVERVIEW

The S3FB42F has an on-chip flash ROM instead of masked ROM. The flash ROM is accessed by serial data format and the type of a half flash. The S3FB42F's embedded 106.5 K-word (213 K-byte) memory has several operating features below:

The S3FB42F has 6 pins used to read/write the flash memory, $V_{DD}/V_{SS,}$ Reset, $V_{PP,}$ SDAT, SCLK.
The flash memory control block supports tool program mode :

TOOL PROGRAM MODE

The 6 pins are connected to a tool board and programmed by Serial OTP Tool(MDS). 12.5V is supplied into the V_{PP} pin. The other modules except flash ROM module are at a reset state.

This mode doesn't support sector erase but chip erase and two protection modes. (hard lock protection/ read protection)

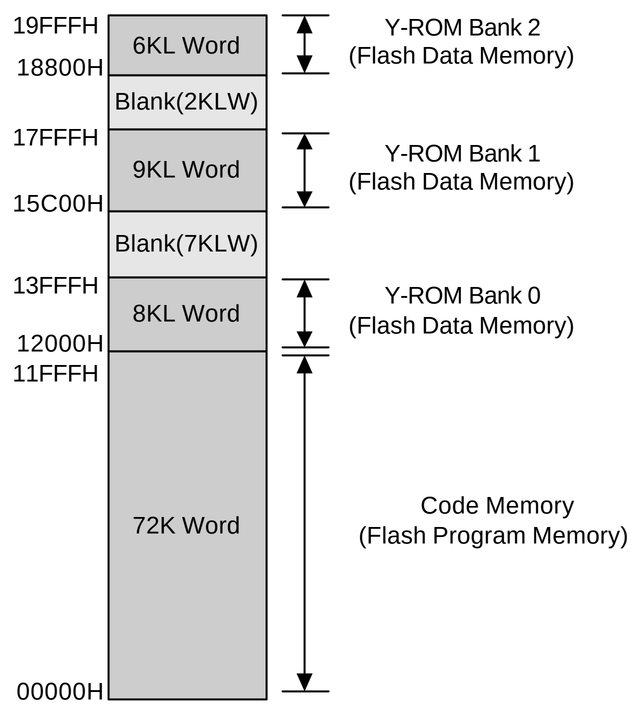


Figure 24-1. Flash memory structure

FLASH MEMORY CONTROL REGISTER

FMCON register controls what is concerned with an internal flash memory including data memory bank selection.

Register	Address	R/W	Description	Reset Value
FMCON	0078h	R/W	Flash memory control register	00h

[0]	Data memory bank selection	S3FB42F has two banks -bank0, bank1- as data memory in CalmRISC side. (See chapter2.Address Space for the banks) This bank selection bit should be controlled before accessing any address in data memory bank0 or bank1. 0: Bank0 1: Bank1
[2:1]	Y-data flash memory bank select	This bank selection bit should be controlled before accessing any address in Y-data flash memory. 00: No Select 01: Select Bank 0 10: Select Bank 1 11: Select Bank 2
[3]	Flash memory accessing speed selection	0: When fxx is under 4MHz 1: When fxx is more than 4MHz
[5:4]	I/O area accessing wait cycle selection	1 cycle or 2 cycles delay time can occurs by setting these bits when any address in I/O area is accessed. 00: Wait 0 cycle 01: Wait 1 cycle 1x: Wait 3 cycles
[6]	ROM accessing wait cycle enable	1 cycle or 2 cycles delay time can occur by setting this bit when ROM area is accessed. 0: Disable 1: Enable 1 cycle stretch
[6]	Flash Data Memory stretch enable	It is recommended to enable this bit when CALM CPU access to data flash memory. 0: Disable 1: Enable 1 cycle stretch (don't care when DSP instruction)

7	6	5	4	3	2	1	0
X	"0"	X	X	X	X	X	X

[0] Data memory bank selection bit
0 = Bank0
1 = Bank1

[2:1] Y-data flash memory selection bits
00 = No Select
01 = Selection Bank 0
10 = Selection Bank 1
11 = Selection Bank 2

[3] Flash memory speed selection bit
00 : When fxx is under 4 MHz
01 : When fxx is more than 4 MHz

[5:4] I/O area accessing wait cycle selection bit
00 = not-used wait cycle
01 = wait 1 cycle
1x = wait 3 cycles

[6] ROM accessing wait cycle enable
0 = disable
1 = enable 1 cycle stretch

[7] Y-data flash memory stretch control
0 = disable
1 = enable 1 cycle stretch (don't care when DSP instruction)

Figure 24-2. Flash Memory Control Register

25 MAC2424

INTRODUCTION

MAC2424 is a 24-bit high performance fixed-point DSP coprocessor for CalmRISC microcontroller. MAC2424 is designed for the mid to high-end audio applications which require low power consumption and portability. It mainly includes a 24-bit arithmetic unit (ARU), a barrel shifter & exponent unit(BEU), a 24-bit x 24-bit multiplier accumulation unit (MAU), and a RAM pointer unit (RPU) for data address generation. Main datapaths are constructed to 24-bit width for audio applications, but it can also perform 16-bit data processing efficiently in 16-bit operation mode.

MAC2424 is designed to be the DSP coprocessor for CalmRISC microcontroller. It receives 12-bit instruction code and command information from CalmRISC via special coprocessor interface and send internal status information to CalmRISC through external condition port.

ARCHITECTURE FEATURES

- 16-bit barrel shifter with support for multi-precision capability
- 24-bit exponent evaluation with support for multi-precision capability
- Four data address RAM pointers with post-modification & modulo capability
- Four index registers with two extended index registers : up to 8-bit index value
- Two direct address RAM pointers for short direct addressing
- Min/Max instruction with pointer latching and modification
- Division step in single cycle
- Conditional instruction execution capability

24-bit Mode Operation

- Signed fractional/integer 24 x 24-bit multiplication in single cycle
- 24 x 24-bit multiplication and 52-bit accumulation in a single cycle
- 24-bit arithmetic operation
- Two 48-bit multiplier accumulator with 4-bit guard
- 32K x 24-bit data memory spaces

16-bit Mode Operation

- Four-Quadrant fractional/integer 16 x 16-bit multiplication in single cycle
- 16 x 16-bit multiplication and 40-bit accumulation in a single cycle
- 16-bit arithmetic operation with 8-bit guard
- Two 32-bit multiplier accumulator with 8-bit guard
- 32K x 16-bit data memory spaces

BLOCK DIAGRAM

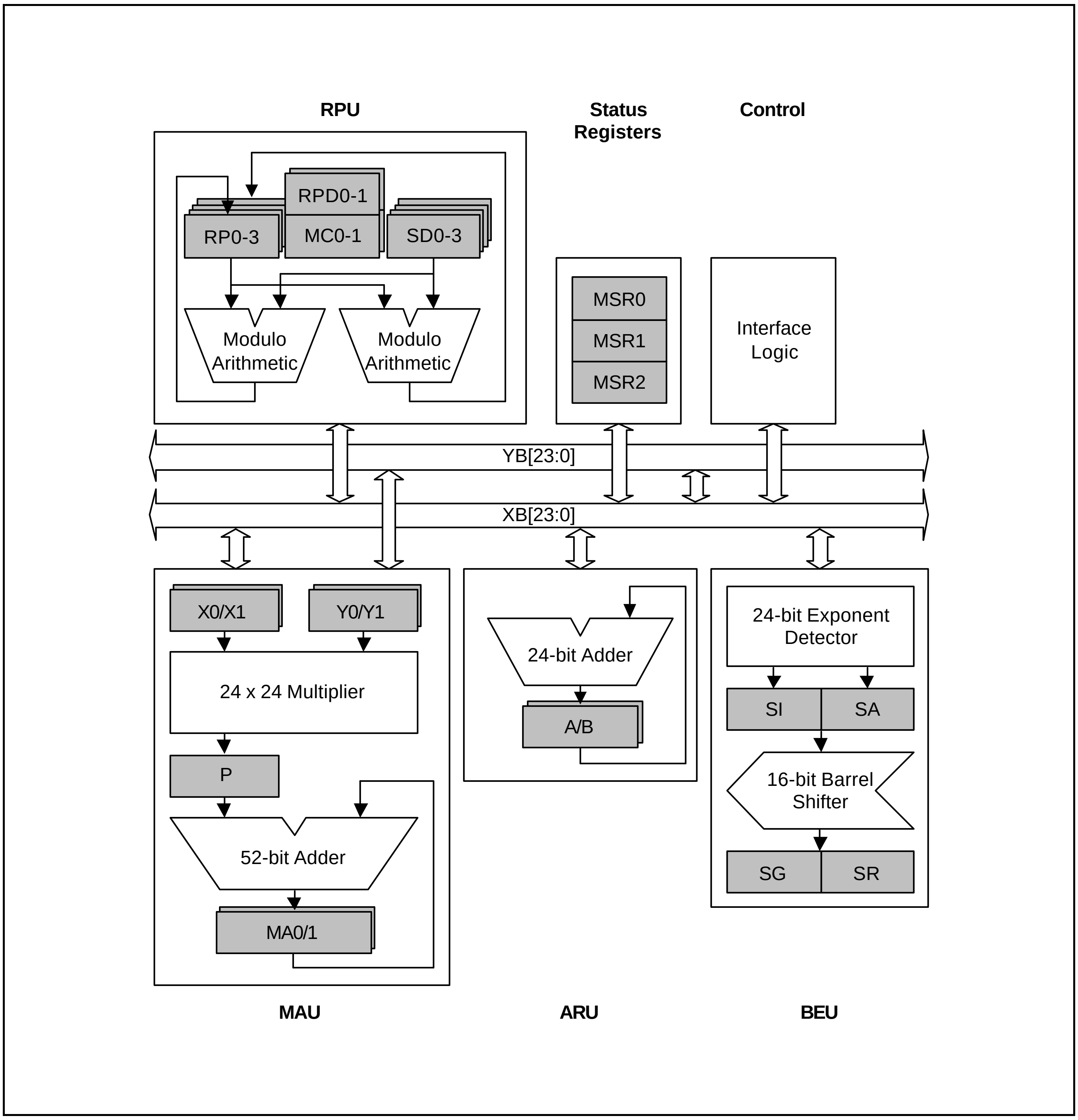


Figure 25-1. MAC2424 Block Diagram

The block diagram shows the main blocks that compose the MAC2424:

- Multiplier Accumulator Unit (MAU)
- Arithmetic Unit (ARU)
- Barrel shifter & Exponent detection Unit (BEU)
- RAM Pointer Unit (RPU)
- Status Registers
- Interface Unit

The MAC2424 DSP coprocessor is organized around two 24-bit data buses (XB, YB). Data movement between the units and memories occur over XD and YD data buses. Each of this data bus has its dedicated 14-bit address bus XA and YA respectively.

I/O DESCRIPTION

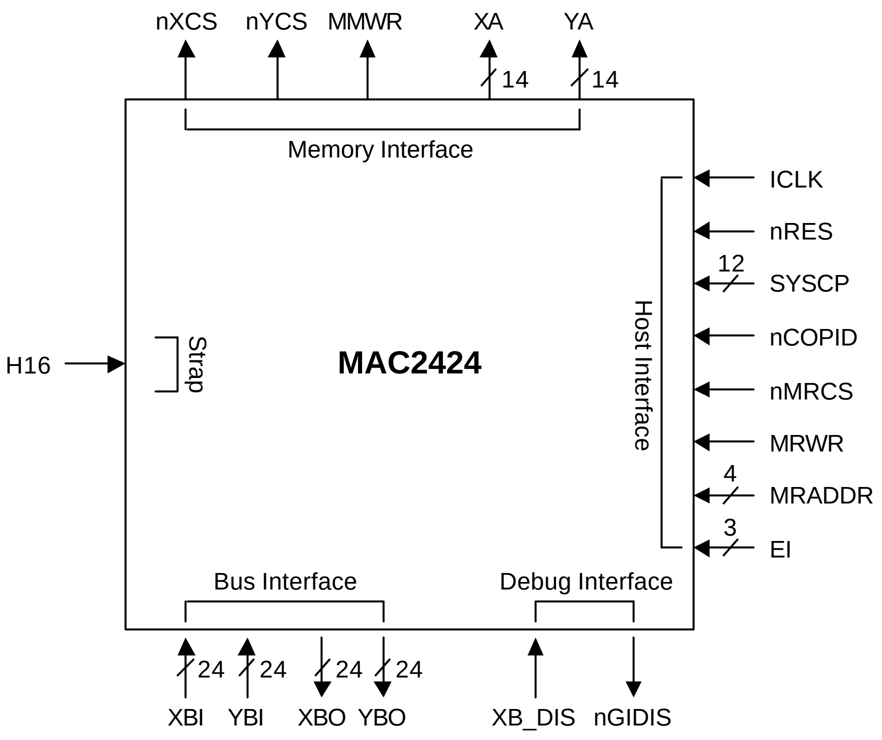


Figure 25-2. MAC2424 Pin Diagram

Two data and address buses are provided with control signals. The address XA and YA are 14-bit, and accesses data memories up to 16 Kbyte with 24-bit data width. The host interface signals receive the coprocessor interface signals from CalmRISC microcontroller, and send the status information through EI signals. For more information about coprocessor interface signals, please refer to CalmRISC Architecture manual.

Table 25-1. MAC2424 Pin Description

Signal Name	Width	Direction	Description
ICLK	1	I	Input Clock
nRES	1	I	Reset Bar
SYSCP	12	I	Instruction Bus
nCOPID	1	I	Instruction Bus Valid Indication Bar
MRADDR	4	I	Internal Register Selection Address
nMRCS	1	I	Internal Register Read/Write Enable Bar
MRWR	1	I	Internal Register Write Enable
EI	3	O	Internal Status Information
nXMCS	1	O	X Memory Chip Select Bar
nYMCS	1	O	Y Memory Chip Select Bar
MMWR	1	O	Memory Write Enable
XA	14	O	X Memory Address
YA	14	O	Y Memory Address
XBI	24	I	X Memory Data Input Bus
YBI	24	I	Y Memory Data Input Bus
XBO	24	O	X Memory Data Output Bus
YBO	24	O	Y Memory Data Output Bus
H16	1	I	16-bit Host Processor Indication
XB_DIS	1	I	X Memory Data Output Bus Disable
nGIDIS	1	O	Global Interrupt Disable for Long Word Instruction

PROGRAMMING MODEL

In this chapter, the important features of each unit in MAC2424 are discussed in details. How the data memories are organized is discussed and data memory addressing modes are explained.

The major components of the MAC2424 are :

- Multiplier Accumulator Unit (MAU)

Multiplier	
– Input Registers	X0, X1, Y0, Y1
– Output Register	P
Multiplier Accumulators	MA0, MA1
Saturation Logic	
Multiplier Accumulator Shifter	
52-bit Arithmetic Unit	
Status Register	MSR1

- Arithmetic Unit (ARU)

Accumulator	A, B
Saturation Logic	
Accumulator Shifter	
24-bit Arithmetic Unit	
Status Registers	MSR0, MSR2

- Barrel shifter & Exponent detection Unit (BEU)

24-bit Exponent Detector	
16-bit Barrel Shifter	
– Input Registers	SA, SI
– Output Registers	SG, SR

- RAM Pointer Unit (RPU)

Two Modulo Address Generators	
Bit-Reverse Generator	
Indirect Address Pointers	RP0, RP1, RP2, RP3
Index Registers	SD0, SD1, SD2, SD3
Extended Index Registers	SD0E, SD3E
Direct Pointers	RPD0, RPD1
Modulo Configuration Registrers	MC0, MC1

MULTIPLIER AND ACCUMULATOR UNIT

The Multiplier and Accumulator Unit contains two main units, the Multiplier Unit and the Accumulator Unit. The detailed block diagram of the Multiplier and Accumulator Unit is shown in Figure 25-3.

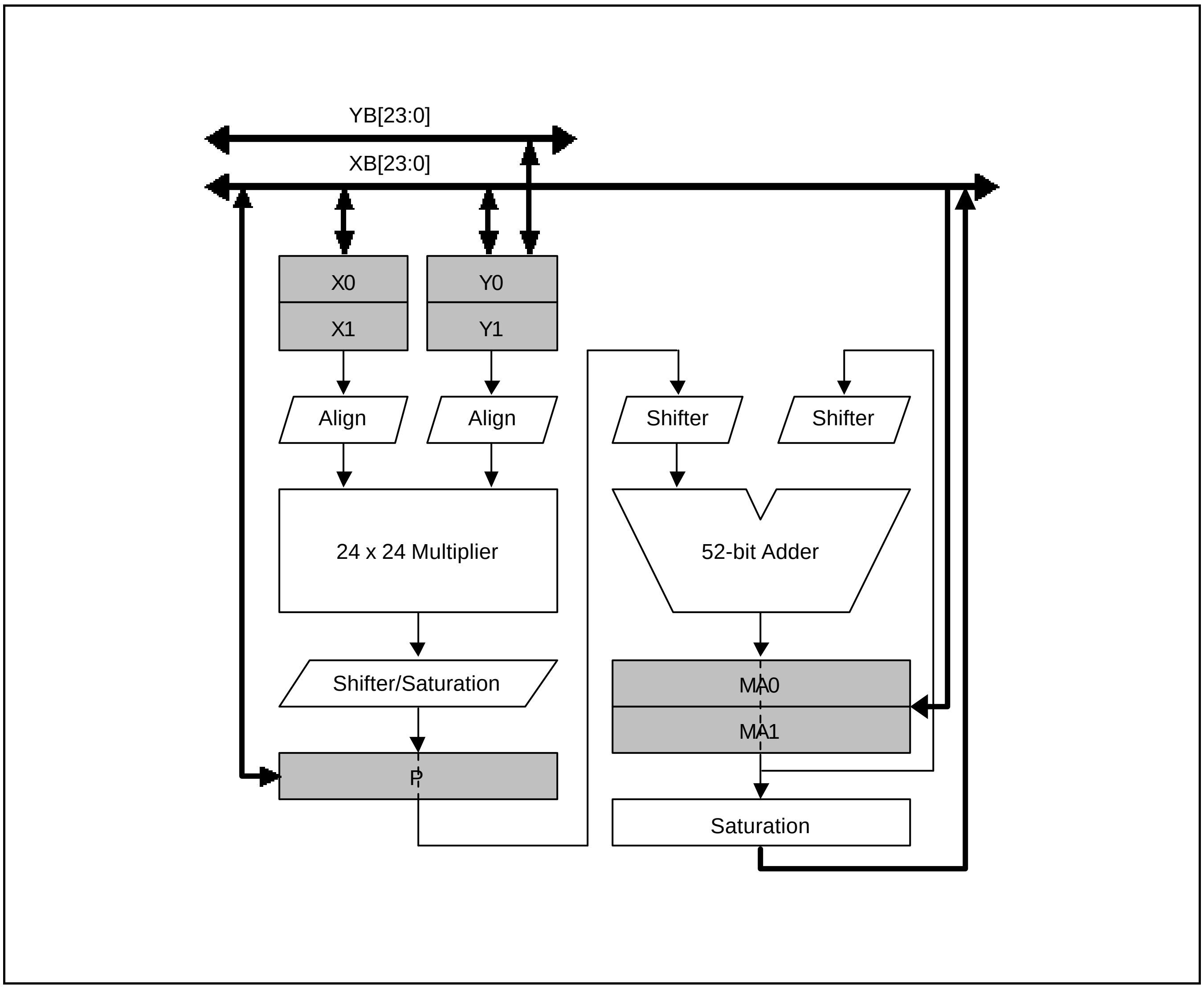


Figure 25-3. Multiplier and Accumulator Unit Block Diagram

Multiplier

The Multiplier unit consists of a 24 by 24 to 48 bit parallel 2's complement single-cycle, non-pipelined multiplier, 4 24-bit input registers (X0, X1, Y0, and Y1), a 48-bit output product register (P), and output shifter & saturation logic. The multiplier performs signed by signed multiplication in 24-bit mode, and 4 quadrant multiplication in 16-bit mode. Together with 52-bit adder in MAU, the MAC2424 can perform a single-cycle Multiply-Accumulate (MAC) operation. The multiplier only operates when multiply instruction is executed. The P register is not updated and the multiplier is not operated after a change in the input registers. This scheme reduces power consumption in multiplier.

In 16-bit operation mode, multiplier input registers, X and Y, are aligned (shifting 4 bits to the left) before multiplication, and 32-bit output result is written in bit 39 to 8 of P register. The bit 47 to 40 of P register is sign-extended, and lower 8-bit part are forced to 0.

PSH1 bit of MSR1 register indicates whether multiplier output is shifted 1 bit to the left or not. If PSH1 bit is set, multiplier output is shifted 1 bit to the left. This operation can be used in the signed fractional multiplication. USM bit of MSR1 register indicates whether multiplier input register is signed or unsigned in 16-bit operation mode. When USM bit is set in 16-bit mode, X1 and Y1 register is interpreted as an unsigned operand. For example, if X1 and Y0 register is selected as multiplier input register, unsigned by signed multiplication is performed. If X1 and Y1 register is selected, unsigned by unsigned multiplication is performed. Note that unsigned operation is only possible in the 16-bit mode.

The X or Y register is read or written via the XB bus, and Y register is written via YB when dual load instruction is executed. The 24-bit most significant portion (MSP) of the P register (PH) or the 24-bit least significant portion (LSP) of the P register (PL) can be written by the XB as an operand. When MSP of the P register is written, LSP of the P register is forced to zero. When LSP of the P register is written, MSP of the P register is not changed. In 16-bit operation mode, read or write operation on PL register is different from 24-bit operation mode. When PL write operation, the 16-bit most significant portion of PL register is written by the 16-bit least significant portion of XB bus, and 8-bit LSP of PL is forced to zero. On PL read operation, the 16-bit most significant portion of PL register is read to the 16-bit least significant portion of XB bus, and 8-bit MSP of XB is sign-extended. The other registers performs the same operation as 24-bit mode.

Overflow Protection in Multiplier

The only case the multiplier overflow occurs is when multiplying 800000h by 800000h in fractional 24-bit mode, and 8000h by 8000h in signed/signed fractional 16-bit mode. (These cases mean –1*-1) : the result should be normally 1, which overflows fractional format. Thus, in this particular case, a multiplier saturation block forces the multiplier result to 7FFFFFFFFFFFh (24-bit mode) or 007FFFFFFF00h (16-bit mode) after internal 1-bit shift to the left and write this value to the product register P.

– Saturation Condition at 24-bit mode: ~Prod[47] & Prod[46] & PSH1
– Saturation Condition at 16-bit mode: ~Prod[39] & Prod[38] & PSH1 & SX & SY

Multiplier Accumulators

Each MAi (i=0,1) is organized as two regular 24-bit registers (MA0H, MA0L, MA1H, MA1L) and two 4-bit extension nibble (MA0E, MA1E) in MSR1 register. The MAi accumulators can serve as the source operand, as well as the destination operand of MA relevant instructions. Only one MA accumulator can be used as an operand at a time according to the BKMA bit of the MSR1 register. If BKMA is set, MA1 register can be used, and if BKMA is reset, MA0 register can be used. Data transfer between two MA accumulators is possible through “ELD MA1, MA0” and “ELD MA0, MA1” instructions. These are the only cases when two MA accumulator is accessible independent on BKMA bit and a full 52-bit MA accumulator is loaded.

The 24-bit most significant portion (MSP) of the MA register (MAiH) or the 24-bit least significant portion (LSP) of the MA register (MAiL) can be written by the XB as an operand. When MAiH register is written, MAiL register is forced to zero and MAiE extension nibble is sign-extended. When MAiL register is written, MAiH and MAiE are not changed.

In 16-bit operation mode, read or write operation on MAiL register is different from 24-bit operation mode. The operation is same as PL register operation. When MAiL write operation, the 16-bit most significant portion of MAiL register is written by the 16-bit least significant portion of XB bus, and 8-bit LSP of MAiL is forced to zero. On MAiL read operation, the 16-bit most significant portion of MAiL register is read to the 16-bit least significant portion of XB bus, and 8-bit MSP of XB is sign-extended. In case of 16-bit mode MAiH write operation, MAiE extension nibble is not sign-extended.

Extension Nibbles

Extension nibbles MA0E and MA1E in MSR1 register offer protection against 48-bit overflows in 24-bit mode operation. When the result of a 52-bit adder output crosses bit 47, it sets VMi flag of MSR1 register (MA register Overflow flag). When the sign is lost beyond the MSB of the extension nibble, it sets MV flag of MSR1 (Memorized Overflow flag) and latches the value.

In 16-bit mode, these extension nibbles are not used at all and 8-bit most significant portion of the MAiH register is used as extension byte. If the result of a 52-bit adder output crosses bit 39, it sets VMi flag.

Overflow Protection in MA Registers

The multiplier accumulator saturation instruction (ESAT instruction) sets the destination MA register to the positive negative maximum value, if selected MA register overflows (VMi bit of MSR1 register is set). In case of 24-bit mode, saturation values are 7FFFFFFFFFFFh (positive overflow) or 800000000000h (negative overflow) for the MA register and extension nibble is sign-extended. In case of 16-bit mode, saturation value is different. When positive overflow occurs, the saturation value is 007FFFFFFF00h for MA register, and when negative overflow, the saturation value is FF8000000000h.

Another saturation condition is when moving from MAiH register through XB bus. This saturation mode is enabled when selected MA register overflows (VMi bit at MSR1 register is set), and overflow protection bit is enabled (OPM bit at MSR1 register is set). In this case the saturation logic will substitute a limited data value having maximum magnitude and the same sign as the source register. The MA register value itself is not changed at all. In case of 24-bit mode, saturation values are 7FFFFFh (positive overflow) or 800000h (negative overflow) and in 16-bit mode, saturation values are 007FFFh or FF8000h.

– Saturation by Instruction: "ESAT" Instruction & VMi
– Saturation by MA Read: Read MAiH & VMi & OPM

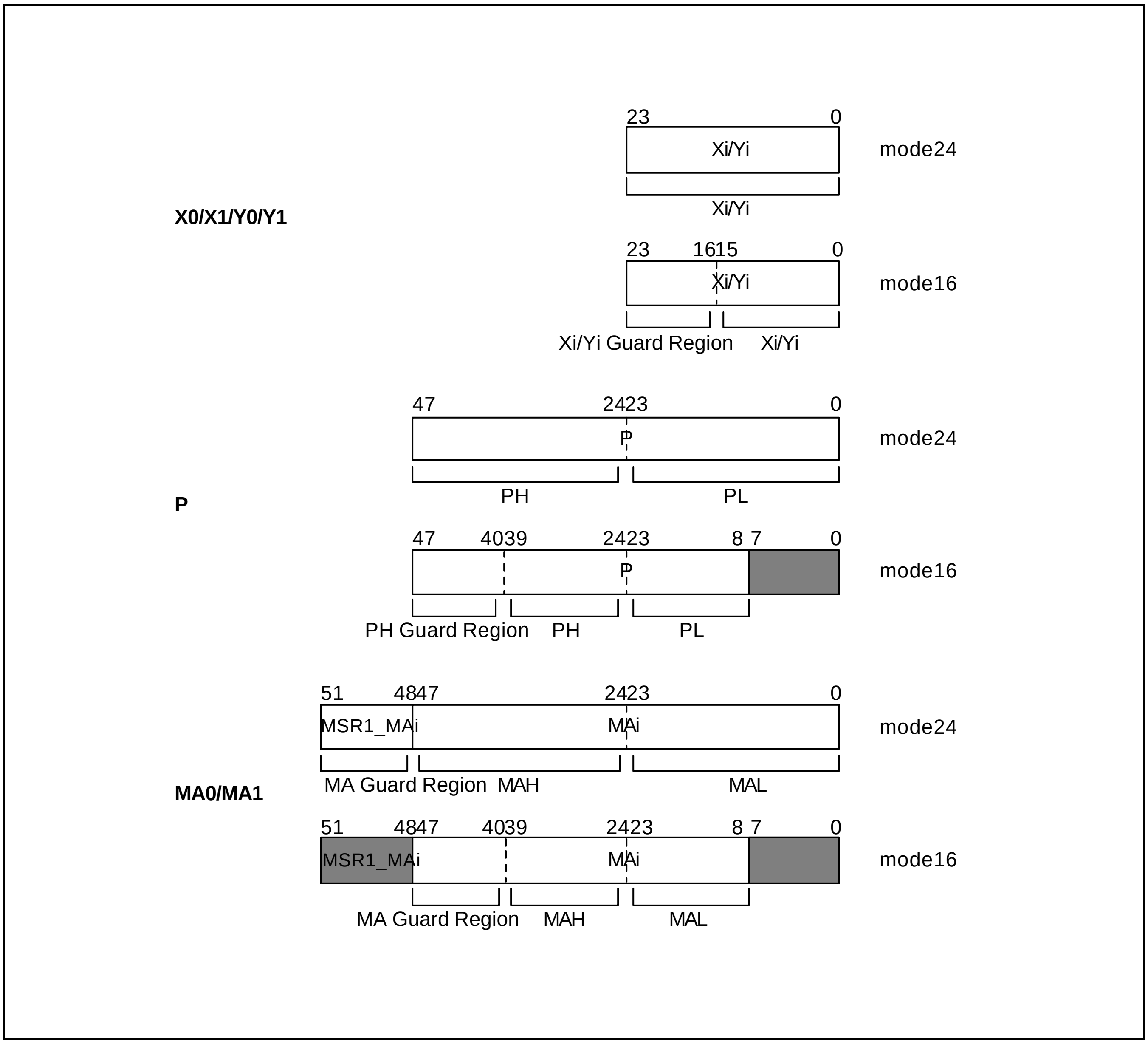


Figure 25-4. MAU Registers Configuration

ARITHMETIC UNIT

The arithmetic unit performs several arithmetic operations on data operands. It is a 52-bit, single-cycle, non-pipelined arithmetic unit. The arithmetic unit receives one operand from MAi, and another operand from P register. The source and destination MA accumulator of arithmetic instruction is always the same.

