

# **S3CC11B/FC11B**

**CalmRISC 16-Bit CMOS  
MICROCONTROLLER  
USER'S MANUAL**

**Revision 0**



**ELECTRONICS**

# Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. Samsung assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained herein.

Samsung reserves the right to make changes in its products or product specifications with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

This publication does not convey to a purchaser of semiconductor devices described herein any license under the patent rights of Samsung or others.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Samsung assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

**S3CC11B/FC11B 16-Bit CMOS Microcontroller  
User's Manual, Revision 0  
Publication Number: 20-S3-CC11B/FC11B-102004**

© 2004 Samsung Electronics

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electric or mechanical, by photocopying, recording, or otherwise, without the prior written consent of Samsung Electronics.

*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-Ri, Giheung- Eup  
Yongin-City, Gyeonggi-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/](http://www.samsungsemi.com/)

Printed in the Republic of Korea

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, for other applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use a Samsung product for any such unintended or unauthorized application, the Buyer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that Samsung was negligent regarding the design or manufacture of said product.

# Preface

The S3CC11B/FC11B *Microcontroller User's Manual* is designed for application designers and programmers who are using the S3CC11B/FC11B 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 seven chapters:

Chapter 1	Product Overview	Chapter 4	Exceptions
Chapter 2	Address Spaces	Chapter 5	Memory Map
Chapter 3	Calm16Core	Chapter 6	Instruction Set

Chapter 1, "Product Overview," is a high-level introduction to S3CC11B/FC11B 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, "Calm16Core," describes the special registers.

Chapter 4, "Exceptions," describes the internal register file.

Chapter 5, "Memory Map," describes the S3CC11B/FC11B memory map structure in detail.

Chapter 6, "Instruction Set," describes the S3CC11B/FC11B instruction set structure in detail.

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 S3CK-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 and 6. Later, you can reference the information in Part I as necessary.

Part II "hardware Descriptions," has detailed information about specific hardware components of the S3CC11B/FC11B microcontroller. Also included in Part II are electrical, mechanical. It has 20 chapters:

Chapter 7	PLL (Phase Locked Loop)	Chapter 18	LCD Controller / Driver
Chapter 8	RESET and Power-Down	Chapter 19	Battery Level Detector
Chapter 9	I/O Ports	Chapter 20	8/16-Bit Serial Interface for External Codec
Chapter 10	Basic Timer	Chapter 21	CaimMAC1616
Chapter 11	Watch Timer	Chapter 22	Program Memory Access Speed
Chapter 12	8-bit Timer 0	Chapter 23	Electrical Data
Chapter 13	16-Bit Timer 1 (8-Bit Timer A & B)	Chapter 24	Mechanical Data
Chapter 14	Serial I/O Interface	Chapter 25	S3FC11B Flash MCU
Chapter 15	SSFDC (Solid State Floppy Disk Card)	Chapter 26	Development Tools
Chapter 16	10-Bit Analog-to-Digital Converter		
Chapter 17	CODEC		

One order form is included at the back of this manual to facilitate customer order for S3CC11B/FC11B 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

### Chapter 1 Product Overview

Introduction.....	1-1
Features.....	1-1
Block Diagram .....	1-3
Pin Assignment.....	1-4
Pin Circuit Diagrams .....	1-9

### Chapter 2 Address Spaces

Overview.....	2-1
Program Memory .....	2-2
Data Memory .....	2-4

### Chapter 3 Calm16Core

Introduction.....	3-1
Features.....	3-1
Registers .....	3-2
General Registers & Extension Registers .....	3-2
Special Registers .....	3-3
Pipeline Structure .....	3-4
Interrupts .....	3-5

### Chapter 4 Exceptions

Overview.....	4-1
Hardware Reset.....	4-1
FIQ Exception.....	4-2
IRQ Exception.....	4-2
TRQ Exception.....	4-2
SWI Exception .....	4-2
Break Exception.....	4-2
Interrupt Sources (IRQ) .....	4-3
Interrupt Structure.....	4-4
Interrupt Control Register.....	4-5
Interrupt Masking Register.....	4-5
Interrupt Priority Register.....	4-5
Interrupt Priority Registers (IPRH: 3F0008H, IPRL: 3F0009H) .....	4-8
Interrupt Id Register .....	4-9

## Table of Contents (Continued)

### Chapter 5 Memory Map

Overview.....	5-1
---------------	-----

### Chapter 6 Instruction Set

ALU Instructions.....	6-1
ALUOP Register, Immediate.....	6-2
ALUOP Register, Register.....	6-3
Load Instructions.....	6-4
LD Register, Register.....	6-4
LD Register, Data Memory / LD Data Memory, Register.....	6-5
LD Register, Program Memory.....	6-7
LD Register, # Immediate.....	6-7
Branch Instructions.....	6-8
Bit Operation.....	6-10
Miscellaneous Instructions .....	6-11
CalmRISC16 Instruction Set Map.....	6-12
Quick Reference.....	6-17

# Table of Contents (Continued)

## Part II — Hardware Descriptions

### Chapter 7 PLL (Phase Locked Loop)

Overview.....	7-1
---------------	-----

### Chapter 8 RESET and Power-Down

Overview.....	8-1
---------------	-----

### Chapter 9 I/O Ports

Port Data Registers.....	9-1
--------------------------	-----

### Chapter 10 Basic Timer

Overview.....	10-1
Basic Timer & Watchdog Timer Block Diagram .....	10-4

### Chapter 11 Watch Timer

Overview.....	11-1
Watch Timer Block Diagram.....	11-3

### Chapter 12 8-Bit Timer 0

Overview.....	12-1
Function Description.....	12-2
Timer 0 Control Register (T0CON) .....	12-3
Block Diagram .....	12-4

### Chapter 13 16-Bit Timer 1 (8-Bit Timer A & B)

Overview.....	13-1
Interval Timer Function.....	13-1
Block Diagram .....	13-4

## Table of Contents (Continued)

### Chapter 14 Serial I/O Interface

Overview.....	14-1
Programming Procedure.....	14-1
SIO Pre-Scaler Register (SIOPS).....	14-3
Block Diagram .....	14-3
Serial I/O Timing Diagrams .....	14-4

### Chapter 15 SSFDC (Solid State Floppy Disk Card)

Overview.....	15-1
SSFDC Register Description .....	15-3
SmartMedia Control Register (SMCON) .....	15-3
SmartMedia ECC Count Register (ECCNT) .....	15-4
SmartMedia ECC Data Register (ECCDATA) .....	15-4
SmartMedia ECC Result Data Register (ECCRST) .....	15-4

### Chapter 16 10-Bit Analog-To-Digital Converter

Overview.....	16-1
Function Description .....	16-1
Conversion Timing .....	16-2
A/D Converter Control Register (ADCON10) .....	16-2
Internal Reference Voltage Levels.....	16-3
Block Diagram .....	16-3

### Chapter 17 Codec

Overview.....	17-1
Features.....	17-1
A/D Converter Control Register (ADCON).....	17-2

### Chapter 18 LCD Controller/Driver

Overview.....	18-1
LCD Circuit Diagram .....	18-2
LCD Display Registers.....	18-3
LCD Control Register (LCON) .....	18-3
LCD Voltage Dividing Resistors.....	18-6

## Table of Contents (Continued)

### Chapter 19 Battery Level Detector

Overview.....	19-1
Battery Level Detector Control Register (BLDCON) .....	19-2

### Chapter 20 8/16-Bit Serial Interface for External Codec

Overview.....	20-1
Programming Procedure.....	20-1
CSIO Control Register (CSIOCON).....	20-2

### Chapter 21 CaimMAC1616

Introduction.....	21-1
Architecture Features .....	21-1
Technology Features .....	21-1
Block Diagram .....	21-2
Programming Model.....	21-3
Multiplier and Accumulator Unit .....	21-4
Arithmetic Unit .....	21-7
Status Register 1 (MSR1) .....	21-11
Ram Pointer Unit .....	21-13
Address Modification .....	21-15
Data Memory Spaces and Organization.....	21-19
Arithmetic Unit .....	21-21
Overflow Protection in Accumulators .....	21-22
External Condition Generation Unit.....	21-24
Status Register 0 (MSR0) .....	21-25
Status Register 2 (MSR2) .....	21-27
Barrel Shifter and Exponent Unit .....	21-29
Barrel Shifter.....	21-29
Shifting Operations .....	21-30
Exponent Block.....	21-33
Instruction Set Map and Summary .....	21-34
Addressing Modes.....	21-34
Instruction Coding.....	21-39
Quick Reference.....	21-55
Instruction Set.....	21-60
Glossary .....	21-60



## Table of Contents (Continued)

### Chapter 22 Program Memory Access Speed

Overview..... 22-1

### Chapter 23 Electrical Data

Overview..... 23-1

### Chapter 24 Mechanical Data

Overview..... 24-1

### Chapter 25 S3FC11B Flash MCU

Overview..... 25-1

### Chapter 26 Development Tools

Overview..... 26-1

    CalmSHINE: IDE (Integrated Development Environment)..... 26-1

    In-Circuit Emulator..... 26-1

    CalmRISC16 C-Compiler: CalmCC16..... 26-1

    CalmRISC16 Relocatable Assembler: Calm8ASM..... 26-1

    CalmRISC16 Linker: Calm8LINK..... 26-1

Emulation Probe Board Configuration ..... 26-2

    Use Clock Setting for External Clock Mode..... 26-3

    Sub Clock Setting ..... 26-3

    The Lowpass Filter for PLL ..... 26-3

    Power Selection..... 26-4

    Clock Selection..... 26-4

    JP1, JP2 Pin Assignment..... 26-5

    JP11 Pin Assignment..... 26-5

# List of Figures

Figure Number	Title	Page Number
1-1	S3CC11B/FC11B Top Block Diagram .....	1-3
1-2	S3CC11B/FC11B Pin Assignments (100-QFP-1420C) .....	1-4
1-3	S3CC11B/FC11B Pin Assignments (100-TQFP-1414) .....	1-5
1-4	Pin Circuit Type 1 .....	1-9
1-5	Pin Circuit Type 2 (nRESET) .....	1-9
1-6	Pin Circuit Type 3 .....	1-9
1-7	Pin Circuit Type 4 (P0.4-P0.7, P1, P2, P3.4-P3.7, P4.0).....	1-10
1-8	Pin Circuit Type 5 (P0.0-P0.3).....	1-10
1-9	Pin Circuit Type 6 (P3.0-P3.3).....	1-11
1-10	Pin Circuit Type 7 .....	1-11
1-11	Pin Circuit Type 7 (P6, P7, P8, P9) .....	1-12
1-12	Pin Circuit Type 9 (P4.1-P4.3, P5) .....	1-12
1-13	Pin Circuit Type 10 (P4.4-P4.7) .....	1-13
2-1	Program Memory Configuration .....	2-2
2-2	Data Memory Configuration.....	2-4
3-1	Register Structure in CalmRISC16 .....	3-2
4-1	Interrupt Sources (IRQ).....	4-3
4-2	Interrupt Structure .....	4-4
4-3	Interrupt Priority Register (IPR) .....	4-8
5-1	Memory Mapped IO Registers.....	5-1
7-1	Phase-Locked Loop Circuit Diagram .....	7-1
7-2	System Clock Circuit Diagram .....	7-4
7-3	External Loop Filter for PLL.....	7-5
9-1	Port Data Register Structure .....	9-1
10-1	Basic Timer & Watchdog Timer Block Diagram.....	10-4
11-1	Watch Timer Block Diagram .....	11-3
12-1	Timer 0 Functional Block Diagram .....	12-4
13-1	Timer 1 Block Diagram .....	13-4

## List of Figures (Continued)

Figure Number	Title	Page Number
14-1	SIO Pre-scaler Register (SIOPS).....	14-3
14-2	SIO Functional Block Diagram .....	14-3
14-3	Serial I/O Timing in Transmit/Receive Mode (Tx at falling, SIOCON.4 = 0) .....	14-4
14-4	Serial I/O Timing in Transmit/Receive Mode (Tx at rising, SIOCON.4 = 1) .....	14-4
15-1	Simple System Configuration .....	15-2
15-2	ECC Processor Block Diagram .....	15-5
16-1	A/D Converter Control Register (ADCON10).....	16-2
16-2	A/D Converter Data Register (ADDATAH10/ADDATAL10) .....	16-3
16-3	A/D Converter Functional Block Diagram .....	16-3
16-4	Recommended A/D Converter Circuit for Highest Absolute Accuracy .....	16-4
17-1	CODEC Block Diagram .....	17-4
17-2	Single-Ended Input Application.....	17-5
18-1	LCD Function Diagram .....	18-1
18-2	LCD Circuit Diagram .....	18-2
18-3	LCD Display Register Organization.....	18-3
18-4	LCD Voltage Dividing Registers Connection.....	18-6
18-5	LCD Signal Waveforms (1/3 Duty, 1/3 Bias).....	18-7
18-6	LCD Signal Waveforms (1/4 Duty, 1/3 Bias).....	18-8
18-7	LCD Signal Waveforms (1/8 Duty, 1/4 Bias).....	18-9
18-9	LCD Signal Waveforms (1/8 Duty, 1/5 Bias).....	18-11
19-1	Block Diagram for Battery Level Detect .....	19-1
19-2	Battery Level Detector Circuit and Control Register .....	19-2
20-1	SIO Block Diagram for External Codec.....	20-4
20-2	8-Bit SIO Timing Diagram for External Codec .....	20-5
20-3	16-Bit SIO Timing Diagram for External Codec .....	20-6

## List of Figures (Continued)

Figure Number	Number	Title	Page
21-1		CalmMAC1616 Block Diagram .....	21-2
21-2		Multiplier and Accumulator Unit Block Diagram.....	21-4
21-3		MAU Registers Configuration .....	21-6
21-4		Integer Division Example.....	21-9
21-5		Fractional Division Example .....	21-10
21-6		MSR1 Register Configuration .....	21-11
21-7		RAM Pointer Unit Block Diagram.....	21-14
21-8		Pointer Register and Index Register Configuration.....	21-15
21-9		Modulo Control Register Configuration.....	21-17
21-10		CalmMAC16 Data Memory Space Map.....	21-19
21-11		CalmMAC16 Data Memory Allocation .....	21-20
21-12		Arithmetic Unit Block Diagram .....	21-22
21-13		Accumulator Register Configuration.....	21-23
21-14		MSR0 Register Configuration .....	21-25
21-15		MSR2 Register Configuration .....	21-27
21-16		Barrel Shifter and Exponent Unit Block Diagram .....	21-29
21-17		Various Barrel Shifter Instruction Operation .....	21-31
21-18		Indirect Addressing Example I (Single Read Operation) .....	21-34
21-19		Indirect Addressing Example II (Dual Read Operation) .....	21-35
21-20		Indirect Addressing Example III (Write Operation).....	21-36
21-21		Short Direct Addressing Example.....	21-36
21-22		Long Direct Addressing Example.....	21-37
21-23		Short Direct Associated Addressing Example.....	21-38
23-1		Operating Voltage Range.....	23-4
23-2		Input Timing for External Interrupts (Ports 0, Ports 4).....	23-5
23-3		Input Timing for RESET .....	23-5
23-4		Stop Mode Release Timing When Initiated by a nRESET.....	23-6
23-5		Stop Mode(main) Release Timing Initiated by Interrupts.....	23-7
23-6		Stop Mode(sub) Release Timing Initiated by Interrupts.....	23-7
23-7		Clock Timing Measurement at $X_{IN}$ .....	23-8
23-8		Clock Timing Measurement at $XT_{IN}$ .....	23-9
24-1		100-QFP-1420C Package Dimensions .....	24-2
24-2		100-TQFP-1414 Package Dimensions.....	24-3
25-1		S3FC11B Pin Assignments (100-QFP-1420C) .....	25-2
25-2		S3FC11B Pin Assignments (100-TQFP-1414).....	25-3
26-1		Emulation Probe Board Configuration .....	26-2

# List of Tables

Table Number	Title	Page Number
1-1	S3CC11B/FC11B Pin Description.....	1-6
4-1	Exceptions.....	4-1
5-1	Registers .....	5-2
6-1	CalmRISC16 Instruction Set Map .....	6-12
6-2	Quick Reference.....	6-17
9-1	Port Data Register Summary .....	9-1
15-1	Control Register Description.....	15-3
19-1	BLDCON Value and Detection Level .....	19-2
21-1	Exponent Evaluation and Normalization Example.....	21-33
23-1	Absolute Maximum Ratings .....	23-1
23-2	D.C. Electrical Characteristics .....	23-1
23-3	A.C. Electrical Characteristics .....	23-5
23-4	Data Retention Supply Voltage in Stop Mode .....	23-6
23-5	Main Oscillator Characteristics.....	23-8
23-6	Sub Oscillator Frequency .....	23-9
23-7	BLD Electrical Characteristics.....	23-10
23-8	PLL Electrical Characteristics.....	23-10
23-9	10-Bit A/D Converter Electrical Characteristics .....	23-10
23-10	ADC/DAC Electrical Characteristics .....	23-11
25-1	Descriptions of Pins Used to Read/Write the FLASH ROM.....	25-4

# List of Instruction Descriptions

Instruction Mnemonic	Full Instruction Name	Page Number
ADC (1)	Add with Carry Register .....	6-21
ADC (2)	Add with Carry Immediate .....	6-22
ADD (1)	Add Register .....	6-23
ADD (2)	Add Small Immediate .....	6-24
ADD (3)	Add Immediate .....	6-25
ADD (4)	Add Extended Register .....	6-26
ADD (5)	Add Immediate to Extended Register .....	6-27
ADD (6)	Add 5-bit Immediate to Extended Register .....	6-28
AND (1)	AND Register .....	6-29
AND (2)	AND Small Immediate .....	6-30
AND (3)	AND Large Immediate .....	6-31
BITop	BIT Operation .....	6-32
BNZD	Branch Not Zero with Autodecrement .....	6-33
BR	Conditional Branch .....	6-34
BRA EC	Branch on External Condition .....	6-35
BREAK	BREAK .....	6-36
BSRD	Branch Subroutine with Delay Slot .....	6-37
CLD	Coprocessor Load .....	6-38
CLRSR	Clear SR .....	6-39
CMP (1)	Compare Register .....	6-40
CMP (2)	Compare Immediate .....	6-41
CMP (3)	Compare Short Immediate .....	6-42
CMPEQ (1)	Compare Equal Extended Register .....	6-43
CMPEQ (2)	Compare Equal Small Immediate .....	6-44
CMPEQ (3)	Compare Equal Large Immediate .....	6-45
COM	Complement .....	6-46
COP	Coprocessor .....	6-47
DECC	Decrement with Carry .....	6-48
DT	Decrement and Test .....	6-49
EXT	Sign-Extend .....	6-50
INCC	Increment with Carry .....	6-51
JMP (1)	Jump Register .....	6-52
JMP (2)	Jump Immediate .....	6-53
LD (1)	Load Register .....	6-54
LD (2)	Load Register .....	6-55
LD (3)	Load Short Immediate .....	6-56
LD (4)	Load Immediate .....	6-57
LD (5)	Load Large Immediate .....	6-58
LD RExt	Load Register Extension .....	6-59
LDB (1)	Load Byte Register Disp .....	6-60
LDB (2)	Load Byte Register Large Disp .....	6-61
LDB (3)	Load Byte Register Indexed .....	6-62
LDB (4)	Load Byte to R0 Register Disp .....	6-63
LDC	Load Code .....	6-64
LD PC	Load Program Counter .....	6-65
LD SvR (1)	Load from Saved Register .....	6-66

## List of Instruction Descriptions (Continued)

Instruction Mnemonic	Full Instruction Name	Page Number
LD SvR (2)	Load to Saved Register.....	6-67
LD SR	Load Status Register.....	6-68
LDW (1)	Load Word Stack Disp. ....	6-69
LDW (2)	Load Word Register Small Disp.....	6-70
LDW (3)	Load Word Register Disp. ....	6-71
LDW (4)	Load Word Register Indexed .....	6-72
LDW (5)	Load Word Register Small Disp.....	6-73
LDW (6)	Load Word Register Disp. ....	6-74
LDW (7)	Load Word Register Indexed .....	6-75
MUL	Multiplication.....	6-76
NOP	No Operation.....	6-77
OR (1)	OR Register .....	6-78
OR (2)	OR Small Immediate .....	6-79
OR (3)	OR Large Immediate .....	6-80
POP (1)	Load Register from Stack .....	6-81
POP (2)	Load Register from Stack .....	6-82
PUSH (1)	Load Register to Stack .....	6-83
PUSH (2)	Load Register to Stack .....	6-84
RETD	Ret. from Subroutine with Delay Slot.....	6-85
RET_FIQ	Return from Fast Interrupt .....	6-86
RET_IRQ	Return from Interrupt.....	6-87
RET_SWI	Return from Software Interrupt.....	6-88
RL	Rotate Left .....	6-89
RR	Rotate Right .....	6-90
RRC	Rotate Right with Carry.....	6-91
SBC (1)	Subtract with Carry Register .....	6-92
SBC (2)	Subtract with Carry Immediate .....	6-93
SETSR	Set SR.....	6-94
SLB	Shift Left Byte.....	6-95
SR	Shift Right.....	6-96
SRA	Shift Right Arithmetic .....	6-97
SRB	Shift Right Byte .....	6-98
SUB (1)	Subtract Register .....	6-99
SUB (2)	Subtract Small Immediate.....	6-100
SUB (3)	Subtract Extended Register .....	6-101
SUB (4)	Subtract Large Immediate .....	6-102
SUB (5)	Subtract 5-bit Immediate.....	6-103
SWI	Software Interrupt .....	6-104
SYS	System.....	6-105
TST (1)	Test Register.....	6-106
TST (2)	Test Small Immediate.....	6-107
TST (3)	Test Large Immediate.....	6-108
TSTSR	Test SR .....	6-109
XOR (1)	XOR Register.....	6-110
XOR (2)	XOR Small Immediate .....	6-111
XOR (3)	XOR Large Immediate.....	6-112

# 1

## PRODUCT OVERVIEW

### INTRODUCTION

The S3FC11B is a calmRISC16 and MAC1616 core-based CMOS single-chip microcontroller. It contains ROM, RAM, 77 I/O pins, programmable 8/16-bit timer/counters, CODEC, PLL, 4-ch A/D converter, 36SEG x 8COM LCD controller/driver, and etc. The S3FC11B can be used for dedicated control functions in a variety of applications, and is especially designed for application with voice synthesizer, voice recognition, or etc.

### FEATURES

#### Memory

- 24K x 16 bits program memory (mtp flash ROM)
- 8K x 16 bits data memory (mtp flash ROM)
- 10K x 8 bits data memory (excluding LCD RAM)

#### 77 I/O Pins

- I/O: 33 pins
- I/O 44 pins (Sharing with segment drive output)

#### SSFDC Interface Logic

- Two selection pins (nCE0, nCE1)

#### 8-Bit Basic Timer

- Programmable interval timer
- 8 kinds of clock source
- Watch-dog timer's clock source (overflow of 8-bit counter)

#### Watchdog Timer

- System reset when 3-bit counter overflow

#### One 8-Bit Timer/Counter 0

- Programmable interval timer
- External event counter function
- PWM function and capture function

#### One 16-Bit Timer/Counter 1

- One 16-bit timer/counter mode
- Two 8-bit timer/counters A/B mode

#### Watch Timer

- Interval time: 3.91ms, 0.25S, 0.5S, and 1S at 32.768 kHz
- 0.5/1/2/4 kHz selectable buzzer output

#### LCD Controller/Driver

- 36 segments and 8 common terminals
- 3, 4 and 8 common selectable
- Internal resistor circuit for LCD bias

#### 8-Bit Serial Interface

- Four programmable operating modes

#### 8/16-Bit Serial Interface for External Codec

- Internal/External clock source selectable
- Two programmable operating modes



**FEATURES (Continued)****Battery Level Detector**

- Programmable low voltage detector
- Two criteria voltage (2.45 V, 2.70 V)

**Phase -Locked Loop (PLL)**

- Programmable clock synthesizer

**Codec**

- 14-bit A/D converter, 14-bit D/A converter
- 3.6 kHz–11 kHz sampling frequency
- 3.0 V–3.6 V operating voltage range

**Analog to Digital Converter (10-bit resolution)**

- 4-channel analog inputs
- 25 $\mu$ S conversion time
- 3.0 V–3.6 V operation voltage range

**Two Power-Down Modes**

- Idle: Only CPU clock stops
- Stop: Selected system clock and CPU clock stop

**Oscillation Sources**

- Crystal or ceramic for main clock
- Programmable oscillation sources for main clock
- 32.768 kHz crystal oscillation circuit for sub clock
- CPU clock divider circuit (divided by 1, 2, 4, or 8)

**Instruction Execution Times**

- Main clocks:
  - 30 ns at 32 MHz when 1 cycle instructions
  - 60 ns at 32 MHz when 2 cycle instructions
- Sub clocks (32.768 kHz):
  - 30.52  $\mu$ s when 1 cycle instructions
  - 61.04  $\mu$ s when 2 cycle instructions

**Operating Voltage Range**

- 2.0 V to 3.6 V

**Operating Temperature Range**

- –25 °C to +85 °C

**Current Consumption**

- Sub idle current: 6.0  $\mu$ A at  $V_{DD} = 3.3V$

**Package Type**

- 100-QFP, 100-TQFP package

BLOCK DIAGRAM

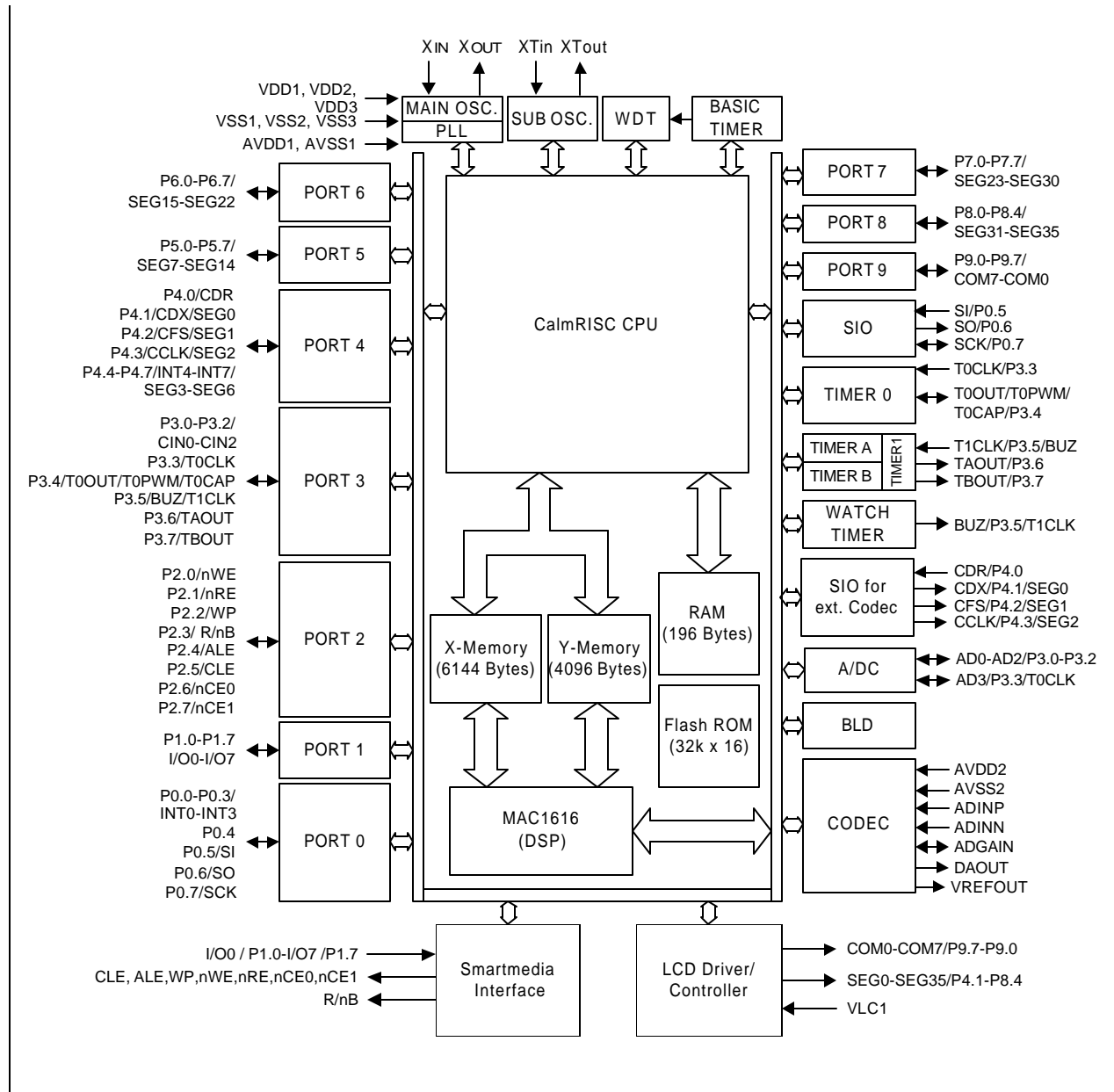


Figure 1-1. S3CC11B/FC11B Top Block Diagram

PIN ASSIGNMENT

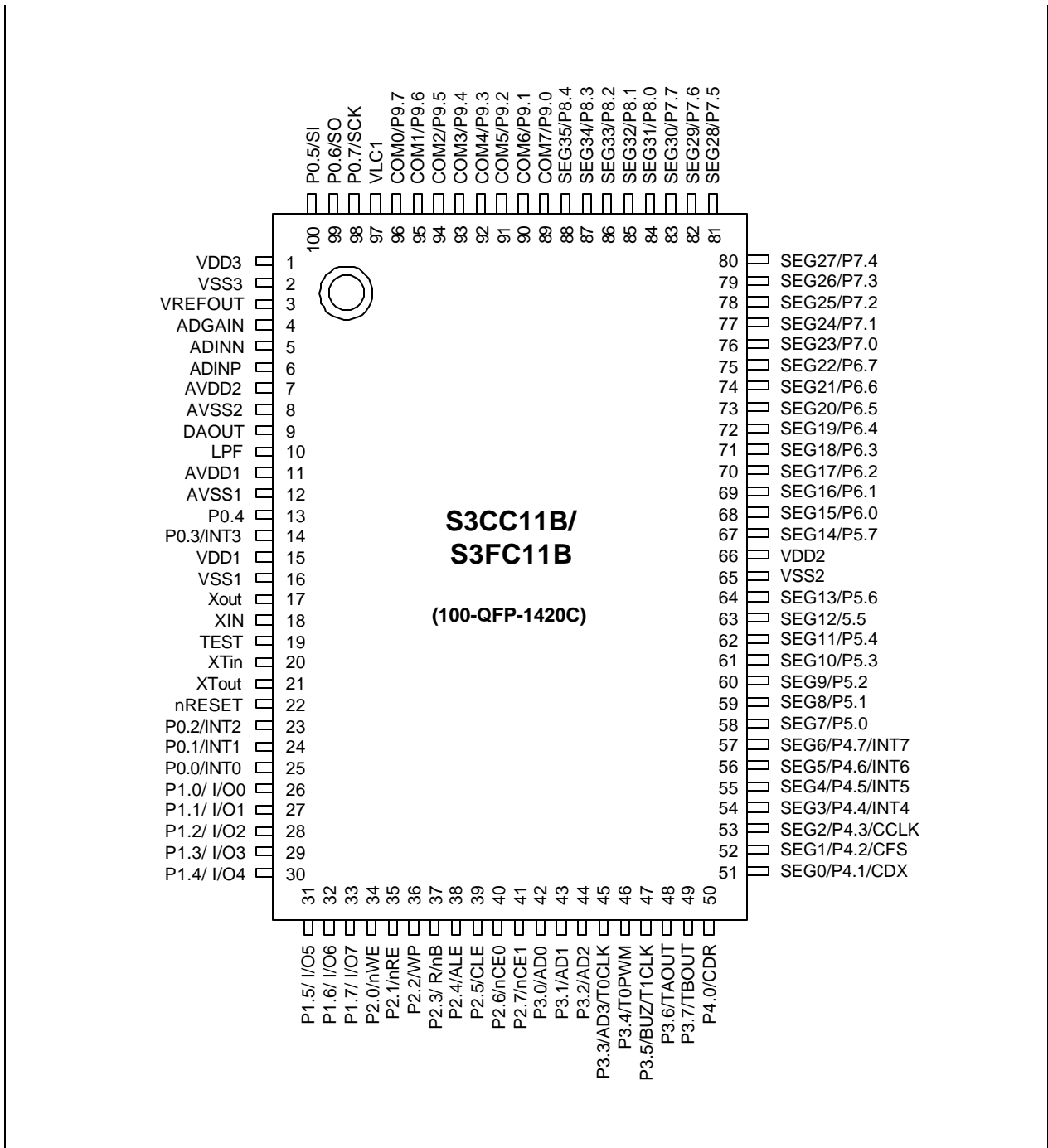


Figure 1-2. S3CC11B/FC11B Pin Assignments (100-QFP-1420C)

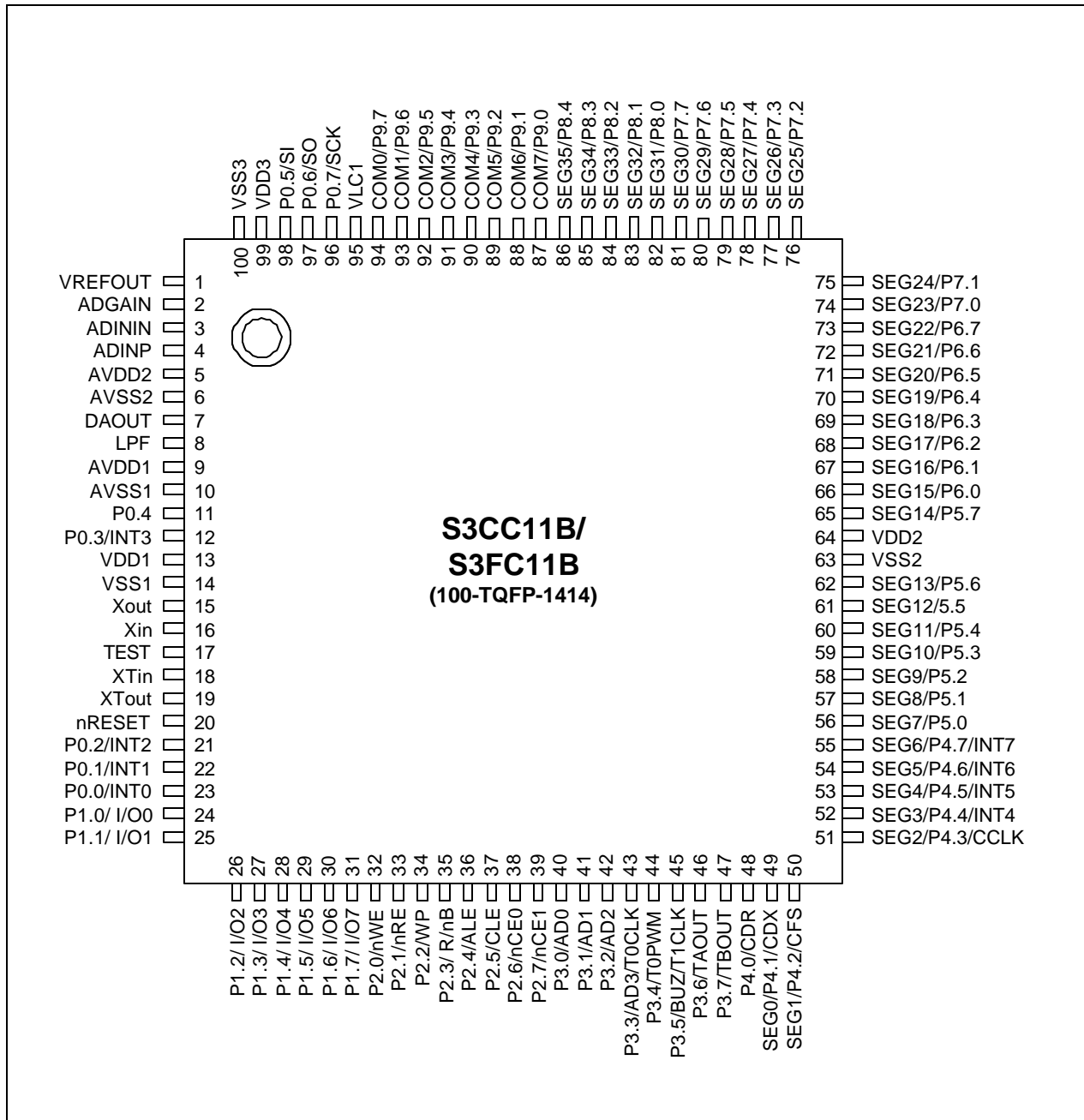


Figure 1-3. S3CC11B/FC11B Pin Assignments (100-TQFP-1414)

Table 1-1. S3CC11B/FC11B Pin Description

Name	Type	Description	Circuit Type	Number	Shared Pins
P0.0 P0.1 P0.2 P0.3 P0.4 P0.5 P0.6 P0.7	I/O	I/O port with bit programmable pins; Schmitt trigger input or push-pull, open-drain output and software assignable pull-ups; P0.0-P0.3 is alternatively used for external interrupt input (noise filters).	5    4	25(23) 24(22) 23(21) 14(12) 13(11) 100(98) 99(97) 98(96)	INT0 INT1 INT2 INT3 – SI SO SCK
P1.0 – P1.7	I/O	I/O port with nibble-programmable pins; Schmitt trigger input or push-pull, open-drain output and software assignable pull-ups; Also configurable as smartmedia interface lines I/O0 – I/O7.	4	26 – 33 (24–31)	I/O0 - I/O7
P2.0 P2.1 P2.2 P2.3 P2.4 P2.5 P2.6 P2.7	I/O	I/O port with bit-programmable pins; Schmitt trigger input or push-pull, open-drain output and software assignable pull-ups; Also configurable as smartmedia interface lines nWE, nRE, WP, R/nB, ALE, CLE, nCE0, and nCE1.	4	34(32) 35(33) 36(34) 37(35) 38(36) 39(37) 40(38) 41(39)	nWE nRE WP R/nB ALE CLE nCE0 nCE1
P3.0 P3.1 P3.2 P3.3 P3.4 P3.5 P3.6 P3.7	I/O	I/O port with bit-programmable pins; Schmitt trigger input or push-pull, open-drain output and software assignable pull-ups. P3.0 – P3.3 is alternatively used for analog input.	6   4	42(40) 43(41) 44(42) 45(43) 46(44) 47(45) 48(46) 49(47)	AD0 AD1 AD2 AD3/T0CLK TOOUT/T0PWM/T0 CAP BUZ/T1CLK TAOUT TBOUT
P4.0 P4.1 P4.2 P4.3 P4.4 – P4.7	I/O	I/O port with bit-programmable pins; Schmitt trigger input or push-pull, open-drain output and software assignable pull-ups. P4.4 – P4.7 is alternatively used for external interrupt input (noise filters, interrupt enable control).	4 9  10	50(48) 51(49) 52(50) 53(51) 54 – 57 (52–55)	CDR CDX/SEG0 CFS/SEG1 CCLK/SEG2 INT4-INT7/ SEG3 -SEG6
P5.0 – P5.6 P5.7	I/O	I/O port with bit-programmable pins; Schmitt trigger input or push-pull, open-drain output and software assignable pull-ups.	9	58 – 64 (56–62) 67(65)	SEG7–SEG13  SEG14
P6.0 – P6.7	I/O	I/O port with nibble-programmable pins; Schmitt trigger input or push-pull output and software assignable pull-ups.	8	68 – 75 (66–73)	SEG15–SEG22
P7.0 – P7.7	I/O	Same general characteristics as port6.	8	76 – 83 (74–81)	SEG23–SEG30

NOTE: The parentheses are a pin number of 100-TQFP package.

Table 1-1. S3CC11B/FC11B Pin Description (Continued)

Name	Type	Description	Circuit Type	Number	Shared Pins
P8.0 – P8.4	I/O	Same general characteristics as port6.	8	84 – 88 (82–86)	SEG31– SEG35
P9.0 – P9.7	I/O	Same general characteristics as port6.	8	89 – 96 (87–94)	COM7–COM0
ADINN	I	Analog negative input pin.	–	5(3)	–
ADINP	I	Analog positive input pin.	–	6(4)	–
ADGAIN	I/O	Analog input gain control pin.	–	4(2)	–
VREFOUT	O	Vref output pin.	–	3(1)	–
DAOUT	O	Digital to analog converter output pin.	–	9(7)	–
LPF	I/O	PLL loop filter pin	–	10(8)	–
COM0 – COM7	I/O	LCD common data output pins.	8	96–89 (94–87)	P9.7-P9.0
SEG0 SEG1 SEG2 SEG3 – SEG6	I/O	LCD segment data output pins.	9  10	51(49) 52(50) 53(51) 54 – 57 (52–55)	P4.1/CDX P4.2/CFS P4.3/CCLK P4.4-P4.7/ INT4-INT7
SEG7 – SEG13 SEG14 SEG15 – SEG35	I/O	LCD segment data output pins.	9  8	58–64 (56–62) 67(65) 68–88 (66–86)	P5.0-P5.6  P5.7 P6.0-P8.4
VLC1	–	LCD power supply pins.	–	97(95)	–
INT0 – INT2	I/O	External interrupt input pins.	5	25–23 (23–21)	P0.0 - P0.2
INT3 INT4 – INT7			10	14(12) 54 – 57 (52–55)	P0.3 P4.4 – P4.7
T1CLK	I/O	Timer 1/A external clock input pin.	4	47(45)	P3.5/BUZ
TAOUT TBOUT	I/O	Timer 1/A and B clock output pins.	4	48(46) 49(47)	P3.6 P3.7
BUZ	I/O	Buzzer signal output pin.	4	47(45)	P3.5/T1CLK
SI	I/O	Serial data input pin.	4	100(98)	P0.5
SO	I/O	Serial data output pin.	4	99(97)	P0.6
SCK	I/O	Serial I/O interface clock signal pin.	4	98(96)	P0.7
T0OUT	I/O	Timer0's interval output pin.	4	46(44)	P3.4/T0PWM/ T0CAP
T0PWM	I/O	Timer0's PWM output pin.	4	46(44)	P3.4/T0OUT/ T0CAP

**NOTE:** The parentheses are a pin number of 100-TQFP package.

Table 1-1. S3CC11B/FC11B Pin Description (Continued)

Name	Type	Description	Circuit Type	Number	Shared Pins
T0CAP	I/O	Timer0's Capture input pin.	4	46(44)	P3.4/T0OUT/ T0PWM
T0CLK	I/O	Timer0's external clock input pin.	6	45(43)	P3.3/AD3
nWE	I/O	Write enable pin.	4	34(32)	P2.0
nRE	I/O	Read enable pin.	4	35(33)	P2.1
WP	I/O	Write protect pin.	4	36(34)	P2.2
R/nB	I/O	Ready and busy status pin.	4	37(35)	P2.3
ALE	I/O	Address latch enable pin.	4	38(36)	P2.4
CLE	I/O	Command latch enable pin.	4	39(37)	P2.5
nCE0	I/O	Chip enable 0 pin.	4	40(38)	P2.6
nCE1	I/O	Chip enable 1 pin.	4	41(39)	P2.7
I/O0 – I/O7	I/O	Smartmedia interface lines.	4	26 – 33 (24–31)	P1.0 – P1.7
AD0 – AD2	I/O	Analog input pins for A/D converter.	6	42 – 44 (40–42)	P3.0 – P3.2
AD3				45(43)	P3.3/T0CLK
CDR	I/O	Receive data input pin for external codec.	4	50(48)	P4.0
CDX	I/O	Transmit data output pin for external codec.	9	51(49)	P4.1/SEG0
CFS	I/O	Frame sync pulse for external codec.	9	52(50)	P4.2/SEG1
CCLK	I/O	Master and bit clock for external codec.	9	53(51)	P4.3/SEG2
AVDD1, AVSS1	–	Analog power pins for PLL block.	–	11, 12 (9, 10)	–
AVDD2, AVSS2	–	Analog power pins for CODEC block.	–	7, 8 (5, 6)	–
nRESET	I	System reset pin.	2	22(20)	–
XTin,XTout	–	Crystal oscillator pins for sub clock.	–	20, 21 (18, 19)	–
Xin, Xout	–	Main oscillator pins.	–	18, 17 (16, 15)	–
TEST	I	Input pin for test.(must be connected to VSS)	–	19(17)	–
VDD1, VSS1	–	Power input pins for CPU.	–	15, 16 (13, 14)	–
VDD2, VSS2	–	Power input pins for peripheral block.	–	66, 65 (64, 63)	–
VDD3, VSS3	–	Power input pins for peripheral block.	–	2,1 (99,100)	–

**NOTE:** The parentheses are a pin number of 100-TQFP package.

### PIN CIRCUIT DIAGRAMS

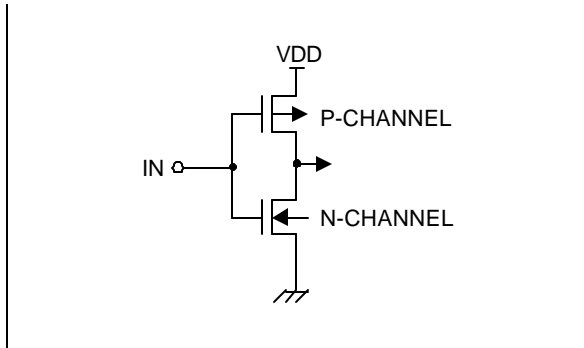


Figure 1-4. Pin Circuit Type 1

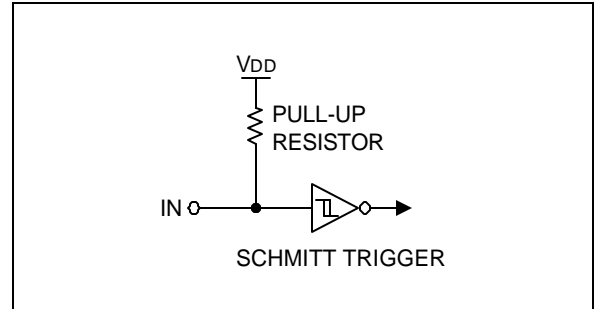


Figure 1-5. Pin Circuit Type 2 (nRESET)

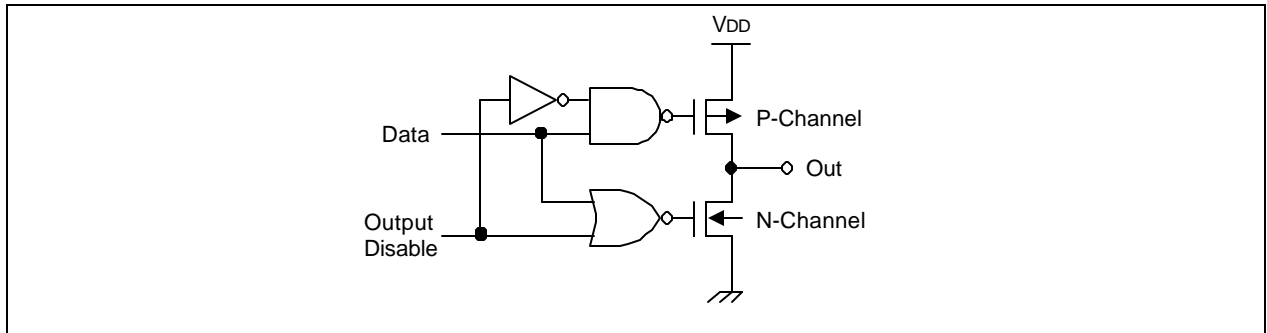


Figure 1-6. Pin Circuit Type 3



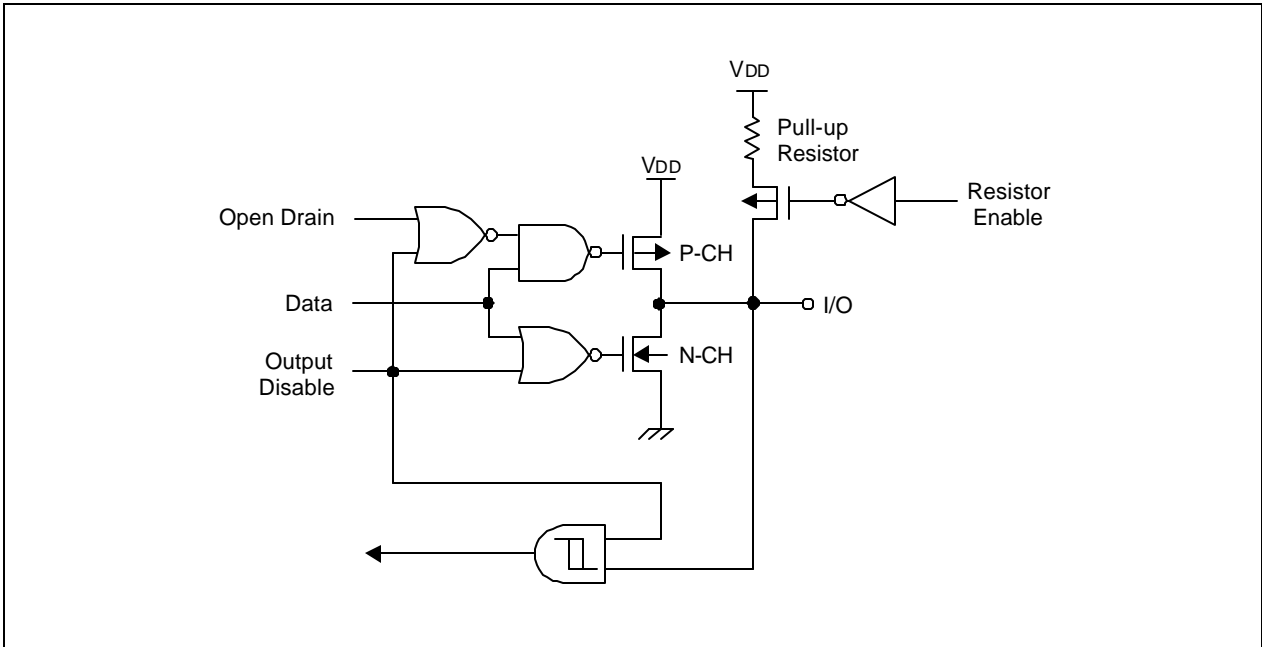


Figure 1-7. Pin Circuit Type 4 (P0.4-P0.7, P1, P2, P3.4-P3.7, P4.0)

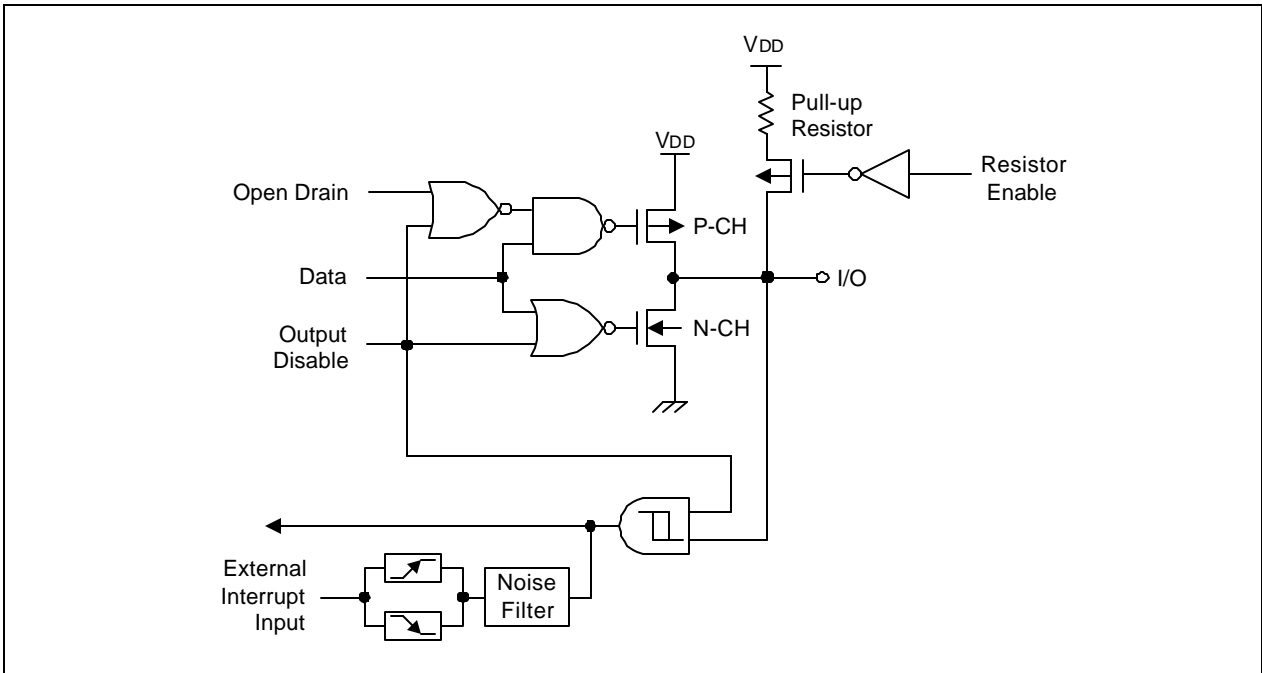


Figure 1-8. Pin Circuit Type 5 (P0.0-P0.3)

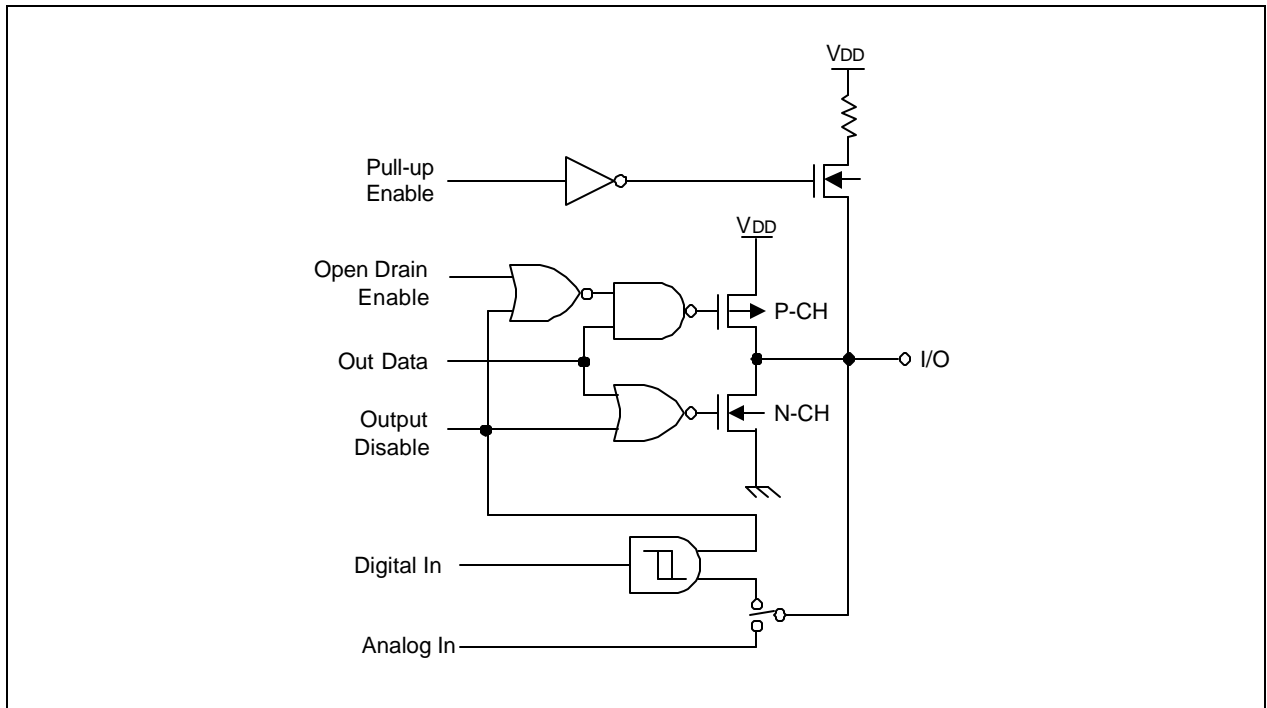


Figure 1-9. Pin Circuit Type 6 (P3.0-P3.3)

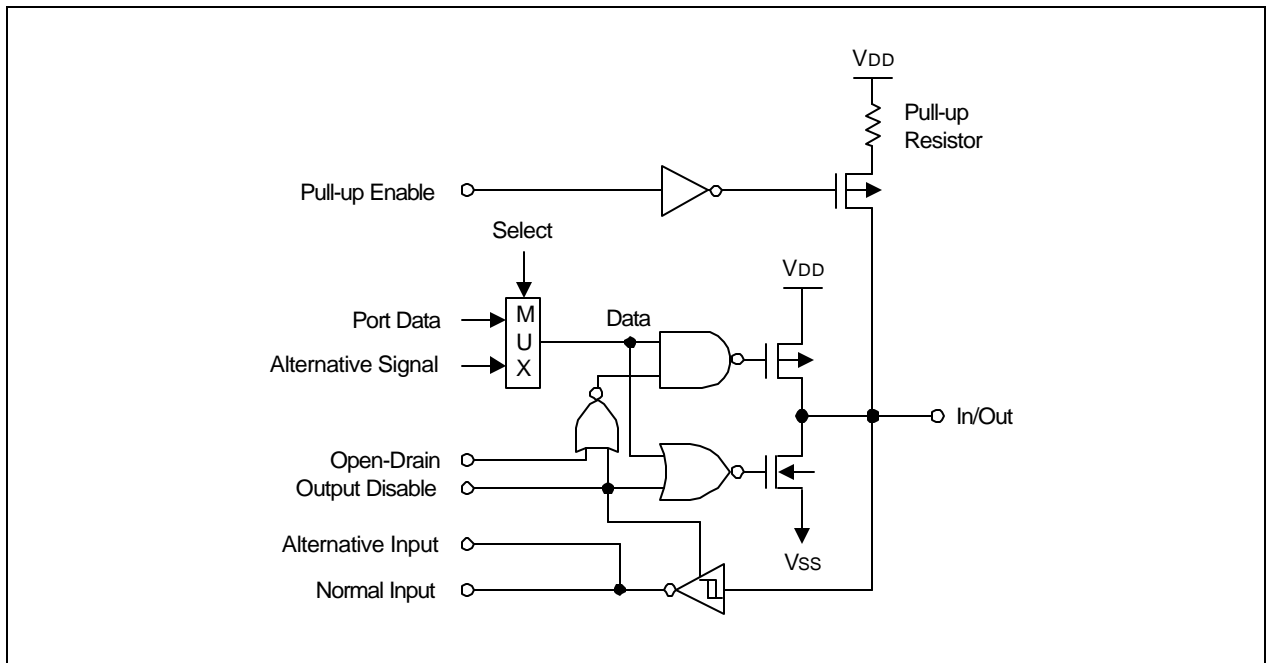


Figure 1-10. Pin Circuit Type 7

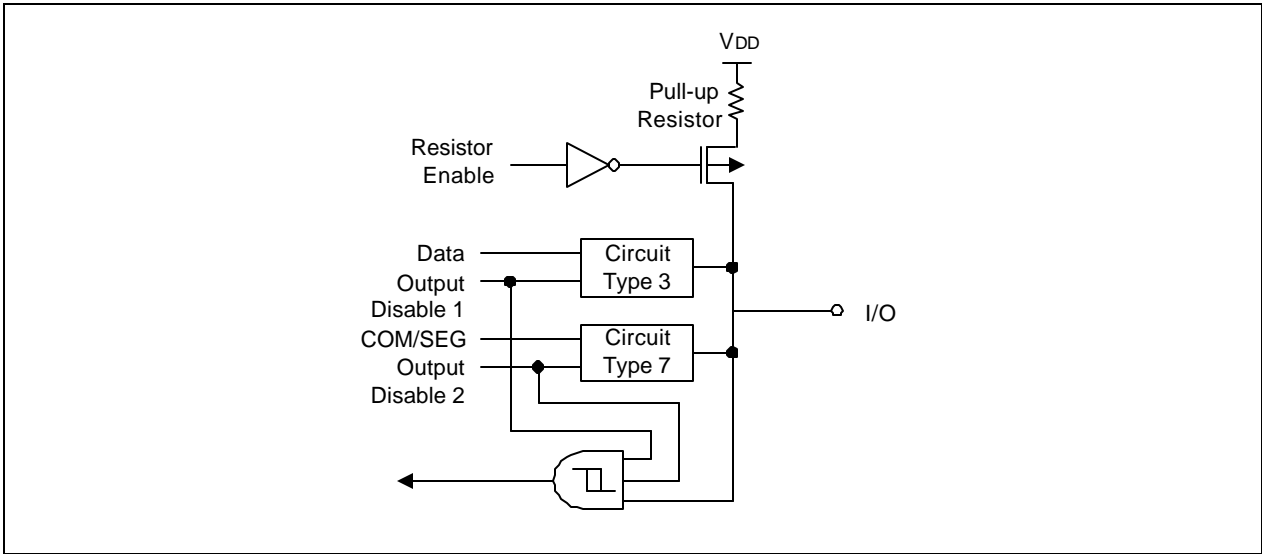


Figure 1-11. Pin Circuit Type 7 (P6, P7, P8, P9)

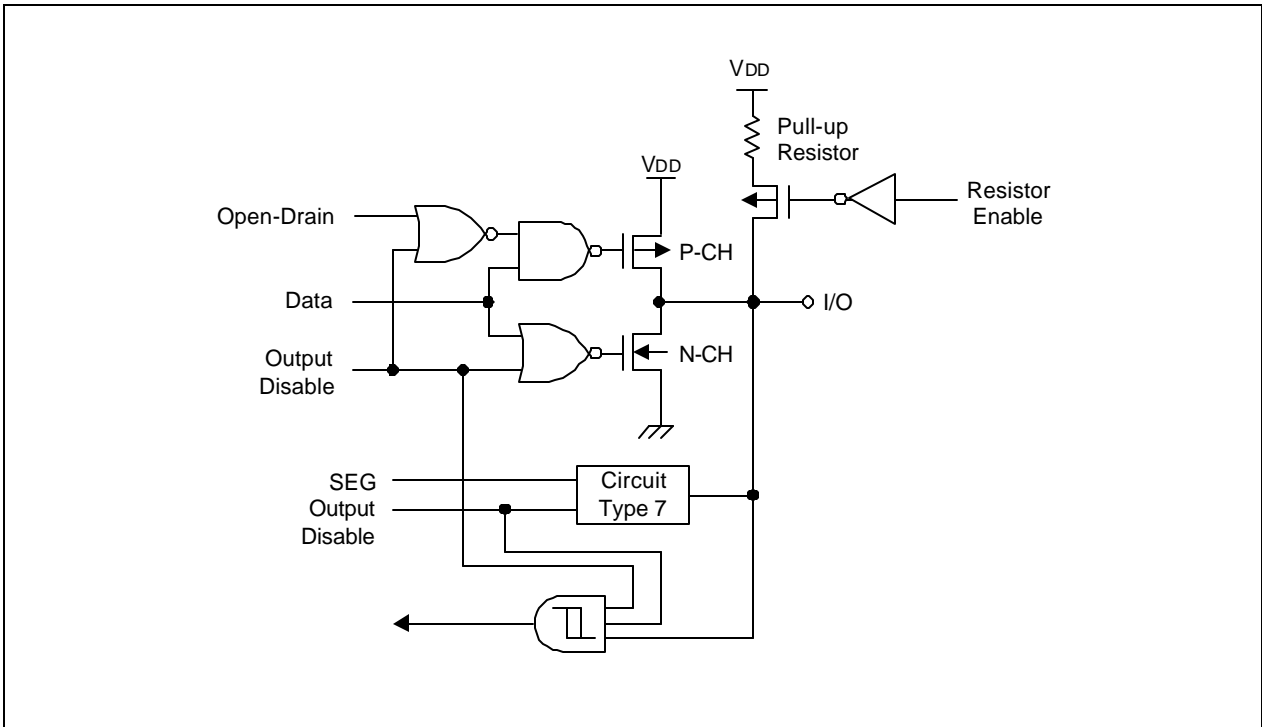


Figure 1-12. Pin Circuit Type 9 (P4.1-P4.3, P5)

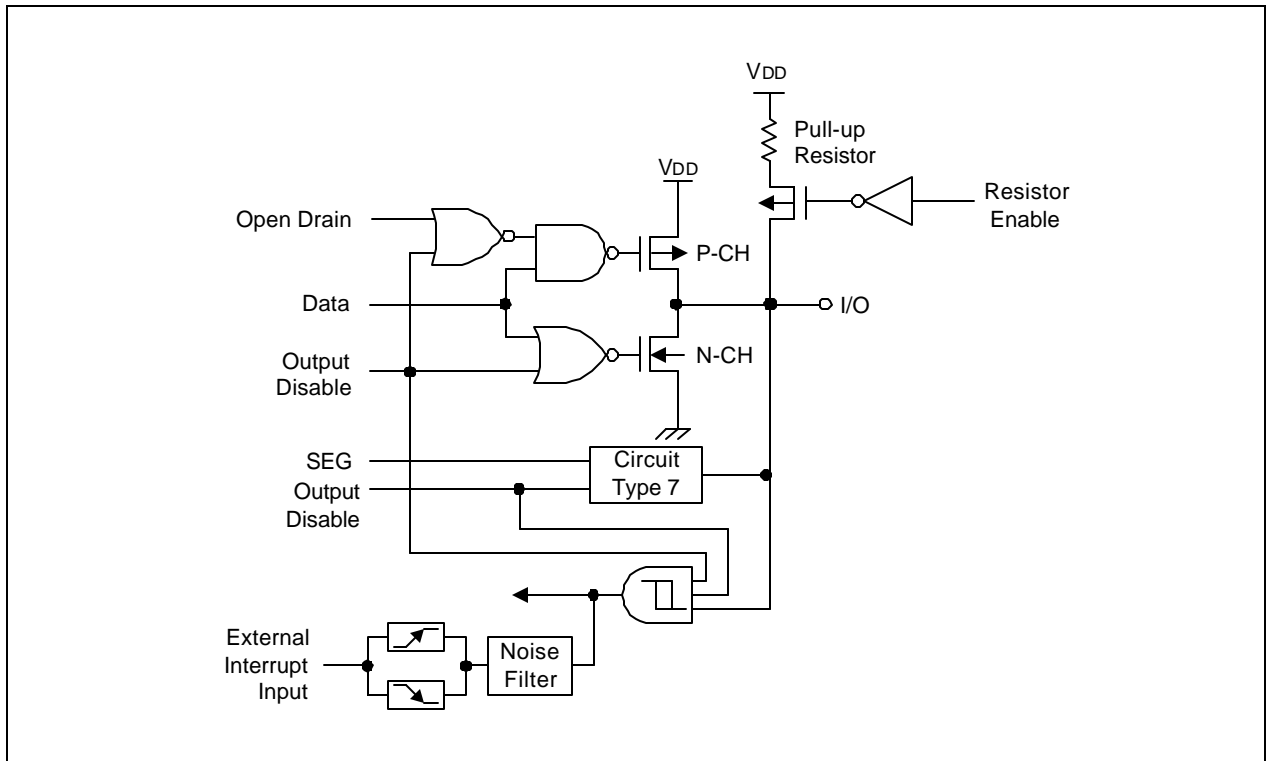


Figure 1-13. Pin Circuit Type 10 (P4.4-P4.7)

## NOTES

# 2 ADDRESS SPACE

## OVERVIEW

CalmRISC16 has 21-bit program address lines, PA[20:0] (equivalent to PC[21:1]), which supports up to 32K word of program memory.

The 32K word program memory space is divided into 24K word internal program memory and 8K word Data Memory (Data ROM) area.

CalmRISC16 also has 22-bit data memory address lines, DA[21:0], which supports up to 10K byte.