The arithmetic unit can perform positive or negative accumulate, add, subtract, shift, and several other operations, most of them in a single cycle. It uses two's complement arithmetics. Some flags (VMi, MV flag) are affected as a result of the arithmetic unit output value. The flags represent the MA register status.

Rounding Provision

Rounding (by adding 800000h to the LSP of the MA register) can be performed by special instruction ("ERND" instruction) in a single cycle: two's complement rounding. After rounding operation, the 24-bit least significant portion of MA register are cleared and the 24-bit most significant portion of MA register are filled with the rounded value.

MA Shifting Capabilities

52-bit MA register can be shifted by 1-bit left or right. All of this shift operation is arithmetic shift operation.

Double Precision Multiplication Support

The arithmetic unit support for double precision multiplication by add or subtract instruction with an alignment option of the P register. The P register can be aligned (shifting 24 bits to the right) before accumulating the partial multiplication result.

Division Possibilities

Two specific instructions ("EDIVQ" and "ERESR" instruction) are used to implement a non-restoring conditional add/subtract division algorithm. The division can be only signed and two operands (dividend and divisor) must be all positive number. The dividend must be a 48-bit operand, located in MA register. : 4-bit extension nibble contains the sign extension of the MA register in 24-bit operation mode. In 16-bit operation mode, the dividend must be a 32-bit operand and 8-bit extension nibble in the MA register must be sign-extended. The divisor must be a 24-bit operand(24-bit mode) or 16-bit operand with sign-extended to 24-bit, located in 24-bit most significant portion of the P register. The 24-bit least significant portion of the P register must be zero.

To obtain a valid result , the value of the dividend must be strictly smaller than the value of divisor (reading operand as fractional data). Else, the quotient could not be expressed in the correct format. (for example, quotient greater than 1 for fractional format). At the end of algorithm, the result is stored in the MA register. (the same which previously contained the dividend) : the quotient in the 24-bit LSP, the significant bit remainder stored in the 24 MSP of the MA register.

Typically 48/24 division can be executed with 24 elementary divide operations (32/24 division with 16 elementary divide operations), preceded by 1 initialization instructions (This instruction is required to perform initial subtraction operation.), and possibly followed by one restoring instruction which restores the true remainder (in case this last one is useful for the next calculations). Note that lower precision can also be obtained by decreasing the number of elementary division step applied.

The operation of elementary instructions for division is as follows.

"EDIVQ" :

This single cycle instruction is repeatedly executed to generate division quotient bits. It calculates one bit of the quotient at a time, computes the new partial remainder, sets VMi bit of the MSR1 register according to the new partial remainder sign. First, this instruction calculates the new partial remainder by adding or subtracting the divisor from the remainder, depending on current VMi bit value.

If current VMi = 0, new partial remainder = old partial remainder – divisor

If current VMi = 1, new partial remainder = old partial remainder + divisor

This add or subtract operation is performed between MA register and P register. Second, this instruction shifts the new partial remainder one bit to the left and moves one bit quotient into the rightmost bit. The one bit quotient bit is the inverted value of the new partial remainder sign-bit.

Quotient bit = ~(sign of new partial remainder)

Third, EDIVQ updates the MA register with shifted new partial remainder value, and updates the VMi bit of MSR1 register with sign value of the new partial remainder. This VMi update determines the operation of the next EDIVQ instruction.

"ERESR":

This single cycle instruction restores the true remainder value. In fact, due to the non-restoring nature of the division algorithm, the last remainder has to be restored or not by adding 2 times the divisor, depending on the VMi bit of MSR1 register previously computed.

If VMi = 0, No Operation is performed

If VMi = 1, Adds two times the divisor to the MA register.

(containing the last calculated remainder in the 24-bit most significant portion)

The new calculated remainder will have to be 24-bit right arithmetical shifted (16-bit right arithmetic shifted in 16-bit mode), in order to be represented in a usual fractional format.

```
          Dividend : 23 (0001 0111)                      Dividend : 17 (0001 0001)
          Divisor   : 6 (0110)                           Divisor   : 6 (0110)

                MA      0 0001 0111                            MA      0 0001 0001

                P         0110 0000                            P         0110 0000

        ESLA :  MA      0 0010 1110                    ESLA :  MA      0 0010 0010

        EDIVQ :    +    1 1010 0000                    EDIVQ :    +    1 1010 0000
                ___________________                            ___________________
                        1 1100 1110                                    1 1100 0010
                MA      1 1001 1100                            MA      1 1000 0100

        EDIVQ :    +    0 0110 0000                    EDIVQ :    +    0 0110 0000
                ___________________                            ___________________
                        1 1111 1100                                    1 1110 0100
                MA      1 1111 1000                            MA      1 1100 1000

        EDIVQ :    +    0 0110 0000                    EDIVQ :    +    0 0110 0000
                ___________________                            ___________________
                        0 0101 1000                                    0 0010 1000
                MA      0 1011 0001                            MA      0 0101 0001

        EDIVQ :    +    1 1010 0000                    EDIVQ :    +    1 1010 0000
                ___________________                            ___________________
                        0 0101 0001                                    1 1111 0001
                MA      0 1010 0011 — Quotient (3)             MA      1 1110 0010 — Quotient (2)
        ERESR :    +    0 0000 0000                    ERESR :    +    0 1100 0000
                ___________________                            ___________________
                MA      0 1010 0011                            MA      0 1010 0011

        ESRA :  MA      0 0101 0001                    ESRA :  MA      0 0101 0001
                          ‾‾‾‾ Remainder (5)                             ‾‾‾‾ Remainder (5)
```

Figure 25-5. Integer Division Example

```
Dividend : 23/128 (0001 0111)                 Dividend : 29/128 (0001 1101)
Divisor   : 6/8 (0110)                        Divisor   : 6/8 (0110)

          MA      0 0001 0111                           MA      0 0001 1101

          P         0110 0000                           P         0110 0000


          MA      0 0001 0111                           MA      0 0001 1101
EDIVQ :      +    1 1010 0000                 EDIVQ :      +    1 1010 0000
                  1 1011 0111                                   1 1011 1101
          MA      1 0110 1110                           MA      1 0111 1010

EDIVQ :      +    0 0110 0000                 EDIVQ :      +    0 0110 0000
                  1 1100 1110                                   1 1101 1010
          MA      1 1001 1100                           MA      1 1011 0100

EDIVQ :      +    0 0110 0000                 EDIVQ :      +    0 0110 0000
                  1 1111 1100                                   0 0001 0100
          MA      1 1111 1000                           MA      0 0010 1001

EDIVQ :      +    0 0110 0000                 EDIVQ :      +    1 1010 0000
                  0 0101 1000   Quotient                        1 1100 1001   Quotient
          MA      0 1011 0001   (1/8)                   MA      1 1001 0010   (2/8)
ERESR :      +    0 0000 0000                 ERESR :      +    0 1100 0000

          MA      0 1011 0001                           MA      0 0101 0010
                                Remainder                                     Remainder
                                (11/128)                                      (5/128)
```

Figure 25-6. Fractional Division Example

A 48/24 integer division example code is as follows

```
ER          VM                      // Initialize Division Step
ESLA        MA                      // Arithmetic Shift Left 1
EDIVQ       MA, P                   // Division Step
….
EDIVQ       MA, P                   // Division Step (24 times)
ERESR       MA, P                   // Remainder Restoring
ESRA        MA                      // Arithmetic Shift Right 1
```

A 48/24 fractional division example code is as follows.

```
ER          VM                      // Initialize Division Step
EDIVQ       MA, P                   // Division Step

….
EDIVQ       MA, P                   // Division Step (24 times)
ERESR       MA, P                   // Remainder Restoring
```

Note that the validity of the division operand must be checked before all of these code : i.e. the dividend is strictly smaller than the divisor. The following two figures show division with 9-bit dividend and 8-bit divisor. (Assume that the MA register and P register are 8-bit wide, and MA guard bit is 1-bit wide.)

STATUS REGISTER 1 (MSR1)

MSR1 register of three MAC2424 status registers (MSR0, MSR1, MSR2) is used to hold the flags, control bits, status bits for MAU. The contents of each field definitions are described as follows.

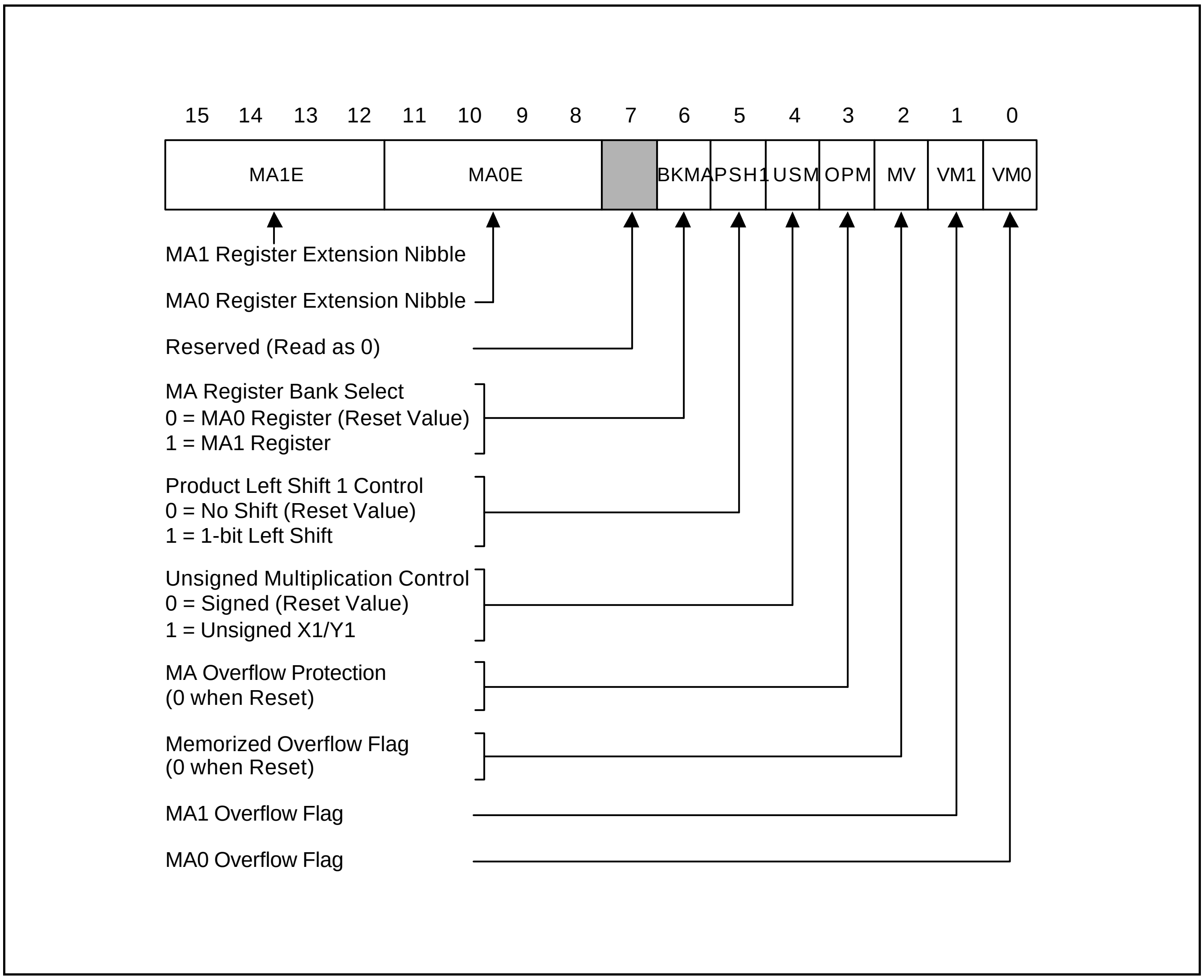


Figure 25-7. MSR1 Register Configuration

MA1E/MA0E – Bit 15–12/Bit 11–8

These four bit nibbles are used as guard bits for MA registers in 24-bit mode operation. These bits are updated when MA register write operation is occurred. In 16-bit mode operation these bits are not affected during MA write operation. These bits are also written during MSR1 register write operation.

BKMA – Bit 6

This bit defines current bank of MA register. Only one MA register of two MA registers is accessible at a time except "ELD MA1, MA0" or "ELD MA0, MA1" instruction. The BKMA bit is only affected when MSR1 register write operation or "ER/ES BKMA" instruction is used. When this bit is set, current bank of MA register is MA1 register, and when this bit is clear, current bank of MA register is MA0 register. The BKMA bit is cleared by a processor reset.

PSH1 – Bit 5

This bit defines multiplier output shift operation. When this bit is set, multiplier output result is 1-bit shifted left. This property can be used for fractional format operand multiplication. When this bit is clear, no shift is executed on the multiplier output. The PSH1 bit can be modified by writing to MSR1 register or "ER/ES PSH1" instruction. The PSH1 bit is cleared by a processor reset.

USM – Bit 4

The USM bit indicates that the X1 or Y1 register is signed or unsigned as a multiplicand. It is only used for product calculation in 16-bit mode operation. In 24-bit mode operation, this bit has no effect. When set, selected multiplicand is interpreted as a unsigned number if X1 or Y1 register is selected. The other registers (X0, Y0) are always signed number. The USM bit can be modified by writing to MSR1 register or "ER/ES USM" instruction. The USM bit is cleared by a processor reset.

OPM – Bit 3

The OPM bit indicates that saturation arithmetic is provided or not when moving from the higher portion of one of the MA registers through the XB bus. When the OPM bit is set(Overflow Protection is enabled), the saturation logic will substitute a limited data value having maximum magnitude and the same sign as the source MA register. If the OPM bit is clear, no saturation is performed. This bit has not effect on a "ESAT" instruction, which always saturates the MA register value. The OPM bit is modified by writing the MSR1 register or "ER/ES OPM" instruction. The OPM bit is cleared by a processor reset.

MV – Bit 2

The MV bit is a memorized 52-bit overflow (in 24-bit mode) or 48-bit overflow (in 16-bit mode). This bit indicates that the guard bits of MA register is overflowed during previous arithmetic operations. This bit is set when overflow on guard bits is occurred and is not cleared when this overflow is cleared. It is only cleared when "ER MV" instruction or MSR1 register write instruction is executed.

VM1/VM0 – Bit 1–0

These bits indicates arithmetic overflow on MA1 register and MA0 register respectively. One of these bits is set if an arithmetic overflow (48-bit overflow when 24-bit operation mode or 40-bit overflow when 16-bit operation) occurs after an arithmetic operation, and cleared otherwise. It represents that the result of an operation cannot be represented in 48 bits (in 24-bit mode) or 40 bits (in 16-bit mode). i.e. these bits are set when 5-bit value of MA[51:47] register is not all the same in 24-bit mode or 9-bit value of MA[47:39] register is not all the same in 16-bit mode. These bits are modified by writing the MSR1 register and one of these bits is written when "ER/ES VM" instruction or all arithmetic instruction according to the current bank of MA register (BKMA bit).

RAM POINTER UNIT

The RAM Pointer Unit (RPU) performs all address storage and effective address calculations necessary to address data operands in data memories. In addition, it supports latching of the modified register in maximum/minimum operations and bit reverse address generation. This unit operates in parallel with other resources to minimize address generation overhead. The RPU performs two types of arithmetics : linear or modulo. The RPU contains four 16-bit indirect address pointer registers (RP0 ~ RP3, also referred to RPi) for indirect addressing, two 16-bit direct address pointer registers (RPD0 ~ RPD1, also referred to RPDi) for short direct form addressing, four 16-bit indirect index registers (SD0 ~ SD3, also referred to SDi) and its extensions (SD0E and SD3E), and two 16-bit modulo configuration registers (MC0 and MC1, also referred to MCi) for modulo control. The MC0 register has effect on RP0 and RP1 pointer register, and the MC1 register has effect on RP2 and RP3 register.

All indirect pointer registers (RPi) and direct pointer registers (RPDi) can be used for both XA and YA for instructions which use only one address register. In this case the X memory and Y memory can be viewed as a single continuous data memory space. the bit 13 to bit 0 of RPi register and RPDi register defines address for X or Y memory, and the bit 14 determines whether the address is for X memory or Y memory. The bit 15 of RPi indicates whether the selected pointer is updated with modulo arithmetic. The RPU can access two data operand simultaneously over XA and YA buses. In dual access case, RP0 is automatically selected as a X memory pointer and RP3 is selected as a Y memory pointer regardless of bit 14 of RP0 and RP3.

All registers in the RPU may be read or written to by the XB as 16-bit data. The detailed block diagram of the RAM Pointer Unit is shown in Figure 25-8.

ADDRESS MODIFICATION

The RPU can generate up to two 14-bit addresses every instruction cycle which can be post-modified by two modifiers: linear and modulo modifier. The address modifiers allow the creation of data structures in the data memory for circular buffers, delay lines, FIFOs, etc. Address modification is performed using 15-bit two's complement linear arithmetics.

Linear (Step) Modifier

During one instruction cycle, one or two of the pointer register, RPi, can be post incremented/decremented by a 2's complement 4-bit step (from –8 to +7). If XSD bit of MSR0 register is set, these 4-bit step is extended to 8-bit (from –128 to +127) by concatenating index register with extended index register (SD0E, SD3E) when selected pointer is RP0 or RP3. The selection of linear modifier type (one out of four) is included in the relevant instructions. The four step values are stores in each index register SDi. If the instruction requires a data memory read operation, S0 (bit 3 to bit 0) or S1 (bit 7 to bit 4) field of SDi register is selected as a index value. If the instruction requires a data memory write operation, D0 (bit 11 to bit 8) or D1(bit 15 to bit 12) field of SDi register is selected as an index value.

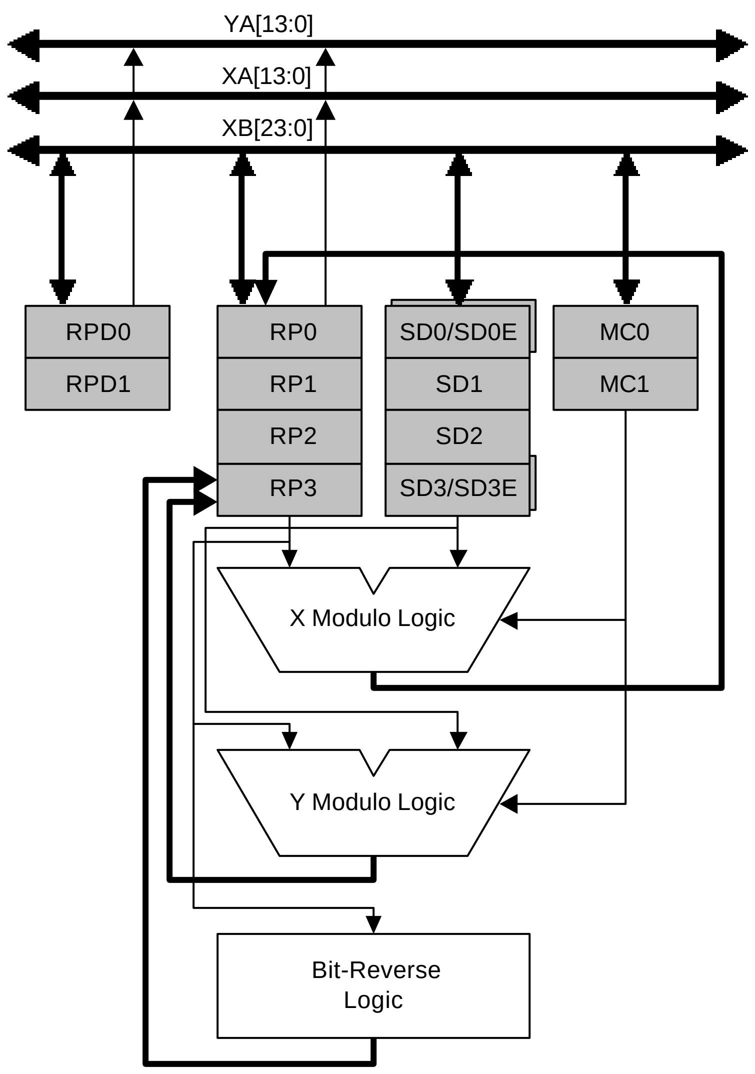


Figure 25-8. RAM Pointer Unit Block Diagram

Modulo Modifier

The two modulo arithmetic units (X, Y Modulo Logic) can update one or two address registers within one instruction cycle. They are capable of performing modulo calculations of up to 2^{10} (=1024). Each register can be set independently to be affected or unaffected by the modulo calculation using the ME bits in the each pointer register. Modulo setting values are stored in 13 least significant bits of modulo configuration registers MC0 and MC1 respectively. The bits 12 to bit 10 of MC0 and MC1 register determines maximum modulo size from 8 to 1024 and the bits 9 to bit 0 of modulo control register defines upper boundary of modulo calculation in the current modulo size. The lower boundary of modulo calculation is automatically defined by modulo size itself. (Refer to figure 25-10)

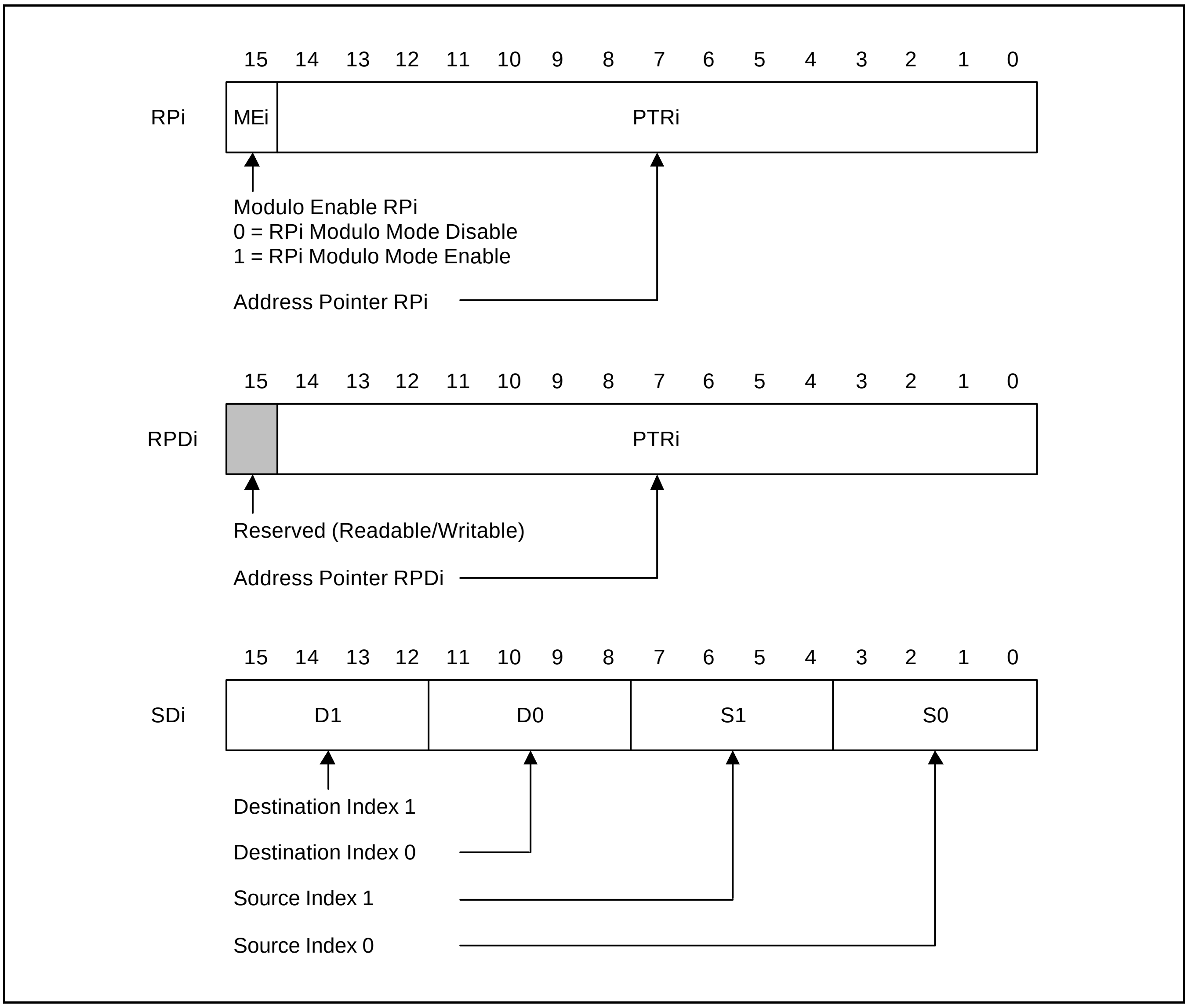


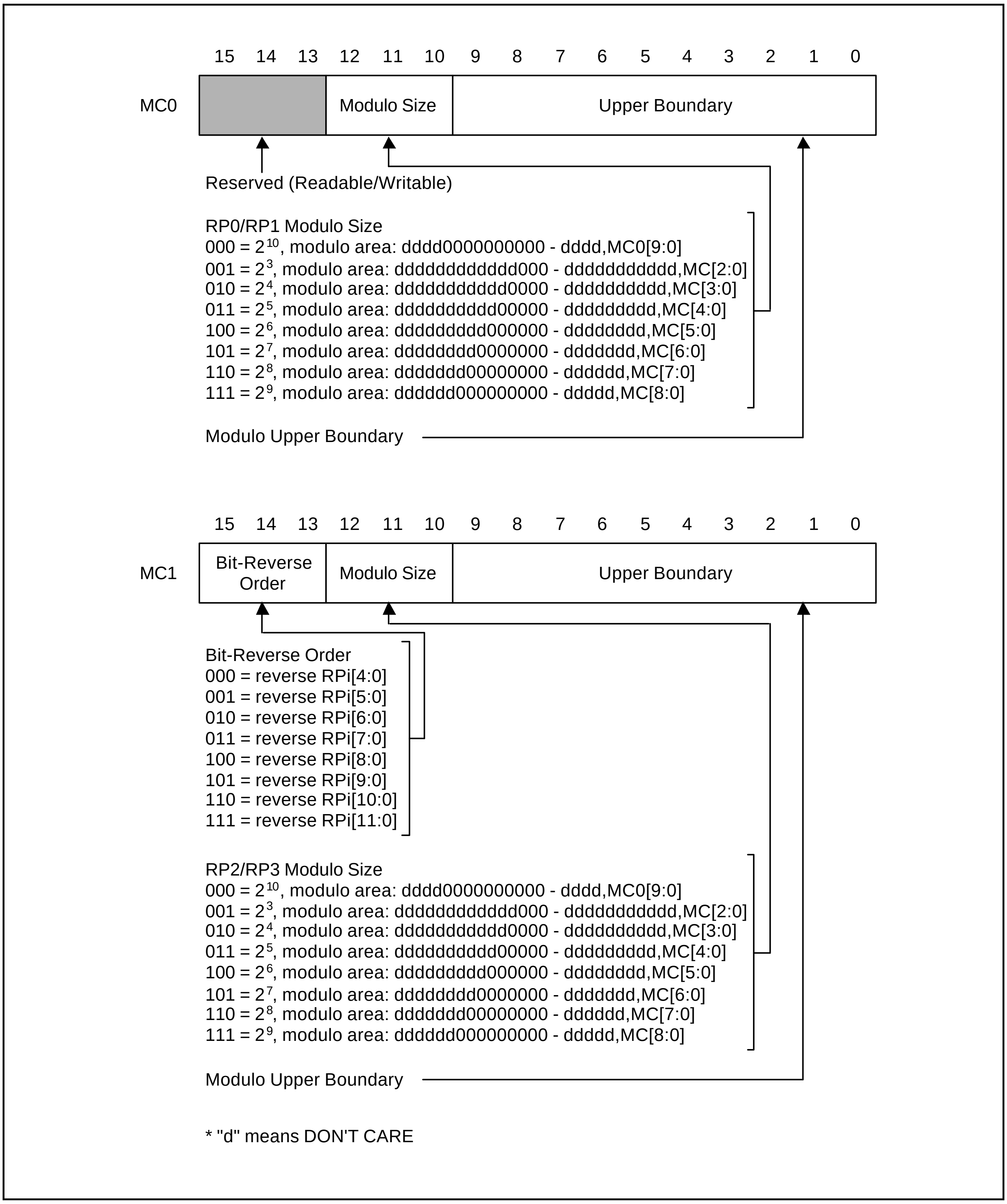
Figure 25-9. Pointer Register and Index Register Configuration