### Memory configuration in CalmRISC16 side

Data Memory:

10K byte internal data memory

Program Memory:

24K word internal program memory

8K word data memory (Data ROM: YROM)

### Memory configuration in CalmMAC24 side

Data Memory:

X-Memory area - 3K word internal memory (6K byte)

Y-Memory area - 2K word internal memory (4K byte)

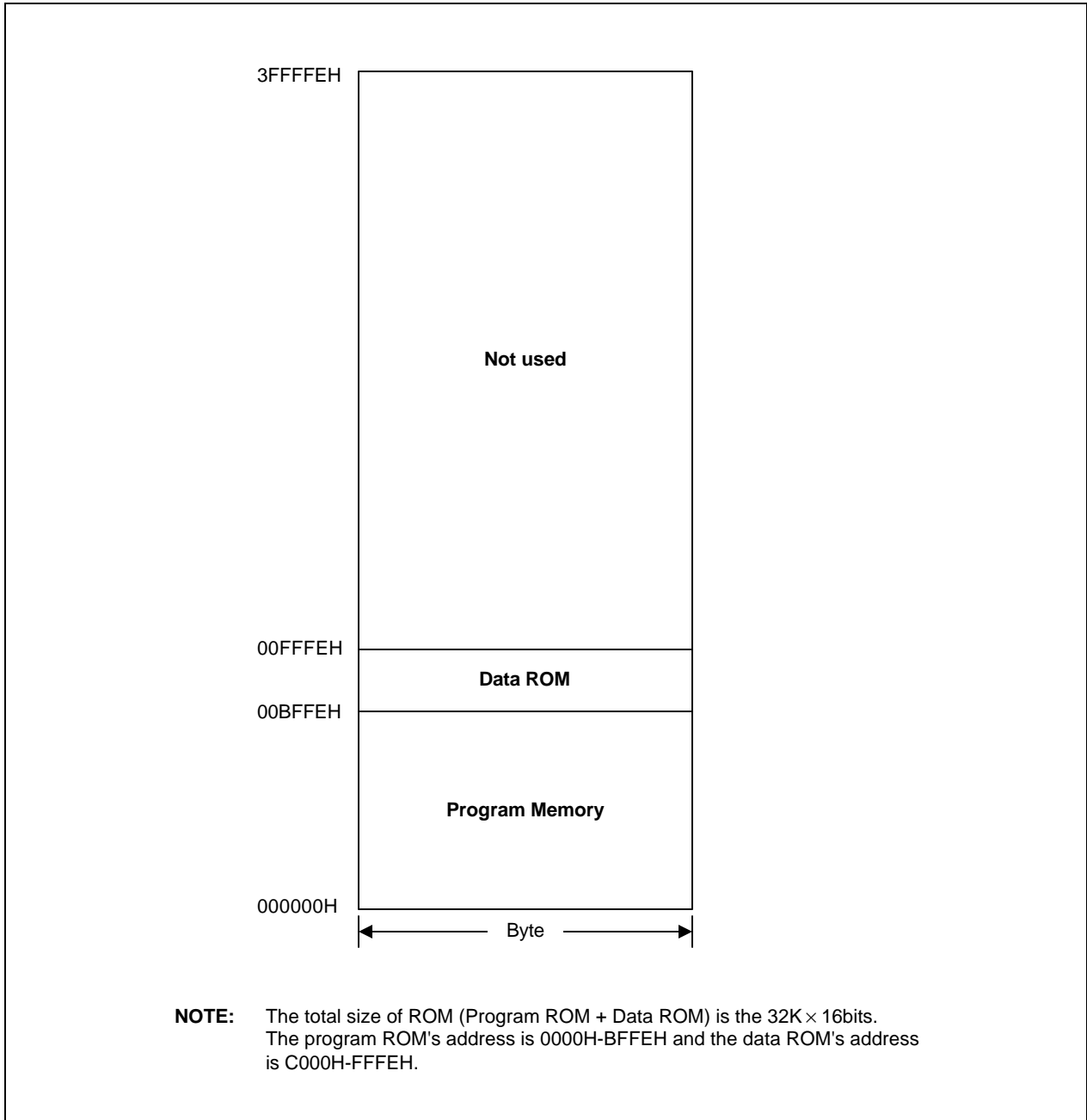
Program Memory:

24K word internal program memory

8K word data memory (Data ROM: YROM)

**PROGRAM MEMORY**

Program memory configuration is shown in Figure 2-1. The total size of ROM (Program ROM + Data ROM) is the  $32\text{K} \times 16$  bits. The program ROM's address is 0000H–BFFE H and the data ROM's address is C000H–FFFE H.



**Figure 2-1. Program Memory Configuration**

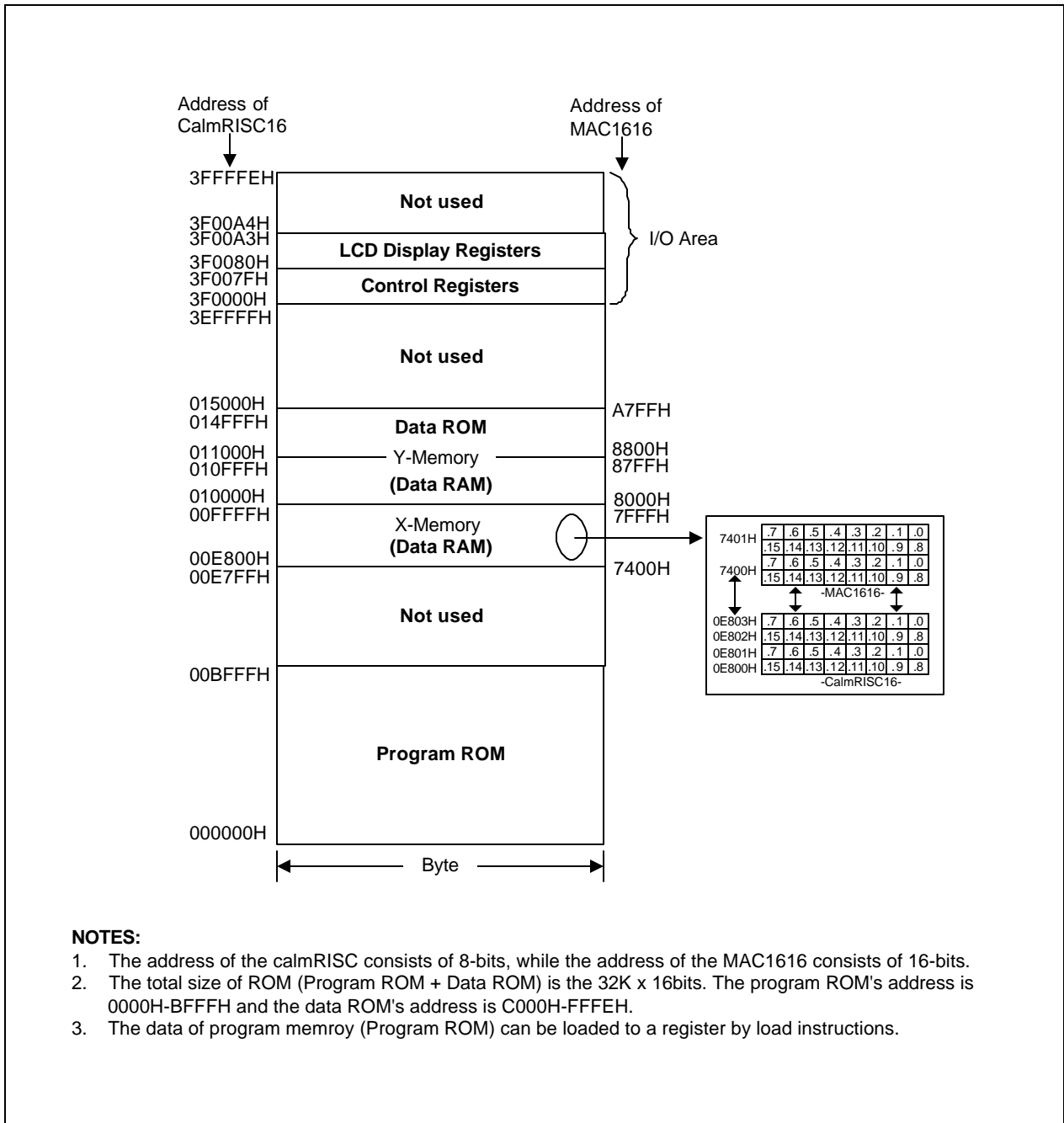
**DATA MEMORY**

Data memory configuration is shown in Figure 2.2. CalmMAC16 only can access the internal data memory and if the memory request tries to access non-existent memory area, FIQ(Fast Interrupt request) is generated. In this case, if FE bit in CalmRISC16's SR register is 1, the violation service routine is called and served. CalmRISC16 can access the internal data memory. But the FIQ is not used in the S3CC11B. The address of the CalmRISC16 consists of 8 bits, while the address of the MAC1616 consists of 16bits. So, if the address of the CalmRISC16 is E800H, the address of the MAC1616 is 7400H. Program ROM can be accessed in the view of ROM (with LDC instruction) that has 0000–BFFE address. Also and in the view of RAM (with LDB, LDW instructions) that has same address.

Data ROM can be accessed in the view of ROM (with LDC instruction) that has C000 – FFFE address. Also and in the view of RAM (with LDB, LDW instructions) that has 11000 – 14FFF address. Of course, Data ROM can be accessed by MAC1616. (The address ranges are 8800H to A7FFH.)

The memory violation (access the non-existent area) FIQ can be also generated.





**NOTES:**

1. The address of the calmRISC consists of 8-bits, while the address of the MAC1616 consists of 16-bits.
2. The total size of ROM (Program ROM + Data ROM) is the 32K x 16bits. The program ROM's address is 0000H-BFFFH and the data ROM's address is C000H-FFFFH.
3. The data of program memroy (Program ROM) can be loaded to a register by load instructions.

**Figure 2-2. Data Memory Configuration**

# 3 Calm16Core

## INTRODUCTION

The main features of CalmRISC16, a 16-bit embedded RISC MCU core, are high performance, low power consumption, and efficient coprocessor interface. It can operate up to 32MHz, and consumes 200 $\mu$ A/MHz @3.3V. When operating with MAC1616, a 16-bit fixed point DSP coprocessor, CalmRISC16 can operate up to 32MHz. Through efficient coprocessor interface, CalmRISC16 provides a powerful and flexible MCU+DSP solution. The following gives brief summary of main features of CalmRISC16.

## FEATURES

### Architecture

- Harvard RISC architecture
- 5-Stage pipeline

### Registers

- Sixteen 16-bit general registers
- Eight 6-bit extension registers
- 22-bit Program Counter (PC)
- 16-bit Status Register (SR)
- Five saved registers for interrupts.

### Instruction Set

- 16-bit instruction width for 1-word instructions
- 32-bit instruction width for 2-word instructions
- Load/Store instruction architecture
- Delayed branch support
- C-language/OS support
- Bit operation for I/O process

### Instruction Execution Time

- One instruction/cycle for basic instructions

### Address Space

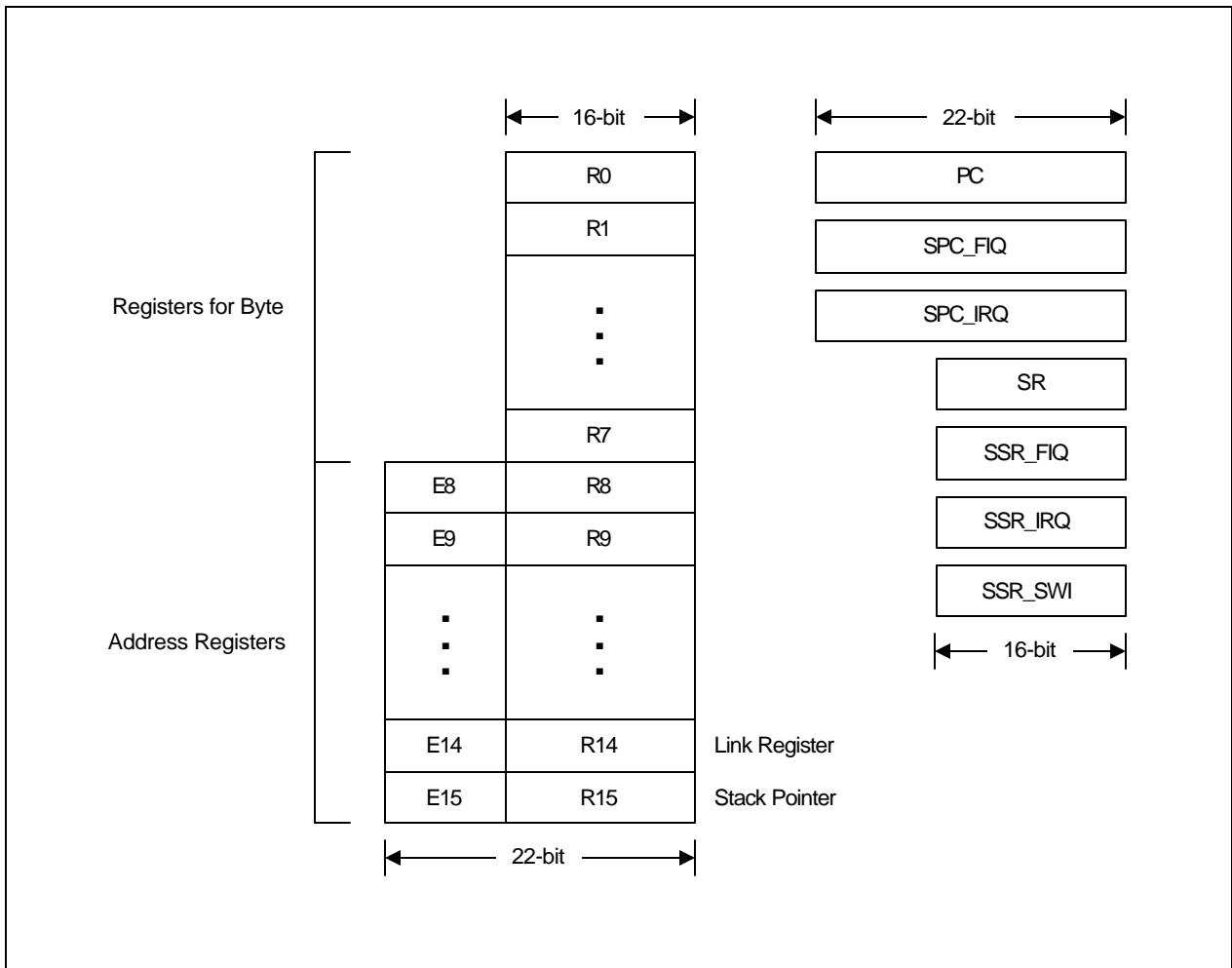
- 2M word for Program Memory
- 4M byte for Data Memory

## REGISTERS

In CalmRISC16 there are sixteen 16-bit general registers, eight 6-bit extension registers, a 16-bit Status Register(SR), a program counter (PC), and five saved registers.

### GENERAL REGISTERS & EXTENSION REGISTERS

The following figure shows the structure of the general registers and the extension registers.



**Figure 3-1. Register Structure in CalmRISC16**

The general registers (from R0 to R15) can be either a source register or a destination register for almost all ALU operations, and can be used as an index register for memory load/store instructions (e.g., LDW R3, @[A8+R2]). The 6-bit extension registers (from E8 to E15) are used to form a 22-bit address register (from A8 to A15) by concatenating with a general register (from R8 to R15). The address registers are used to generate 22-bit program and data addresses.

## SPECIAL REGISTERS

The special registers consist of 16-bit SR (Status Register), 22-bit PC (Program Counter), and saved registers for IRQ(interrupt), FIQ(fast interrupt), and SWI(software interrupt). When IRQ interrupt occurs, the most significant 6 bits of the return address are saved in SPCH\_IRQ, the least significant 16 bits of the return address are saved in SPCL\_IRQ, and the status register is saved in SSR\_IRQ. When FIQ interrupt occurs, the most significant 6 bits of the return address are saved in SPCH\_FIQ, the least significant 16 bits of the return address are saved in SPCL\_FIQ, and the status register is saved in SSR\_FIQ. When a SWI instruction is executed, the return address is saved in A14 register (E14 concatenated with R14), and the status register is saved in SSR\_SWI. The least significant bit of PC, SPCL\_IRQ and SPCL\_FIQ is read only and its value is always 0.

— The 16-bit register SR has the following format.

15								8	7							0
T	-	-	--	-	-	-	-	-	-	PM	Z1	Z0	V	TE	IE	FE

- FE: FIQ enable bit, FIQ is enabled when FE is set.
- IE: IRQ enable bit, IRQ is enabled when IE is set.
- TE: TRQ enable bit, Trace is enabled when TE is set.
- V: overflow flag, set/clear accordingly when arithmetic instructions are executed.
- Z0: zero flag of R6, set when R6 equals zero and used as the branch condition when BNZD instruction with R6 is executed.
- Z1: zero flag of R7, set when R7 equals zero and used as the branch condition when BNZD instruction with R7 is executed.
- PM: privilege mode bit. PM = 1 for privilege mode and PM = 0 for user mode
- T: true flag, set/clear as a result of an ALU operation.

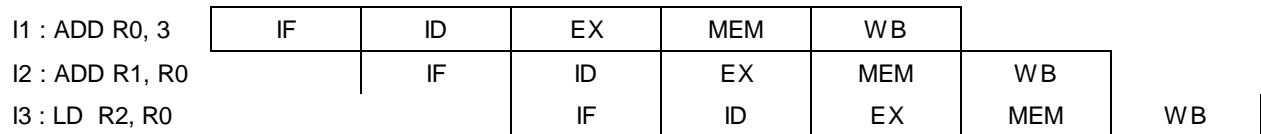
FE, IE, TE, and PM bits can be modified only when PM = 1 (privilege mode). The only way to change from user mode to privilege mode is via interrupts including SWI instructions. The reserved bit of SR (from bit 7 to bit 14) can be used for other purposes without any notice. Hence programmers should not depend on the value of the reserved bits in their programming. The reserved bits are read as 0 value.

## PIPELINE STRUCTURE

CalmRISC16 has a 5-stage pipeline architecture. It takes 5 cycles for an instruction to do its operation. In a pipeline architecture, instructions are executed overlapped, hence the throughput is one instruction per cycle. Due to data dependency, control dependency, and 2 word instructions, the throughput is about 1.2 on the average. The following diagram depicts the 5-stage pipeline structure.



In the first stage, which is called IF (Instruction Fetch) stage, an instruction is fetched from program memory. In the second stage, which is called ID (Instruction Decoding) stage, the fetched instruction is decoded, and the appropriate operands, if any, for ALU operation are prepared. In the case of branch or jump instructions, the target address is calculated in ID stage. In the third stage, which is called EX (Execution) stage, ALU operation and data address calculation are executed. In the fourth stage, which is called MEM (Memory) stage, data transfer from/to data memory or program memory is executed. In the fifth stage, which is called WB (Write Back) stage, a write-back to register file can be executed. The following figure shows an example of pipeline progress when 3 consecutive instructions are executed.



In the above example, the instruction I2 needs the result of the instruction I1 before I1 completes. To resolve this problem, the EX stage result of I1 is forwarded to ID stage of I2. Similar forwarding mechanism occurs from MEM stage of I1 to ID stage of I3.

The pipeline cannot progress (called a pipeline stall) due to a data dependency, a control dependency, or a resource conflict.

When a source operand of an ALU instruction is from a register, which is loaded from memory in the previous instruction, 1 cycle of pipeline stall occurs (called load stall). Such load stalls can be avoided by smart reordering of the instruction sequences. CalmRISC16 has 2 classes of branch instructions, those with a delay slot and without a delay slot. Non-delay slot branch instructions incurs a 1 cycle pipeline stall if the branch is taken, due to a control dependency. For branch instructions with a delay slot, no cycle waste is incurred if the delay slot is filled with a useful instruction (or non NOP instruction). Pipeline stalls due to resource conflicts occurs when two different instructions access at the same cycle the same resource such as the data memory and the program memory. LDC (data load from program memory) instruction causes a resource conflict on the program memory. Bit operations such as BITR and BITS (read-modify-write instructions) cause a resource conflict on the data memory.

## INTERRUPTS

In CalmRISC16, there are five interrupts: RESET, FIQ, IRQ, TRQ, SWI. The RESET, FIQ, and IRQ interrupts correspond to external requests. TRQ and SWI interrupts are initiated by an instruction (therefore, in a deterministic way). The following table shows a summary of interrupts.

Name	Priority	Address	Description
RESET	1	000000h	Hardware Reset
FIQ	3	000002h	Fast Interrupt Request
IRQ	5	000004h	Interrupt Request
TRQ	2	000006h	Trace Request
SWI	4	000008h– 0000feh	Software Interrupt

When nRES (an input pin CalmRISC16 core) signal is released (transition from 0 to 1), “JMP addr:22” is automatically executed by CalmRISC16. Among the 22-bit address addr:22, the most significant 6 bits are forced to 0, and the least significant 16 bits are the contents of 000000h (i.e., reset vector address) of the program memory. In other words, “JMP {6’h00, PM[000000h]}” instruction is forced to the pipeline. The initial value of PM bit is 1 (that is, in privilege mode) and the initial values of other bits in SR register are 0. All other registers are not initialized (i.e., unknown).

When nFIQ (an input pin CalmRISC16 core) signal is active (transition from 1 to 0), “JMP addr:22” instruction is automatically executed by CalmRISC16. The address of FIQ interrupt service routine is in 000002h (i.e., FIQ vector address) of the program memory (i.e., “JMP {6’h00, PM[000002h]}”). The return address is saved in {SPCH\_FIQ, SPCL\_FIQ} register pair, and the SR value is saved in SSR\_FIQ register. PM bit is set. FE, IE, and TE bits are cleared. When RET\_FIQ instruction is executed, SR value is restored from SSR\_FIQ, and the return address is restored into PC from {SPCH\_FIQ, SPCL\_FIQ}.

When nIRQ signal (an input pin CalmRISC16 core) is active (transition from 1 to 0), “JMP {6’h00, PM[000004h]}” instruction is forced to the instruction pipeline. The return address is saved in {SPCH\_IRQ, SPCL\_IRQ} register pair, and the SR value is saved in SSR\_IRQ register. PM bit is set. IE and TE bits are cleared. When RET\_IRQ instruction is executed, SR value is restored from SSR\_IRQ, and return address is restored to PC from {SPCH\_IRQ, SPCL\_IRQ}.

When TE bit is set, TRQ interrupt happens and “JMP {6’h00, PM[000006h]}” instruction is executed right after each instruction is executed. TRQ interrupt uses the saved registers of IRQ(that is, {SPCH\_IRQ, SPCL\_IRQ} register pair and SSR\_IRQ) to save the return address and SR, respectively. PM bit is set. IE, TE bits are cleared.

When “SWI imm:6” instruction is executed, the return address is saved in the register A14, and the value of SR is saved in SSR\_SWI. Then the program sequence jumps to the address (imm:6 \* 4). PM bit is set. IE and TE bits are cleared. “SWI 0” and “SWI 1” are prohibited because the addresses are reserved for other interrupts. When RET\_SWI instruction is executed, SR is restored from SSR\_SWI, and the return address is restored to PC from A14.

### NOTES

1. 6’h00 is defined as 00 (or zero) in 6 bits
2. imm:6 is defined as 6-bit immediate number

## NOTES

# 4 EXCEPTIONS

## OVERVIEW

Exceptions in CalmRISC16 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 FIQ starts from 0H:PM[000002H]. Note that “:” means concatenation and PM[\*] stands for the 16-bit content at the address \* of the program memory. When an IRQ or FIQ occurs, current PC is pushed in the SPC\_IRQ, SPC\_FIQ on an exception. And if SWI is executed, current PC is pushed in the E14:R14 register.

**Table 4-1. Exceptions**

Name	Address	Priority	Description
Reset	000000H	1 <sup>st</sup>	Exception due to reset release.
FIQ	000002H	3 <sup>rd</sup>	Exception due to <i>nFIQ</i> signal. Maskable by setting FE
IRQ	000004H	5 <sup>th</sup>	Exception due to <i>nIRQ</i> signal. Maskable by setting IE
TRQ	000006H	2 <sup>nd</sup>	Exception due to TE bit in SR register
SWI	000008H– 0000FEH	4 <sup>th</sup>	Exception due to SWI execution

**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 nRES (an input pin CalmRISC16 core) signal is released (transition from 0 to 1), “JMP addr:22” is automatically executed by CalmRISC16. Among the 22-bit address addr:22, the most significant 6 bits are forced to 0, and the least significant 16 bits are the contents of 000000h (i.e., reset vector address) of the program memory. In other words, “JMP {6’h00, PM[000000h]}” instruction is forced to the pipeline. The initial value of PM bit is 1 (that is, in privilege mode) and the initial values of other bits in SR register are 0. All other registers are not initialized (i.e., unknown).



### FIQ EXCEPTION

When nFIQ (an input pin CalmRISC16 core) signal is active (transition from 1 to 0), "JMP addr:22" instruction is automatically executed by CalmRISC16. The address of FIQ interrupt service routine is in 000002h (i.e., FIQ vector address) of the program memory (i.e., "JMP {6'h00, PM[000002h]}"). The return address is saved in {SPCH\_FIQ, SPCL\_FIQ} register pair, and the SR value is saved in SSR\_FIQ register. PM bit is set. FE, IE, and TE bits are cleared. When RET\_FIQ instruction is executed, SR value is restored from SSR\_FIQ, and the return address is restored into PC from {SPCH\_FIQ, SPCL\_FIQ}. But the FIQ is not used in the S3CC11B.

### IRQ EXCEPTION

When nIRQ signal (an input pin CalmRISC16 core) is active (transition from 1 to 0), "JMP {6'h00, PM[000004h]}" instruction is forced to the instruction pipeline. The return address is saved in {SPCH\_IRQ, SPCL\_IRQ} register pair, and the SR value is saved in SSR\_IRQ register. PM bit is set. IE and TE bits are cleared. When RET\_IRQ instruction is executed, SR value is restored from SSR\_IRQ, and return address is restored to PC from {SPCH\_IRQ, SPCL\_IRQ}.

### TRQ EXCEPTION

When TE bit is set, TRQ interrupt happens and "JMP {6'h00, PM[000006h]}" instruction is executed right after each instruction is executed. TRQ interrupt uses the saved registers of IRQ(that is, {SPCH\_IRQ, SPCL\_IRQ} register pair and SSR\_IRQ) to save the return address and SR, respectively. PM bit is set. IE, TE bits are cleared.

### SWI EXCEPTION

When "SWI imm:6" instruction is executed, the return address is saved in the register A14, and the value of SR is saved in SSR\_SWI. Then the program sequence jumps to the address (imm:6 \* 4). PM bit is set. IE and TE bits are cleared. "SWI 0" and "SWI 1" are prohibited because the addresses are reserved for other interrupts. When RET\_SWI instruction is executed, SR is restored from SSR\_SWI, and the return address is restored to PC from A14.

### BREAK EXCEPTION

Break exception is reserved only for an in-circuit debugger. When a core input signal, *BKREQ*, is high, the CalmRISC16 core is halted or in the break mode, until *BKREQ* is deactivated. Another way to drive the CalmRISC16 core into the break mode is by executing a break instruction, *BREAK*. When *BREAK* is fetched, it is decoded and the CalmRISC16 core output signal *nBKACK* is generated. 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*. 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.

### NOTE

imm:6 is defined as 6-bit immediate number

### FIQ Sources

If the memory request tries to access non-existent memory area, FIQ is generated. In this case, if the FE bit in SR is 1, then FIQ routine is called and executed. But the FIQ is not used in the S3CC11B.

### INTERRUPT SOURCES (IRQ)

Level	Vector	Priority	Source	RESET (CLEAR)
RESET	000000H	1	Hardware Reset	-
FIQ	000002H	3	Fast Interrupt Request	H/W, S/W
IRQ	000002H	5	Timer 0 match/capture	H/W, S/W
			Timer 0 overflow	H/W, S/W
			Timer 1/A match	H/W, S/W
			Timer B match	H/W, S/W
			Basic Timer overflow	H/W, S/W
			CODEC INT	H/W, S/W
			SIO INT for external Codec	H/W, S/W
			Watch timer INT	H/W, S/W
			SIO INT	H/W, S/W
			Ext INT0	H/W, S/W
			Ext INT1	H/W, S/W
			Ext INT2	H/W, S/W
			Ext INT3	H/W, S/W
			Ext INT4	H/W, S/W
		Ext INT5		
		Ext INT6		
		Ext INT7		
TRQ	000006H	2	Trace Interrupt Request	H/W
SWI	000008H ~ 0000FEH	4	Software Interrupt	-

**NOTES:**

1. The IRQ vector has several interrupt sources. The priority of the sources is controlled by setting the IPRH/IPRL registers.
2. External interrupts are triggered by a rising or falling edge, depending on the corresponding control register setting. Ext INT4-Ext INT7 have no interrupt pending bit but have an enable bit.

Figure 4-1. Interrupt Sources (IRQ)

## INTERRUPT STRUCTURE

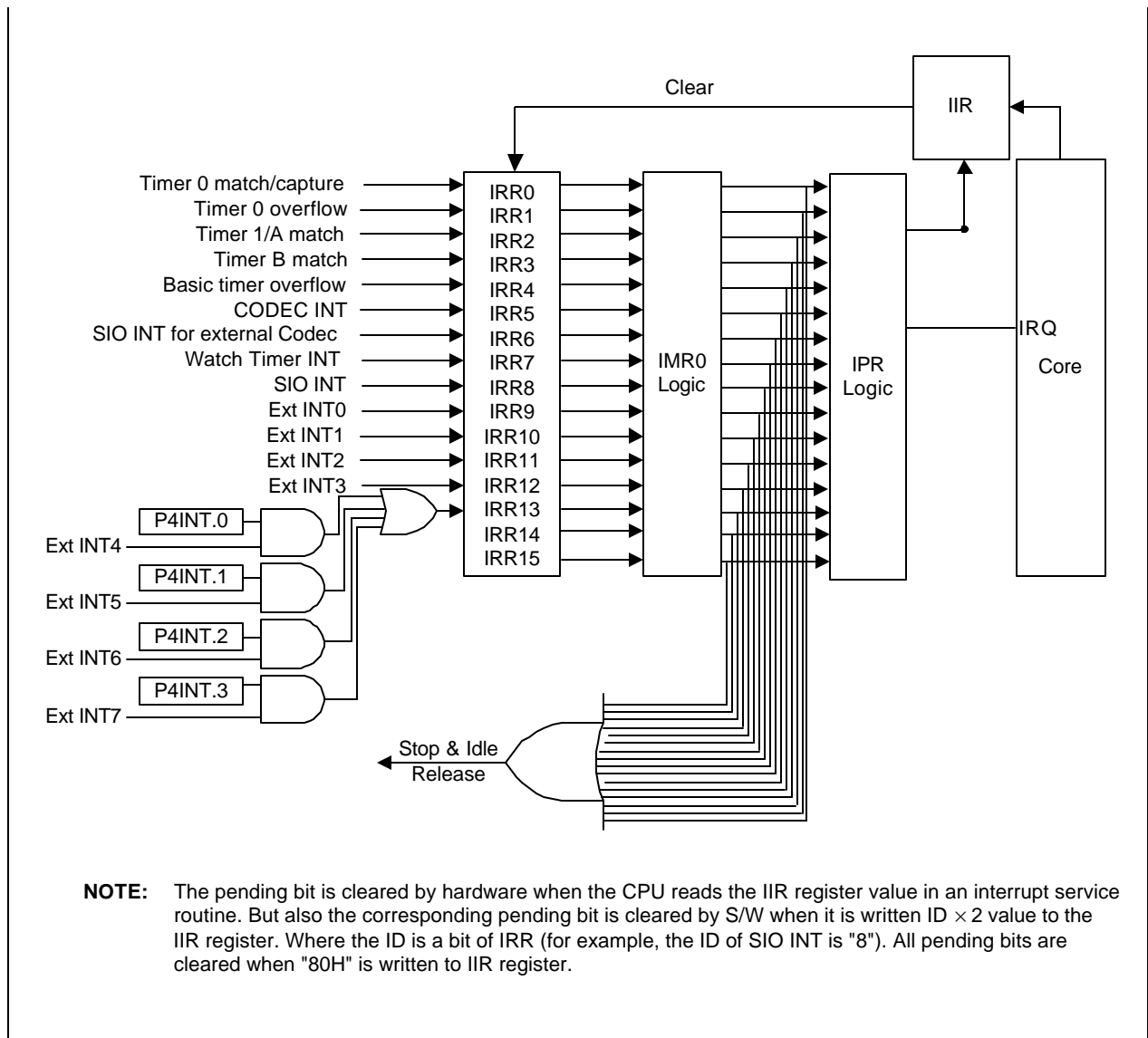


Figure 4-2. Interrupt Structure

## **INTERRUPT CONTROL REGISTER**

The calmRISC16 has 4-types registers, IRR, IMR, IIR, IPR.

## **INTERRUPT MASKING REGISTER**

Interrupt masking register is IMR. The role of IMR masks the pending interrupt. Although any interrupt source sets the interrupt pending register, the interrupt cannot be send to the core if the interrupt is masked.

0: mask (default value)

1: unmask

## **INTERRUPT PROIRITY REGISTER**

Interrupt priority register is IPR. The IPR register determine the serving order of interrupts when any interrupts of 21 sources occur simultaneously.

**IMRH — Interrupt Mask Register High****3F0006H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	–	–	0	0	0	0	0	0
<b>Read/Write</b>	–	–	R/W	R/W	R	R/W	R/W	R/W
<b>Addressing Mode</b>	Register addressing mode only							

**.7–.6****Bits 7–6**

0	Always logic "0"
---	------------------

**.5****External P4.4-P4.7(IRR.13) Interrupt Enable Bit**

0	Disable interrupt request
1	Enable interrupt request

**.4****External P0.3(IRR.12)**

0	Disable interrupt request
1	Enable interrupt request

**.3****External P0.2(IRR.11) Interrupt Enable Bit**

0	Disable interrupt request
1	Enable interrupt request

**.2****External P0.1(IRR.10) Interrupt Enable Bit**

0	Disable interrupt request
1	Enable interrupt request

**.1****External P0.0(IRR.9) Interrupt Enable Bit**

0	Disable interrupt request
1	Enable interrupt request

**.0****Serial I/O(IRR.8) Interrupt Enable Bit**

0	Disable interrupt request
1	Enable interrupt request

**IMRL — Interrupt Mask Register Low****3F0007H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W
<b>Addressing Mode</b>	Register addressing mode only							

**.7****Watch Timer(IRR.7) Interrupt Enable Bit**

0	Disable Interrupt request
1	Enable Interrupt request

**.6****SIO for External Codec(IRR.6) Interrupt Enable Bit**

0	Disable Interrupt request
1	Enable Interrupt request

**.5****CODEC(IRR.5) Interrupt Enable Bit**

0	Disable Interrupt request
1	Enable Interrupt request

**.4****Basic Timer Overflow(IRR.4) Interrupt Enable Bit**

0	Disable interrupt request
1	Enable interrupt request

**.3****Timer B Match(IRR.3) Interrupt Enable Bit**

0	Disable interrupt request
1	Enable interrupt request

**.2****Timer 1/A Match(IRR.2) Interrupt Enable Bit**

0	Disable interrupt request
1	Enable interrupt request

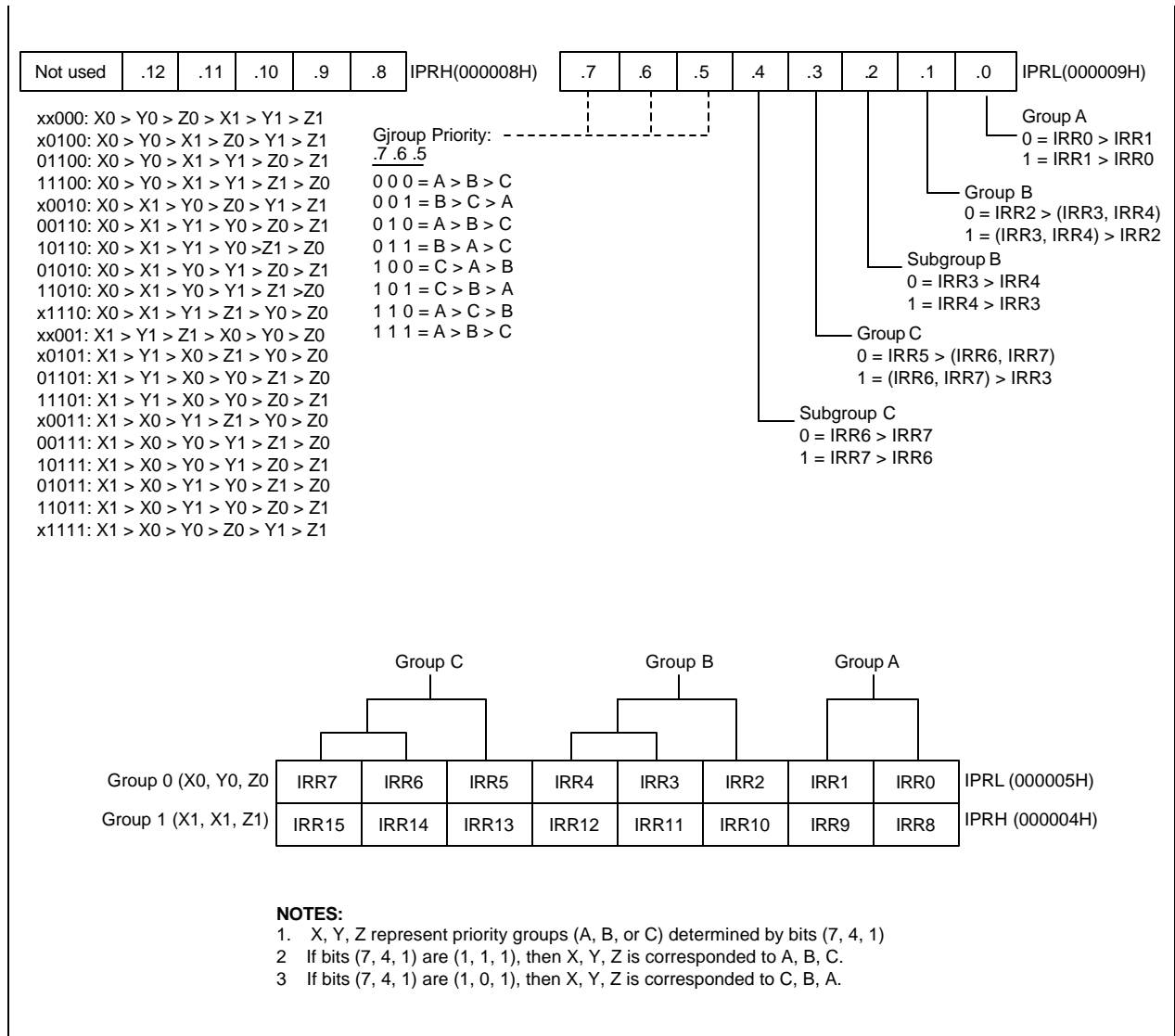
**.1****Timer 0 Overflow(IRR.1) Interrupt Enable Bit**

0	Disable interrupt request
1	Enable interrupt request

**.0****Timer 0 Match or Capture(IRR.0) Interrupt Enable Bit**

0	Disable interrupt request
1	Enable interrupt request

**INTERRUPT PRORITY REGISTERS (IPRH: 3F0008H, IPRL: 3F0009H)**



**Figure 4-3. Interrupt Priority Register (IPR)**

**INTERRUPT ID REGISTER**

Interrupt ID register (IIR) represents an "ID" of the interrupt to be serviced. When any interrupt of 21 sources requests a service from core, the core can select the target interrupt source by reading IIR.

The pending bit is cleared by hardware when the CPU reads the IIR register value in an interrupt service routine. But also the corresponding bit is cleared by S/W when it is written ID\*2 value to the IIR register.

Where the ID is a bit of IRR (For example, the ID of SIO INT is "8"). All pending bits are cleared when "80H" is written to IIR register.



**NOTES**

# 5

## MEMORY MAP

### OVERVIEW

To support the control of peripheral hardware, the address for peripheral control registers are memory-mapped to the area higher than 3F0000H. 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 S3CC11B/FC11B 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 control register is divided into three areas.

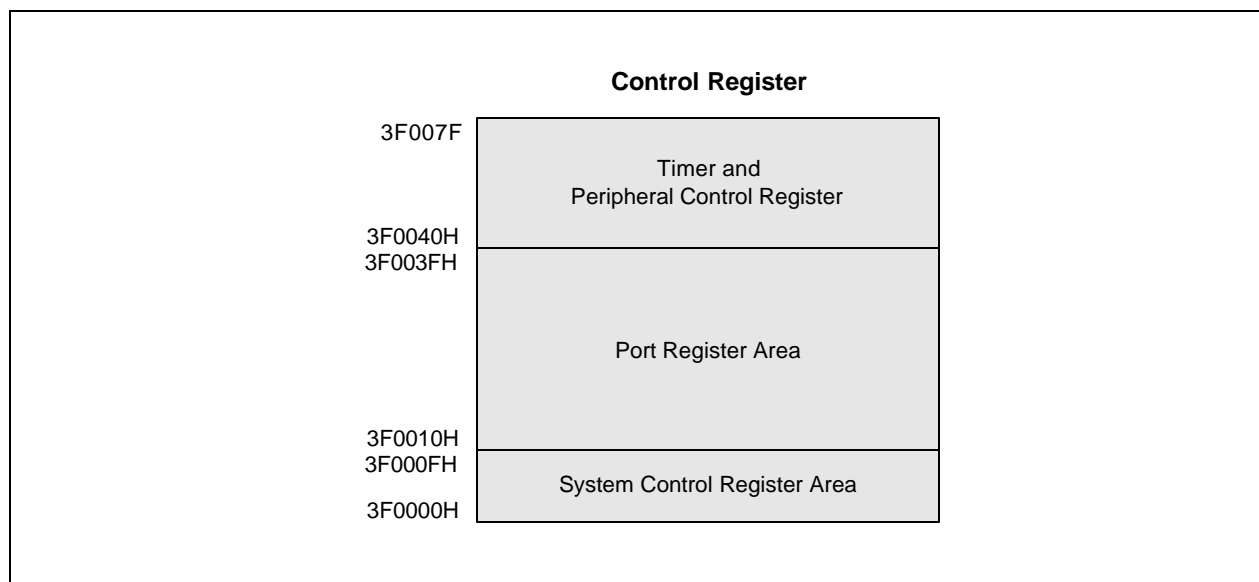


Figure 5-1. Memory Mapped IO Registers

Table 5-1. Registers

Register Name	Mnemonic	Hex	Reset	R/W
Locations 3F0000H, 3F0001H are not mapped				
Clock control register	CLKCON	3F0002H	00H	R/W
Oscillator control register	OSCCON	3F0003H	00H	R/W
Interrupt request register high	IRRH	3F0004H	00H	R/W
Interrupt request register low	IRRL	3F0005H	00H	R/W
Interrupt mask register high	IMRH	3F0006H	00H	R/W
Interrupt mask register low	IMRL	3F0007H	00H	R/W
Interrupt priority register high	IPRH	3F0008H	00H	R/W
Interrupt priority register low	IPRL	3F0009H	00H	R/W
Location 3F000AH is not mapped				
Interrupt ID register	IIR	3F000BH	–	R/W
Basic timer control register	BTCON	3F000CH	<b>70H</b>	R/W
Basic timer counter	BTCNT	3F000DH	00H	R
Watchdog timer enable register	WDTEN	3F000EH	00H	R/W
Location 3F000FH is not mapped				
Port 0 data register	P0	3F0010H	00H	R/W
Port 1 data register	P1	3F0011H	00H	R/W
Port 2 data register	P2	3F0012H	00H	R/W
Port 3 data register	P3	3F0013H	00H	R/W
Port 4 data register	P4	3F0014H	00H	R/W
Port 5 data register	P5	3F0015H	00H	R/W
Port 6 data register	P6	3F0016H	00H	R/W
Port 7 data register	P7	3F0017H	00H	R/W
Port 8 data register	P8	3F0018H	00H	R/W
Port 9 data register	P9	3F0019H	00H	R/W
Locations 3F001AH-3F001FH are not mapped				
Port 0 control register high	P0CONH	3F0020H	00H	R/W
Port 0 control register low	P0CONL	3F0021H	00H	R/W
Port 0 pull-up resistors enable register	P0PUR	3F0022H	00H	R/W
Port 0 interrupt state setting register	P0STA	3F0023H	00H	R/W
Port 1 control register	P1CON	3F0024H	00H	R/W
Locations 3F0025H-3F0027H are not mapped				
Port 2 control register high	P2CONH	3F0028H	00H	R/W
Port 2 control register low	P2CONL	3F0029H	00H	R/W
Port 2 pull-up resistors enable register	P2PUR	3F002AH	00H	R/W
Location 3F002BH is not mapped				

Table 5-1. Registers (Continued)

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
Port 3 control register high	P3CONH	3F002CH		00H	R/W
Port 3 control register low	P3CONL	3F002DH		00H	R/W
Port 3 pull-up resistors enable register	P3PUR	3F002EH		00H	R/W
Location 3F002FH is not mapped					
Port 4 control register high	P4CONH	3F0030H		00H	R/W
Port 4 control register low	P4CONL	3F0031H		00H	R/W
Port 4 pull-up resistors enable register	P4PUR	3F0032H		00H	R/W
Port 4 interrupt control register	P4INT	3F0033H		00H	R/W
Port 5 control register high	P5CONH	3F0034H		00H	R/W
Port 5 control register low	P5CONL	3F0035H		00H	R/W
Port 5 pull-up resistors enable register	P5PUR	3F0036H		00H	R/W
Location 3F0037H is not mapped					
Port 6 control register	P6CON	3F0038H		00H	R/W
Location 3F0039H is not mapped					
Port 7 control register	P7CON	3F003AH		00H	R/W
Location 3F003BH is not mapped					
Port 8 control register	P8CON	3F003CH		00H	R/W
Location 3F003DH is not mapped					
Port 9 control register	P9CON	3F003EH		00H	R/W
Location 3F003FH is not mapped					
Timer 0 counter register	T0CNT	3F0040H		00H	R
Timer 0 data register	T0DATA	3F0041H		FFH	R/W
Timer 0 control register	T0CON	3F0042H		00H	R/W
Location 3F0043H is not mapped					
Timer A counter register	TACNT	3F0044H		00H	R
Timer B counter register	TBCNT	3F0045H		00H	R
Timer A data register	TADATA	3F0046H		FFH	R/W
Timer B data register	TBDATA	3F0047H		FFH	R/W
Timer 1/A control register	TACON	3F0048H		00H	R/W
Timer B control register	TBCON	3F0049H		00H	R/W
Locations 3F004AH-3F004BH are not mapped					
SIO data register high byte for external codec	CSIODATAH	3F004CH		00H	R/W
SIO control register low byte for external codec	CSIODATAL	3F004DH		00H	R/W
SIO control register for external codec	CSIOCON	3F004EH		00H	R/W
Locations 3F004FH-3F0050H are not mapped					
10-bit A/D converter control register	ADCON10	3F0051H		00H	R/W
10-bit A/D converter data register high	ADDATAH10	3F0052H		–	R/W
10-bit A/D converter data register low	ADDATAL10	3F0053H		–	R/W

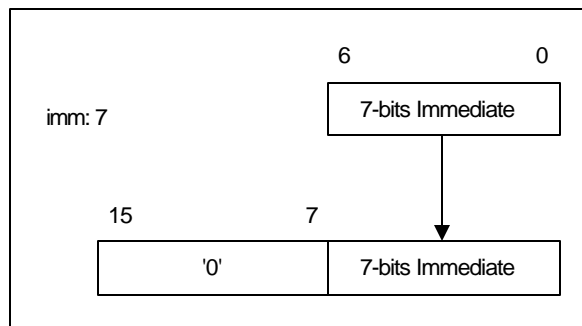
Table 5-1. Registers (Continued)

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
SmartMedia control register	SMCON	3F0058H		x0H	R/W
ECC count register	ECCNT	3F0059H		00H	R/W
ECC data register high	ECCH	3F005AH		00H	R/W
ECC data register low	ECCL	3F005BH		00H	R/W
ECC data register extension	ECCX	3F005CH		00H	R/W
ECC clear register	ECCCLR	3F005DH		00H	W
ECC result data register high	ECCRSTH	3F005EH		00H	R/W
ECC result data register low	ECCRSTL	3F005FH		00H	R/W
Locations 3F0060H-3F0063H are not mapped					
Codec control register	CDCON	3F0064H		00H	R/W
Locations 3F0065H-3F0067H are not mapped					
A/D data register high	ADDATAH	3F0068H		–	R
A/D data register low	ADDATAL	3F0069H		–	R
D/A data register high	DADATAH	3F006AH		00H	R/W
D/A data register low	DADATAL	3F006BH		00H	R/W
SIO data register	SIODATA	3F006CH		00H	R/W
SIO pre-scale register	SIOPS	3F006DH		00H	R/W
SIO control register	SIOCON	3F006EH		00H	R/W
Location 3F006FH is not mapped					
Watch timer control register	WTCON	3F0070H		00H	R/W
Location 3F0071H is not mapped					
LCD control register	LCON	3F0072H		00H	R/W
LCD mode register	LMOD	3F0073H		00H	R/W
Battery level detector control register	BLDCON	3F0074H		00H	R/W
Location 3F0075H is not mapped					
PLL control register	PLLCON	3F0076H		00H	R/W
PLL data register	PLLDATA	3F0077H		00H	R/W
Flash memory control register	FMCON	3F0078H		00H	R/W
Locations 3F0079H-3F007FH are not mapped					

# 6 INSTRUCTION SET

## ALU INSTRUCTIONS

In operations between a 16-bit general register and an immediate value, the immediate value is zero-extended to 16-bit. The following figure shows an example of 7-bit immediate numbers.



In operations between a 22-bit register and an immediate value, the immediate value is zero-extended to 22-bit. In operations between a 22-bit register and a 16-bit register, the 16-bit register is zero-extended to 22-bit. The overflow flag in a 16-bit arithmetic operation is saved to V flag in SR register. ALU instructions are classified into 3 classes as follows.

- ALUop Register, Immediate
- ALUop Register, Register
- ALUop Register

**ALUOP REGISTER, IMMEDIATE****ADD/ADC/SUB/SBC/AND/OR/XOR/TST/CMP/CMPU Rn, #imm:16**

The instructions perform an ALU operation of which source operands are a 16-bit general register Rn and a 16-bit immediate value. In the instructions TST/CMP/CMPU, only T flag is updated accordingly as the result. In the instructions ADD/ADC/SUB/SBC, the value of T flag is the carry flag of the operations, and the value of V flag indicates whether overflow or underflow occurs. In the instructions AND/OR/XOR/TST, the value of T flag indicates whether the result is zero (T=1). "CMP {GT|GE|EQ}, Rn, #imm:16" instructions are for signed comparison operations (GT for greater than, GE for greater than or equal to and EQ for equal to), and "CMPU {GT|GE}, Rn, #imm:16" instructions are for unsigned comparison operations.

**NOTE:** imm:16 is defined as a 16-bit immediate number

**ADD/SUB An, #imm:16**

The immediate value is zero-extended to 22-bit value. No flag update occurs.

**ADD/SUB Rn, #imm:7**

The immediate value is zero-extended to 16-bit value. T flag is updated to the carry of the operation. V flag is updated.

**AND/OR/XOR/TST R0, #imm:8**

The immediate value is zero-extended to 16-bit value. T flag indicates whether the lower 8-bit of the logical operation result is zero.

**CMP EQ, Rn, #imm:8**

The immediate value is zero-extended to 16-bit value. Rn is restricted to R0 to R7. T flag is updated as the result of the instruction.

**CMP GE, Rn, #imm:6**

The immediate value is zero-extended to 16-bit value. The instruction is for signed compare. T flag is updated as the result of the instruction.

**ADD/SUB An, #imm:5**

The immediate value is zero-extended to 22-bit value. No flag is updated.

**ALUOP REGISTER, REGISTER****ADD/SUB/ADC/SBC/AND/OR/XOR/TST/CMP/CMPU Rn, Ri**

The instructions perform an ALU operation of which source operands are a pair of 16-bit general registers. In the instructions TST/CMP/CMPU, only T flag is updated as the result. In the instructions ADD/ADC/SUB/SBC, the value of T flag is the carry of the operations, and the value of V flag indicates whether overflow or underflow occurs. In the instructions AND/OR/XOR/TST, the value of T flag indicates whether the result is zero. "CMP {GT|GE|EQ}, Rn, Ri" instructions are for signed comparison, and "CMPU {GT|GE}, Rn, Ri" instructions are for unsigned comparison.

**ADD/SUB An, Ri**

16-bit general register Ri is zero-extended to 22-bit value. The result is saved in the 22-bit register An. No flag update occurs.

**CMP EQ, An, Ai**

The instruction compares two 22-bit registers.

**MUL {SS|SU|US|UU}, Rn, Ri**

The general registers Rn and Ri can be one of R0 to R7. The instruction multiplies the lower byte of Rn and the lower byte of Ri, and the 16-bit result is saved in Rn. The optional field, SS, SU, US, and UU, indicates whether the source operands are signed value or unsigned value. The first letter of the two letter qualifiers corresponds to Rn, and the second corresponds to Ri. For example, in the instruction "MUL SU, R0, R1", the 8-bit signed value in the lower byte of R0 and the 8-bit unsigned value in the lower byte of R1 are multiplied, and the 16-bit result is saved in R0.

**RR/RL/RRC/SR/SRA/SLB/SRB/DT/INCC/DECC/COM/COM2/COMC/EXT Rn**

For "DT Rn"(Decrement and Test) and "COM Rn"(Complement) instructions, T flag indicates whether the result is zero. In the instruction of "EXT Rn"(Sign Extend), no flag update occurs. In all other instructions, carry-out of the operation is transferred to T flag. In the instruction of DT, INCC, and DECC, V flag indicates whether overflow or underflow occurs.



## LOAD INSTRUCTIONS

“Load instructions” move data from register/memory/immediate to register/memory. When the destination is a memory location, only general registers and extension registers can be the source. We can classify “Load instructions” into the following 4 classes.

- LD Register, Register
- LD Register, Immediate
- LD Data Memory, Register / LD Register, Data Memory
- LD Register, Program Memory

### LD REGISTER, REGISTER

#### LD Rn, Ri / LD An, Ai

The instructions move 16-bit or 22-bit data from the source register to the destination register. When the destination register is R6/R7, the zero flag Z0/Z1 is updated. In all other cases, no flag update occurs.

#### LD Rn, Ei / LD En, Ri

In the instruction “LD Rn, Ei”, the 6-bit data in Ei is zero-extended to 16-bit data, and then transferred to Rn. When the destination register is R6/R7, the zero flag Z0/Z1 is updated. In the instruction “LD En, Ri”, least significant 6 bits of Ri are transferred to En. Rn/Ri is one of the registers from R0 to R7.

### LD R0, SPR / LD SPR, R0

#### SPR : SR, SPCL\_FIQ, SPCH\_FIQ, SSR\_FIQ, SPCL\_IRQ, SPCH\_IRQ, SSR\_IRQ, SSR\_SWI

The instructions transfer data between SPR (Special Purpose Registers) and R0. No flag update occurs except the case that the destination register is SR.

### LD An, PC

The instruction moves the value of (PC+4) to An.

**LD REGISTER, DATA MEMORY / LD DATA MEMORY, REGISTER****LDW Rn, @[SP+edisp:9] / LDW @[SP+edisp:9], Rn**

The instructions transfer 16-bit data between a general register Rn and the memory location at the address of (SP+edisp:9). Note SP is another name of A15. edisp:9 is an even positive displacement from 0 to 510. edisp:9 is encoded into an 8-bit displacement value in the instruction map because the LSB is always 0. When the address is calculated, the 8-bit displacement field is shifted to the left by one bit, and then the result is added to the value of SP. Even if the address might be specified as odd in assembly mnemonic, the LSB of the address should be truncated to zero for word alignment.

**LDW Rn, @[Ai+edisp:5] / LDW @[Ai+edisp:5], Rn**

The instructions transfer 16-bit data between a general register Rn and the memory location at the address of (Ai+edisp:5). edisp:5 is an even positive displacement from 0 to 30. edisp:5 is encoded into an 4-bit displacement value in the instruction map because the LSB is always 0. When the address is calculated, the 4-bit displacement field is shifted to the left by one bit, and then the result is added to the value of Ai. Even if the address might be specified as odd in assembly mnemonic, the LSB of the address should be truncated to zero for word alignment.

**LDW Rn, @[Ai+disp:16] / LDW @[Ai+disp:16], Rn**

The instructions transfer 16-bit data between a general register Rn and the memory location at the address of (Ai+disp:16). disp:16 is an positive displacement from 0 to FFFFh. If the address is odd, the LSB of the address is set to zero for word alignment.

**LDW Rn, @[Ai+Rj] / LDW @[Ai+Rj], Rn**

The instructions transfer 16-bit data between a general register Rn and the memory location at the address of (Ai+Rj). The value of Rj is zero-extended to 22-bit value. If the address is odd, the LSB of the address is set to zero for word alignment.

**LDW An, @[Ai+edisp:5] / LDW @[Ai+edisp:5], An**

The instructions transfer 22-bit data between an address register An and the memory location at the address of (Ai+edisp:5). edisp:5 is an even positive displacement from 0 to 30. edisp:5 is encoded into an 4-bit displacement value in the instruction map because the LSB is always 0. When the address is calculated, the 4-bit displacement field is shifted to the left by one bit, and then the result is added to the value of Ai. Even if the address might be specified as odd in assembly mnemonic, the LSB of the address should be truncated to zero for word alignment.

**LDW An, @[Ai+disp:16] / LDW @[Ai+disp:16], An**

The instructions transfer 22-bit data between an address register An and the memory location at the address of (Ai+disp:16). disp:16 is an positive displacement from 0 to FFFFh. If the address is odd, the LSB of the address is set to zero for word alignment.

**LDW An, @[Ai+Rj] / LDW @[Ai+Rj], An**

The instructions transfer 22-bit data between an address register An and the memory location at the address of (Ai+Rj). The value of Rj is zero-extended to 22-bit value. If the address is odd, the LSB of the address is set to zero for word alignment.

**PUSH Rn/PUSH Rn, Rm/PUSH An/ PUSH An, Am**

The instruction "PUSH Rn" transfers 16-bit data from the register Rn to the memory location at the address of SP, and then decrements the value of SP by 2. The register Rn should not be R15. The operation of "PUSH R15" is undefined. The instruction "PUSH Rn, Rm" pushes Rn and then Rm. The registers Rn and Rm should not be the same. The registers Rn and Rm should not be R15. The instruction "PUSH An" pushes Rn and then En. When the extension register En is pushed, the value of En is zero-extended to 16-bit data. The register An should not be A15. The instruction "PUSH An, Am" pushes An and then Am. The registers An and Am should not be the same

**POP Rn/POP Rn, Rm/POP An/ POP An, Am**

The instruction "POP Rn" increments the value of SP by 2, and then transfers 16-bit data to the register Rn from the memory location at the address of SP. The register Rn should not be R15. The operation of "POP R15" is undefined. The instruction "POP Rn, Rm" pops Rn and then Rm. The registers Rn and Rm should not be the same. The registers Rn and Rm should not be R15. The instruction "POP An" pops En and then Rn. When the extension register En is popped, the least significant 6 bits are transferred to En. The register An should not be A15. The instruction "POP An, Am" pops An and then Am. The registers An and Am should not be the same

**LDB Rn, @[Ai+disp:4] / LDB @[Ai+disp:4], Rn**

The instructions transfer 8-bit data between the general register Rn and the memory location at the address of (Ai+disp:4). disp:4 is a positive displacement from 0 to 15. The general register Rn is one R0 to R7. In the instruction "LDB Rn, @[Ai+disp:4]", the 8-bit data is zero-extended to 16-bit data, and then written into Rn. In the instruction "LDB @[Ai+disp:8], Rn", the least significant byte of Rn is transferred to the memory.

**LDB Rn, @[Ai+disp:16] / LDB @[Ai+disp:16], Rn**

The instructions transfer 8-bit data between the general register Rn and the memory location at the address of (Ai+disp:16). disp:16 is a positive displacement from 0 to FFFFh. The general register Rn is one of R0 to R7. In the instruction "LDB Rn, @[Ai+disp:16]", the 8-bit data is zero-extended to 16-bit data, and then written into Rn. In the instruction "LDB @[Ai+disp:16], Rn", the least significant byte of Rn is transferred to the memory.

**LDB R0, @[A8+disp:8] / LDB @[A8+disp:8], Rn**

The instructions transfer 8-bit data between the general register R0 and the memory location at the address of (A8+disp:8). disp:8 is a positive displacement from 0 to 255. In the instruction "LDB R0, @[A8+disp:8]", the 8-bit data is zero-extended to 16-bit data, and then written into R0. In the instruction "LDB @[A8+disp:8], R0", the least significant byte of R0 is transferred to the memory.

**LDB Rn, @[Ai+Rj] / LDB @[Ai+Rj], Rn**

The instructions transfer 8-bit data between the general register Rn and the memory location at the address of (Ai+Rj). The value of Rj is zero-extended to 22-bit value. The general register Rn is one of the 8 registers from R0 to R7. In the instruction "LDB Rn, @[Ai+Rj]", the 8-bit data is zero-extended to 16-bit data, and then written into Rn. In the instruction "LDB @[Ai+Rj], Rn", the least significant byte of Rn is transferred to the memory.

**LD REGISTER, PROGRAM MEMORY****LDC Rn, @Ai**

The instruction transfers 16-bit data to Rn from program memory at the address of Ai.

**LD REGISTER, # IMMEDIATE****LD Rn, #imm:8 / LD Rn, #imm:16 / LD An, #imm:22**

The instructions move an immediate data to a register. In the instruction “LD Rn, #imm:8”, the immediate value is zero-extended to 16-bit value.

## BRANCH INSTRUCTIONS

CalmRISC16 has 2 classes of branch instructions: with a delay slot and without a delay slot. If a delay slot is filled with a useful instruction (or an instruction which is not NOP), then the performance degradation due to the control dependency can be minimized. However, if the delay slot cannot be used, then it should be NOP instruction, which can increase the program code size. In this case, the corresponding branch instruction without a delay slot can be used to avoid using NOP.

Some instructions are not permitted to be in the delay slot. The prohibited instructions are as follows.

- All 2-word instructions
- All branch and jump instructions including SWI, RETD, RET\_SWI, RET\_IRQ, RET
- BREAK instructions

When a prohibited instruction is in the delay slot, the operation of CalmRISC16 is undefined or unpredictable.

### **BSRD eoffset:13**

In the instruction, called branch subroutine with a delay slot, the value (PC + 4) is saved into A14 register, the instruction in the delay slot is executed, and then the program sequence is moved to (PC + 2 + eoffset:13), where PC is the address of the instruction "BSRD eoffset:13". The immediate value eoffset:13 is sign-extended to 22-bit and then added to (PC+2). In general, the 13-bit offset field appears as a label in assembly programs. If the instruction in the delay slot reads the value of A14, the value (PC+4) is read. The even offset eoffset:13 is encoded to 12bit signed offset in instruction map by dropping the least significant bit.

### **BRA/BRAD/BRT/BRTD/BRF/BRFD eoffset:11**

In the branch instructions, the target address is (PC + 2 + eoffset:11). The immediate value eoffset:11 is sign-extended to 22-bit and then added to (PC+2). The "D" in the mnemonic stands for a delay slot. In general, the 11-bit offset field appears as a label in assembly programs. BRA and BRAD instructions always branch to the target address. BRT and BRTD instructions branch to the target address if T flag is set. BRF and BRFD instructions branch to the target address if T flag is cleared. BRAD/BRTD/BRFD instructions are delay slot branch instructions, therefore the instruction in the delay slot is executed before the branch to the target address or the branch decision is made. The even offset eoffset:11 is encoded to 10-bit signed offset in instruction map by dropping the least significant bit.

### **BRA/BRAD EC:2, eoffset:8**

In the branch instructions, the target address is (PC + 2 + eoffset:8). The immediate value eoffset:8 is sign-extended to 22-bit and then added to (PC+2). The EC:2 field indicates one of the 4 external conditions from EC0 to EC3 (input pin signals to CalmRISC16). When the external condition corresponding to EC:2 is set, the program branches to the target address. BRAD has a delay slot. The even offset eoffset:8 is encoded to 7-bit signed offset in instruction map by dropping the least significant bit.

**BNZD R6/R7, eoffset:8**

In the branch instruction, the target address is  $(PC + 2 + \text{eoffset:8})$ . The immediate value `eoffset:8` is sign-extended to 22-bit and then added to  $(PC+2)$ . “BNZD R6, `eoffset:8`” instruction branches to the target address if Z0 flag is cleared. “BNZD R7, `eoffset:8`” instruction branches if Z1 flag is cleared. Before the branch operation, the instruction decrements R6/R7, updates Z0/Z1 flag according to the decrement result, and then executes the instruction in the delay slot. The instruction is used to manage loop counter with just one cycle overhead. In the end of the loop, the value of R6/R7 is  $-1$ . When the instruction in the delay slot read the Z0/Z1 flag, the result after the decrement is read. The even offset `eoffset:8` is encoded to 7-bit signed offset in instruction map by dropping the least significant bit.

**JMP/JPT/JPF/JSR addr:22**

The target address of the instructions is `addr:22`. JMP always branches to the target address. JPT branches to the target address if the T flag is set. JPF branches if the T flag is cleared. JSR always branches to the target address with saving the return address  $(PC+4)$  into A14. The instructions are 2 word instructions.

**JMP/JPT/JPF/JSR Ai**

The target address of the instructions is the value of Ai. JMP always branches to the target address. JPT branches to the target address if the T flag is set. JPF branches if the T flag is cleared. JSR always branches to the target address with saving the return address  $(PC+2)$  into A14.

**SWI #imm:6/ RET\_SWI/RET\_IRQ/RET\_FIQ**

refer to the section for interrupts.

**RETD**

The instruction branches to the address in A14 after the execution of the instruction in the delay slot. When there is no useful instruction adequate to the delay slot, “JMP A14” can be used instead of “RETD”.

## BIT OPERATION

The bit operations manipulate a bit in SR register or in a memory location.

### **BITR/BITS/BITC/BITT @[A8+R1], #imm:3**

The source as well as the destination is the 8-bit data in the data memory at the address (A8 + R1). The #imm:3 field chooses a bit position among the 8 bits. BITR resets the bit #imm:3 of the source, and then writes the result to the destination, the same memory location. BITS sets the bit #imm:3 of the source, and then writes the result to the destination. BITC complements the bit #imm:3 of the source, and then writes the result to the destination. BITT does not write any data to the destination. T flag indicates whether the bit #imm:3 of the source is zero. In other words, when the bit #imm:3 of the source is zero, T flag is set. BITR and BITS can be used to implement a semaphore mechanism or lock acquisition/release.

### **CLRSR/SETSR/TSTSR bit**

**bit : FE, IE, TE, Z0, Z1, V, PM**

CLRSR instruction clears the corresponding bit of SR. SETSR instruction sets the corresponding bit of SR. TSTSR tests whether the corresponding bit is zero, and stores the result in T flag. For example, when IE flag is zero, "TSTSR IE" instruction sets the T flag. We can clear the T flag by the instruction "CMP GT, R0, R0". We can set the T flag by the instruction "CMP EQ, R0, R0".

## MISCELLANEOUS INSTRUCTIONS

### **SYS #imm:5**

The instruction activates the output port nSYSID. The #imm:5 is transferred to outside on DA[4:0]. The most significant 17 bits remain unchanged. The instruction is for system command to outside such as power down modes.

### **COP #imm:13**

The instruction activates the output port nCOPID. The #imm:13 is transferred to outside on COPIR[12:0]. The instruction is used to transfer instruction to coprocessor. The #imm:13 may be from 200h to 1FFFh.

### **CLD Rn, #imm:5 / CLD #imm:5, Rn**

The instruction activates the output port nCOPID, nCLDID, and CLDWR. The least significant 13 bits of the instruction is transferred to outside on COPIR[12:0]. The #imm:5 is transferred to outside on DA[4:0]. The instructions move 16-bit data between Rn and a coprocessor register implied by the #imm:5 field. CLDWR signal indicates whether the data movement is from CalmRISC16 to coprocessor. The register Rn is one 8 registers from R0 to R7.

### **NOP**

No operation.

### **BREAK**

The software break instruction activates nBRK signal, and holds PA for one cycle. It's for debugging operation.