For proper modulo calculation, the following constraints must be satisfied. (M = modulo size, S = step size)

1. Only the p LSBs of RPi can be modified during modulo operation, where p is the minimal integer that satisfies $2^p \geq M$. RPi should be initiated with a number whose p LSBs are less than M.
2. $M \geq S$

The modulo modifier operation, which is a post-modification of the RPi register, is defined as follows

```
if ((RPi == Upper Boundary in k LSBs) and (q > 0)) then
        RPi k LSB ← 0
else if ((RPi == Lower Boundary in k LSBs) and (q < 0)) then
        RPi k LSB ← Upper Boundary in k LSBs
else
        RPi k LSB ← RPi + q (k LSBs)
where k is defined by MCi[12:10]
```



RP0/RP1 Modulo Size
000 = 2^{10}, modulo area: dddd0000000000 - dddd,MC0[9:0]
001 = 2^{3}, modulo area: ddddddddddddd000 - ddddddddddd,MC[2:0]
010 = 2^{4}, modulo area: ddddddddddd0000 - dddddddddd,MC[3:0]
011 = 2^{5}, modulo area: dddddddddd00000 - ddddddddd,MC[4:0]
100 = 2^{6}, modulo area: ddddddddd000000 - dddddddd,MC[5:0]
101 = 2^{7}, modulo area: dddddddd0000000 - ddddddd,MC[6:0]
110 = 2^{8}, modulo area: ddddddd00000000 - dddddd,MC[7:0]
111 = 2^{9}, modulo area: dddddd000000000 - ddddd,MC[8:0]

Modulo Upper Boundary

Bit-Reverse Order
000 = reverse RPi[4:0]
001 = reverse RPi[5:0]
010 = reverse RPi[6:0]
011 = reverse RPi[7:0]
100 = reverse RPi[8:0]
101 = reverse RPi[9:0]
110 = reverse RPi[10:0]
111 = reverse RPi[11:0]

RP2/RP3 Modulo Size
000 = 2^{10}, modulo area: dddd0000000000 - dddd,MC0[9:0]
001 = 2^{3}, modulo area: ddddddddddddd000 - ddddddddddd,MC[2:0]
010 = 2^{4}, modulo area: ddddddddddd0000 - dddddddddd,MC[3:0]
011 = 2^{5}, modulo area: dddddddddd00000 - ddddddddd,MC[4:0]
100 = 2^{6}, modulo area: ddddddddd000000 - dddddddd,MC[5:0]
101 = 2^{7}, modulo area: dddddddd0000000 - ddddddd,MC[6:0]
110 = 2^{8}, modulo area: ddddddd00000000 - dddddd,MC[7:0]
111 = 2^{9}, modulo area: dddddd000000000 - ddddd,MC[8:0]

Modulo Upper Boundary

* "d" means DON'T CARE

Figure 25-10. Modulo Control Register Configuration

The modulo calculation examples are as follows.

1. Full Modulo with Step = 1 (selected by instruction and index register value)
   MC0 = 000_001_0000000111 (Upper Boundary = 7, Lower Boundary = 0, Modulo Size = 8)
   RPi = 0010h
   0010h → 0011h → 0012h → 0013h → 0014h → 0015h → 0016h → 0017h → 0010h → 0011h
2. Full Modulo with Step = 3 (selected by instruction and index register value)
   MC0 = 000_001_0000000111 (Upper Boundary = 7, Lower Boundary = 0, Modulo Size = 8)
   RPi = 0320h
   0320h → 0323h → 0326h → 0321h → 0324h → 0327h → 0322h → 0325h → 0320h → 0323h
3. Part Modulo with Step = -2 (selected by instruction and index register value)
   MC0 = 000_001_0000000101(Upper Boundary = 5, Lower Boundary = 0, Modulo Size = 8)
   RPi = 2014h
   2014h → 2012h → 2010h → 2014h → 2102h

The total number of circular buffer (modulo addressing active area) is defined by 32K/Modulo size. i.e. if current modulo size is 32, the total number of circular buffer is 1024.

Bit Reverse Capabilities

The bit-reverse addressing is useful for radix-2 FFT(Fast Fourier Transform) calculations. The MAC2424 DSP coprocessor does not support the bit-reverse addressing itself. But it supports the bit field reverse capabilities in the form of instruction. The "ERPR" instruction selects a source address pointer RPi and performs bit reverse operation according to the bit field specified in bit 15 to bit 13 of MC1 register. The result bit pattern is written to the RP3 register pointer field. (bit 14 to bit 0) In this way, RP3 has a bit-reversed address value of source pointer value. Note that the data buffer size is always a power of 2 up to 2^{12}.

Index Extension

When an instruction with indirect addressing is executed, the current value of selected address pointer register RPi provides address on XA and YA buses. Meanwhile, the current address is incremented by the value contained into the selected index value contained into the selected bit field of selected index register, and stored back into RPi at the end of instruction execution.

The 4-bit index values can be considered as a signed number, so the maximum increment value is 7(0111b) and the maximum decrement value is –8(1000b). If the 4-bit index value is insufficient for use, the index values can be extended to 8-bit values when RP0 or RP3 register is selected as an address pointer register. In this case, all index values are extended to 8-bit by concatenating with SD0E or SD3E register. The bit field of SD0E and SD3E is the same as other index register SDi. The index extension registers are enabled when the XSD bit of MSR0 register is set. Otherwise, those are disabled. If the extension index registers are enable, index values for indirect addressing becomes to 8-bit during addressing with RP0 and RP3 pointer register, and current index register becomes the extended index register instead of the regular index register: i.e. When a index register is read or written by a load instruction, SD0E register or SD3E register is selected as a source operand or a destination operand, instead of SD0 or SD3 register. For each of SD0/SD0E or SD3/SD3E, only one register is accessible at a time.

DATA MEMORY SPACES AND ORGANIZATION

The MAC2424 DSP coprocessor has only data memory spaces. The program memory can only be accessed by CalmRISC, host processor. The data memory space is shared with host processor. The CalmRISC has 16-bit data memory address, so it can access up to 64 Kbyte data memory space.

The MAC2424 access data memory with 24-bit width or 16-bit width. It can access upto 32 Kword (word = 2-byte or 3-byte). The data space is divided into a lower 16 Kword X data space and a higher 16 Kword Y data space. When two data memory access are needed in an instruction, one is accessed in X data space, and the other is accessed in Y memory space. When one data memory access is needed, the access is occurred in X or Y data memory space according to the address.

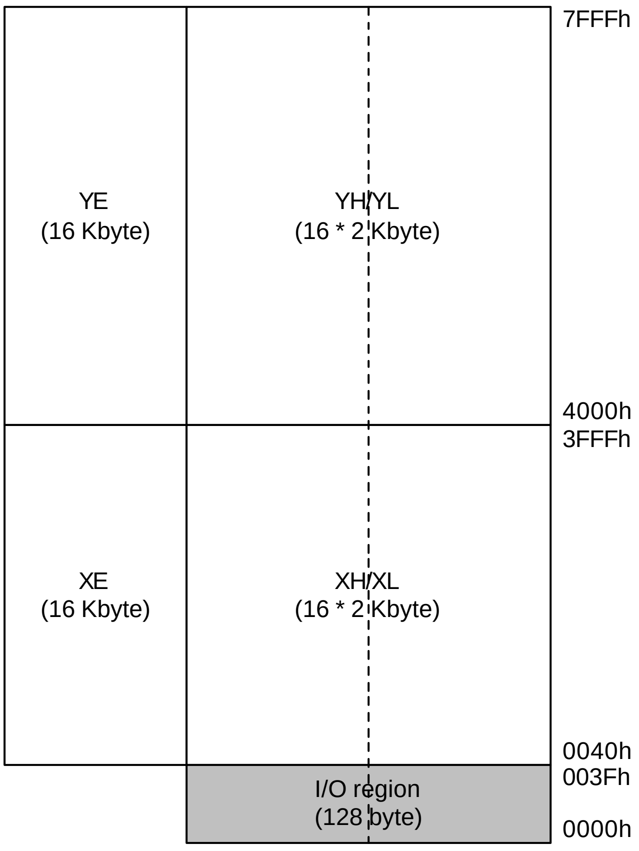


Figure 25-11. Data Memory Space Map

Each space is divided into 3 16 Kbyte XE/XH/XL or YE/YH/YL region when 24-bit data is needed, or 2 16 Kbyte XH/XL or YH/YL region when 16-bit data is needed, respectively. Each space can contain RAM or ROM, and can be off-chip or on-chip. In the X data space, the lower 128 byte locations are reserved for memory-mapped I/O. The MAC2424 coprocessor can not access the I/O region. only host processor can access. The configuration of this region depends on the specific chip configuration.

When 24-bit width data memory is used, the total memory space becomes to 96 Kbyte (16 Kbyte * 6). Because CalmRISC can only access 64 Kbyte memory space, the extended memory regions (XE and YE) are shadowed in the high address memory region (XH and YH). So, CalmRISC can access XH/XL pair or XE/XL pair in a time. The selection of shadowed region can be accomplished with "SYS #imm" instruction in CalmRISC. (Refer to each evaluation chip specification)

ARITHMETIC UNIT

The Arithmetic Unit (ARU) performs all arithmetic operations on data operands. It is a 24-bit, single cycle, non-pipelined arithmetic unit. The MAC2424 is a coprocessor of CalmRISC microcontroller. So, all the logical operation and other bit manipulation operations can be performed in CalmRISC. Thus, the MAC2424 has not logical units and bit manipulation units at all.

The ARU receives one operand from Ai(A or B) register, and another operand from either the MSB part of MA register, the XB bus, or from Ai. Operations between the two Ai register are possible. The source and destination Ai register of an ARU instruction is always the same. The XB bus input is used for transferring one of the MAC2424 register content, an immediate operand, or the content of a data memory location, addressed in direct addressing mode or in indirect addressing mode as a source operand. The flags in the MSR0 register are affected as a result of the ARU output. In most of the instructions where the ARU result is transferred to one of Ai registers, the flags represent the Ai register status. The detailed block diagram of the Arithmetic Unit is shown in Figure 25-12.

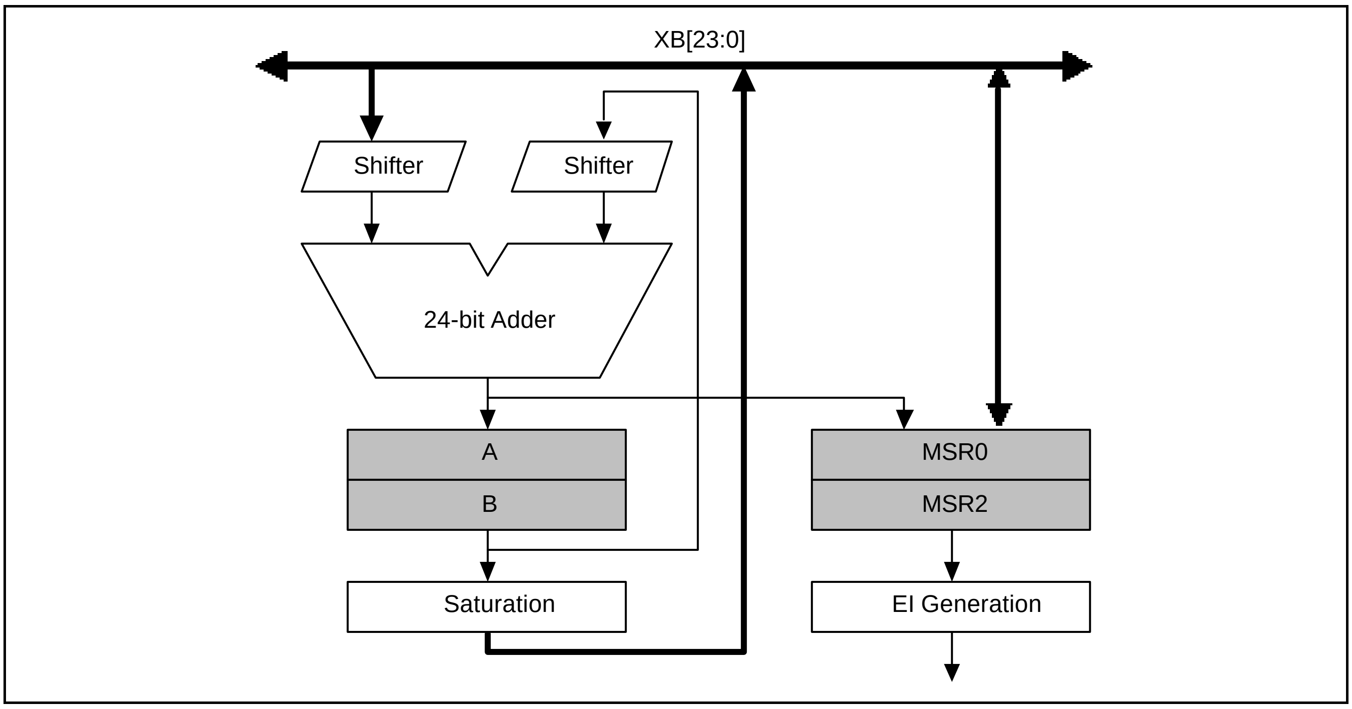


Figure 25-12. Arithmetic Unit Block Diagram

The ARU can perform add, subtract, compare, several other arithmetic operations (such as increment, decrement, negate, and absolute), and some arithmetic shift operations. It uses two's complement arithmetic.

A, B ACCUMULATORS

Each Ai (A or B) register is organized as a regular 24-bit register. The Ai accumulators can serve as the source operand, as well as the destination operand of the ARU instructions. The Ai registers can be read or written though the XB bus. In the 16-bit mode operation, Ai register is organized as a regular 16-bit register (bit 15 to bit 0) and 8-bit extension guard bits. (bit 23 to bit 16) When the result of a 24-bit adder output crosses bit 15, it sets Vi(VA or VB) bit of MSR0 register (A/B register Overflow flag). The extension guard bits offer protection against 16-bit overflows up to 255 overflows or underflows. If the sign is lost beyond the MSB of the extension guard bits, the result is lost and the value can not be recovered. There is no overflow indication at 24-bit boundary in 16-bit operation mode. In 24-bit operation mode, when the result of a 24-bit adder output crosses bit 23, it sets Vi.

OVERFLOW PROTECTION IN A/B ACCUMULATORS

The Ai accumulator saturation is performed differently according to the current operation mode. In 24-bit operation mode, the selected accumulator value is saturated during arithmetic operation which causes overflow, if overflow protection bit (OPA or OPB bit in MSR0 register) is enabled. The limited values are 7FFFFFh (positive overflow), or 800000h (negative overflow). During accumulator register read through XB bus, the saturation is not occurred. Contrary, in 16-bit operation mode, saturation is not occurred during arithmetic operation. The saturation is only occurred during accumulator register read through XB bus, if overflow protection is enabled and overflow occurred (OPA/OPB bit of MSR0 register is set, and VA/VB bit of MSR0 register is set). The saturated values are 007FFFh (positive overflow) or FF8000h (negative overflow).

– Saturation Condition at 24-bit mode : Arithmetic instruction & 24-bit Overflow & OPA/OPB
– Saturation Condition at 16-bit mode : Read A/B & VA/VB & OPA/OPB

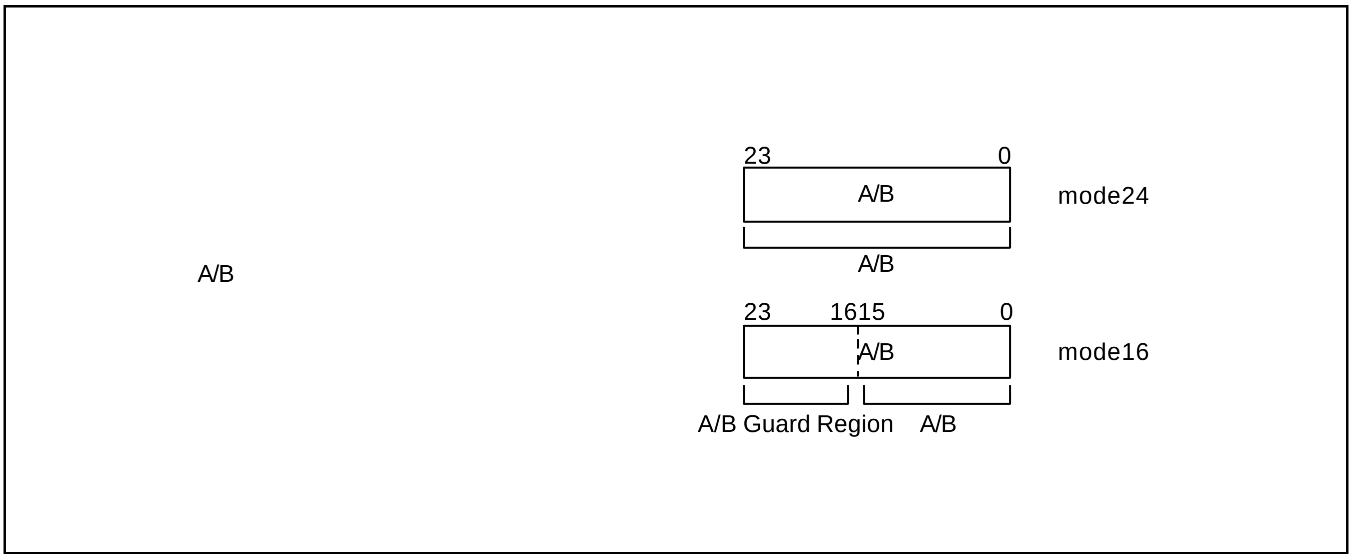


Figure 25-13. Ai Accumulator Register Configuration

ARITHMETIC UNIT

Maximum-Minimum Possibilities

Two Cycle maximum/minimum operations are available with pointer latching and modification. One of the Ai accumulator register holds the maximum value in a "EMAX" instruction, or the minimum value in a "EMIN" instruction. In the first cycle, the one accumulator register is compared with the operand by "ECP" instruction, and this instruction updates N flag value. In the second cycle, this value is copied to the above defined accumulator register. The address pointer register which generates address (except RP3) can be post-modified according to the specified mode in the instruction. When the new maximum or minimum number is found, the previous pointer value is latched into the LSB 15-bit field of RP3 pointer register. For more details, refer to "EMAX" and "EMIN" instructions on the instruction set.

The examples which searches block elements are as follows

```
Loop_Start:
            ECP  A, @RP0+S0                       // Compare Two Values (S0 must be 0)
            EMAX(EMIN) A, @RP0+S1                 // Conditional Load (S1 must be search index)
            JP Loop_Start
```

Conditional Instruction Execution

Some instructions can be performed according to the T flag value of MSR0 register. These instructions may operate when the T flag is set, and do nothing if the T flag is cleared. The instructions which have suffix "T" are this type of instructions. ("emod1" type instruction. The conditional instruction execution capabilities can reduce the use of branch instructions which require several cycles.

Shifting Operations

A few options of shifting are available in the ARU and all of them are performed in a single cycle. All shift operations performed in the ARU are arithmetic shift operations : i.e. right shift filling the MSBs with sign values and left shift filling with LSBs with zeros. The source and destination operands are one of 24-bit Ai accumulator registers. The shift instructions performed in the ARU are all conditional instructions. The shift amount is limited to 1 and 8, right or left respectively. The shift with carry is also supported.

Multi-Precision Support

Various instructions which help multi-precision arithmetic operation, are provided in the MAC2424. The instructions with suffix "C" indicates that the operation is performed on source operand and current carry flag value. By using this instructions, double precision or more precision arithmetics can be accomplished. The following shows one example of multi-precision arithmetic.

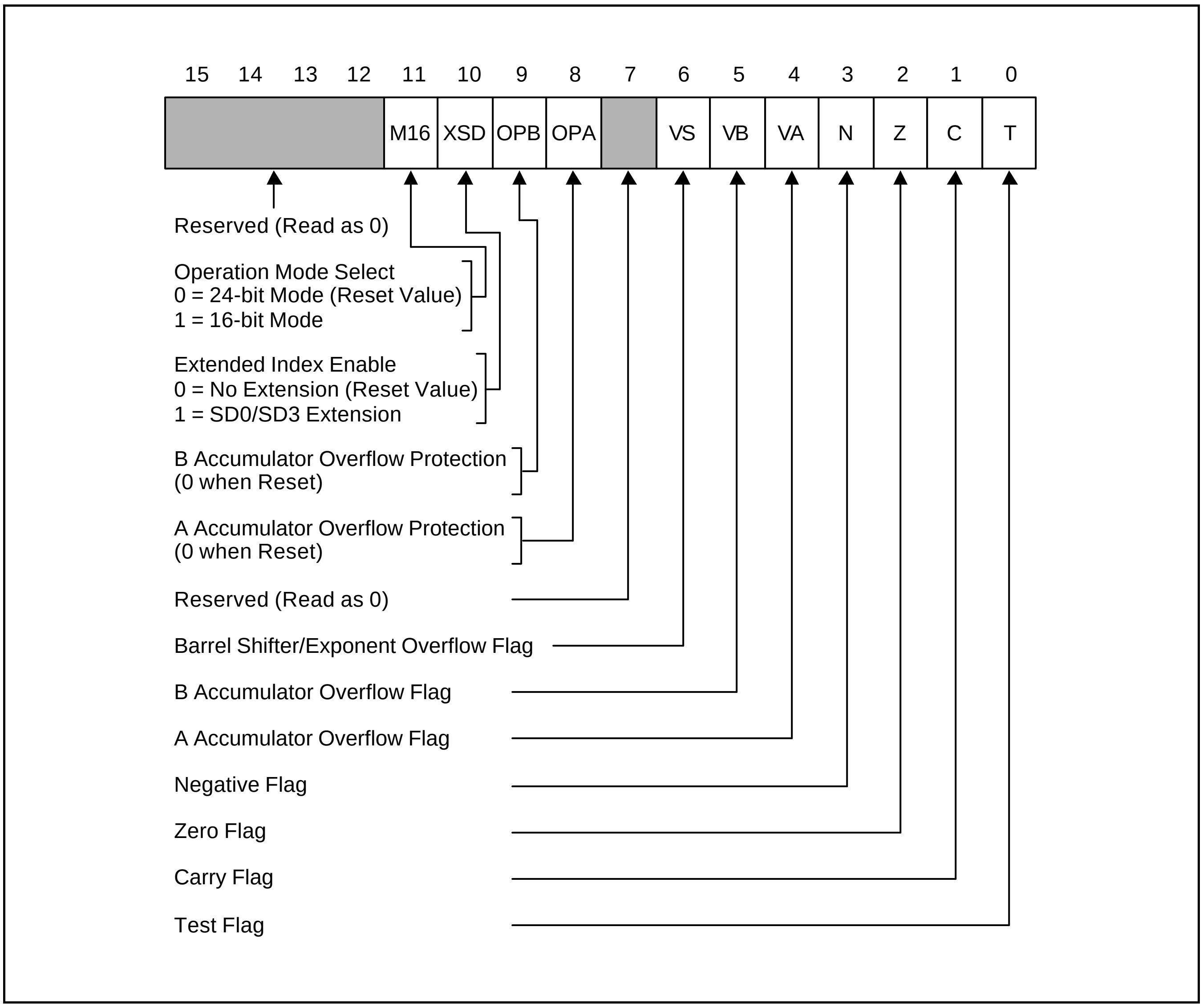
```
// 3-cycle Double Precision Addition (A:B + 2 memory operand)
EADD B, @RP0+S0                        // Lower Part Addition
EINCC A                                // Carry Propagation
EADD A, @RP0+S0                        // Higher Part Addition
```

EXTERNAL CONDITION GENERATION UNIT

The MAC2424 can generates and send the status information or control information after instruction execution to the host processor CalmRISC through EI[2:0] pin (Refer to Pin Diagram). The CalmRISC can change the program sequence according to this information by use of a conditional branch instruction that uses EI pin values as a branch condition. The EI generation block in the ARU selects one of status register value or combination of status register values according to the SECi (I=0,1,2) field in the MSR2 register. (Refer to MSR2 register configuration)

STATUS REGISTER 0 (MSR0)

MSR0 register of three MAC2424 status registers (MSR0, MSR1, MSR2) is used to hold the flags, control bits, status bits for the ARU and BEU(Barrel Shifter and Exponent Unit). The contents of each field definitions are described as follows.



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
M16 XSD OPB OPA VS VB VA N Z C T
Reserved (Read as 0)
Operation Mode Select
0 = 24-bit Mode (Reset Value)
1 = 16-bit Mode
Extended Index Enable
0 = No Extension (Reset Value)
1 = SD0/SD3 Extension
B Accumulator Overflow Protection
(0 when Reset)
A Accumulator Overflow Protection
(0 when Reset)
Reserved (Read as 0)
Barrel Shifter/Exponent Overflow Flag
B Accumulator Overflow Flag
A Accumulator Overflow Flag
Negative Flag
Zero Flag
Carry Flag
Test Flag

Figure 25-14. MSR0 Register Configuration

M16 – Bit 11

This bit defines current operation mode of the MAC2424 DSP coprocessor. If this bit is set, it indicates the current operation mode is 16-bit mode, and data registers and flags are configured to 16-bit mode. If this bit is clear (reset state), the MAC2424 operates on normal 24-bit mode. The M16bit is only affected when MSR0 register write operation or "ER/ES M16" instruction is used.