## CALMRISC16 INSTRUCTION SET MAP

Table 6-1. CalmRISC16 Instruction Set Map

	15					8 7				0			
ADD Rn, #imm:7	0	0	0	0	Rn	0	Imm:7						
SUB Rn, #imm:7	0	0	0	0	Rn	1	Imm:7						
LD Rn, #imm:8	0	0	0	1	Rn	Imm:8							
LDW Rn, @[SP + edisp:9]	0	0	1	0	Rn	Edisp:9							
LDW @[SP + edisp:9], Ri	0	0	1	1	Ri	Edisp:9							
LDW Rn, @[Ai + edisp:5]	0	1	0	0	Rn	0	Ai	Edisp:5					
LDW Rn, @[Ai + Rj]	0	1	0	0	Rn	1	Ai	Rj					
LDW @[An + edisp:5], Ri	0	1	0	1	Ri	0	An	Edisp:5					
LDW @[An + Rm], Ri	0	1	0	1	Ri	1	An	Rm					
LDB Dn, @[Ai + disp:4]	0	1	1	0	0	Dn	0	Ai	Disp:4				
LDB Dn, @[Ai + Rj]	0	1	1	0	0	Dn	1	Ai	Rj				
LDW An, @[Ai + disp:4]	0	1	1	0	1	An	0	Ai	Disp:4				
LDW An, @[Ai + Rj]	0	1	1	0	1	An	1	Ai	Rj				
LDB @[An + disp:4], Di	0	1	1	1	0	Di	0	An	Disp:4				
LDB @[An + Rm], Di	0	1	1	1	0	Di	1	An	Rm				
LDW @[An + disp:4], Ai	0	1	1	1	1	Ai	0	An	Disp:4				
LDW @[An + Rm], Ai	0	1	1	1	1	Ai	1	An	Rm				
ADD Rn, Ri	1	0	0	0	Rn	0	0	0	0	Ri			
SUB Rn, Ri	1	0	0	0	Rn	0	0	0	1	Ri			
ADC Rn, Ri	1	0	0	0	Rn	0	0	1	0	Ri			
SBC Rn, Ri	1	0	0	0	Rn	0	0	1	1	Ri			
AND Rn, Ri	1	0	0	0	Rn	0	1	0	0	Ri			
OR Rn, Ri	1	0	0	0	Rn	0	1	0	1	Ri			
XOR Rn, Ri	1	0	0	0	Rn	0	1	1	0	Ri			
TST Rn, Ri	1	0	0	0	Rn	0	1	1	1	Ri			
CMP GE, Rn, Ri	1	0	0	0	Rn	1	0	0	0	Ri			
CMP GT, Rn, Ri	1	0	0	0	Rn	1	0	0	1	Ri			
CMPU GE, Rn, Ri	1	0	0	0	Rn	1	0	1	0	Ri			
CMPU GT, Rn, Ri	1	0	0	0	Rn	1	0	1	1	Ri			
CMP EQ, Rn, Ri	1	0	0	0	Rn	1	1	0	0	Ri			
LD Rn, Ri	1	0	0	0	Rn	1	1	0	1	Ri			
RR Rn	1	0	0	0	0	0	0	0	1	1	1	0	Rn
RL Rn	1	0	0	0	0	0	0	1	1	1	1	0	Rn

Table 6-1. CalmRISC16 Instruction Set Map (Continued)

	15					8 7					0			
RRC Rn	1	0	0	0	0	0	1	0	1	1	1	0	Rn	
SRB Rn	1	0	0	0	0	0	1	1	1	1	1	0	Rn	
SR Rn	1	0	0	0	0	1	0	0	1	1	1	0	Rn	
SRA Rn	1	0	0	0	0	1	0	1	1	1	1	0	Rn	
JPF Ai	1	0	0	0	0	1	1	0	1	1	1	0	0	Ai
JPT Ai	1	0	0	0	0	1	1	0	1	1	1	0	1	Ai
JMP Ai	1	0	0	0	0	1	1	1	1	1	1	0	0	Ai
JSR Ai	1	0	0	0	0	1	1	1	1	1	1	0	1	Ai
SLB Rn	1	0	0	0	1	0	0	0	1	1	1	0	Rn	
DT Rn	1	0	0	0	1	0	0	1	1	1	1	0	Rn	
INCC Rn	1	0	0	0	1	0	1	0	1	1	1	0	Rn	
DECC Rn	1	0	0	0	1	0	1	1	1	1	1	0	Rn	
COM Rn	1	0	0	0	1	1	0	0	1	1	1	0	Rn	
COM2 Rn	1	0	0	0	1	1	0	1	1	1	1	0	Rn	
COMC Rn	1	0	0	0	1	1	1	0	1	1	1	0	Rn	
EXT Rn	1	0	0	0	1	1	1	1	1	1	1	0	Rn	
ADD Rn, #imm:16	1	0	0	0	0	0	0	0	1	1	1	1	Rn	
ADD An, #imm:16	1	0	0	0	0	0	0	1	1	1	1	1	0	An
SUB An, #imm:16	1	0	0	0	0	0	0	1	1	1	1	1	1	An
ADC Rn, #imm:16	1	0	0	0	0	0	1	0	1	1	1	1	Rn	
SBC Rn, #imm:16	1	0	0	0	0	0	1	1	1	1	1	1	Rn	
AND Rn, #imm:16	1	0	0	0	0	1	0	0	1	1	1	1	Rn	
OR Rn, #imm:16	1	0	0	0	0	1	0	1	1	1	1	1	Rn	
XOR Rn, #imm:16	1	0	0	0	0	1	1	0	1	1	1	1	Rn	
TST Rn, #imm:16	1	0	0	0	0	1	1	1	1	1	1	1	Rn	
CMP GE, Rn, #imm:16	1	0	0	0	1	0	0	0	1	1	1	1	Rn	
CMP GT, Rn, #imm:16	1	0	0	0	1	0	0	1	1	1	1	1	Rn	
CMPU GE, Rn, #imm:16	1	0	0	0	1	0	1	0	1	1	1	1	Rn	
CMPU GT, Rn, #imm:16	1	0	0	0	1	0	1	1	1	1	1	1	Rn	
CMP EQ, Rn, #imm:16	1	0	0	0	1	1	0	0	1	1	1	1	Rn	
LD Rn, #imm:16	1	0	0	0	1	1	0	1	1	1	1	1	Rn	
Reserved	1	0	0	0	1	1	1		1	1	1	1		
CMP EQ, Dn, #imm:8	1	0	0	1	0	Dn			Imm:8					
AND R0, #imm:8	1	0	0	1	1	0	0	0	Imm:8					

Table 6-1. CalmRISC16 Instruction Set Map (Continued)

	15							8 7							0		
OR R0, #imm:8	1	0	0	1	1	0	0	1	Imm:8								
XOR R0, #imm:8	1	0	0	1	1	0	1	0	Imm:8								
TST R0, #imm:8	1	0	0	1	1	0	1	1	Imm:8								
LDB R0, @[A8+ disp:8]	1	0	0	1	1	1	0	0	Disp:8								
LDB @[A8+ disp:8],R0	1	0	0	1	1	1	0	1	Disp:8								
BITR @[A8+R1], bs:3	1	0	0	1	1	1	1	0	0	0	0	0	0	Bs:3			
BITS @[A8+R1], bs:3	1	0	0	1	1	1	1	0	0	0	0	0	1	Bs:3			
BITC @[A8+R1], bs:3	1	0	0	1	1	1	1	0	0	0	0	1	0	Bs:3			
BITT @[A8+R1], bs:3	1	0	0	1	1	1	1	0	0	0	0	1	1	Bs:3			
SYS #imm:5	1	0	0	1	1	1	1	0	0	0	1	Imm:5					
SWI #imm:6	1	0	0	1	1	1	1	0	0	1	Imm:6						
CLRSR bs:3	1	0	0	1	1	1	1	0	1	0	0	0	0	Bs:3			
SETSR bs:3	1	0	0	1	1	1	1	0	1	0	0	0	1	Bs:3			
TSTSR bs:3	1	0	0	1	1	1	1	0	1	0	0	1	0	Bs:3			
NOP	1	0	0	1	1	1	1	0	1	0	0	1	1	0	0	0	
BREAK	1	0	0	1	1	1	1	0	1	0	0	1	1	0	0	1	
LD R0, SR	1	0	0	1	1	1	1	0	1	0	0	1	1	0	1	0	
LD SR, R0	1	0	0	1	1	1	1	0	1	0	0	1	1	0	1	1	
RET_FIQ	1	0	0	1	1	1	1	0	1	0	0	1	1	1	0	0	
RET_IRQ	1	0	0	1	1	1	1	0	1	0	0	1	1	1	0	1	
RET_SWI	1	0	0	1	1	1	1	0	1	0	0	1	1	1	1	0	
RETD	1	0	0	1	1	1	1	0	1	0	0	1	1	1	1	1	
LD R0, SPCL_FIQ	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	0	
LD R0, SPCH_FIQ	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	1	
LD R0, SSR_FIQ	1	0	0	1	1	1	1	0	1	0	1	0	0	0	1	0	
Reserved	1	0	0	1	1	1	1	0	1	0	1	0	0	0	1	1	
LD R0, SPCL_IRQ	1	0	0	1	1	1	1	0	1	0	1	0	0	1	0	0	
LD R0, SPCH_IRQ	1	0	0	1	1	1	1	0	1	0	1	0	0	1	0	1	
LD R0, SSR_IRQ	1	0	0	1	1	1	1	0	1	0	1	0	0	1	1	0	
Reserved	1	0	0	1	1	1	1	0	1	0	1	0	0	1	1	1	
Reserved	1	0	0	1	1	1	1	0	1	0	1	0	1	0	0		
LD R0, SSR_SWI	1	0	0	1	1	1	1	0	1	0	1	0	1	0	1	0	
Reserved	1	0	0	1	1	1	1	0	1	0	1	0	1	0	1	1	
Reserved	1	0	0	1	1	1	1	0	1	0	1	0	1	1			

Table 6-1. CalmRISC16 Instruction Set Map (Continued)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LD SPCL_FIQ, R0	1	0	0	1	1	1	1	0	1	0	1	1	0	0	0	0
LD SPCH_FIQ, R0	1	0	0	1	1	1	1	0	1	0	1	1	0	0	0	1
LD SSR_FIQ, R0	1	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
Reserved	1	0	0	1	1	1	1	0	1	0	1	1	0	0	1	1
LD SPCL_IRQ, R0	1	0	0	1	1	1	1	0	1	0	1	1	0	1	0	0
LD SPCH_IRQ, R0	1	0	0	1	1	1	1	0	1	0	1	1	0	1	0	1
LD SSR_IRQ, R0	1	0	0	1	1	1	1	0	1	0	1	1	0	1	1	0
Reserved	1	0	0	1	1	1	1	0	1	0	1	1	0	1	1	1
Reserved	1	0	0	1	1	1	1	0	1	0	1	1	1	0	0	
LD SSR_SWI, R0	1	0	0	1	1	1	1	0	1	0	1	1	1	0	1	0
Reserved	1	0	0	1	1	1	1	0	1	0	1	1	1	0	1	1
Reserved	1	0	0	1	1	1	1	0	1	0	1	1	1	1		
Reserved	1	0	0	1	1	1	1	0	1	1	0					
Reserved	1	0	0	1	1	1	1	0	1	1	1	0				
LD An, PC	1	0	0	1	1	1	1	0	1	1	1	1	0		An	
Reserved	1	0	0	1	1	1	1	0	1	1	1	1	1			
JPF adr:22	1	0	0	1	1	1	1	1	0	0					Adr[21:16]	
JPT adr:22	1	0	0	1	1	1	1	1	0	1					Adr[21:16]	
JMP adr:22	1	0	0	1	1	1	1	1	1	0					Adr[21:16]	
JSR adr:22	1	0	0	1	1	1	1	1	1	1					Adr[21:16]	
LDC Rn, @Ai	1	0	1	0			Rn		0	0	0	0	0		Ai	
Reserved	1	0	1	0					0	0	0	0	1			
LD Dn, Ei	1	0	1	0	0		Dn		0	0	0	1	0		Ei	
LD En, Di	1	0	1	0	0		Di		0	0	0	1	1		En	
CMP EQ, An, Ai	1	0	1	0	1		An		0	0	0	1	0		Ai	
LD An, Ai	1	0	1	0	1		An		0	0	0	1	1		Ai	
LDW Rn, @[Ai+disp:16]	1	0	1	0			Rn		0	0	1	0	0		Ai	
LDW @[An+disp:16], Ri	1	0	1	0			Ri		0	0	1	0	1		An	
LDB Dn, @[Ai+disp:16]	1	0	1	0	0		Dn		0	0	1	1	0		Ai	
LDB @[An+disp:16], Di	1	0	1	0	0		Di		0	0	1	1	1		An	
LDW An, @[Ai+disp:16]	1	0	1	0	1		An		0	0	1	1	0		Ai	
LDW @[An+disp:16], Ai	1	0	1	0	1		Ai		0	0	1	1	1		An	
CMP GE, Dn, #imm:6	1	0	1	0	0		Dn		0	1					Imm:6	
ADD An, #imm:5	1	0	1	0	1		An		0	1	0				imm:5	
SUB An, #imm:5	1	0	1	0	1		An		0	1	1				imm:5	

Table 6-1. CalmRISC16 Instruction Set Map (Continued)

	15					8 7				0		
CMP EQ, An, #imm:22	1	0	1	0	0	An	1	0	Imm[21:16]			
LD An, #imm:22	1	0	1	0	1	An	1	0	Imm[21:16]			
ADD An, Ri	1	0	1	0	0	An	1	1	0	0	Ri	
SUB An, Ri	1	0	1	0	1	An	1	1	0	0	Ri	
MUL UU, Dn, Di	1	0	1	0	0	Dn	1	1	0	1	0	Di
MUL US, Dn, Di	1	0	1	0	0	Dn	1	1	0	1	1	Di
MUL SU, Dn, Di	1	0	1	0	1	Dn	1	1	0	1	0	Di
MUL SS, Dn, Di	1	0	1	0	1	Dn	1	1	0	1	1	Di
POP Rn[, Rm]	1	0	1	0	Rm		1	1	1	0	0	Rn
Reserved	1	0	1	0	0			1	1	1	0	1
POP An[, Am]	1	0	1	0	1	Am	1	1	1	0	1	An
PUSH Rn[, Rm]	1	0	1	0	Rm		1	1	1	1	0	Rn
Reserved	1	0	1	0	0			1	1	1	1	1
PUSH An[, Am]	1	0	1	0	1	Am	1	1	1	1	1	An
BSRD eoffset:13	1	0	1	1	Eoffset:13							
BRA EC:2, eoffset:8	1	1	0	0	0	0	0	EC:2	Eoffset:8			
Reserved	1	1	0	0	0	0	1					
BRAD EC:2, eoffset:8	1	1	0	0	0	1	0	EC:2	Eoffset:8			
BNZD H, eoffset:8	1	1	0	0	0	1	1	H	0	Eoffset:8		
Reserved	1	1	0	0	0	1	1	1				
BRA eoffset:11	1	1	0	0	1	0	Eoffset:11					
BRAD eoffset:11	1	1	0	0	1	1	Eoffset:11					
BRF eoffset:11	1	1	0	1	0	0	Eoffset:11					
BRFD eoffset:11	1	1	0	1	0	1	Eoffset:11					
BRT eoffset:11	1	1	0	1	1	0	Eoffset:11					
BRTD eoffset:11	1	1	0	1	1	1	Eoffset:11					
CLD Dn, imm:5	1	1	1	0	0	0	0	imm:5	0	Dn		
CLD imm:5, Di	1	1	1	0	0	0	0	imm:5	1	Di		
COP imm:13	1	1	1	Imm:13								

- Dn[15:0] : R0-R7
- H[15:0] : R6, R7
- An[21:0] : A8-A15, concatenation of En and Rn
- En[5:0] : E8-E15, MS 6-bit of An
- SP : equal to A15
- EC:2 : EC0,EC1,EC2,EC3
- Disp : unsigned displacement
- Eoffset : even signed offset
- Edisp : even unsigned displacement

## QUICK REFERENCE

Table 6-2. Quick Reference

Instruction	op1	op2	operation	flag
ADD SUB	Rn	#imm:7 Ri	op1 <- op1 + op2 op1 <- op1 + ~op2 + 1	T=C, Z0, Z1,V
LD	Rn	#imm:8 #imm:16 Ri	op1 <- op2	Z0, Z1
LDW	Rn	@[SP+edisp:9] @[Ai+edisp:5] @[Ai+Rj] @[Ai+disp:16]	op1 <- op2	–
LDW	@[SP+edisp:9] @[An+edisp:5] @[An+Rm] @[Ai+disp:16]	Ri	op1 <- op2	–
LDW	An	@[Ai+edisp:5] @[Ai+Rj] @[Ai+disp:16]	op1 <- op2	–
LDW	@[An+edisp:5] @[An+Rm] @[Ai+disp:16]	Ai	op1 <- op2	–
LDB	Dn	@[SP+disp:8] @[Ai+disp:4] @[Ai+Rj] @[Ai+disp:16]	op1<-{8'h0,op2[7:0]}	–
LDB	R0	@[A8+disp:8]	op1<-{8'h0,op2[7:0]}	–
LDB	@[SP+disp:8] @[An+disp:4] @[Ai+Rj] @[Ai+disp:16]	Di	op1 <- op2[7:0]	–
LDB	@[A8+disp:8]	R0	op1 <- op2[7:0]	–
ADC SBC	Rn	Ri #imm:16	op1 <- op1 + op2 + T op1 <- op1 + ~op2 + T	T=C,V, Z0,Z1
AND OR XOR	Rn	Ri #imm:16	op1 <- op1 & op2 op1 <- op1   op2 op1 <- op1 ^ op2	T=Z, Z0,Z1
TST	Rn	Ri #imm:16	op1 & op2	T=Z

Table 6-2. Quick Reference (Continued)

Instruction	op1	op2	operation	flag
CMP GE CMP GT CMPU GE CMPU GT CMP EQ	Rn	Ri  #imm:16	op1 + ~op2 + 1, T=~N op1 + ~op2 + 1, T=~N&~Z op1 + ~op2 + 1, T=C op1 + ~op2 + 1, T=C&~Z op1 + ~op2 + 1, T=Z	T
RR RL RRC SRB SR SRA SLB	Rn	-	op1 <- {op1[0],op1[15:1]} op1 <- {op1[14:0],op1[15]} op1 <- {T,op1[15:1]} op1 <- {8'h00,op1[15:8]} op1 <- {0,op1[15:1]} op1 <- {op1[15],op1[15:1]} op1 <- {op1[7:0],8'h00}	T=op1[0] T=op1[15] T=op1[0] T=op1[7] T=op1[0] T=op1[0] T= op1[8]
DT	Rn		op1 <- op1 + 0xffff	T=Z, Z0,Z1,V
COM	Rn		op1 <- ~op1	T=Z,Z0, Z1
INCC DECC COM2 COMC	Rn		op1 <- op1 + T op1 <- op1 + 0xffff + T op1 <- ~op1 + 1 op1 <- ~op1 + T	T=C,Z0, Z1
EXT	Rn		op1<~{8{op1[7]},op1[7:0]}	Z0, Z1
JPF JPT JMP JSR	Ai  addr:22		if(T==0) PC <- op1 if(T==1) PC <- op1 PC <- op1 A14 <- PC+(2 4), PC<-op1	-
ADD  ADD SUB	Rn  An	#imm:16  #imm:16 #imm:5 Ri	op1 <- op1 + op2  op1 <- op1 + op2 op1 <- op1 - op2	T=C, Z0,Z1,V  -
CMP EQ	Dn	#imm:8	op1 + ~op2 + 1	T=Z
AND OR XOR TST	R0	#imm:8	op1 <- op1 & {8'h00,op2} op1 <- op1   {8'h00,op2} op1 <- op1 ^ {8'h00,op2} op1 & {8'h00,op2}	T=Z[7:0]
BITR BITS BITC BITT	@[A8+R1]	bs:3	op1[op2] <- 0 op1[op2] <- 1 op1[op2] <- ~op1[op2] op1[op2] <- op1[op2]	T= ~op1[op2]
SYS	#imm:5	-	DA[4:0] <- op1	-
SWI	#imm:6	-	A14 <- PC+2, PC <- op2*4	IE, TE

Table 6-2. Quick Reference (Continued)

Instruction	op1	op2	operation	flag
CLRSR SETSR TSTSR	bs:3	–	SR[op1] <- 0 SR[op1] <- 1 T <- ~SR[op1]	–
RETD	–	–	PC <- A14	–
LD	R0	SR SPCL_FIQ SPCH_FIQ SSR_FIQ SPCL_IRQ SPCH_IRQ SSR_IRQ SSR_SWI	op1 <- op2	–
LD	SR SPCL_FIQ SPCH_FIQ SSR_FIQ SPCL_IRQ SPCH_IRQ SSR_IRQ SSR_SWI	R0	op1 <- op2	–
LD	An	PC Ai #imm:22	op1 <- op2 + 4 op1 <- op2 op1 <- op2	–
CMP EQ	An	Ai #imm:22	op1 + ~op2 + 1	T=Z[22:0]
LDC	Rn	@Ai	op1 <- PM[op2]	–
LD	Rn	Ei	op1 <- {10'h000, op2}	–
LD	En	Ri	op1 <- op2[5:0]	–
CMP GE	Dn	#imm:6	op1 + ~op2 + 1	T=~N
MUL UU MUL US MUL SU MUL SS	Dn	Di	op1 <- {0,op1[7:0]} * {0,op2[7:0]} op1 <- {0,op1[7:0]} * {op2[7],op2[7:0]} op1 <- {op1[7],op1[7:0]} * {0,op2[7:0]} op1 <- {op1[7],op1[7:0]} * {op2[7],op2[7:0]}	–
POP	Rn	Rm	op1 <- @[SP+2], op2 <- @[SP+4], SP <- SP+4	–
PUSH	Rn	Rm	@[SP] <- op1, @[SP-2] <- op2, SP <- SP-4	–



Table 6-2. Quick Reference (Continued)

Instruction	op1	op2	operation	flag
POP	An	Am	En<-@[SP+2], Rn<-@[SP+4], Em<-@[SP+6], Rm<-@[SP+8], SP<-SP+8	-
PUSH	An	Am	@[SP]<-Rn, @[SP-2]<-En, @[SP-4]<-Rm, @[SP-6]<-Em, SP<-SP-8	-
BSRD	eoffset:13	-	A14 <- PC+2, PC <- PC + 2 + op1	-
BRA/BRAD	EC:2	eoffset:8	if(EC:2 == 1) PC <- PC + 2 + op2	-
BNZD	R6	eoffset:8	if(Z0 == 0) PC <- PC + 2 + op2 R6 <- R6 - 1	Z0
BNZD	R7	eoffset:8	if(Z1 == 0) PC <- PC + 2 + op2 R7 <- R7 - 1	Z1
BRA/BRAD	eoffset:11	-	PC <- PC + 2 + op1	-
BRF/BRFD	eoffset:11	-	if(T==0) PC <- PC + 2 + op1	-
BRT/BRTD	eoffset:11	-	if(T==1) PC <- PC + 2+op1	-
CLD	Dn	imm:5	op1 <- Coprocessor[op2]	-
CLD	imm:5	Di	Coprocessor[op1] <- op2	
COP	imm:13	-	COPIR <- op2	

## ADC (1) – Add with Carry Register

**Format:** ADC Rn, Ri

**Description:** The ADC (Add with Carry Register) instruction is used to synthesize 32-bit addition. If register pairs R0, R1 and R2, R3 hold 32-bit values (R0 and R2 hold the least-significant word), the following instructions leave the 32-bit sum in R0, R1:

ADD R0, R2

ADC R1, R3

The instruction ADC R0, R0 produces a single-bit Rotate Left with Carry (17-bit rotate through the carry) on R0.

ADC adds the value of register Rn, and the value of the Carry flag (stored in the T bit), and the value of register Ri, and stores the result in register Rn. The T bit and the V flag are updated based on the result.

## ADC (2) – Add with Carry Immediate

**Format:** ADC Rn, #<imm:16>

**Description:** The ADC (Add with Carry Immediate) instruction is used to synthesize 32-bit addition with an immediate operand. If register pair R0, R1 holds a 32-bit value (R0 holds the least-significant word), the following instructions leave the 32-bit sum with 87653456h in R0, R1:

```
ADD R0, #3456h
```

```
ADC R1, #8765h
```

ADC adds the value of register Rn, and the value of the Carry flag (stored in the T bit), and the 16-bit immediate operand, and stores the result in register Rd. The T bit and the V flag are updated based on the result.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	0	1	0	1	1	1	1	Rn	

**Operation:** Rn := Rn + <imm:16> + T bit  
 T bit := Carry from (Rn + <imm:16> + T bit)  
 V flag := Overflow from (Rn + <imm:16> + T bit)  
 if(Rn == R6/R7) Z0/Z1 flag := ((Rn + <imm:16>) == 0)

**Exceptions:** None.

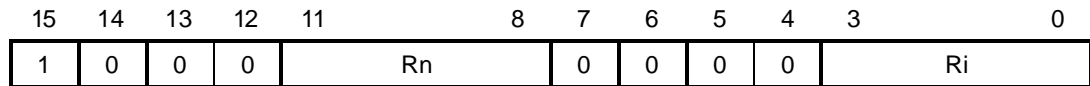
**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of ADC Rn, <imm:16> takes 2 cycles.

## ADD (1) – Add Register

**Format:** ADD Rn, Ri

**Description:** The ADD (Add Register) instruction is used to add two 16-bit values in registers. 32-bit addition can be achieved by executing ADC instruction in pair with this instruction.

ADD adds the value of register Rn, and the value of register Ri, and stores the result in register Rn. The T bit and the V flag are updated based on the result.



**Operation:** Rn := Rn + Ri  
 T bit := Carry from (Rn + Ri)  
 V flag := Overflow from (Rn + Ri)  
 if(Rn == R6/R7) Z0/Z1 flag := ((Rn + Ri) == 0)

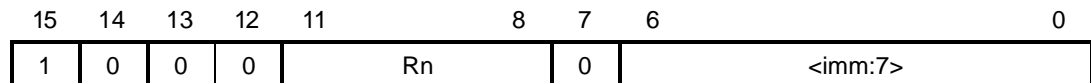
**Exceptions:** None

**Notes:** None

## ADD (2) – Add Small Immediate

**Format:** ADD Rn, #<imm:7>

**Description:** This form of ADD instruction is used to add a 7-bit (positive) immediate value to a register. ADD adds the value of register Rn, and the value of <imm:7>, and stores the result in register Rn. The T bit and the V flag are updated based on the result.



**Operation:** Rn := Rn + <imm:7>  
 T bit := Carry from (Rn + <imm:7>)  
 V flag := Overflow from (Rn + <imm:7>)  
 if(Rn == R6/R7) Z0/Z1 flag := ((Rn + <imm:7>) == 0)

**Exceptions:** None

**Notes:** <imm:7> is an unsigned amount.

## ADD (3) – Add Immediate

**Format:** ADD Rn, #<imm:16>

**Description:** The ADD (Add Immediate) instruction is used to add a 16-bit immediate value to a register. 32-bit addition or subtraction can be achieved by executing ADC or SBC instruction in pair with this instruction.

ADD adds the value of register Rn, and the value of <imm:16>, and stores the result in register Rn. The T bit and the V flag are updated based on the result.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	0	0	0	1	1	1	1	Rn	

**Operation:** Rn := Rn + <imm:16>  
 T bit := Carry from (Rn + <imm:16>)  
 V flag := Overflow from (Rn + <imm:16>)  
 if(Rn == R6/R7) Z0/Z1 flag := ((Rn + <imm:16>) == 0)

**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of ADD Rn, <imm:16> takes 2 cycles. The instruction "SUB Rn, #<imm:16>" does not exist.

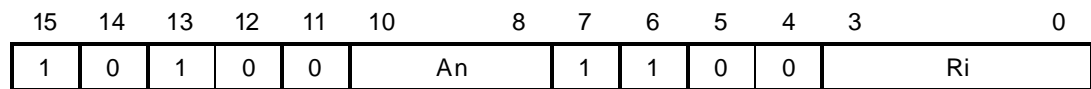
The result of "SUB Rn, #<imm:16>" instruction is identical with the result of "ADD Rn, #(2's complement of <imm:16>)" except when <imm:16> is zero. In that case, "SUB Rn, #<imm:7>" can be used.

## ADD (4) – Add Extended Register

**Format:** ADD An, Ri

**Description:** The ADD (Add Extended Register) instruction is used to add a 16-bit unsigned register value to a 22-bit register.

This instruction adds the value of 16-bit register Ri, and the value of 22-bit register An, and stores the result in register An.



**Operation:**  $An := An + Ri$

**Exceptions:** None

**Notes:** None

## ADD (5) – Add Immediate to Extended Register

**Format:** ADD An, #<imm:16>

**Description:** This form of ADD instruction is used to add a 16-bit unsigned immediate value to a 22-bit register. This instruction adds the value of <imm:16> to the value of An, and stores the result in register An.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	0
1	0	0	0	0	0	0	1	1	1	1	1	0	An	

**Operation:**  $An := An + \langle imm:16 \rangle$

**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of this instruction takes 2 cycles.

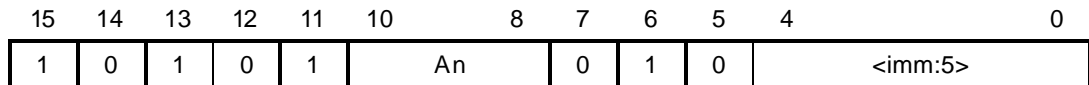


## ADD (6) – Add 5-bit Immediate to Extended Register

**Format:** ADD An, #<imm:5>

**Description:** This form of ADD instruction is used to add a 5-bit unsigned immediate value to a 22-bit register.

This instruction adds the value of 5-bit immediate <imm:5>, and the value of 22-bit register An, and stores the result in register An.



**Operation:**  $An := An + \langle imm:5 \rangle$

**Exceptions:** None

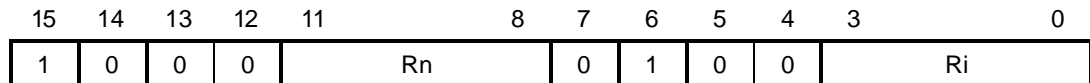
**Notes:** <imm:5> is an unsigned amount.

## AND (1) – AND Register

**Format:** AND Rn, Ri

**Description:** The AND (AND Register) instruction is used to perform bitwise AND operation on two values in registers, Rn and Ri.

The result is stored in register Rn. The T bit is updated based on the result.



**Operation:** Rn := Rn & Ri  
 T bit := ((Rn & Ri) == 0)  
 if(Rn == R6/R7) Z0/Z1 flag := ((Rn & Ri) == 0)

**Exceptions:** None

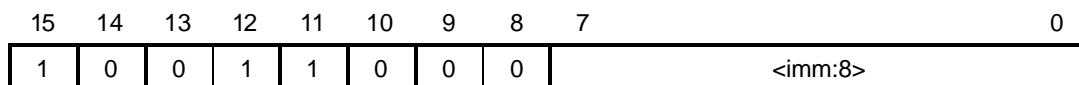
**Notes:** None

## AND (2) – AND Small Immediate

**Format:** AND R0, #<imm:8>

**Description:** The AND (AND Small Immediate) instruction is used to perform an 8-bit bitwise AND operation on two values in register R0 and <imm:8>.

The result is stored in register R0. The T bit is updated based on the result.



**Operation:** R0 := R0 & <imm:8>

T bit := ((R0 & <imm:8>)[7:0] == 0)

**Exceptions:** None

**Notes:** The register used in this operation is fixed to R0. Therefore, the operand should be placed in R0 before this instruction executes. <imm:8> is zero-extended to a 16-bit value before operation.

## AND (3) – AND Large Immediate

**Format:** AND Rn, #<imm:16>

**Description:** This type of AND instruction is used to perform bitwise AND operation on two values in register Rn and <imm:16>.

The result is stored in register Rn. The T bit is updated based on the result.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	1	0	0	1	1	1	1	Rn	

**Operation:** Rn := Rn & <imm:16>

T bit := ((Rn & <imm:16>) == 0)

if(Rn == R6/R7) Z0/Z1 flag := ((Rn & <imm:16>) == 0)

**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of this instruction takes 2 cycles.

## BITop – BIT Operation

**Format:** BITop @[A8+R1], #<bs:3>

**Description:** The BITop (Bit Operation) instruction is used to perform a bit operation on an 8-bit memory value. The allowed operations include reset (BITR), set (BITS), complement (BITC), and test (BITT).

BITop fetches the value of memory location specified by @(A8+R1), performs the specified operation on the specified bit, and stores the result back into the same memory location

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	1	0	0	0	0	OP	<bs:3>			

**Operation:** Temp := MEM[A8+R1]  
 T bit := ~Temp[<bs:3>]  
 if (BITop != BITT) {  
   Result := BITop(Temp, <bs:3>)  
   MEM[A8+R1] := Result  
 }

Here, BITop is BITR (OP == 00) | BITS (01) | BITC (10) | BITT (11). The bit location of these operations is specified by <bs:3>.

**Exceptions:** None

**Notes:** The address used to access data memory is obtained from the addition of two registers A8 and R1. No other registers can be used for this address calculation.

If you want to use a instruction which cause the change of T flag, you must add the nop instruction between two instructions.

BITR   @[A8+R1], #<bs:3>  
 NOP  
 CMP   EQ, R0, R2

## BNZD – Branch Not Zero with Autodecrement

**Format:** BNZD H, <offset:8>

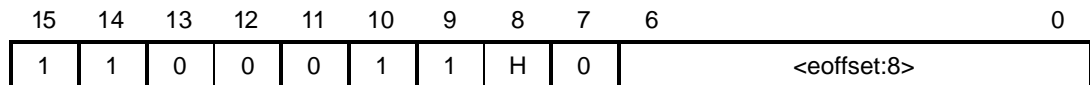
**Description:** The BNZD (Branch Not Zero with Delay Slot) instruction is used to change the program flow when the specified register value does not evaluate to zero. After evaluation, the value in register is automatically decremented. A typical usage of this instruction is as a backward branch at the end of a loop.

```

LOOP:
...
BNZD R6, LOOP // if (Z0 != 0) go back to LOOP
ADD R4, 3     // delay slot

```

In the above example, R6 is used as the loop counter. After specified loop iterations, BNZD is not taken and the control will come out of the loop, and R6 will have -1. For a loop with “N” iterations, the counter register used should be initially set to “(N-1)”. BNZD has a single delay slot; the instruction that immediately follows BNZD will be executed always regardless of whether BNZD is taken or not.



**Operation:**

```

if(H == R6) {
if(Z0 != 0) PC := PC + 2 + <offset:8>
R6 := R6 - 1
Z0 := ((R6-1) == 0)
} else { // H == R7
Same mechanism as the case R6
}

```

H is a register specifier denoting either R6 or R7.

**Exceptions:** None

**Notes:** When BNZD checks if H is zero by looking up the Z0 (for R6) or Z1 (for R7) bit in SR, these flags are updated as BNZD decrements the value of the register. For the first iteration, however, the user is responsible for resetting the flag, Z0 or Z1, before the loop starts execution.



## BRA EC – Branch on External Condition

**Format:** BRA(D) EC:2 <offset:8>

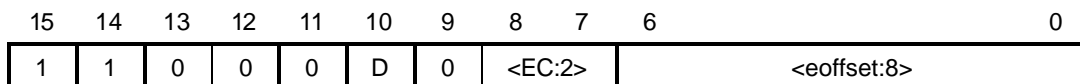
**Description:** The BRA EC (Branch on External Condition) instruction is used to change the program flow when a certain external condition is set. A typical usage of this instruction is to branch after a coprocessor operation as shown below:

```

COP <operation>
NOP
NOP
BRA EC0 OVERFLOW
...
OVERFLOW: ...
...

```

The BRA EC instruction checks the specified external condition (instead of checking the T bit as other branch instructions) and branch to the specified program address. There can be up to 4 external conditions, specified by the <EC:2> field in the instruction.



**Operation:** if (ExternalCondition\_n == True)  
PC := PC + 2 + <offset:8>

**Exceptions:** None

**Notes:** None



## BREAK – BREAK

**Format:** BREAK

**Description:** The BREAK instruction suspends the CalmRISC core for 1 cycle by keeping PC from increasing. Processor resumes execution after 1 cycle. This instruction is used for debugging purposes only and thus should not be used in normal operating modes. A core signal nBRK is asserted low for the cycle.

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

**Operation:** No operation with PC suspended for a single cycle.

**Exceptions:** None

**Notes:** None

## BSRD – Branch Subroutine with Delay Slot

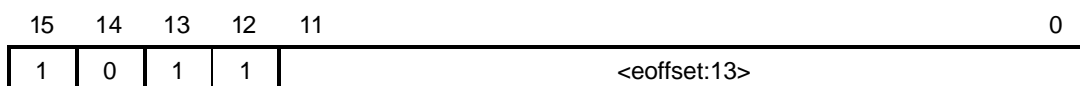
**Format:** BSRD <offset:13>

**Description:** The BSRD (Branch Subroutine with Delay slot) instruction is used to change the program flow to a subroutine by assigning the address of the subroutine to PC after saving the return address (PC+4) in the link register, or A14.

The address of the subroutine is calculated by:

1. sign-extending <offset:13> to 22 bits
2. adding this to the PC (which contains the address of the branch instruction plus 1)

After executing the subroutine, the program flow can return back to the instruction that follows the BSRD instruction by setting PC with the value stored in A14 (see JMP Ai instruction in page 7-52 and RET instruction in page 7-85). This instruction has a delay slot; the instruction that immediately follows BSRD will be always executed.



**Operation:** A14 := PC + 4  
PC := PC + 2 + <offset:13>

**Exceptions:** None

**Notes:** None

## CLD – Coprocessor Load

**Format:** CLD Dn, <imm:5> / CLD <imm:5>, Di

**Description:** The CLD (Coprocessor Load) instruction is used to transfer data from and to coprocessor by generating the core signals nCLDID and CLDWR. The content of DA[4:0] is <imm:5>, the address of coprocessor register to be read or written.

When a data item is read from coprocessor (CLD Dn, <imm:5>), it is stored in Dn. When a data item is written to coprocessor, it should be prepared in Di.

15	14	13	12	11	10	9	8		4	3	2	0
1	1	1	0	0	0	0	imm:5			M	Dn/Di	

**Operation:** (M == 0, read)  
 DA[4:0] := <imm:5>  
 nCLDID := 0  
 CLDWR := 0  
 Dn := (<imm:5>)  
 (M == 1, write)  
 DA[4:0] := <imm:5>  
 nCLDID := 0  
 CLDWR := 1  
 (<imm:5>) := Di

**Exceptions:** None

**Notes:** This instruction has a delay slot, because this instruction is 2-cycle instruction.

## CLRSR – Clear SR

**Format:** CLRSR bs:3

**Description:** The CLRSR (Clear SR) instruction is used to clear a specified bit in SR as follows:

CLRSR FE / IE / TE / V / Z0 / Z1 / PM

To clear the T bit, one can do as follows:

CMP GT, R0, R0

To turn on a specified bit in SR, the SETSR instruction is used.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	1	0	1	0	0	0	0	<bs:3>		

**Operation:** SR[<bs:3>] := 0

**Exceptions:** None

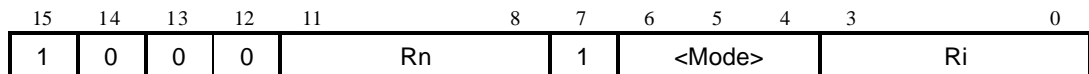
**Notes:** None

## CMP (1) – Compare Register

**Format:** CMPmode Rn, Ri

**Description:** The CMP (Compare Register) instruction is used to compare two values in registers Rn and Ri. The allowed modes include GE (Greater or Equal), GT (Greater Than), UGE (Unsigned Greater or Equal), UGT (Unsigned Greater Than), and EQ (Equal).

CMP subtracts the value of Ri from the value of Rn and performs comparison based on the result. The contents of Rn and Ri are not changed after this operation. The T bit is updated for later reference.



**Operation:** Temp := Rn - Ri

T bit := ~Negative if (<Mode> == GE)  
 ~Negative && ~Zero if (<Mode> == GT)  
 Carry if (<Mode> == UGE)  
 Carry && ~Zero if (<Mode> == UGT)  
 Zero if (<Mode> == EQ)

<Mode> encoding: GE (000), GT (001), UGE (010), UGT (011), and EQ (100).

**Exceptions:** None

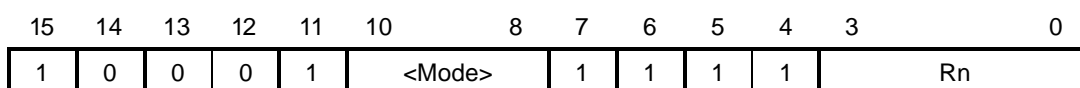
**Notes:** None

## CMP (2) – Compare Immediate

**Format:** CMPmode Rn, #<imm:16>

**Description:** The CMP (Compare Immediate) instruction is used to compare two values in register Rn and <imm:16>. The allowed modes include GE (Greater or Equal), GT (Greater Than), UGE (Unsigned Greater or Equal), UGT (Unsigned Greater Than), and EQ (Equal).

CMP subtracts the value of <imm:16> from the value of Rn and performs comparison based on the result. The contents of Rn is not changed, however, after this operation. The T bit is updated for later reference.



**Operation:** Temp := Rn - <imm:16>

T bit := ~Negative if (<Mode> == GE)

~Negative && ~Zero if (<Mode> == GT)

Carry if (<Mode> == UGE)

Carry && ~Zero if (<Mode> == UGT)

Zero if (<Mode> == EQ)

<Mode> encoding: GE (000), GT (001), UGE (010), UGT (011), and EQ (100).

**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of CMPmode #<imm:16> takes 2 cycles.

## CMP (3) – Compare Short Immediate

**Format:** CMP GE, Dn, #<imm:6>

**Description:** The CMP (Compare Immediate) instruction is used to perform signed-comparison of the register Dn and an unsigned immediate value <imm:6>. Dn is one of the registers from R0 to R7. CMP subtracts the value of <imm:6> from the value of Dn and performs signed-comparison based on the result. The contents of Dn is not changed, however, after this operation. The T bit is updated for later reference.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	Dn			0	1	imm:6					

**Operation:** T bit := ~Negative of (Rn - <imm:6>)

**Exceptions:** None

**Notes:** None

## CMPEQ (1) – Compare Equal Extended Register

**Format:** CMP EQ, An, Ai

**Description:** The CMP EQ (Compare Equal Extended Register) instruction is used to compare two values in registers An and Ai.

This instruction is a restricted form of more general CMPmode instructions for a 22-bit equality comparison between register values.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	An			0	0	0	1	0	Ai		

**Operation:** T bit := (An == Ai)

An or Ai refers to registers from A8 to A15 with their 6-bit extensions.

**Exceptions:** None

**Notes:** None

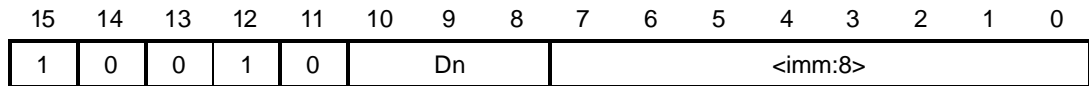


## CMPEQ (2) – Compare Equal Small Immediate

**Format:** CMP EQ, Dn, #<imm:8>

**Description:** The CMP EQ (Compare Equal Small Immediate) instruction is used to compare two values in register Dn and <imm:8>. <imm:8> is zero-extended to 16 bits before comparison.

This instruction is a restricted form of more general CMPmode instructions for an 8-bit equality comparison between a register value and an immediate value.



**Operation:** T bit := ((Dn - <imm:8>) == 0)  
Dn refers to registers R0 - R8.

**Exceptions:** None

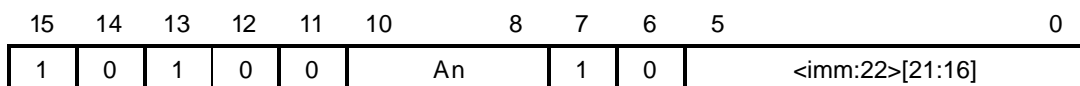
**Notes:** None

## CMPEQ (3) – Compare Equal Large Immediate

**Format:** CMP EQ An, #<imm:22>

**Description:** The CMP EQ (Compare Equal Large Immediate) instruction is used to compare two values in register An and <imm:22>.

This instruction is a restricted form of more general CMPmode instructions for a 22-bit equality comparison between a register value and an immediate value.



**Operation:** T bit := Zero from (An - <imm:22>)

An refers to registers from A8 to A15 with their 6-bit extensions.

**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate (<imm:22>[15:0]) follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of CMP EQ <imm:22> takes 2 cycles.

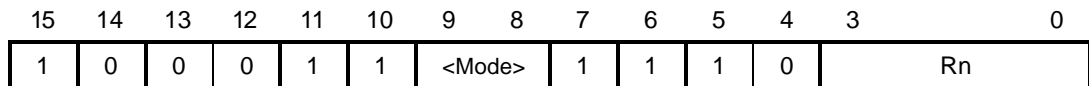
## COM – Complement

**Format:** COMmode Rn

**Description:** The COM (Complement) instruction is used to compute 1's or 2's complement of a register value Rn. Utilizing various modes, 32-bit complement operation can be done. If register pair R0, R1 holds a 32-bit value (R0 holds the least-significant word), the following instructions leave the 32-bit 2's complement in R0, R1:

```
COM2 R0    // 2's complement
COMC R1    // 2's complement with carry
```

COM computes the 1's complement of the value of register Rn. COM2 computes the 2's complement, and COMC computes the 2's complement value when T bit has been set. If T bit is clear, COM2 is equivalent to COM.



**Operation:**

```
if (<Mode> == 00) { // COM
    Rn := ~Rn
    T bit := (Rn == 0)
}
if (<Mode> == 01) { // COM2
    Rn := ~Rn + 1
    T bit := Carry from (~Rn + 1)
}
if (<Mode> == 10) { // COMC
    Rn := ~Rn + T bit
    T bit := Carry from (~Rn + T)
}
Encoding of <Mode>:
00: COM, 01: COM2, 10: COMC
if(Rn == R6/R7) Z0/Z1 := Zero flag of the result.
```

**Exceptions:** None

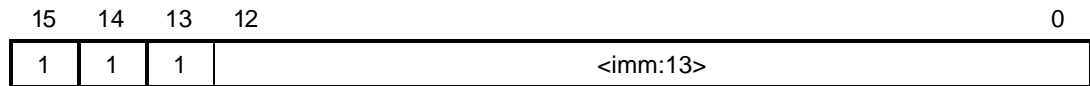
**Notes:** None

## COP – Coprocessor

**Format:** COP <imm:13>

**Description:** The COP (Coprocessor) instruction is used to perform a coprocessor operation, specified by <imm:13>. Certain coprocessor operations set external conditions, upon which branches can be executed (see BRECN instructions).

The <imm:13> should be greater or equal to 0x200.



**Operation:** Perform a coprocessor operation by placing signals on core output pins as follows:

Core output signal COPIR[12:0] := <imm:13>

Core output signal nCOPID := LOW

**Exceptions:** None

**Notes:** None

## DECC – Decrement with Carry

**Format:** DECC Rn

**Description:** The DECC (Decrement with Carry) instruction is used to synthesize 32-bit decrement. If register pair R0, R1 holds a 32-bit value (R0 holds the least-significant word), the following instructions leave the 32-bit decremented value in R0, R1:

```
DEC R0    // this is implemented by ADD R0, -1
DECC R1
```

DECC decrements the value of Rn by 1 only if the Carry flag (stored in the T bit) is clear, and stores the result back in register Rn. The T bit and the V flag are updated based on the result.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	1	0	1	1	1	1	1	0	Rn	

**Operation:** Rn := Rn - 1 + T bit  
 T bit := Carry from (Rn - 1 + T bit)  
 V flag := Overflow from (Rn - 1 + T bit)  
 if(Rn == R6/R7) Z0/Z1 := ((Rn - 1 + T) == 0)

**Exceptions:** None

**Notes:** None

## DT – Decrement and Test

**Format:** DT Rn

**Description:** The DT (Decrement and Test) instruction is used to decrement the value of a specified register and test it. This instruction provides a compact way to control register indexing for loops. The T bit and the V flag are updated based on the result.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	1	0	0	1	1	1	1	0	Rn	

**Operation:** Rn := Rn - 1  
 T bit := ((Rn - 1) == 0)  
 V flag := Overflow from (Rn - 1)  
 if(Rn == R6/R7) Z0/Z1 := ((Rn - 1) == 0)

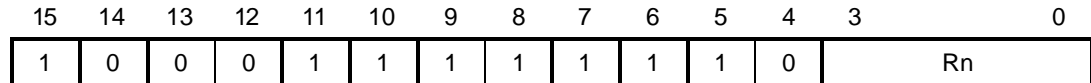
**Exceptions:** None

**Notes:** None

## EXT – Sign-Extend

**Format:** EXT Rn

**Description:** The EXT (Sign Extend) instruction is used to sign-extend an 8-bit value in Rn. This instruction copies Rn[7] to Rn[15:8].



**Operation:** All bits from Rn[15] to Rn[8] := Rn[7]  
if(Rn == R6/R7) Z0/Z1 := (Result == 0)

**Exceptions:** None

**Notes:** None

## INCC – Increment with Carry

**Format:** INCC Rn

**Description:** The INCC (Increment with Carry) instruction is used to synthesize 32-bit increment. If register pair R0, R1 holds a 32-bit value (R0 holds the least-significant word), the following instructions leave the 32-bit incremented value in R0, R1:

INC R0 // will be replaced by ADD R0, 1

INCC R1

INCC increments the value of Rn by 1 only if the Carry flag (stored in the T bit) is set, and stores the result back in register Rn. The T bit and the V flag are updated based on the result.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	1	0	1	0	1	1	1	0	Rn	

**Operation:** Rn := Rn + T bit  
 T bit := Carry from (Rn + T bit)  
 V flag := Overflow from (Rn + T bit)  
 if(Rn == R6/R7) Z0/Z1 := ((Rn + T0) == 0)

**Exceptions:** None

**Notes:** None



## JMP (1) – Jump Register

**Format:** JPF/JPT/JMP/JSR Ai

**Description:** The Jump Register instructions change the program flow by assigning the value of register Ai into PC.

JPF and JPT are conditional jumps that check the T bit to determine whether or not to jump to the target address. JMP unconditionally jumps to the target. JSR is an unconditional jump but saves the return address (the immediately following instruction to JSR) in the link register, A14. At the end of each subroutine, JMP A14 will change the program flow back to the original call site.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	1	1	M[1]	1	1	1	0	M[0]	Ai		

**Operation:** (M == 00, JPF)  
 if (T bit == FALSE)  
 PC := Ai  
 (M == 01, JPT)  
 if (T bit == TRUE)  
 PC := Ai  
 (M == 10, JMP)  
 PC := Ai  
 (M == 11, JSR)  
 A14 := PC + 2  
 PC := Ai

**Exceptions:** None

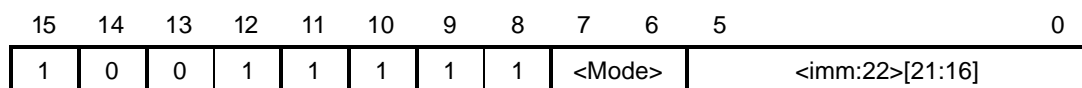
**Notes:** There is no delay slot for these instructions. Therefore, when conditional branch JPF or JPT is taken, the instruction in the pipeline which is fetched from PC+2 will be squashed. In case of JMP and JSR (always taken), the following instruction fetched will be always squashed.

## JMP (2) – Jump Immediate

**Format:** JPF/JPT/JMP/JSR <imm:22>

**Description:** The Jump Immediate instructions change the program flow by assigning the value of <imm:22> into PC.

JPF and JPT are conditional jumps that check the T bit to determine whether or not to jump to the target address. JMP unconditionally jumps to the target. JSR is an unconditional jump but saves the return address (the immediately following instruction to JSR) in the link register, A14. At the end of each subroutine, JMP A14 will change the program flow back to the original call site.



**Operation:** (<Mode> == 00, JPF)

if (T bit == FALSE)

PC := <imm:22>

(<Mode> == 01, JPT)

if (T bit == TRUE)

PC := <imm:22>

(<Mode> == 10, JMP)

PC := <imm:22>

(<Mode> == 11, JSR)

A14 := PC + 4

PC := <imm:22>

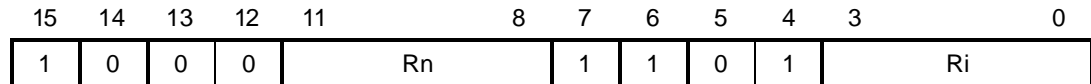
**Exceptions:** None

**Notes:** These are 2-word instructions, where the 16-bit immediate (<imm:22>[15:0]) follows the instruction word shown above. As fetching of a 2-word instruction takes 2 cycles, no later instructions will be in processor pipeline when the branch is taken (thus no squashing).

## LD (1) – Load Register

**Format:** LD Rn, Ri

**Description:** The LD (Load Register) instruction is used to transfer a register value to a register.



**Operation:** Rn := Ri  
if(Rn == R6/R7) Z0/Z1 := (Ri == 0)

**Exceptions:** None

**Notes:** None

## LD (2) – Load Register

**Format:** LD An, Ai

**Description:** This form of LD instruction (Load Extended Register) is used to load a 22-bit register value to a 22-bit register.

15	14	13	12	11	10	8	7	6	5	4	3	2	0
1	0	1	0	1	An	0	0	0	1	1			Ai

**Operation:** An := Ai

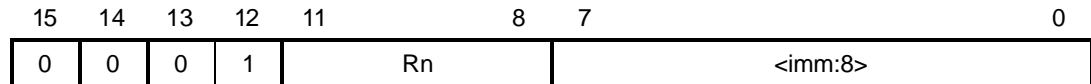
**Exceptions:** None

**Notes:** None

## LD (3) – Load Short Immediate

**Format:** LD Rn, #<imm:8>

**Description:** The LD (Load Short Immediate) instruction is used to load an 8-bit immediate value to a register.



**Operation:** Rn[15:8] := 0, Rn[7:0] := <imm:8>  
 if(Rn == R6/R7) Z0/Z1 := (<imm:8> == 0)

**Exceptions:** None

**Notes:** None

## LD (4) – Load Immediate

**Format:** LD Rn, #<imm:16>

**Description:** This form of LD instruction (Load Immediate) is used to load a 16-bit immediate value to a register.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	1	1	0	1	1	1	1	1		Rn

**Operation:** Rn := <imm:16>  
if(Rn == R6/R7) Z0/Z1 := (<imm:16> == 0)

**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of this instruction takes 2 cycles.



## LD RExt – Load Register Extension

**Format:** LD Dn, Ei / LD En, Di

**Description:** The LD RExt (Load Register Extension) instructions are used to transfer a register value to and from a 6-bit extension register.

15	14	13	12	11	10	8	7	6	5	4	3	2	0
1	0	1	0	0	Dn(or Di)	0	0	0	1	M	Ei (or En)		

**Operation:** (M == 0, LD Dn, Ei)  
 Dn := Ei (zero-extended to 16 bits)  
 (M == 1, LD En, Di)  
 En := Di (lower 6 bits only)

**Exceptions:** None

**Notes:** None



## LDB (1) – Load Byte Register Disp.

**Format:** LDB Dn, @[Ai+<disp:4>] / LDB @[An+<disp:4>], Di

**Description:** The LDB (Load Byte Register Displacement) instruction is used to load a byte from or to data memory at the location specified by the register Ai and a 4-bit displacement.

15	14	13	12	11	10	8	7	6	4	3	0
0	1	1	M	0	Dn or Di	0	Ai or An			<disp:4>	

**Operation:** (M == 0, LDB Dn, @[Ai+<disp:4>])  
 Dn := DM[(Ai+<disp:4>)]  
 (M == 1, LDB @[An+<disp:4>], Di)  
 DM[(An+<disp:4>)] := Di

**Exceptions:** None

**Notes:** None

## LDB (2) – Load Byte Register Large Disp.

**Format:** LDB Dn, @[Ai+<disp:16>] / LDB @[An+<disp:16>], Di

**Description:** The LDB (Load Byte Register Large Displacement) instruction is used to load a byte from or to data memory at the location specified by the register Ai and a 16-bit displacement.

15	14	13	12	11	10	8	7	6	5	4	3	2	0
1	0	1	0	0	Dn or Di	0	0	1	1	M	Ai or An		

**Operation:** (M == 0, LDB Dn, @[Ai+<disp:16>])  
 Dn := DM[(Ai+<disp:16>)]  
 (M == 1, LDB @[An+<disp:16>], Di)  
 DM[(An+<disp:16>)] := Di

**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of this instruction takes 2 cycles.

## LDB (3) – Load Byte Register Indexed

**Format:** LDB Dn, @[Ai+Rj] / LDB @[An+Rm], Di

**Description:** The LDB (Load Byte Register Indexed) instruction is used to load a byte from or to data memory at the location specified by the register Ai (or An) and the second register Rj (or Rm).

15	14	13	12	11	10	8	7	6	4	3	0
0	1	1	M	0	Dn or Di	1	Ai or An	Rj or Rm			

**Operation:** (M == 0, LDB Dn, @[Ai+Rj])  
 Dn := DM[(Ai+Rj)]  
 (M == 1, LDB @[An+Rm], Di)  
 DM[(An+ Rm)] := Di

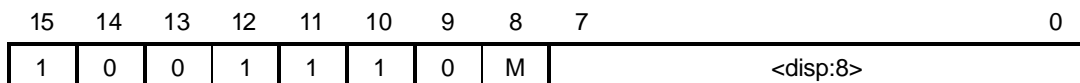
**Exceptions:** None

**Notes:** None

## LDB (4) – Load Byte to R0 Register Disp.

**Format:** LDB R0, @[A8+<disp:8>] / LDB @[A8+<disp:8>], A8

**Description:** The LDB (Load Byte to R0 Register Displacement) instruction is used to load a byte from or to data memory at the location specified by the register A8 and an 8-bit displacement.



**Operation:** (M == 0, LDB R0, @[A8+<disp:8>])  
 R0 := DM[(A8+<disp:8>)]  
 (M == 1, LDB @[A8+<disp:8>], R0)  
 DM[(A8+<disp:8>)] := R0

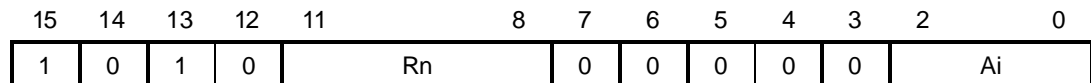
**Exceptions:** None

**Notes:** This single-word instruction allows a user to access a wider range of data memory than the LDB (1) instruction by providing a larger displacement, at the expense of the restrictions that only the R0 and A8 registers are used for data transfer and address computation.

## LDC – Load Code

**Format:** LDC Rn, @Ai

**Description:** The LDC instruction is used to transfer a register value from the program memory. The program memory address is specified by the 22-bit register An. LDC is useful to look up the data stored in program memory, such as the coefficient table for certain numerical algorithms.



**Operation:** Rn := PM[Ai]

**Exceptions:** None

**Notes:** None

## LD PC – Load Program Counter

**Format:** LD An, PC

**Description:** The LD PC (Load Program Counter) instruction is used to transfer the value of PC into a 22-bit register An. This instruction provides a way to implement position independent code (PIC) on CalmRISC16 even in the absence of general virtual memory support. After executing this instruction, An will be used to compute a PC-relative location of a data item or a code section.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	0
1	0	0	1	1	1	1	0	1	1	1	1	0	An	

**Operation:** An := PC + 4

**Exceptions:** None

**Notes:** None

## LD SvR (1) – Load from Saved Register

**Format:** LD R0, SPCL\_\* / LD R0, SPCH\_\* / LD R0, SSR\_\*

**Description:** The LD SvR (Load from Saved Register) instructions are used to transfer a value from the specified interrupt register, e.g., SSR\_FIQ. Only R0 register is used for this data transfer.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	1	1	1	1	0	1	0	1	0	<RS>	

**Operation:** R0 := <specified\_saved\_register>  
 Encoding for <RS> (Register Specifier):  
 0000: SPCL\_FIQ, 0001: SPCH\_FIQ, 0010: SSR\_FIQ,  
 0100: SPCL\_IRQ, 0101: SPCH\_IRQ, 0110: SSR\_IRQ,  
 1010: SSR\_SWI

**Exceptions:** None

**Notes:** None

## LD SvR (2) – Load to Saved Register

**Format:** LD SPCL\_\*, R0 / LD SPCH\_\*, R0 / LD SSR\_\*, R0

**Description:** The LD SvR (Load to Saved Register) instructions are used to transfer a value to the specified interrupt register, e.g., SSR\_FIQ. Only R0 register is used for this data transfer.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	1	1	1	1	0	1	0	1	1	<RS>	

**Operation:** <specified\_saved\_register> := R0  
 Encoding for <RS> (Register Specifier):  
 0000: SPCL\_FIQ, 0001: SPCH\_FIQ, 0010: SSR\_FIQ,  
 0100: SPCL\_IRQ, 0101: SPCH\_IRQ, 0110: SSR\_IRQ,  
 1010: SSR\_SWI

**Exceptions:** None

**Notes:** None



## LD SR – Load Status Register

**Format:** LD R0, SR / LD SR, R0

**Description:** The LD SR (Load Status Register) instruction is used to transfer a value to and from SR. Only R0 register is used for this operation.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	1	0	1	0	0	1	1	0	1	M

**Operation:** (M == 0, LD R0, SR)  
R0 := SR  
(M == 1, LD SR, R0)  
SR := R0

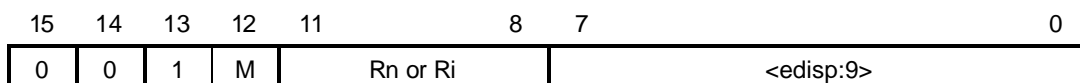
**Exceptions:** None

**Notes:** None

## LDW (1) – Load Word Stack Disp.

**Format:** LDW Rn, @[SP+<edisp:9>] / LDW @[SP+<edisp:9>], Ri

**Description:** The LDW (Load Word Stack Displacement) instruction is used to load a word from or to data memory at the location specified by the SP register (or A15) and an even 9-bit displacement. <edisp:9>, from 0 to 510, is encoded into 8-bit displacement by dropping the least significant bit.



**Operation:** (M == 0, LDW Rn, @[SP+<edisp:9>])  
 Rn := DM[(SP + <edisp:9>)]  
 (M == 1, LDW @[SP+<edisp:9>], Ri)  
 DM[(SP + <edisp:9>)] := Ri

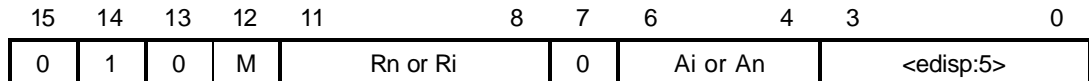
**Exceptions:** None

**Notes:** For memory transfer per word, the (byte) address need to be aligned to be even. Thus, if (SP + <edisp:9>) is an odd number, it will be made even by clearing the least significant bit. <edisp:9> can denote an even number from 0 to 510.

## LDW (2) – Load Word Register Small Disp.

**Format:** LDW Rn, @[Ai+<edisp:5>] / LDW @[An+<edisp:5>], Ri

**Description:** The LDW (Load Word Register Displacement) instruction is used to load a word from or to data memory at the location specified by the register Ai and a 5-bit even displacement from 0 to 30. <edisp:5> is encoded to 4-bit number by dropping the least significant bit.



**Operation:** (M == 0, LDW Rn, @[Ai+<edisp:5>])  
 Rn := DM[(Ai + <edisp:5>)]  
 (M == 1, LDW @[An+<edisp:5>], Ri)  
 DM[(An + <edisp:5>)] := Ri

**Exceptions:** None

**Notes:** For memory transfer per word, the (byte) address need to be aligned to be even. Thus, if (Ai + <edisp:5>) is an odd number, it will be made even by clearing the least significant bit. <edisp:5> can denote an even number from 0 to 30.

## LDW (3) – Load Word Register Disp.

**Format:** LDW Rn, @[Ai+<disp:16>] / LDW @[An+<disp:16>], Ri

**Description:** The LDW (Load Word Register Large Displacement) instruction is used to load a word from or to data memory at the location specified by the register Ai and a 16-bit displacement.

15	14	13	12	11		8	7	6	5	4	3	2	0
1	0	1	0	Rn or Ri			0	0	1	0	M	Ai or An	

**Operation:** (M == 0, LDW Rn, @[Ai+<disp:16>])  
 Rn := DM[(Ai + <disp:16>)]  
 (M == 1, LDW @[An+<disp:16>], Ri)  
 DM[(An + <disp:16>)] := Ri

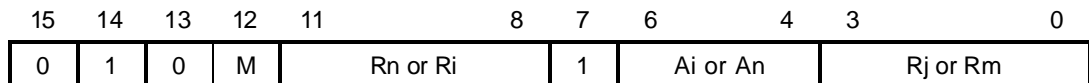
**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of this instruction takes 2 cycles. For memory transfer per word, the (byte) address need to be aligned to be even. Thus, if (Ai + <disp:16>) is an odd number, it will be made even by clearing the least significant bit.

## LDW (4) – Load Word Register Indexed

**Format:** LDW Rn, @[Ai+Rj] / LDW @[An+Rm], Ri

**Description:** The LDW (Load Word Register Indexed) instruction is used to load a word from or to data memory at the location specified by the register Ai (or An) and the second register Rj (or Rm), which is an unsigned value.



**Operation:** (M == 0, LDW Rn, @[Ai+Rj])  
 Rn := DM[(Ai+Rj)]  
 (M == 1, LDW @[An+Rm], Ri)  
 DM[(An+Rm)] := Ri

**Exceptions:** None

**Notes:** For memory transfer per word, the (byte) address needs to be aligned to be even. Thus, if (Ai + Rj) or (An + Rm) is an odd number, it will be made even by clearing the least significant bit.

## LDW (5) – Load Word Register Small Disp.

**Format:** LDW An, @[Ai+<edisp:5>] / LDW @[Ai+<edisp:5>], An

**Description:** The LDW (Load Word Register Displacement) instruction is used to load 2 word from or to data memory at the location specified by the register Ai and a 5-bit even displacement from 0 to 30. <edisp:5> is encoded to 4-bit number by dropping the least significant bit.

15	14	13	12	11	10	8	7	6	4	3	0
0	1	1	M	1	An	0	Ai	<edisp:5>			

**Operation:** (M == 0, LDW An, @[Ai+<edisp:5>])

En := DM[(Ai + <edisp:5>)]

Rn := DM[(Ai + <edisp:5> + 2)]

(M == 1, LDW @[Ai+<edisp:5>], An)

DM[(Ai + <edisp:5>)] := En

DM[(Ai + <edisp:5> + 2)] := Rn

**Exceptions:** None

**Notes:** For memory transfer per word, the (byte) address need to be aligned to be even. Thus, if (Ai + <edisp:5>) is an odd number, it will be made even by clearing the least significant bit. <edisp:5> can denote an even number from 0 to 30.

## LDW (6) – Load Word Register Disp.

**Format:** LDW An, @[Ai+<disp:16>] / LDW @[Ai+<disp:16>], An

**Description:** The LDW (Load Word Register Large Displacement) instruction is used to load 2 word from or to data memory at the location specified by the register Ai and a 16-bit displacement.

15	14	13	12	11	10	8	7	6	5	4	3	2	0
1	0	1	0	1	An	0	0	1	1	M			Ai

**Operation:** (M == 0, LDW An, @[Ai+<disp:16>])  
 En := DM[(Ai + <disp:16>)]  
 Rn := DM[(Ai + <disp:16> + 2)]  
 (M == 1, LDW @[Ai+<disp:16>], An)  
 DM[(Ai + <disp:16>)] := En  
 DM[(Ai + <disp:16> + 2)] := Rn

**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of this instruction takes 2 cycles. For memory transfer per word, the (byte) address need to be aligned to be even. Thus, if (Ai + <disp:16>) is an odd number, it will be made even by clearing the least significant bit.

## LDW (7) – Load Word Register Indexed

**Format:** LDW An, @[Ai+Rj] / LDW @[Ai+Rj], An

**Description:** The LDW (Load Word Register Indexed) instruction is used to load 2 word from or to data memory at the location specified by the register Ai and the second register Rj, which is an unsigned value.

15	14	13	12	11	10	8	7	6	4	3	0
0	1	1	M	1	An	1	Ai	Rj			

**Operation:** (M == 0, LDW An, @[Ai + Rj])  
 $E_n := DM[(Ai + Rj)]$   
 $R_n := DM[(Ai + Rj + 2)]$   
 (M == 1, LDW @[Ai + Rj], An)  
 $DM[(Ai + Rj)] := E_n$   
 $DM[(Ai + Rj + 2)] := R_n$

**Exceptions:** None

**Notes:** For memory transfer per word, the (byte) address needs to be aligned to be even. Thus, if (Ai + Rj) is an odd number, it will be made even by clearing the least significant bit.



## MUL – Multiplication

**Format:** MUL Mode, Dn, Di

**Description:** The instruction MUL performs 8x8 multiplication of the least significant byte of Dn and the least significant byte of Di. Dn and Di are registers from R0 to R7. The 16-bit multiplication result is written back to Dn. The mode is one of UU, US, SU, SS. The mode indicates each operand is signed value or unsigned value.

15	14	13	12	11	10	8	7	6	5	4	3	2	0
1	0	1	0	M1	Dn			1	1	0	1	M2	Di

**Operation:**

```

if(M1 == 0 && M2 == 0) // mode = UU
    Dn := lower 16 bits of ({0,Dn[7:0]} * {0, Di[7:0]})
else if(M1 == 0 && M2 == 1) // mode == US
    Dn := lower 16 bits of ({0,Dn[7:0]} * {Di[7],Di[7:0]})
else if(M1 == 1 && M2 == 0) // mode == SU
    Dn := lower 16 bits of ({Dn[7],Dn[7:0]} * {0,Di[7:0]})
else // mode == SS
    Dn := lower 16 bits of ({Dn[7],Dn[7:0]} * {Di[7],Di[7:0]})

```

**Exceptions:** None

**Notes:** None

## NOP – No Operation

**Format:** NOP

**Description:** The NOP (No Operation) instruction does not perform any operation.

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

**Operation:** None

**Exceptions:** None

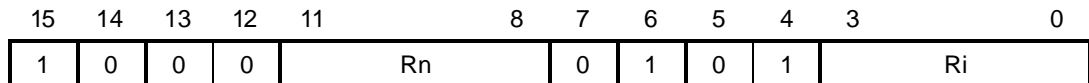
**Notes:** None

## OR (1) – OR Register

**Format:** OR Rn, Ri

**Description:** The OR (OR Register) instruction is used to perform bitwise OR operation on two values in registers, Rn and Ri.

The result is stored in register Rn. The T bit is updated based on the result.



**Operation:** Rn := Rn | Ri  
 T bit := ((Rn | Ri) == 0)  
 if(Rn == R6/R7) Z0/Z1 := ((Rn|Ri) == 0)

**Exceptions:** None

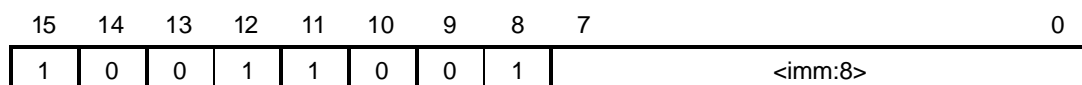
**Notes:** None

## OR (2) – OR Small Immediate

**Format:** OR R0, #<imm:8>

**Description:** The OR (OR Small Immediate) instruction is used to perform bitwise OR operation on two values in register R0 and <imm:8>.

The result is stored in register R0. The T bit is updated based on the result.



**Operation:** R0 := R0 | <imm:8>

T bit := ((R0 | <imm:8>)[7:0] == 0)

**Exceptions:** None

**Notes:** The register used in this operation is fixed to R0. Therefore, the operand should be placed in R0 before this instruction executes. <imm:8> is zero-extended to a 16-bit value before operation.

## OR (3) – OR Large Immediate

**Format:** OR Rn, #<imm:16>

**Description:** This type of OR instruction is used to perform bitwise OR operation on two values in register Rn and <imm:16>.

The result is stored in register R0. The T bit is updated based on the result.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	1	0	1	1	1	1	1	Rn	

**Operation:** Rn := Rn | <imm:16>  
 T bit := ((Rn | <imm:16>) == 0)  
 if(Rn == R6/R7) Z0/Z1 := ((Rn | <imm:16>) == 0)

**Exceptions:** None

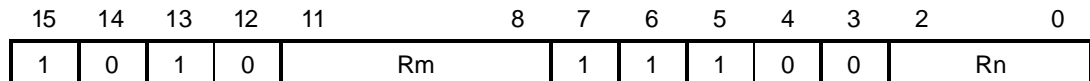
**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of this instruction takes 2 cycles.

## POP (1) – Load Register from Stack

**Format:** POP Rn, Rm / POP Rn

**Description:** The POP instruction load one or two 16-bit data from software stack to general registers. In the instruction of “POP Rn, Rm”, there are some restrictions on Rn and Rm.

- Rn and Rm should not be R15.
- If Rn is one of the 8 registers from R0 to R7, Rm should also be one of them. If Rn is one of the registers from R8 to R14, Rm should also be one of them. For example, “POP R7, R8” is illegal.
- If Rn is the same as Rm, pop operation occurs only once. “POP Rn, Rn” is equivalent to “POP Rn”.



**Operation:**

```

if(Rn == Rm) { // POP Rn
    Rn := DM[SP + 2]
    SP := SP + 2
} else {
    Rn := DM[SP + 2]
    Rm := DM[SP + 4]
    SP := SP + 4
}

```

**Exceptions:** None

**Notes:** None

## POP (2) – Load Register from Stack

**Format:** POP An, Am / POP An

**Description:** The POP instruction load one or two 22-bit data from software stack to extended registers. In the instruction of “POP An, Am”, there are some restrictions on An and Am.

- An and Am should not be A15.
- If An is the same as Am, pop operation occurs only once. “POP An, An” is equivalent to “POP An”.

15	14	13	12	11	10		8	7	6	5	4	3	2	0
1	0	1	0	1	Am		1	1	1	0	1	An		

**Operation:**

```

if(An == Am) { // POP An
    En := lower 6 bits of DM[SP + 2]
    Rn := DM[SP + 4]
    SP := SP + 4
} else {
    En := lower 6 bits of DM[SP + 2]
    Rn := DM[SP + 4]
    Em := lower 6 bits of DM[SP + 6]
    Rm := DM[SP + 8]
    SP := SP + 8
}

```

**Exceptions:** None

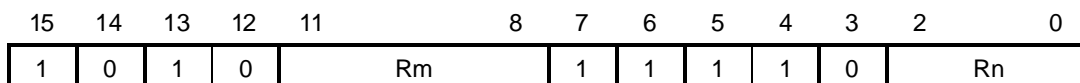
**Notes:** None

## PUSH (1) – Load Register to Stack

**Format:** PUSH Rn, Rm / PUSH Rn

**Description:** The PUSH instruction load one or two 16-bit data from general registers to software stack. In the instruction of “PUSH Rn, Rm”, there are some restrictions on Rn and Rm.

- Rn and Rm should not be R15.
- If Rn is one of the 8 registers from R0 to R7, Rm should also be one of them. If Rn is one of the registers from R8 to R14, Rm should also be one of them. For example, “PUSH R7, R8” is illegal.
- If Rn is the same as Rm, push operation occurs only once. “PUSH Rn, Rn” is equivalent to “PUSH Rn”.



**Operation:**

```

if(Rn == Rm) { // PUSH Rn
    DM[SP] := Rn
    SP := SP - 2
} else {
    DM[SP] := Rn
    DM[SP - 2] := Rm
    SP := SP - 4
}

```

**Exceptions:** None

**Notes:** None

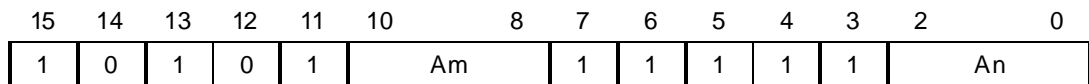


## PUSH (2) – Load Register to Stack

**Format:** PUSH An, Am / PUSH An

**Description:** The PUSH instruction load one or two 22-bit data to software stack from extended registers. In the instruction of “PUSH An, Am”, there are some restrictions on An and Am.

- An and Am should not be A15.
- If An is the same as Am, push operation occurs only once. “PUSH An, An” is equivalent to “PUSH An”.



**Operation:**

```

if(An == Am) { // PUSH An
    DM[SP] := Rn
    DM[SP - 2] := {10'h000, En}
    SP := SP - 4
} else {
    DM[SP] := Rn
    DM[SP - 2] := {10'h000, En}
    DM[SP - 4] := Rm
    DM[SP - 6] := {10'h000, Em}
    SP := SP - 8
}

```

**Exceptions:** None

**Notes:** None

## RETD – Ret. from Subroutine with Delay Slot

**Format:** RETD

**Description:** The RETD (Return from Subroutine with Delay Slot) instruction is used to finish a subroutine and return by jumping to the address specified by the link register or A14. The difference between RETD and JMP A14 is that RETD has a delay slot, which allows efficient implementation of small subroutines.

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

**Operation:** PC := A14

**Exceptions:** None

**Notes:** None

## RET\_FIQ – Return from Fast Interrupt

**Format:** RET\_FIQ

**Description:** The RET\_FIQ (Return from Fast Interrupt) instruction is used to finish a FIQ handler and resume the normal program execution. When this instruction is executed, SSR\_FIQ (saved SR) is restored into SR, and the program control transfers to (SPCH\_FIQ:SPCL\_FIQ).

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

**Operation:** SR := SSR\_FIQ  
PC := (SPCH\_FIQ:SPCL\_FIQ)

**Exceptions:** None

**Notes:** Fast Interrupt is requested through the core signal nFIQ. When the request is acknowledged, SR and current PC are saved in the designated registers (namely SSR\_FIQ and SPCH\_FIQ:SPCL\_FIQ) assigned for FIQ processing. Such bits in SR as FE, IE, and TE are cleared, and PM is set.

## RET\_IRQ – Return from Interrupt

**Format:** RET\_IRQ

**Description:** The RET\_IRQ (Return from Interrupt) instruction is used to finish an IRQ handler and resume the normal program execution. When this instruction is executed, SSR\_IRQ (saved SR) is restored into SR, and the program control transfers to (SPCH\_IRQ:SPCL\_IRQ).

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

**Operation:** SR := SSR\_IRQ  
PC := (SPCH\_IRQ:SPCL\_IRQ)

**Exceptions:** None

**Notes:** Interrupt is requested through the core signals nIRQ. When the request is acknowledged, SR and current PC are saved in the designated registers (namely SSR\_IRQ and SPCH\_IRQ:SPCL\_IRQ) assigned for IRQ processing. Such bits in SR as IE and TE are cleared, and PM is set.

## RET\_SWI – Return from Software Interrupt

**Format:** RET\_SWI

**Description:** The RET\_SWI (Return from Software Interrupt) instruction is used to finish a SWI handler and resume the normal program execution. When this instruction is executed, SSR\_FIQ (saved SR) is restored into SR, and the program control transfers to the address A14 (link register).

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

**Operation:** SR := SSR\_SWI  
PC := A14

**Exceptions:** None

**Notes:** Software interrupt is initiated by executing a SWI instruction from applications. When SWI instruction is executed, SR and current PC are saved in the designated registers (namely SSR\_SWI and A14) assigned for SWI processing.

## RL – Rotate Left

**Format:** RL Rn

**Description:** The RL (Rotate Left) instruction rotates the value of Rn left by one bit and stores the result back in Rn. T bit is updated as a result of this operation.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	0	0	1	1	1	1	0	Rn	

**Operation:** Rn := Rn << 1, Rn[0] = MSB of Rn before rotation  
T bit := MSB of Rn before rotation

**Exceptions:** None

**Notes:** None

## RR – Rotate Right

**Format:** RR Rn

**Description:** The RR (Rotate Right) instruction rotates the value of Rn right by one bit and stores the result back in Rn. T bit is updated as a result of this operation.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	0	0	0	1	1	1	0	Rn	

**Operation:** Rn := Rn >> 1, MSB of Rn = Rn[0] before rotation  
T bit := Rn[0] before rotation

**Exceptions:** None

**Notes:** None

## RRC – Rotate Right with Carry

**Format:** RRC Rn

**Description:** The RRC (Rotate Right with Carry) instruction rotates the value of (Rn:T bit) right by one bit and stores the result back in Rn. T bit is updated as a result of this operation.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	0	1	0	1	1	1	0	Rn	

**Operation:** Rn := Rn >> 1, MSB of Rn = T bit before rotation  
T bit := Rn[0] before rotation

**Exceptions:** None

**Notes:** None



## SBC (1) – Subtract with Carry Register

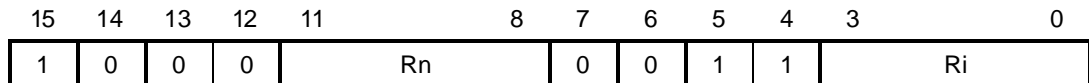
**Format:** SBC Rn, Ri

**Description:** The SBC (Subtract with Carry) instruction is used to synthesize 32-bit subtraction. If register pairs R0, R1 and R2, R3 hold 32-bit values (R0 and R2 hold the least-significant word), the following instructions leave the 32-bit result in R0, R1:

SUB R0, R2

SBC R1, R3

SBC subtracts the value of register Ri, and the value of the Carry flag (stored in the T bit), from the value of register Rn, and stores the result in register Rn. The T bit and the V flag are updated based on the result.



**Operation:**  $Rn := Rn + \sim Ri + T \text{ bit}$   
 T bit := Carry from  $(Rn + \sim Ri + T \text{ bit})$   
 V flag := Overflow from  $(Rn + \sim Ri + T \text{ bit})$   
 if  $(Rn == R6/R7)$  Z0/Z1 :=  $((Rn + \sim Ri + T) == 0)$

**Exceptions:** None

**Notes:** None

## SBC (2) – Subtract with Carry Immediate

**Format:** SBC Rn, #<imm:16>

**Description:** The SBC (Subtract with Carry immediate) instruction is used to synthesize 32-bit subtraction with an immediate operand. If register pair R0, R1 holds a 32-bit value (R0 holds the least-significant word), the following instructions leave the 32-bit subtraction result with 34157856h in R0, R1:

```
SUB R0, #7856h
```

```
SBC R1, #3415h
```

SBC subtracts the value of <imm:16>, and the value of the Carry flag (stored in the T bit), from the value of Rn, and stores the result in register Rd. The T bit and the V flag are updated based on the result.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	0	1	1	1	1	1	1		Rn

**Operation:**  $Rn := Rn + \sim\langle imm:16 \rangle + T \text{ bit}$   
 T bit := Carry from  $(Rn + \sim\langle imm:16 \rangle + T \text{ bit})$   
 V flag := Overflow from  $(Rn + \sim\langle imm:16 \rangle + T \text{ bit})$   
 if  $(Rn == R6/R7)$  Z0/Z1 :=  $((Rn + \sim\langle imm:16 \rangle + T) == 0)$

**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of this instruction takes 2 cycles.

## SETSR – Set SR

**Format:** SETSR bs:3

**Description:** The SETSR (Set SR) instruction is used to set a specified bit in SR as follows:

SETSR FE / IE / TE / V / Z0 / Z1 / PM

To set the T bit, one can do as follows:

CMP EQ, R0, R0

To clear a specified bit in SR, the CLRSR instruction is used.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	0
1	0	0	1	1	1	1	0	1	0	0	0	1	<bs:3>	

**Operation:** SR[<bs:3>] := 1

**Exceptions:** None

**Notes:** None

## SLB – Shift Left Byte

**Format:** SLB Rn

**Description:** The SLB (Shift Left Byte) instruction shift the value of Rn left by 8 bit and stores the result back in Rn. The low 8 bit positions are filled with 0's. T bit is updated as a result of this operation.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	1	1	1	1	0	1	0	0	0	Rn	

**Operation:** SR[15:8] := Rn[7:0] and Rn[7:0] := 8'h00  
T bit := Rn[8] before shifting

**Exceptions:** None

**Notes:** None

## SR – Shift Right

**Format:** SR Rn

**Description:** The SR (Shift Right) instruction shifts the value of Rn right by one bit and stores the result back in Rn. T bit is updated as a result of this operation.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	1	0	0	1	1	1	0	Rn	

**Operation:** Rn := Rn >> 1, with Rn[15] set to 0  
T bit := Rn[0] before shifting

**Exceptions:** None

**Notes:** None

## SRA – Shift Right Arithmetic

**Format:** SRA Rn

**Description:** The SRA (Shift Right Arithmetic) instruction shifts the value of Rn right by one bit and stores the result back in Rn. While doing so, the original sign bit (most significant bit) is copied to the most significant bit of the result. T bit is updated as a result of this operation.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	1	0	1	1	1	1	0	Rn	

**Operation:** Rn := Rn >> 1, with Rn[15] set to the original value  
T bit := Rn[0] before shifting

**Exceptions:** None

**Notes:** None

## SRB – Shift Right Byte

**Format:** SRB Rn

**Description:** The SRB (Shift Right Byte) instruction shifts the value of Rn right by 8 bit and stores the result back in Rn. The high 8 bit positions are filled with 0's. T bit is updated as a result of this operation.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	1	1	1	0	Rn			

**Operation:** Rn[7:0] := Rn[15:8] and Rn[15:8] := 8'h00  
T bit := Rn[7] before shifting

**Exceptions:** None

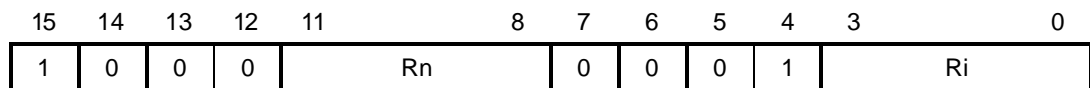
**Notes:** None

## SUB (1) – Subtract Register

**Format:** SUB Rn, Ri

**Description:** The SUB (Subtract Register) instruction is used to subtract a 16-bit register value from another 16-bit register value. 32-bit subtraction can be achieved by executing SBC instruction in pair with this instruction.

SUB subtracts the value of register Ri from the value of Rn, and stores the result in register Rn. The T bit and the V flag are updated based on the result.



**Operation:** Rn := Rn - Ri  
 T bit := Carry from (Rn - Ri)  
 V flag := Overflow from (Rn - Ri)  
 if(Rn == R6/R7) Z0/Z1 := ((Rn - Ri) == 0)

**Exceptions:** None

**Notes:** None

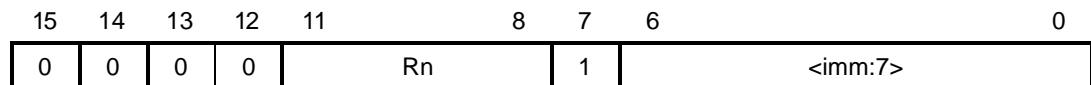


## SUB (2) – Subtract Small Immediate

**Format:** SUB Rn, #<imm:7>

**Description:** This form of SUB instruction is used to subtract a 7-bit immediate value from a register.

It subtracts the value of <imm:7> from the value of register Rn, and stores the result in register Rn. The T bit and the V flag is updated based on the result.



**Operation:** Rn := Rn - <imm:7>

T bit := Carry from (Rn - <imm:7>)

V flag := Overflow from (Rn - <imm:7>)

if(Rn == R6/R7) Z0/Z1 := ((Rn - <imm:7>) == 0)

**Exceptions:** None

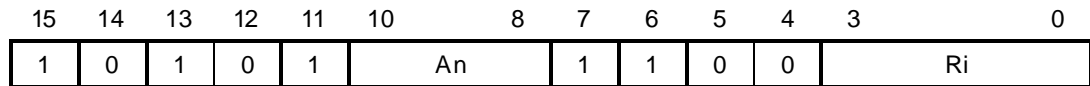
**Notes:** <imm:7> is an unsigned amount.

## SUB (3) – Subtract Extended Register

**Format:** SUB An, Ri

**Description:** This form of SUB instruction (Subtract Extended Register) is used to add a 16-bit unsigned register value from a 22-bit value in register.

This instruction subtracts the value of 16-bit register Ri from the value of 22-bit register An, and stores the result in register An.



**Operation:**  $An := An - Ri$

**Exceptions:** None

**Notes:** None

## SUB (4) – Subtract Large Immediate

**Format:** SUB An, #<imm:16>

**Description:** The SUB (Subtract Large Immediate) instruction is used to subtract a 16-bit unsigned immediate value from a 22-bit register.

SUB subtracts the value of <imm:16> from the value of An, and stores the result in register An.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	0
1	0	0	0	0	0	0	1	1	1	1	1	1	An	

**Operation:** An := An - <imm:16>

**Exceptions:** None

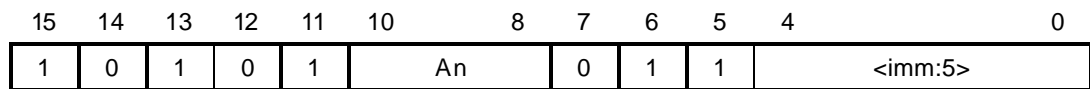
**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of this instruction takes 2 cycles.

## SUB (5) – Subtract 5-bit Immediate

**Format:** SUB An, #<imm:5>

**Description:** This form of SUB instruction (Subtract Extended Register) is used to subtract a 5-bit unsigned immediate value from a 22-bit register.

This instruction subtracts the value of 5-bit immediate <imm:5> from the value of 22-bit register An, and stores the result in register An.



**Operation:** An := An - <imm:5>

**Exceptions:** None

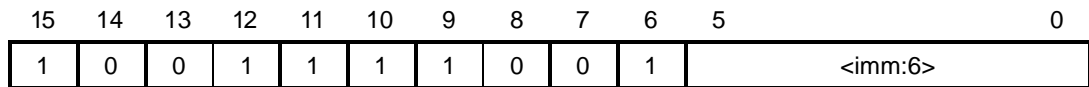
**Notes:** None

## SWI – Software Interrupt

**Format:** SWI #<imm:6>

**Description:** The SWI (Software Interrupt) instruction performs a specified set of operations (i.e., an SWI handler). This instruction can be used as an interface to the low-level system software such as operating system.

Executing this instruction is similar to performing a function call. However, interrupts (IRQ and TRQ) will be masked off so that when a software interrupt is handled, it can be seen as an uninterruptible operation. Note that FIQ can still be triggered when an SWI is serviced. Return from a SWI handler is done by RET\_SWI unlike normal function calls.



**Operation:**

- A14 := PC + 2
- SSR\_SWI := SR
- IE := 0, TE := 0
- PC := <imm:6> << 2

**Exceptions:** None

**Notes:** Program addresses from 000000h to 0000feh are reserved for SWI handlers. SWI vectors 0 and 1 are not used, as the addresses from 000000h to 000007h are reserved for other interrupts.

## SYS – System

**Format:** SYS #<imm:5>

**Description:** The SYS (System) instruction is used for system peripheral interfacing using DA[4:0] and nSYSID core signals.

15	14	13	12	11	10	9	8	7	6	5	4	0
1	0	0	1	1	1	1	0	0	0	1	<imm:5>	

**Operation:** core output signal DA[4:0] := <imm:5>, DA[21:5] := (unchanged)  
core output signal nSYSID := LOW

**Exceptions:** None

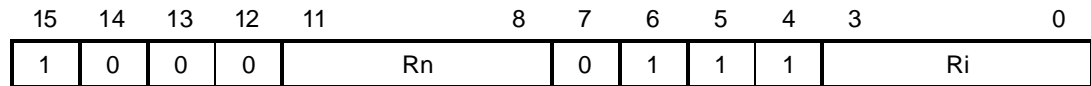
**Notes:** None

## TST (1) – Test Register

**Format:** TST Rn, Ri

**Description:** The TST (TST Register) instruction is used to determine if many bits of a register are all clear, or if at least one bit of a register is set.

TST performs a comparison by logically ANDing the value of register Rn with the value of Ri.  
T bit is set according to the result.



**Operation:** Temp := Rn & Ri  
T bit := ((Rn & Ri) == 0)

**Exceptions:** None

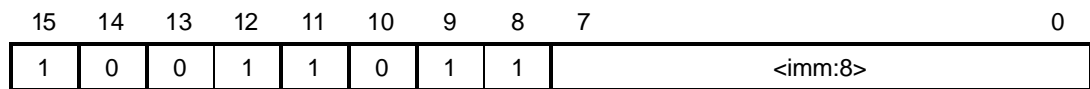
**Notes:** None

## TST (2) – Test Small Immediate

**Format:** TST R0, #<imm:8>

**Description:** This type of TST instruction is used to determine if many bits of a register are all clear, or if at least one bit of a register is set.

TST performs a comparison by logically ANDing the value of register Rn with the value of Ri. T bit is set according to the result.



**Operation:** Temp n := Rn & <imm:8>  
T bit := ((Rn & <imm:8>)[7:0] == 0)

**Exceptions:** None

**Notes:** The register used in this operation is fixed to R0. Therefore, the operand should be placed in R0 before this instruction executes.



## TST (3) – Test Large Immediate

**Format:** TST Rn, #<imm:16>

**Description:** This type of TST instruction is used to determine if many bits of a register are all clear, or if at least one bit of a register is set.

TST performs a comparison by logically ANDing the value of register Rn with the value of Ri. T bit is set according to the result.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	1	1	1	1	1	1	1	Rn	

**Operation:** Temp := Rn & <imm:16>  
T bit := ((Rn & <imm:16>) == 0)

**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of this instruction takes 2 cycles.

## TSTSR – Test SR

**Format:** TSTSR bs:3

**Description:** The TSTSR (Test SR) instruction is used to test a specified bit in SR as the following example shows:

TST FE / IE / TE / V / Z0 / Z1 / PM

To set or clear a specified bit, the SETSR or CLRSR instruction is used.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2		0
	1	0	0	1	1	1	1	0	1	0	0	1	0	<bs:3>		

**Operation:** T bit := ~SR[<bs:3>]

**Exceptions:** None

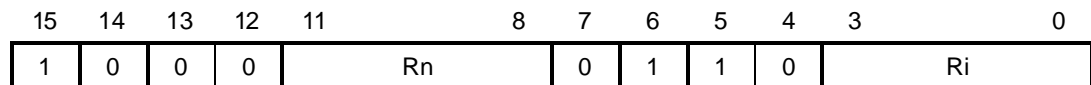
**Notes:** None

## XOR (1) – XOR Register

**Format:** XOR Rn, Ri

**Description:** The XOR (XOR Register) instruction is used to perform bitwise XOR operation on two values in registers, Rn and Ri.

The result is stored in register Rn. The T bit is updated based on the result.



**Operation:**  $Rn = Rn \wedge Ri$   
 T bit =  $((Rn \wedge Ri) == 0)$   
 if(Rn == R6/R7) Z0/Z1 :=  $((Rn \wedge Ri) == 0)$

**Exceptions:** None

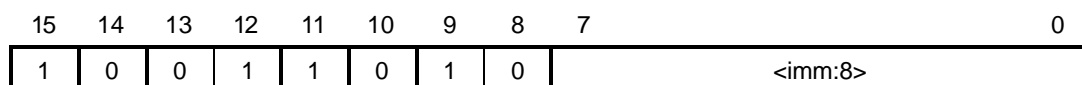
**Notes:** None

## XOR (2) – XOR Small Immediate

**Format:** XOR R0, #<imm:8>

**Description:** This type of XOR instruction is used to perform bitwise XOR operation on two values in register R0 and <imm:8>.

The result is stored in register R0. The T bit is updated based on the result.



**Operation:**  $R0 = R0 \wedge \text{<imm:8>}$

T bit =  $((R0 \wedge \text{<imm:8>})[7:0] == 0)$

**Exceptions:** None

**Notes:** The register used in this operation is fixed to R0. Therefore, the operand should be placed in R0 before this instruction executes. <imm:8> is zero-extended to a 16-bit value before operation.

## XOR (3) – XOR Large Immediate

**Format:** XOR Rn, #<imm:16>

**Description:** This type of XOR instruction is used to perform bitwise XOR operation on two values in register Rn and <imm:16>.

The result is stored in register Rn. The T bit is updated based on the result.

15	14	13	12	11	10	9	8	7	6	5	4	3	0
1	0	0	0	0	1	1	0	1	1	1	1		Rn

**Operation:**  $Rn = Rn \wedge \langle imm:16 \rangle$   
 T bit =  $((Rn \wedge \langle imm:16 \rangle) == 0)$   
 if  $(Rn == R6/R7)$  Z0/Z1 :=  $((Rn \wedge \langle imm:16 \rangle) == 0)$

**Exceptions:** None

**Notes:** This is a 2-word instruction, where the 16-bit immediate follows the instruction word shown above. Unlike 1-word instructions, therefore, fetching of this instruction takes 2 cycles.

# 7 PLL (PHASE LOCKED LOOP)

## OVERVIEW

S3CC11B/FC11B builds clock synthesizer for system clock generation, which can operate external crystal for reference, using internal phase-locked loop (PLL) and voltage-controlled oscillator (VCO). The input clock to the PLL block should be 2.048 MHz by the pre-scaler.

### System Clock Circuit

The system clock circuit has the following component:

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

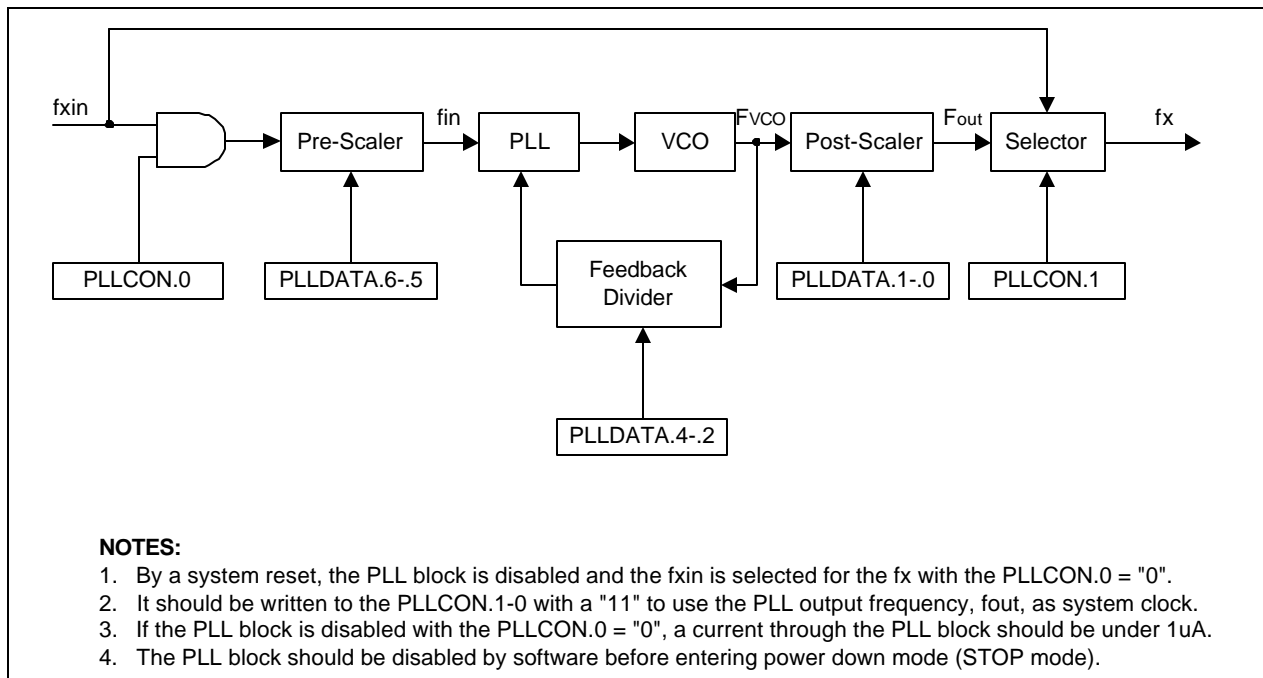


Figure 7-1. Phase-Locked Loop Circuit Diagram

## PLLCON — PLL Data Register

3F0076H

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	–	–	–	–	–	–	0	0
Read/Write	–	–	–	–	–	–	R/W	R/W

.7–.2

**Bits 7–2**

Not used
----------

.1

**fx Selection Bit**

0	Select fxin
1	Select fout

**Note:** Where fxin is the main oscillator clock and fout is the PLL clock.

.0

**PLL Enable Bit**

0	Disable PLL
0	Enable PLL

## PLLDATA — PLL Data Register

3F0077H

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	–	0	0	0	0	0	0	0
Read/Write	–	R/W	R/W	R/W	R/W	R/W	R/W	R/W

.7

**Bits 7**

Not used
----------

.6–.5

**Pre-Scaler Bits**

$f_{xin} \div 2^N$ , N = 0, 1, and 2.
---------------------------------------

.4–.2

**Feedback Divider Bits**

$f_{vco} \div (N + 8)$ , N = 0, 2, 4, 6, ....., and 14.
---

.1–.0

**Post-Scaler Bits**

$f_{vco} \div (N + 1)$ , N = 0, 1, 2, and 3.
--

**OSCCON** — Oscillator Control Register

3F0003H

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	–	–	–	–	0	0	0	0
Read/Write	–	–	–	–	R/W	R/W	R/W	R/W
.7–.4	<b>Bits 7–4</b> Not used							
.3	<b>Main Oscillator Control Bit</b>							
	0	Main oscillator RUN						
	1	Main oscillator STOP						
.2	<b>Sub Oscillator Control Bit</b>							
	0	Sub oscillator RUN						
	1	Sub oscillator STOP						
.1	<b>Bit 1</b> Not used							
.0	<b>System Clock Source Selection Bit</b>							
	0	Select main oscillator for system clock						
	1	Select sub oscillator for system clock						

**CLKCON** — Clock Control Register

3F0002H

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	–	–	–	–	–	–	0	0
Read/Write	–	–	–	–	–	–	R/W	R/W
.7–.2	<b>Bits 7–2</b> Not used							
.1–.0	<b>System Clock Selection Bits</b>							
	0	0	fxx/8					
	0	1	fxx/4					
	1	0	fxx/2					
	1	1	fxx/1					



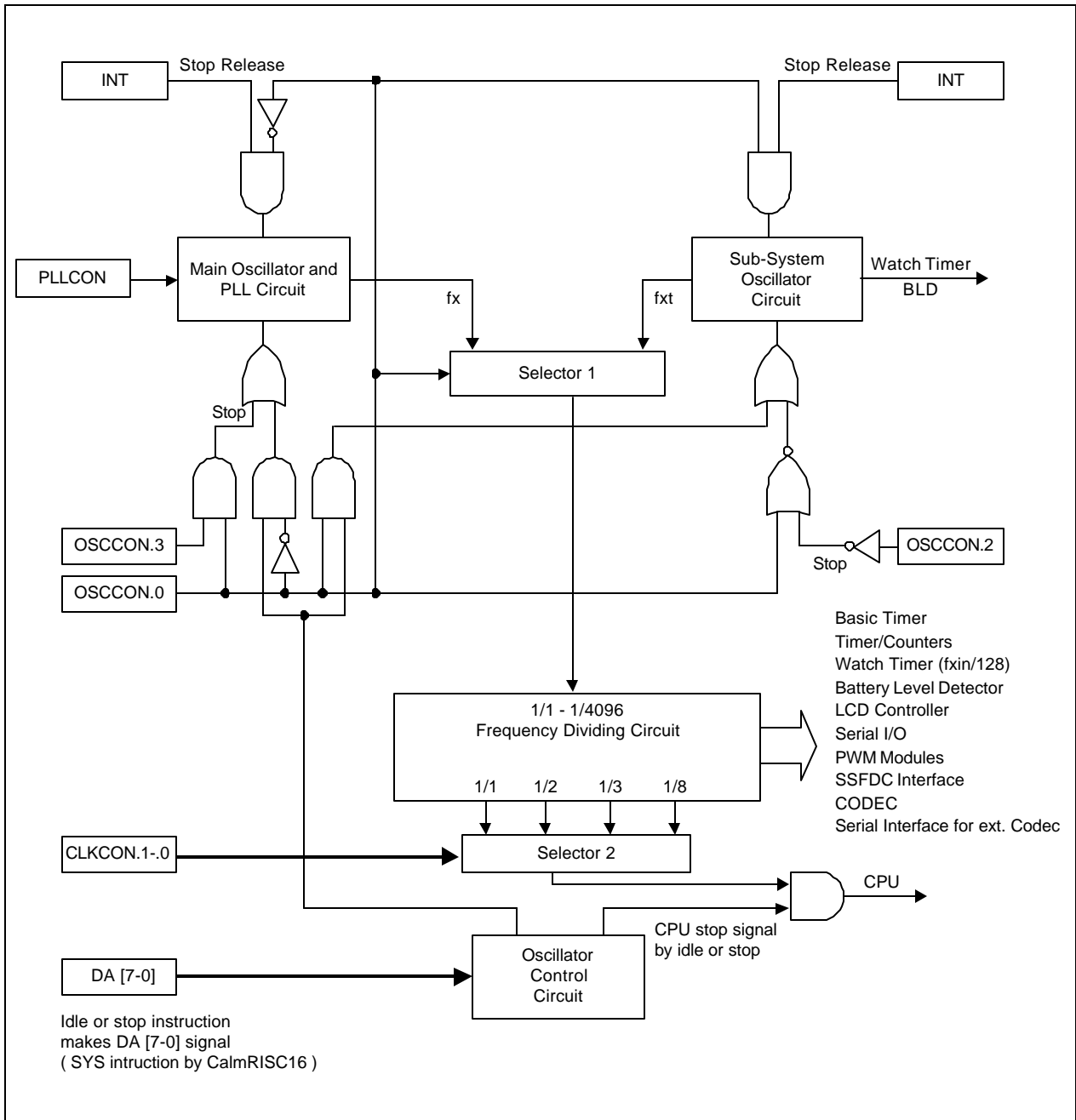


Figure 7-2. System Clock Circuit Diagram

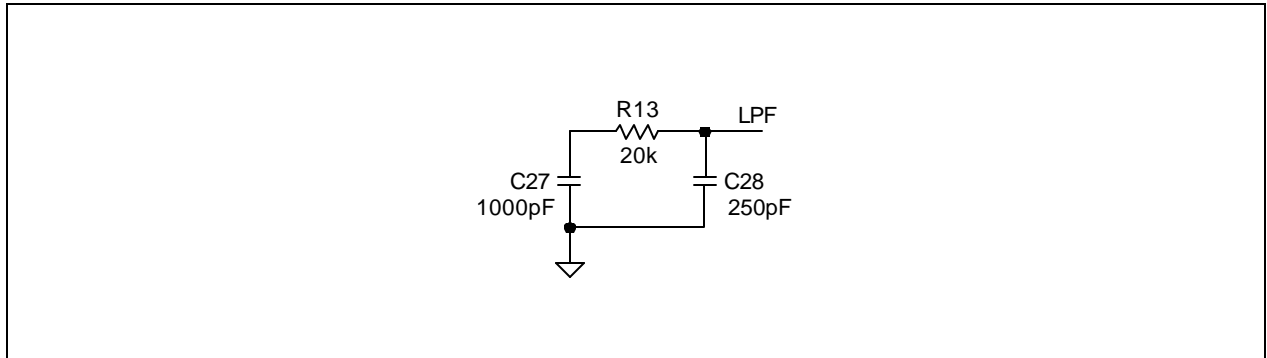


Figure 7-3. External Loop Filter for PLL

## NOTES

# 8 RESET AND POWER-DOWN

## OVERVIEW

During a power-on reset, the voltage at  $V_{DD}$  goes to High level and the nRESET pin is forced to Low level. The nRESET signal is input through a Schmitt trigger circuit where it is then synchronized with the CPU clock. This procedure brings S3CC11B into a known operating status.

For the time for CPU clock oscillation to stabilize, the nRESET 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 disabled.
- All Ports are set to input 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 00000H.
- When the programmed oscillation stabilization time interval has elapsed, the instruction stored in 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 want to use the basic timer watchdog function, you can enable it by writing some value other than '1010 0101b' to the WDTEN register.

## NOTES

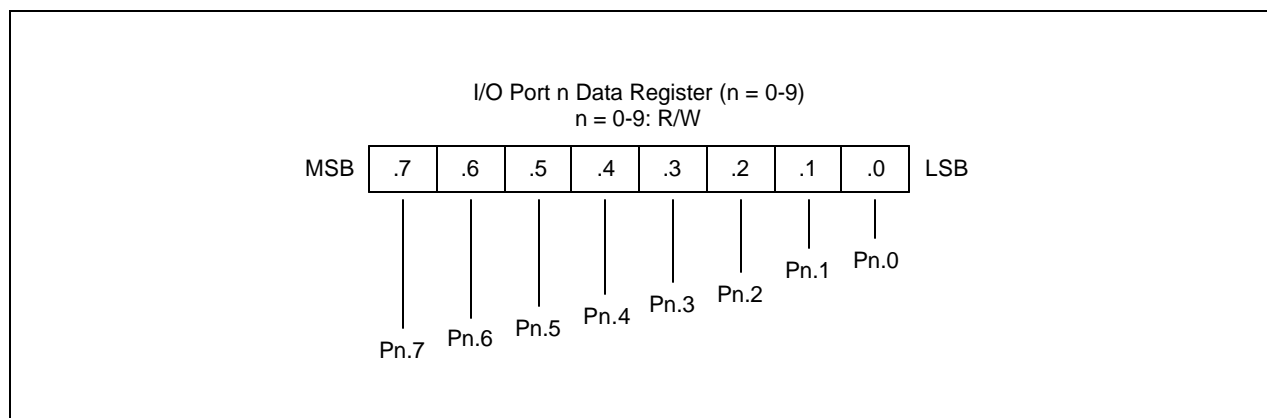
# 9 I/O PORTS

## PORT DATA REGISTERS

All ten port data registers have the identical structure shown in Figure 9-1 below:

**Table 9-1. Port Data Register Summary**

Register Name	Mnemonic	Address	Reset Value	R/W
Port 0 Data Register	P0	3F0010H	00H	R/W
Port 1 Data Register	P1	3F0011H	00H	R/W
Port 2 Data Register	P2	3F0012H	00H	R/W
Port 3 Data Register	P3	3F0013H	00H	R/W
Port 4 Data Register	P4	3F0014H	00H	R/W
Port 5 Data Register	P5	3F0015H	00H	R/W
Port 6 Data Register	P6	3F0016H	00H	R/W
Port 7 Data Register	P7	3F0017H	00H	R/W
Port 8 Data Register	P8	3F0018H	00H	R/W
Port 9 Data Register	P9	3F0019H	00H	R/W



**Figure 9-1. Port Data Register Structure**

**P0CONH — Port 0 Control Register High****3F0020H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.6****P0.7/SCK Configuration Bits**

0	0	Schmitt trigger input(SCK input)
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function(SCK output)

**.5–.4****P0.6/SO Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function(SO output)

**.3–.2****P0.5/SI Configuration Bits2**

0	0	Schmitt trigger input(SI input)
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**.1–.0****P0.4 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**P0CONL — Port 0 Control Register Low****3F0021H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.6****P0.3/INT3 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**.5–.4****P0.2/INT2 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**.3–.2****P0.1/INT1 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**.1–.0****P0.0/INT0 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available



**POPUR — Port 0 Pull-Up Resistors Enable Register****3F0022H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7 P0.7's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.6 P0.6's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.5 P0.5's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.4 P0.4's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.3 P0.3's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.2 P0.2's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.1 P0.1's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.0 P0.0's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**POSTA**— Port 0 Interrupt State Setting Register**3F0023H**

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.4****Bits 7–4**

Not used

**.3****P0.3's Interrupt State Setting Bit**

0	Falling edge interrupt
1	Rising edge interrupt

**.2****P0.2's Interrupt State Setting Bit**

0	Falling edge interrupt
1	Rising edge interrupt

**.1****P0.1's Interrupt State Setting Bit**

0	Falling edge interrupt
1	Rising edge interrupt

**.0****P0.0's Interrupt State Setting Bit**

0	Falling edge interrupt
1	Rising edge interrupt

**P1CON** — Port 1 Control Register**3F0024H**

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	–	–	0	0	0	0	0	0
Read/Write	–	–	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.6****Bits 7–6**

Not used

**.5–.3****P1.7/ I/O7 – P1.4/ I/O4 Configuration Bits**

0	0	0	Schmitt trigger input
0	0	1	Schmitt trigger input; Pull-up resistor enable
0	1	0	Push-pull output
0	1	1	N-channel open-drain output
1	0	0	N-channel open-drain output; Pull-up resistor enable

**.2–.0****P1.3/ I/O3 – P1.0/ I/O0 Configuration Bits**

0	0	0	Schmitt trigger input
0	0	1	Schmitt trigger input; Pull-up resistor enable
0	1	0	Push-pull output
0	1	1	N-channel open-drain output
1	0	0	N-channel open-drain output; Pull-up resistor enable

**NOTE:** When the SmartMedia control(SMCON) register is enabled, the read or write operation for port 1 activate the ECC block and the pull-up resistors should be automatically disabled to reduce current consumption through them. The ECC block capture the data on port 1 access and execute ECC operation.

**P2CONH** — Port 2 Control Register High**3F0028H**

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.6****P2.7/nCE1 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**.5–.4****P2.6/nCE0 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**.3–.2****P2.5/CLE Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**.1–.0****P2.4/ALE Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**NOTE:** When the SmartMedia control(SMCON) register is enabled, the access of port 2 generate the read or write strobe signal to the SmartMedia memory. However, other pins for SmartMedia 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.

**P2CONL — Port 2 Control Register Low****3F0029H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.6****P2.3/ R/nB Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**.5–.4****P2.2/WP Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**.3–.2****P2.1/nRE Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**.1–.0****P2.0//nWE Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**NOTE:** When the SmartMedia control(SMCON) register is enabled, the access of port 2 generate the read or write strobe signal to the SmartMedia memory. However, other pins for SmartMedia 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.

**P2PUR — Port 2 Pull-Up Resistors Enable Register****3F002AH**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

<b>.7</b>	<b>P2.7's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.6</b>	<b>P2.6's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.5</b>	<b>P2.5's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.4</b>	<b>P2.4's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.3</b>	<b>P2.3's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.2</b>	<b>P2.2's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.1</b>	<b>P2.1's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.0</b>	<b>P2.0's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						

**NOTE:** When the SmartMedia control(SMCON) register is enabled, the pull-up resistors should be automatically disabled to reduce current consumption through them.

**P3CONH — Port 3 Control Register High****3F002CH**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.6****P3.7/TBOUT Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (TBOUT output)

**.5–.4****P3.6/TAOUT Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (TAOUT output)

**.3–.2****P3.5/BUZ/T1CLK Configuration Bits**

0	0	Schmitt trigger input (T1CLK input)
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (BUZ output)

**.1–.0****P3.4/T0OUT/T0PWM/T0CAP Configuration Bits**

0	0	Schmitt trigger input (Capture input at T0CAP)
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (T0OUT or T0PWM output)

**P3CONL — Port 3 Control Register Low****3F002DH**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.6****3.3/AD3/T0CLK Configuration Bits**

0	0	Schmitt trigger input (T0CK input)
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (AD3 input)

**.5–.4****P3.2/AD2 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (AD2 input)

**.3–.2****P3.1/AD1 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (AD1 input)

**.1–.0****P3.0/AD0 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative f Alternative function (AD0 input)



**P3PUR — Port 3 Pull-Up Resistors Enable Register****3F002EH**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7****P3.7's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.6****P3.6's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.5****P3.5's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.4****P3.4's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.3****P3.3's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.2****P3.2's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.1****P3.1's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**.0****P3.0's Pull-up Resistor Enable Bit**

0	Disable pull-up resistor
1	Enable pull-up resistor

**P4CONH** — Port 4 Control Register High**3F0030H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.6****P4.7/SEG6 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG6 output)

**.5–.4****P4.6/SEG5 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG5 output)

**.3–.2****P4.5/SEG4 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG4 output)

**.1–.0****P4.4/SEG3 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG3 output)

**P4CONL — Port 4 Control Register Low****3F0031H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.6****P4.3/SEG2/CCLK Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG2 or CCLK output)

**.5–.4****P4.2/SEG1/CFS Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG1 or CFS output)

**.3–.2****P4.1/SEG0/CDX Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG0 or CDX output)

**.1–.0****P4.0/CDR Configuration Bits**

0	0	Schmitt trigger input (CDR input)
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Not available

**P4PUR — Port 4 Pull-Up Resistors Enable Register****3F0032H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

<b>.7</b>	<b>P4.7's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.6</b>	<b>P4.6's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.5</b>	<b>P4.5's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.4</b>	<b>P4.4's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.3</b>	<b>P4.3's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.2</b>	<b>P4.2's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.1</b>	<b>P4.1's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						
<b>.0</b>	<b>P4.0's Pull-up Resistor Enable Bit</b>							
	0	Disable pull-up resistor						
	1	Enable pull-up resistor						

**P4INT — Port 4 Interrupt Control Register****3F0033H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7 P4.7's Interrupt State Setting Bit**

0	Falling edge interrupt
1	Rising edge interrupt

**.6 P4.6's Interrupt State Setting Bit**

0	Falling edge interrupt
1	Rising edge interrupt

**.5 P4.5's Interrupt State Setting Bit**

0	Falling edge interrupt
1	Rising edge interrupt

**.4 P4.4's Interrupt State Setting Bit**

0	Falling edge interrupt
1	Rising edge interrupt

**.3 P4.7's Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

**.2 P4.6's Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

**.1 P4.5's Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

**.0 P4.4's Interrupt Enable Bit**

0	Disable interrupt
1	Enable interrupt

**P5CONH** — Port 5 Control Register High**3F0034H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.6****P5.7/SEG14 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG14 output)

**.5–.4****P5.6/SEG13 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG13 output)

**.3–.2****P5.5/SEG12 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG12 output)

**.1–.0****P5.4/SEG11 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG11 output)

**P5CONL — Port 5 Control Register Low****3F0035H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.6****P5.3/SEG10 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG10 output)

**.5–.4****P5.2/SEG9 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG9 output)

**.3–.2****P5.1/SEG8 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG8 output)

**.1–.0****P5.0/SEG7 Configuration Bits**

0	0	Schmitt trigger input
0	1	Push-pull output
1	0	N-channel open-drain output
1	1	Alternative function (SEG7 output)

**P5PUR — Port 5 Pull-Up Resistors Enable Register****3F0036H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

<b>.7</b>	<b>P5.7's Pull-up Resistor Enable Bit</b>	
0	Disable pull-up resistor	
1	Enable pull-up resistor	
<b>.6</b>	<b>P5.6's Pull-up Resistor Enable Bit</b>	
0	Disable pull-up resistor	
1	Enable pull-up resistor	
<b>.5</b>	<b>P5.5's Pull-up Resistor Enable Bit</b>	
0	Disable pull-up resistor	
1	Enable pull-up resistor	
<b>.4</b>	<b>P5.4's Pull-up Resistor Enable Bit</b>	
0	Disable pull-up resistor	
1	Enable pull-up resistor	
<b>.3</b>	<b>P5.3's Pull-up Resistor Enable Bit</b>	
0	Disable pull-up resistor	
1	Enable pull-up resistor	
<b>.2</b>	<b>P5.2's Pull-up Resistor Enable Bit</b>	
0	Disable pull-up resistor	
1	Enable pull-up resistor	
<b>.1</b>	<b>P5.1's Pull-up Resistor Enable Bit</b>	
0	Disable pull-up resistor	
1	Enable pull-up resistor	
<b>.0</b>	<b>P5.0's Pull-up Resistor Enable Bit</b>	
0	Disable pull-up resistor	
1	Enable pull-up resistor	



**P6CON** — Port 6 Control Register**3F0038H**

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	–	–	–	–	0	0	0	0
Read/Write	–	–	–	–	R/W	R/W	R/W	R/W

**.7–.4****Bits 7–4**

Not used

**.3–.2****P6.7/SEG22 – P6.4/SEG19 Configuration Bits**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; Pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (SEG22 – SEG19 signal output)

**.1–.0****P6.3/SEG18 – P6.0/SEG15 Configuration Bits**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; Pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (SEG18 – SEG15 signal output)

**P7CON** — Port 7 Control Register**3F003AH**

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	–	–	–	–	0	0	0	0
Read/Write	–	–	–	–	R/W	R/W	R/W	R/W

**.7–.4****Bits 7–4**

Not used

**.3–.2****P7.7/SEG30 – P7.4/SEG27 Configuration Bits**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; Pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (SEG30 – SEG27 signal output)

**.1–.0****P7.3/SEG26 – P7.0/SEG23 Configuration Bits**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; Pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (SEG26 - SEG23 signal output)

**P8CON** — Port 8 Control Register**3F003CH**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	–	–	–	–	0	0	0	0
<b>Read/Write</b>	–	–	–	–	R/W	R/W	R/W	R/W

**.7–.4****Bits 7–4**

Not used

**.3–.2****P8.4/SEG35 Configuration Bits**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; Pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (SEG35 signal output)

**.1–.0****P8.3/SEG34 – P8.0/SEG31 Configuration Bits**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; Pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (SEG31 – SEG34 signal output)

**P9CON** — Port 9 Control Register**3F003EH**

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	–	–	0	0	0	0	0	0
Read/Write	–	–	R/W	R/W	R/W	R/W	R/W	R/W

.7–.6

**Bits 7–6**

Not used

.5–.4

**P9.5/COM2 – P9.7/COM0 Configuration Bits**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; Pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (COM0 – COM2 signal output)

.3–.2

**P9.4/COM3 Configuration Bits**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; Pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (COM3 signal output)

.1–.0

**P9.0/COM7 – P9.3/COM4 Configuration Bits**

0	0	Schmitt trigger input
0	1	Schmitt trigger input; Pull-up resistor enable
1	0	Push-pull output
1	1	Alternative function (COM4 – COM7 signal output)

**NOTES**

# 10 BASIC TIMER

## OVERVIEW

The basic timer's primary function is to measure a predefined time interval. The standard time interval is equal to 256 basic clock pulses and the period of a clock pulse can be selected by basic timer control register.

The 8bit counter register, BTCNT, is increased each time the clock signal, which can be selected by the clock signal selection field in basic timer control register, is detected. BTCNT will increase until an overflow occurs. An overflow internally sets an interrupt pending flag to signal that the predefined time has elapsed. An interrupt request (BTINT) is then generated, BTCNT is cleared to zero, and the counting continues from 00h.

### Oscillation Stabilization Interval Timer Function

You can also use the basic timer to program a specific oscillation stabilization interval after a reset or when STOP mode is released by an external interrupt.

In STOP mode, whenever a reset or an external interrupt occurs, the oscillator starts and releases the CPU from STOP mode to normal mode. The BTCNT value then starts increasing at the rate of  $f_{OSC}/2048$  (for reset), or at the rate of the preset clock source (for an external interrupt). When BTCNT is increased to 80h, basic timer generates CPU start signal to indicate that the stabilization interval has elapsed, gating the clock signal on to the CPU so that it can resume its normal operation.

In summary, the following events occur during the STOP mode release:

1. We assume that, in STOP mode, a power-on reset or an external interrupt occurs to trigger a STOP mode release and oscillation starts.
2. If a power-on reset occurs, the BTCNT will increase at the rate of  $f_{OSC}/2048$ . If an external interrupt is used to release the STOP mode, the BTCNT value increases at the rate of the preset clock source.
3. Clock oscillation stabilization interval begins and continues until BTCNT becomes 80h.
4. When a BTCNT is 10h, the CPU start signal is generated and the normal CPU operation resumes.

### Watchdog Timer Function

The basic timer can also be used as a "watchdog" timer to detect inadvertent program loops, i.e. infinite loop, by system or program operation errors. For this purpose, instructions that clear the watchdog timer counter register within a given period should be executed at proper points in a program. If an instruction that clears the watchdog timer counter register is not executed within the period and the watchdog timer overflows, a reset signal is generated so that the system will be restarted. Operations of a watchdog timer are as follows:

1. Each time BTCNT overflows, the overflow signal is sent to the watchdog timer counter, WTCNT.
2. If WDCNT overflows, a system-reset signal is generated.

As the reset signal sets WDTCON as A5H and this value disables the watchdog timer.

**BTCON** — Basic Timer Control Register

3F000CH

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	–	1	1	1	–	–	0	0
<b>Read/Write</b>	–	R/W	R/W	R/W	–	–	R/W	R/W

**.7**

Not used
----------

**.6–.4** **Basic Timer Clock Selection Bits**

0	0	0	fxx/2
0	0	1	fxx/4
0	1	0	fxx/16
0	1	1	fxx/32
1	0	0	fxx/128
1	0	1	fxx/256
1	1	0	fxx/1024
1	1	1	fxx/2048

**.3–.2** **Bits 3–2**

Not used
----------

**.1** **Basic Timer Counter Clear Bit**

0	Don't care
1	Clear basic timer counter

**.0** **Watchdog Timer Counter Clear Bit**

0	Don't care
1	Clear watchdog timer counter

**WDTEN** — Watch-Dog Timer Enable Register**3F000EH**

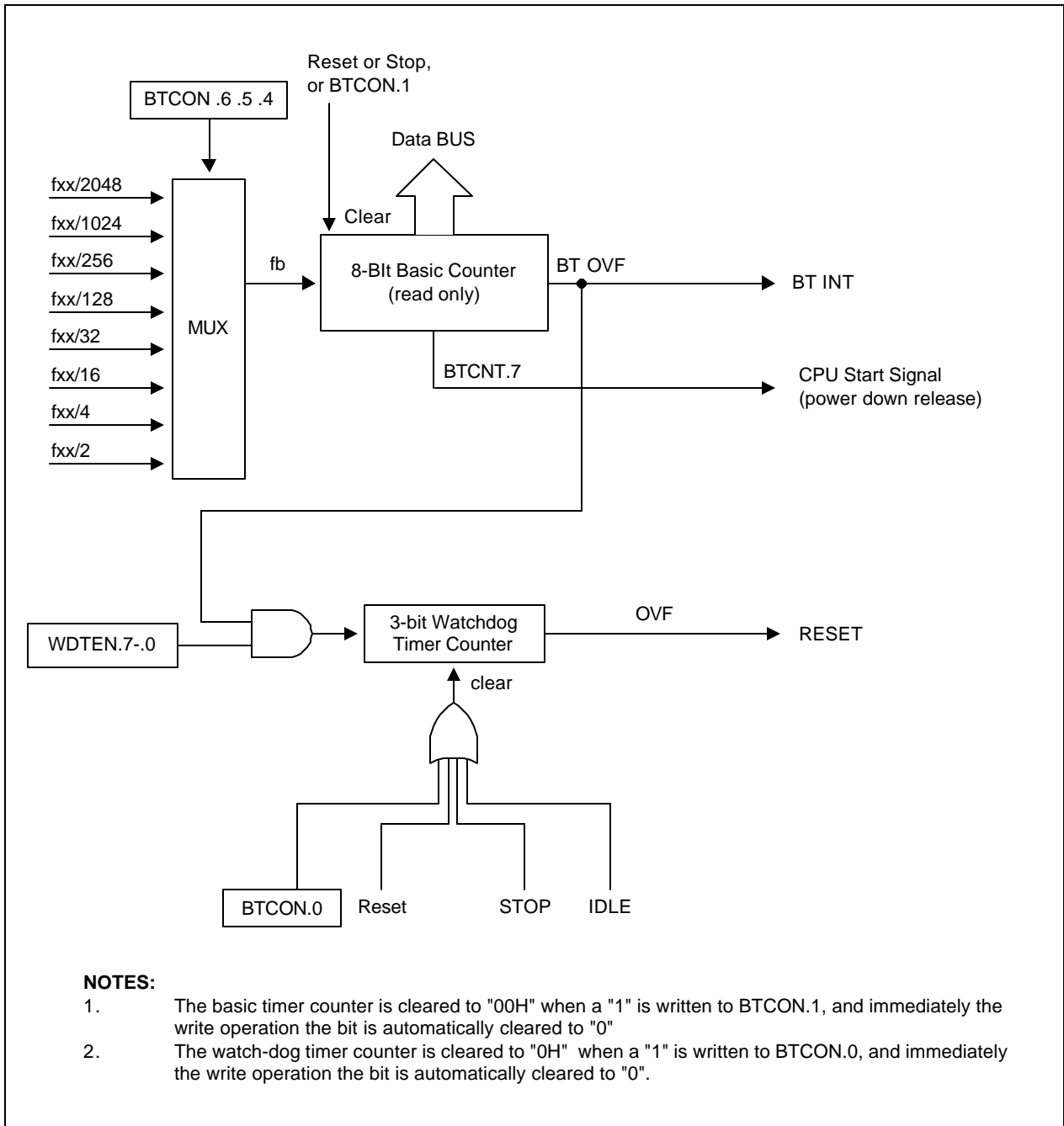
<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7–.0****Watch-dog Timer Enable Bits**

10100101	Disable watch-dog timer
Other values	Enable watch-dog timer



**BASIC TIMER & WATCHDOG TIMER BLOCK DIAGRAM**



**Figure 10-1. Basic Timer & Watchdog Timer Block Diagram**

# 11

## WATCH TIMER

### OVERVIEW

Watch timer functions include read time and watch-time measurements. After the watch timer starts and time elapses, the watch timer interrupt is automatically set to "1", and interrupt requests commence in 3.91ms, 0.25s, 0.5s or 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[3:2] 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

**WTCON** — Watch Timer Control Register**3F0070H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	–	0	0	0	0	0	0	0
<b>Read/Write</b>	–	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7****Bit 7**

Not used

**.6****Watch Timer Clock Source Selection Bit (When WTCON.1 = "0" Only)**

0	fxin/128
1	fxin/64

**.5–.4****Buzzer Signal Selection Bits**

0	0	0.5 kHz
0	1	1 kHz
1	0	2 kHz
1	1	4 kHz

**.3–.2****Watch Timer Speed Selection Bits**

0	0	Set watch timer interrupt to 1s
0	1	Set watch timer interrupt to 0.5s
1	0	Set watch timer interrupt to 0.25s
1	1	Set watch timer interrupt to 3.91ms

**.1****Watch Timer Clock Selection Bit**

0	Select clock divided by $2^6$ or $2^7$ (fxin/64 or fxin/128)
1	Select sub clock (fxt)

**.0****Watch Timer Enable Bit**

0	Disable watch timer; Clear frequency dividing circuits
1	Enable watch timer

WATCH TIMER BLOCK DIAGRAM

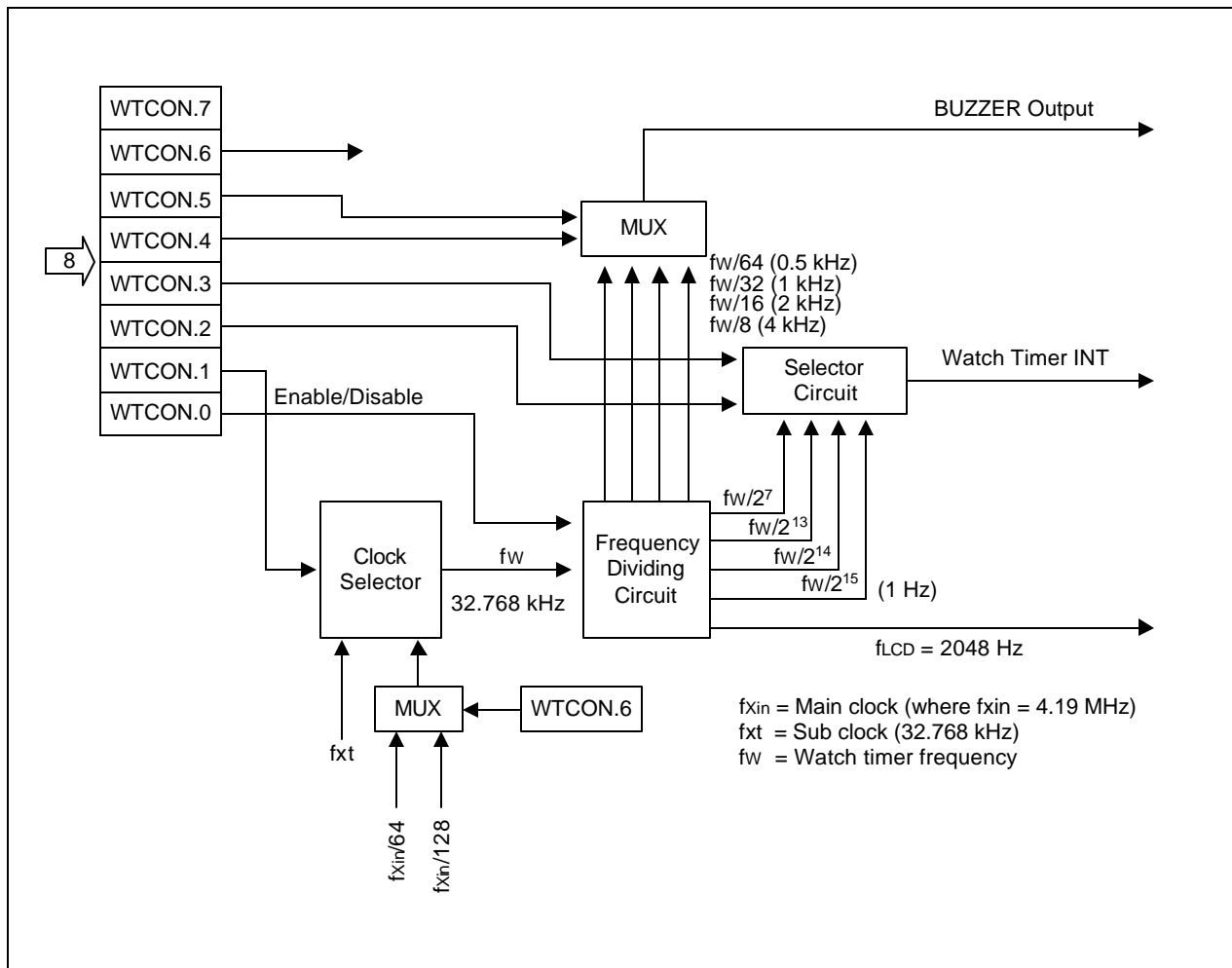


Figure 11-1. Watch Timer Block Diagram

**NOTES**

# 12

## 8-BIT TIMER 0

### OVERVIEW

The 8-bit timer 0 is an 8-bit general-purpose timer/counter. Timer 0 has three operating modes, one of which you select using the appropriate T0CON setting:

- Interval timer mode (Toggle output at T0 pin)
- Capture input mode with a rising or falling edge trigger at the T0CAP pin
- PWM mode (TOPWM)

## FUNCTION DESCRIPTION

### Timer 0 Interrupts

The Timer 0 module can generate two interrupts: the Timer 0 overflow interrupt (T0OVF), and the Timer 0 match/capture interrupt (T0INT).

### Interval Timer Function

The Timer 0 module can generate an interrupt: the Timer 0 match interrupt (T0INT).

In interval timer mode, a match signal is generated(,) and T0 is toggled when the counter value is identical to the value written to the T0 reference data register, T0DATA. The match signal generates a Timer 0 match interrupt and clears the counter.

If, for example, you write the value 10H to T0DATA and 0AH to T0CON, the counter will increment until it reaches 10H. At this point, the T0 interrupt request is generated and the counter value is reset and counting resumes.

### Pulse Width Modulation Mode

Pulse width modulation (PWM) mode lets you program the width (duration) of the pulse that is output at the T0PWM pin. As in interval timer mode, a match signal is generated when the counter value is identical to the value written to the Timer 0 data register. In PWM mode, however, the match signal does not clear the counter but can generate a match interrupt. The counter runs continuously, overflowing at FFH, and then repeats the incrementing from 00H. Whenever an overflow occurs, an overflow(OVF) interrupt can be generated.

Although you can use the match or the overflow interrupt in PWM mode, interrupts are not typically used in PWM-type applications. Instead, the pulse at the T0PWM pin is held to High level as long as the reference data value is less than or equal to ( $\leq$ ) the counter value, and then the pulse is held to Low level for as long as the data value is greater than ( $>$ ) the counter value. One pulse width is equal to  $t_{CLK} \times 256$ .

### Capture Mode

In capture mode, a signal edge that is detected at the T0CAP pin opens a gate and loads the current counter value into the T0 data register. You can select the rising or falling edges to trigger this operation.

Timer 0 also gives you capture input source: the signal edge at the T0CAP pin. You select the capture input by setting the value of the Timer 0 capture input selection bit in the port control register.

Both kinds of Timer 0 interrupts can be used in capture mode: the Timer 0 overflow interrupt is generated whenever a counter overflow occurs; the Timer 0 match/capture interrupt is generated whenever the counter value is loaded into the T0 data register.

By reading the captured data value in T0DATA and assuming a specific value for the Timer 0 clock frequency, you can calculate the pulse width (duration) of the signal that is being input at the T0CAP pin.

**TIMER 0 CONTROL REGISTER (T0CON)**

You use the Timer 0 control register, T0CON, to

- Select the Timer 0 operating mode (interval timer, capture mode, or PWM mode)
- Select the Timer 0 input clock frequency
- Clear the Timer 0 counter, T0CNT
- Enable the Timer 0 counter

A reset clears T0CON to '00H'. This sets Timer 0 to normal interval timer mode, selects an input clock frequency of  $f_{OSC}/4096$ , and disables Timer 0 counting operation. You can clear the Timer 0 counter at any time during normal operation by writing a "1" to T0CON.3.