XSD – Bit 10

This bit defines current bank of index register for index register read or write operation, and the length of index value for address modification. When this bit is set, the current bank of index register is SD0E and SD3E instead of SD0 and SD3, respectively. When clear, the current index registers are SD0 and SD3. (reset state) During indirect addressing mode, pointer register RPi is post-modified by index register value. If XSD is set, the width of index value becomes to 8-bit by concatenating extension index register and normal index register. If clear, the normal 4-bit index value is applied. The XSD bit can be modified by writing to MSR0 register or "ER/ES XSD" instruction. The XSD bit is cleared by a processor reset.

OPB/OPA – Bit 9/Bit 8

The OPB/OPA bit indicates that saturation arithmetic in the ARU is provided or not when overflow is occurred during data move or arithmetic operation. The overflow protection can be applied to A and B register respectively. If this bit is set, the saturation logic will substitute a limited value having maximum magnitude and the same sign as the source Ai register during overflow. If clear, no saturation is performed, and overflow is not protected by the MAC2424. The OPA/OPB bit can be modified by writing to MSR0 register or "ER/ES OPA/OPB" instruction. The OPA/OPB bit is cleared by a processor reset.

VS – Bit 6

The VS bit is a overflow flag for BEU(Barrel Shifter and Exponent Unit). This bit is set if arithmetic overflow is occurred during shift operation or exponent evaluation on BEU registers. When the instructions which performs BEU operation writes this bit as a overflow flag instead of VA or VB bit. The VS bit indicates that the result of a shift operation can not be represented in 16-bit SR register, or the source value of an exponent operation is all zero or all one. The VS bit can be modified by writing to MSR0 register instruction.

VB/VA – Bit 5 / Bit 4

The VA or VB bit is a overflow flag for ARU Ai accumulators. This bit is set if arithmetic overflow is occurred during arithmetic operation on Ai accumulator registers in ARU. The VA and VB bit indicates that the result of an arithmetic operation can not be represented in 24-bit A and B register in 24-bit mode operation and the result crosses 16-bit boundary in A and B register at 16-bit mode. Only one of two bits is updated by arithmetic operation according to the destination operand. The VA and VB bit can be modified simultaneously by writing to MSR0 register instruction.

N – Bit 3

The N bit is a sign flag for ARU or BEU operation result. This bit is set if ARU or BEU operation result value is a negative value, and cleared otherwise. The N flag is the same as the MSB of the output if current operation does not generate overflow. If overflow is occurred during instruction execution, the value of N flag is the negated value of the MSB of the output. The N bit can be modified by instructions writing to MSR0 register.

Z – Bit 2

The Z bit is a zero flag for ARU or BEU operation result. This bit is set when ARU or BEU operation result value is zero, and cleared otherwise. The Z bit can be modified by instructions writing to MSR0 register, explicitly.

C – Bit 1

The C bit is a carry flag for ARU or BEU operation result. This bit is set when ARU or BEU operation generates carry, and cleared otherwise. The C bit is not affected by "ELD" instruction because this instruction does not generate carry all the times. The C bit can be modified by instructions writing to MSR0 register, explicitly.

T – Bit 0

The T bit is a test flag that evaluates various conditions when "ETST" instruction is executed. This flag value can be used as a condition during executing a conditional instruction (instructions that have a suffix "T"). The conditional instructions can only be executed when the T bit is set. Otherwise, performs no operation. The T bit can be modified by instructions writing to MSR0 register, explicitly.

STATUS REGISTER 2 (MSR2)

MSR2 register of three MAC2424 status registers (MSR0, MSR1, MSR2) is used to select EI port of the MAC2424 from various flags and status information in MSR0 and MSR1 register. The MSR2 register is used at external condition generation unit in the ARU. The contents of each field definitions are described as follows.

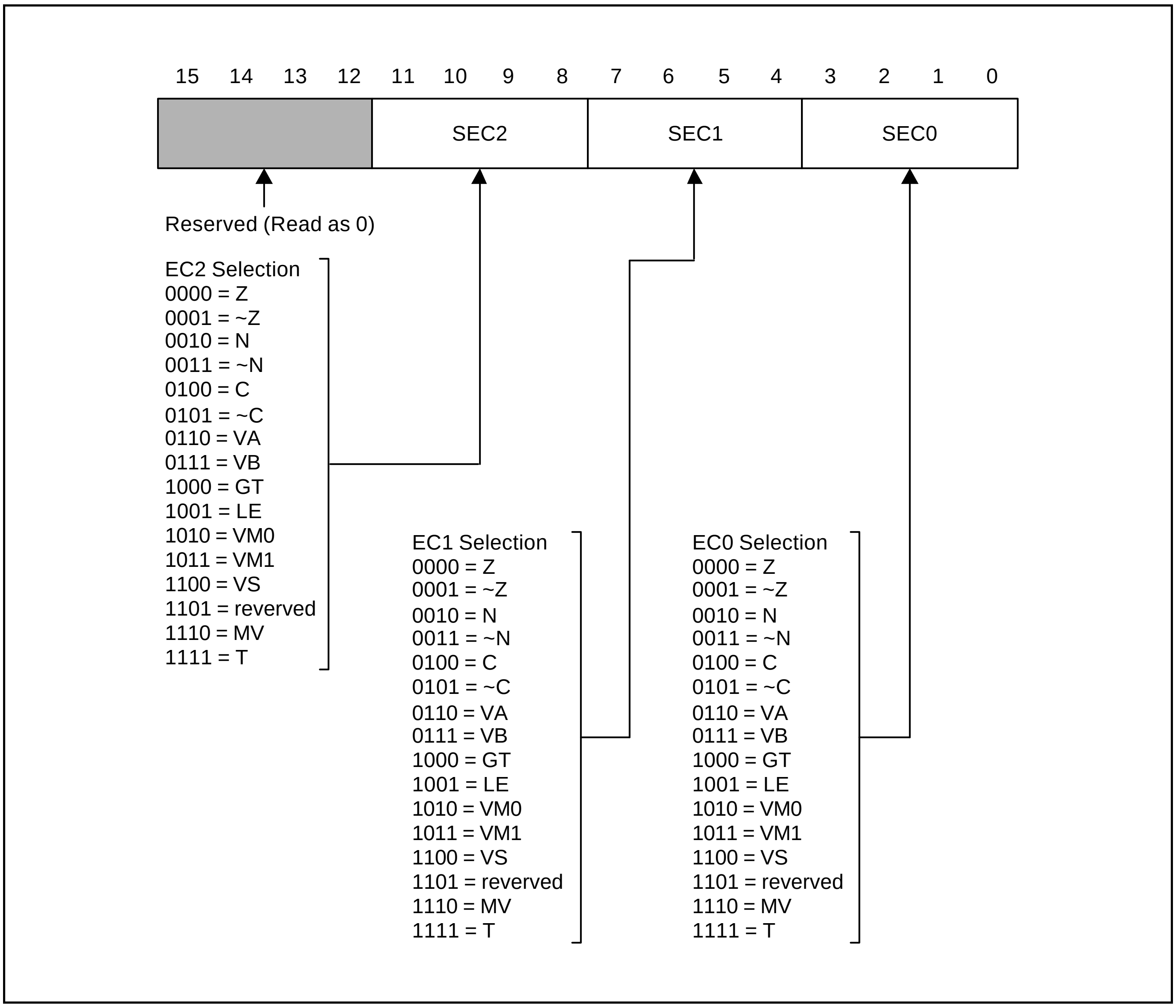


Figure 25-15. MSR2 Register Configuration

BARREL SHIFTER AND EXPONENT UNIT

The Barrel Shifter and Exponent Unit (BEU) performs several shifting operations and exponent evaluations. It contains a 16-bit, single cycle, non-pipelined barrel shifter and 24-bit exponent evaluation unit. The detailed block diagram of the Barrel Shifter and Exponent Unit is shown in figure 25-16.

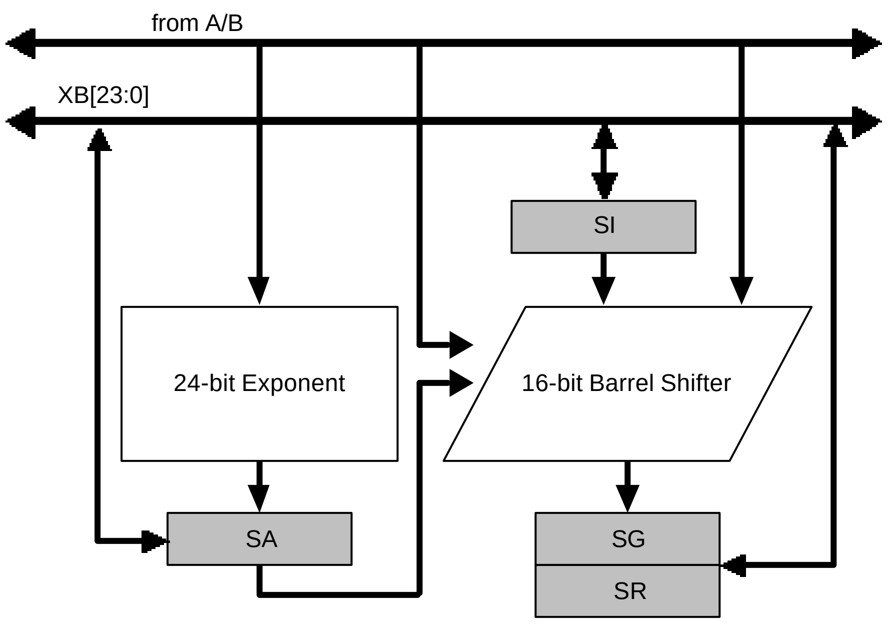


Figure 25-16. Barrel Shifter and Exponent Unit Block Diagram

BARREL SHIFTER

The barrel shifter performs standard arithmetic and logical shift, and several special shift operations. It is a 32-bit left and right, single-cycle, non-pipelined barrel shifter. The barrel shifter receives the source operand from either one of the 24-bit two Ai accumulator registers or 16-bit SI register. When selected source operand is Ai register, 16 LSBs of 24-bit register value are only valid. The upper 8-bit values are ignored. It also receives the shift amount value from either one of the 24-bit two Ai accumulator registers or 6-bit SA register. Because the maximum amount of shift is from –32 (right shift 32-bit) to +31 (left shift 31 bit), 6-bit shift amount is sufficient. When Ai register is used as the shift amount register, 6 LSBs of 24-bit register value are only valid. The amount of shifts is only determined by a value in the one of these three register and can not be determined by a constant embedded in the instruction opcode (immediate shift amount is not supported). The barrel shifter takes 16-bit input operand and 6-bit amount value, and generates 32-bit shifted output values. The destination of shifted value is two 16-bit shift output register SG and SR register. The SG register holds the value of shifted out, and the SR register holds the shifted 16-bit values.

The flags are affected as a result of the barrel shifter output, as well as a result of the ARU output. When the result is transferred into the barrel shifter output register, the flags represent the shifter output register status. The C, N, and Z flag in MSR0 register is used common to the ARU and the BEU, but the V flag is different. The ARU uses the VA and VB flags as overflow flag, and the BEU uses the VS flag as overflow flag.

Shifting Operations

Several shift operations are available using the barrel shifter, all of them are performed in a single cycle. The detailed operations of each shift instruction are depicted in figure 2.15. If 6-bit shift amount value is positive, shift left operation is performed and if negative, shift right operation is performed. After all barrel shifter operation is performed, the carry flag has the bit value which is shifted out finally.

"ESFT" instruction performs a standard logical shift operation. The shifted bit pattern is stored into the 16-bit SR register (Shifter Result register), and the shifted out bit pattern is stored into the 16-bit SG register (Shifter Guard register). When shift left operation, MSBs of SG register and LSBs of SR register is filled with zeros. When shift right operation, LSBs of SG register and MSBs of SR register is filled with zeros. "ESFTA" instruction performs a standard arithmetic shift operation. the operation is all the same as a logical shift except that the MSBs of SG register or MSBs of SR register is sign-extended instead of being filled with zeros.

"ESFTD" instruction is provided for double precision shift operation. With this instruction, one can shift 32-bit number stored in two registers. Unlike standard logical and arithmetic shift, this instruction only updates the SG register with the values that is ORed previous SG register value and shifted out result from barrel shifter. The following codes are examples of double precision shift operation.

```
// Double Precision Left ({SG,SR} ← {B,A} <<SA
ESFT        A,SA                        // Lower Part Shift
ESFTD       B,SA                        // Upper Part Shift
// Double Precision Right ({SR,SG} ← {B,A}>>SA
ESFT        B,SA                        // Upper Part Shift
ESFTD       A,SA                        // Lower Part Shift
```

"ESFTL" instruction is used for bit-stream manipulation. It links the previously shifted data with the current data. The operation of this instruction is the same as logical shift instruction except that the shifted out result is ORed with previous SG register values. This ORing process makes it possible to concatenate the previous data and the current data. This instruction is valid only when the magnitude of shift amount is greater than 16. The linking process example is as follows.

```
// Left Link ({SG,SR} ← B<<A and link SI
ESFT        B,A                       // Previous Data Shift
ESUB        A,#16                     // Preprocessing for Linking
ESFTL       SI,A                      // Current Data Shift

// Right Link ({SR,SG} ← B>>A and link SI
ESFT        B,A                       // Previous Data Shift
EADD        A,#16                     // Preprocessing for Linking
ESFTL       SI,A                      // Current Data Shift
```

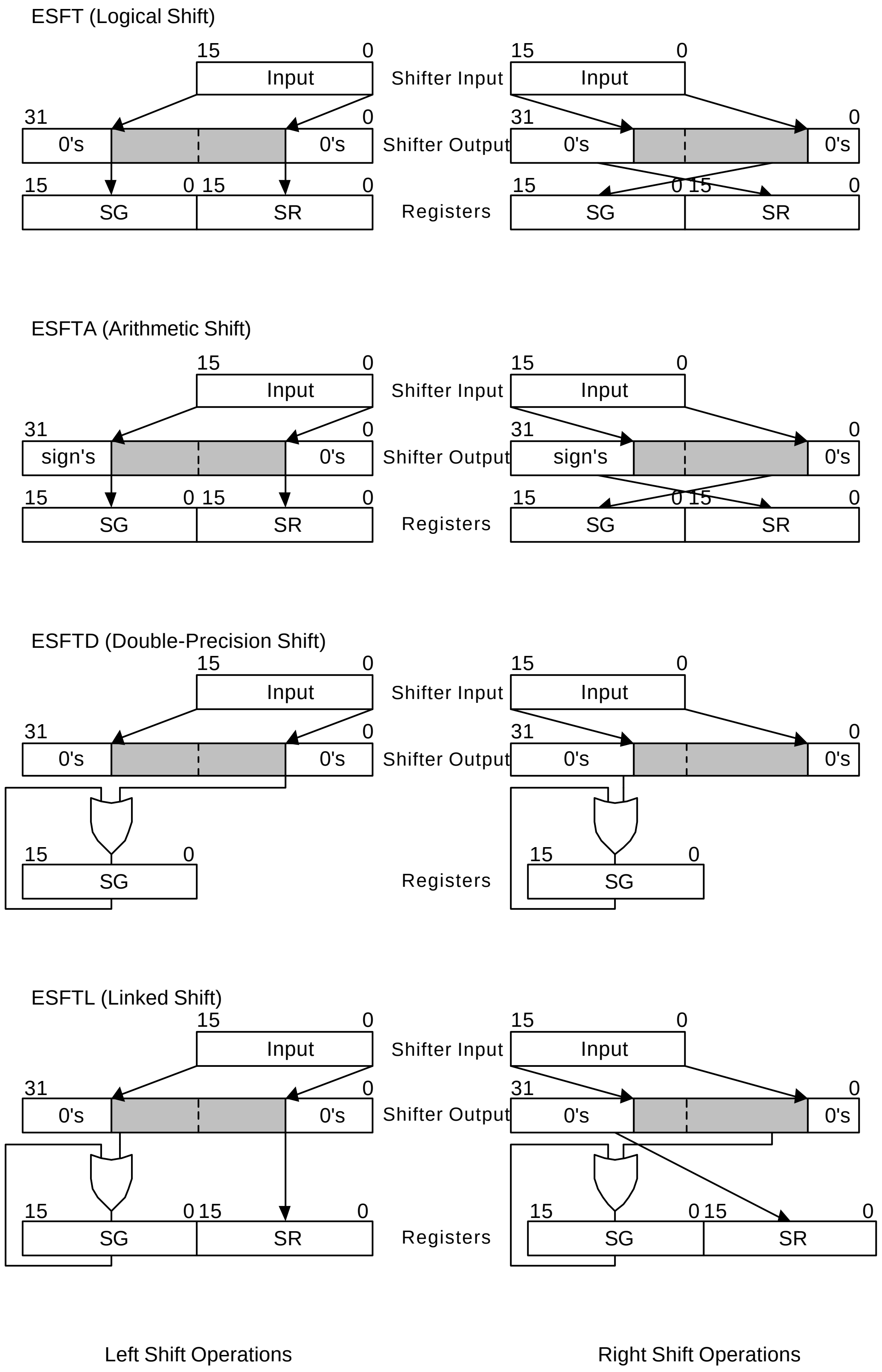


Figure 25-17. Various Barrel Shifter Instruction Operation

Bit-Field Operation

The barrel shifter supports a bit-field masking operation. This operation can be used for data bit-stream manipulation only. Various bit-field operations such as bit set, bit reset, bit change, and bit test operation is supported in CalmRISC, host processor. So the MAC2424 need not powerful bit operation capabilities. "ENMSK" instruction is provided for bit-pattern masking. This instrucion masks MSBs of SG register with selected mask pattern. The mask pattern is generated according to the 4-bit immediate operand embedded in the instruction.

EXPONENT BLOCK

The exponent block performs exponent evaluation of one of the two 24-bit accumulator registers Ai. The result of this operation is a signed 6-bit value, and transferred into the Shift Amount register (SA). The source operand is unaffected by this calculation.

The algorithm for determining the exponent result for a 24-bit number is as follows. Let N be the number of the sign bits (i.e. the number of MSBs equal to bit 23) found in the evaluated number. The exponent result is N-1. This means that the exponent is evaluated with respect to bit 24. Therefore, the exponent result is always greater than or equal to zero. (Refer to following table as examples) A non-zero result represents an un-normalized number. When evaluating the exponent value of one of the Ai accumulator, the result is the amount of left shifts that should be executed in order to normalize the source operand. An exponent result equal to zero represents a normalized number.

Table 25-2. Exponent Evaluation and Normalization Example

Evaluated Number	N	Exponent Result	Normalized Number
00001101….	4	3 (shift left by 3)	01101….
11101010….	3	2 (shift left by 2)	101010…
00000011….	6	5 (shift left by 5)	011….....
11111011….	5	4 (shift left by 4)	1011…….

Normalization

Full normalization can be achieved in 2 cycles, using "EEXP" instruction, followed by "ESFT" instruction. The "EEXP" instruction evaluates the exponent value of one of the Ai register. The second instruction "ESFT" is shifting the evaluated number, according to the exponent result stored at SA register.

```
// Normalization
EEXP A
ESFT A,SA
```

The block normalization is also possible using the exponent unit and "EMIN" instruction. The "EMIN" instruction can select the minimum exponent value from all evaluated exponent result.

Double Precision Supports

The MAC2424 Coprocessor has an instruction which can evaluate exponent values of double precision 48-bit data operand. Double precision exponent evaluation can be achieved in 2 cycles, using a standard exponent valuation instruction ("EEXP"), followed by "EEXPC" instruction. The "EEXP" instruction sets the VS flag when the source operand has the all one value or the all zero value and sets the C flag with the LSB bit value of the source operand. The C flag transfer the sign information of higher 24-bit data. After "EEXP" instruction is executed, the "EEXPC" instruction evaluates the exponent value of lower 24-bit data and carry if the VS flag is set. And then the calculated exponent value is added with previous SA register value. In this way, full double precision exponent calculation can be done.

```
// Double Precision Exponent Evaluation about {A,B}
EEXP        A
EEXPC       B
```

INSTRUCTION SET MAP AND SUMMARY

ADDRESSING MODES

Various addressing modes, including indirect linear and modulo addressing, short and long direct addressing, and immediate, are implemented in the MAC2424 coprocessor.

(1) Indirect Addressing Mode

Indirect Addressing for Read Operation
@RP0+S0, @RP0+S1, @RP1+S0, @RP1+S1, @RP2+S0, @RP2+S1, @RP3+S0, @RP3+S1
One of the RPU pointer registers (RP0, RP1, RP2, RP3) points to one of the 32K data words. The data location content, pointed to by the pointer register, is the source operand. The RPi pointer register is modified with one of two 4-bit or 8-bit source index values (S0 or S1 field) which reside in the index register after the instruction is executed. The source index values are sign extended to 15-bit and added to 15-bit pointer values in RPi register. The RP1 and RP2 register can only use 4-bit source index value. The RP0 and RP3 register can use extended 8-bit source index value if XSD bit of MSR0 register is set.

EADD A, @RP0+S1 (When XSD = 1)

	Before Execution	After Execution
A	008010h	**008021h**
RP0 (no modulo)	0010h	**0033h**
Data Loacation 10h	000011h	000011h
SD0E	01**2**2h	0122h
SD0	F3**3**3h	F333h

Figure 25-18. Indirect Addressing Example I (Read Operation)

Indirect Addressing for Write Operation

@RP0+D0, @RP0+D1, @RP1+D0, @RP1+D1, @RP2+D0, @RP2+D1, @RP3+D0, @RP3+D1

One of the RPU pointer registers (RP0, RP1, RP2, RP3) points to one of the 32K data words. The data location content, pointed to by the pointer register, is the destination operand. The RPi pointer register is modified with one of two 4-bit or 8-bit destination index values (D0 or D1 field) which reside in the index register after the instruction is executed. The destination index values are sign extended to 15-bit and added to 15-bit pointer value in RPi register. The RP1 and RP2 register can only use 4-bit source index value. The RP0 and RP3 register can use extended 8-bit source index value if XSD bit of MSR0 register is set.

ELD @RP1+D0, B

	Before Execution	After Execution
B	008010h	008010h
RP1 (no modulo)	0020h	**0018h**
Data Loacation 20h	000011h	**008010h**
SD1	1**8**19h	1819h

Figure 25-19. Indirect Addressing Example II (Write Operation)

(2) Direct Addressing Mode

Short direct Addressing
form I : rpd1.adr:4
form II: rpdi.adr:5
The data location, one of the 32K data word, is one of the source operand or destination operand. The 15-bit data location is composed of the page number in the MSB 10 or 11 bits of RPD0 or RPD1 register (except bit 15) and the direct address field (the offset in the page) in the instruction code. The short direct addressing form I only uses RPD1 register as a page value, and the form II uses RPD0 or RPD1 register specified in instruction code. The LSB 5 or 6 bits of RPD0 or RPD1 register is not used at all. And the bit 15 of RPD0 or RPD1 register is not used, either.

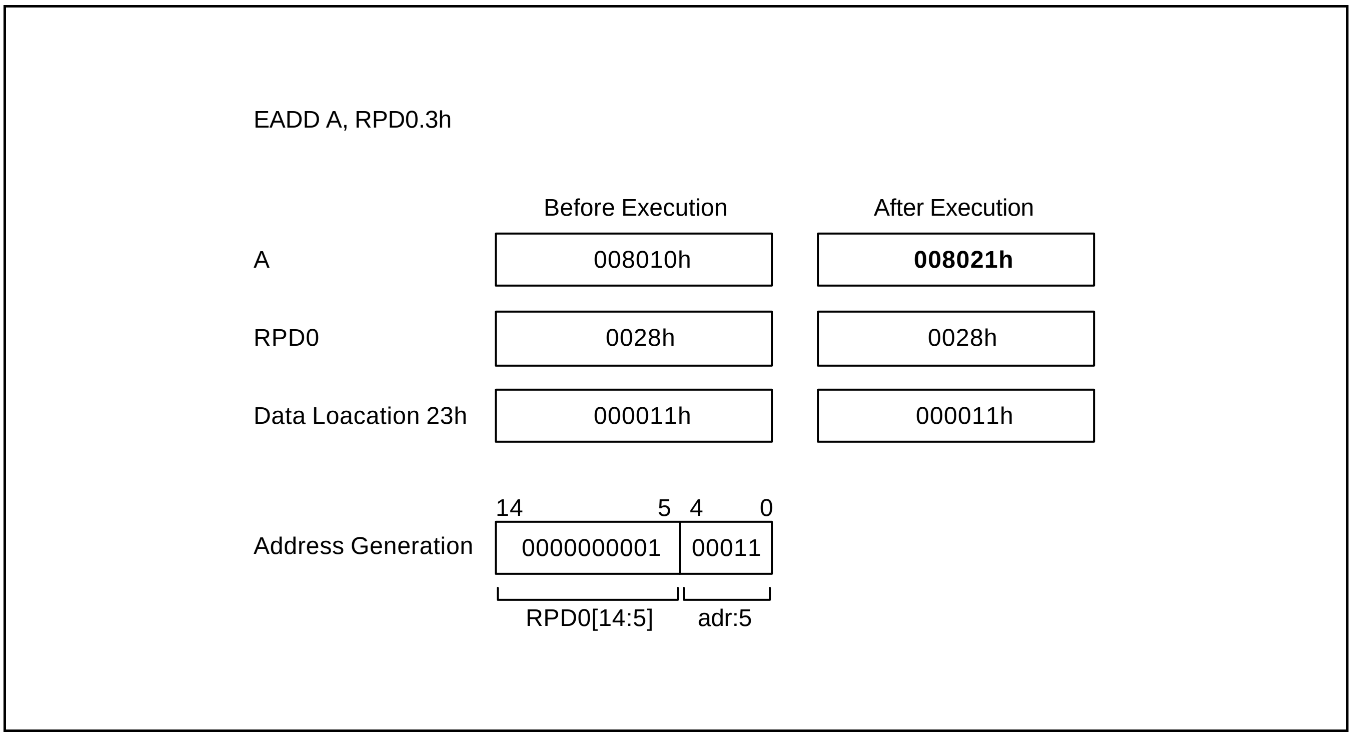


Figure 25-20. Short Direct Addressing Example

Long Direct Addressing
adr:15
The data location, one of the 32K data word, is one of the source operand or destination operand. The 15-bit data location is specified as the second word of the instruction. There is no use of the page bits in the RPDi register in this mode.