BLOCK DIAGRAM

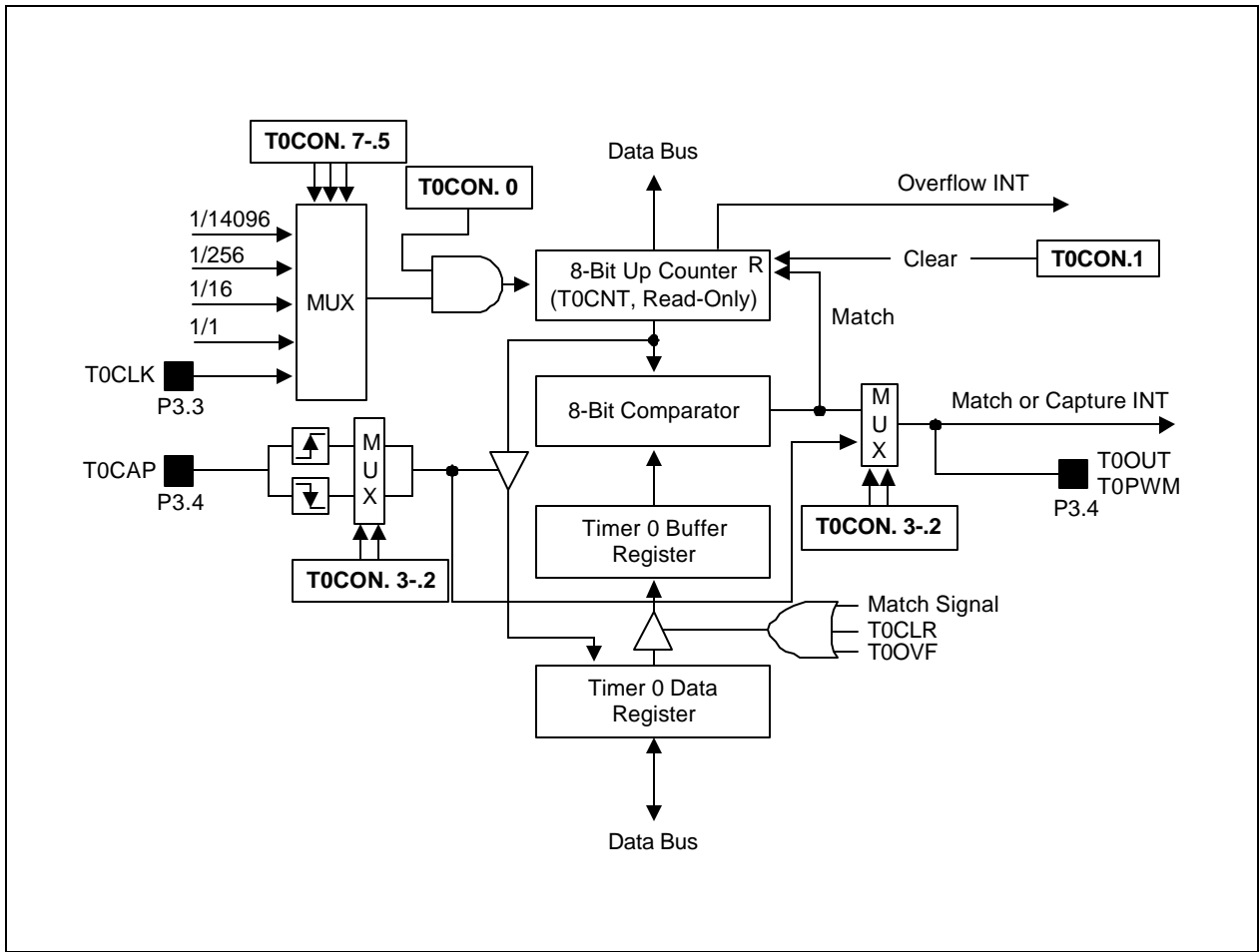


Figure 12-1. Timer 0 Functional Block Diagram

**T0CON** — Timer 0 Control Register**3F0042H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	–	0	0	0	0
<b>Read/Write</b>	R/W	R/W	R/W	–	R/W	R/W	R/W	R/W

**.7–.5****Timer 0 Clock Selection Bits**

0	0	0	fx/4096
0	0	1	fx/256
0	1	0	fx/16
0	1	1	fx/1
1	0	0	External clock (at T0CLK pin)
Other values			Not used for S3FC11B

**.4****Bit 4**

fxin/128
----------

**.3–.2****Timer 0 Operating Mode Selection Bits**

0	0	Interval mode
0	1	Capture mode (capture on rising edge, counter running, OVF)
1	0	Capture mode (capture on falling edge, counter running, OVF)
1	1	PWM mode (OVF interrupt can occur)

**.1****Timer 0 Counter Clear Bit**

0	No effect
1	Clear the timer 0 counter (when write)

**.0****Timer 0 Counter Enable Bit**

0	Disable counting operation
1	Enable counting operation

**NOTES**

# 13

## 16-BIT TIMER 1 (8-BIT TIMER A & B)

### OVERVIEW

The 16-bit timer 1 is used in one 16-bit timer or two 8-bit timers. When Bit 2 of TBCON is "1", it operates as one 16-bit timer. When it is "0", it operates as two 8-bit timers. When it operates as one 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 the S/W.

### INTERVAL TIMER FUNCTION

The timer A&B module can generate an interrupt: the Timer A and/or Timer B match interrupt (TAINT, TBINT). In interval timer mode, a match signal is generated when the counter value is identical to the value written to the reference data register, TADATA/TBDATA. The match signal generates Timer A and/or Timer B match interrupt and clears the counter.

TB pin can be toggled whenever the timer B match interrupt occurs if I/O port setting is appropriate.

**TACON** — Timer 1/A Control Register**3F0048H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	–	0	0	0	–	–	0	0
<b>Read/Write</b>	–	R/W	R/W	R/W	–	–	R/W	R/W

**.7****Bit 7**

Not used

**.6–.4****Timer 1/A Clock Selection Bits**

0	0	0	fxx/4096
0	0	1	fxx/512
0	1	0	fxx/64
0	1	1	fxx/8
1	0	0	fxx (system clock)
1	0	1	fxt (sub clock)
1	1	0	T1CLK (external clock)
1	1	1	Not used for S3FC11B

**.3–.2****Bit 3–2**

Not used

**.1****Timer 1/A Counter Clear Bit**

0	No effect
1	Clear the timer 1/A counter (when write)

**.0****Timer 1/A Counter Enable Bit**

0	Disable counting operation
1	Enable counting operation

**TBCON** — Timer B Control Register**3F0049H**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	–	0	0	0	0	0	0	0
<b>Read/Write</b>	–	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7****Bit 7**

Not used
----------

**.6–.4****Timer B Clock Selection Bits**

0	0	0	fxx/4096
0	0	1	fxx/512
0	1	0	fxx/64
0	1	1	fxx/8
1	0	0	fxt (sub clock)
Other values			Not used for S3FC11B

**.3****16-Bit Operation Timer B Clock Input Selection Bit**

0	Timer A overflow out
1	Timer A interval out

**.2****Timer B Mode Selection Bit**

0	8-bit operation mode
1	16-bit operation mode

**.1****Timer B Counter Clear Bit**

0	No effect
1	Clear the timer B counter (when write)

**.0****Timer B Counter Enable Bit**

0	Disable counting operation
1	Enable counting operation

BLOCK DIAGRAM

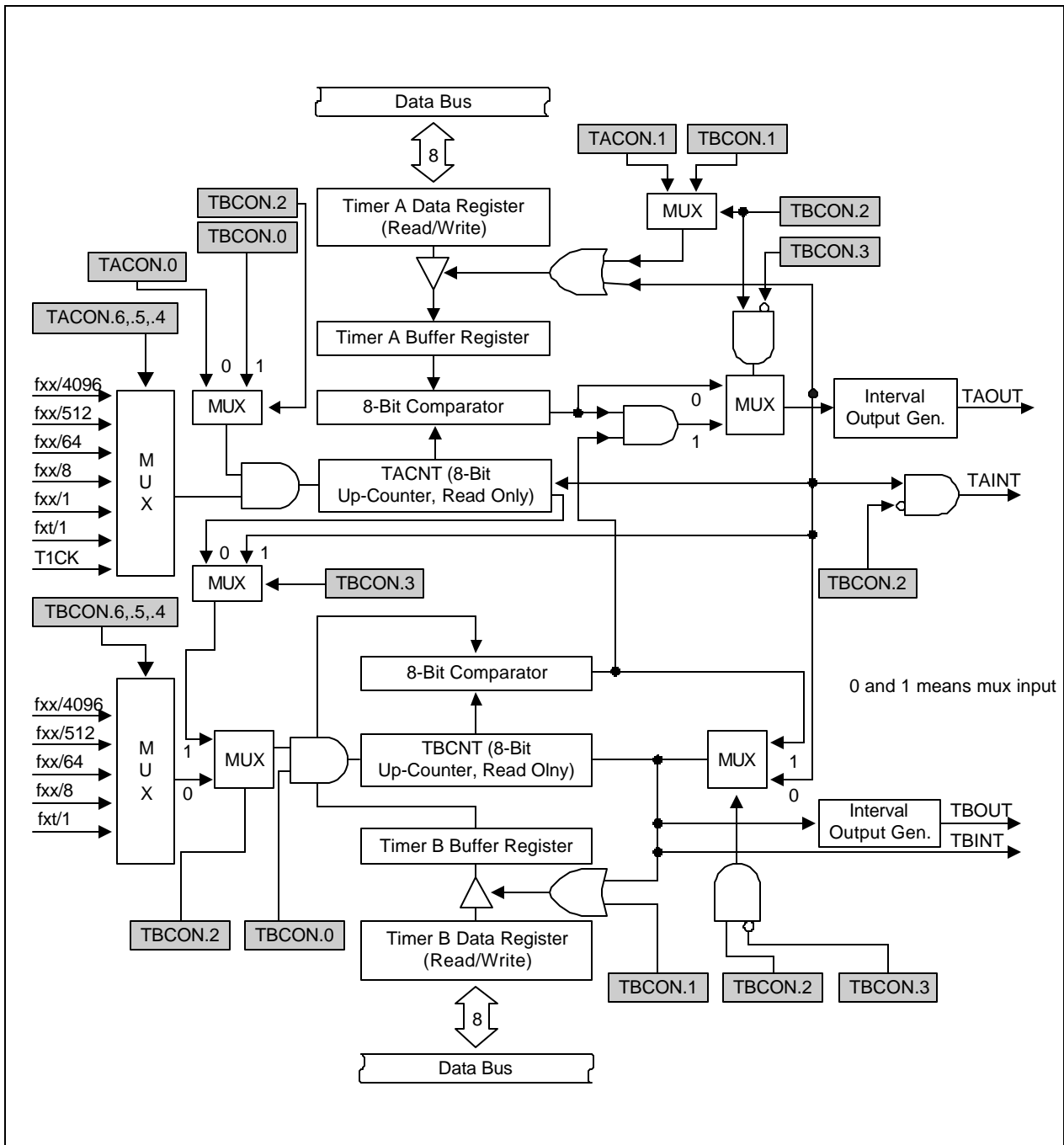


Figure 13-1. Timer 1 Block Diagram

# 14 SERIAL I/O INTERFACE

## OVERVIEW

The SIO module can transmit or receive 8-bit serial data at a frequency determined by its corresponding control register settings. To ensure flexible data transmission rates, you can select an internal or external clock source.

## PROGRAMMING PROCEDURE

To program the SIO modules, follow these basic steps:

1. Configure the I/O pins at port (SO,nSCK, SI) by loading the appropriate value to the P0CONH register, if necessary.
2. Load an 8-bit value to the SIOCON register to properly configure the serial I/O module. In this operation, SIOCON.2 must be set to "1" to enable the data shifter.
3. When you transmit data to the serial buffer, write data to SIODATA and set SIOCON.3 to 1, the shift operation starts.
4. When the shift operation (transmit/receive) is completed, the SIO pending bit is set to "1", and a SIO interrupt request is generated.



**SIOCON** — Serial I/O Control Register**3F006EH**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	0	0	0	0	0	0	–	–
<b>Read/Write</b>	R/W	R/W	R/W	R/W	R/W	R/W	–	–

**.7****Serial I/O Shift Clock Selection Bit**

0	Internal clock
1	External clock(SCK)

**.6****Data Direction Control Bit**

0	MSB-first mode
1	LSB-first mode

**.5****Serial I/O Mode Selection Bit**

0	Receive-only mode
1	Transmit/receive mode

**.4****Tx/Rx Edge Selection Bit**

0	Tx at falling edges, Rx at rising edges
1	Rx at falling edges, Tx at rising edges

**.3****Serial I/O Counter Clear and Shift Start Bit**

0	No effect
1	Clear 3-bit counter and start shifting (This bit is automatically cleared to logic zero immediately after starting shift)

**.2****Serial I/O Shift Operation Enable Bit**

0	Disable shifter and clock counter
1	Enable shifter and clock counter

**.1–0****Bits 1–0**

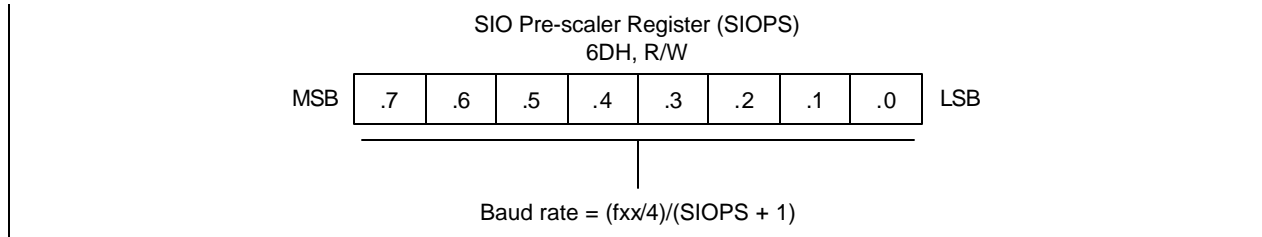
Not used
----------

**SIO PRE-SCALER REGISTER (SIOPS)**

The values stored in the SIO pre-scaler registers, SIOPS, lets you determine the SIO clock rate (baud rate) as follows:

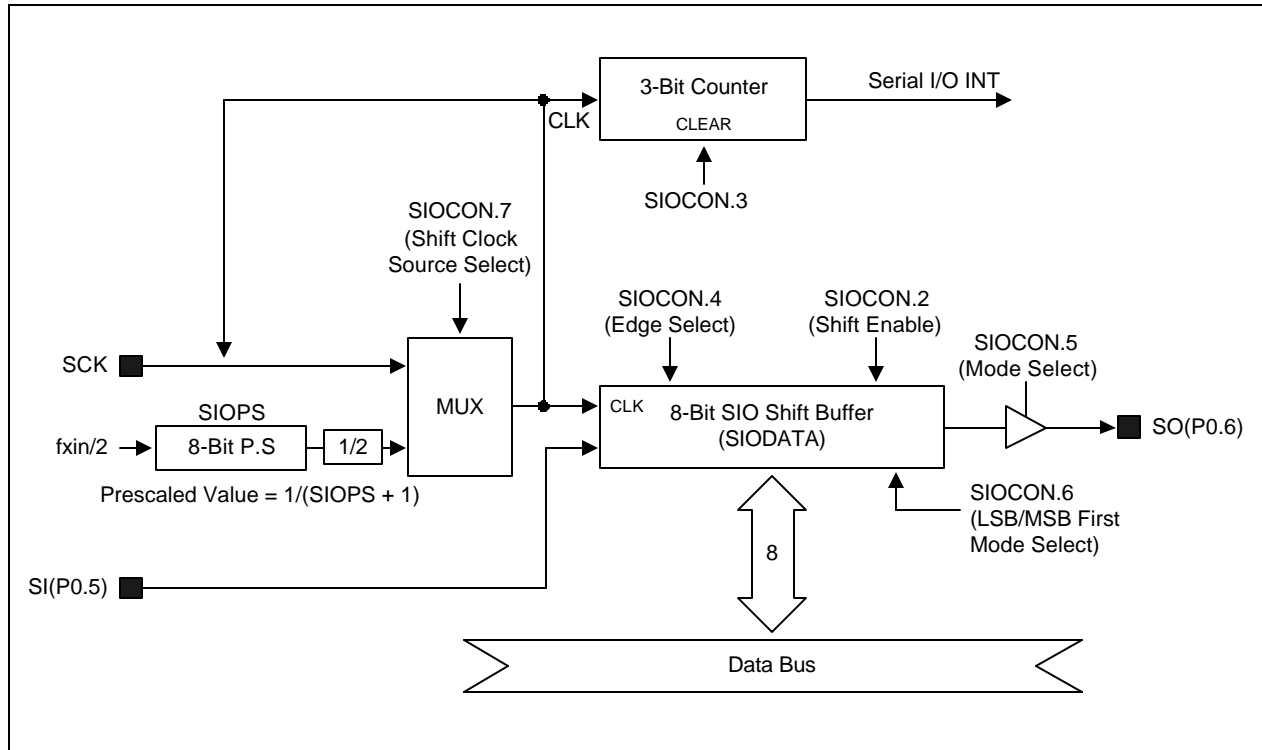
$$\text{Baud rate} = \text{Input clock}/(\text{Pre-scaler value} + 1), \text{ or SCLK input clock}$$

where the input clock is  $fx/4$



**Figure 14-1. SIO Pre-scaler Register (SIOPS)**

**BLOCK DIAGRAM**



**Figure 14-2. SIO Functional Block Diagram**

SERIAL I/O TIMING DIAGRAMS

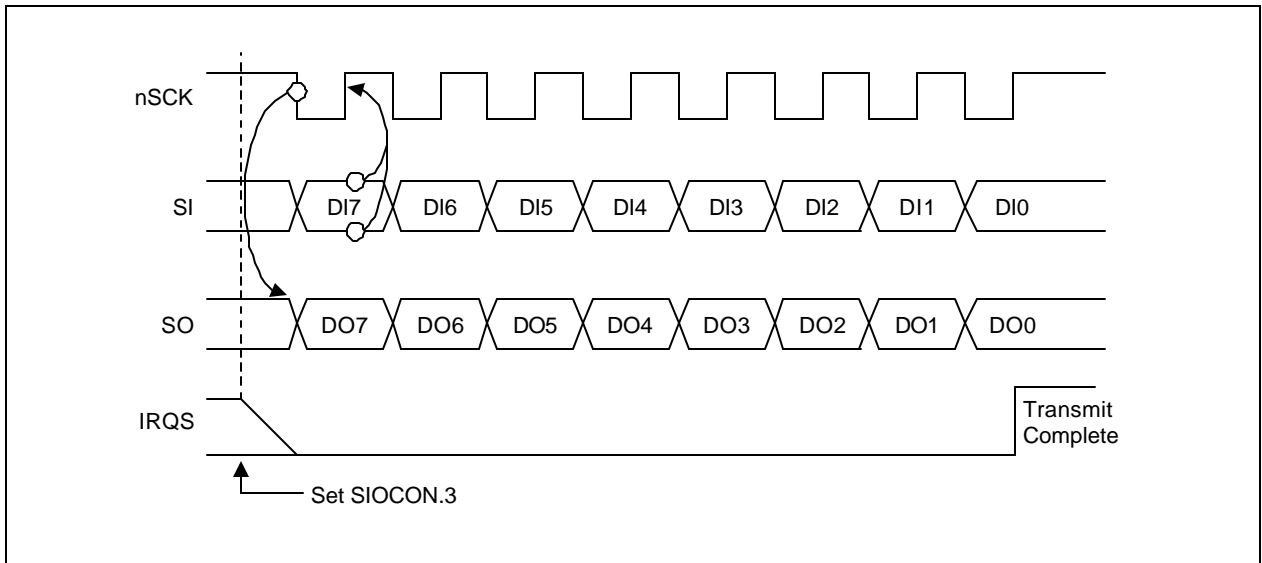


Figure 14-3. Serial I/O Timing in Transmit/Receive Mode (Tx at falling, SIOCON.4 = 0)

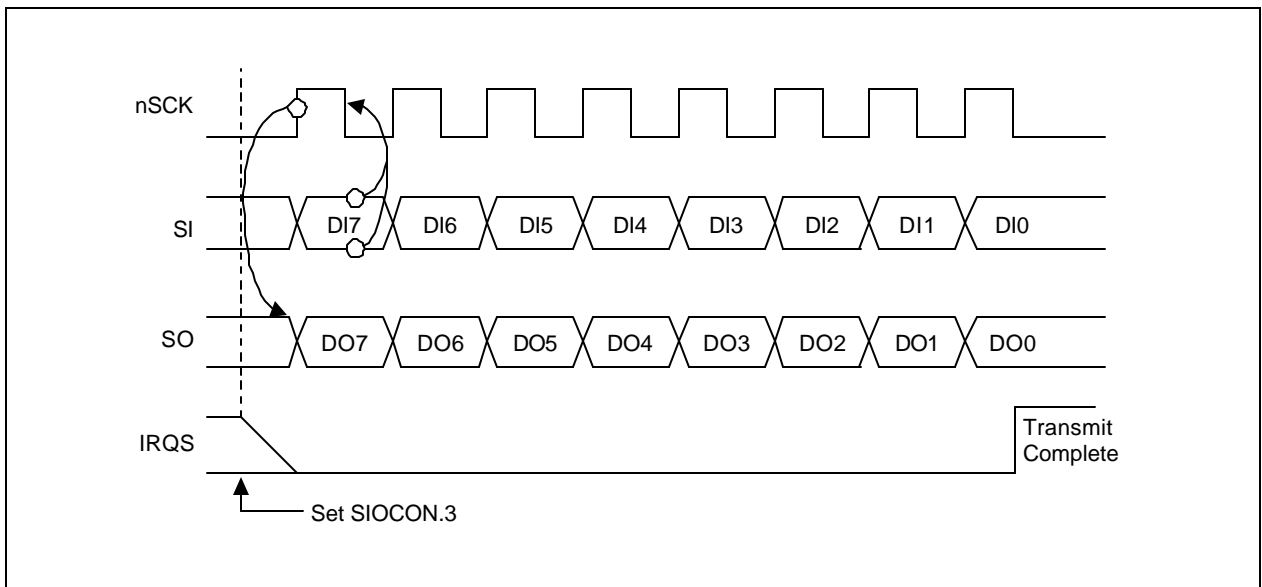


Figure 14-4. Serial I/O Timing in Transmit/Receive Mode (Tx at rising, SIOCON.4 = 1)

# 15

## SSFDC (SOLID STATE FLOPPY DISK CARD)

### OVERVIEW

S3CC11B/FC11B build interface logic for SmartMedia™ card, called as SSFDC, solid state floppy disk card. The SSFDC interface includes 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, CalmRISC16 supports 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 provides the balanced cost and power consumption without the de-gradation 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.

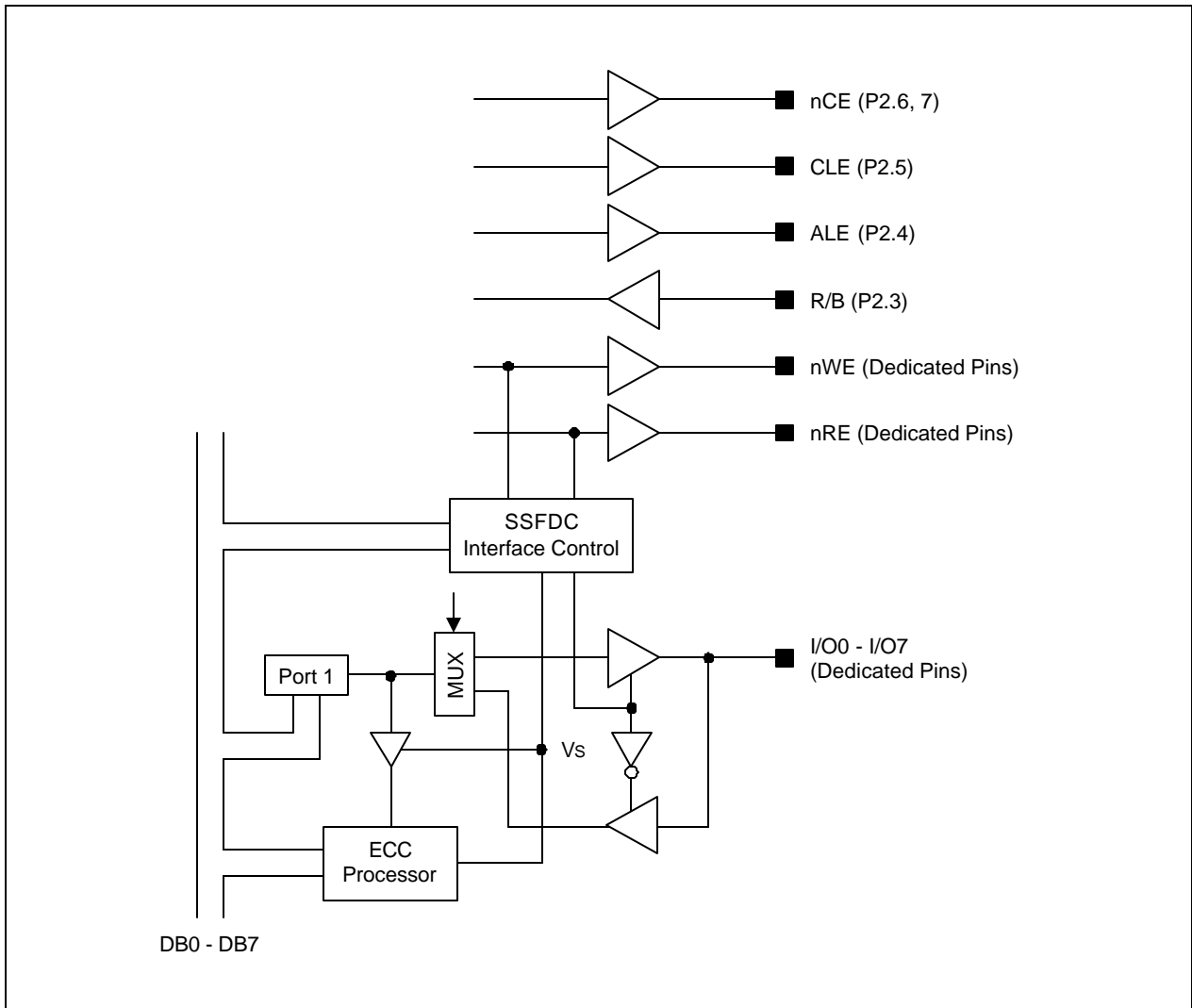


Figure 15-1. Simple System Configuration

**SSFDC REGISTER DESCRIPTION**

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

**Table 15-1. Control Register Description**

Register	Address	R/W/C	Description
SMCON	3F0058H	R/W	SmartMedia control register
ECCNT	3F0059H	R/W	ECC count register
ECCH/L/X	3F005AH 3F005BH 3F005CH	R/W	ECC data register high/low/extension
ECCCLR	3F005DH	W	ECC clear register
ECCRSTH/L	3F005EH 3F005FH	R/W	ECC result data register low/high

**SMARTMEDIA CONTROL REGISTER (SMCON)**

Register	Address	R/W	Description	Reset Value
SMCON	3F0058H	R/W	SmartMedia control register	00H

- [0] ECC Enable      This bit enables or disables 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 1.  
"1": Enable    "0": Disable.
- [1] Enable SmartMedia interface      This bit controls the operation of SmartMedia block. When this bit is set as "1", Port 1 is activated as I/O data bus of SmartMedia interface. SmartMedia control signal is generated whenever accessing the Port 1.
- [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: 4 wait in nWE or nRD signal  
11: 8 wait in nWE or nRD signal

**SMARTMEDIA ECC COUNT REGISTER (ECCNT)**

Register	Address	R/W	Description	Reset Value
ECCNT	3F0059H	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	3F005CH	R/W	SmartMedia ECC data extension register	00h
ECCH	3F005AH	R/W	SmartMedia ECC data high register	00h
ECCL	3F005BH	R/W	SmartMedia ECC data low register	00h

[7:0] Data Data field acts as ECC data register when SMCON.0, *Enable* bit is set. The access instruction to Port 1 executes a 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
ECCRSTH	3F005EH	R/W	SmartMedia ECC result data register high	00h
ECCRSTL	3F005FH	R/W	SmartMedia ECC result data register low	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[4:3] 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.

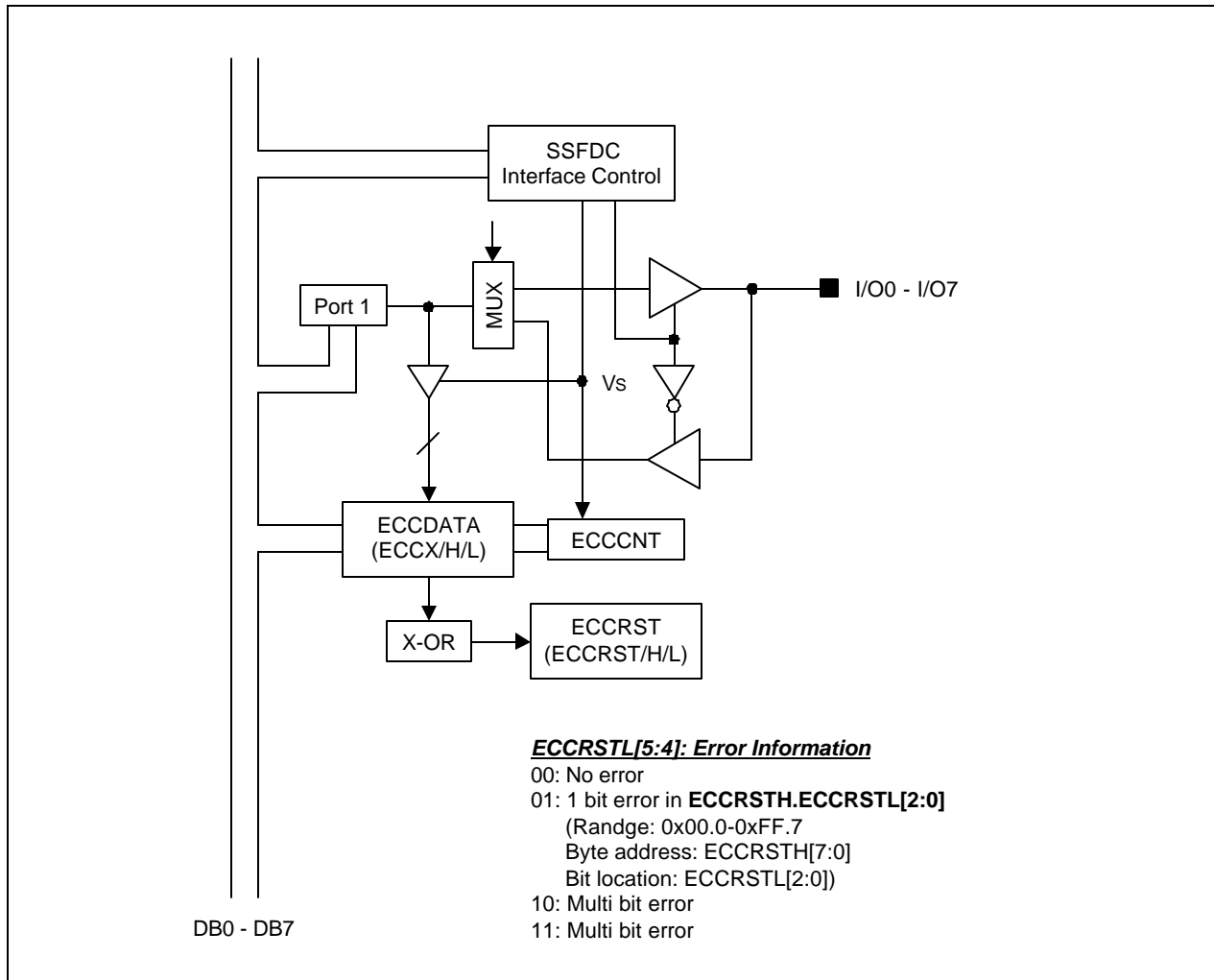


Figure 15-2. ECC Processor Block Diagram



## NOTES

# 16

## 10-BIT ANALOG-TO-DIGITAL CONVERTER

### OVERVIEW

The 10-bit A/D converter (ADC) module uses successive approximation logic to convert analog levels entering at one of the eight input channels to equivalent 10-bit digital values. 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 (ADCON10)
- Four multiplexed analog data input pins (AD0–AD3)
- 10-bit A/D conversion data output register (ADDATAH10/ADDATAL10)

### FUNCTION DESCRIPTION

To initiate an analog-to-digital conversion procedure, at first you must set with alternative function for ADC input enable at port 3, the pin set with alternative function can be used for ADC analog input. And you write the channel selection data in the A/D converter control register ADCON10.4–5 to select one of the eight analog input pins (AD0–AD3) and set the conversion start or enable bit, ADCON10.0. The read-write ADCON10 register is located in address 51H. The pins which are not used for ADC can be used for normal I/O or T0CLK signal.

During a normal conversion, ADC logic initially sets the successive approximation register to 800H (the approximate half-way point of an 10-bit register). This register is then updated automatically during each conversion step. The successive approximation block performs 10-bit conversions for one input channel at a time. You can dynamically select different channels by manipulating the channel selection bit value (ADCON10.5–4) in the ADCON10 register. To start the A/D conversion, you should set the enable bit, ADCON10.0. When a conversion is completed, ADCON10.3, the end-of-conversion(EOC) bit is automatically set to 1 and the result is dumped into the ADDATAH10/ADDATAL10 register where it can be read. The A/D converter then enters an idle state. Remember to read the contents of ADDATAH10/ADDATAL10 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 AD0–AD3 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 ADC operation is finished.

## CONVERSION TIMING

The A/D conversion process requires 4 steps (4 clock edges) to convert each bit and 10 clocks to set-up A/D conversion. Therefore, total of 50 clocks are required to complete an 10-bit conversion: When fxx/8 is selected for conversion clock with an 4.5 MHz fxx clock frequency, one clock cycle is 1.78 us. Each bit conversion requires 4 clocks, the conversion rate is calculated as follows:

$$4 \text{ clocks/bit} \times 10\text{-bit} + \text{set-up time} = 50 \text{ clocks}, 50 \text{ clock} \times 1.78 \text{ us} = 89 \text{ us at } 0.56 \text{ MHz (4.5 MHz/8)}$$

Note that A/D converter needs at least 25μs for conversion time.

## A/D CONVERTER CONTROL REGISTER (ADCON10)

The A/D converter control register, ADCON10, is located at address 51H. It has three functions:

- Analog input pin selection (bits 4–5)
- End-of-conversion status detection (bit 3)
- ADC clock selection (bits 2 and 1)
- A/D operation start or disable (bit 0)

After a reset, the start bit is turned off. You can select only one analog input channel at a time. Other analog input pins (AD0–AD3) can be selected dynamically by manipulating the ADCON10.4–.5 bits. And the pins not used for analog input can be used for normal I/O or T0CLK function.

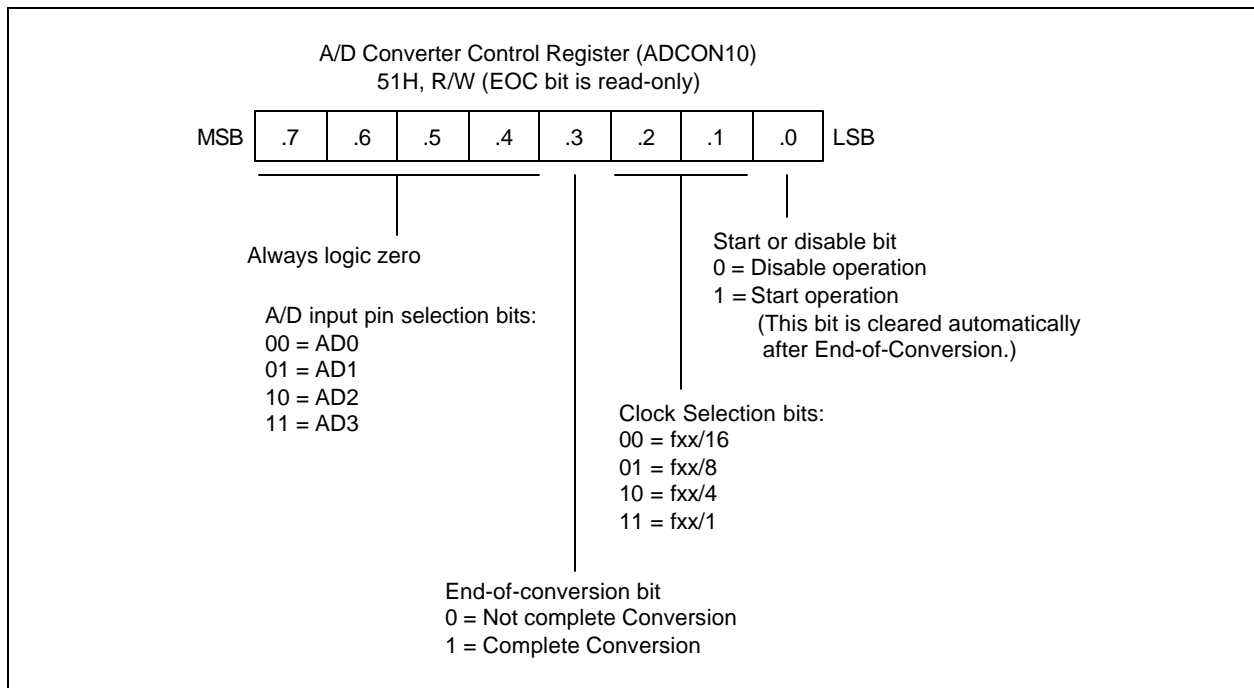


Figure 16-1. A/D Converter Control Register (ADCON10)

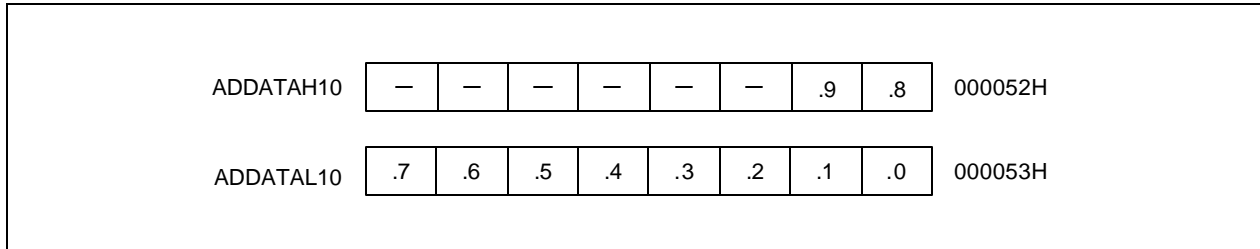


Figure 16-2. A/D Converter Data Register (ADDATAH10/ADDATAL10)

**INTERNAL REFERENCE VOLTAGE LEVELS**

In the ADC function block, the analog input voltage level is compared to the reference voltage. The analog input level must remain within the range  $AV_{SS}$  to  $AV_{REF}$ .

Different reference voltage levels are generated internally along the resistor tree during the analog conversion process for each conversion step. The reference voltage level for the first conversion bit is always  $1/2 AV_{REF}$ .

**BLOCK DIAGRAM**

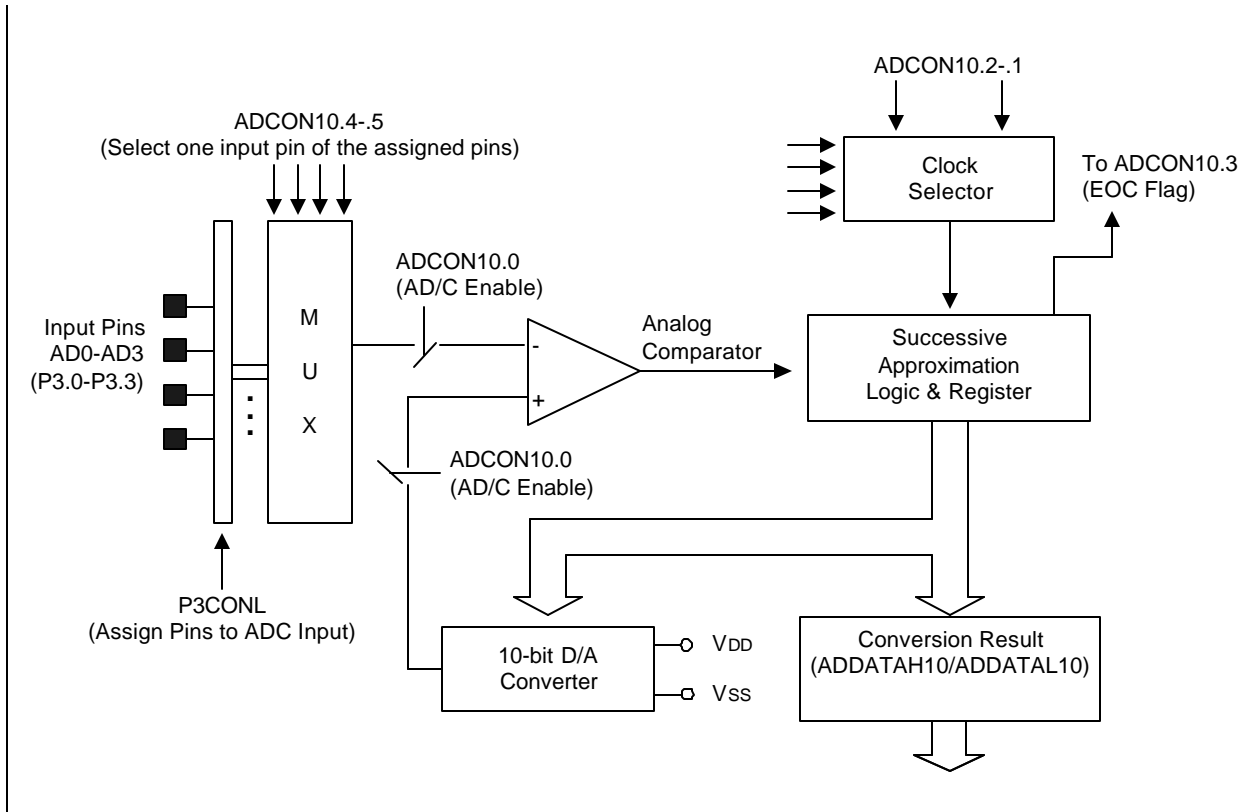


Figure 16-3. A/D Converter Functional Block Diagram

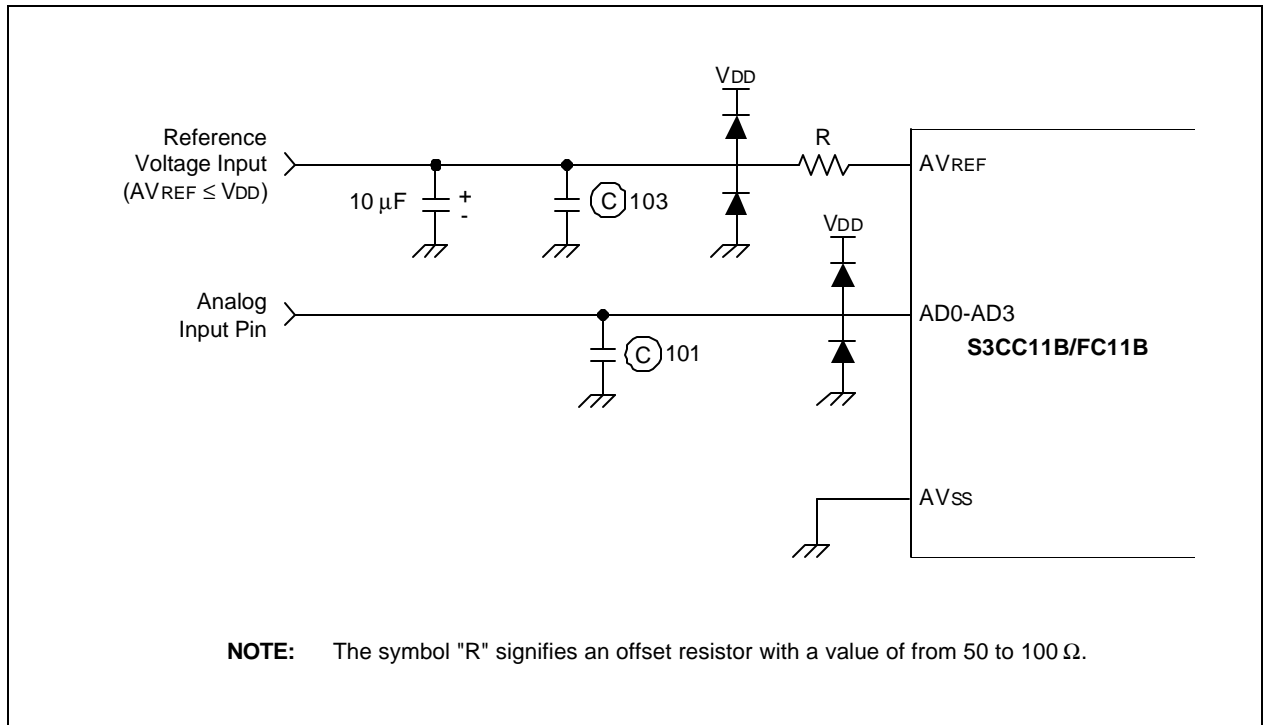


Figure 16-4. Recommended A/D Converter Circuit for Highest Absolute Accuracy

# 17

## CODEC

### OVERVIEW

The CODEC is Sigma-Delta type ADC for speech and telephony applications. The CODEC contains both digital IIR/FIR filters, and an on-chip voltage reference circuit is included to allow supply operations.

### FEATURES

- 256X oversampling
- On chip decimation filter for ADC
- On chip interpolation filter for DAC

**CODEC CONTROL REGISTER (CDCON)**

User can select the CODEC input clock for dividing higher crystal by controlling CDCON.

A/D converted data are 14-bit resolution and are input to ADDATAH (High byte), ADDATAL (Low byte) in 16-bit data format also, D/A converted data are 14-bit resolution and are input to DADATAH (high byte), DADATAL (low byte) in 16-bit data format. Because CODEC use 256X over-sampling, for 8 kHz sampling, when crystal is 2.048 MHz (= 8 kHz × 256), user must select fx as CODEC input clock.

And when crystal is 4.096 MHz (= 2 × 8 kHz × 256), user must select fx/2 as CODEC input clock.

**CDCON** — CODEC Control Register**3F0064H**

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	0	0	0	0	–	0	0	0
Read/Write	R/W	R/W	R/W	R/W	–	R/W	R/W	R/W

**.7****A/D Converter Enable Bit**

0	Disable A/D converter
1	Enable A/D converter

**.6****D/A Converter Enable Bit**

0	Disable D/A converter
1	Enable D/A converter

**.5****Codec Frequency Dividing Circuit Enable Bit**

0	Disable codec frequency dividing circuit
1	Enable codec frequency dividing circuit

**.4****Mute Control Bit**

0	Enable mute(Low out)
1	Disable mute(Data out)

**.3****Bit 3**

Not used	
----------	--

**.2–.0****Codec Input Clock Selection Bits**

0	0	0	$f_{256S} = f_{xin} \div 1$
0	0	1	$f_{256S} = f_{xin} \div 2$
0	1	0	$f_{256S} = f_{xin} \div 3$
0	1	1	$f_{256S} = f_{xin} \div 4$
1	0	0	$f_{256S} = f_{xin} \div 5$
1	0	1	$f_{256S} = f_{xin} \div 6$
1	1	0	$f_{256S} = f_{xin} \div 8$
1	1	1	$f_{256S} = f_{xin} \div 10$



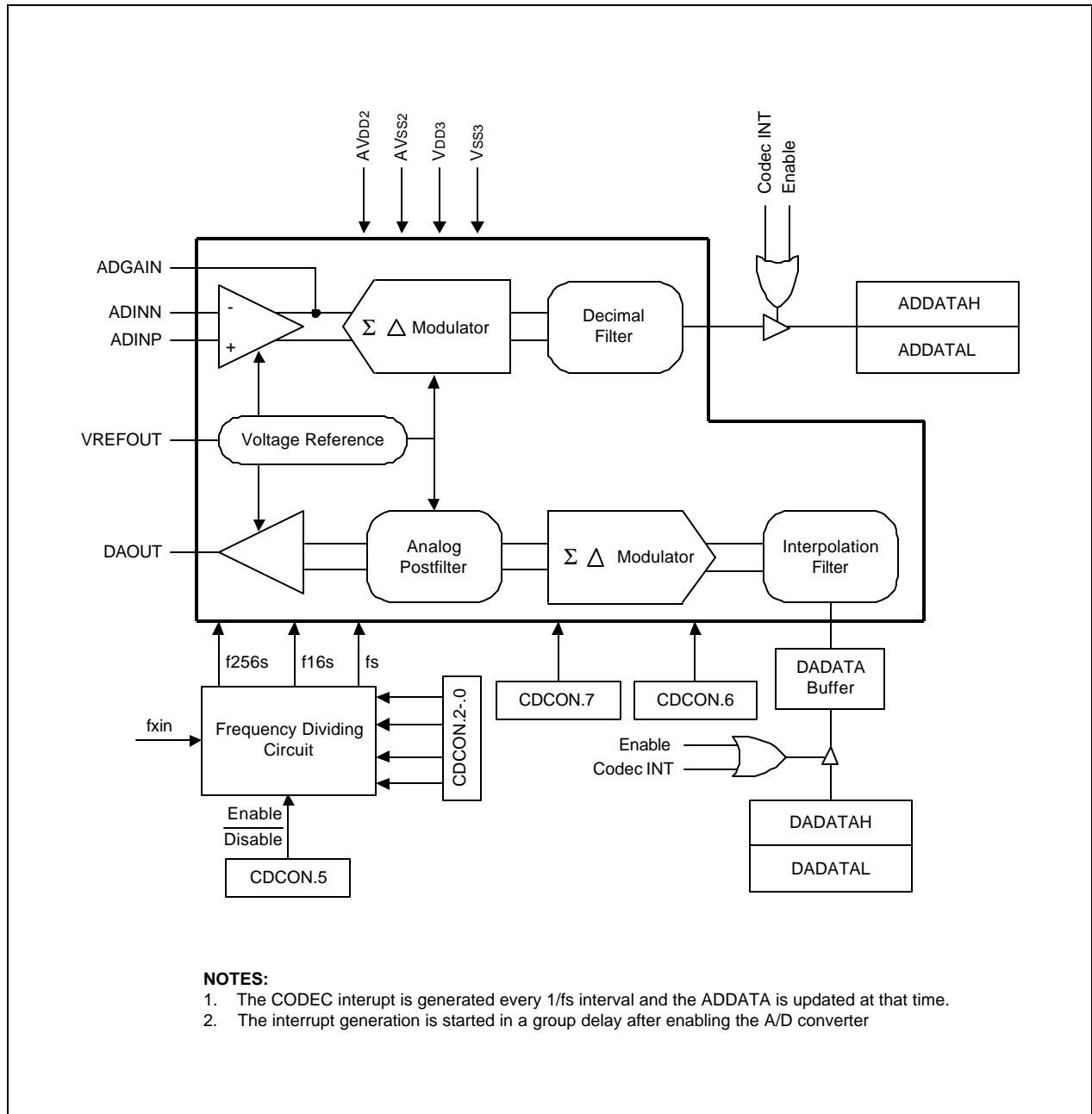


Figure 17-1. CODEC Block Diagram

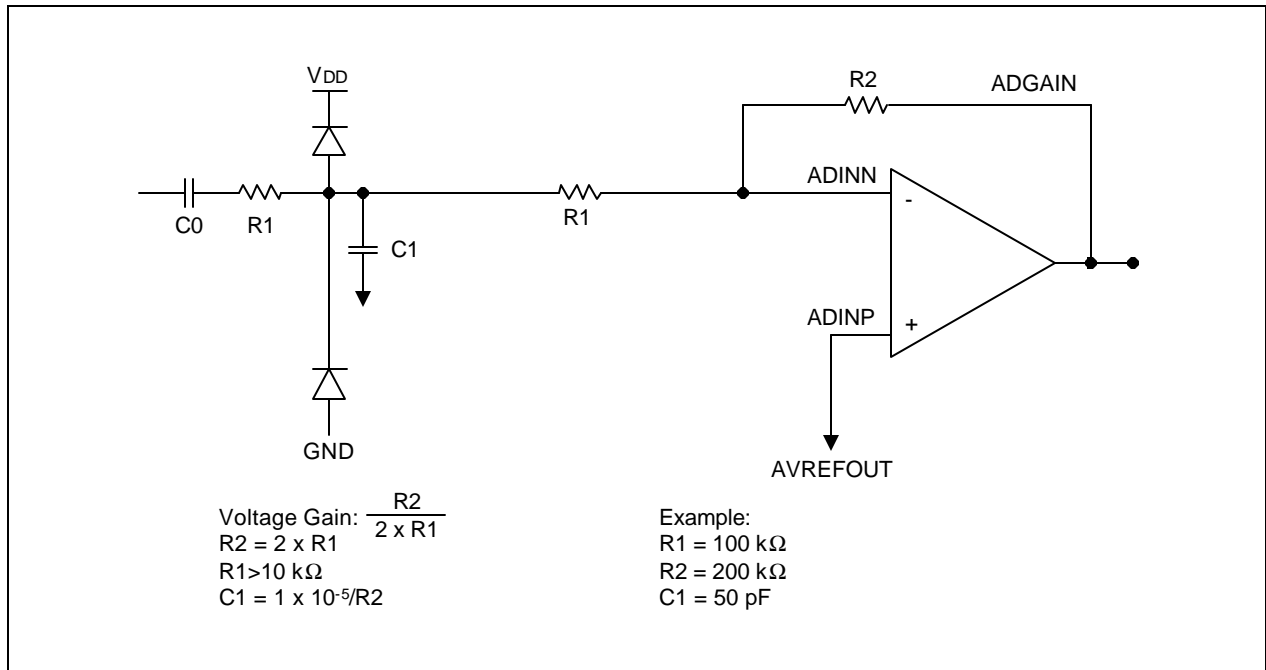


Figure 17-2. Single-Ended Input Application

**NOTES**

# 18

## LCD CONTROLLER / DRIVER

### OVERVIEW

The S3CC11B/FC11B microcontroller can directly drive an up-to-288-dot (36 segments x 8 commons) LCD panel. Its LCD block has the following components:

- LCD controller/driver
- Display RAM for storing display data
- 36 segment output pins (SEG0–SEG35)
- 8 common output pins (COM0–COM7)
- Internal resistor circuit for LCD bias
- $V_{LC1}$  pin for controlling the driver and bias voltage

The LCD control register, LCON, is used to turn the LCD display on and off, switch the current to the dividing resistors for the LCD display, and frame frequency. Data written to the LCD display RAM can be automatically transferred to the segment signal pins without any program control.

When a subsystem clock is selected as the LCD clock source, the LCD display is enabled even in the main clock stop or idle mode.

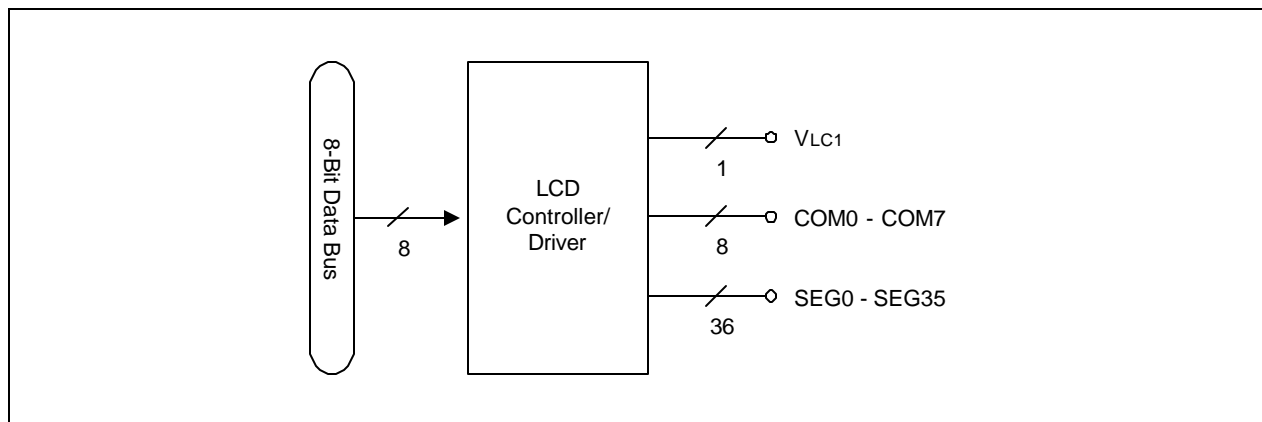


Figure 18-1. LCD Function Diagram

LCD CIRCUIT DIAGRAM

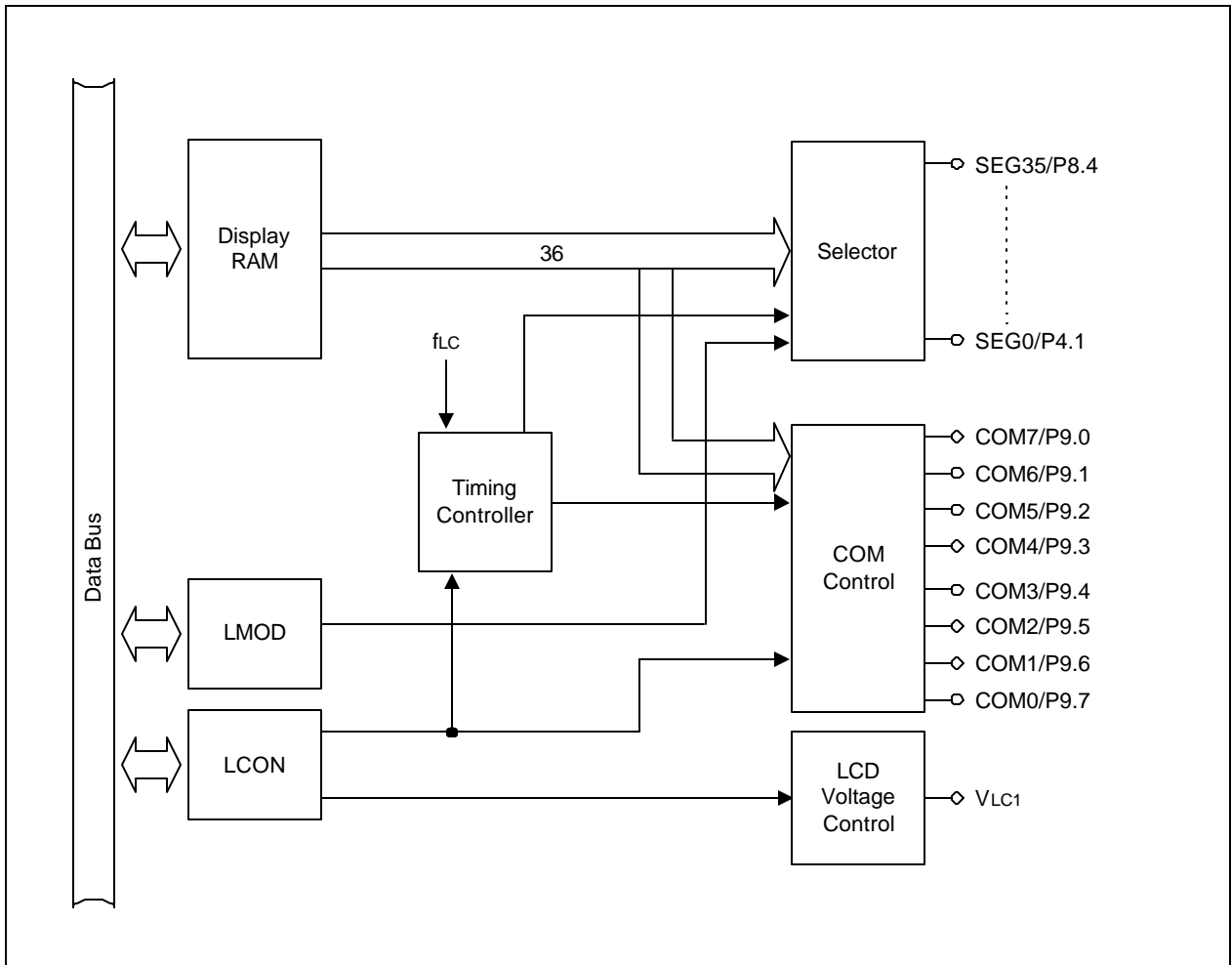
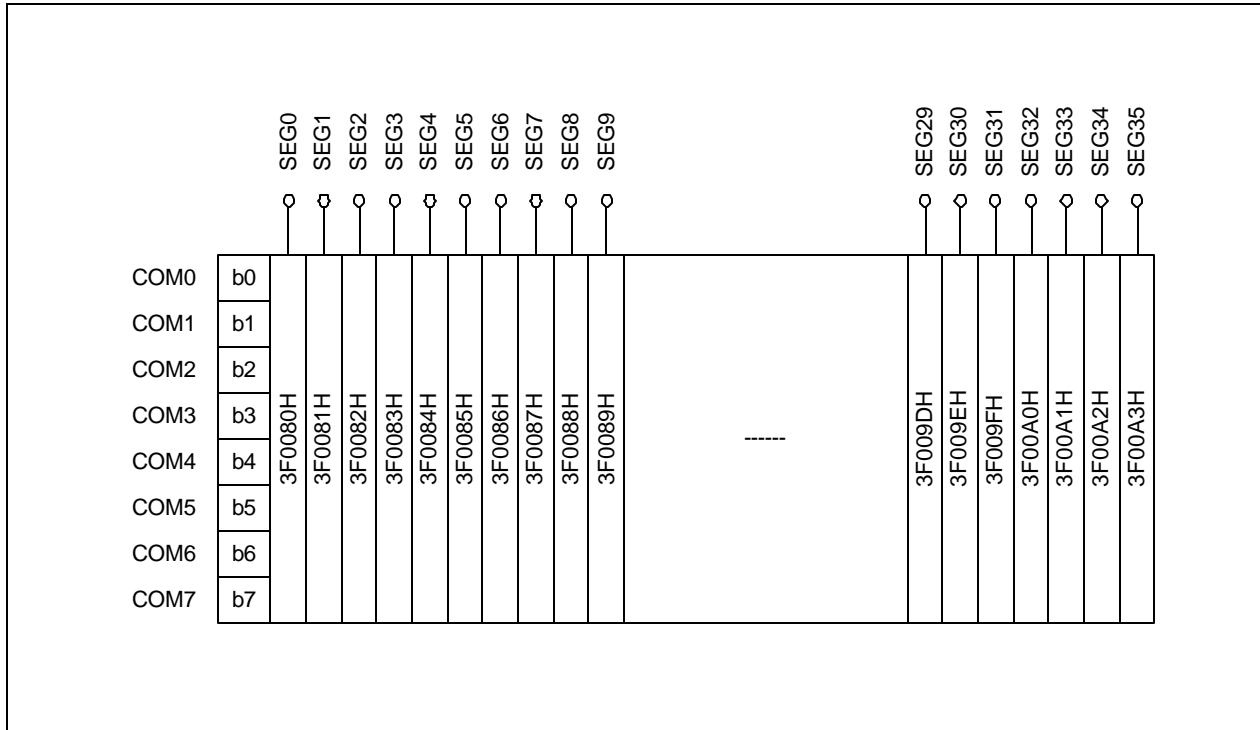


Figure 18-2. LCD Circuit Diagram

**LCD DISPLAY REGISTERS**

3F0080–3F00A3H are used as LCD data memory. These locations can be addressed by 1-bit or 8-bit instructions. If the bit value of a display segment is "1", the LCD display is turned on. If the bit value is "0", the display is turned off.

Display RAM data are sent out through the segment pins, SEG0–SEG35, using the direct memory access (DMA) method that is synchronized with the  $f_{LCD}$  signal. RAM addresses in this location that are not used for LCD display can be allocated to general-purpose use.



**Figure 18-3. LCD Display Register Organization**

**LCD CONTROL REGISTER (LCON)**

The LCD control register (LCON) is used to turn the LCD display on and off, LCD frame frequency, and control the flow of the current to the dividing resistors in the LCD circuit. After a RESET, all LCON values are cleared to "0". This turns the LCD display off and stops the flow of the current to the dividing resistors.

**LCON** — LCD Control Register**3F0072H**

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	0	0	–	–	0	0	0	0
Read/Write	R/W	R/W	–	–	R/W	R/W	R/W	R/W

**.7–6****LCD Display Control Bits**

0	0	Display off, P-Tr off
0	1	Normal display (using $V_{LC1}$ with external voltage), P-Tr off
1	0	Not available
1	1	Normal display (using $V_{LC1}$ with internal voltage), P-Tr on

**.5–4****Bits 5–4**

Not used		
----------	--	--

**.3–2****LCD Duty and Bias Selection Bits**

0	0	1/3 duty (COM0–COM2 select), 1/3bias
0	1	1/4 duty (COM0–COM3 select), 1/3bias
1	0	1/8 duty (COM0–COM7 select), 1/4bias
1	1	1/8 duty (COM0–COM7 select), 1/5bias

**.1–0****LCD Clock Selection Bits**

0	0	$f_w/2^7$ (256 Hz when $f_w$ is 32.768 kHz)
0	1	$f_w/2^6$ (512 Hz when $f_w$ is 32.768 kHz)
1	0	$f_w/2^5$ (1,024 Hz when $f_w$ is 32.768 kHz)
1	1	$f_w/2^4$ (2,048 Hz when $f_w$ is 32.768 kHz)

**LMOD** — LCD Mode Control Register**3F0073H**

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	–	–	–	–	–	0	0	0
Read/Write	–	–	–	–	–	R/W	R/W	R/W

**.7–3****Bits 7–3**

Not used

**.2****SEG2 Signal Selection Bit (When P4.3 is selected as alternative function only)**

0	CCLK output
1	SEG2 output

**.1****SEG1 Signal Selection Bit (When P4.2 is selected as alternative function only)**

0	CFS output
1	SEG1 output

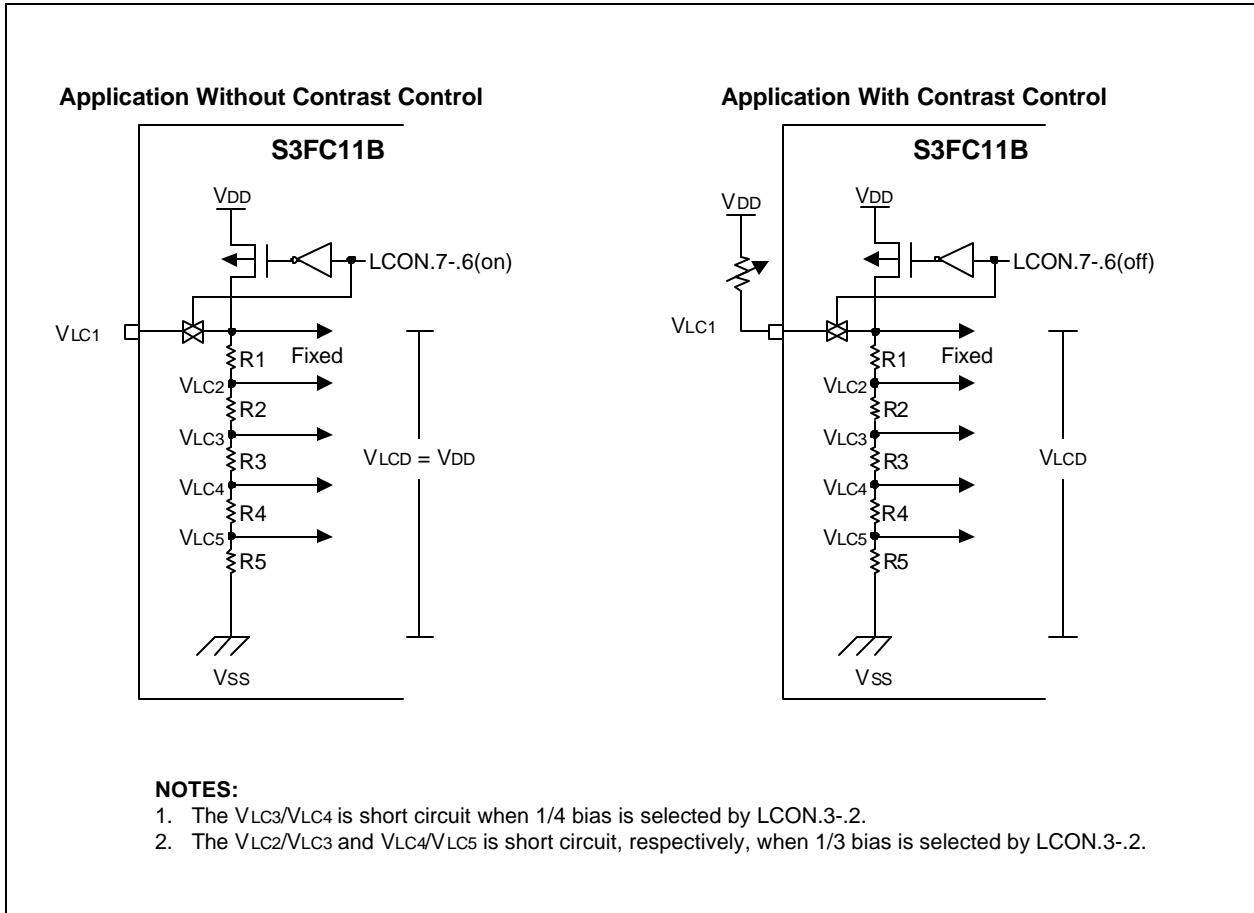
**.0****SEG0 Signal Selection Bit (When P4.1 is selected as alternative function only)**

0	CDX output
1	SEG0 output



**LCD VOLTAGE DIVIDING RESISTORS**

On-chip voltage dividing resistors for the LCD drive power supply are fixed to the  $V_{LC1} - V_{LC5}$  pins. Figure 15-5 shows the bias connections for the S3CC11B/FC11B LCD drive power supply. To cut off the flow of current through the dividing resistor, manipulate bits 7 and 6 of the LCON register.



**Figure 18-4. LCD Voltage Dividing Registers Connection**

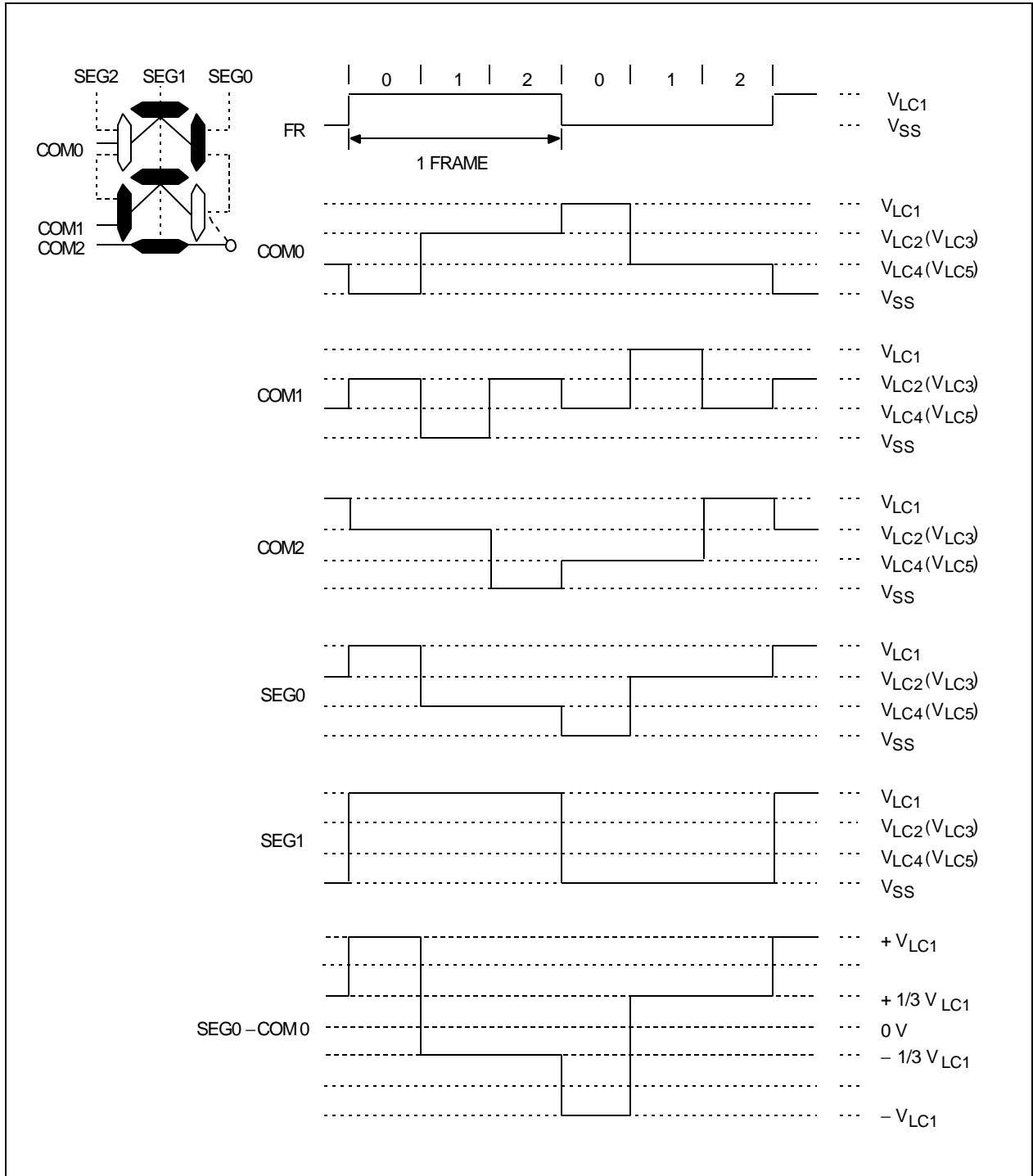


Figure 18-5. LCD Signal Waveforms (1/3 Duty, 1/3 Bias)

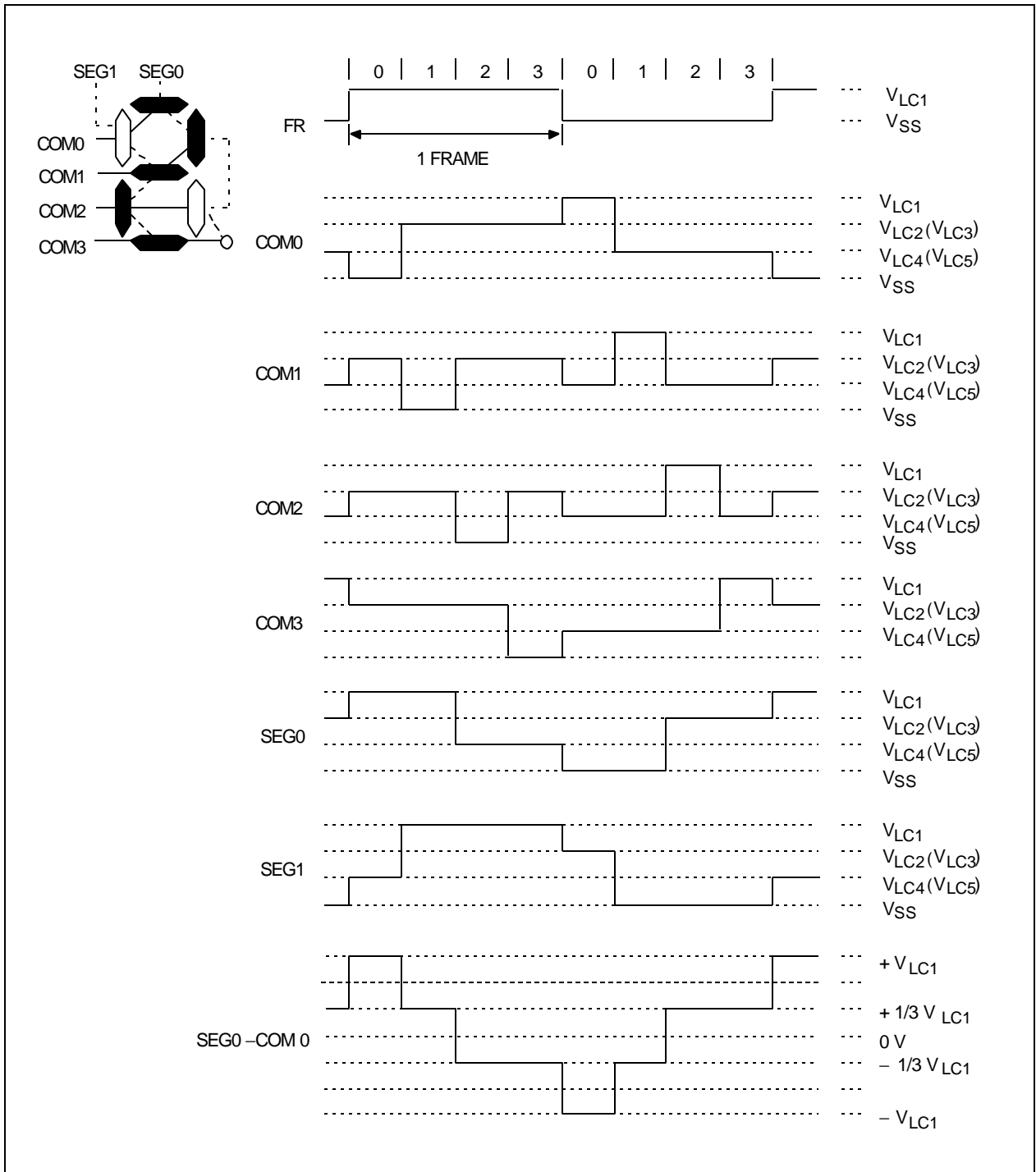


Figure 18-6. LCD Signal Waveforms (1/4 Duty, 1/3 Bias)

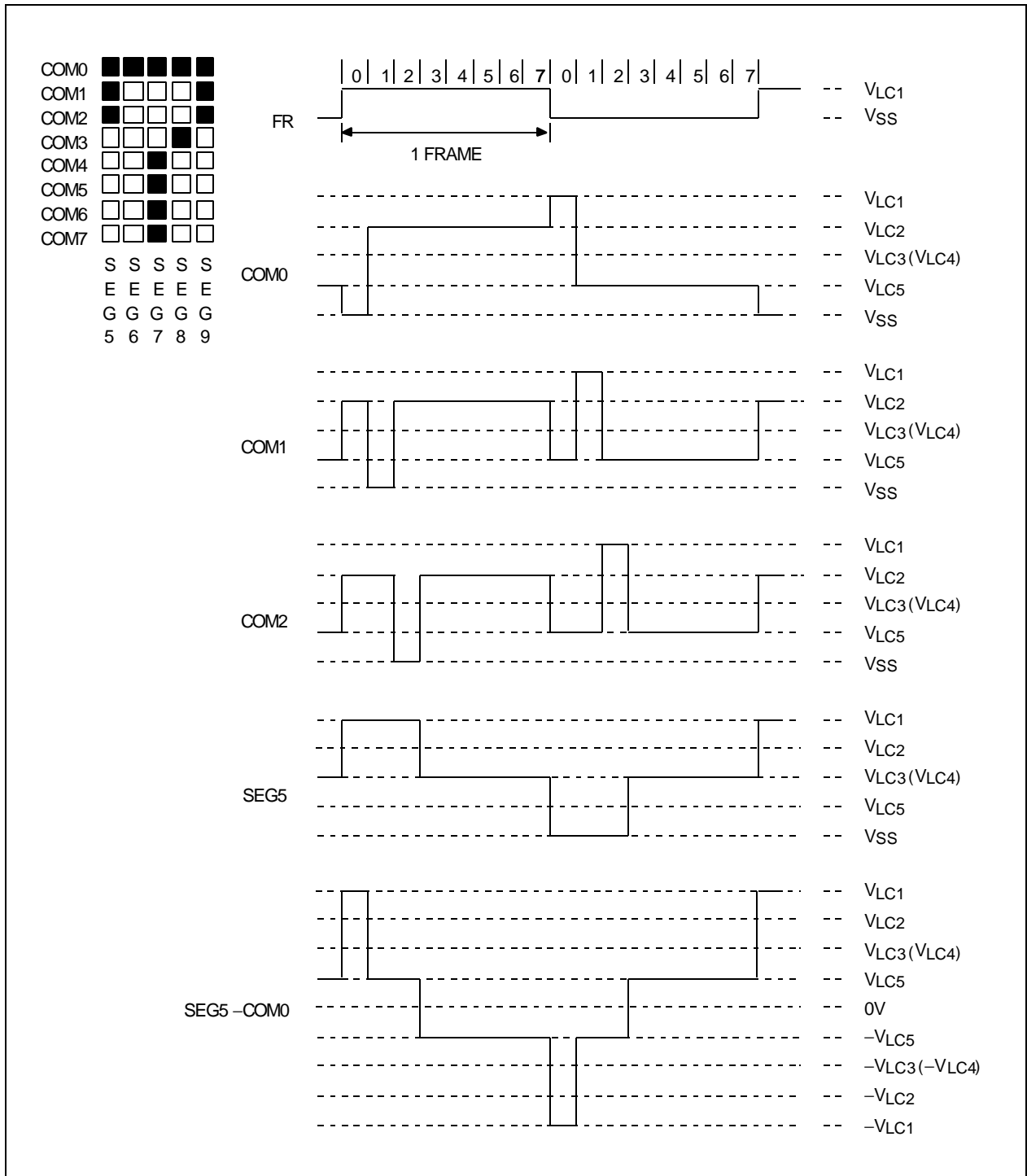


Figure 18-7. LCD Signal Waveforms (1/8 Duty, 1/4 Bias)

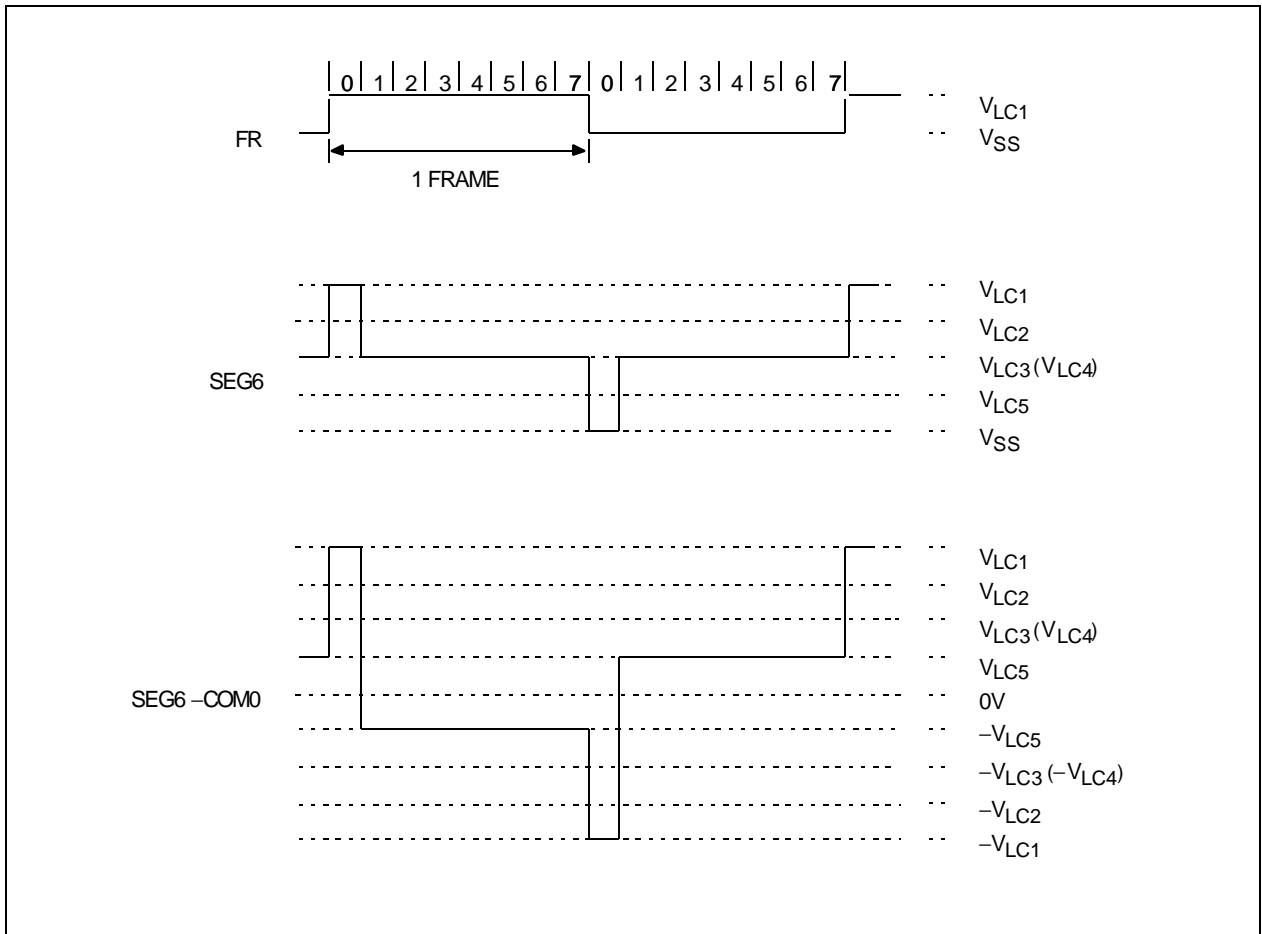


Figure 18-8. LCD Signal Waveforms (1/8 Duty, 1/4 Bias) (Continued)

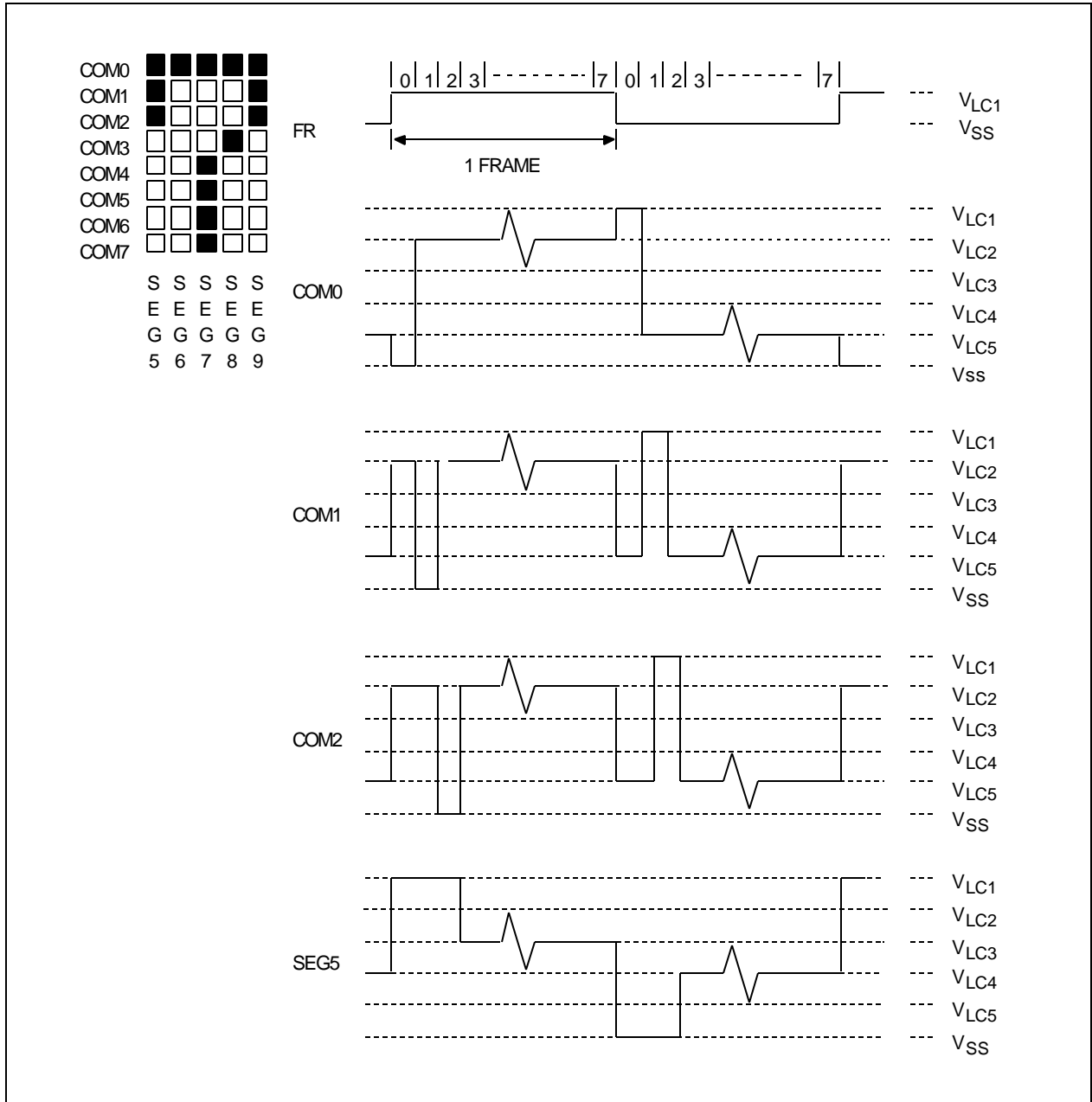


Figure 18-9. LCD Signal Waveforms (1/8 Duty, 1/5 Bias)

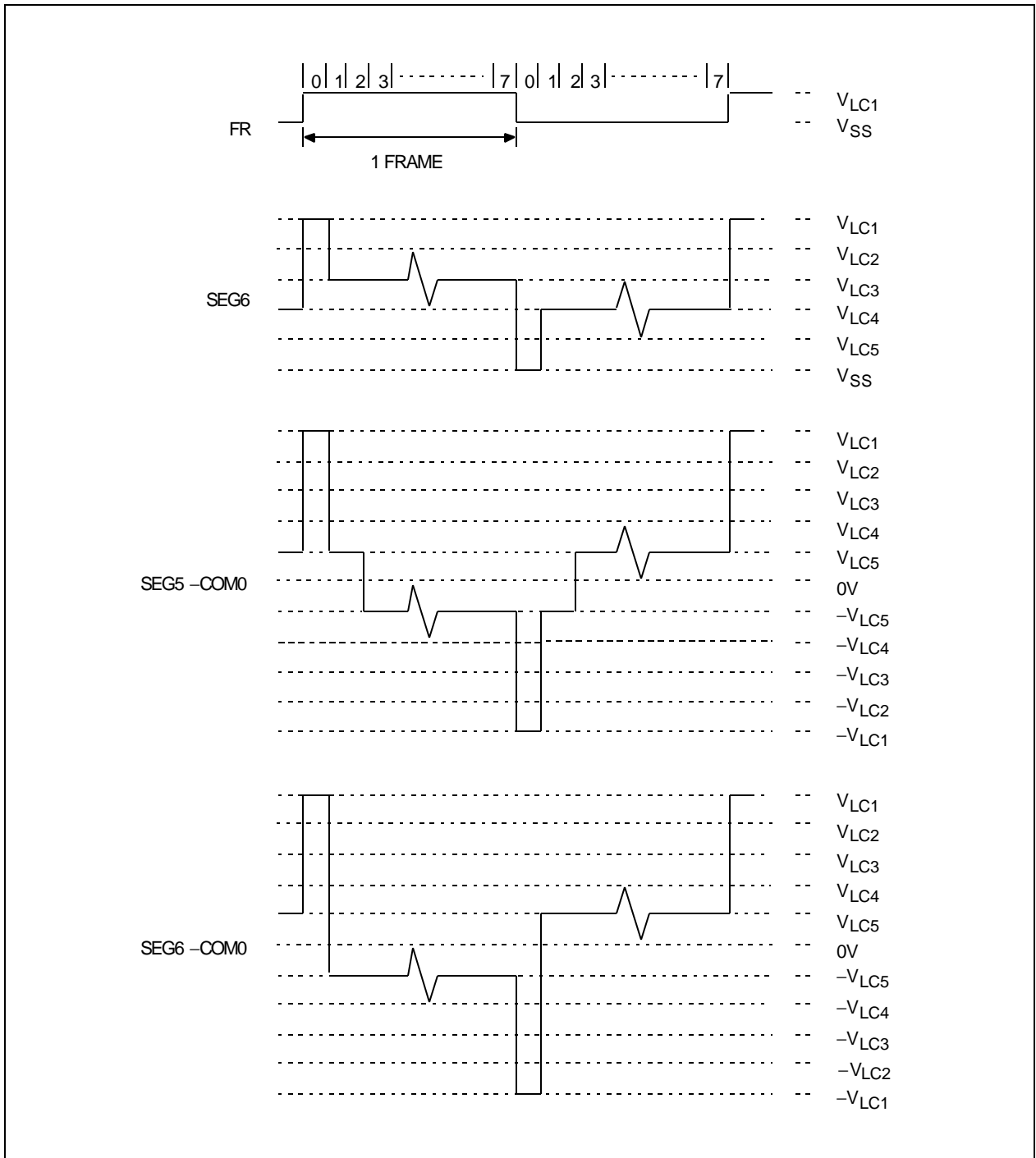


Figure 18-10. LCD Signal Waveforms (1/8 Duty, 1/5 Bias) (Continued)

# 19

## BATTERY LEVEL DETECTOR

### OVERVIEW

The S3CC11B/FC11B micro-controller has a built-in BLD (Battery Level Detector) circuit which allows detection of power voltage drop through software. Turning the BLD operation on and off can be controlled by software. Because the IC consumes a large amount of current during BLD operation. It is recommended that the BLD operation should be kept OFF unless it is necessary. Also the BLD criteria voltage can be set by the software. The criteria voltage can be set by matching to one of the 2 kinds of voltage.

2.45 V or 2.70 V ( $V_{DD}$  reference voltage)

The BLD block works only when BLDCON.0 is set. If VDD level is lower than the reference voltage selected with BLDCON.4-2, BLDCON.1 will be set. If VDD level is higher, BLDCON.1 will be cleared. When users need to minimize current consumption, do not operate the BLD block.

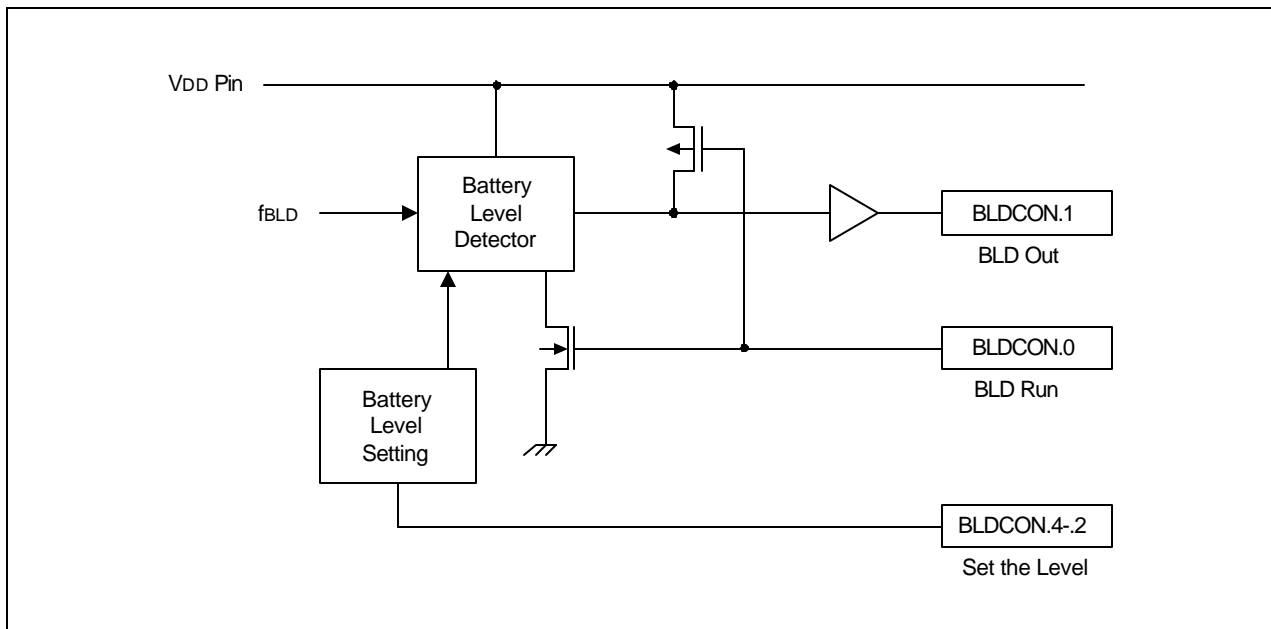
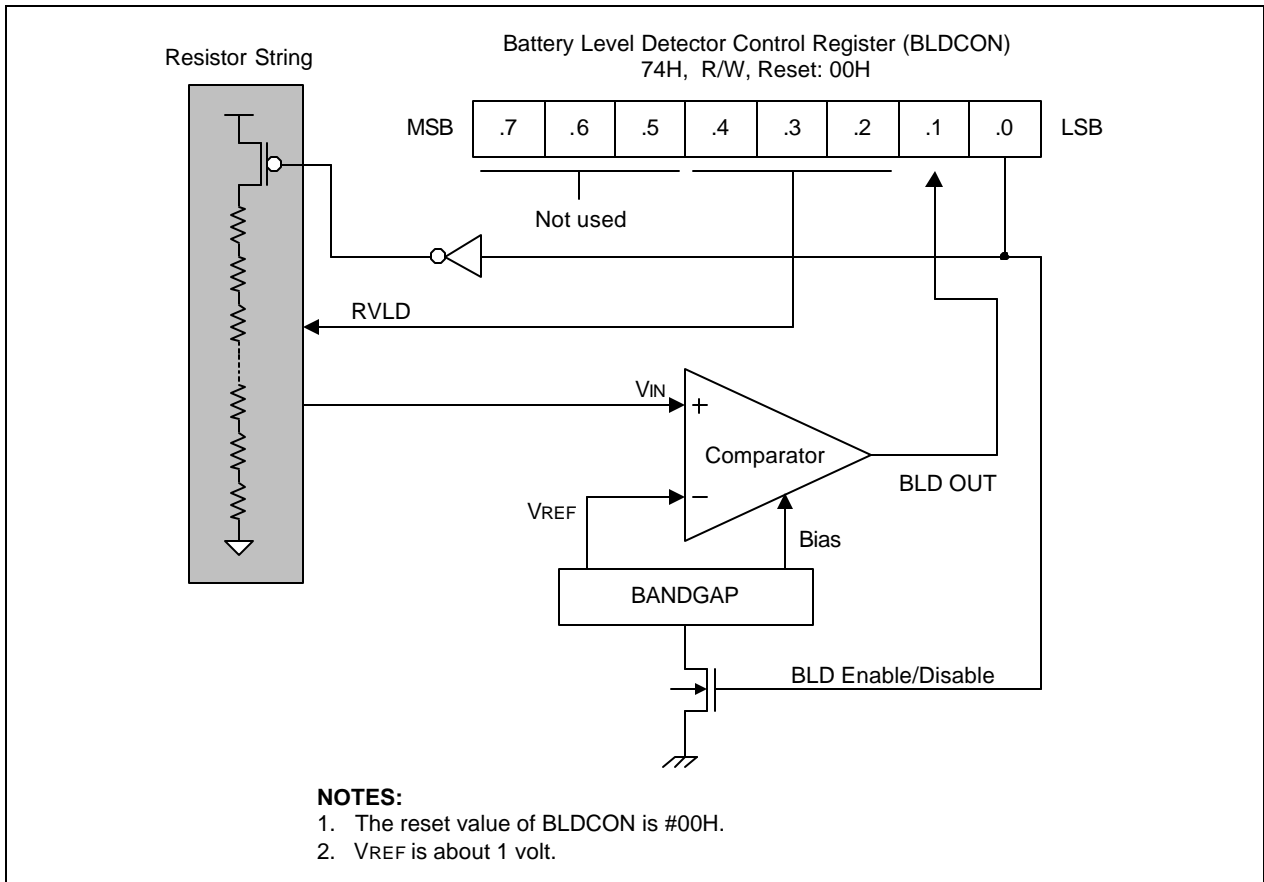


Figure 19-1. Block Diagram for Battery Level Detect



**BATTERY LEVEL DETECTOR CONTROL REGISTER (BLDCON)**

The bit 0 of BLDCON controls to run or disable the operation of battery level detect. Basically this  $V_{BLD}$  is set as invalid by system reset and it can be changed in 2 kinds voltages by selecting Battery Level Detect Control register (BLDCON). When you write 3-bit data value to BLDCON, an established resistor string is selected and the  $V_{BLD}$  is fixed in accordance with this resistor. Table 16-1 shows specific  $V_{BLD}$  of 2 levels.



**Figure 19-2. Battery Level Detector Circuit and Control Register**

**Table 19-1. BLDCON Value and Detection Level**

VLDCON .4-.2	$V_{BLD}$
0 0 0	-
0 0 1	2.45 V
1 1 1	2.70 V

# 20

## 8/16-BIT SERIAL INTERFACE FOR EXTERNAL CODEC

### OVERVIEW

8/16-bit serial interface for external codec, CSIO, can interface with voice CODEC<sup>(note)</sup>. The components of each CSIO function block are :

- 8-bit control register (CSIOCON)
- 16-bit Data buffer (CSIODATAH, CSIODATAL)
- Serial data I/O pins (CDX, CDR)
- Frame sync. pin (CFS)
- External clock input/out pin (CCLK)

The CSIO module can transmit or receive 16-bit serial data configured by its corresponding control register settings. The CSIO module operates with master mode only.

### PROGRAMMING PROCEDURE

To program the CSIO modules, follow these basic steps:

1. Load an 8-bit value to the CSIOCON control register to properly configure the CSIO module.
2. The CSIO interrupt request is automatically generated at the end of 16-bit shifting.
3. In the CSIO interrupt routine, read/write ADC/DAC data.
4. Repeat steps 3 to 4.

### NOTE

Voice codec: MC145483DW

**CSIO CONTROL REGISTER (CSIOCON)**

The control register for CSIO interface module, CSIOCON, is located at 4E. It has the control settings for the CSIO module.

- 8/16 serial interface for external codec
- Shift clock selection
- Short/long frame sync type selection
- Edge selection for shift operation
- Shift operation (transmit/receive) enable

**CSIOCON — SIO Control Register for External Codec****3F004EH**

<b>Bit Identifier</b>	<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
<b>Reset Value</b>	–	0	0	0	0	0	0	0
<b>Read/Write</b>	–	R/W	R/W	R/W	R/W	R/W	R/W	R/W

**.7****Bit 7**

Not used

**.6****8/16-Bit Serial I/O Selection Bit**

0	Select 8-bit serial interface for external codec
1	Select 16-bit serial interface for external codec

**.5–.3****Shift Clock Selection Bits**

0	0	0	$f_{CSIO} = f_{xin} \div 1$
0	0	1	$f_{CSIO} = f_{xin} \div 2$
0	1	0	$f_{CSIO} = f_{xin} \div 3$
0	1	1	$f_{CSIO} = f_{xin} \div 4$
1	0	0	$f_{CSIO} = f_{xin} \div 5$
1	0	1	$f_{CSIO} = f_{xin} \div 6$
1	1	0	$f_{CSIO} = f_{xin} \div 8$
1	1	1	$f_{CSIO} = f_{xin} \div 10$

**.2****Frame Sync Type Selection Bit**

0	Select short frame sync type
1	Select long frame sync type

**.1****Shift Clock Edge Selection Bit**

0	DX at rising edges, DR at falling edges
1	DX at falling edges, DR at rising edges

**.0****Shift Operation Control Bit**

0	Disable shift operation (SIO for external codec)
1	Enable shift operation (SIO for external codec)

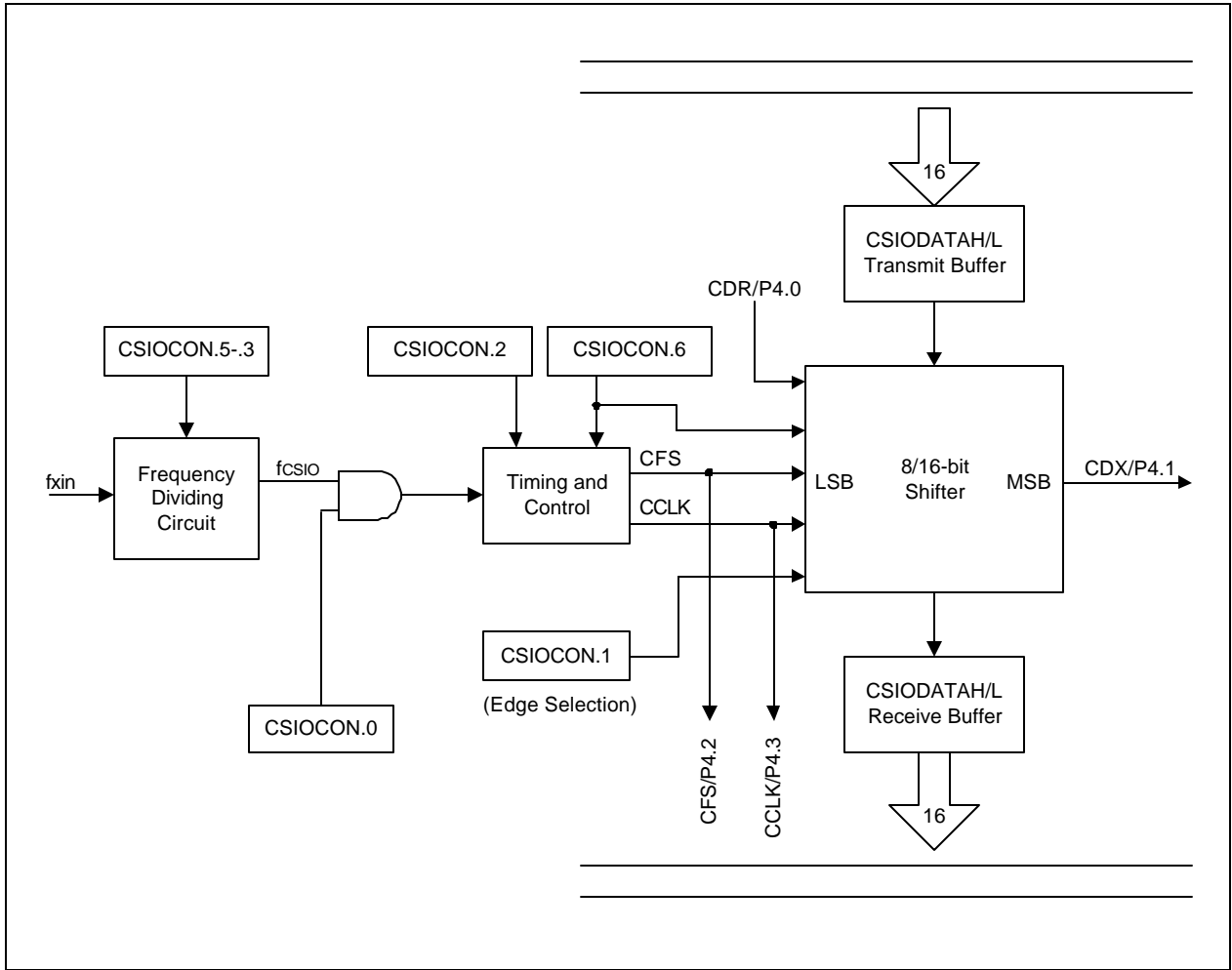


Figure 20-1. SIO Block Diagram for External Codec

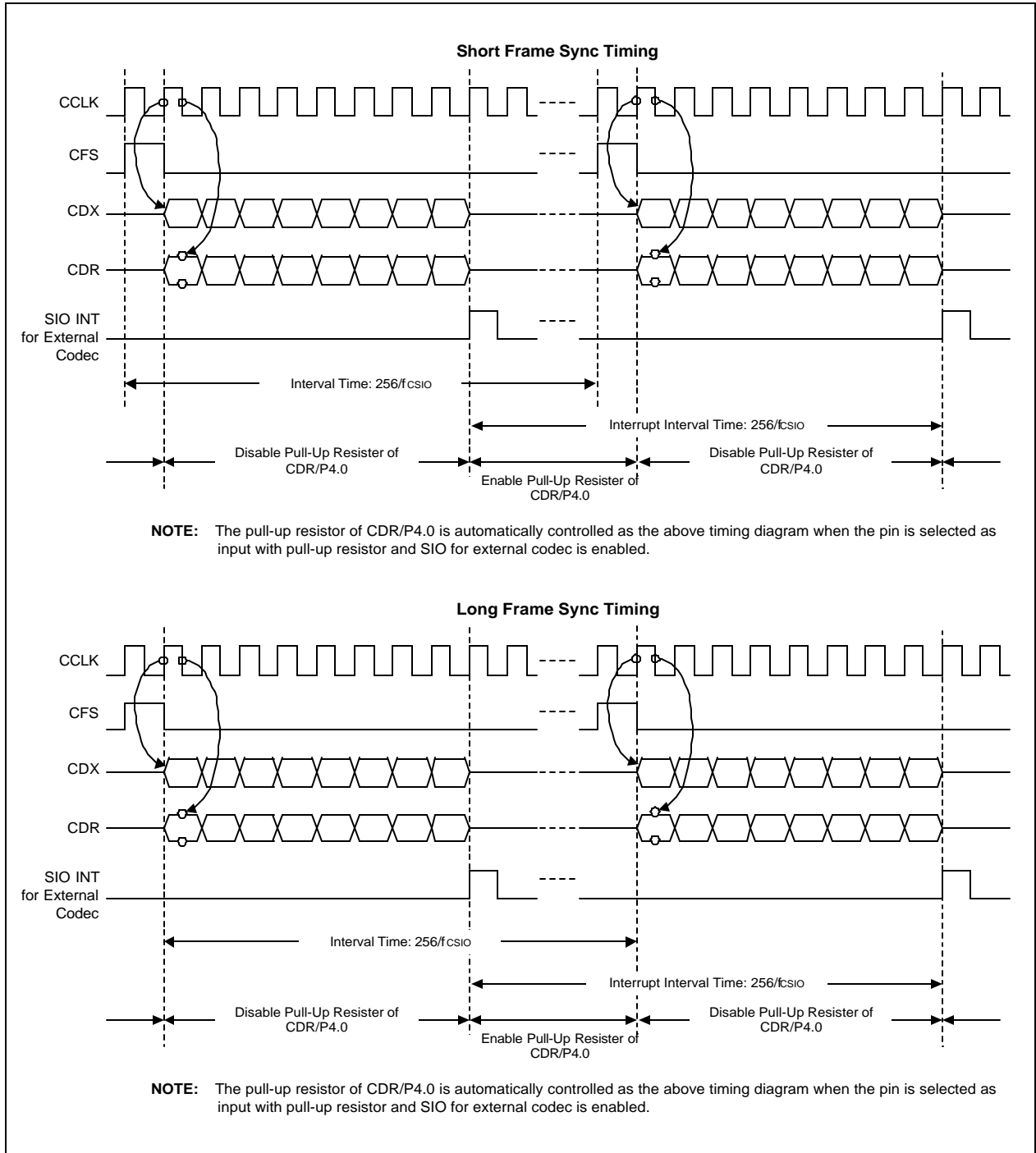


Figure 20-2. 8-Bit SIO Timing Diagram for External Codec

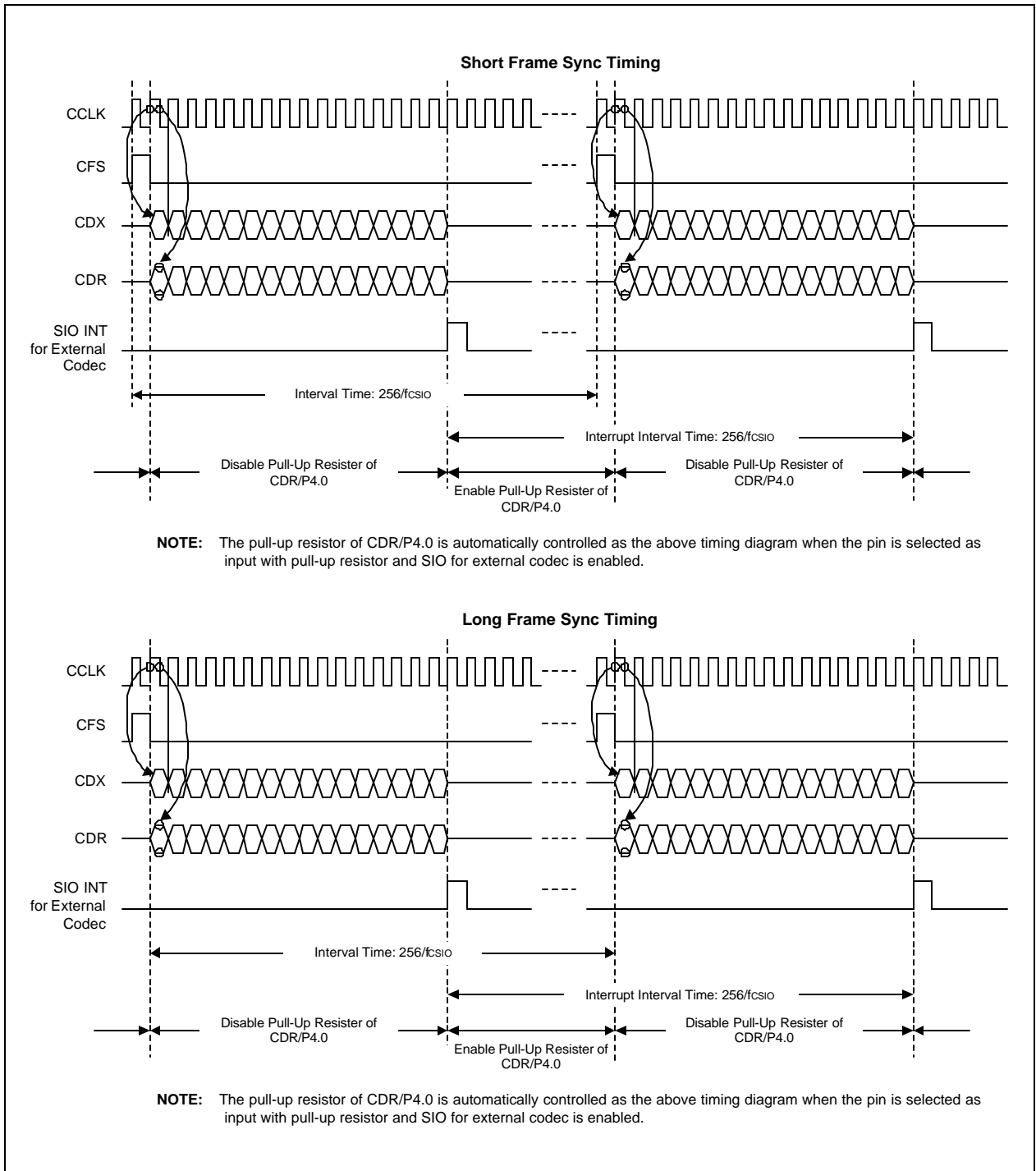


Figure 20-3. 16-Bit SIO Timing Diagram for External Codec

# 21

## CalmMAC1616

### INTRODUCTION

CalmMAC16 is a 16-bit high performance fixed-point DSP coprocessor for CalmRISC16 microcontroller. CalmMAC16 is designed for the mid to high-end audio applications which require low power consumption and portability. It mainly includes a 16-bit arithmetic unit (ARU), a barrel shifter & exponent unit (BEU), a 16-bit x 16-bit multiplier accumulation unit (MAU), and a RAM pointer unit (RPU) for data address generation. Main datapaths are constructed to 16-bit width for audio applications.

CalmMAC16 is designed to be the DSP coprocessor for CalmRISC16 microcontroller. It receives 13-bit instruction code and command information from CalmRISC16 via special coprocessor interface and sends internal status information to host processor, CalmRISC16 through external condition port.

### ARCHITECTURE FEATURES

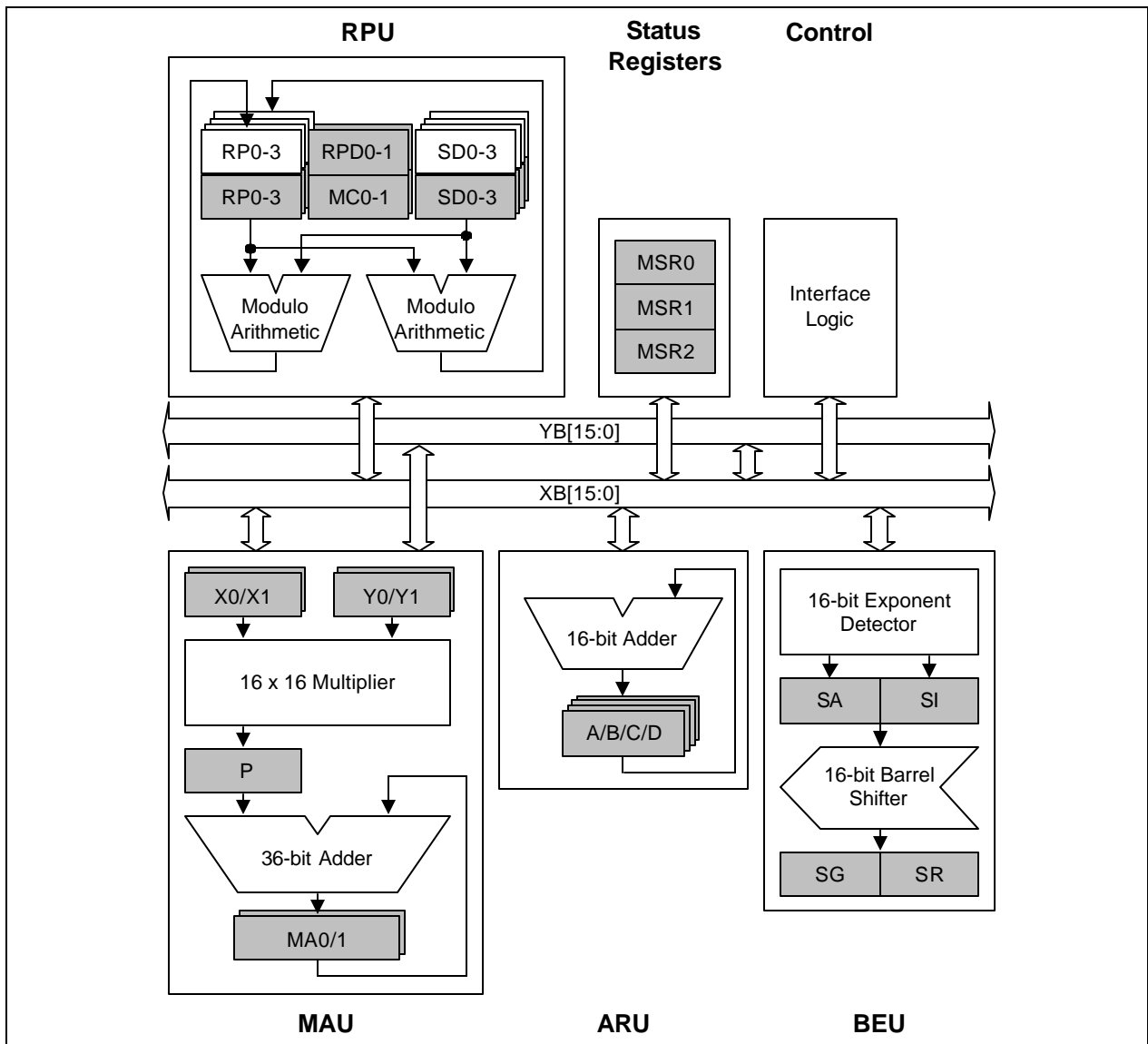
- 16-bit barrel shifting with support for multi-precision capability
- 16-bit exponent evaluation with support for multi-precision capability
- 4 data address RAM pointers with post-modification & modulo capability
- 4 index registers with 2 extended index registers : up to 8-bit index value
- 2 direct address RAM pointers for short direct addressing
- Min/Max instruction with pointer latching and modification
- Division step in single cycle
- Conditional instruction execution capability
- Four-Quadrant fractional/integer 16 x 16-bit multiplication in single cycle
- 16 x 16-bit multiplication and 36-bit accumulation in a single cycle
- 16-bit arithmetic operation
- 2 32-bit multiplier accumulator with 4-bit guard
- 2 32K x 16-bit data memory spaces

### TECHNOLOGY FEATURES

- 0.35u triple metal CMOS technology
- 12ns cycle time at 3.0V, 125C, Worst Process condition
- Fully static design



**BLOCK DIAGRAM**



**Figure 21-1. CalmMAC1616 Block Diagram**

The block diagram shows the main blocks that compose the CalmMAC16:

- Multiplier Accumulator Unit (MAU)
- Arithmetic Unit (ARU)
- Barrel shifter & Exponent detection Unit (BEU)
- RAM Pointer Unit (RPU)
- Status Registers
- Interface Unit

## PROGRAMMING MODEL

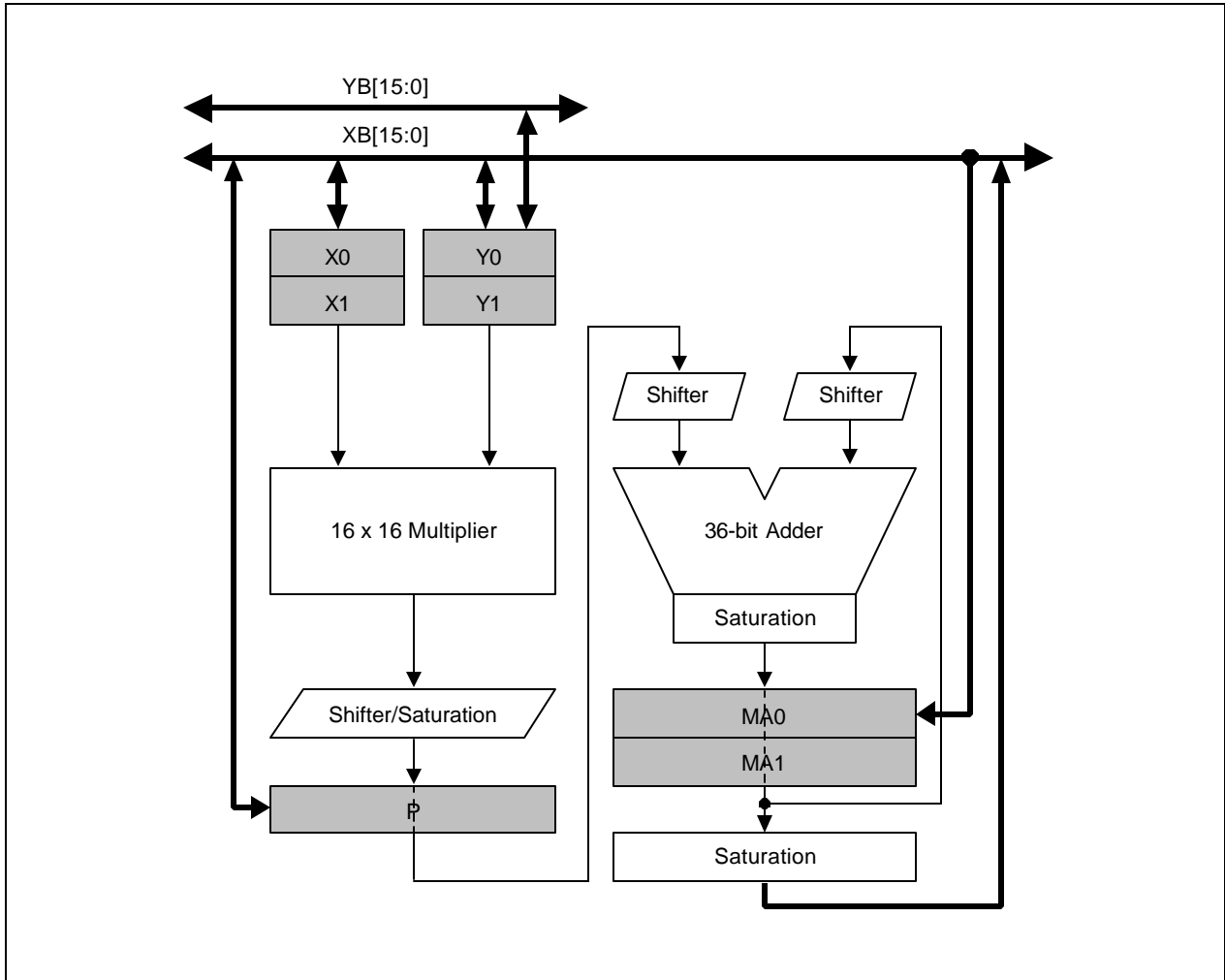
In this chapter, the important features of each unit in CalmMAC16 are discussed in details. How the data memories are organized is discussed and data memory addressing modes are explained.

The major components of the CalmMAC16 are:

- Multiplier Accumulator Unit (MAU)
  - Multiplier
    - Input Registers X0, X1, Y0, Y1
    - Output Register P
  - Multiplier Accumulators MA0, MA1
  - Saturation Logic
  - Multiplier Accumulator Shifter
  - 36-bit Arithmetic Unit
  - Status Register MSR1
- Arithmetic Unit (ARU)
  - Accumulator A, B, C, D
  - Saturation Logic
  - Accumulator Shifter
  - 16-bit Arithmetic Unit
  - Status Registers MSR0, MSR2
- Barrel shifter & Exponent detection Unit (BEU)
  - 16-bit Exponent Detector
  - 16-bit Barrel Shifter
    - Input Registers SA, SI
    - Output Registers SG, SR
- RAM Pointer Unit (RPU)
  - 2 Modulo Address Generator
  - 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 Registers MC0, MC1
  - Alternative Bank Pointers RP0, RP1, RP2, RP3
  - Alternative Bank Index Registers SD0, SD1, SD2, SD3
  - Alternative Bank Extended Index Registers SD0E, SD3E

**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 21-2.



**Figure 21-2. Multiplier and Accumulator Unit Block Diagram**

## Multiplier

The Multiplier unit consists of a 16 by 16 to 32 bit parallel 2's complement single-cycle, non-pipelined multiplier, 4 16-bit input registers (X0, X1, Y0, and Y1), a 32-bit output product register (P), and output shifter & saturation logic. The multiplier can perform 4-quadrant multiplication. (signed by signed, unsigned by signed, signed by unsigned, and unsigned by unsigned) Together with 36-bit adder in MAU, the CalmMAC16 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 operates after a change in the input registers. This scheme reduces power consumption in multiplier.

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. When USM bit is set, 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.

The X or Y register can be read or written via the XB bus, and Y register can be written via YB when dual load instruction is executed. The 16-bit most significant portion (MSP) of the P register (PH) or the 16-bit least significant portion (LSP) of the P register (PL) can be written through 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.

### Overflow Protection in Multiplier

The only case the multiplier overflow occurs is when multiplying 8000h by 8000h in signed-by-signed fractional multiplication. (These case means  $-1*-1$ ) : the result should be normally 1, which overflows fractional format. Thus, in this particular case, the multiplier saturation block forces the multiplier result to 7FFFFFFFh after internal 1-bit shift to the left and write this value to the product register P.

- Saturation Condition:  $\sim Prod[31] \& Prod[30] \& PSH1 \& SX \& SY$   
(*Prod* : product result, *PSH1* : Fractional Indication, *SX* : Signed X operand, *SY* : Signed Y operand)

### Multiplier Accumulators

Each MA<sub>i</sub> (i=0,1) is organized as two regular 16-bit registers (MA0H, MA0L, MA1H, MA1L) and two 4-bit extension nibble (MA0E, MA1E) in MSR1 register. The MA<sub>i</sub> accumulators can serve as the source operand, as well as the destination operand of MA relevant instructions. 36-bit full data transfer between two MA accumulators is possible through "ELD MA1, MA0" and "ELD MA0, MA1" instructions.

The 16-bit most significant portion (MSP) of the MA register (MA<sub>i</sub>H) or the 16-bit least significant portion (LSP) of the MA register (MA<sub>i</sub>L) can be written by the XB as an operand. When MA<sub>i</sub>H register is written, MA<sub>i</sub>L register is forced to zero and MA<sub>i</sub>E extension nibble is sign-extended. When MA<sub>i</sub>L register is written, MA<sub>i</sub>H and MA<sub>i</sub>E are not changed.

Registers MA<sub>i</sub>H and MA<sub>i</sub>L can also be used as general-purpose temporary 16-bit data registers.

**Extension Nibbles**

Extension nibbles MA0E and MA1E in MSR1 register offer protection against 32-bit overflows. When the result of a 36-bit adder output crosses bit 31, it sets VMi flag of MSR1 register (MA register Overflow flag). Upto 15 overflows or underflows are possible using the extension nibble, after which the sign is lost beyond the MSB of the extension nibble, setting MV flag of MSR1 (Memorized Overflow flag) and latching the value.

Registers MA0E and MA1E can not be accessed independently. Those registers are read or written as a part of MSR1 register, during MSR1 register read or write instruction.

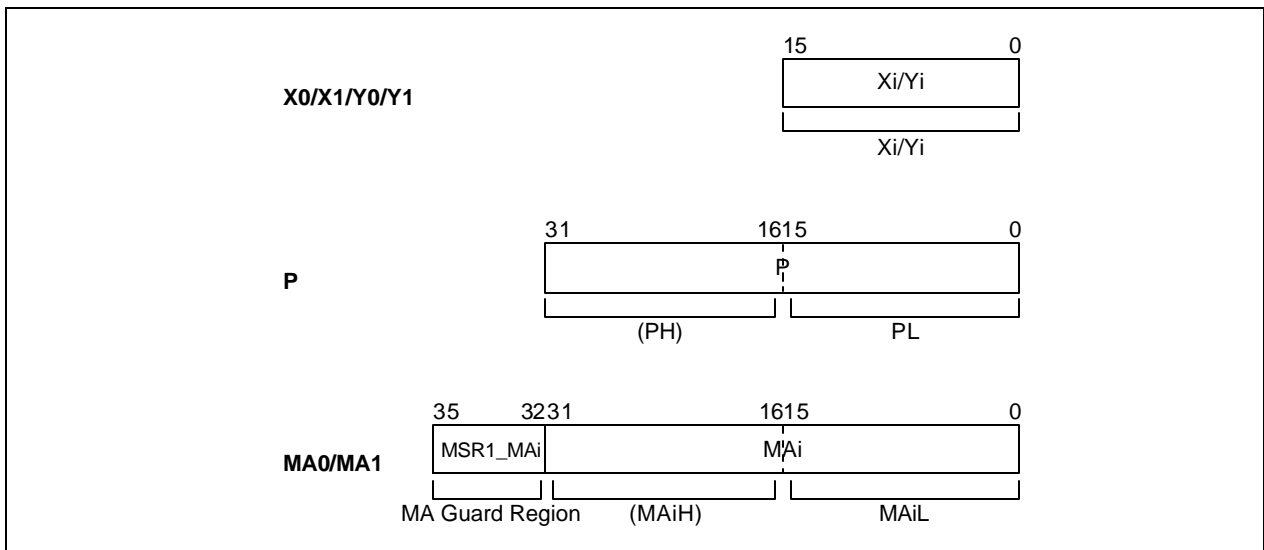
**Overflow Protection in MA Registers**

The multiplier accumulator saturation instruction (ESAT instruction) sets the destination MA register to the plus or minus maximum value, if selected MA register overflows (VMi bit of MSR1 register is set). Saturation values are 7FFFFFFFh (positive overflow) or 80000000h (negative overflow) for the MA register and extension nibble is sign-extended.

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. Saturation values are 7FFFh (positive overflow) or 8000h (negative overflow).

The last saturation condition is when enabling saturation on multiplier accumulators during arithmetic calculations by setting the OPMA bit of MSR1 register. When overflow from the high portion of an MAi accumulator to the extension bits occurs during MAi arithmetic operation and the OPMA bit is set, the accumulator is limited to a full-scale 32-bit positive (7FFFFFFFh) or negative (80000000h) value.

- Saturation by Instruction : “ESAT” Instruction & VMi
- Saturation by MA Read : Read MAiH & VMi & OPM
- Saturation by Arithmetic Operation : Arithmetic Instruction on MAi & VMi & OPMA



**Figure 21-3. MAU Registers Configuration**

## ARITHMETIC UNIT

The arithmetic unit performs several arithmetic operations on data operands. It is a 36-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, all 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

Two rounding operations are enabled inside the CalmMAC16 : the first one concerns the whole 32-bit MAi accumulator, the second concerns a higher 16-bit portion of MAi register (MAiH) or a higher 16-bit portion of P register (PRN) during 16-bit arithmetic operation in ARU.

The first rounding facility is provided by the "ERND" instruction. It can be applied only to a multiplier accumulator. The rounding operation is always a two's complement rounding operation.

- If bit 15 of MAiL is 1, 1 is added in bit 16 position of MA register, the result is stored in MAiH register, and MAiL is not changed.
- If bit 15 of MAiL is 0 MAiH and MAiL register remain unchanged. The second rounding is provided as a form of source operand (MAiRN or PRN). When the source operand of 16-bit arithmetic operation in ARU is specified as MAiRN, the rounded value of 16-bit higher portion of MAi register is read as a source operand. When the source operand is specified as PRN, the rounded value of 16-bit higher portion of P register is read as a source operand. The value of MA register or P register itself is not changed at all.

### MA Shifting Capabilities

Two shift operations are enabled inside the CalmMAC16 : the first one concerns the whole 32-bit MAi accumulator register and 4-bit extension nibble, the second concerns a higher 16-bit portion of MAi register (MAiH) during 16-bit arithmetic operation in ARU. Each of the two multiplier accumulators can be shifted arithmetically by 1-bit left or right.

The first shift operation is provided by the "ESLA" (1-bit shift left arithmetic) or "ESRA" (1-bit shift right arithmetic) instruction. The second shifting is provided as a form of source operand (MAiSL or MAiSR). When the source operand of 16-bit arithmetic operation in ARU is specified as MAiSL, the 1-bit left shifted value of 16-bit higher portion of MAi register is read as a source operand. When the source operand is specified as MAiSR, the 1-bit right shifted value of 16-bit higher portion of MAi register is read. The value of MA register itself is not changed at all.

### 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. ("EADD MAi, PSH" or "ESUB MAi, PSH" instruction). In this case, the P register is aligned (shifting 16 bits to the right) before accumulating the partial multiplication result.

An example of different multiplication is in the multiplication of 32-bit by 16-bit numbers, where two multiplication and an addition are needed : multiplying the 16-bit number with the lower and upper portion of a 32-bit (double precision) number and addition of each partial product value. The signed by signed operation is used to multiply the 16-bit signed number with the upper, signed portion of the 32-bit number. The signed by unsigned operation is used to multiply the 16-bit signed number with the lower, unsigned portion of the 32-bit number. After the signed by unsigned operation is executed, it is recommended to accumulate the aligned (using "EADD MAi, PSH" instruction) result of the signed by unsigned operation with the signed by signed operation result. For the multiplication of two double precision (32-bit) numbers, the unsigned by signed operation can be used. Note that in all cases, only upper 32-bit result can be calculated.

### 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 32-bit operand, located in MA register. : 4-bit extension nibble contains the sign extension of the MA register in 16-bit operation mode. The divisor must be a 16-bit operand located in 16-bit most significant portion of the P register. The 16-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 16-bit LSP, the significant bit remainder stored in the 16 MSP of the MA register.

Typically 32/16 division can be executed 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 NQ 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 NQ bit value.

*If current NQ = 0, new partial remainder = old partial remainder – divisor*

*If current NQ = 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 one bit left the new partial remainder 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 NQ bit of MSR1 register with sign value of the new partial remainder. This NQ 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 NQ bit of MSR1 register previously computed.

*If NQ = 0, No Operation is performed*

*If NQ = 1, Adds two times the divisor to the MA register.*

*(containing the last calculated remainder in the 16-bit most significant portion)*

The new calculated remainder will have to be 16-bit right arithmetical shifted, in order to be represented in a usual fractional format.