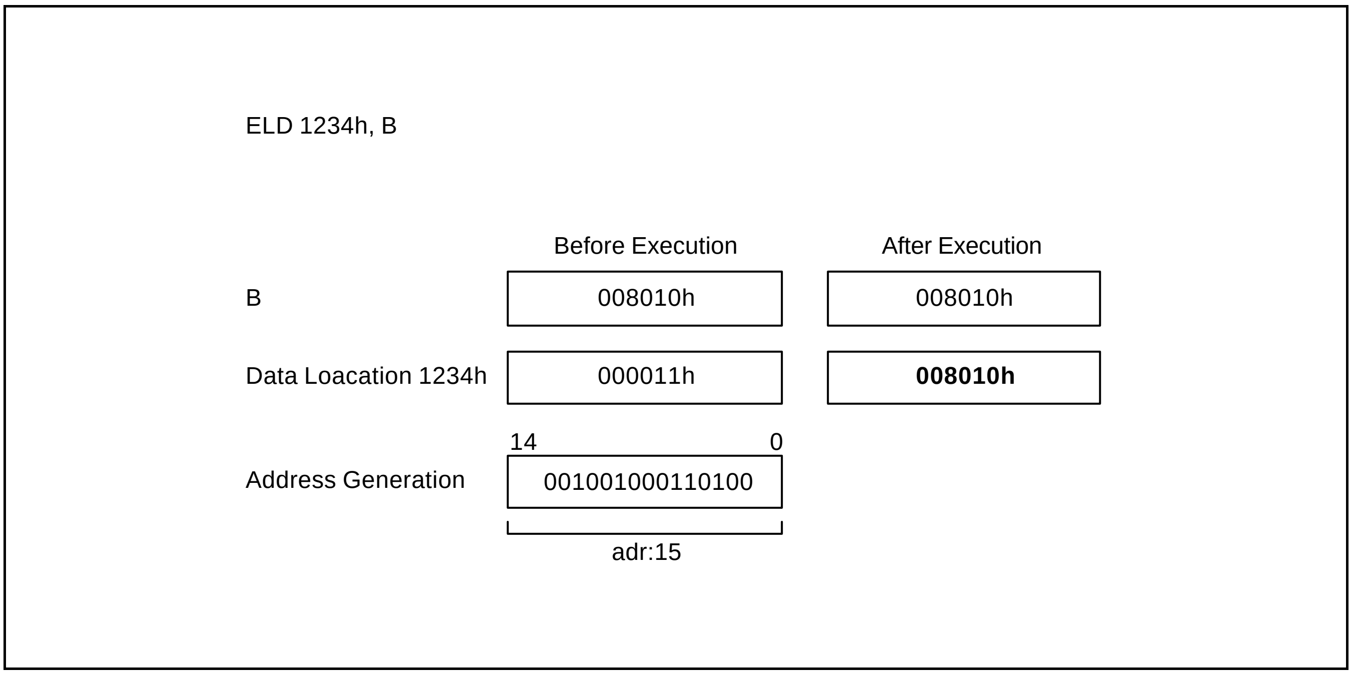


Figure 25-21. Long Direct Addressing Example

(3) Immediate Mode

Short Immediate
form I : #imm:4
form II: #imm:5
The form I is used for 4-bit register field load in "ESDi" instruction and "ESECi" instruction, or masking pattern generation in "ENMSK" instruction. The form II is used for one of the source operands. The 5-bit value is right-justified and sign-extended to the 24-bit operand.

Long Immediate
form I : #imm:15
form II: #imm:16
The form I is used only when “ERPN” instruction is executed. The 15-bit immeidate value is used as an long index values for address pointer register modification. The form II is used for one of the source operands. The 16-bit value is right-justified and sign-extended to the 24-bit operand when the destination operand is 24-bit. When the destination register has 16-bit width, the immediate value is no changed. The long immediate requires the second instruction code.

INSTRUCTION CODING

(1) Abbreviation Definition and Encoding

- rps

Mnemonic	Encoding	Description
RP0+S0	000	RP0 post-modified by SD0 S0 field
RP0+S1	001	RP0 post-modified by SD0 S1 field
RP1+S0	010	RP1 post-modified by SD1 S0 field
RP1+S1	011	RP1 post-modified by SD1 S1 field
RP2+S0	100	RP2 post-modified by SD2 S0 field
RP2+S1	101	RP2 post-modified by SD2 S1 field
RP3+S0	110	RP3 post-modified by SD3 S0 field
RP3+S1	111	RP3 post-modified by SD3 S1 field

- rpd

Mnemonic	Encoding	Description
RP0+D0	000	RP0 post-modified by SD0 D0 field
RP0+D1	001	RP0 post-modified by SD0 D1 field
RP1+D0	010	RP1 post-modified by SD1 D0 field
RP1+D1	011	RP1 post-modified by SD1 D1 field
RP2+D0	100	RP2 post-modified by SD2 D0 field
RP2+D1	101	RP2 post-modified by SD2 D1 field
RP3+D0	110	RP3 post-modified by SD3 D0 field
RP3+D1	111	RP3 post-modified by SD3 D1 field

- rp0s

Mnemonic	Encoding	Description
RP0+S0	0	RP0 post-modified by SD0 S0 field
RP0+S1	1	RP0 post-modified by SD0 S1 field

- rp3s

Mnemonic	Encoding	Description
RP3+S0	0	RP3 post-modified by SD3 S0 field
RP3+S1	1	RP3 post-modified by SD3 S1 field

Abbreviation Definition and Encoding (Continued)

- **mg1/mg1d/mg1s**

Mnemonic	Encoding	Description
Y0	000	Y0[23:0] register
Y1	001	Y1[23:0] register
X0	010	X0[23:0] register
X1	011	X1[23:0] register
P	100	P[47:0] / P[47:24] register
PL	101	P[23:0] register
MA	110	current bank MA[51:0] / MA[47:24]
MAL	111	current bank MA[23:0]

- **mg2/mg2d/mg2s**

Mnemonic	Encoding	Description
RP0	000	RP0[15:0] register
RP1	001	RP1[15:0] register
RP2	010	RP2[15:0] register
RP3	011	RP3[15:0] register
RPD0	100	RPD0[15:0] register
RPD1	101	RPD1[15:0] register
MC0	110	MC0[15:0] register
MC1	111	MC1[15:0] register

- **rpi**

Mnemonic	Encoding	Description
RP0	00	RP0[15:0] register
RP1	01	RP1[15:0] register
RP2	10	RP2[15:0] register
RP3	11	RP3[15:0] register

- **sdi/sdis/ sdid**

Mnemonic	Encoding	Description
SD0	00	current bank of SD0[15:0] register (SD0 or SD0E)
SD1	01	SD1[15:0] register
SD2	10	SD2[15:0] register
SD3	11	current bank of SD3[15:0] register (SD3 or SD3E)

Abbreviation Definition and Encoding (Continued)

- **mg**

Mnemonic	Encoding	Description
Y0	00000	Y0[23:0] register
Y1	00001	Y1[23:0] register
X0	00010	X0[23:0] register
X1	00011	X1[23:0] register
P	00100	P[47:0] / P[47:24] register
PL	00101	P[23:0] register
MA	00110	current bank MA[51:0] / MA[47:24] register
MAL	00111	current bank MA[23:0] register
RP0	01000	RP0[15:0] register
RP1	01001	RP1[15:0] register
RP2	01010	RP2[15:0] register
RP3	01011	RP3[15:0] register
RPD0	01100	RPD0[15:0] register
RPD1	01101	RPD1[15:0] register
MC0	01110	MC0[15:0] register
MC1	01111	MC1[15:0] register
SD0	01000	current bank of SD0[15:0] register (SD0 or SD0E)
SD1	01001	SD1[15:0] register
SD2	01010	SD2[15:0] register
SD3	01011	current bank of SD3[15:0] register (SD3 or SD3E)
SA	01100	SA[5:0] register
SI	01101	SI[15:0] register
SG	01110	SG[15:0] register
SR	01111	SR[15:0] register
MSR0	11000	MSR0[15:0] register
MSR1	11001	MSR1[15:0] register
MSR2	11010	MSR2[15:0] register
–	11011	reserved
MASR	11100	arithmetic right one bit shifted current MA[47:24] register
MASL	11101	arithmetic left one bit shifted current MA[47:24] register
MARN	11110	rounded current MA[47:24] register
PRN	11111	rounded P[47:24] register

* Grayed Field : read only register

Abbreviation Definition and Encoding (Continued)

- **mgx**

Mnemonic	Encoding	Description
Y0	00	Y0[23:0] register
Y1	01	Y1[23:0] register
X0	10	X0[23:0] register
X1	11	X1[23:0] register

- **mga**

Mnemonic	Encoding	Description
P	00	P[47:0] / P[47:24] register
A	01	A[23:0] register
MA	10	current bank MA[51:0] / MA[47:24] register
B	11	B[23:0] register

- **srg/srgd/srgs**

Mnemonic	Encoding	Description
SA	00	SA[5:0] register
SI	01	SI[15:0] register
SG	10	SG[15:0] register
SR	11	SR[15:0] register

- **asr**

Mnemonic	Encoding	Description
A	00	A[15:0] register
B	01	B[15:0] register
SI	10	SI[15:0] register
SR	11	SR[15:0] register

- **asa**

Mnemonic	Encoding	Description
A	00	A[5:0] register
B	01	B[5:0] register
SA	10	SA[5:0] register
–	11	reserved

Abbreviation Definition and Encoding (Continued)

- **Ai**

Mnemonic	Encoding	Description
A	0	A[23:0] register
B	1	B[23:0] register

- **mci**

Mnemonic	Encoding	Description
MC0	0	MC0[15:0] register
MC1	1	MC1[15:0] register

- **bs**

Mnemonic	Encoding	Description
OPA	0000	MSR0[5]
OPB	0001	MSR0[6]
–	0010	reserved
–	0011	reserved
ME0	0100	RP0[15]
ME1	0101	RP1[15]
ME2	0110	RP2[15]
ME3	0111	RP3[15]
OPM	1000	MSR1[3]
PSH1	1001	MSR1[4]
USM	1010	MSR1[5]
BKMA	1011	MSR1[6]
MV	1100	MSR1[2]
XSD	1101	MSR1[9]
M16	1110	MSR1[10]
VM	1111	MSR1[1] or MSR1[0] by current MA register bank

- **ns**

Mnemonic	Encoding	Description
S0	00	SDi[3:0] register
S1	01	SDi[7:4] register
D0	10	SDi[11:8] register
D1	11	SDi[15:12] register

Abbreviation Definition and Encoding (Continued)

- **ereg**

Mnemonic	Encoding	Description
AHL/AH	0000	A[23:16] register
AHL/AH	0001	A[23:16] register
ALH/AL	0010	A[15:8] register or A[15:0] register
ALL/AL	0011	A[7:0] register or A[15:0] register
BHL/BH	0100	B[23:16] register
BHL/BH	0101	B[23:16] register
BLH/BL	0110	B[15:8] register or B[15:0] register
BLL/BL	0111	B[7:0] register or B[15:0] register
SA	1000	SA[5:0] register
SA	1001	SA[5:0] register
SIH/SI	1010	SI[15:8] register or SI[15:0] register
SIL/SI	1011	SI[7:0] register or SI[15:0] register
SGH/SG	1100	SG[15:8] register or SG[15:0] register
SGL/SG	1101	SG[7:0] register or SG[15:0] register
SRH/SR	1110	SR[15:8] register or SR[15:0] register
SRL/SR	1111	SR[7:0] register or SR[15:0] register

- 1st : CalmRISC8 as a host / 2nd : CalmRISC16 as a host

Abbreviation Definition and Encoding (Continued)

- **cct**

Mnemonic	Encoding	Description
Z	0000	Z = 1
NZ	0001	Z = 0
NEG	0010	N = 1
POS	0011	N = 0
C	0100	C = 1
NC	0101	C = 0
VA	0110	VA = 1
VB	0111	VB = 1
GT	1000	N = 0 and Z = 0
LE	1001	N = 1 or Z = 1
VM0	1010	VM0 = 1
VM1	1011	VM1 = 1
VS	1100	VS = 1
–	1101	reserved
MV	1110	MV = 1
–	1111	reserved

Abbreviation Definition and Encoding (Continued)

- **emod0**

Mnemonic	Encoding	Description
ELD	00	Load
EADD	01	Add
ESUB	10	Subtract
ECP	11	Compare

- **emod1**

Mnemonic	Encoding	Description
ESRA(T)	0000	Arithmetic shift right 1-bit
ESLA(T)	0001	Arithmetic shift left 1-bit
ESRA8(T)	0010	Arithmetic shift right 8-bit
ESLA8(T)	0011	Arithmetic shift left 8-bit
ESRC(T)	0100	Arithmetic shift right 1-bit with Carry
ESLC(T)	0101	Arithmetic shift left 1-bit with Carry
EINCC(T)	0110	Increment with Carry
EDECC(T)	0111	Decrement with Carry
ENEG(T)	1000	Negate
EABS(T)	1001	Absolute
EFS16(T)	1010	Force to Sign bit 23 ~ bit 8 by bit 7
EFZ16(T)	1011	Force to Zero bit 23 ~ bit 8
EFS8(T)	1100	Force to Sign bit 23 ~ bit 16 by bit 15
EFZ8(T)	1101	Force to Zero bit 23 ~ bit 16
EEXP(T)	1110	Exponent detection
EEXPC(T)	1111	Exponent detection with Carry

NOTE: "T" suffix means that instruction is executed when T flag is set.

- **XiYi**

Mnemonic	Encoding	Description
X0Y0	00	X0[23:0] * Y0[23:0]
X0Y1	01	X0[23:0] * Y1[23:0]
X1Y0	10	X1[23:0] * Y0[23:0]
X1Y1	11	X1[23:0] * Y1[23:0]

Abbreviation Definition and Encoding (Continued)

- **emod2**

Mnemonic	Encoding	Description
ESRA	0000	Arithmetic shift right 1-bit
ESLA	0001	Arithmetic shift left 1-bit
ERND	0010	Rounding
ECR	0011	Clear
ESAT	0100	Saturate
ERESR	0101	Restore Remainder
–	0110	reserved
–	0111	reserved
ELD MA0,MA1	1000	Load from MA1 to MA0
ELD MA1,MA0	1001	Load from MA0 to MA1
EADD MA,P	1010	Add MA and P
ESUB MA,P	1011	Subtract P from MA
EADD MA,PSH	1100	Add MA and 24-bit right shifted P
ESUB MA,PSH	1101	Subtract 24-bit right shifted P from MA
EDIVQ	1110	Division Step
–	1111	reserved

- **Xi**

Mnemonic	Encoding	Description
X0	0	X0[23:0] register
X1	1	X1[23:0] register

- **Yi**

Mnemonic	Encoding	Description
Y0	0	Y0[23:0] register
Y1	1	Y1[23:0] register

- **rs**

Mnemonic	Encoding	Description
ER	0	Bit Reset Instruction
ES	1	Bit Set Instruction

(2) Overall COP instruction set map

Instruction	11	10	9	8	7	6	5	4	3	2	1	0
EMAD XiYi mgx,@rps	0	0	0	0	0	XiYi		mgx		rps		
EMSB XiYi mgx,@rps	0	0	0	0	1	XiYi		mgx		rps		
EMLD XiYi mgx,@rps	0	0	0	1	0	XiYi		mgx		rps		
EMUL XiYi mgx,@rps	0	0	0	1	1	XiYi		mgx		rps		
EADD A,MA mgx,@rps	0	0	1	0	0	0	0	mgx		rps		
ESUB A,MA mgx,@rps	0	0	1	0	0	0	1	mgx		rps		
ELD A,MA mgx,@rps	0	0	1	0	0	1	0	mgx		rps		
EADD MA,P mgx,@rps	0	0	1	0	0	1	1	mgx		rps		
ESUB MA,P mgx,@rps	0	0	1	0	1	0	0	mgx		rps		
ELD MA,P mgx,@rps	0	0	1	0	1	0	1	mgx		rps		
EADD MA,P @rpd,mga	0	0	1	0	1	1	0	mga		rpd		
ESUB MA,P @rpd,mga	0	0	1	0	1	1	1	mga		rpd		
ELD MA,P @rpd,mga	0	0	1	1	0	0	0	mga		rpd		
EADD A,MA @rpd,mga	0	0	1	1	0	0	1	mga		rpd		
ESUB A,MA @rpd,mga	0	0	1	1	0	1	0	mga		rpd		
ELD A,MA @rpd,mga	0	0	1	1	0	1	1	mga		rpd		
EADD A,MA MA,@rps	0	0	1	1	1	0	0	0	0	rps		
ESUB A,MA MA,@rps	0	0	1	1	1	0	0	0	1	rps		
ELD A,MA MA,@rps	0	0	1	1	1	0	0	1	0	rps		
EADD MA,P A,@rps	0	0	1	1	1	0	0	1	1	rps		
ESUB MA,P A,@rps	0	0	1	1	1	0	1	0	0	rps		
ELD MA,P A,@rps	0	0	1	1	1	0	1	0	1	rps		
ELD Ai,@rps	0	0	1	1	1	0	1	1	Ai	rps		
EADD Ai,@rps	0	0	1	1	1	1	0	0	Ai	rps		
ESUB Ai,@rps	0	0	1	1	1	1	0	1	Ai	rps		
ECP Ai,@rps	0	0	1	1	1	1	1	0	Ai	rps		
ELD @rpd,Ai	0	0	1	1	1	1	1	1	Ai	rpd		
ELD mg1,@rps	0	1	0	0	0	0	mg1			rps		
ELD @rpd,mg1	0	1	0	0	0	1	mg1			rpd		

NOTE: "d" means DON'T CARE.

Overall COP instruction set map (Continued)

Instruction	11	10	9	8	7	6	5	4	3	2	1	0
ELD srg,@rps	0	1	0	0	1	0	0	srg		rps		
ELD @rpd,srg	0	1	0	0	1	0	1	srg		rpd		
EMAX Ai,@rps	0	1	0	0	1	1	0	0	Ai	rps		
EMIN Ai,@rps	0	1	0	0	1	1	0	1	Ai	rps		
ETST cct	0	1	0	0	1	1	1	0	cct			
ELD Xi,@rp0s Yi,@rp3s	0	1	0	0	1	1	1	1	Yi	rp3s	Xi	rp0s
EMAD XiYi Xi,@rp0s Yi,@rp3s	0	1	0	1	0	0	XiYi		Yi	rp3s	Xi	rp0s
EMSB XiYi Xi,@rp0s Yi,@rp3s	0	1	0	1	0	1	XiYi		Yi	rp3s	Xi	rp0s
EMLD XiYi Xi,@rp0s Yi,@rp3s	0	1	0	1	1	0	XiYi		Yi	rp3s	Xi	rp0s
EMUL XiYi Xi,@rp0s Yi,@rp3s	0	1	0	1	1	1	XiYi		Yi	rp3s	Xi	rp0s
ELD rpi,rpd1.adr:4	0	1	1	0	0	0	rpi		adr:4			
ELD rpd1.adr:4,rpi	0	1	1	0	0	1	rpi		adr:4			
ELD rpdi.adr:5,Ai	0	1	1	0	1	rpdi	Ai	adr:5				
ELD Ai,rpdi.adr:5	0	1	1	1	0	rpdi	Ai	adr:5				
EADD Ai,rpdi.adr:5	0	1	1	1	1	rpdi	Ai	adr:5				
ESUB Ai,rpdi.adr:5	1	0	0	0	0	rpdi	Ai	adr:5				
ECP Ai,rpdi.adr:5	1	0	0	0	1	rpdi	Ai	adr:5				
ELD mgx,#imm:16	1	0	0	1	0	0	mgx		imm:16			
ELD rpi,#imm:16	1	0	0	1	0	1	rpi		imm:16			
ELD sdi,#imm:16	1	0	0	1	1	0	sdi		imm:16			
ERPN rpi,#imm:15	1	0	0	1	1	1	0	rpi		Imm:15		
ELD mci,#imm:16	1	0	0	1	1	1	1	mci	imm:16			
ELD Ai,#imm:16	1	0	1	0	0	0	0	Ai	imm:16			
EADD Ai,#imm:16	1	0	1	0	0	0	1	Ai	imm:16			
ESUB Ai,#imm:16	1	0	1	0	0	1	0	Ai	imm:16			
ECP Ai,#imm:16	1	0	1	0	0	1	1	Ai	imm:16			
ELD adr:15,Ai	1	0	1	0	1	0	0	0	Ai	adr:15		
ELD Ai,adr:15	1	0	1	0	1	0	0	1	Ai	adr:15		
EADD Ai,adr:15	1	0	1	0	1	0	1	0	Ai	adr:15		
ESUB Ai,adr:15	1	0	1	0	1	0	1	1	Ai	adr:15		
ECP Ai,adr:15	1	0	1	0	1	1	0	0	Ai	adr:15		

NOTE: "d" means DON'T CARE.
"Gray" means 2 word Instruction

Overall COP instruction set map (Continued)

Instruction	11	10	9	8	7	6	5	4	3	2	1	0
EMOD2 MA	1	0	1	0	1	1	0	1	emod2			
ER/ES bs	1	0	1	0	1	1	1	rs	bs			
ELD mg1d,mg1s	1	0	1	1	0	0	mg1d			mg1s		
ELD mg2d,mg2s	1	0	1	1	0	1	mg2d			mg2s		
ELD sdid,sdis	1	0	1	1	1	0	0	0	sdid		sdis	
ELD srgd,srgs	1	0	1	1	1	0	0	1	srgd		srgs	
ERPS rps	1	0	1	1	1	0	1	0	0	rps		
ERPD rpd	1	0	1	1	1	0	1	0	1	rpd		
ERPR rpi	1	0	1	1	1	0	1	1	0	0	rpi	
reserved	1	0	1	1	1	0	1	1	1	ddd		
EMOD1 Ai	1	0	1	1	1	1	ts	Ai	emod1			
EMAD XiYi A,MA	1	1	0	0	0	0	0	0	0	0	XiYi	
EMSB XiYi A,MA	1	1	0	0	0	0	0	0	0	1	XiYi	
EMLD XiYi A,MA	1	1	0	0	0	0	0	0	1	0	XiYi	
EMUL XiYi A,MA	1	1	0	0	0	0	0	0	1	1	XiYi	
EMAD XiYi A,MASL	1	1	0	0	0	0	0	1	0	0	XiYi	
EMSB XiYi A,MASL	1	1	0	0	0	0	0	1	0	1	XiYi	
EMLD XiYi A,MASL	1	1	0	0	0	0	0	1	1	0	XiYi	
EMUL XiYi A,MASL	1	1	0	0	0	0	0	1	1	1	XiYi	
EMAD XiYi A,MASR	1	1	0	0	0	0	1	0	0	0	XiYi	
EMSB XiYi A,MASR	1	1	0	0	0	0	1	0	0	1	XiYi	
EMLD XiYi A,MASR	1	1	0	0	0	0	1	0	1	0	XiYi	
EMUL XiYi A,MASR	1	1	0	0	0	0	1	0	1	1	XiYi	
EMAD XiYi	1	1	0	0	0	0	1	1	0	0	XiYi	
EMSB XiYi	1	1	0	0	0	0	1	1	0	1	XiYi	
EMLD XiYi	1	1	0	0	0	0	1	1	1	0	XiYi	
EMUL XiYi	1	1	0	0	0	0	1	1	1	1	XiYi	
ESFT asr,asa	1	1	0	0	0	1	0	0	asa		asr	
ESFTA asr,asa	1	1	0	0	0	1	0	1	asa		asr	
ESFTL asr,asa	1	1	0	0	0	1	1	0	asa		asr	
ESFTD asr,asa	1	1	0	0	0	1	1	1	asa		asr	

NOTE: "d" means DON'T CARE.

Overall COP instruction set map (Continued)

Instruction	11	10	9	8	7	6	5	4	3	2	1	0
ELD SA,#imm:5	1	1	0	0	1	0	0	imm:5				
ENMSK SG,#imm:4	1	1	0	0	1	0	1	0	imm:4			
EMOD0 Aid,Ais	1	1	0	0	1	0	1	1	emod0		Aid	Ais
ELD mg,Ai	1	1	0	0	1	1	Ai	mg				
EMOD0 Ai,mg	1	1	0	1	emod0		Ai	mg				
ESD0 ns #imm:4	1	1	1	0	0	0	ns		imm:4			
ESD1 ns #imm:4	1	1	1	0	0	1	ns		imm:4			
ESD2 ns #imm:4	1	1	1	0	1	0	ns		imm:4			
ESD3 ns #imm:4	1	1	1	0	1	1	ns		imm:4			
ENOP	1	1	1	1	0	0	0	0	dddd			
ESEC0 #imm:4	1	1	1	1	0	0	0	1	imm:4			
ESEC1 #imm:4	1	1	1	1	0	0	1	0	imm:4			
ESEC2 #imm:4	1	1	1	1	0	0	1	1	imm:4			
ELD Ai,#imm:5	1	1	1	1	0	1	Ai	imm:5				
EADD Ai,#imm:5	1	1	1	1	1	0	Ai	imm:5				
ECP Ai,#imm:5	1	1	1	1	1	1	Ai	imm:5				

NOTE: "d" means DON'T CARE.

QUICK REFERENCE

opc	op1	op2	op3	op4	op5	Function	Flag
EMAD	XiYi	mgx	@rps	–	–	MA ← MA+P, P ← Xi*Yi, op2 ← op3	VMi
EMSB						MA ← MA-P, P ← Xi*Yi, op2 ← op3	VMi
EMLD						MA ← P, P ← Xi*Yi, op2 ← op3	VMi
EMUL						P ← Xi*Yi, op2 ← op3	–
EMAD	XiYi	Xi	@rp0s	Yi	@rp3s	MA ← MA+P, P ← Xi*Yi, op2 ← op3, op4 ← op5	VMi
EMSB						MA ← MA-P, P ← Xi*Yi, op2 ← op3, op4 ← op5	VMi
EMLD						MA ← P, P ← Xi*Yi, op2 ← op3, op4 ← op5	VMi
EMUL						P ← Xi*Yi, op2 ← op3, op4 ← op5	–
EMAD	XiYi	A	MA/ MASR/ MASL			MA ← MA+P, P ← Xi*Yi, op2 ← op3	VMi,Z,VA,N
EMSB						MA ← MA-P, P ← Xi*Yi, op2 ← op3	VMi,Z,VA,N
EMLD						MA ← P, P ← Xi*Yi, op2 ← op3	VMi,Z,VA,N
EMUL						P ← Xi*Yi, op2 ← op3	N,VA,Z
EMAD	XiYi	–	–	–	–	MA ← MA+P, P ← Xi*Yi	VMi
EMSB						MA ← MA-P, P ← Xi*Yi	VMi
EMLD						MA ← P, P ← Xi*Yi	VMi
EMUL						P ← Xi*Yi	–
EADD	A	MA	mgx	@rps	–	op1 ← op1+op2, op3 ← op4	C,Z,VA,N
ESUB						op1 ← op1-op2, op3 ← op4	C,Z,VA,N
ELD						op1 ← op2, op3 ← op4	Z,VA,N
EADD	MA	P	mgx	@rps	–	op1 ← op1+op2, op3 ← op4	VMi
ESUB						op1 ← op1-op2, op3 ← op4	VMi
ELD						op1 ← op2, op3 ← op4	–

QUICK REFERENCE

opc	op1	op2	op3	op4	op5	Function	Flag
EADD	A	MA	@rpd	mga	–	op1 ← op1+op2, op3 ← op4	C,Z,VA,N
ESUB						op1 ← op1-op2, op3 ← op4	C,Z,VA,N
ELD						op1 ← op2, op3 ← op4	Z,VA,N
EADD	MA	P	@rpd	mga	–	op1 ← op1+op2, op3 ← op4	VMi
ESUB						op1 ← op1-op2, op3 ← op4	VMi
ELD						op1 ← op2, op3 ← op4	–
EADD	A	MA	MA	@rps	–	op1 ← op1+op2, op3 ← op4	C,Z,VA,N,VMi
ESUB						op1 ← op1-op2, op3 ← op4	C,Z,VA,N,VMi
ELD						op1 ← op2, op3 ← op4	Z,VA,N,VMi
EADD	MA	P	A	@rps	–	op1 ← op1+op2, op3 ← op4	VMi,Z,VA,N
ESUB						op1 ← op1-op2, op3 ← op4	VMi,Z,VA,N
ELD						op1 ← op2, op3 ← op4	Z,VA,N
ELD	Ai	mg/@rps Ai/adr:15 rpdi.adr:5 #imm:16	–	–	–	op1 ← op2	Z,Vi,N
EADD						op1 ← op1+op2	C,Z,Vi,N
ESUB						op1 ← op1-op2	C,Z,Vi,N
ECP						op1-op2	C,Z,Vi,N
ELD	Ai	#imm:5				op1 ← op2	Z,Vi,N
EADD						op1 ← op1+op2	C,Z,Vi,N
ECP						op1-op2	C,Z,Vi,N
ECP	Xi	@rp0s	Yi	@rp3s	–	op1 ← op2, op3 ← op4	–

NOTE: opc - opcode, op1 - operand1, op2 - operand2, op3 - operand3, op4 - operand4, op5 - operand5
VMi - VM0 or VM1 according to the current MA bank (when VMi is written, MV is written)
Vi - VA or VB according to the Ai bit

Quick Reference (Continued)

opc	op1	op2	Function	Flag
ELD	@rpd/mg adr:15 rpdi:adr:5	Ai	op1 ← op2	–
ELD	mgx/mci	#imm:16	op1 ← op2	–
ELD	mg1	mg1/@rps	op1 ← op2	-(VMi) (note)
ELD	srg	srg/@rps	op1 ← op2	–
ELD	@rpd	mg1/srg	op1 ← op2	–
ELD	mg2	mg2	op1 ← op2	–
ELD	rpi	rpd1.adr:4 #imm:16	op1 ← op2	–
ELD	sdi	sdi/#imm:16	op1 ← op2	–
ELD	MA1	MA0	op1 ← op2	VM1
ELD	MA0	MA1	op1 ← op2	VM0
ELD	rpd1.adr:4	rpi	op1 ← op2	–
EADD	MA	P/PSH	op1 ← op1+op2	VMi
ESUB			op1 ← op1-op2	VMi
EMAX	Ai	@rps	if (N=1) op1 ← op2, RP3 ← rpi	Z,Vi,N
EMIN			if (N=0) op1 ← op2, RP3 ← rpi	Z,Vi,N
ERPN	rpi	#imm:15	op1 ← mod(op1+op2)	–
ERPS	rps	–	op1 ← mod(op1+Si)	–
ERPD	rpd	–	op1 ← mod(op1+Di)	–
ERPR	rpi	–	RP3 ← bit_reverse(op1)	–
ETST	cct	–	T ← cct	T
ER/ES	bs	–	op1 ← 0/1	–
ESFT	asr	asa	{SG,SR} ← op1<</>>op2 logical shift	C,Z,VS,N
ESFTA			{SG,SR} ← op1<</>>op2 arithmetic shift	C,Z,VS,N
ESFTD			SG ← SG|(op1<</>>op2)	C,Z,VS,N
ESFTL			SR ← op1<</>>op2, SG<-SG|(op1<</>>op2)	C,Z,VS,N

Quick Reference (Continued)

opc	op1	op2	Function	Flag
ENMSK	SG	#imm:4	SG ← SG&mask_pattern by #imm:4	Z,VS,N
ESD0	ns	#imm:4	SD0.ns ← op2	–
ESD1			SD1.ns ← op2	–
ESD2			SD2.ns ← op2	–
ESD3			SD3.ns ← op2	–
ENOP	–	–	No Operation	–
ESEC0	#imm:4	–	MSR2.SEC0 ← op2	–
ESEC1			MSR2.SEC1 ← op2	–
ESEC2			MSR2.SEC2 ← op2	–

EDIVQ	MA	P	if (VMi=0)
			    new remainder = op1-op2
			else if (VMi=1)
			    new remainder = op1+op2
			if (new remainder>0)
			    op1 ← new_remainder<<1 + 1
			else
			    op1 ← new_remainder<<1
				VMi

NOTE: VMi is affected when op1 is “MA”

Quick Reference (Continued)

opc	op1	op2	Function	Flag
ERESR	MA	–	if (VMi=1) op1 ← op1+2*P	VMi
ESLA			op1 ← op1<<1 arithmetic	VMi
ESRA			op1 ← op1>>1 arithmetic	VMi
ECR			op1 ← 0	VMi
ESAT			op1 ← saturated(op1)	VMi
ERND			op1 ← op1+800000h, op1[23:0]<-0	VMi
ESLA/ESLAT (note)	Ai	–	op1 ← op1<<1 arithmetic	C,Z,Vi,N
ESRA/ESRAT (note)			op1 ← op1>>1 arithmetic	C,Z,Vi,N
ESLA8/ESLA8T (note)			op1 ← op1<<8 arithmetic	C,Z,Vi,N
ESRA8/ESRA8T (note)			op1 ← op1>>8 arithmetic	C,Z,Vi,N
ESLC/ESLCT (note)			op1 ← {op1[22:0],C}	C,Z,Vi,N
ESRC/ESRCT (note)			op1 ← {C,op1[23:1]}	C,Z,Vi,N
EINCC/EINCCT (note)			op1 ← op1+C	C,Z,Vi,N
EDECC/EDECCT (note)			op1 ← op1-~C	C,Z,Vi,N
EABS/EABST (note)			op1 ← |op1|	C,Z,Vi,N
ENEG/ENEGT (note)			op1 ← ~op1+1	C,Z,Vi,N
EFS16/EFS16T (note)			op1[23:8] ← op1[7]	C,Z,Vi,N
EFZ16/EFZ16T (note)			op1[23:8] ← 0	C,Z,Vi,N
EFS8/EFS8T (note)			op1[23:16] ← op1[16]	C,Z,Vi,N
EFZ8/EFZ8T (note)			op1[23:16] ← 0	C,Z,Vi,N
EEXP/EEXPT (note)			SA ← exponent of (op1)	C,Z,VS,N
EEXPC/EEXPCT (note)			if (VS=1) SA ← SA+exponent of ({C,op1})	C,Z,VS,N

NOTE: if T=1, instruction is executed

INSTRUCTION SET

GLOSSARY

This chapter describes the MAC2424 instruction set, with the details of each instruction. The following notations are used for the description.

Notation	Interpretation
<opN>	Operand N. N can be omitted if there is only one operand. Typically, <op1> is the destination (and source) operand and <op2> is a source operand.
<dest>, <src>	Destination and source operand for load.
adr:N	N-bit direct address specifier
imm:N	N-bit immediate number
&	Bit-wise AND
|	Bit-wise OR
~	Bit-wise NOT
^	Bit-wise XOR
N**M	Mth power of N

It is further noted that only the affected flags are described in the tables in this section. That is, if a flag is not affected by an operation, it is NOT specified.

INSTRUCTION DESCRIPTION

EABS/EABST* – Absolute

Format: EABS(T) Ai

Operation: Ai ← |Ai|
This instruction calculates the absolute value of one of 24-bit Accumulator(Ai), and stores the result back into the same Accumulator.

Flags: C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
Vi**: set if overflow is generated. Reset if not.
N: exclusive OR of Vi and MSB of result.

Notes: * EABST instruction can be executed only when the T flag is set.
Otherwise, No operation is performed.
** Vi denotes for VA or VB according to Ai

Examples:
```
EABS   A
EABST  B
```

# of Words: 1

EADD[1)] – Add Accumulator

Format: EADD Ai, <op>
<op>: @rps
rpdi.adr: 5/adr:15
#simm: 5/#simm:16
Ai
mg

Operation: Ai ← Ai + <op>
This instruction adds the values of one of 24-bit Accumulator(Ai) and <op> together, and stores the result back into the same Accumulator.

Flags: C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
Vi*: set if overflow is generated. Reset if not.
N: exclusive OR of Vi and MSB of result.

Notes: * Vi denotes for VA or VB according to Ai

Examples:
```
EADD A, @RP0+S0
EADD B, RPD1.5h
EADD A, #0486h
EADD B, A
EADD A, RP0
```

# of Words: 1
2 when <op> is: adr:15 or #simm:16

EADD[2)] – Add Accumulator with One Parallel Move

Format:	EADD A, MA <dest>,<src> <dest>,<src>: mgx, @rps MA, @rps @rpd, mga
Operation:	A <- A + MA, <dest> <- <src> This instruction adds the values of 24-bit Accumulator A and higher 24-bit part of Multiplier Accumulator MA together, and stores the result back into Accumulator A. This instruction also stores source operand from memory or register to destination register or memory.
Flags:	C: set if carry is generated by addition. Reset if not. Z: set if result is zero by addition. Reset if not. VA: set if overflow is generated by addition. Reset if not. N: exclusive OR of V and MSB of result by addition.
Notes:	None.
Examples:	EADD A, MA X0,@RP0+S1 EADD A, MA MA,@RP1+S0 EADD A, MA @RP3+D1, A
# of Words:	1

EADD[3)] – Add Multiplier Accumulator

Format:	EADD MA, <op> <op>: P / PSH
Operation:	MA ← MA + <op> This instruction adds the values of 52-bit Multiplier Accumulator MA and <op> together, and stores the result back into Multiplier Accumulator MA.
Flags:	VMi*: set if result is overflowed to guard-bits. Reset if not. MV: set if guard-bit is overflowed. Unchanged if not.
Notes:	* VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	EADD MA, P EADD MA, PSH
# of Words:	1

EADD[4)] – Add Multiplier Accumulator with One Parallel Move

Format: EADD MA, P, <dest>,<src>
<dest>,<src>: mgx, @rps
A, @rps
@rpd, mga

Operation: MA ← MA + P, <dest> ← <src>
This instruction adds the values of 52-bit Multiplier Accumulator MA and Product Register P together, and stores the result back into Multiplier Accumulator MA. This instruction also stores source operand from memory or register to destination register or memory.

Flags: VMi*: set if result is overflowed to guard-bits. Reset if not.
MV: set if guard-bit is overflowed. Unchanged if not.

Notes: * VMi denotes for VM0 or VM1 according to the current MA bank.

Examples:

```
EADD MA, P   Y0, @RP1+S1
EADD MA, P   A, @RP2+S0
EADD MA, P   @RP0+D0, B
```

# of Words: 1

ECLD – Coprocessor Accumulator Load from host processor

Format: ECLD ereg, GPR
ECLD GPR, ereg

Operation: ereg ← GRP or GPR ← ereg
This instruction moves the selected 8-bit general purpose register value of host processor to the selected 8-bit field of Ai(A or B) accumulator register or moves the selected 8-bit field of Ai(A or B) accumulator register to the 8-bit general purpose register. This instruction is mapped to “CLD” instruction of CalmRISC microcontroller.

Flags: –

Notes: –

Examples: ECLD ALL, R0
ECLD R1, BHL

# of Words: 1

ECP – Compare Accumulator

Format: ECP Ai, <op>
<op>: @rps
rpdi.adr:5 / adr:15
#simm:5 / #simm:16
Ai
mg

Operation: Ai - <op>
This Instruction compares the values of Accumulator Ai and <op> by subtracting <op> from Accumulator. Content of Accumulator is not changed.

Flags: C: set if carry is generated. Reset if not.
Z: set if result is zero. Reset if not.
Vi*: set if overflow is generated. Reset if not.
N: exclusive OR of Vi and MSB of result.

Notes: * Vi denotes for VA or VB according to Ai

Examples:
```
ECP A, @RP0+S0
ECP B, RPD1.5h
ECP A, #0486h
ECP B, A
ECP A, RP0
```

# of Words: 1
2 when <op> is : adr:15 or #simm:16

ECR – Clear MA Accumulator

Format:	ECR MA
Operation:	MA $\leftarrow$ 0 This Instruction clears the value of current bank MA accumulator. The extension nibble of selected MA accumulator is also cleared in 24-bit operation mode and unchanged in 16-bit operation mode.
Flags:	VMi*: 0.
Notes:	* VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	ECR MA
# of Words:	1

EDECC/EDECCT* – Decrement with Carry

Format:	EDECC(T) Ai
Operation:	Ai ← Ai - ~C This instruction subtracts 1 from the value of one of 24-bit Accumulator(Ai) if current carry flag is cleared, and stores the result back into the same Accumulator.
Flags:	C: set if carry is generated. Reset if not. Z: set if result is zero. Reset if not. Vi**: set if overflow is generated. Reset if not. N: exclusive OR of Vi and MSB of result.
Notes:	* EDECCT instruction can be executed only when the T flag is set. Otherwise, No operation is performed. ** Vi denotes for VA or VB according to Ai
Examples:	EDECC A EDECCT B
# of Words:	1

EDIVQ – Division Step

Format: EDIVQ MA,P

Operation:

```
if (VMi = 0)
    Adder output ← MA – P
else
    Adder output ← MA + P
if (Adder output > 0)
    MA ← Adder output * 2 + 1
else
    MA ← Adder output * 2
```

This Instruction adds or subtracts current bank MA accumulator from P register according to the VMi bit value and calculates one bit quotient and new partial remainder.

Flags: VMi*: if (Adder output > 0) Vmi ← 0, else VMi ← 1

Notes: * VMi denotes for VM0 or VM1 according to the current MA bank.

Examples: EDIVQ MA,P

# of Words: 1

EEXP/EEXPT* – Exponent Value Evaluation

Format:	EEXP(T) Ai
Operation:	SA ← exponent(Ai) This instruction evaluates the exponent value of one of 24-bit Accumulator(Ai), and stores the result back into 5-bit SA register.
Flags:	C: set if LSB of source Ai accumulator is 1. Reset if not. Z: set if exponent evaluation result is zero. Reset if not. VS: set if the value of source Ai accumulator is all zeroes or all ones. Reset if not. N: reset.
Notes:	* EEXPT instruction can be executed only when the T flag is set. Otherwise, No operation is performed.
Examples:	EEXP A EEXPT B
# of Words:	1

EEXPC/EEXPCT* – Exponent Value Evaluation with Carry

Format:	EEXPC(T) Ai

Operation:

```
if (VS = 1)
    SA ← exponent({C,Ai})
else
    no operation
```

This instruction evaluates the exponent value which concatenates carry and one of 24-bit Accumulator(Ai), adds the result with SA register value, and stores the added result back into 5-bit SA register. It can be used for multi-precision exponent evaluation.

Flags:

C: set if LSB of source Ai accumulator is 1. Reset if not.
Z: set if exponent evaluation result is zero. Reset if not.
VS: set if the value of carry and source Ai accumulator is all zeroes or all ones. Reset if not.
N: reset.

Notes:

* EEXPCT instruction can be executed only when the T flag is set.
Otherwise, No operation is performed.

Examples:

```
EEXPC   A
EEXPCT  B
```

# of Words: 1

EFS16/EFS16T* – Force to Sign MSB16 bits

Format:	EFS16(T) Ai
Operation:	Ai ← {16{Ai[7]},Ai[7:0]} This instruction forces the value of MSB 16 bits of 24-bit Accumulator(Ai) with byte sign bit of Ai register(Ai[7]), and stores the result back into the same Accumulator.
Flags:	C: Reset. Z: set if result is zero. Reset if not. Vi**: Reset. N: MSB of result.
Notes:	* EFS16T instruction can be executed only when the T flag is set. Otherwise, No operation is performed. ** Vi denotes for VA or VB according to Ai
Examples:	EFS16 A EFS16T B
# of Words:	1

EFS8/EFS8T* – Force to Sign MSB8 bits

Format: EFS8(T) Ai

Operation: Ai $\leftarrow$ {8{Ai[15]},Ai[15:0]}
This instruction forces the value of MSB 8 bits of 24-bit Accumulator(Ai) with word sign bit of Ai register(Ai[15]), and stores the result back into the same Accumulator.

Flags: C: Reset.
Z: set if result is zero. Reset if not.
Vi**: Reset.
N: MSB of result.

Notes: * EFS8T instruction can be executed only when the T flag is set.
Otherwise, No operation is performed.
** Vi denotes for VA or VB according to Ai

Examples:

```
EFS8   A
EFS8T  B
```

# of Words: 1

EFZ16/EFZ16T* – Force to Zero MSB16 bits

Format:	EFZ16(T) Ai
Operation:	Ai $\leftarrow$ {16{0},Ai[7:0]} This instruction forces the value of MSB 16 bits of 24-bit Accumulator(Ai) with zero, and stores the result back into the same Accumulator.
Flags:	C: Reset. Z: set if result is zero. Reset if not. Vi**: Reset. N: Reset.
Notes:	* EFZ16T instruction can be executed only when the T flag is set. Otherwise, No operation is performed. ** Vi denotes for VA or VB according to Ai
Examples:	EFZ16 A EFZ16T B
# of Words:	1

EFZ8/EFZ8T* – Force to Zero MSB8 bits

Format:	EFZ8(T) Ai
Operation:	Ai ← {8{0},Ai[15:0]} This instruction forces the value of MSB 8 bits of 24-bit Accumulator(Ai) with zero, and stores the result back into the same Accumulator.
Flags:	C: Reset. Z: set if result is zero. Reset if not. Vi**: Reset. N : Reset.
Notes:	* EFZ8T instruction can be executed only when the T flag is set. Otherwise, No operation is performed. ** Vi denotes for VA or VB according to Ai
Examples:	EFZ8 A EFZ8T B
# of Words:	1

EINCC/EINCCT* – Increment with Carry

Format:	EINCC(T) Ai
Operation:	Ai ← Ai + C This instruction adds 1 from the value of one of 24-bit Accumulator(Ai) if current carry flag is set, and stores the result back into the same Accumulator.
Flags:	C: set if carry is generated. Reset if not. Z: set if result is zero. Reset if not. Vi**: set if overflow is generated. Reset if not. N: exclusive OR of Vi and MSB of result.
Notes:	* EINCCT instruction can be executed only when the T flag is set. Otherwise, No operation is performed. ** Vi denotes for VA or VB according to Ai
Examples:	EINCC A EINCCT B
# of Words:	1

ELD[1)] – Load Accumulator

Format:	ELD Ai, <op> <op>: @rps rpdi.adr:5 / adr:15 #simm:5 / #simm:16 Ai mg
Operation:	Ai ← <op> This instruction load <op> value to the one of 24-bit Accumulator(Ai).
Flags:	Z: set if result is zero. Reset if not. Vi*: set if overflow is generated. Reset if not. N: set if loaded value is negative.
Notes:	* Vi denotes for VA or VB according to Ai
Examples:	ELD A, @RP0+S0 ELD B, RPD1.5h ELD A, #0486h ELD B, A ELD A, RP0
# of Words:	1 2 when <op> is : adr:15 or #simm:16

ELD[2)] – Load Accumulator with One Parallel Move

Format:	ELD A, MA <dest>,<src> <dest>,<src>: mgx, @rps MA, @rps @rpd, mga
Operation:	A $\leftarrow$ MA, <dest> $\leftarrow$ <src> This instruction load higher 24-bit part of Multiplier Accumulator MA to the 24-bit Accumulator A. This instruction also stores source operand from memory or register to destination register or memory.
Flags:	Z: set if result is zero by load. Reset if not. VA: set if overflow is generated by load. Reset if not. N: set if loaded value is negative.
Notes:	None.
Examples:	ELD A, MA X0,@RP0+S1 ELD A, MA MA,@RP1+S0 ELD A, MA @RP3+D1, A
# of Words:	1

ELD[3)] – Load Multiplier Accumulator

Format:	ELD MA0, MA1 ELD MA1, MA0
Operation:	MAi ← Maj This instruction loads the value of the one 52-bit Multiplier Accumulator MA from the other Multiplier Accumulator.
Flags:	VMi*: set if result is overflowed to guard-bits. Reset if not.
Notes:	* VMi denotes for VM0 or VM1 according to destination Multiplier Accumulator.
Examples:	ELD MA1, MA0 ELD MA0, MA1
# of Words:	1

ELD[4)] – Load Multiplier Accumulator with One Parallel Move

Format: ELD MA, P   <dest>,<src>

<dest>,<src>: mgx, @rps
A, @rps
@rpd, mga

Operation: MA ← A, <dest> ← <src>

This instruction load sign-extended 48-bit Product register P to the 52-bit Multiplier Accumulator MA. This instruction also stores source operand from memory or register to destination register or memory.

Flags: VMi*: set if result is overflowed to guard-bits. Reset if not

Notes: * VMi denotes for VM0 or VM1 according to destination Multiplier Accumulator.

Examples:

```
ELD MA, P    X0,@RP0+S1
ELD MA, P    A,@RP1+S0
ELD MA, P    @RP3+D1, A
ELD MA, P    @RP0+D0, MA  ;   @RP0+D0 ← MA
                                 MA  ← P
```

# of Words: 1

ELD[5)] – Load Other Registers or Memory

Format: ELD <dest>, <src>

<dest>,<src>: mg1, @rps
srg, @rps
@rpd, Ai
@rpd, mg1
@rpd, srg
rpi, rpd1.adr:4
rpd1.adr:4, rpi
rpdi.adr:5, Ai
adr:15, Ai
mgx, #imm:16
rpi, #imm:16
sdi, #imm:16
mci, #imm:16
SA, #imm:5
mg1d, mg1s
mg2d, mg2s
sdid, sdis
srgd, srgs
mg, Ai

Operation: <dest> $\leftarrow$ <src>
This instruction load <src> value to <dest>. If the width of <src> is less than the width of <dest>, <dest> is sign-extended, and if more, LSB part of <src> is written to <dest>

Flags: No effect when <dest> is not MA or Ai.
When <dest> is MA :
VMi*: set if result is overflowed to guard-bits. Reset if not
When <dest> is Ai :
Z: set if result is zero. Reset if not.
Vi**: set if overflow is generated. Reset if not.
N: set if loaded value is negative.

Notes: * VMi denotes for VM0 or VM1 according to destination Multiplier Accumulator.
** Vi denotes for VA or VB according to Ai

Examples:
```
ELD @RP0+D0, B
ELD RPD1.5h, RP2
ELD MC0, #0486h
ELD RPD1, MC0
ELD X0, Y1
ELD MA, P ;  MAH  ← PH
```

# of Words: 1
2 when <dest> or <src> is : adr:15 or #imm:16

ELD[6)] – Double Load

Format:	ELD Xi,@rp0s Yi,@rp3s
Operation:	Xi ← operand1 by @rp0s, Yi ← operand2 by @rp3s This instruction loads two operands from data memory (one from X memory space, and the other from Y memory space) to the specified 24-bit Xi and Yi register, respectively.
Flags:	–
Notes:	–
Examples:	ELD X0,@RP0+S1 Y1,@RP3+S0
# of Words:	1

EMAD[1)] – Multiply and Add

Format:	EMAD Xi,Yi
Operation:	MA ← MA + P, P ← Xi * Yi This instruction adds the values of 52-bit Multiplier Accumulator MA and P register together, and stores the result back into Multiplier Accumulator MA. At the same time, multiplier multiplies Xi register value and Yi register value, and stores the result to the P register.
Flags:	VMi*: set if result is overflowed to guard-bits. Reset if not. MV: set if guard-bit is overflowed. Unchanged if not.
Notes:	* VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	EMAD X1Y0
# of Words:	1

EMAD[2)] – Multiply and Add with One Parallel Move

Format: EMAD Xi,Yi <dest>,<src>
<dest>,<src>: A,MA
A,MASR*
A,MASL**
mgx,@rps

Operation: MA $\leftarrow$ MA + P, P $\leftarrow$ Xi * Yi, <dest> $\leftarrow$ <src>
This instruction adds the values of 52-bit Multiplier Accumulator MA and P register together, and stores the result back into Multiplier Accumulator MA. At the same time, multiplier multiplies Xi register value and Yi register value, and stores the result to the P register. This instruction also stores source operand from data memory or 24-bit higher portion of the MA register to the destination register.

Flags: VMi***: set if result is overflowed to guard-bits. Reset if not.
MV: set if guard-bit is overflowed. Unchanged if not.
When <dest> is Ai
Z: set if the value to Ai is zero by load. Reset if not.
VA: set if overflow is generated by load. Reset if not.
N: set if loaded value is negative.

Notes: * MASR: 1-bit right shifted MA[47:24]
** MASL: 1-bit left shifted MA[47:24]
*** VMi denotes for VM0 or VM1 according to the current MA bank.

Examples: EMAD X1Y0 A,MASR
EMAD X0Y0 X0,@RP1+S1

# of Words: 1

EMAD[3)] – Multiply and Add with Two Parallel Moves

Format: EMAD Xi,Yi Xi,@rp0s Yi,@rp3s

Operation: MA ← MA + P, P ← Xi * Yi, Xi ← operand1 by @rp0s, Yi ← operand2 by @rp3s
This instruction adds the values of 52-bit Multiplier Accumulator MA and P register together, and stores the result back into Multiplier Accumulator MA. At the same time, multiplier multiplies Xi register value and Yi register value, and stores the result to the P register. This instruction also stores two source operand from data memory (one from X memory space and one from Y memory space) to the 24-bit Xi register and Yi register respectively.

Flags: VMi*: set if result is overflowed to guard-bits. Reset if not.
MV: set if guard-bit is overflowed. Unchanged if not.

Notes: * VMi denotes for VM0 or VM1 according to the current MA bank.

Examples: EMAD X1Y0 X0,@RP0+S1 Y0,@RP3+S0

# of Words: 1

EMAX – Maximum Value Load

Format: EMAX Ai, <op>
<op>: @rps

Operation:

```
if (N = 1)
    Ai ← <op>, RP3 ← current pointer value
else
    No Operation (Only pointer is updated)
```

This instruction conditionally loads <op> value to the one of 24-bit Accumulator(Ai) and latches the current pointer value to the RP3 pointer when N flag of MSR0 register is set. Otherwise, no operation is performed

Flags:

Z: set if result is zero. Reset if not.
Vi*: set if overflow is generated. Reset if not.
N: set if loaded value is negative.

Notes: * Vi denotes for VA or VB according to Ai

Examples: EMAX A, @RP0+S0

# of Words: 1

EMIN – Minimum Value Load

Format: EMIN Ai, <op>
<op>: @rps

Operation:

```
if (N = 0)
    Ai ← <op>, RP3 ← current pointer value
else
    No Operation (Only pointer is updated)
```

This instruction conditionally loads <op> value to the one of 24-bit Accumulator(Ai) and latches the current pointer value to the RP3 pointer when N flag of MSR0 register is cleared. Otherwise, no operation is performed

Flags: Z: set if result is zero. Reset if not.
Vi*: set if overflow is generated. Reset if not.
N: set if loaded value is negative.

Notes: * Vi denotes for VA or VB according to Ai

Examples: EMIN B, @RP0+S0

# of Words: 1

EMLD[1)] – Multiply and Load

Format:	EMLD Xi,Yi
Operation:	MA ← P, P ← Xi * Yi This instruction loads the P register value to the values of 52-bit Multiplier Accumulator MA At the same time, multiplier multiplies Xi register value and Yi register value, and stores the result to the P register.
Flags:	VMi*: set if result is overflowed to guard-bits. Reset if not. MV: set if guard-bit is overflowed. Unchanged if not.
Notes:	* VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	EMLD X1Y0
# of Words:	1

EMLD[2)] – Multiply and Load with One Parallel Move

Format: EMLD Xi,Yi <dest>,<src>
<dest>,<src>: A,MA
A,MASR*
A,MASL**
mgx,@rps

Operation: MA ← P, P ← Xi * Yi, <dest> ← <src>
This instruction loads the P register value to the values of 52-bit Multiplier Accumulator. At the same time, multiplier multiplies Xi register value and Yi register value, and stores the result to the P register. This instruction also stores source operand from data memory or 24-bit higher portion of the MA register to the destination register.

Flags: VMi***: set if result is overflowed to guard-bits. Reset if not.
MV: set if guard-bit is overflowed. Unchanged if not.
When <dest> is Ai
Z: set if the value to Ai is zero by load. Reset if not.
VA: set if overflow is generated by load. Reset if not.
N: set if loaded value is negative.

Notes: * MASR: 1-bit right shifted MA[47:24]
** MASL: 1-bit left shifted MA[47:24]
*** VMi denotes for VM0 or VM1 according to the current MA bank.

Examples:
```
EMLD X1Y0  A,MASR
EMLD X0Y0  X0,@RP1+S1
```

# of Words: 1

EMLD[3)] – Multiply and Load with Two Parallel Moves

Format: EMLD Xi,Yi Xi,@rp0s Yi,@rp3s

Operation: MA ← P, P ← Xi * Yi, Xi ← operand1 by @rp0s, Yi ← operand2 by @rp3s
This instruction loads the P register value to the values of 52-bit Multiplier Accumulator. At the same time, multiplier multiplies Xi register value and Yi register value, and stores the result to the P register. This instruction also stores two source operands from data memory (one from X memory space and one from Y memory space) to the 24-bit Xi register and Yi register respectively.

Flags: VMi*: set if result is overflowed to guard-bits. Reset if not.
MV: set if guard-bit is overflowed. Unchanged if not.