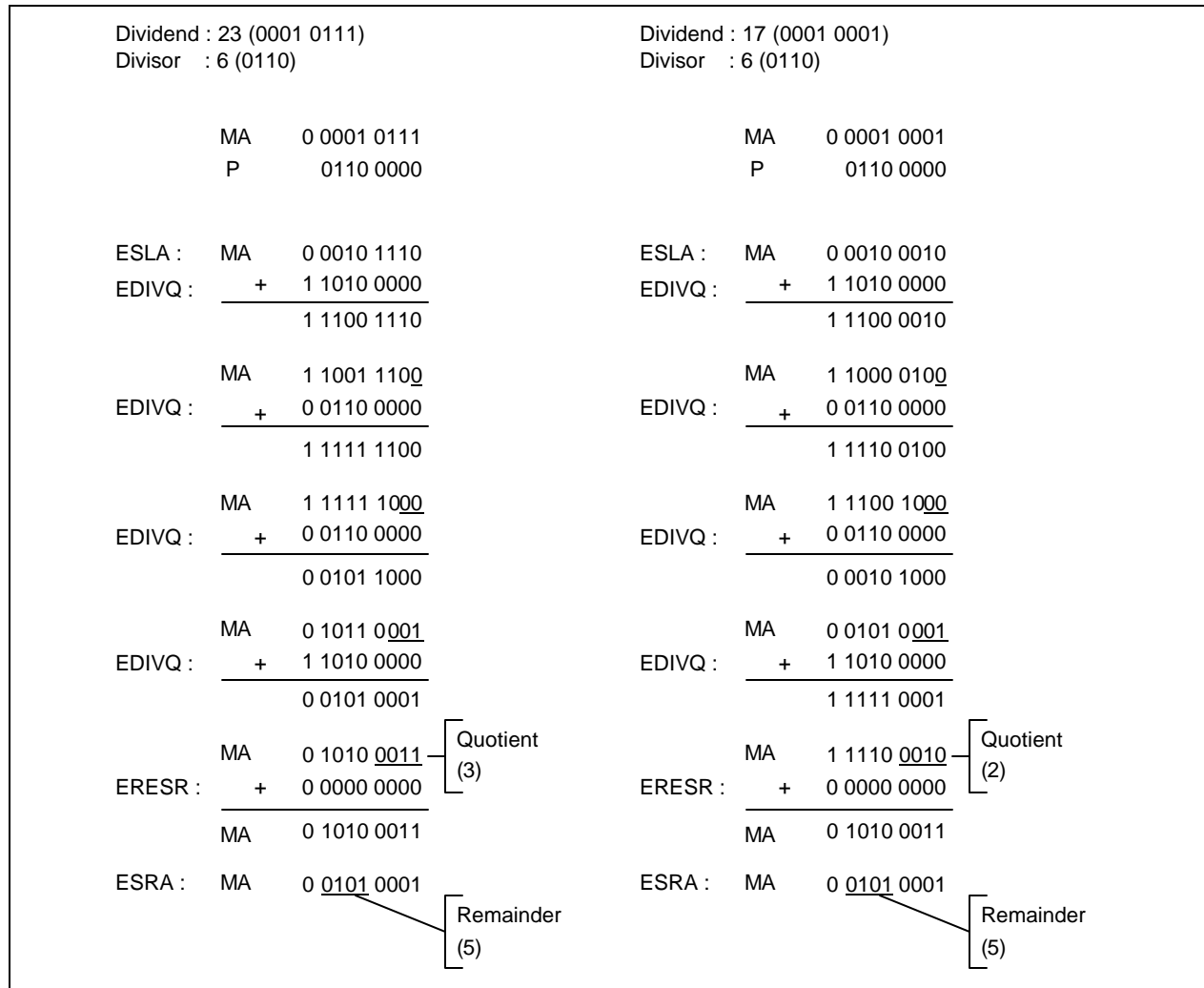


Figure 21-4. Integer Division Example

A 32/16 integer division example code is as follows

```

ER   NQ           // Initialize Division Step
ESLA MA           // Arithmetic Shift Left 1
EDIVQ MA, P       // Division Step
....
EDIVQ MA, P       // Division Step (16 times)
ERESR MA, P       // Remainder Restoring
ESRA  MA          // Arithmetic Shift Right 1

```



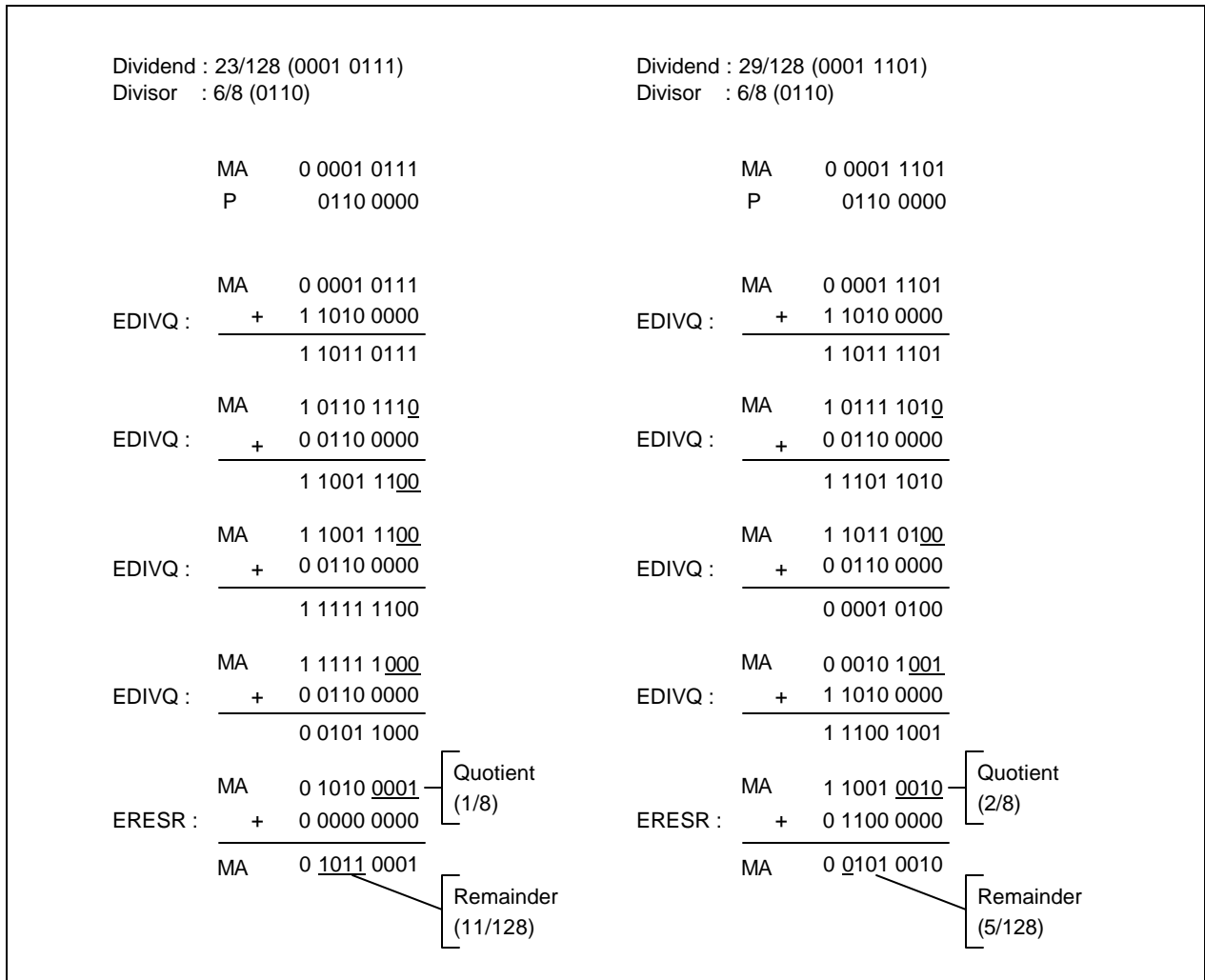


Figure 21-5. Fractional Division Example

A 32/16 fractional division example code is as follows.

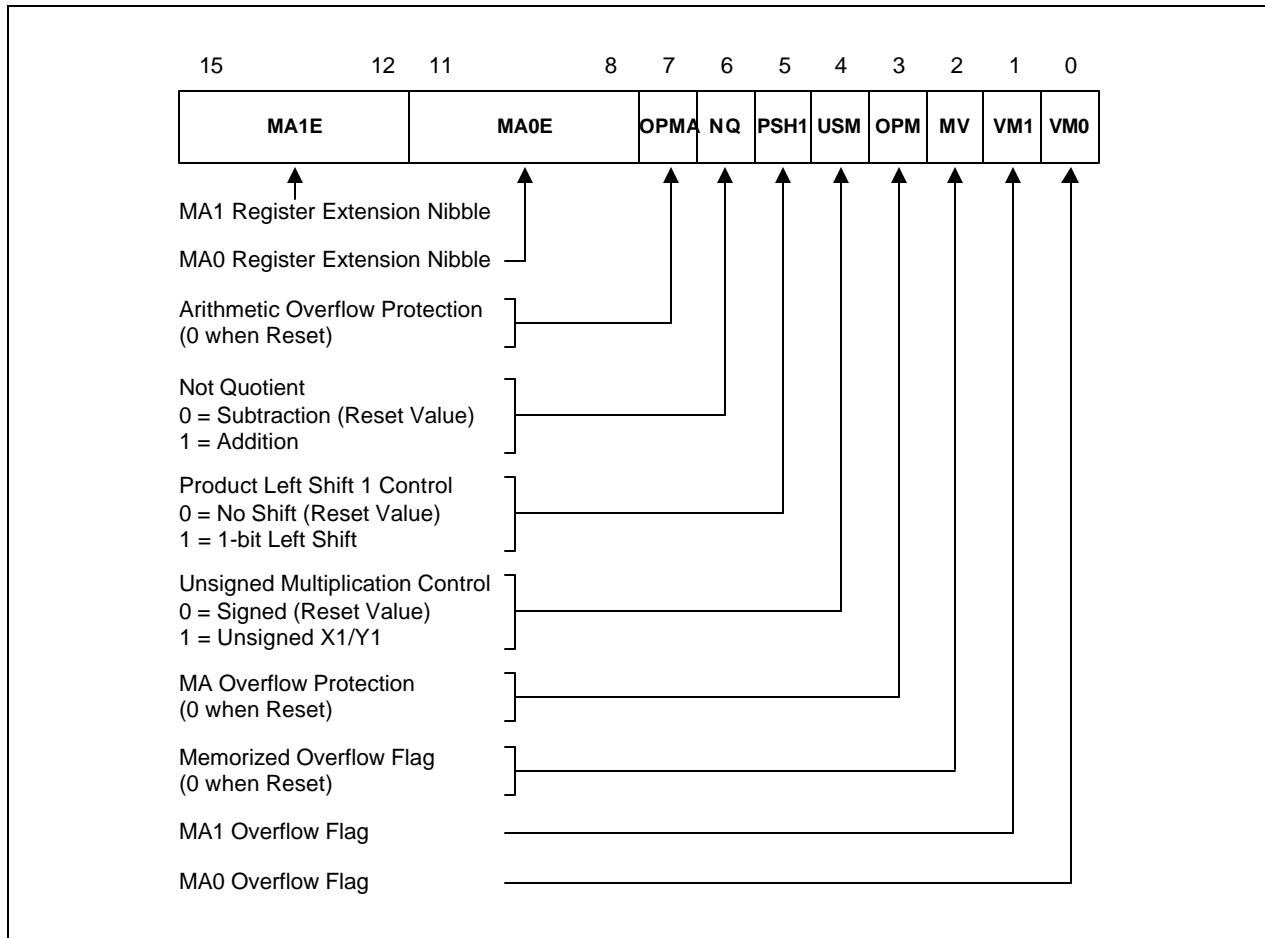
```

ER  NQ          // Initialize Division Step
EDIVQ MA, P     // Division Step
....
EDIVQ MA, P     // Division Step (16 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 previous 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 CalmMAC16 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. If MSR1 register is used as a 16-bit source operand in 16-bit arithmetic operation, the 16-bit MSR1 register is zero-extended to a 16-bit operand.



**Figure 21-6. MSR1 Register Configuration**

**MA1E/MA0E – Bit 15~12 / Bit 11~8**

These four bit nibbles are used as guard bits for MA registers. These bits are updated when MA register write operation is occurred. These bits are also written during MSR1 register write operation.

**OPMA – Bit 7**

The OPMA bit indicates that saturation arithmetic is provided or not when arithmetic operation on one of the MA registers. When the OPMA bit is set (Overflow Protection is enabled) and overflow is occurred during arithmetic operation, the saturation logic will substitute a limited data value having maximum magnitude and the same sign as the source MA register. If the OPMA bit is clear, no saturation is performed. This bit has no effect on a “ESAT” instruction, which always saturates the MA register value. The OPMA bit is modified by writing the MSR1 register or “ER/ES OPMA” instruction. The OPMA bit is cleared by a processor reset.

**NQ – Bit 6**

This bit defines next operation of division step. When this bit is clear, the next division instruction subtracts P register from MA register, and when this bit is set, the next division instruction adds P register value from MA register. It also defines next operation of restoring instruction. If this bit is set to 0, the next restoring instruction adds 0 to MA register and if this bit is set to 1, adds two times the divisor (P register value) to the MA. The NQ bit is affected when MSR1 register write operation, “ER/ES NQ” instruction, or division step (“EDIVQ” instruction) is executed. The NQ 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. 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 numbers. 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 no 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 36-bit overflow. 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 indicate arithmetic overflow on MA1 register and MA0 register respectively. One of these bits is set if an arithmetic overflow (32-bit overflow) occurs after an arithmetic operation, and cleared otherwise. It represents that the result of an operation cannot be represented in 32 bits. i.e. these bits are set when 5-bit value of MA[35:31] register is not all the same in 16-bit mode. These bits are modified by writing the MSR1 register or all instructions that write one of MA register.

## 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. In addition, it contains four alternative bank pointer register (RP0B ~ RP3B), four alternative index registers (SD0B ~ SD3B), and two alternative bank extension index register (SD0BE and SD3BE) supported by an individual bank exchange.

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 14 to bit 0 of RPi register and RPDi register defines address for X or Y memory, and the bit 15 determines whether the address is for X memory or Y memory. The bit 15 to bit 12 of MSR0 register (MEi bit) indicates whether the each pointer is updated with modulo arithmetic. The bit 15 to bit 12 of MSR2 register (BK<sub>i</sub> bit) defines the current bank of each pointer. When this bit is set to 1, the pointer register of alternative bank is selected as a address register, and the index register of alternative bank is selected as a index value. "EBK #imm:4" (Bank definition instruction) instruction specifies bank of each pointer and index register. Four bit immediate field indicates each pointer and index, i.e. bit 3 of imm:4 specifies the bank of RP3 and SD3 register, and bit 2 of imm:4 specifies the bank of RP2 and SD2 register. For example, if "EBK #1110b" instruction is executed, current bank of RP3, RP2, and RP1 is bank 1, and current bank of RP0 is bank 0. When the bank of pointer register is changed, the bank of each index register including extended index register is automatically changed. The bank of pointer can be changed by executing "EBK" instruction, "ER/ES BK<sub>i</sub>" instruction, or the instruction that writes MSR2 register.

The RPU can access two data operand simultaneously over XA and YA buses. In dual access case, RP0 or RP1 is selected as a X memory pointer and RP3 is selected as a Y memory pointer regardless of bit 15 of RP0 and RP3.

All registers in the RPU may be read or written to by the XB as 16-bit operands, thus can serve as general-purpose register. If one of the RPU register is read as a 16-bit operand, the 16-bit value is zero-extended to 16-bit value.

The detailed block diagram of the RAM Pointer Unit is shown in Figure 21-7.

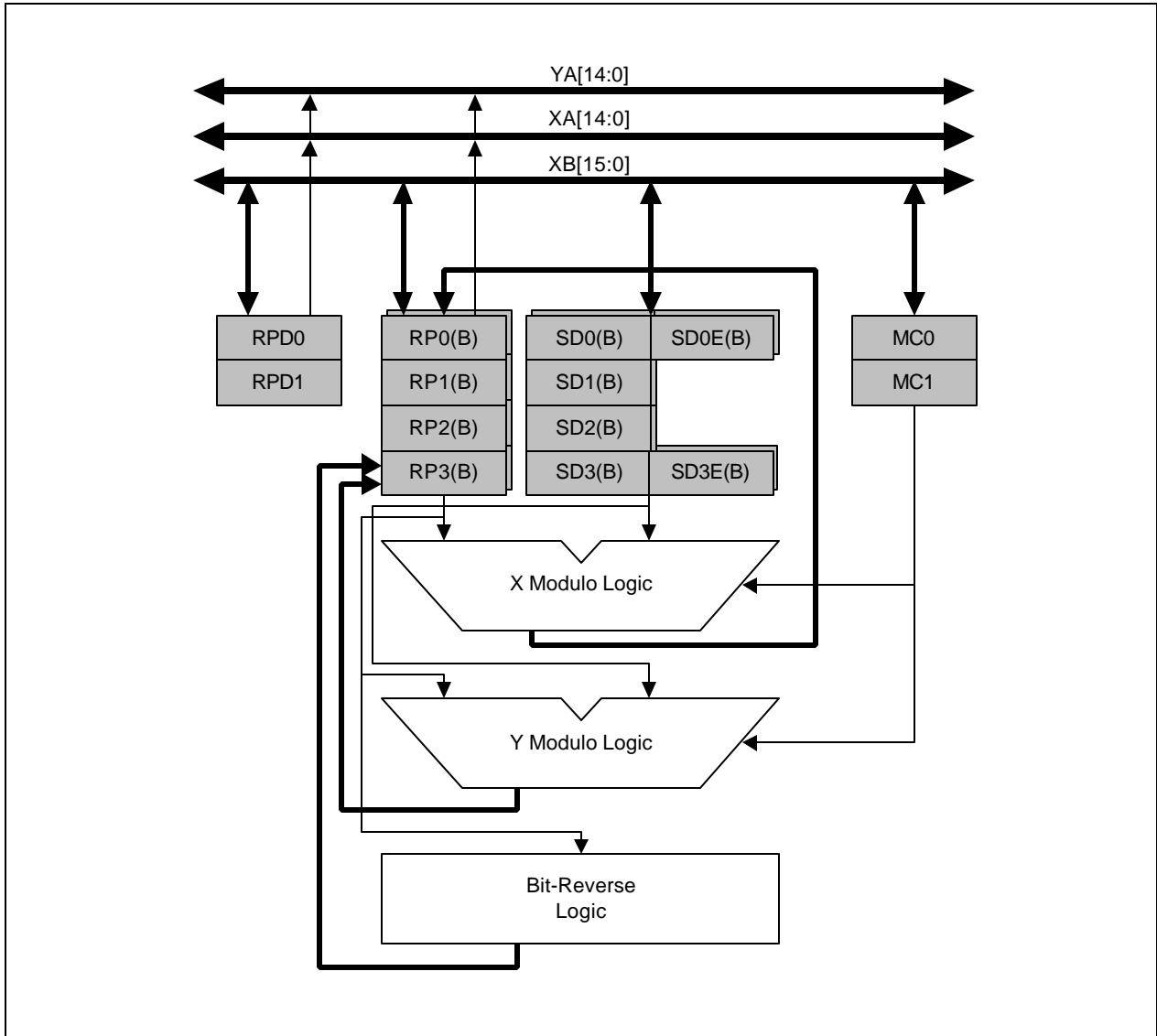


Figure 21-7. RAM Pointer Unit Block Diagram

## ADDRESS MODIFICATION

The RPU can generate up to two 15-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 16-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 an 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.

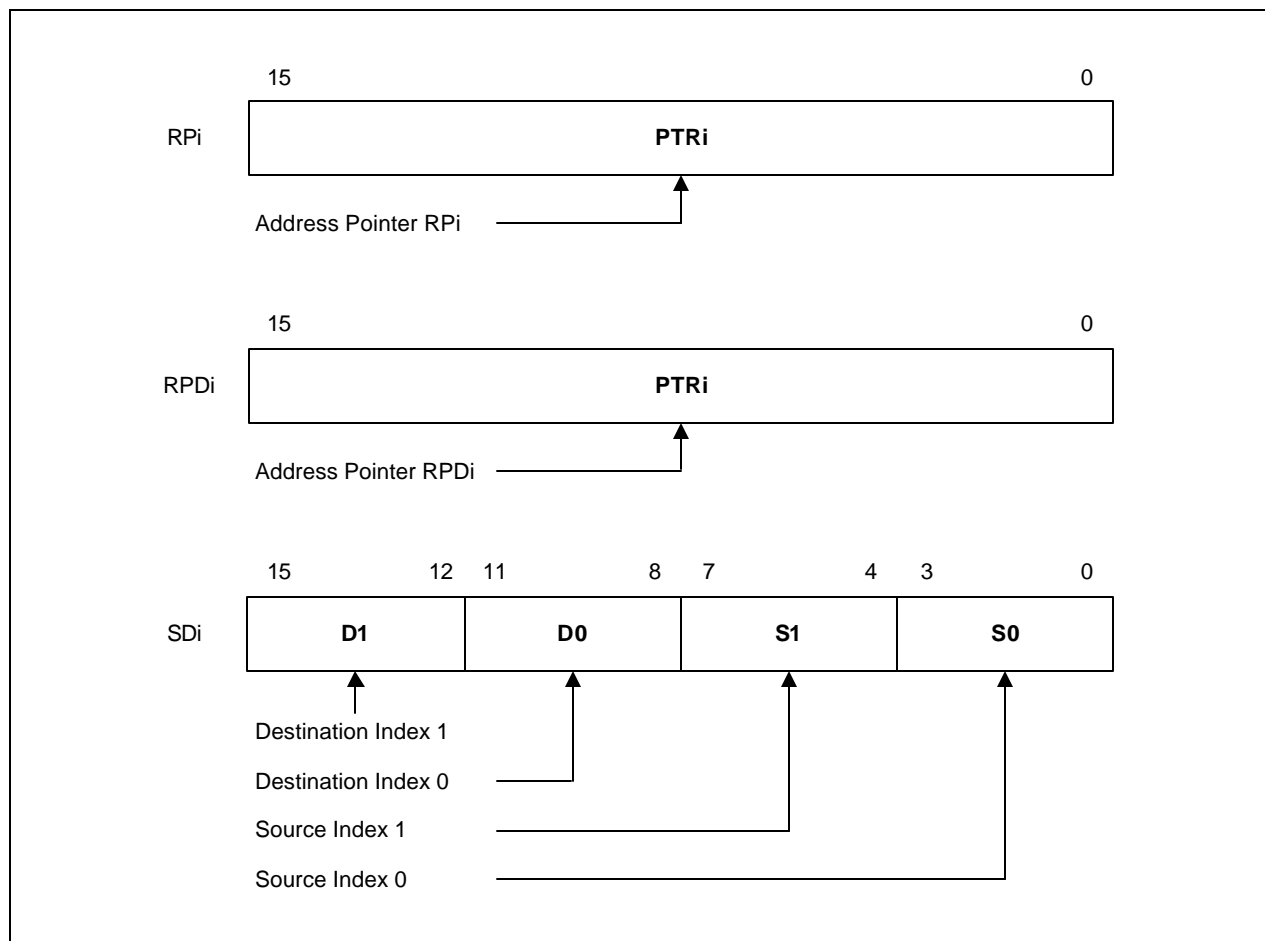


Figure 21-8. Pointer Register and Index Register Configuration

### 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 MSR0 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 21-9)

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 ((R<sub>Pi</sub> = Upper Boundary in k LSBs) and (S > 0)) then*  
     *R<sub>Pi</sub> k LSB  $\rightarrow$  0*  
*else if ((R<sub>Pi</sub> = Lower Boundary in k LSBs) and (S < 0)) then*  
     *R<sub>Pi</sub> k LSB  $\rightarrow$  Upper Boundary in k LSBs*  
*else*  
     *R<sub>Pi</sub> k LSB  $\rightarrow$  R<sub>Pi</sub> + S (k LSBs)*  
*where k is defined by MC<sub>i</sub>[12:10]*

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  $\rightarrow$  0011h  $\rightarrow$  0012h  $\rightarrow$  0013h  $\rightarrow$  0014h  $\rightarrow$  0015h  $\rightarrow$  0016h  $\rightarrow$  0017h  $\rightarrow$  0010h  $\rightarrow$  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  $\rightarrow$  0323h  $\rightarrow$  0326h  $\rightarrow$  0321h  $\rightarrow$  0324h  $\rightarrow$  0327h  $\rightarrow$  0322h  $\rightarrow$  0325h  $\rightarrow$  0320h  $\rightarrow$  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  $\rightarrow$  2012h  $\rightarrow$  2010h  $\rightarrow$  2014h  $\rightarrow$  2102h

The total number of circular buffer (modulo addressing active area) is defined by 64K / Modulo size. i.e. if current modulo size is 64, the total number of circular buffer is 1024.

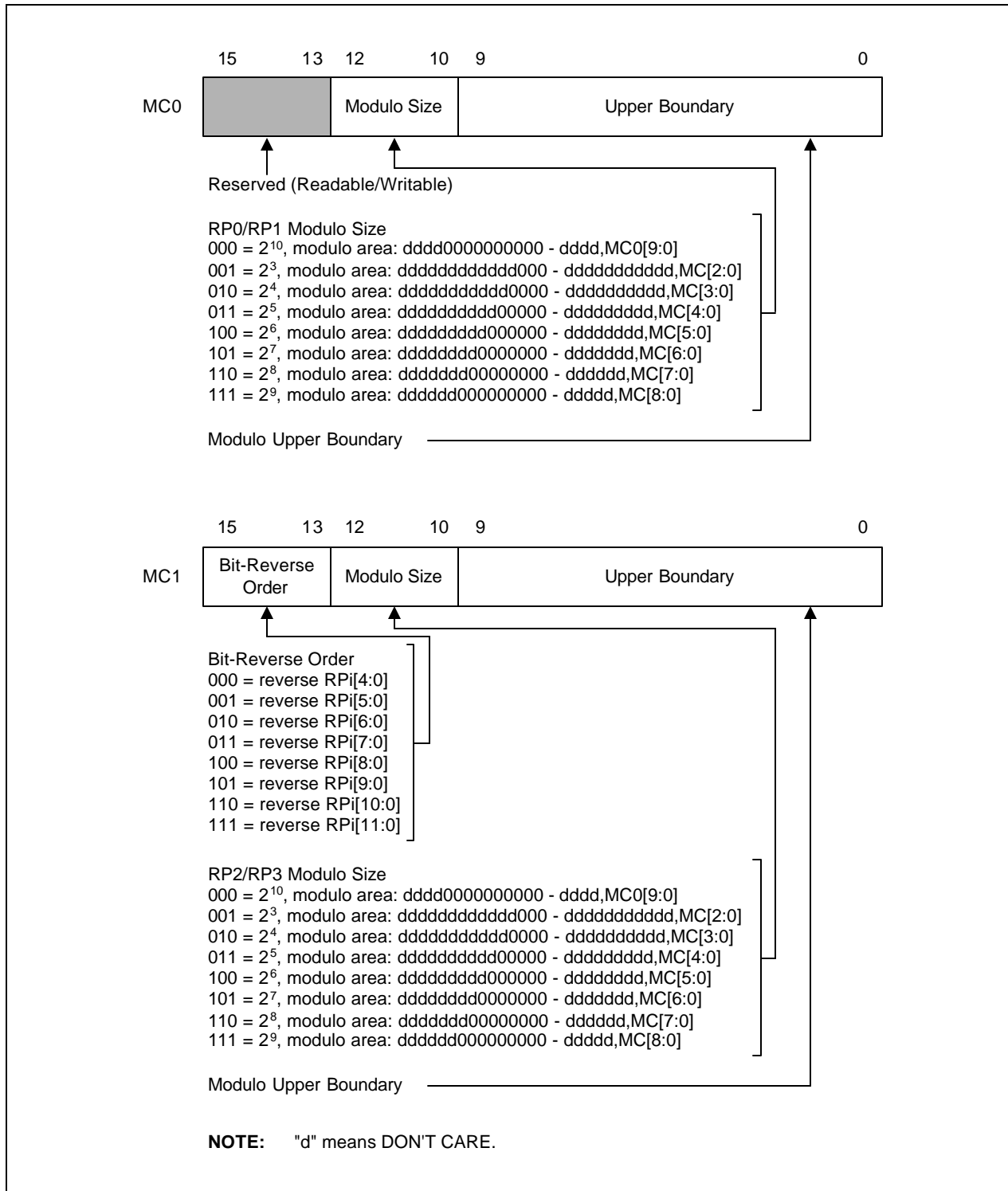


Figure 21-9. Modulo Control Register Configuration



### Bit Reverse Capabilities

The bit-reverse addressing is useful for radix-2 FFT(Fast Fourier Transform) calculations. The CalmMAC16 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. (Refer to Figure 21-9) The result bit pattern is written to the current bank RP3 register. 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

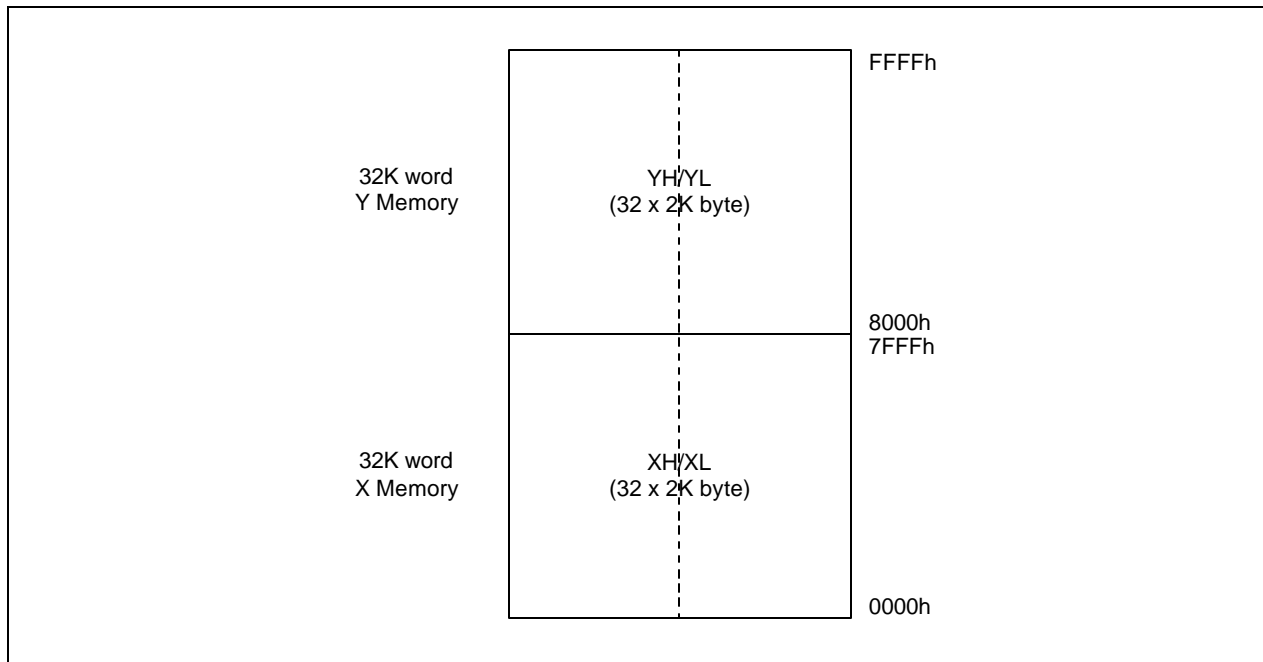
When an instruction with indirect addressing is executed, the current value of selected address pointer register RPi provides address on XA and YA buses. Meanwhile, the current address is incremented by the value contained into the selected index value contained into the selected bit field of selected 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 enabled, 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 CalmMAC16 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 22-bit data memory address, so it can access up to 4M byte data memory space.

The CalmMAC16 access data memory with 16-bit width. It can access upto 64K word (word = 2-bytes). The data space is divided into a lower 32K word X data space and a higher 32K word 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 21-10. CalmMAC16 Data Memory Space Map**

Each space is divided into three 32K byte XH/XL or YH/YL region. Each space can contain RAM or ROM, and can be off-chip or on-chip. The configuration of this region depends on the specific chip configuration. (Figure 21-10) 16-bit data of X memory (XH and XL memory), 16-bit data of Y memory (YH and YL memory), can be allocated to any 256K byte region from 4M byte data memory space of CalmRISC16. The X memory space and Y memory space can be mapped in the separated region, but CalmMAC16 can access a continuous data space i.e. looking at the two memory as a single continuous data memory.

The data memory space of CalmMAC16 may contain slow memories and peripherals as well as fast memories and peripherals. When using slow memories, additional wait cycles have to be inserted through DBWAIT pin of CalmMAC16.

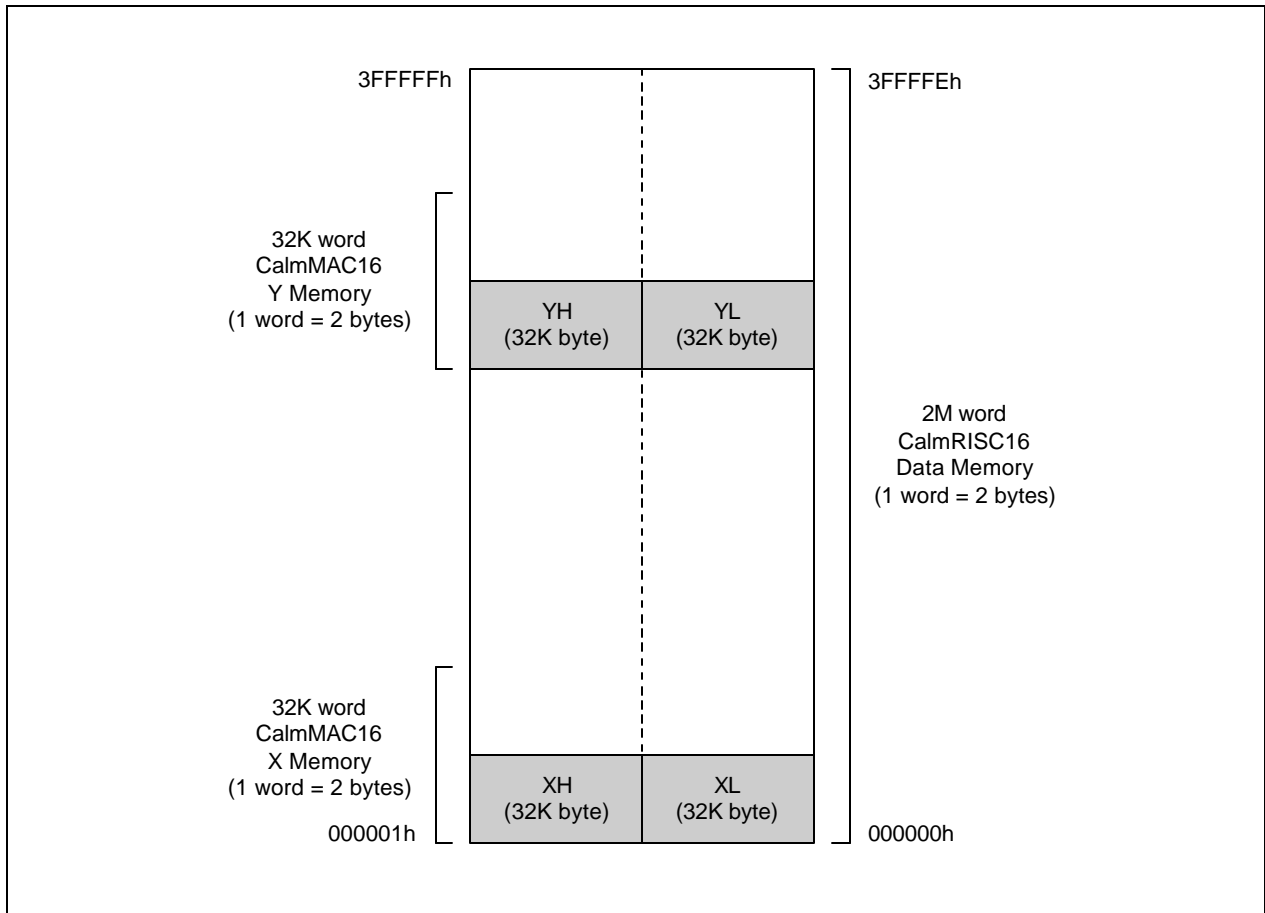


Figure 21-11. CalmMAC16 Data Memory Allocation

## ARITHMETIC UNIT

The Arithmetic Unit (ARU) performs all arithmetic operations on data operands. It is a 16-bit, single cycle, non-pipelined arithmetic unit. The CalmMAC16 is a coprocessor of CalmRISC16 microcontroller. So, all the logical operation and other bit manipulation operations can be performed in CalmRISC16. Thus, the CalmMAC16 has not logical units and bit manipulation units at all.

The ARU receives one operand from  $A_i$  (A or B) or  $C_i$  (C or D) register, and another operand from either the MSB part of MA register, the XB bus, or from  $A_i$  or  $C_i$ . Operations between the two accumulator registers are possible. The source and destination accumulator register of an ARU instruction is always the same. The XB bus input is used for transferring one of the CalmMAC16 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. But the flags are not affected during data load from data memory location to a accumulator or during CLD instruction. In most of the instructions where the ARU result is transferred to one of accumulator registers, the flags represent the accumulator register status. The detailed block diagram of the Arithmetic Unit is shown in Figure 21-12.

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.

### Main Accumulators : A/B

Each  $A_i$  (A or B) register is organized as a regular 16-bit register. The  $A_i$  accumulators can serve as the source operand, as well as the destination operand of all ARU instructions and serve as a source operand of exponent instruction. The  $A_i$  registers can be read or written though the XB bus. It can be read or written to the data memory during some MAU instructions and some ARU instructions (parallel move)

### Auxiliary Accumulators : C/D

Each  $C_i$  (C or D) register is organized as a regular 16-bit register and can serve as the source operand, as well as the destination operand of some ARU instructions and serve as a source operand of exponent instruction. Some ARU instruction can only access main accumulators A/B as a source or destination operand, and auxiliary accumulators C/D are only accessed in some special instructions. The  $C_i$  registers can be read or written though the XB bus. It can be read or written to the data memory during some ARU instructions (parallel move)

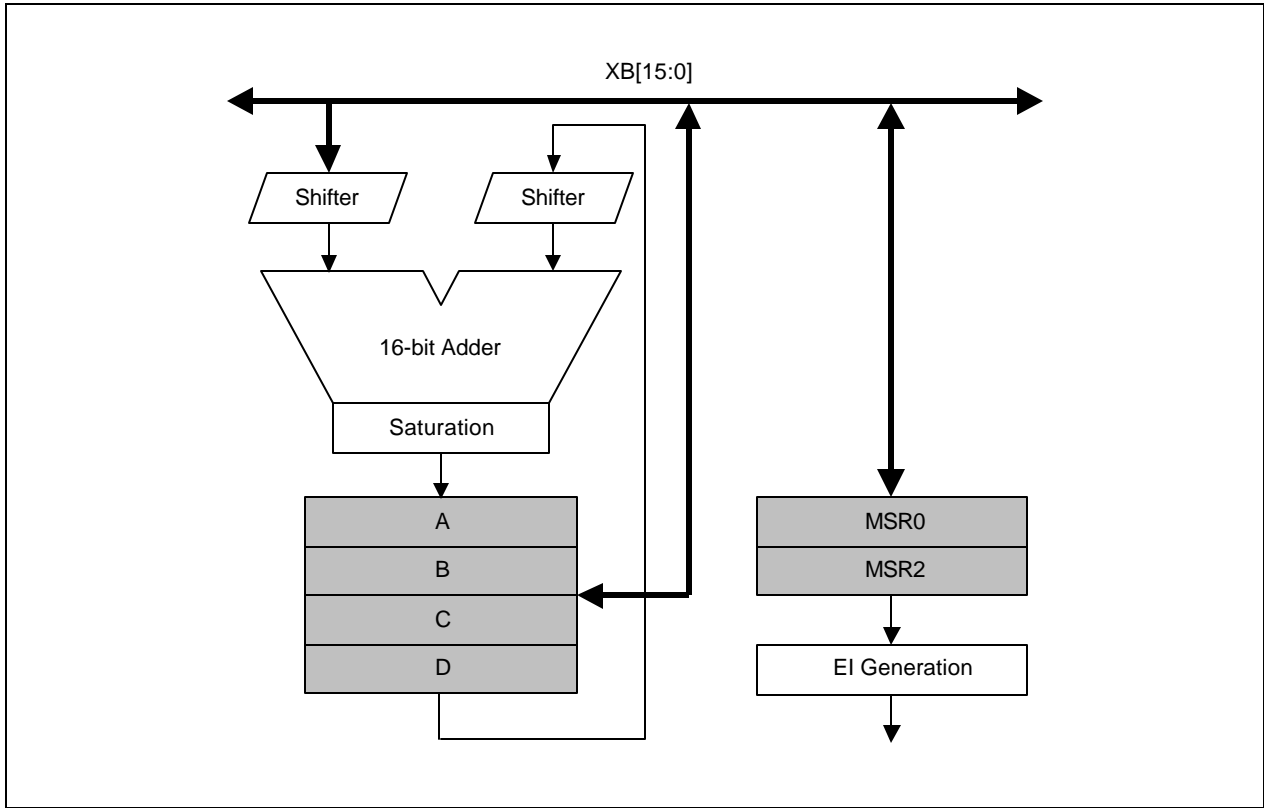
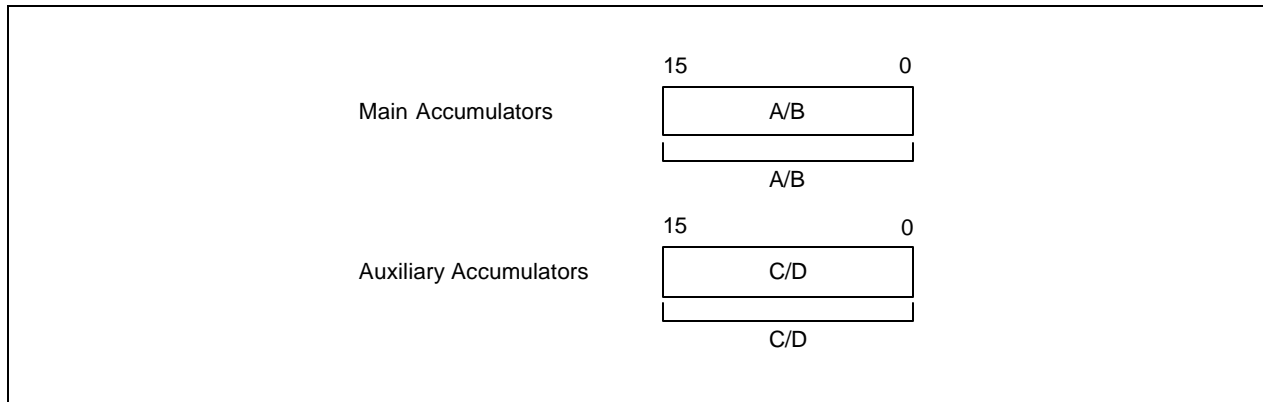


Figure 21-12. Arithmetic Unit Block Diagram

**Overflow Protection in Accumulators**

The Ai or Ci accumulator saturation is performed during arithmetic operation that causes overflow, if overflow protection bit (OP in MSR0 register) is enabled. The limited values are 7FFFh (positive overflow), or 8000h (negative overflow). During accumulator register read through XB bus, the saturation is not occurred.

- Saturation Condition: Arithmetic instruction & 16-bit Overflow & OP



**Figure 21-13. Accumulator Register Configuration**

### Maximum-Minimum Possibilities

A single Cycle maximum/minimum operation is available with pointer latching and modification. One of the  $A_i$  accumulator registers, defined in the instruction, holds the maximum value in a “EMAX” instruction, or the minimum value in a “EMIN” instruction. In one cycle, the two accumulators are compared, and when a new maximal or minimal number is found, this value is copied to the above defined accumulator. In the same instruction, one of pointer register  $RP_i$  (except  $RP_3$  pointer) can be used as a buffer pointer. The address pointer register that generates address can be post-modified according to the specified mode in the instruction. When the new maximum or minimum number is found, the address register (user invisible register) value is latched into  $RP_3$  pointer register. The address register stores original pointer register value during pointer modification instructions (instructions with indirect addressing, “ERPS/ERPD” instruction, or “ERP\_N” instruction). For more details, refer to “EMAX” and “EMIN” instructions in chapter 4 on the instruction set.

The examples which searches block elements are as follows

```
ELD C, @RP0+S0 // 1st Data load
```

Loop\_start:

```
EMAX(EMIN) A,C C,@RP0+S0// 1st Min/Max evaluation, 2nd Data load
```

```
JP Loop_start
```

```
EMAX(EMIN) A,C // Last Min/Max evaluation
```

### Conditional Instruction Execution

Some instructions can be performed according to the T flag value of  $MSR_0$  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 16-bit  $A_i$  or  $C_i$  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 CalmMAC16. The instructions with suffix "C" indicates that the operation is performed on source operand and current carry flag value. By using these 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 + C:D)
EADD B, D      // Lower Part Addition
EINCC A       // Carry Propagation
EADD A, C      // Higher Part Addition
```

### EXTERNAL CONDITION GENERATION UNIT

The CalmMAC16 can generate and send the status information or control information after instruction execution to the host processor CalmRISC16 through EI[3:0] pin (Refer to Pin Diagram). The CalmRISC16 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 for EI[2:0]. (Refer to MSR2 register configuration) EI[3] pin selects one of status register value or combination of status register values according to the test field of "ETST cc EC3" instruction. So, the EI[2:0] pin is always changes the value if corresponding status register bit value is changed, but EI[3] is only changed after executing "ETST cc EC3" instruction. (Refer to "ETST" instruction)

In a high speed system, which operates at full clock speed (32 MHz) with CalmRISC16 and CalmMAC16, a branch instruction using EI[2:0] value as a branch condition can not immediately follow the instruction that changes EI[2:0] value. In this case, a "NOP" (no operation) instruction must be inserted between the branch instruction and the ARU instruction. On the other hand, in a medium and low speed system, the branch instruction can immediately follow any instruction that changes EI values. The following shows the examples.

```
// Branching in high speed system
EADD A,C      // Update Status Flags & EI[2:0]
ENOP
BRA EC0, Label1

// Branching in medium to low speed system
EADD B,D      // Update Status Flags & EI[2:0]
BRA EC1, Label2
```

In case of branch instruction using EI[3] as a branch condition, a "ETST cc EC3" instruction must be executed before the branch instruction, because only the "ETST" instruction evaluates the EI[3] pin values. The following shows an example of branching with EI[3]

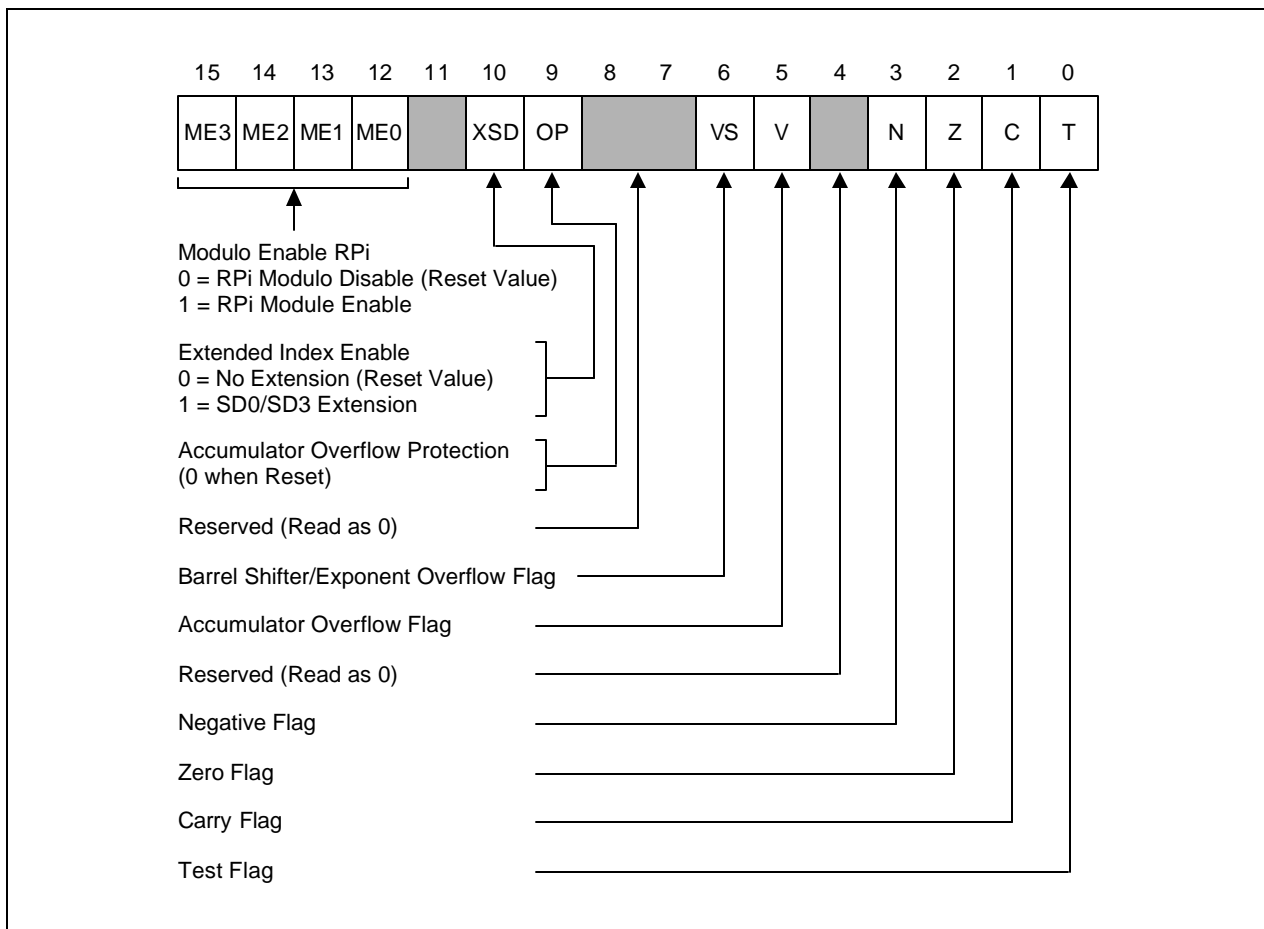
```
// Branching with EI[3]
EADD A,C      // Update Status Flags
ETST NC, EC3  // Update EI[3] port value
BRA EC3, Label3
```

**STATUS REGISTER 0 (MSR0)**

MSR0 register of three CalmMAC16 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.

**ME3/ME2/ME1/ME0 – Bit 15 – Bit 12**

These bits define modulo options of the corresponding pointer register for address modification. When this bit is cleared, the current bank of corresponding RPi register will be modified as specified by the instruction regardless of the modulo options that is specified in MCI registers. When this bit is set, the current bank of pointer register will be modified using the suitable modulo. The MEi bits are cleared by a processor reset. The MEi bits can be modified by writing to MSR0 register, or “ER/ES” instruction.



**Figure 21-14. MSR0 Register Configuration**



**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 XS D” instruction. The XSD bit is cleared by a processor reset.

**OP – Bit 9**

The OP bit indicates that saturation arithmetic in the ARU is provided or not when overflow is occurred during arithmetic operation. The overflow protection can be applied to all of the four accumulator registers. If this bit is set, the saturation logic will substitute a limited value having maximum magnitude and the same sign as the source accumulator register during overflow. If clear, no saturation is performed, and overflow is not protected by the CalmMAC16. The OP bit can be modified by writing to MSR0 register or “ER/ES OP” instruction. The OP 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 V 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.

**V – Bit 5**

The V bit is a overflow flag for ARU accumulators. This bit is set if arithmetic overflow is occurred during arithmetic operation on a destination accumulator register in ARU. The V bit indicates that the result of an arithmetic operation can not be represented in 16-bit accumulator register. The V 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 cc T” 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 CalmMAC16 status registers (MSR0, MSR1, MSR2) is used to select EI[2:0] port of the CalmMAC16 from various flags and status information in MSR0 and MSR1 register and to specify current bank of each pointer and index register. The MSR2 register is used at external condition generation unit in the ARU. The contents of each field definitions are described as follows.

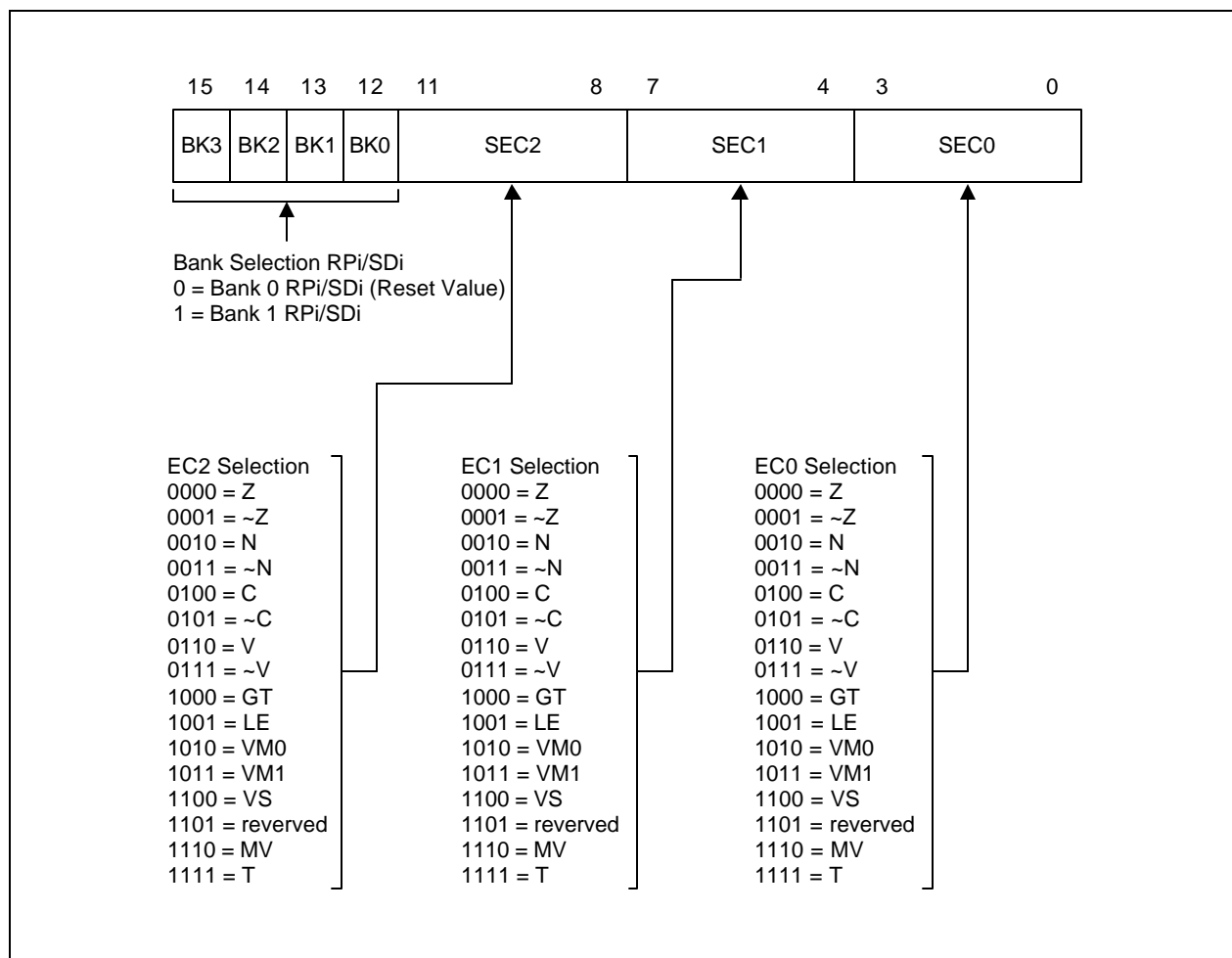


Figure 21-15. MSR2 Register Configuration

**BK3/BK2/BK1/BK0 – Bit 15 – Bit 12**

These bits define current banks of the corresponding pointer and index register for address generation and address modification.

Clear - bank 0 pointer and index register is selected

Set - bank 1 pointer and index register is selected.

The BK<sub>i</sub> bits are cleared by a processor reset. The BK<sub>i</sub> bits can be modified by writing to MSR2 register, “ER/ES BK<sub>i</sub>” instruction, or “EBK” instruction. The writing to MSR2 and “EBK” instruction can change the whole four banks of each pointer register and index register. On the other hand, “ER/ES” instruction changes only one bank of pointer and index register.

**SEC2/SEC1/SEC0 – Bit 11 – Bit 0**

These bits defines the logic state of the EI[2:0] pin according to status information of CalmMAC16 processor. For example, if SEC2 value is “0000b”, the EI[2] pin monitors Z flag value of MSR0 register. The logic state of the EI pin is changed immediately after SEC<sub>i</sub> bit field value is changed or corresponding condition flag bit value is changed. The SEC<sub>i</sub> bits can be modified by a instruction writing to the MSR2 register, or “ESEC<sub>i</sub>” instructions.

## 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 16-bit exponent evaluation unit. The detailed block diagram of the Barrel Shifter and Exponent Unit is shown in Figure 21-16.

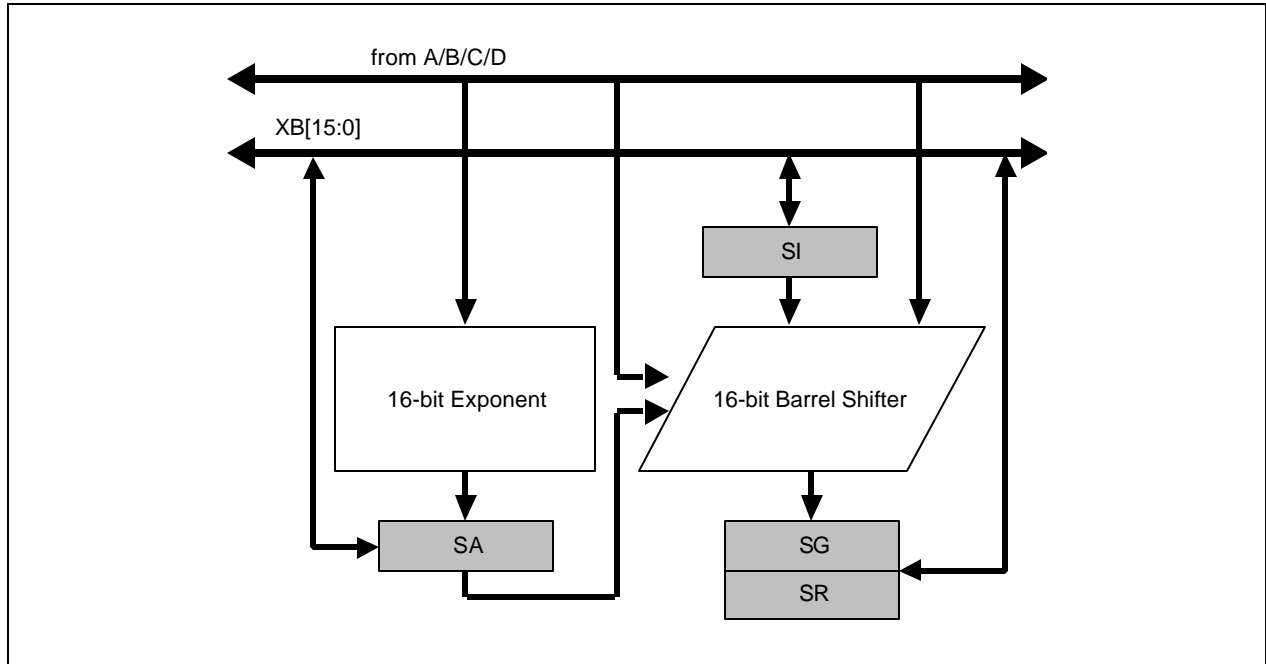


Figure 21-16. Barrel Shifter and Exponent Unit Block Diagram

## 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 16-bit two  $A_i$  (A or B) accumulator registers or 16-bit SI register. It also receives the shift amount value from either one of the 16-bit two  $A_i$  accumulator registers or 7-bit SA register. Because the maximum amount of shift is from  $-32$  (right shift 32-bit) to  $+32$  (left shift 32 bit), 7-bit shift amount is sufficient. When  $A_i$  register is used as the shift amount register, 7 LSBs of 16-bit register value are only valid. If the shift value is greater than 32 or less than  $-32$ , the shifter generates the same result as shift 32-bit or shift  $-32$ -bit. 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 7-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 V flag 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 2.16. If 7-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
ESFT    A,SA // Lower Part Shift
```

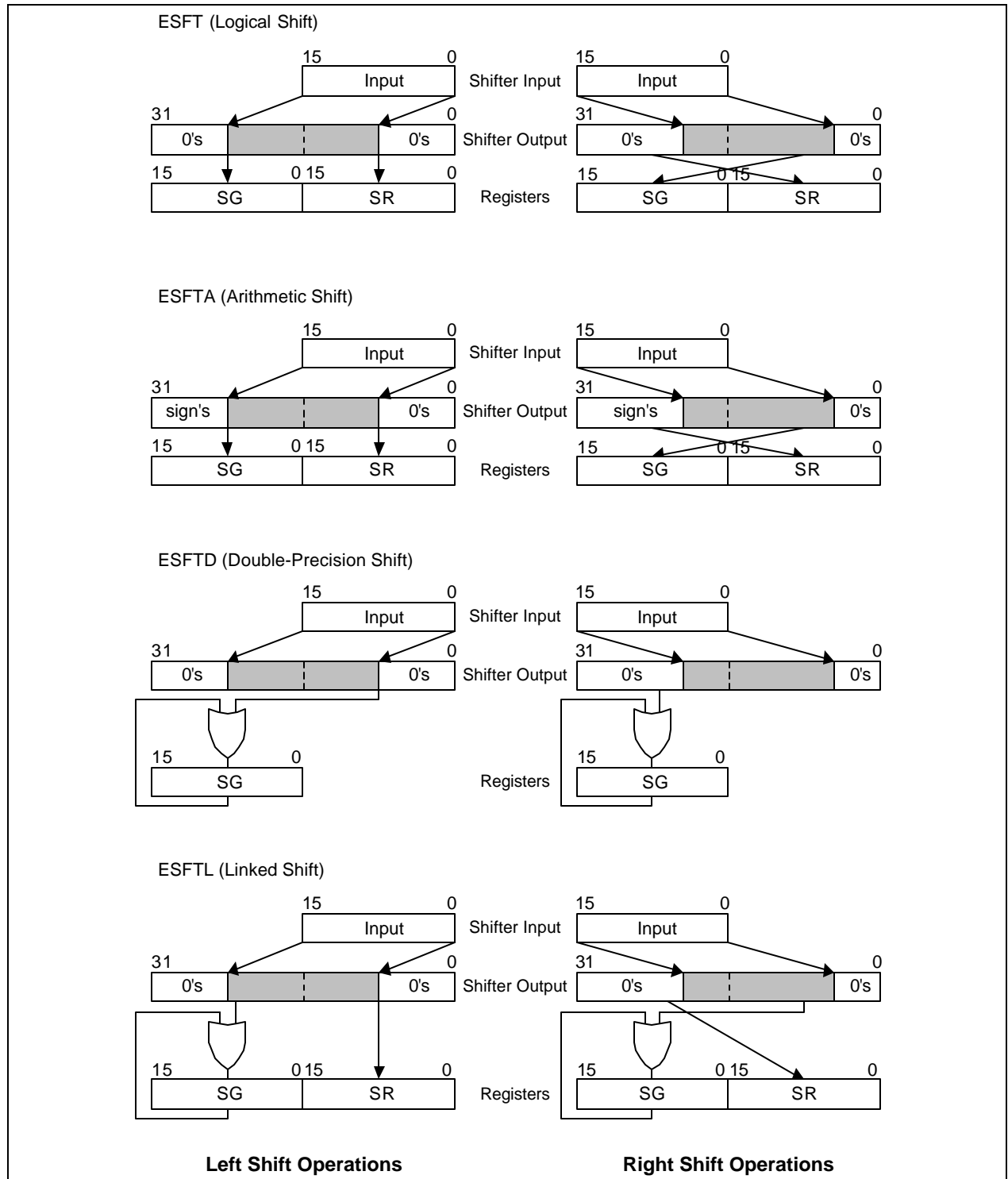


Figure 21-17. Various Barrel Shifter Instruction Operation

“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
```

### 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 CalmRISC16, host processor. So the CalmMAC16 need not powerful bit operation capabilities. “ENMSK” instruction is provided for bit-pattern masking. This instruction 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 four 16-bit accumulator registers A, B, C, D. The result of this operation is a signed 7-bit value, and transferred into the Shift Amount register (SA). The source operand is unaffected by this calculation.

**Table 21-1. 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.....

The algorithm for determining the exponent result for a 16-bit number is as follows. Let N be the number of the sign bits (i.e. the number of MSBs equal to bit 15) found in the evaluated number. The exponent result is N-1. This means that the exponent is evaluated with respect to bit 16. 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 accumulator register, 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.

### 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 CalmMAC16 DSP coprocessor has an instruction which can evaluate exponent values of double precision 32-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 16-bit data. After "EEXP" instruction is executed, the "EEXPC" instruction evaluates the exponent value of lower 16-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 CalmMAC16 coprocessor.

#### Indirect Addressing Mode

##### Indirect Addressing for Single Read Operation

**@RP0+S0 / @RP0+S1 / @RP1+S0 / @RP1+S1 /  
@RP2+S0 / @RP2+S1 / @RP3+S0 / @RP3+S1**

One of the current bank pointer registers (RP0, RP1, RP2, RP3) points to one of the 64K 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 16-bit and added to 16-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.

##### Indirect Addressing for Dual Read Operation

**@RP0+Si (i = 0,1) and @RP3+Si (i = 0,1)  
@RP1+Si (i = 0,1) and @RP3+Si (i = 0,1)**

One of the current bank pointer registers RP0 or RP1 points to one of the lower 32K data words (X data memory), and the current bank RP3 pointer register points to one of the upper 32K data words (Y data memory). The data location contents, pointed to by the pointer registers, are the source operands. The pointer registers are 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 16-bit and added to 16-bit pointer values in pointer registers. The RP1 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	8010h	<b>0011h</b>
RP0 (no modulo)	0010h	<b>0033h</b>
Data Location 10h	0011h	0011h
SD0E	01 <u>2</u> 2h	0122h
SD0	F <u>3</u> 3h	F33h

Figure 21-18. Indirect Addressing Example I (Single Read Operation)

<b>ELD X0, @RP1+S0, Y1, @RP3+S1 (When XSD = 0)</b>		
	Before Execution	After Execution
X0	3456h	<b>4321h</b>
Y1	9ABCh	<b>A987h</b>
RP1 (no modulo)	1001h	<b>1000h</b>
RP3 (no modulo)	8001h	<b>8003h</b>
Data in 1001h	4321h	4321h
Data in 8001h	A987h	A987h
SD1	1F1Fh	1F1Fh
SD3	2E2Eh	2E2Eh

**Figure 21-19. Indirect Addressing Example II (Dual 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 current pointer registers (RP0, RP1, RP2, RP3) points to one of the 64K data words. The data location content, pointed to by the pointer register, is the destination operand. The RP<sub>i</sub> 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 16-bit and added to 16-bit pointer value in RP<sub>i</sub> 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.

<b>ELD @RP1+D0, B</b>		
	Before Execution	After Execution
B	8010h	8010h
RP1 (no modulo)	0020h	<b>0018h</b>
Data Location 20h	0011h	<b>8010h</b>
SD1	<b>18</b> 19h	1819h

**Figure 21-20. Indirect Addressing Example III (Write Operation)**

**Direct Addressing Mode**

**Short direct Addressing**

**RPD0.adr:5 / RPD1.adr:5**

The data location, one of the 64K data word, is one of the source operand or destination operand. The 16-bit data location is composed of the page number in the MSB 11 bits of RPD0 or RPD1 register and the direct address field (the offset in the page) in the instruction code. The short direct addressing uses RPD0 or RPD1 register specified in instruction code as a page value. The LSB 5 bits of RPD0 or RPD1 register is not used at all.