Notes: * VMi denotes for VM0 or VM1 according to the current MA bank.

Examples: EMLD X1Y0 X0,@RP0+S1 Y0,@RP3+S0

# of Words: 1

EMSB[1)] – Multiply and Subtract

Format:	EMSB Xi,Yi
Operation:	MA ← MA - P, P ← Xi * Yi This instruction subtracts the P register from the values of 52-bit Multiplier Accumulator MA, and stores the result back into Multiplier Accumulator MA. At the same time, multiplier multiplies Xi register value and Yi register value, and stores the result to the P register.
Flags:	VMi*: set if result is overflowed to guard-bits. Reset if not. MV: set if guard-bit is overflowed. Unchanged if not.
Notes:	* VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	EMSB X1Y0
# of Words:	1

EMSB[2)] – Multiply and Subtract with One Parallel Move

Format:	EMSB Xi,Yi <dest>,<src> <dest>,<src>: A,MA A,MASR* A,MASL** mgx,@rps
Operation:	MA ← MA - P, P ← Xi * Yi, <dest> ← <src> This instruction subtracts the P register from the values of 52-bit Multiplier Accumulator MA, and stores the result back into Multiplier Accumulator MA. At the same time, multiplier multiplies Xi register value and Yi register value, and stores the result to the P register. This instruction also stores source operand from data memory or 24-bit higher portion of the MA register to the destination register.
Flags:	VMi***: set if result is overflowed to guard-bits. Reset if not. MV: set if guard-bit is overflowed. Unchanged if not. When <dest> is Ai Z: set if the value to Ai is zero by load. Reset if not. VA: set if overflow is generated by load. Reset if not. N: set if loaded value is negative.
Notes:	* MASR: 1-bit right shifted MA[47:24] ** MASL: 1-bit left shifted MA[47:24] *** VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	EMSB X1Y0 A,MASR EMSB X0Y0 X0,@RP1+S1
# of Words:	1

EMSB[3)] – Multiply and Subtract with Two Parallel Moves

Format:	EMSB Xi,Yi Xi,@rp0s Yi,@rp3s
Operation:	MA ← MA - P, P ← Xi * Yi, Xi ← operand1 by @rp0s, Yi ← operand2 by @rp3s This instruction subtracts the P register from the values of 52-bit Multiplier Accumulator MA, and stores the result back into Multiplier Accumulator MA. At the same time, multiplier multiplies Xi register value and Yi register value, and stores the result to the P register. This instruction also stores two source operand from data memory (one from X memory space and one from Y memory space) to the 24-bit Xi register and Yi register respectively.
Flags:	VMi*: set if result is overflowed to guard-bits. Reset if not. MV: set if guard-bit is overflowed. Unchanged if not.
Notes:	* VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	EMSB X1Y0 X0,@RP0+S1 Y0,@RP3+S0
# of Words:	1

EMUL[1)] – Multiply

Format:	EMLU Xi,Yi
Operation:	P ← Xi * Yi This instruction multiplies Xi register value and Yi register value, and stores the result to the P register.
Flags:	–
Notes:	–
Examples:	EMUL X1Y0
# of Words:	1

EMUL[2)] – Multiply with One Parallel Move

Format: EMUL Xi,Yi <dest>,<src>
<dest>,<src>: A,MA
A,MASR*
A,MASL**
mgx,@rps

Operation: P ← Xi * Yi, <dest> ← <src>
This instruction multiplies Xi register value and Yi register value, and stores the result to the P register. This instruction also stores source operand from data memory or 24-bit higher portion of the MA register to the destination register.

Flags: When <dest> is Ai
Z: set if the value to Ai is zero by load. Reset if not.
VA: set if overflow is generated by load. Reset if not.
N: set if loaded value is negative.

Notes: * MASR : 1-bit right shifted MA[47:24]
** MASL : 1-bit left shifted MA[47:24]

Examples: EMUL X1Y0 A,MASR
EMUL X0Y0 X0,@RP1+S1

# of Words: 1

EMUL[3)] – Multiply with Two Parallel Moves

Format:	EMUL Xi,Yi Xi,@rp0s Yi,@rp3s
Operation:	P ← Xi * Yi, Xi <- operand1 by @rp0s, Yi ← operand2 by @rp3s This instruction multiplies Xi register value and Yi register value, and stores the result to the P register. This instruction also stores two source operand from data memory (one from X memory space and one from Y memory space) to the 24-bit Xi register and Yi register respectively.
Flags:	–
Notes:	–
Examples:	EMUL X1Y0 X0,@RP0+S1 Y0,@RP3+S0
# of Words:	1

ENEG/ENEGT* – Negate

Format:	ENEG(T) Ai
Operation:	Ai ← ~Ai + 1 This instruction negates the value of one of 24-bit Accumulator(Ai), and stores the result back into the same Accumulator.
Flags:	C: set if carry is generated. Reset if not. Z: set if result is zero. Reset if not. Vi**: set if overflow is generated. Reset if not. N: exclusive OR of Vi and MSB of result.
Notes:	* ENEGT instruction can be executed only when the T flag is set. Otherwise, No operation is performed. ** Vi denotes for VA or VB according to Ai
Examples:	ENEG A ENEGT B
# of Words:	1

ENMSK – Masking SG

Format:	ENMSK SG,#imm:4
Operation:	SG ← SG & mask pattern This instruction masks MSB n bit (n = 16 - #imm:4) of SG register, and stores back the result into the SG register.
Flags:	Z: set if result is zero. Reset if not. VS: Reset. N: MSB of result.
Notes:	–
Examples:	ENMSK SG,#3h
# of Words:	1

ENOP – No Operation

Format:	ENOP
Operation:	No operation.
Flags:	–
Notes:	–
Examples:	ENOP
# of Words:	1

ER – Bit Reset

Format:	ER bs*
Operation:	specified bit in bs field ← 0 This instruction sets the specified bit in bs field to 0.
Flags:	–
Notes:	* If bs field is VM, the current bank of VMi bit is cleared. i.e. VM0 is cleared when BKMA bit is 1 and VM1 is cleared when BKMA bit is 0.
Examples:	ER OPA ER ME3
# of Words:	1

ERESR – Restoring Remainder

Format: ERESR MA,P

Operation:

```
if (VMi = 0)
    Adder output ← MA + 0
else
    Adder output ← MA + 2*P
```

This Instruction adds two times of the P register and current bank MA accumulator when VMi bit of MSR1 register is set. Else, performs no operation. It calculates true remainder value of non-restoring division.

Flags: VMi*: set if result is overflowed to guard-bits. Reset if not.
MV: set if guard-bit is overflowed. Unchanged if not.

Notes: * VMi denotes for VM0 or VM1 according to the current MA bank.

Examples: ERESR MA,P

# of Words: 1

ERND – Round

Format:	ERND MA
Operation:	MA ← MA + 0000000800000h, MA[23:0] ← 0 This Instruction adds current bank 52-bit MA accumulator and rounding constant and stores the result value into MSB part of the same register. The LSB 24-bit of the MA register is cleared. It performs two's complement rounding.
Flags:	VMi*: set if result is overflowed to guard-bits. Reset if not. MV: set if guard-bit is overflowed. Unchanged if not.
Notes:	* VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	ERND MA
# of Words:	1

ERPD – Update Pointer with Destination Index

Format:	ERPD rpd
Operation:	RPi ← mod (RPi + D0/D1) This Instruction updates the selected pointer with the selected index value. The modulo arithmetic affect the result value when ME bit of selected pointer is set. It only modifies the pointer without memory access.
Flags:	–
Notes:	–
Examples:	ERPD RP0+D1
# of Words:	1

ERPN – Update Pointer with Immediate Value

Format: ERPN rpi,#imm:15

Operation: RPi ← mod (RPi + #imm:15)

This Instruction updates the selected pointer with 15-bit immediate value. The modulo arithmetic affect the result value when ME bit of selected pointer is set. It only modifies the pointer without memory access.

Flags: –

Notes: –

Examples: ERPN RP3,#1555h

# of Words: 2

ERPR – Bit-Reverse Pointer

Format:	ERPR rpi
Operation:	RP3 ← bit-reverse (RPi) This Instruction generates the reversed bit pattern on LSB n bit of the selected pointer according to the MC1[15:13] bit values which specifies bit reverse order. (Refer to MC1 register configuration in chapter 2) The result bit pattern is written to RP3 register pointer field. The source pointer value is not changed at all and the ME bit of RP3 is not changed, either.
Flags:	–
Notes:	–
Examples:	ERPR RP2
# of Words:	1

ERPS – Update Pointer with Source Index

Format:	ERPS rps
Operation:	RPi ← mod (RPi + S0/S1) This Instruction updates the selected pointer with the selected index value. The modulo arithmetic affect the result value when ME bit of selected pointer is set. It only modifies the pointer without memory access.
Flags:	–
Notes:	–
Examples:	ERPS RP0+S1
# of Words:	1

ES – Bit Set

Format:	ES bs*
Operation:	specified bit in bs field ← 1 This instruction sets the specified bit in bs field to 1.
Flags:	–
Notes:	* If bs field is VM, the current bank of VMi bit is set. i.e. VM0 is set when BKMA bit is 1 and VM1 is set when BKMA bit is 0.
Examples:	ES OPA ES ME3
# of Words:	1

ESAT – Saturate

Format: ESAT MA

Operation: if (VMi == 1)
MA ← maximum magnitude
This Instruction sets the 52-bit MA accumulator to the plus of minus maximum value when selected MA register overflows. When no overflow occur, the MA register is not changed.

Flags: VMi*: Reset

Notes: * VMi denotes for VM0 or VM1 according to the current MA bank.

Examples: ESAT MA

# of Words: 1

ESD0/ESD1/ESD2/ESD3 – Source/Destination Index Load

Format: ESD0* ns #imm:4
ESD1 ns #imm:4
ESD2 ns #imm:4
ESD3* ns #imm:4

Operation: specified SDi register bit field in ns field ← #imm:4
This instruction loads 4-bit immediate value to the specified bit field of SDi register. Only 4-bit field of 16-bit value is changed.

Flags: –

Notes: * If XSD bit of MSR0 register is 1, the selected register is the extended index registers (SD0E and SD3E). Else, the selected register is the regular index register. (SD0 and SD3)

Examples:

```
ESD0  D0   #3h
ESD1  S1   #Fh
```

# of Words: 1

ESEC0/ESEC1/ESEC2 – EI Selection Field Load

Format:	ESEC0 #imm:4 ESEC1 #imm:4 ESEC2 #imm:4
Operation:	specified SECi (I=0~2) field of MSR2 register ← #imm:4 This instruction loads 4-bit immediate value to the specified bit field of MSR2 register. Only 4-bit field of 16-bit value is changed.
Flags:	–
Notes:	–
Examples:	ESEC0 #3h ESEC1 #Fh
# of Words:	1

ESFT – Logical Shift by Barrel Shifter

Format:	ESFT asr,asa
Operation:	{SR,SG} ← asr<<asa This instruction shifts the value of 16-bit asr values by the amount of 6-bit asa. If the value of asa is positive, left shift operation is performed, and if the value of asa is negative right shift operation is performed. The 16-bit shifted result is stored into SR register and the 16-bit shifted out result is stored into SG register. The other bits of SR and SG register are filled with zeros.
Flags:	C: set if last shifted out bit is 1. Reset if not. Unchanged when shift amount is 0. Z: set if SR result is zero. Reset if not. VS: Reset. N: MSB of SR result.
Notes:	–
Examples:	ESFT A, B ESFT SI,SA
# of Words:	1

ESFTA – Arithmetic Shift by Barrel Shifter

Format:	ESFTA asr,asa
Operation:	{SR,SG} ← asr<<asa This instruction shifts the value of 16-bit asr values by the amount of 6-bit asa. If the value of asa is positive, left shift operation is performed, and if the value of asa is negative right shift operation is performed. The 16-bit shifted result is stored into SR register and the 16-bit shifted out result is stored into SG register. The remainder MSB bits of SR or SG register are sign extended and the remainder LSB bits are filled with zeros.
Flags:	C: set if last shifted out bit is 1. Reset if not. Unchanged when shift amount is 0. Z: set if SR result is zero. Reset if not. VS: set if overflow is generated. Reset if not. N: MSB of SR result.
Notes:	–
Examples:	ESFTA A, B ESFTA SI,SA
# of Words:	1

ESFTD – Double Shift by Barrel Shifter

Format:	ESFTD asr,asa
Operation:	SG ← SG | (asr<<asa) This instruction shifts the value of 16-bit asr values by the amount of 6-bit asa. If the value of asa is positive, left shift operation is performed, and if the value of asa is negative right shift operation is performed. The 16-bit shifted result is ORed with previous SG register value ,and then stored into SG register.
Flags:	C: set if last shifted out bit is 1. Reset if not. Unchanged when shift amount is 0. Z: set if SG result is zero. Reset if not. VS: Reset. N: MSB of SG result.
Notes:	–
Examples:	ESFTD A, B ESFTD SI,SA
# of Words:	1

ESFTL – Linked Shift by Barrel Shifter

Format: ESFTL asr,asa

Operation: SR ← asr<<asa, SG ← SG | (asr<<asa)
This instruction shifts the value of 16-bit asr values by the amount of 6-bit asa. If the value of asa is positive, left shift operation is performed, and if the value of asa is negative right shift operation is performed. The 16-bit shifted result is stored into SR register and the 16-bit shifted out result is ORed with previous SG value and stored into SG register. The other bits of SR register are filled with zeros.

Flags: C: set if last shifted out bit is 1. Reset if not. Unchanged when shift amount is 0.
Z: set if SR result is zero. Reset if not.
VS: Reset.
N: MSB of SR result.

Notes: –

Examples:
```
ESFTL  A, B
ESFTL  SI,SA
```

# of Words: 1

ESLA[1)]/ESLAT* – Arithmetic 1-bit Left Shift Accumulator

Format: ESLA(T) Ai

Operation: Ai ← Ai <<1
This instruction shifts the value of one of 24-bit Accumulator(Ai) to 1-bit left , and stores the result back into the same accumulator.

Flags: C: set if shifted out bit is 1. Reset if not.
Z: set if result is zero. Reset if not.
Vi**: set if overflow is generated. Reset if not.
N: exclusive OR of Vi and MSB of result.

Notes: * ESLAT instruction can be executed only when the T flag is set.
Otherwise, No operation is performed.
** Vi denotes for VA or VB according to Ai

Examples:
```
ESLA   A
ESLAT  B
```

# of Words: 1

ESLA[2)] – Arithmetic 1-bit Left Shift Multiplier Accumulator

Format:	ESLA MA
Operation:	MA ← MA <<1 This instruction shifts the value of current bank 52-bit Multiplier Accumulator MA to 1-bit left , and stores the result back into the same Multiplier Accumulator.
Flags:	VMi*: set if result is overflowed to guard-bits. Reset if not. MV: set if guard-bit is overflowed. Unchanged if not.
Notes:	* VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	ESLA MA
# of Words:	1

ESLA8/ESLA8T* – Arithmetic 8-bit Left Shift Accumulator

Format:	ESLA8(T) Ai
Operation:	Ai ← Ai <<8 This instruction shifts the value of one of 24-bit Accumulator(Ai) to 8-bit left , and stores the result back into the same accumulator.
Flags:	C: set if last shifted out bit is 1. Reset if not. Z: set if result is zero. Reset if not. Vi**: set if overflow is generated. Reset if not. N: exclusive OR of Vi and MSB of result.
Notes:	* ESLA8T instruction can be executed only when the T flag is set. Otherwise, No operation is performed. ** Vi denotes for VA or VB according to Ai
Examples:	ESLA8 A ESLA8T B
# of Words:	1

ESLC/ESLCT* – Arithmetic 1-bit Left Shift Accumulator with Carry

Format:	ESLC(T) Ai
Operation:	Ai ← Ai <<1, Ai[0] ← C This instruction shifts the value of one of 24-bit Accumulator(Ai) to 1-bit left with carry : i.e. the carry bit is shifted into LSB of Ai register, and stores the result back into the same accumulator.
Flags:	C: set if shifted out bit is 1. Reset if not. Z: set if result is zero. Reset if not. Vi**: set if overflow is generated. Reset if not. N: exclusive OR of Vi and MSB of result.
Notes:	* ESLCT instruction can be executed only when the T flag is set. Otherwise, No operation is performed. ** Vi denotes for VA or VB according to Ai
Examples:	ESLC A ESLCT B
# of Words:	1

ESRA$^{1)}$/ESRAT* – Arithmetic 1-bit Right Shift Accumulator

Format:	ESRA(T) Ai
Operation:	Ai ← Ai >>1 This instruction shifts the value of one of 24-bit Accumulator (Ai) to 1-bit right, and stores the result back into the same accumulator.
Flags:	C: set if shifted out bit is 1. Reset if not. Z: set if result is zero. Reset if not. Vi**: set if overflow is generated. Reset if not. N: exclusive OR of Vi and MSB of result.
Notes:	* ESLRT instruction can be executed only when the T flag is set. Otherwise, No operation is performed. ** Vi denotes for VA or VB according to Ai
Examples:	ESRA A ESRAT B
# of Words:	1

ESRA[2)] – Arithmetic 1-bit Right Shift Multiplier Accumulator

Format:	ESRA MA
Operation:	MA $\leftarrow$ MA >>1 This instruction shifts the value of current bank 52-bit Multiplier Accumulator MA to 1-bit right, and stores the result back into the same Multiplier Accumulator.
Flags:	VMi*: set if result is overflowed to guard-bits. Reset if not. MV: set if guard-bit is overflowed. Unchanged if not.
Notes:	* VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	ESRA MA
# of Words:	1

ESRA8/ESRA8T* – Arithmetic 8-bit Right Shift Accumulator

Format:	ESRA8(T) Ai
Operation:	Ai ← Ai >>8 This instruction shifts the value of one of 24-bit Accumulator(Ai) to 8-bit right, and stores the result back into the same accumulator.
Flags:	C: set if last shifted out bit is 1. Reset if not. Z: set if result is zero. Reset if not. Vi**: set if overflow is generated. Reset if not. N: exclusive OR of Vi and MSB of result.
Notes:	* ESRA8T instruction can be executed only when the T flag is set. Otherwise, No operation is performed. ** Vi denotes for VA or VB according to Ai
Examples:	ESRA8 A ESRA8T B
# of Words:	1

ESRC/ESRCT* – Arithmetic 1-bit Right Shift Accumulator with Carry

Format: ESRC(T) Ai

Operation: Ai ← Ai >>1, Ai[23] ← C
This instruction shifts the value of one of 24-bit Accumulator(Ai) to 1-bit right with carry : i.e. the carry bit is shifted into MSB of Ai register, and stores the result back into the same accumulator.

Flags: C: set if shifted out bit is 1. Reset if not.
Z: set if result is zero. Reset if not.
Vi**: set if overflow is generated. Reset if not.
N: exclusive OR of Vi and MSB of result.

Notes: * ESRCT instruction can be executed only when the T flag is set.
Otherwise, No operation is performed.
** Vi denotes for VA or VB according to Ai

Examples:
```
ESRC   A
ESRCT  B
```

# of Words: 1

ESUB[1)] – Subtract Accumulator

Format:	ESUB Ai, <op> <op>: @rps rpdi.adr:5 / adr:15 #simm:16 Ai mg
Operation:	Ai ← Ai - <op> This instruction subtracts <op> value from the value of one of 24-bit Accumulator(Ai), and stores the result back into the same Accumulator.
Flags:	C: set if carry is generated. Reset if not. Z: set if result is zero. Reset if not. Vi*: set if overflow is generated. Reset if not. N: exclusive OR of Vi and MSB of result.
Notes:	* Vi denotes for VA or VB according to Ai
Examples:	ESUB A, @RP0+S0 ESUB B, RPD1.5h ESUB A, #0486h ESUB B, A ESUB A, RP0
# of Words:	1 2 when <op> is : adr:15 or #simm:16

ESUB$^{2)}$ – Subtract Accumulator with One Parallel Move

Format:	ESUB A, MA <dest>,<src> <dest>,<src>: mgx, @rps MA, @rps @rpd, mga
Operation:	A ← A - MA, <dest> ← <src> This instruction subtracts higher 24-bit part of Multiplier Accumulator MA from the value of 24-bit Accumulator A, and stores the result back into Accumulator A. This instruction also stores source operand from memory or register to destination register or memory.
Flags:	C: set if carry is generated by addition. Reset if not. Z: set if result is zero by addition. Reset if not. VA: set if overflow is generated by addition. Reset if not. N: exclusive OR of V and MSB of result by addition.
Notes:	None.
Examples:	ESUB A, MA X0,@RP0+S1 ESUB A, MA MA,@RP1+S0 ESUB A, MA @RP3+D1, A
# of Words:	1

ESUB[3)] – Subtract Multiplier Accumulator

Format:	ESUB MA, <op>
	<op>: P / PSH
Operation:	MA ← MA - <op>
	This instruction subtracts <op> value from the values of 52-bit Multiplier Accumulator MA, and stores the result back into Multiplier Accumulator MA.
Flags:	VMi*: set if result is overflowed to guard-bits. Reset if not.
	MV: set if guard-bit is overflowed. Unchanged if not.
Notes:	* VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	ESUB MA, P
	ESUB MA, PSH
# of Words:	1

ESUB[4)] – Subtract Multiplier Accumulator with One Parallel Move

Format:	ESUB MA, P <dest>,<src> <dest>,<src>: mgx, @rps A, @rps @rpd, mga
Operation:	MA $\leftarrow$ MA - P, <dest> $\leftarrow$ <src> This instruction subtracts the value of the Product register P from the value of 52-bit Multiplier Accumulator MA, and stores the result back into Multiplier Accumulator MA. This instruction also stores source operand from memory or register to destination register or memory.
Flags:	VMi*: set if result is overflowed to guard-bits. Reset if not. MV: set if guard-bit is overflowed. Unchanged if not.
Notes:	* VMi denotes for VM0 or VM1 according to the current MA bank.
Examples:	ESUB MA, P Y0, @RP1+S1 ESUB MA, P A, @RP2+S0 ESUB MA, P @RP0+D0, B
# of Words:	1

ETST – Test

Format: ETST cct

Operation:

```
if (cct is true)
    T ← 1
else
    T ← 0
```

This instruction sets the T flag of MSR0 register to 1 if condition specified in cct field is evaluated to truth. Else, resets the T flag. This instruction must be executed before executing the conditional instructions.

Flags: T: set/reset according to the condition

Notes: –

Examples:

```
ETST   GT
ETST   NEG
```

# of Words: 1

26 ELECTRICAL DATA

OVERVIEW

Table 26-1. Absolute Maximum Ratings

(T_A = 25°C)

Parameter	Symbol	Conditions	Rating	Unit
Supply voltage	V_{DD}	–	– 0.3 to + 4.5	V
Input voltage	V_I	–	– 0.3 to V_{DD} + 0.3	V
Output voltage	V_O	–	– 0.3 to V_{DD} + 0.3	V
Output current high	I_{OH}	One I/O pin active	– 15	mA
		All I/O pins active	– 100	
Output current low	I_{OL}	One I/O pin active	+ 20	mA
		Total pin current for ports 1, 2, 3	+ 150	
Operating temperature	T_A	–	– 40 to + 85	°C
Storage temperature	T_{STG}	–	– 65 to + 150	°C

Table 26-2. D.C. Electrical Characteristics

(T_A = – 40°C to + 85°C, V_{DD} = 3.0 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Operating Voltage	V_{DD}	f_{OSC} = 30 MHz	3.0	–	3.6	V
Input high voltage	V_{IH0}	RESET	0.85 V_{DD}	–	V_{DD}	V
	V_{IH1}	Test, P2, P3, P4, P8, P9	0.8 V_{DD}			
	V_{IH2}	All input pins except V_{IH0},V_{IH1} and V_{IH3}	0.7 V_{DD}			
	V_{IH3}	X_{IN}, XT_{IN}	V_{DD}– 0.5			
Input low voltage	V_{IL1}	Test, RESET, P2, P3, P4, P8, P9	0	–	0.2 V_{DD}	V
	V_{IL2}	All input pins except V_{IL1} and V_{IL3}			0.3 V_{DD}	

	V_{IL2}	X_{IN}, XT_{IN}			0.4	

Table 26-2. D.C. Electrical Characteristics (Continued)

(T_A = – 40°C to + 85°C, V_{DD} = 3.0 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Output high voltage	V_{OH1}	V_{DD} = 3.0 V to 3.6 V I_{OH} = – 1 mA	V_{DD}– 1.0	–	–	V
Output low voltage	V_{OL1}	V_{DD} = 3.0 V to 3.6 V I_{OL} = 5 mA All output pins	–	–	1.0	V
Input high leakage current	I_{LIH1}	V_{IN} = V_{DD} All input pins except I_{LIH2}	–	–	3	uA
	I_{LIH2}	V_{IN} = V_{DD} X_{IN}, XT_{IN}			20	
Input low leakage current	I_{LIL1}	V_{IN} = 0 V All input pins except I_{LIL2}	–	–	-3	
	I_{LIL2}	V_{IN} = 0 V X_{IN}, XT_{IN}, RESET			-20	
Output high leakage current	I_{LOH}	V_{OUT} = V_{DD} All I/O pins and Output pins	–	–	5	
Output low leakage current	I_{LOL}	V_{OUT} = 0 V All I/O pins and Output pins	–	–	-5	
Pull-up resistor	R_{L1}	V_{IN} = 0 V; V_{DD} = 3.3 V; T_A=25°C All input pins except R_{L2}	50	100	200	KΩ
	R_{L2}	V_{IN} = 0 V; V_{DD} = 3.3 V; T_A=25°C RESET only	100	250	400	
Supply current (1)	I_{DD1}	V_{DD} = 3.3 V 30 MHz crystal oscillator	–	35	70	mA
		V_{DD} = 3.3 V 32.768kHz crystal		210	400	uA
	I_{DD2}	Idle mode: V_{DD} = 3.3 V 30 MHz crystal oscillator	–	5	13	mA
		Idle mode: V_{DD} = 3.3 V 32.768kHz crystal oscillator		15	30	uA
	I_{DD3}	Stop mode V_{DD} = 3 V ± 10%	–	1	10	uA

NOTE: 1. Supply current does not include current drawn through internal pull-up resistors or external output current loads.

Table 26-3. A.C. Electrical Characteristics

(T_A = –40°C to + 85°C, V_{DD} = 3.0 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Interrupt input high, low width	t_{INTH}, t_{INTL}	P4.0-P4.1, P5.0-P5.5 at V_{DD} = 3.3 V	200	–	–	ns
RESET input low width	t_{RSL}	V_{DD} = 3.3 V	10	–	–	us

NOTE: User must keep a larger value than the Min value.

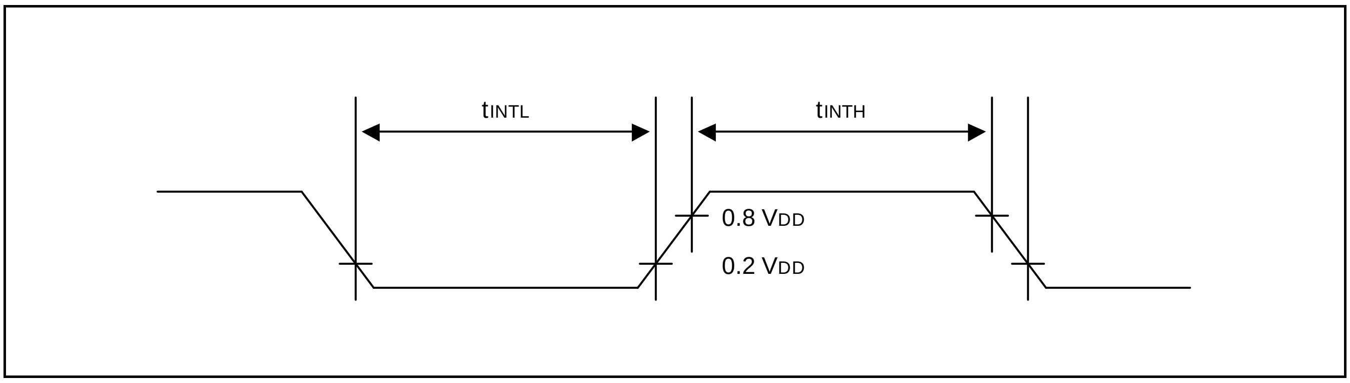


Figure 26-1. Input Timing for External Interrupts (Port 4, Port5)

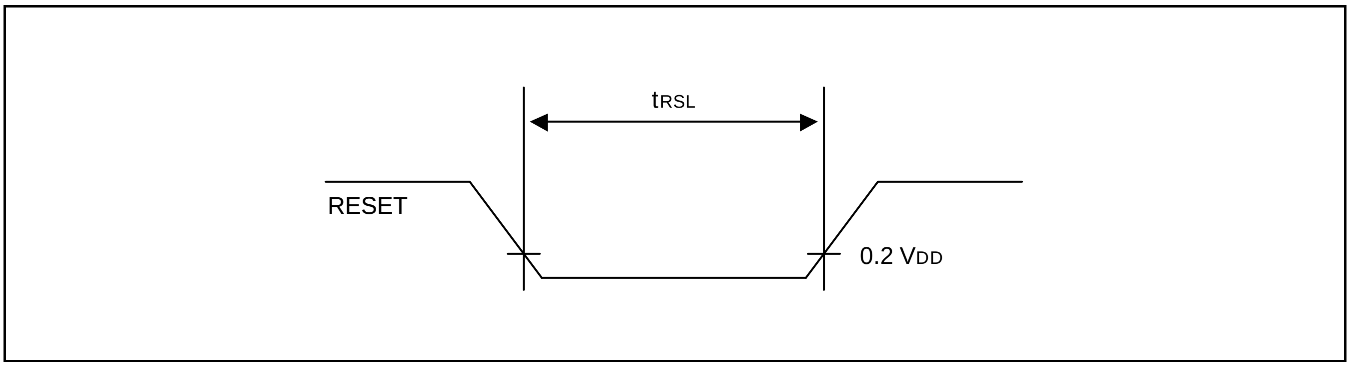


Figure 26-2. Input Timing for RESET

Table 26-4. Input/Output Capacitance

(T_A = – 40 °C to + 85 °C, V_{DD} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Input capacitance	C_{IN}	f = 1 MHz; unmeasured pins are returned to V_{SS}	–	–	10	pF
Output capacitance	C_{OUT}					

I/O capacitance	C_{IO}					

Table 26-5. A/D Converter Electrical Characteristics

(T_A = – 40 °C to + 85 °C, V_{DD} = 3.0 V to 3.6 V, V_{SS} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Resolution	–	–	–	8	–	bit
Total accuracy	–	V_{DD} = 3.3 V Conversion time = 5us	–	–	± 2	
Integral Linearity Error	ILE	AV_{REF} = 3.3 V		–	± 1	
Differential Linearity Error	DLE	AV_{SS} = 0 V		–	± 1	LSB
Offset Error of Top	EOT	–		± 1	± 2	
Offset Error of Bottom	EOB	–		± 0.5	± 2	
Conversion time [(1)]	t_{CON}	–	20	–	–	us
Analog input voltage	V_{IAN}	–	AV_{SS}	–	AV_{REF}	V
Analog input impedance	R_{AN}	–	2	1000	–	Mohm
Analog reference voltage	AV_{REF}	–	V_{DD}	–	V_{DD}	V
Analog ground	AV_{SS}	–	V_{SS}	–	V_{SS}	V
Analog input current	I_{ADIN}	AV_{REF} = V_{DD} = 3.3 V	–	–	10	uA
Analog block current [(2)]	I_{ADC}	AV_{REF} = V_{DD} = 3.3 V		1	3	mA
		AV_{REF} = V_{DD} = 3 V		0.5	1.5	mA

NOTES:
1. 'Conversion time' is the time required from the moment a conversion operation starts until it ends.
2. I_{ADC} is an operating current during A/D conversion.

Table 26-6. I^2S Master Transmitter with Data Rate of 2.5 MHz (10%) (Unit: ns)

Parameter	Min	Typ	Max	Condition
Clock period T	360	400	440	Ttr = 360
Clock HIGH tHC	160	–	–	min > 0.35T = 140 (at typical data rate)
Clock LOW tLC	160	–	–	min > 0.35T = 140 (at typical data rate)
Delay tdtr	–	–	300	max < 0.80T = 320 (at typical data rate)
Hold time thtr	100	–	–	min > 0
Clock rise-time tRC	–	–	60	max > 0.15T = 54 (atrelevent in slave mode)

Table 26-7. I^{2}S Slave Receiver with Data Rate of 2.5 MHz (10%) (Unit: ns)

Parameter	Min	Typ	Max	Condition
Clock period T	360	400	440	Ttr = 360
Clock HIGH tHC	110	–	–	min < 0.35T = 126
Clock LOW tLC	110	–	–	min < 0.35T = 126
Set-up time tsr	60	–	–	min < 0.20T = 72
Hold time thtr	0	–	–	min < 0

Table 26-8. Flash Memory D.C. Electrical Characteristics

(T_A = – 40°C to + 85°C, V_{DD} = 3.0 V to 3.6 V, V_{PP} = 12.5V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Logic power supply	V_{DD}	–	3.0	3.3	3.6	V
power supply for programming Flash cell	V_{PP}	–	12.25	12.5	12.75	V
Flash memory operating current (FI_{DD})	FI_{DD1}	V_{DD} = 3.0 – 3.6V during reading	–	20	40	mA
	FI_{DD2}	V_{DD} = 3.0 – 3.6V during programming	–	10	20	mA
	FI_{DD3}	V_{DD} = 3.0 – 3.6V during erasing	–	10	20	mA

Table 26-9. Flash Memory A.C. Electrical Characteristics

(T_A = – 40 °C to + 85 °C, V_{DD} = 3.0 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Programming time (1)	Ftp	V_{DD} = 3.3 – 3.6V	20	30	300	uS
Chip Erasing Time (2)	Ftp1		–	–	10	mS
Sector Erasing time (3)	Ftp2		–	2		mS
Data access time	Ft_{RS}		–	50	–	nS
Number of writing/erasing	Fnwe	–	–	50,000		Times

NOTES:

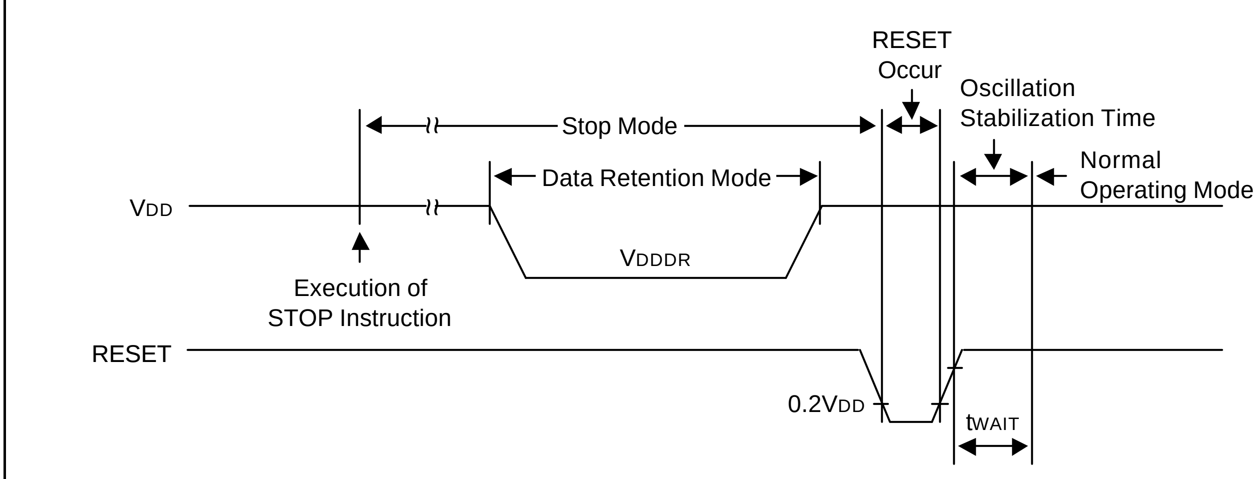
1. The programming time is the time during which one word (32-bit) is programmed.
2. The chip erasing time is the time during which all 256K-byte block is erased.
3. The sector erasing time is the time during which all 512-byte block is erased.

Table 26-10. Data Retention Supply Voltage in Stop Mode

(T_A = – 40°C to + 85°C, V_{DD} = 3.0 V to 3.6V)

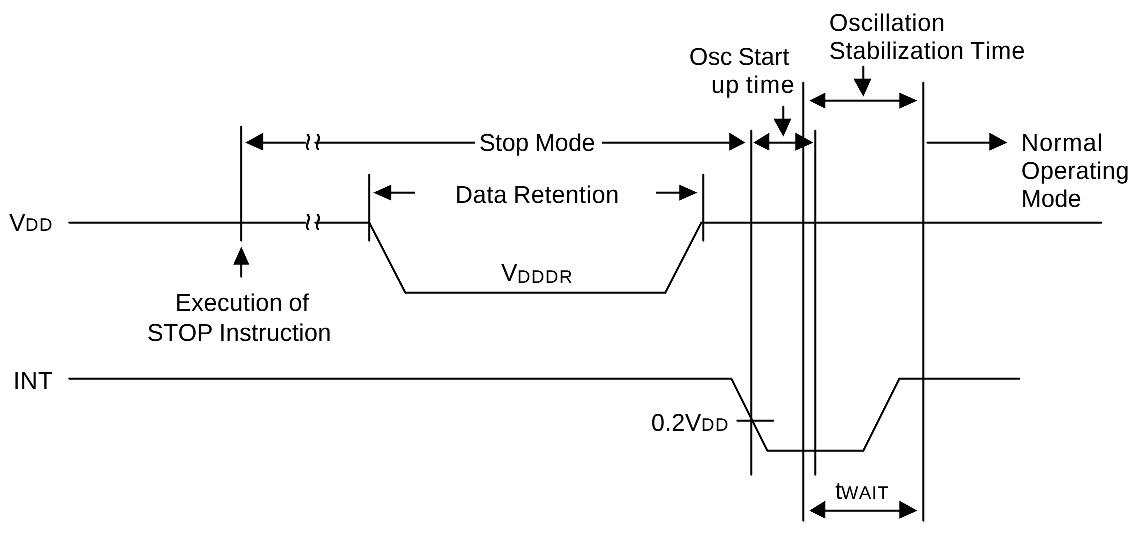
Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	V_{DDDR}	Normal operation	2	–	3.6	V
Data retention supply current	I_{DDDR}	V_{DDDR} = 2V	–	–	1	μA

NOTE: Supply current does not include a current which drawn through internal pull-up resistors or external output current loads.



NOTE: t_{WAIT} is the same as 2048 x 16 x 1/fxx

Figure 26-3. Stop Mode Release Timing When Initiated by a RESET



NOTE: t_{WAIT} is the same as 2048 x 16 x 1/fxx. The value of 2048 which is selected for the clock source of the basic timer counter can be changed. Then the value of t_{WAIT} will be changed.

Figure 26-4. Stop Mode Release Timing When Initiated by Interrupts

Table 26-11. Synchronous SIO Electrical Characteristics

(T_A = – 40°C to + 85°C V_{DD} = 3.0 V to 3.6 V, V_{SS} = 0 V, fxx = 30 MHz oscillator)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
SCK Cycle time	t_{CYC}	–	200	–	–	ns
Serial Clock High Width	t_{SCKH}	–	60	–	–	
Serial Clock Low Width	t_{SCKL}	–	60	–	–	
Serial Output data delay time	t_{OD}	–	–	–	50	
Serial Input data setup time	t_{ID}	–	40	–	–	
Serial Input data Hold time	t_{IH}	–	100	–	–	

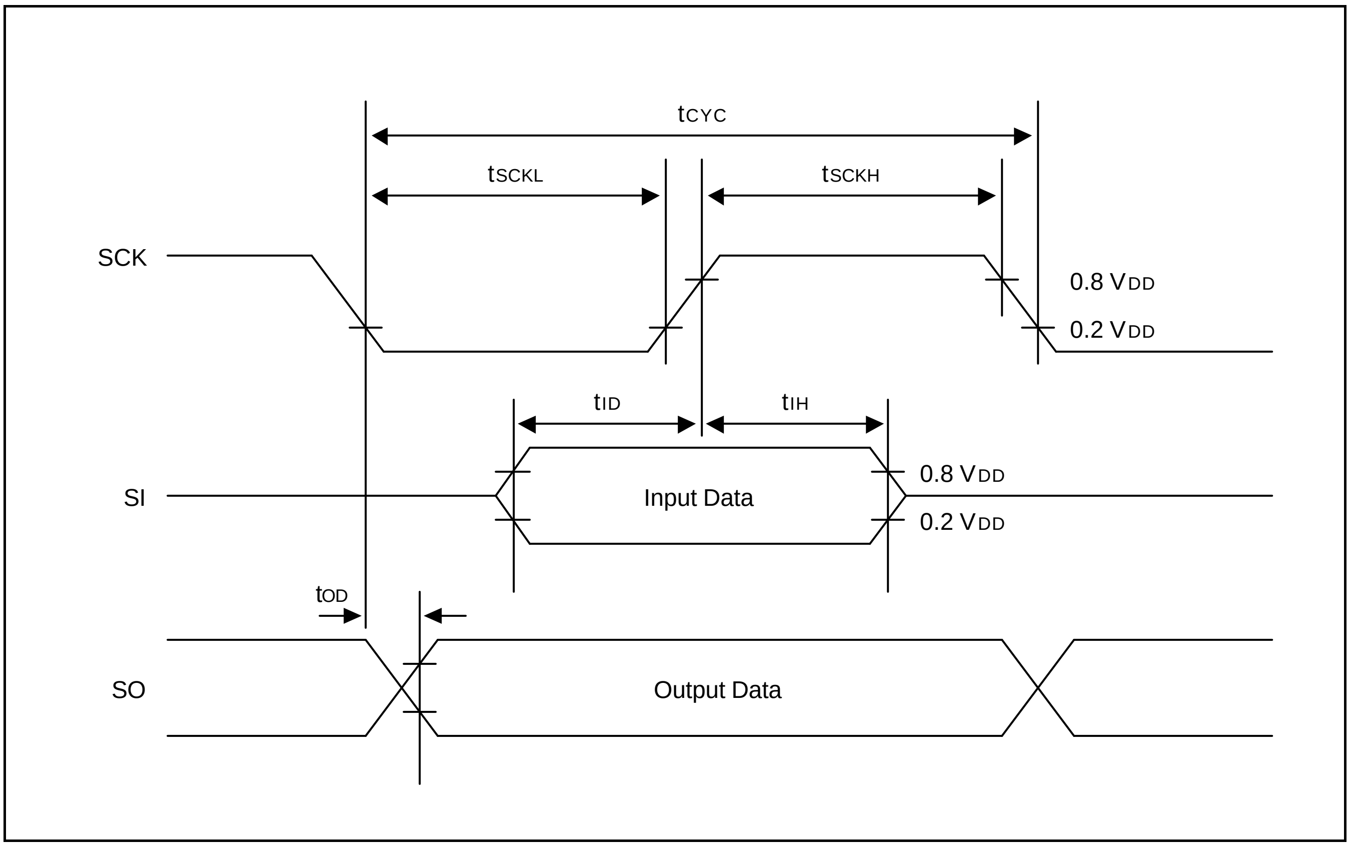
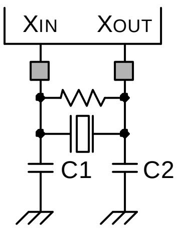
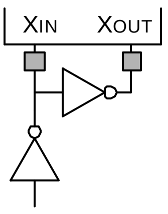


Figure 26-5. Serial Data Transfer Timing

Table 26-12. Main Oscillator Frequency (fosc1)

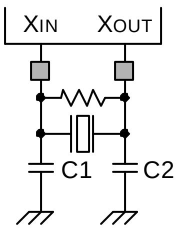
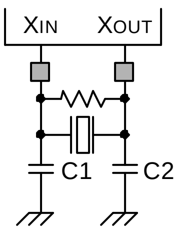
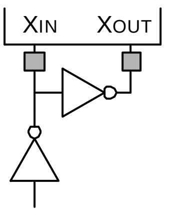
(T_A = – 40°C to + 85°C V_{DD} = 3.0 V to 3.6 V)

Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
Crystal		Oscillation frequency	32	32.768	35	kHz
		Stabilization time	–	1	3	s
External clock		X_{IN} input frequency	32	–	35	kHz
		X_{IN} input high and low level width ($t_{XH,}$ t_{XL})	14	–	16	us

NOTE: Oscillation stabilization time (t_{ST1}) is the time that the amplitude of a oscillator input rich to 0.8 $V_{DD,}$ after a power-on occurs, or when Stop mode is ended by a RESET or a interrupt signal.

Table 26-13. Sub Oscillator Frequency (fosc2)

(T_A = – 40°C to + 85°C V_{DD} = 3.0 V to 3.6 V)

Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
Crystal		Crystal oscillation frequency	–	–	35	MHz
		Stabilization time	–	–	10	ms
Ceramic		Ceramic oscillation frequency	–	–	35	MHz
		Stabilization time	–	–	4	ms
External clock		X_{IN} input frequency	–	–	35	MHz
		X_{IN} input high and low level width (t_{XH}, t_{XL})	14	–	–	ns

NOTE: Oscillation stabilization time (t_{ST1}) is the time that the amplitude of a oscillator input rich to 0.8 V_{DD}, after a power-on occurs, or when Stop mode is ended by a RESET or a interrupt signal.

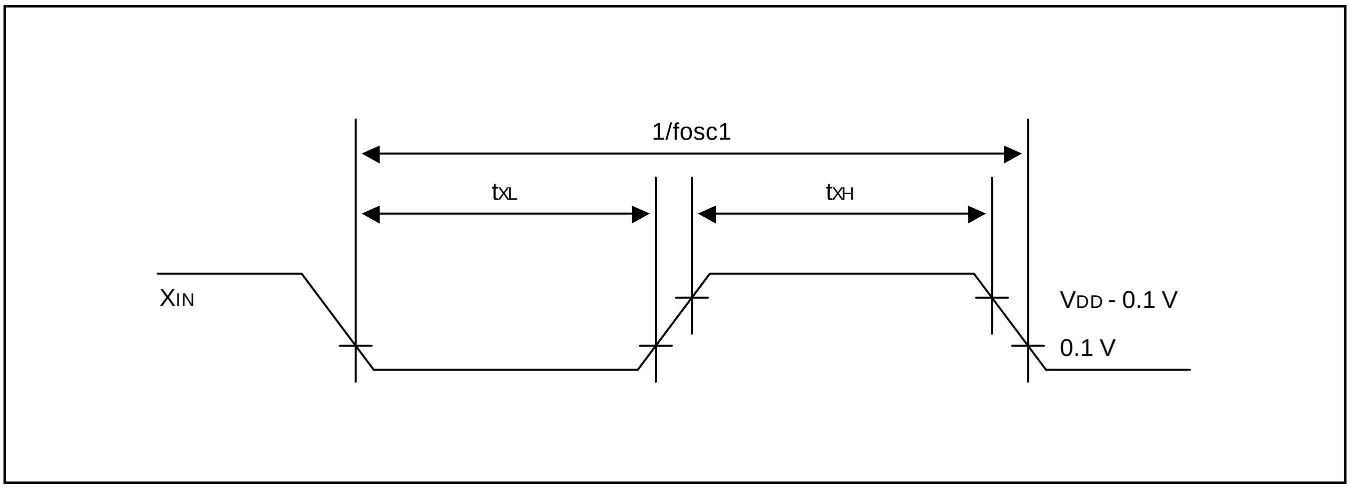


Figure 26-6. Clock Timing Measurement at X_{IN}

NOTES

27 MECHANICAL DATA

OVERVIEW

The S3FB42F is available in a 100-QFP-1420C package and a 100-TQFP-1414 package.

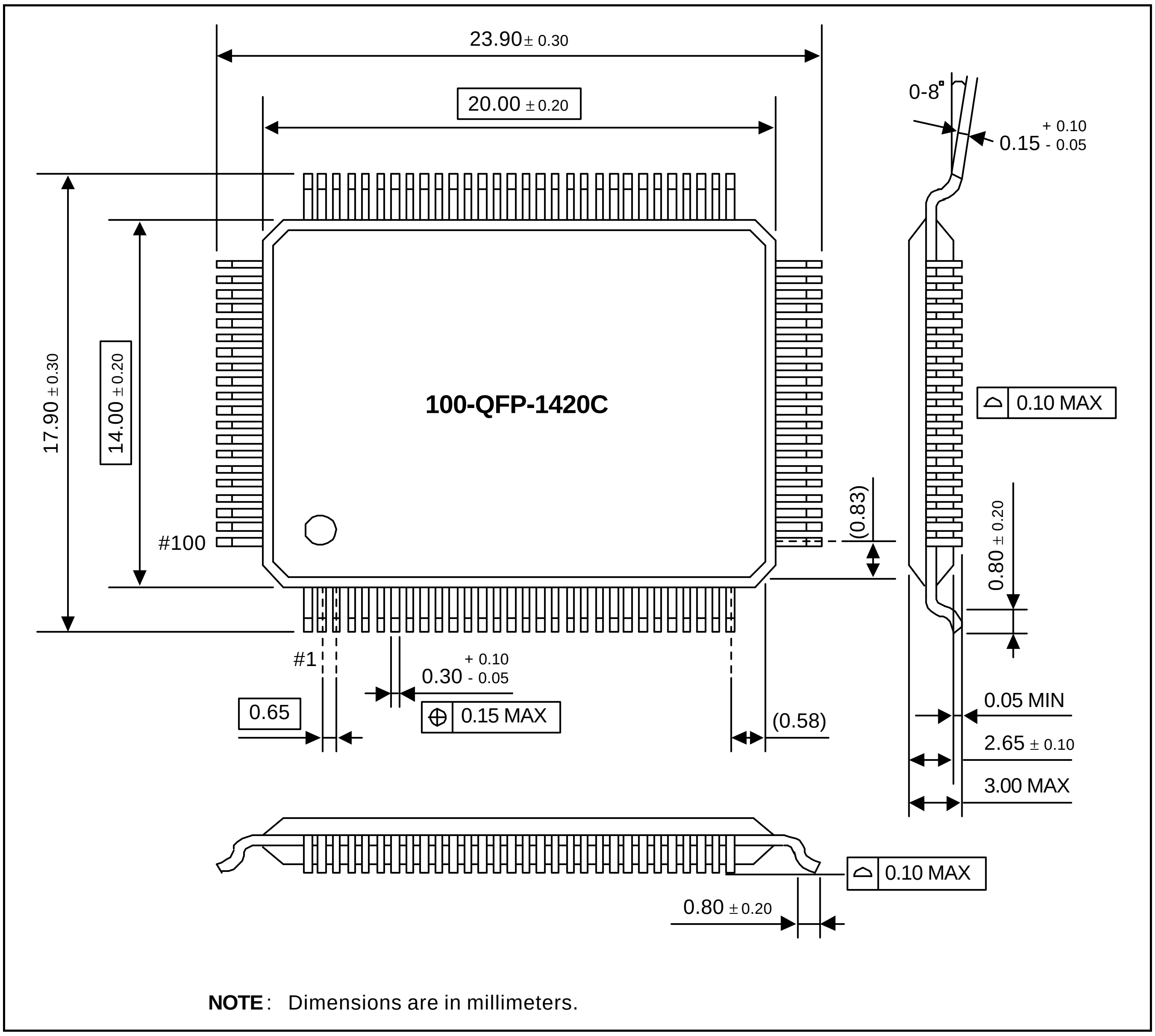
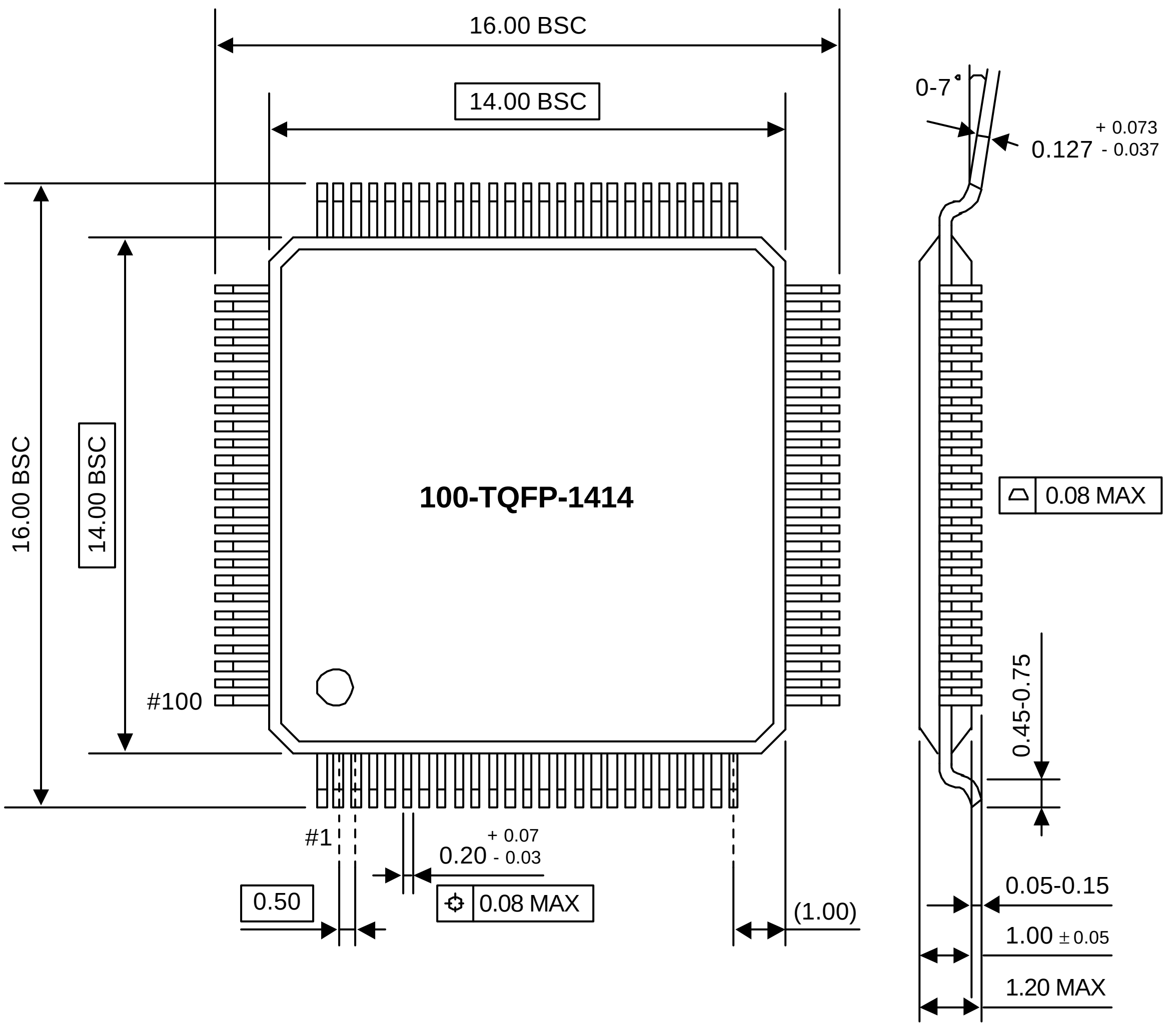


Figure 27-1. 100-QFP-1420C Package Dimensions



NOTE: Dimensions are in millimeters.

Figure 27-2. 100-TQFP-1414 Package Dimensions