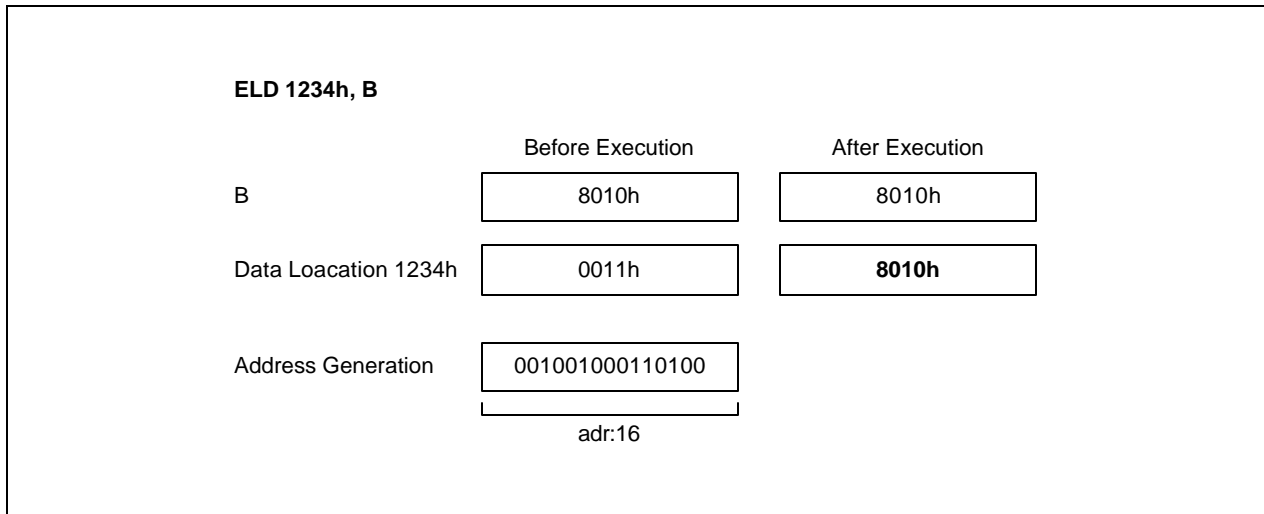
<b>ELD A, RPD0.3h</b>				
	Before Execution	After Execution		
A	8010h	<b>0011h</b>		
RPD0	0028h	0028h		
Data Location 23h	0011h	0011h		
Address Generation	<table border="1"> <tr> <td>000000001</td> <td>00011</td> </tr> </table>	000000001	00011	
000000001	00011			
	<table border="1"> <tr> <td>RPD0[15:5]</td> <td>adr:5</td> </tr> </table>	RPD0[15:5]	adr:5	
RPD0[15:5]	adr:5			

**Figure 21-21. Short Direct Addressing Example**

## Long Direct Addressing

**adr:16**

The data location, one of the 64K data word, is one of the source operand or destination operand. The 16-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.

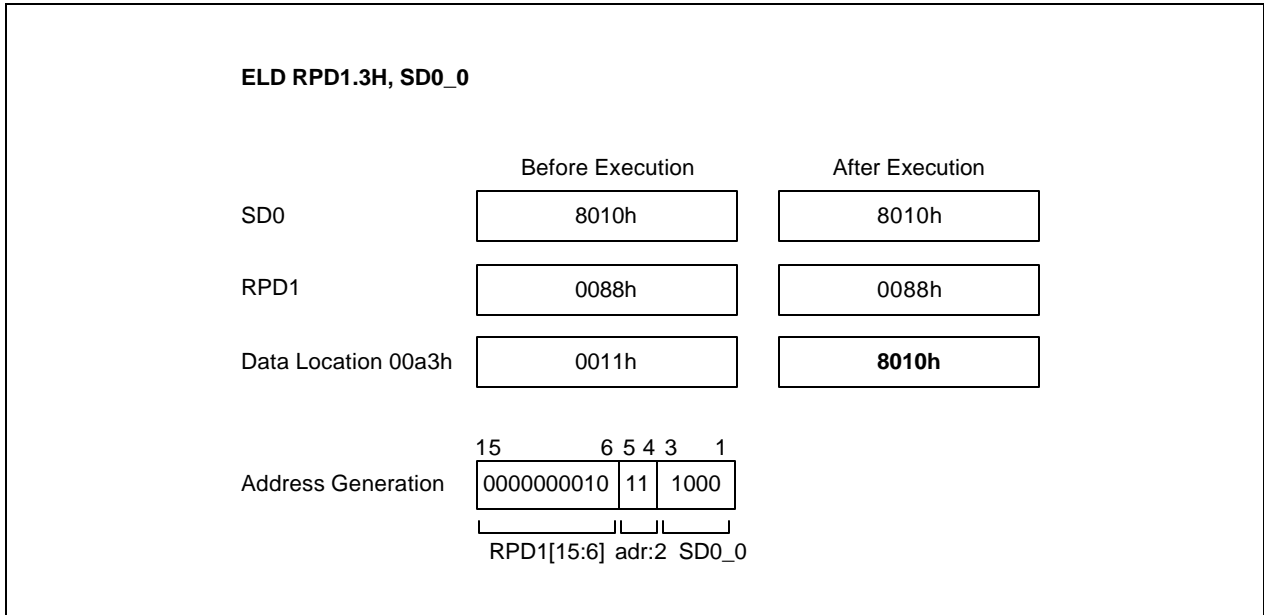


**Figure 21-22. Long Direct Addressing Example**

## Short Direct Associated Addressing

**RPD1.adr:2**

The data location, one of the 64K data word, is one of the source operand or destination operand. The 16-bit data location is composed of the page number in the MSB 10 bits of RPD1 register, the 2-bit direct address field (the offset in the page) in the instruction code, and destination or source register name itself. The source or destination register will be one of a set of pointer register (RP0 ~ RP3), two sets of index register (SD0\_0 ~ SD3\_0 and SD0\_1 ~ SD3\_1), and two sets of modulo control register (MC0\_0 ~ MC1\_0 and MC0\_1 ~ MC1\_1). One of 16 registers itself specifies 4-bit address field. With this addressing mode, user can keep up to 4 sets of pointer registers, 8 set of index registers, and 8 set of modulo control registers at one time. The short direct associated addressing uses only RPD1 register as a page value. The LSB 6 bits of RPD0 register is not used at all.



**Figure 21-23. Short Direct Associated Addressing Example**

**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, “EBK” 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 16-bit operand when the destination register has 16-bit width. If the destination register has 16-bit width, it is sign-extended to the 16-bit operand.

**Long Immediate**

**#imm:16**

The long immediate form is used for one of the source operands. The 16-bit value is right-justified and sign-extended to the 16-bit operand when the destination operand is 16-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****Abbreviation Definition and Encoding**

- rps

<b>Mnemonic</b>	<b>Encoding</b>	<b>Description</b>
RP0+S0	000	RP0 post-modified by SD0 S0 field
RP1+S0	001	RP1 post-modified by SD1 S0 field
RP2+S0	010	RP2 post-modified by SD2 S0 field
RP3+S0	011	RP3 post-modified by SD3 S0 field
RP0+S1	100	RP0 post-modified by SD0 S1 field
RP1+S1	101	RP1 post-modified by SD1 S1 field
RP2+S1	110	RP2 post-modified by SD2 S1 field
RP3+S1	111	RP3 post-modified by SD3 S1 field

- rpd

<b>Mnemonic</b>	<b>Encoding</b>	<b>Description</b>
RP0+D0	000	RP0 post-modified by SD0 D0 field
RP1+D0	001	RP1 post-modified by SD1 D0 field
RP2+D0	010	RP2 post-modified by SD2 D0 field
RP3+D0	011	RP3 post-modified by SD3 D0 field
RP0+D1	100	RP0 post-modified by SD0 D1 field
RP1+D1	101	RP1 post-modified by SD1 D1 field
RP2+D1	110	RP2 post-modified by SD2 D1 field
RP3+D1	111	RP3 post-modified by SD3 D1 field

- rp01s

<b>Mnemonic</b>	<b>Encoding</b>	<b>Description</b>
RP0+S0	00	RP0 post-modified by SD0 S0 field
RP1+S0	01	RP1 post-modified by SD1 S0 field
RP0+S1	10	RP0 post-modified by SD0 S1 field
RP1+S1	11	RP1 post-modified by SD1 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

- mg1

Mnemonic	Encoding	Description
Y0	000	Y0[15:0] register
Y1	001	Y1[15:0] register
X0	010	X0[15:0] register
X1	011	X1[15:0] register
MA0(H)	100	MA0[35:0] / MA0[31:16] register
MA0L	101	MA0[15:0] register
MA1(H)	110	MA1[35:0] / MA1[31:16] register
MA1L	111	MA1[15:0] register

- mg2

Mnemonic	Encoding	Description
RP0	000	Current bank RP0[15:0] register
RP1	001	Current bank RP1[15:0] register
RP2	010	Current bank RP2[15:0] register
RP3	011	Current bank 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

- sdi

Mnemonic	Encoding	Description
SD0	00	Current bank SD0[15:0] register (SD0 or SD0E)
SD1	01	Current bank SD1[15:0] register
SD2	10	Current bank SD2[15:0] register
SD3	11	Current bank SD3[15:0] register (SD3 or SD3E)

- Ai

Mnemonic	Encoding	Description
A	0	A[15:0] register
B	1	B[15:0] register

- Ci

Mnemonic	Encoding	Description
C	0	C[15:0] register
D	1	D[15:0] register

- An

Mnemonic	Encoding	Description
A	00	A[15:0] register
B	01	B[15:0] register
C	10	C[15:0] register
D	11	D[15:0] register

- rpui

Mnemonic	Encoding	Description
RP0	0000	Current bank RP0[15:0] register
RP1	0001	Current bank RP1[15:0] register
RP2	0010	Current bank RP2[15:0] register
RP3	0011	Current bank RP3[15:0] register
MC0_0	0100	MC0[15:0] register (set 0)
MC1_0	0101	MC1[15:0] register (set 0)
MC0_1	0110	MC0[15:0] register (set 1)
MC1_1	0111	MC1[15:0] register (set 1)
SD0_0	1000	Current bank SD0[15:0] register (set 0)
SD1_0	1001	Current bank SD1[15:0] register (set 0)
SD2_0	1010	Current bank SD2[15:0] register (set 0)
SD3_0	1011	Current bank SD3[15:0] register (set 0)
SD0_1	1100	Current bank SD0[15:0] register (set 1)
SD1_1	1101	Current bank SD1[15:0] register (set 1)
SD2_1	1110	Current bank SD2[15:0] register (set 1)
SD3_1	1111	Current bank SD3[15:0] register (set 1)



- mga

Mnemonic	Encoding	Description
MA0	00	MA0[35:0] / MA0[31:16] register
MA1	01	MA1[35:0] / MA1[31:16] register
A	10	A[15:0] register
B	11	B[15:0] register

- mgx

Mnemonic	Encoding	Description
Y0	00	Y0[15:0] register
Y1	01	Y1[15:0] register
X0	10	X0[15:0] register
X1	11	X1[15:0] register

- mg

Mnemonic	Encoding	Description
MA0(H)	00000	MA0[35:0] / MA0[31:16] register
MA0L	00001	MA0[15:0] register
MA1(H)	00010	MA1[35:0] / MA1[31:16] register
MA1L	00011	MA1[15:0] register
MA0SR	00100	Arithmetic right one bit shifted MA0[31:16] register
MA0SL	00101	Arithmetic left one bit shifted MA0[31:16] register
MA1SR	00110	Arithmetic right one bit shifted MA1[31:16] register
MA1SL	00111	Arithmetic left one bit shifted MA1[31:16] register
RP0	01000	Current bank RP0[15:0] register
RP1	01001	Current bank RP1[15:0] register
RP2	01010	Current bank RP2[15:0] register
RP3	01011	Current bank 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 SD0[15:0]/SD0E register
SD1	01001	Current bank SD1[15:0] register
SD2	01010	Current bank SD2[15:0] register
SD3	01011	Current bank SD3[15:0]/SD3E register
SA	01100	SA[6:0] register
SI	01101	SI[15:0] register
SG	01110	SG[15:0] register
SR	01111	SR[15:0] register
P(H)	11000	P[31:16] register
PL	11001	P[15:0] register
MA0RN	11010	Rounded MA0[31:16] register
MA1RN	11011	Rounded MA1[31:16] register
MSR0	11100	MSR0[15:0] register
MSR1	11101	MSR1[15:0] register
MSR2	11110	MSR2[15:0] register
PRN	11111	Rounded P[31:16] register

**NOTE:** Grayed Field : read only register

- mci

Mnemonic	Encoding	Description
MC0	0	MC0[15:0] register
MC1	1	MC1[15:0] register

- srg

Mnemonic	Encoding	Description
SA	00	SA[6:0] 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[6:0] register
B	01	B[6:0] register
SA	10	SA[6:0] register
–	11	reserved

- bs

Mnemonic	Encoding	Description
BK0	0000	MSR2[12]
BK1	0001	MSR2[13]
BK2	0010	MSR2[14]
BK3	0011	MSR2[15]
ME0	0100	MSR0[12]
ME1	0101	MSR0[13]
ME2	0110	MSR0[14]
ME3	0111	MSR0[15]
OPM	1000	MSR1[3]
OPMA	1001	MSR1[7]
OP	1010	MSR0[9]
USM	1011	MSR1[4]
MV	1100	MSR1[2]
XSD	1101	MSR0[10]
PSH1	1110	MSR1[5]
NQ	1111	MSR1[6]

- ereg

Mnemonic	Encoding	Description
–	0000	Reserved
–	0001	Reserved
–	0010	Reserved
–	0011	Reserved
A	0100	A[15:0] register
B	0101	B[15:0] register
C	0110	C[15:0] register
D	0111	D[15:0] register
SA	1000	SA[6:0] register
SI	1001	SI[15:0] register
SG	1010	SG[15:0] register
SR	1011	SR[15:0] register
–	1100	Reserved
–	1101	Reserved
–	1110	Reserved
	1111	Reserved

- 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

- emod0

Mnemonic	Encoding	Description
ELD	00	Load
EADD	01	Add
ESUB	10	Subtract
ECP	11	Compare

- Pi

Mnemonic	Encoding	Description
P(H)	0	P[31:16] register
PL	1	P[15:0] register

- 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
V	0110	V = 1
NV	0111	V = 0
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

- 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
EFS8(T)	1010	Force to Sign bit 15 ~ bit 8 by bit 7
EFZ8(T)	1011	Force to Zero bit 15 ~ bit 8
-	1100	Reserved
-	1101	Reserved
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.

- 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 MAi,MAi'	1000	Load from MAi' to MAi
–	1001	Reserved
EADD MAi,P	1010	Add MAi and P
ESUB MAi,P	1011	Subtract P from MAi
EADD MAi,PSH	1100	Add MAi and 16-bit right shifted P
ESUB MAi,PSH	1101	Subtract 16-bit right shifted P from MAi
EDIVQ	1110	Division Step
–	1111	Reserved

- XiYi

Mnemonic	Encoding	Description
X0Y0	00	$X0[15:0] * Y0[15:0]$
X0Y1	01	$X0[15:0] * Y1[15:0]$
X1Y0	10	$X1[15:0] * Y0[15:0]$
X1Y1	11	$X1[15:0] * Y1[15:0]$

- Xi / Yi

Mnemonic	Encoding	Description
X0 / Y0	0	$X0[15:0] / Y0[15:0]$ register
X1 / Y1	1	$X1[15:0] / Y1[15:0]$ register



- rs

Mnemonic	Encoding	Description
ER	0	Bit reset instruction
ES	1	Bit set instruction

- Mi

Mnemonic	Encoding	Description
MA0	0	MA0[31:0] register
MA1	1	MA1[31:0] register

- rpi

Mnemonic	Encoding	Description
RP0	00	Current bank RP0[15:0] register
RP1	01	Current bank RP1[15:0] register
RP2	10	Current bank RP2[15:0] register
RP3	11	Current bank RP3[15:0] register

## Overall COP Instruction Set Map

Instruction	12	11	10	9	8	7	6	5	4	3	2	1	0
ECLD	0	0	0	0	imm:5					LS	Dn		
ELD mg,#imm:16	0	0	0	1	0	mg					imm:3		
EMOD0 An,#imm:16	0	0	0	1	1	0	mod0	An		imm:3			
ELD mgx,#imm:16	0	0	0	1	1	1	0	0	mgx	imm:3			
ERPn rpi, #imm:16	0	0	0	1	1	1	0	1	rpi	imm:3			
ELD An,adr:16	0	0	0	1	1	1	1	0	An	adr:3			
ELD adr:16,An	0	0	0	1	1	1	1	1	An	adr:3			
EMAD Mi, XiYi, mgx,@rps	0	0	1	0	XiYi		mgx	0	rps				
EMSB Mi, XiYi, mgx,@rps	0	0	1	0	XiYi		mgx	1	rps				
EMLD Mi, XiYi, mgx,@rps	0	0	1	1	XiYi		mgx	0	rps				
EMUL XiYi, mgx,@rps	0	0	1	1	XiYi		mgx	1	0	rps			
EADD Mi,P, mgx,@rps	0	0	1	1	0	Mi	mgx	1	1	rps			
ESUB Mi,P, mgx,@rps	0	0	1	1	1	Mi	mgx	1	1	rps			
EADD Mi,P, An,@rps	0	1	0	0	0	Mi	An	0	0	rps			
ESUB Mi,P, An,@rps	0	1	0	0	0	Mi	An	0	1	rps			
ELD Mi,P, An,@rps	0	1	0	0	0	Mi	An	1	0	rps			
ELD Mi,P, mgx,@rps	0	1	0	0	0	Mi	mgx	1	1	rps			
EADD Mi,P, @rpd,mga	0	1	0	0	1	Mi	mga	0	0	rpd			
ESUB Mi,P, @rpd,mga	0	1	0	0	1	Mi	mga	0	1	rpd			
ELD Mi,P, @rpd,mga	0	1	0	0	1	Mi	mga	1	0	rpd			
EADD Mi,P, @rpd,P	0	1	0	0	1	Mi	0	0	1	1	rpd		
ESUB Mi,P, @rpd,P	0	1	0	0	1	Mi	0	1	1	1	rpd		
ELD Mi,P, @rpd,P	0	1	0	0	1	Mi	1	0	1	1	rpd		
<b>Reserved</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>d</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>d</b>	<b>d</b>	<b>d</b>
EADD Ai,Mi, mgx,@rps	0	1	0	1	0	Mi	mgx	0	Ai	rps			
ESUB Ai,Mi, mgx,@rps	0	1	0	1	0	Mi	mgx	1	Ai	rps			
ELD Ai,Mi, mgx,@rps	0	1	0	1	1	Mi	mgx	0	Ai	rps			
EADD Ai,Mi, Mi,@rps	0	1	0	1	1	Mi	0	0	1	Ai	rps		
ESUB Ai,Mi, Mi,@rps	0	1	0	1	1	Mi	0	1	1	Ai	rps		
ELD Ai,Mi, Mi,@rps	0	1	0	1	1	Mi	1	0	1	Ai	rps		
ELD Pi,@rps	0	1	0	1	1	Pi	1	1	1	0	rps		
<b>Reserved</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>d</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>d</b>	<b>d</b>	<b>d</b>

NOTE: "d" means DONT Care.

Overall COP Instruction Set Map (Continued)

Instruction	12	11	10	9	8	7	6	5	4	3	2	1	0
EADD Ai,Mi, @rpd,mga	0	1	1	0	0	Mi	mga	0	Ai	rpd			
ESUB Ai,Mi, @rpd,mga	0	1	1	0	0	Mi	mga	1	Ai	rpd			
ELD Ai,Mi, @rpd,mga	0	1	1	0	1	Mi	mga	0	Ai	rpd			
EADD Ai,Mi, @rpd,P	0	1	1	0	1	Mi	0	0	1	Ai	rpd		
ESUB Ai,Mi, @rpd,P	0	1	1	0	1	Mi	0	1	1	Ai	rpd		
ELD Ai,Mi, @rpd,P	0	1	1	0	1	Mi	1	0	1	Ai	rpd		
ELD @rpd,P	0	1	1	0	1	Pi	1	1	1	0	rpd		
<b>reserved</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>d</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>d</b>	<b>d</b>	<b>d</b>
EADD Ai,Ci, Cj,@rps	0	1	1	1	0	0	0	Ci	Cj	Ai	rps		
ESUB Ai,Ci, Cj,@rps	0	1	1	1	0	0	1	Ci	Cj	Ai	rps		
ECP Ai,Ci, Cj,@rps	0	1	1	1	0	1	0	Ci	Cj	Ai	rps		
EMAX Ai,Ci, Ci,@rps	0	1	1	1	0	1	1	Ci	0	Ai	rps		
EMIN Ai,Ci, Ci,@rps	0	1	1	1	0	1	1	Ci	1	Ai	rps		
ELD mg1,@rps	0	1	1	1	1	mg1			0	0	rps		
ELD An,@rps	0	1	1	1	1	0	An	0	1	rps			
ELD srg,@rps	0	1	1	1	1	1	srg	0	1	rps			
ELD @rpd,mg1	0	1	1	1	1	mg1			1	0	rpd		
ELD @rpd,An	0	1	1	1	1	0	An	1	1	rpd			
ELD @rps,srg	0	1	1	1	1	1	srg	1	1	rpd			
EMAD Mi, XiYi, Xi,@rp01s, Yi,@rp3s	1	0	0	0	XiYi			Xi	Yi	0	Mi	rp3	rp01s
EMSB Mi, XiYi, Xi,@rp01s, Yi,@rp3s	1	0	0	0	XiYi			Xi	Yi	1	Mi	rp3	rp01s
EMLD Mi, XiYi, Xi,@rp01s, Yi,@rp3s	1	0	0	1	XiYi			Xi	Yi	0	Mi	rp3	rp01s
EMUL XiYi, Xi,@rp01s, Yi,@rp3s	1	0	0	1	XiYi			Xi	Yi	1	0	rp3	rp01s
ELD Xi,@rp01s, Yi,@rp3s	1	0	0	1	0	0	Xi	Yi	1	1	rp3	rp01s	
<b>reserved</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>d</b>	<b>d</b>	<b>1</b>	<b>1</b>	<b>d</b>	<b>d</b>	<b>d</b>
<b>reserved</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>d</b>	<b>d</b>	<b>d</b>	<b>1</b>	<b>1</b>	<b>d</b>	<b>d</b>	<b>d</b>
ESFT asr,asa	1	0	1	0	0	0	0	0	0	asr		asa	
ESFTA asr,asa	1	0	1	0	0	0	0	0	1	asr		asa	
ESFTL asr,asa	1	0	1	0	0	0	0	1	0	asr		asa	
ESFTD asr,asa	1	0	1	0	0	0	0	1	1	asr		asa	
ELD SA,#imm:5	1	0	1	0	0	0	1	0	imm:5				
ENMSK SG,#imm:4	1	0	1	0	0	0	1	1	0	imm:4			
ELD srgd,srgd	1	0	1	0	0	0	1	1	1	srgs		srgd	

## Overall COP Instruction Set Map (Continued)

Instruction	12	11	10	9	8	7	6	5	4	3	2	1	0
ELD rpui,rpd1.adr:2	1	0	1	0	0	1	0	adr:2		rpui			
ELD rpd1.adr:2,rpui	1	0	1	0	0	1	1	adr:2		rpui			
ESD0 ns,#imm:4	1	0	1	0	1	0	0	ns		imm:4			
ESD1 ns,#imm:4	1	0	1	0	1	0	1	ns		imm:4			
ESD2 ns,#imm:4	1	0	1	0	1	1	0	ns		imm:4			
ESD3 ns,#imm:4	1	0	1	0	1	1	1	ns		imm:4			
ELD An,rpdi.adr:5	1	0	1	1	0	rpd An		adr:5					
ELD rpdi.adr:5,An	1	0	1	1	1	rpd An		adr:5					
EMOD0 An,mg	1	1	0	0	mod0		An		mg				
EMOD0 An,Am	1	1	0	1	0	0	An 0		mod0		Am		
EMOD0 An,mgx	1	1	0	1	0	0	An 1		mod0		mgx		
ELD mg,An	1	1	0	1	0	1	An		mg				
EMAD Mi, XiYi, Ai,Mj	1	1	0	1	1	0	0	0	Mi	Ai	Mj	XiYi	
EMSB Mi, XiYi, Ai,Mj	1	1	0	1	1	0	0	1	Mi	Ai	Mj	XiYi	
EMLD Mi, XiYi, Ai,Mj	1	1	0	1	1	0	1	0	Mi	Ai	Mj	XiYi	
EMUL XiYi, Ai,Mj	1	1	0	1	1	0	1	1	0	Ai	Mj	XiYi	
EMAX Ai,Ci	1	1	0	1	1	0	1	1	1	Ai	Ci	0	0
EMIN Ai,Ci	1	1	0	1	1	0	1	1	1	Ai	Ci	0	1
EMAX Ai,Ai'	1	1	0	1	1	0	1	1	1	Ai	0	1	0
EMIN Ai,Ai'	1	1	0	1	1	0	1	1	1	Ai	0	1	1
NOP	1	1	0	1	1	0	1	1	1	d	1	1	d
ELD mg1d,mg1s	1	1	0	1	1	1	0	mg1s		mg1d			
ELD mg2d,mg2s	1	1	0	1	1	1	1	mg2s		mg2d			
EMAD Mi, XiYi, Ai,MjSL	1	1	1	0	0	0	0	0	Mi	Ai	Mj	XiYi	
EMSB Mi, XiYi, Ai,MjSL	1	1	1	0	0	0	0	1	Mi	Ai	Mj	XiYi	
EMLD Mi, XiYi, Ai,MjSL	1	1	1	0	0	0	1	0	Mi	Ai	Mj	XiYi	
EMUL XiYi, Ai,MjSL	1	1	1	0	0	0	1	1	0	Ai	Mj	XiYi	
EMAD Mi, XiYi	1	1	1	0	0	0	1	1	1	0	Mi	XiYi	
EMSB Mi, XiYi	1	1	1	0	0	0	1	1	1	1	Mi	XiYi	

Overall COP instruction set map (Continued)

Instruction	12	11	10	9	8	7	6	5	4	3	2	1	0
EMAD Mi, XiYi, Ai,MjSR	1	1	1	0	0	1	0	0	Mi	Ai	Mj	XiYi	
EMSB Mi, XiYi, Ai,MjSR	1	1	1	0	0	1	0	1	Mi	Ai	Mj	XiYi	
EMLD Mi, XiYi, Ai,MjSR	1	1	1	0	0	1	1	0	Mi	Ai	Mj	XiYi	
EMUL XiYi, Ai,MjSR	1	1	1	0	0	1	1	1	0	Ai	Mj	XiYi	
EMLD Mi, XiYi	1	1	1	0	0	1	1	1	1	0	Mi	XiYi	
EMUL XiYi	1	1	1	0	0	1	1	1	1	1	0	XiYi	
ERPR rpi	1	1	1	0	0	1	1	1	1	1	1	rpi	
ELD An,#imm:5	1	1	1	0	1	0	An			imm:5			
EADD An,#imm:5	1	1	1	0	1	1	An			imm:5			
ECP An,#imm:5	1	1	1	1	0	0	An			imm:5			
EMOD1 An	1	1	1	1	0	1	An		T	mod1			
ERPN rpi,An	1	1	1	1	1	0	0	0	0	An		rpi	
ERPS rps	1	1	1	1	1	0	0	0	1	0	rps		
ERPD rpd	1	1	1	1	1	0	0	0	1	1	rpd		
EMOD2 Mi	1	1	1	1	1	0	0	1	Mi	mod2			
ETST cc T/EC3	1	1	1	1	1	0	1	0	TE	cc			
ER/ES bs	1	1	1	1	1	0	1	1	ES	bs			
ELD Pi,mg1	1	1	1	1	1	1	0	0	0	Pi		mg1	
ELD mg1,Pi	1	1	1	1	1	1	0	0	1	Pi		mg1	
ELD mgx,An	1	1	1	1	1	1	0	1	0	An		mgx	
ELD sdis,sdis	1	1	1	1	1	1	0	1	1	sdis		sdis	
EBK #imm:4	1	1	1	1	1	1	1	0	0	imm:4			
ESEC0 #imm:4	1	1	1	1	1	1	1	0	1	imm:4			
ESEC1 #imm:4	1	1	1	1	1	1	1	1	0	imm:4			
ESEC2 #imm:4	1	1	1	1	1	1	1	1	1	imm:4			

## QUICK REFERENCE

opc	op1	op2	op3	op4	op5	op6	Function	Flag
EMAD	Mi	XiYi	mgx	@rps	-	-	Mi<-Mi+P, P<-Xi*Yi, op3<-op4	VMi
EMSB							Mi<-Mi-P, P<-Xi*Yi, op3<-op4	VMi
EMLD							Mi<- P, P<-Xi*Yi, op3<-op4	VMi
EMUL							-	P<-Xi*Yi, op3<-op4
EMAD	Mi	XiYi	Ai	Mj MjSR MjSL	-	-	Mi<-Mi+P, P<-Xi*Yi, op3<-op4	VMi, V,N,Z
EMSB							Mi<-Mi-P, P<-Xi*Yi, op3<-op4	VMi, V,N,Z
EMLD							Mi<- P, P<-Xi*Yi, op3<-op4	VMi, V,N,Z
EMUL	-	XiYi	Xi	@rp01s	Yi	@rp3s	P<-Xi*Yi, op3<-op4	VMi, V,N,Z
EMAD	Mi						Mi<-Mi+P, P<-Xi*Yi, op3<-op4, op5<-op6	VMi
EMSB	Mi						Mi<-Mi-P, P<-Xi*Yi, op3<-op4, op5<-op6	VMi
EMLD	Mi						Mi<- P, P<-Xi*Yi, op3<-op4, op5<-op6	VMi
EMUL	-						P<-Xi*Yi, op3<-op4, op5<-op6	VMi
EMAD	Mi	XiYi	-	-	-	-	Mi<-Mi+P, P<-Xi*Yi	VMi
EMSB							Mi<-Mi-P, P<-Xi*Yi	VMi
EMLD							Mi<- P, P<-Xi*Yi	VMi
EMUL							-	P<-Xi*Yi
EADD	Mi	P	mgx An	@rps	-	-	Mi<-Mi+P, op3<-op4	VMi
ESUB							Mi<-Mi-P, op3<-op4	VMi
ELD							Mi←P, op3<-op4	VMi
EADD	Mi	P	@rpd	mga P	-	-	Mi<-Mi+P, op3<-op4	VMi
ESUB							Mi<-Mi-P, op3<-op4	VMi
ELD							Mi←P, op3<-op4	VMi
EADD	Ai	Mi	mgx	@rps	-	-	Ai<-Ai+Mi, op3<-op4	V,N,Z,C
ESUB							Ai<-Ai-Mi, op3<-op4	V,N,Z,C
ELD							Ai<- Mi, op3<-op4	V,N,Z

## QUICK REFERENCE (Continued)

opc	op1	op2	op3	op4	op5	op6	Function	Flag
EADD	Ai	Mi	Mi	@rps	-	-	Ai<-Ai+Mi, op3<-op4	V,N,Z,C, VMi
ESUB							Ai<-Ai-Mi, op3<-op4	V,N,Z,C, VMi
ELD							Ai<- Mi, op3<-op4	V,N,Z,V Mi
EADD	Ai	Mi	@rpd	mga P	-	-	Ai<-Ai+Mi, op3<-op4	V,N,Z,C
ESUB							Ai<-Ai-Mi, op3<-op4	V,N,Z,C
ELD							Ai<- Mi, op3<-op4	V,N,Z
ELD	An	mg Am mgx imm:16	-	-	-	-	An<-op2	V,N,Z
EADD							An<-An+op2	V,N,Z,C
ESUB							An<-An-op2	V,N,Z,C
ECP							An-op2	V,N,Z,C
ELD	An	imm:5	-	-	-	-	An<-op2	V,N,Z
EADD							An<-An+op2	V,N,Z,C
ECP							An-op2	V,N,Z,C

\* opc – opcode, opi- operand i

VMi – VM0 or VM1 according to Mi (w hen VMi is written, MV is written)

## Quick Reference (Continued)

opc	op1	op2	op3	op4	Function	Flag
ELD	Xi	@rp01s	Yi	@rp3s	op1<-op2, op3<-op4	-
ELD	An	adr:16 rpd1.adr:5 @rps	-	-	An<-op2	V,N,Z
ELD	adr:16 rpd1.adr:5 @rpd	An	-	-	op1<-An	-
ELD	mgx	imm:16 An	-	-	op1<-op2	-
ELD	mg1	mg1 Pi @rps	-	-	op1<-op2	-(VMi)*
ELD	mg	imm:16 An	-	-	op1<-op2	-(VMi)*
ELD	Pi	mg1 @rps	-	-	op1<-op2	-
ELD	srg	srg @rps	-	-	op1<-op2	-
ELD	@rpd	Pi mg1 srg	-	-	op1<-op2	-
ELD	rpui	rpd1.adr:2	-	-	op1<-op2	-
ELD	rpd1.adr:2	rpui	-	-	op1<-op2	-
ELD	Mi	Mi'	-	-	Mi<-Mi'	VMi
EADD	Mi	P PSH	-	-	Mi<-Mi+op2	VMi
ESUB			-	-	Mi<-Mi-op2	VMi
EADD	Ai	Ci	Cj	@rps	Ai<-Ai+Ci, op3<-op4	V,N,Z,C
ESUB		-	-	-	Ai<-Ai-Ci, op3<-op4	V,N,Z,C
ECP		-	-	-	Ai-Ci, op3<-op4	V,N,Z,C
EMAX	Ai	Ci	Ci	@rps	Ai<-max(Ai,Ci), op3<-op4, RP3<-address	V,N,Z,C
EMIN			-	-	Ai<-min(Ai,Ci), op3<-op4, RP3<-address	V,N,Z,C
ERPNI	rpi	imm:16 An	-	-	rpi<-mod(rpi+op2)	-
ECLD	Dn	ereg	-	-	Dn<-ereg	-



## Quick Reference (Continued)

opc	op1	op2	op3	op4	Function	Flag
ECLD	ereg	Dn	-	-	ereg<-Dn	-
ELD	SA	imm:5	-	-	SA<-op2	-
EMAX	Ai	Ci Ai'	-	-	Ai<-max(Ai,op2), RP3<-address	V,N,Z,C
EMIN			-	-	Ai<-min(Ai,op2), RP3<-address	V,N,Z,C
ELD	mg2	mg2	-	-	op1<-op2	-
ELD	sdi	sdi	-	-	op1<-op2	-
ENOP	-	-	-	-	No Operation	-

**NOTE:** VMi is affected when op1 is MAi(H)

## Quick Reference (Continued)

opc	op1	op2	Function	Flag
ESFT	asr	asa	{SG,SR}<-op1</>>op2 (logical)	VS,N,Z,C
ESFTA			{SG,SR}<-op1</>>op2 (arithmetic)	VS,N,Z,C
ESFTL			SG<-SG (op1</>>op2)	VS,N,Z,C
ESFTD			SR<-op1</>>op2, SG<-SG (op1</>>op2)	VS,N,Z,C
ENMSK	SG	imm:4	SG<-SG&(mask_pattern by imm)	VS,N,Z
ESD0	ns	imm:4	SD0.ns<-op2	-
ESD1			SD1.ns<-op2	-
ESD2			SD2.ns<-op2	-
ESD3			SD3.ns<-op2	-
ERPS	rps	-	op1<-mod(op1+Si)	-
ERPD	rpd	-	op1<-mod(op1+Di)	-
ERPR	rpi	-	RP3<-bit_reverse(op1)	-
ESEC0	imm:4	-	MSR2.SEC0<-op1	-
ESEC1			MSR2.SEC1<-op1	-
ESEC2			MSR2.SEC2<-op1	-
EBK	imm:4	-	MSR2[15:12]<-op1	-
ER	bs	-	op1<-0	-
ES			op1<-1	-
ETST	cct	T EC3	op2<-1 if (cct )	-
EDIVQ	Mi	-	Division Step	VMi, NQ
ERESR			Mi<-Mi+2P if (NQ=1)	VMi

## Quick Reference (Continued)

opc	op1	op2	Function	Flag
ESLA			Mi<-Mi<<1	VMi
ESRA			Mi<-Mi>>1	VMi
ECR			Mi<-0	VMi
ESAT			Mi<-saturated(Mi)	VMi
ERND			Mi<-Mi+8000h	VMi
ESLA(T*)	An	-	op1<-op1<<1 (arithmetic)	V,N,Z,C
ESRA(T*)			op1<-op1>>1 (arithmetic)	V,N,Z,C
ESLA8(T*)			op1<-op1<<8 (arithmetic)	V,N,Z,C
ESRA8(T*)			op1<-op1>>8 (arithmetic)	V,N,Z,C
ESLC(T*)			op1<-{op1[14:0],C}	V,N,Z,C
ESRC(T*)			op1<-{C,op1[15:1]}	V,N,Z,C
EINCC(T*)			op1<-op1+C	V,N,Z,C
EDECC(T*)			op1<-op1-C'	V,N,Z,C
EABS(T*)			op1<- op1}	V,N,Z,C
ENEG(T*)			op1<-op1'+1	V,N,Z,C
EFS8(T*)			op1[15:8]<-op1[7]	V,N,Z,C
EFZ8(T*)			op1[15:8]<-0	V,N,Z,C
EEXP(T*)			SA<-exp(op1)	VS,N,Z,C
EEXPC(T*)			SA<-SA+exp(C,op1)	VS,N,Z,C

\* if T=1, instruction is executed

## INSTRUCTION SET

### GLOSSARY

This chapter describes the CalmMAC16 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.

## EABS/EABST (Note) – Absolute

**Format:** EABS(T) An

**Operation:** An <- |An|

This instruction calculates the absolute value of one of 16-bit Accumulator (An), 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.

**V:** Set if overflow is generated. Reset if not.

**N:** Exclusive OR of V and MSB of result. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** EABST instruction can be executed only when the T flag is set. Otherwise, No operation is performed.

**Examples:** EABS A  
EABST C  
# of Words: 1

## EADD <sup>(1)</sup> – Add Accumulator

**Format:** EADD An, <op>  
<op>: #simm:5 / #simm:16  
Am  
mg / mgx

**Operation:** An ← An + <op>  
This instruction adds the values of one of 16-bit Accumulators (An) and <op> together, and stores the result back into the same Accumulator. If <op> is immediate value, it is first right adjusted and sign-extended to 16-bit value. If <op> is 16-bit register, it is zero-extended.

**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. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** None

**Example s:** EADD A, #0486h  
EADD C, A  
EADD A, RPO

**# of Words:** 1  
2 when <op> is : #simm:16

## EADD (2) – Add Accumulator w/ One Parallel Move

**Format:**

1. EADD Ai, Mi, <dest>, <src>  
     <dest>, <src>: mgx, @rps  
     Mi, @rps  
     @rpd, mga  
     @rpd, P

2. EADD Ai, Ci, Cj, @rps

**Operation:**

1.  $A_i \leftarrow A_i + M_i$ , <dest>  $\leftarrow$  <src>
2.  $A_i \leftarrow A_i + C_i$ , Cj  $\leftarrow$  @rps

This instruction adds the values of 16-bit Accumulator Ai (A or B register) and higher 16-bit part of Multiplier Accumulator MAi (MA0 or MA1 register) or the value of 16-bit Accumulator Ci (C or D register) together, and stores the result back into Accumulator Ai. This instruction also stores a source operand from memory or register to the destination register or memory location.

**Flags:**

- C:** Set if carry is generated by addition. Reset if not.
- Z:** Set if result is zero by addition. Reset if not.
- V:** Set if overflow is generated by addition. Reset if not.
- N:** Exclusive OR of V and MSB of result by addition. Refer to Chapter 2 for more detailed explanation about this convention.
- VMi<sup>\*</sup>:** Set if result is overflowed to guard-bits. Reset if not.

**Notes:** \* VMi denotes for VM0 or VM1 according to Mi if <dest> is Mi.

**Examples:**

```
EADD A, MA0, X0, @RP0+S1
EADD B, MA1, MA1, @RP1+S0
EADD A, MA1, @RP3+D1, MA0
EADD B, D, C, @RP2+S1
```

**# of Words:** 1

## EADD <sup>(3)</sup> – Add Multiplier Accumulator

**Format:** EADD Mi, <op>  
<op>: P / PSH

**Operation:** MAi ← MAi + <op>  
This instruction adds the values of 36-bit Multiplier Accumulator MAi (MA0 or MA1 register) and <op> together, and stores the result back into Multiplier Accumulator MAi. The “PSH” means 16-bit arithmetic right shifted P register value.

**Flags:** **VMi**<sup>\*</sup>: 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 Mi.

**Examples:** EADD MA0, P  
EADD MA1, PSH

**# of Words:** 1

## EADD<sup>(4)</sup> – Add Multiplier Accumulator w/ One Parallel Move

**Format:** EADD Mi, P, <dest>, <src>  
<dest>, <src>: mgx, @rps  
An, @rps  
@rpd, mga  
@rpd, P

**Operation:** MAi <- MAi + P, <dest> <- <src>  
This instruction adds the values of 36-bit Multiplier Accumulator MAi (MA0 or MA1 register) and Product Register P together, and stores the result back into Multiplier Accumulator MAi. 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 destination Mi.

**Examples:** EADD MA0, P, Y0, @RP1+S1  
EADD MA1, P, C, @RP2+S0  
EADD MA1, P, @RP0+D0, B

**# of Words:** 1



## **EBK – Pointer/Index Register Bank Select**

**Format:** EBK #imm:4

**Operation:** MSR2[15:12] <- imm:4

This instruction loads the 4-bit immediate value to the specified bit field of MSR2 register (bit 15 ~ bit 12). Only 4-bit field of 16-bit register value is changed

**Flags:** –

**Notes:** –

**Examples:** EBK #1010b

**# of Words:** 1

## ECLD – Coprocessor Accumulator Load from host processor

**Format:** ECLD ereg, Dn  
ECLD Dn, ereg

**Operation:** ereg <- Dn or Dn <- ereg  
This instruction moves the selected 16-bit general purpose register value of host processor to the An (A, B, C, or D) accumulator register or shifter register (SA, SR, SG, SI), or vice versa. This instruction is mapped to “CLD” instruction of CalmRISC microcontroller.

**Flags:** –

**Notes:** This instruction has delay slot. Because this instruction is 2-cycle instruction.

**Examples:** ECLD A, R0  
ECLD R3, BH  
ECLD SI, R3

**# of Words:** 1

## ECP<sup>(1)</sup> – Compare Accumulator

**Format:** ECP An, <op>  
<op>: #simm:5 / #simm:16  
Am  
mg / mgx

**Operation:** An - <op>  
This Instruction compares the values of Accumulator An and <op> by subtracting <op> from Accumulator. Content of Accumulator is not changed. If <op> is immediate value, it is first right adjusted and sign-extended to 16-bit value. If <op> is 16-bit register, it is zero-extended.

**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. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** –

**Examples:** ECP A, #0486h  
ECP C, A  
ECP D, RPO

**# of Words:** 1  
2 when <op> is : #simm:16

## ECP<sup>(2)</sup> – Compare Accumulator w/ One Parallel Move

**Format:** ECP Ai, Ci, Cj,@rps

**Operation:** Ai - Ci, Cj <- @rps

This Instruction compares the values of Accumulator Ai (A or B register) and Ci (C or D register) by subtracting Ci from Ai. Content of Accumulator Ai is not changed. This instruction also stores a source operand from memory or register to the destination register or memory location.

**Flags: C:** Set if carry is generated by addition. Reset if not.

**Z:** Set if result is zero by addition. Reset if not.

**V:** Set if overflow is generated by addition. Reset if not.

**N:** Exclusive OR of V and MSB of result by addition. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** None.

**Examples:** ECP B, D, C,@RP2+S1

**# of Words:** 1

## ECR – Clear MA Accumulator

**Format:** ECR Mi

**Operation:** MAi ← 0

This Instruction clears the value of 36-bit MAi (MA0 or MA1) accumulator. The extension nibble of selected MA accumulator is also cleared.

**Flags:** VMi<sup>\*</sup>: 0.

**Notes:** \* VMi denotes for VM0 or VM1 according to Mi.

**Examples:** ECR MA1

**# of Words:** 1

## EDECC/EDECCT<sup>\*</sup> – Decrement with Carry

**Format:** EDECC(T) An

**Operation:**  $A_n \leftarrow A_n - \sim C$

This instruction subtracts 1 from the value of one of 16-bit Accumulator (An) 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.

**V:** Set if overflow is generated. Reset if not.

**N:** Exclusive OR of V and MSB of result. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** \* EDECCT instruction can be executed only when the T flag is set.  
Otherwise, No operation is performed.

**Examples:** EDECC A  
EDECCT D

**# of Words:** 1

## EDIVQ – Division Step

**Format:** EDIVQ Mi,P

**Operation:** if (NQ = 0)  
    Adder output  $\leftarrow$  MAi – P  
else  
    Adder output  $\leftarrow$  MAi + P  
(Adder output > 0)  
    MAi  $\leftarrow$  Adder output \* 2 + 1  
else  
    MAi  $\leftarrow$  Adder output \* 2

This Instruction adds or subtracts one of the MAi accumulator from P register according to the NQ bit value and calculates one bit quotient and new partial remainder.

**Flags:** **NQ:** If (Adder output > 0) NQ  $\leftarrow$  0, else NQ  $\leftarrow$  1  
**VMi<sup>\*</sup>:** 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 Mi.

**Examples:** EDIVQ MA0,P

**# of Words:** 1

## **EEXP/EEXPT** \* – Exponent Value Evaluation

**Format:** EEXP(T) An

**Operation:** SA <- exponent(An)

This instruction evaluates the exponent value of one of 16-bit Accumulator (An), and stores the result back into 7-bit SA register.

**Flags: C:** Set if LSB of source An accumulator is 1. Reset if not.

**Z:** Set if exponent evaluation result is zero. Reset if not.

**VS:** Set if the value of source An 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 C

**# of Words:** 1



## EEXPC/EEXPCT<sup>\*</sup> – Exponent Value Evaluation with Carry

**Format:** EEXPC(T) An

**Operation:** if (VS = 1)  
SA ← exponent({C,An})  
else  
no operation

This instruction evaluates the exponent value which concatenates carry and one of 16-bit Accumulator (An), adds the result with SA register value, and stores the added result back into 7-bit SA register. It can be used for multi-precision exponent evaluation.

**Flags:** **C:** Set if LSB of source An 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 An 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 D  
EEXPCT B

**# of Words:** 1

## EFS/EFST<sup>\*</sup> – Force to Sign MSB8 bits

**Format:** EFS(T) An

**Operation:**  $A_n \leftarrow \{8\{A_n[7]\}, A_n[7:0]\}$

This instruction forces the value of MSB 8 bits of 16-bit Accumulator (An) with byte sign bit of An register (An[7]), and stores the result back into the same Accumulator.

**Flags: C:** Reset.

**Z:** Set if result is zero. Reset if not.

**V:** Reset.

**N:** MSB of result.

**Notes:** \* EFS8T instruction can be executed only when the T flag is set.

Otherwise, No operation is performed.

**Examples:** EFS8 A

EFS8T D

**# of Words:** 1

**EFZ/EFZT** \* – Force to Zero MSB 8bits**Format:** EFZ(T) An**Operation:**  $A_n \leftarrow \{8\{0\}, A_n[7:0]\}$ 

This instruction forces the value of MSB 16 bits of 16-bit Accumulator (An) with zero, and stores the result back into the same Accumulator.

**Flags: C:** Reset.**Z:** Set if result is zero. Reset if not.**V:** Reset.**N:** Reset.**Notes:** \* EFZ8T instruction can be executed only when the T flag is set.

Otherwise, No operation is performed.

**Examples:** EFZ8 C

EFZ8T B

**# of Words:** 1

## EINCC/EINCCT<sup>\*</sup> – Increment with Carry

**Format:** EINCC(T) An

**Operation:**  $An \leftarrow An + C$

This instruction adds 1 from the value of one of 16-bit Accumulator (An) 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.

**V:** Set if overflow is generated. Reset if not.

**N:** Exclusive OR of V and MSB of result. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** \* EINCCT instruction can be executed only when the T flag is set.

Otherwise, No operation is performed.

**Examples:** EINCC A

EINCCT C

**# of Words:** 1

## ELD <sup>(1)</sup> – Load Accumulator

**Format:**

1. ELD An, <mem>  
<mem> : @rps  
          rpd1.adr:5 / adr:16
2. ELD An, <op>  
<op> : Am  
          #simm:5 / #simm:16  
          mgx / mg

**Operation:** An <- <mem> or <op>

This instruction load <mem> or <op> value to the one of 16-bit Accumulator (An). If <op> is immediate value, it is first right adjusted and sign-extended to 16-bit value. If <op> is 16-bit register, it is zero-extended.

**Flags:**

- Z<sup>\*</sup>:** Set if result is zero. Reset if not.
- V<sup>\*</sup>:** Set if overflow is generated. Reset if not.
- N<sup>\*</sup>:** Set if loaded value is negative.

**Notes:** \* Flags are not affected when a source operand is from memory.

**Examples:**

- ELD A, @RP0+S0
- ELD B, RPD1.5h
- ELD C, #0486h
- ELD D, A
- ELD A, RP0

**# of Words:** 1  
2 when <op> is : adr:16 or #simm:16

## ELD<sup>(2)</sup> – Load Accumulator w/ One Parallel Move

**Format:** ELD Ai, Mi, <dest>,<src>  
 <dest>,<src>: mgx, @rps  
 Mi, @rps  
 @rpd, mga  
 @rpd, P

**Operation:**  $A_i \leftarrow MA_i$ ,  $\langle \text{dest} \rangle \leftarrow \langle \text{src} \rangle$   
 This instruction load higher 16-bit part of Multiplier Accumulator MA<sub>i</sub> to the 16-bit Accumulator A<sub>i</sub>. 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.  
**V:** Set if overflow is generated by load. Reset if not.  
**N:** Set if loaded value is negative.  
**VMi<sup>\*</sup>:** Set if result is overflowed to guard-bits. Reset if not.

**Notes:** \* VM<sub>i</sub> denotes for VM<sub>0</sub> or VM<sub>1</sub> according to M<sub>i</sub> if <dest> is M<sub>i</sub>.

**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 <- MAi'  
This instruction loads the value of the one 36-bit Multiplier Accumulator MAi 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 w/ One Parallel Move

**Format:** ELD Mi, P, <dest>, <src>  
<dest>, <src>: mgx, @rps  
An, @rps  
@rpd, mga  
@rpd, P

**Operation:** MAi <- P, <dest> <- <src>  
This instruction load sign-extended 32-bit Product register P to the 36-bit Multiplier Accumulator MAi. 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 MA0, P, X0, @RP0+S1  
ELD MA1, P, A, @RP1+S0  
ELD MA1, P, @RP3+D1, A

**# of Words:** 1



## ELD <sup>(5)</sup> – Load Other Registers or Memory

**Format:** ELD <dest>, <src>  
 <dest>,<src>: mg1, @rps  
 srg, @rps  
 Pi, @rps  
 @rpd, An  
 @rpd, mg1  
 @rpd, srg  
 @rpd, Pi  
 rpui, rpd1.adr:2  
 rpd1.adr:2, rpui  
 rpdi.adr:5, An  
 adr:16, An  
 mgx, #simm:16  
 mg, #simm:16  
 SA, #simm:5  
 mg1d, mg1s  
 mg2d, mg2s  
 sdid, sdis  
 srgd, srgs  
 mg, An  
 mgx, An  
 Pi, mg1  
 mg1, Pi

**Operation:** <dest> <- <src>  
 This instruction load <src> value to <dest>. If the width of immediate is less than the width of <dest>, the immediate field is sign-extended and if the width of <src> is more than the width of <dest>, LSB part of <src> is written to <dest>.

**Flags:** **VMi**<sup>\*</sup>: Set if result is overflowed to guard-bits. Reset if not.

**Notes:** \* VMi denotes for VM0 or VM1 according to destination Mi if <dest> is Mi.

**ELD <sup>(5)</sup> – Load Other Registers or Memory (Continued)**

**Examples:**     ELD @RP0+D0, B  
                  ELD RPD1.5h, RP2  
                  ELD MC0, #0486h  
                  ELD RPD1, MC0  
                  ELD X0, Y1

**# of Words:**    1  
                  2 when <dest> or <src> is : adr:16 or #imm:16

## ELD <sup>(6)</sup> – Dual Load

**Format:** ELD Xi,@rp01s, Yi,@rp3s

**Operation:** Xi <- operand1 by @rp01s, 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 16-bit Xi and Yi register, respectively.

**Flags:** –

**Notes:** –

**Examples:** ELD X0,@RP1+S1, Y1,@RP3+S0

**# of Words:** 1

## EMAD<sup>(1)</sup> – Multiply and Add

**Format:** EMAD Mi, XiYi

**Operation:**  $MA_i \leftarrow MA_i + P$ ,  $P \leftarrow X_i * Y_i$

This instruction adds the values of 36-bit Multiplier Accumulator MA<sub>i</sub> and P register together, and stores the result back into Multiplier Accumulator MA<sub>i</sub>. At the same time, multiplier multiplies X<sub>i</sub> register value and Y<sub>i</sub> register value, and stores the result to the P register.

**Flags: VMi<sup>\*</sup>:** 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 Mi.

**Examples:** EMAD MA0, X1Y0

**# of Words:** 1

## EMAD <sup>(2)</sup> – Multiply and Add w/ One Parallel Move

**Format:** EMAD Mi, XiYi, <dest>,<src>  
 <dest>,<src> : Ai,Mj  
 Ai,MjSR\*  
 Ai,MjSL\*\*  
 mgx,@rps

**Operation:** MAi ← MAi + P, P ← Xi \* Yi, <dest> ← <src>  
 This instruction adds the values of 36-bit Multiplier Accumulator MAi and P register together, and stores the result back into Multiplier Accumulator MAi. 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 16-bit higher portion of the MAj register to the destination register.

**Flags:** **VMi<sup>\*\*\*</sup>**: 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.  
**V**: Set if overflow is generated by load. Reset if not.  
**N**: Set if loaded value is negative.

**Notes:** \* MjSR : 1-bit right shifted MAj[31:16]  
 \*\* MjSL : 1-bit left shifted MAj[31:16]  
 \*\*\* VMi denotes for VM0 or VM1 according to the Mi.

**Examples:** EMAD MA0, X1Y0, A,MA1SR  
 EMAD MA1, X0Y0, X0,@RP1+S1

**# of Words:** 1

## EMAD <sup>(3)</sup> – Multiply and Add w/ Two Parallel Moves

**Format:** EMAD Mi, XiYi, Xi,@rp01s, Yi,@rp3s

**Operation:** MAi <- MAi + P, P <- Xi \* Yi, Xi <- operand1 by @rp01s, Yi <- operand2 by @rp3s

This instruction adds the values of 36-bit Multiplier Accumulator MAi and P register together, and stores the result back into Multiplier Accumulator MAi. 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 16-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 MA0, X1Y0, X0,@RP0+S1, Y0,@RP3+S0

**# of Words:** 1

## EMAX <sup>(1)</sup> – Maximum Value Load

**Format:** EMAX Ai, <op>  
<op>: Ci  
Ai'

**Operation:** if (<op> >= Ai)  
Ai <- <op>, RP3 <- previous address with RPi register

This instruction conditionally loads <op> value to the one of 16-bit Accumulator(Ai) and latches the previous address value to the RP3 pointer when <op> is greater than or equal to Ai. Otherwise, no operation is performed.

**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. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** \* Flags are generated from the operation (Ai - <op>)

**Examples:** EMAX A, C

**# of Words:** 1

## EMAX <sup>(2)</sup> – Maximum Value Load w/ One Parallel Move

**Format:** EMAX Ai, Ci, Ci,@rps

**Operation:** if (Ci >= Ai)

Ai <- Ci, Ci <- @rps, RP3 <- previous address with RPi register

This instruction conditionally loads Ci value to the one of 16-bit Accumulator (Ai) and latches the previous address value to the RP3 pointer when <op> is greater than or equal to Ai. Otherwise, no operation is performed. This instruction also stores source operand from data memory to the destination accumulator (the same accumulator register Ci). RP3 register can not be used as a pointer register of parallel move part.

**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. Refer to Programming MODEL Part for more detailed explanation about this convention.

**Notes:** \* Flags are generated from the operation (Ai – Ci)

**Examples:** EMAX A, D, D, @RP1+S1

**# of Words:** 1



## EMIN <sup>(1)</sup> – Minimum Value Load

**Format:** EMIN Ai, <op>  
<op>: Ci  
Ai'

**Operation:** if (<op> <= Ai)  
Ai <- <op>, RP3 <- previous address with RPi register  
This instruction conditionally loads <op> value to the one of 16-bit Accumulator(Ai) and latches the previous address value to the RP3 pointer when <op> is less than or equal to Ai. Otherwise, no operation is performed.

**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. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** \* Flags are generated from the operation (<op> - Ai)

**Examples:** EMIN B, D

**# of Words:** 1

## EMIN<sup>(2)</sup> – Minimum Value Load w/ One Parallel Load

**Format:** EMIN Ai, Ci, Ci,@rps

**Operation:** if (Ci <= Ai)

Ai <- Ci, Ci <- @rps, RP3 <- previous address with RPi register

This instruction conditionally loads <op> value to the one of 16-bit Accumulator (Ai) and latches the previous address value to the RP3 pointer when <op> is less than or equal to Ai. Otherwise, no operation is performed. This instruction also stores source operand from data memory to the destination accumulator (the same accumulator register Ci). RP3 register can not be used as a pointer register of parallel move part.

**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. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** \* Flags are generated from the operation (Ci - Ai)

**Examples:** EMIN B, D, D, @RP0+S1

**# of Words:** 1

## EMLD <sup>(1)</sup> – Multiply and Load

**Format:** EMLD Mi, XiYi

**Operation:** MAi ← P, P ← Xi \* Yi

This instruction loads the P register value to the values of 36-bit Multiplier Accumulator MAi. At the same time, multiplier multiplies Xi register value and Yi register value, and stores the result to the P register.

**Flags:** **VMi<sup>\*</sup>**: 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 Mi.

**Examples:** EMLD MA0, X1Y0

**# of Words:** 1

## EMLD <sup>(2)</sup> – Multiply and Load w/ One Parallel Move

**Format:** EMLD Mi, XiYi, <dest>,<src>  
 <dest>,<src> : Ai,Mj  
           Ai,MjSR<sup>\*</sup>  
           Ai,MjSL<sup>\*\*</sup>  
           mgx,@rps

**Operation:** MAi <- P, P <- Xi \* Yi, <dest> <- <src>  
 This instruction loads the P register value to the one of 36-bit Multiplier Accumulator MAi. 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 16-bit higher portion of the MAj register to the destination register.

**Flags:** **VMi<sup>\*\*\*</sup>:** 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.  
**V:** Set if overflow is generated by load. Reset if not.  
**N:** Set if loaded value is negative.

**Notes:** \* MjSR: 1-bit right shifted MAj[31:16]  
 \*\* MjSL: 1-bit left shifted MAj[31:16]  
 \*\*\* VMi denotes for VM0 or VM1 according to Mi.

**Examples:** EMLD MA0, X1Y0, A,MA1SR  
 EMLD MA1, X0Y0, X0,@RP1+S1

**# of Words:** 1

## **EMLD** <sup>(3)</sup> – Multiply and Load w/ Two Parallel Moves

**Format:** EMLD Mi, XiYi, Xi,@rp01s, Yi,@rp3s

**Operation:** MAi <- P, P <- Xi \* Yi, Xi <- operand1 by @rp01s, Yi <- operand2 by @rp3s

This instruction loads the P register value to one of 36-bit Multiplier Accumulator MAi. 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 16-bit Xi register and Yi register respectively.

**Flags: VMi<sup>\*</sup>:** 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 Mi.

**Examples:** EMLD MA1, X1Y0, X0,@RP1+S1, Y0,@RP3+S0

**# of Words:** 1

## EMSB <sup>(1)</sup> – Multiply and Subtract

**Format:** EMSB Mi, XiYi

**Operation:**  $MA_i \leftarrow MA_i - P$ ,  $P \leftarrow X_i * Y_i$

This instruction subtracts the P register from the values of 36-bit Multiplier Accumulator MA<sub>i</sub>, and stores the result back into the same Multiplier Accumulator MA<sub>i</sub>. At the same time, multiplier multiplies Xi register value and Yi register value, and stores the result to the P register.

**Flags: VMi<sup>\*</sup>:** 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 Mi.

**Examples:** EMSB MA0, X1Y0

**# of Words:** 1

## EMSB<sup>(2)</sup> – Multiply and Subtract w/ One Parallel Move

**Format:** EMSB Mi, XiYi, <dest>,<src>  
 <dest>,<src> : Ai,Mj  
                   Ai,MjSR<sup>\*</sup>  
                   Ai,MjSL<sup>\*\*</sup>  
                   mgx,@rps

**Operation:** MAi <- MAi - P, P <- Xi \* Yi, <dest> <- <src>  
 This instruction subtracts the P register from the values of 36-bit Multiplier Accumulator MAi, and stores the result back into the same Multiplier Accumulator MAi. 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 16-bit higher portion of the MAj register to the destination register.

**Flags:** **VMi<sup>\*\*\*</sup>**: 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.  
**V**: Set if overflow is generated by load. Reset if not.  
**N**: Set if loaded value is negative.

**Notes:** \* MjSR : 1-bit right shifted MAj[31:16]  
 \*\* MjSL : 1-bit left shifted MAj[31:16]  
 \*\*\* VMi denotes for VM0 or VM1 according to Mi.

**Examples:** EMSB MA0, X1Y0, A,MA0SR  
 EMSB MA1, X0Y0, X0,@RP1+S1

**# of Words:** 1

## EMSB<sup>(3)</sup> – Multiply and Subtract w/ Two Parallel Moves

**Format:** EMSB Mi, XiYi, Xi,@rp01s, Yi,@rp3s

**Operation:** MAi <- MAi - P, P <- Xi \* Yi, Xi <- operand1 by @rp01s, Yi <- operand2 by @rp3s

This instruction subtracts the P register from the values of 36-bit Multiplier Accumulator MAi, and stores the result back into the same Multiplier Accumulator MAi. 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 16-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 Mi.

**Examples:** EMSB MA1, X1Y0, X0,@RP0+S1, Y0,@RP3+S0

**# of Words:** 1



## EMUL<sup>(1)</sup> – Multiply

**Format:** EMLU XiYi

**Operation:**  $P \leftarrow X_i * Y_i$

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 w/ One Parallel Move

**Format:** EMUL XiYi, <dest>,<src>  
<dest>,<src> : Ai,Mj  
Ai,MjSR'  
Ai,MjSL''  
mgx,@rps

**Operation:**  $P \leftarrow X_i * Y_i$ ,  $\langle \text{dest} \rangle \leftarrow \langle \text{src} \rangle$   
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 16-bit higher portion of the MAj register to the destination register.

**Flags:** When <dest> is Ai  
**Z:** Set if the value to Ai is zero by load. Reset if not.  
**V:** Set if overflow is generated by load. Reset if not.  
**N:** Set if loaded value is negative.

**Notes:** \* MjSR : 1-bit right shifted MAj[31:16]  
\*\* MjSL : 1-bit left shifted MAj[31:16]

**Examples:** EMUL X1Y0, A,MA1SR  
EMUL X0Y0, X0,@RP1+S1

**# of Words:** 1

## EMUL <sup>(3)</sup> – Multiply w/ Two Parallel Moves

**Format:** EMUL XiYi, Xi,@rp01s, Yi,@rp3s

**Operation:** P <- Xi \* Yi, Xi <- operand1 by @rp01s, 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 operands from data memory (one from X memory space and one from Y memory space) to the 16-bit Xi register and Yi register respectively.

**Flags:** –

**Notes:** –

**Examples:** EMUL X1Y0, X0,@RP0+S1, Y0,@RP3+S0

**# of Words:** 1

## ENEG/ENEGT<sup>\*</sup> – Negate

**Format:** ENEG(T) An

**Operation:**  $A_n \leftarrow \sim A_n + 1$

This instruction negates the value of one of 16-bit Accumulator (An), 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.

**V:** set if overflow is generated. Reset if not.

**N:** exclusive OR of V and MSB of result. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** \* ENEGT instruction can be executed only when the T flag is set.

Otherwise, No operation is performed.

**Examples:** ENEG A

ENEGT C

**# of Words:** 1

## ENMSK – Masking SG

**Format:** ENMSK SG,#imm:4

**Operation:** SG[15:0] <- SG[15:0] & mask pattern

This instruction masks MSB n bit (n = 16 - #imm:4) of SG[15:0] register, and stores back the result into the SG[15:0] register.

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

**VS:** Reset.

**N:** Reset.

**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  $\leftarrow$  0  
This instruction sets the specified bit in bs field to 0.

**Flags:** –

**Notes:** –

**Examples:** ER OP  
ER ME3

**# of Words:** 1

## ERESR – Restoring Remainder

**Format:** ERESR Mi,P

**Operation:** if (NQ = 0)  
Adder output  $\leftarrow$  MAi + 0  
else  
Adder output  $\leftarrow$  MAi + 2\*P

This Instruction adds two times of the P register and one of the MAi accumulator when NQ bit of MSR1 register is set. Else, performs no operation. It calculates true remainder value of non-restoring division.

**Flags:** **VMi<sup>+</sup>**: 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 Mi.

**Examples:** ERESR MA1,P

**# of Words:** 1



## ERND – Round

**Format:** ERND Mi

**Operation:** MAi <- MAi + 000008000h

This Instruction adds one of the 36-bit MAi accumulator and rounding constant and stores the result value into the same accumulator register. It performs two's complement rounding.

**Flags: VMi<sup>\*</sup>:** 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 Mi.

**Examples:** ERND MA0

**# of Words:** 1

## ERPD – Update Pointer w/ Destination Index

**Format:** ERPD rpd

**Operation:**  $R_{Pi} \leftarrow \text{mod}(R_{Pi} + D_0/D_1)$

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

## ERPNI – Update Pointer w/ Immediate Value

**Format:** ERPNI rpi,<op>  
<op> : #imm:16  
An

**Operation:**  $RPI \leftarrow \text{mod}(RPI + \text{<op>})$   
This Instruction updates the selected pointer with 16-bit <op> value. If <op> is one of 16-bit An register, LSB 16-bit of the accumulator An is only valid. 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:** ERPNI RP3,#1555h  
ERPNI RP1,A

**# of Words:** 1  
2 when <op> is #imm:16

## 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 current bank RP3 register pointer field. The source pointer value is not changed at all.

**Flags:** –

**Notes:** –

**Examples:** ERPR RP2

**# of Words:** 1

## ERPS – Update Pointer w/ Source Index

**Format:** ERPS rps

**Operation:**  $R_{Pi} \leftarrow \text{mod}(R_{Pi} + S_0/S_1)$

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  $\leftarrow$  1  
This instruction sets the specified bit in bs field to 1.

**Flags:** –

**Notes:** –

**Examples** ES OP  
ES ME3

**# of Words:** 1

## ESAT – Saturate

**Format:** ESAT Mi

**Operation:** if (VMi == 1)

MAi ← maximum magnitude

This Instruction sets the 36-bit MAi accumulator to the plus or minus maximum value when selected MAi register overflows. When no overflow occur, the MAi register is not changed.

**Flags:** VMi\*: Reset

**Notes:** \* VMi denotes for VM0 or VM1 according to Mi.

**Examples:** ESAT MA0

**# of Words:** 1

## ESD0/ESD1/ESD2/ESD3 – Source/Destination Index Load

**Format:** ESD0<sup>i</sup> ns, #imm:4  
ESD1 ns, #imm:4  
ESD2 ns, #imm:4  
ESD3<sup>i</sup> 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 current bank 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:** D0 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 7-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 7-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:**  $SG \leftarrow SG \mid (asr \lll asa)$

This instruction shifts the value of 16-bit asr values by the amount of 7-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:**  $SR \leftarrow asr \ll asa, SG \leftarrow SG | (asr \ll asa)$

This instruction shifts the value of 16-bit asr values by the amount of 7-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 <sup>(1)</sup>/ESLAT <sup>\*</sup> – Arithmetic 1-bit Left Shift Accumulator

**Format:**       ESLA(T) An

**Operation:**    An <- An<<1

This instruction shifts the value of one of 16-bit Accumulator (An) 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.

**V:**             Set if overflow is generated. Reset if not.

**N:**             Exclusive OR of V and MSB of result. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:**    \* ESLAT instruction can be executed only when the T flag is set.

Otherwise, No operation is performed.

**Examples:**     ESLA  A

                  ESLAT D

**# of Words:**    1

## ESLA <sup>(2)</sup> – Arithmetic 1-bit Left Shift Multiplier Accumulator

**Format:**       ESLA Mi

**Operation:**   MAi ← MAi <<1

This instruction shifts one of the 36-bit Multiplier Accumulator MAi to 1-bit left , and stores the result back into the same Multiplier Accumulator.

**Flags: VMi<sup>\*</sup>:**   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 Mi.

**Examples:**     ESLA   MA0

**# of Words:**    1

## ESLA8/ESLA8T<sup>\*</sup> – Arithmetic 8-bit Left Shift Accumulator

**Format:** ESLA8(T) An

**Operation:**  $An \leftarrow An \ll 8$

This instruction shifts the value of one of 16-bit Accumulator (An) 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.

**V:** Set if overflow is generated. Reset if not.

**N:** Exclusive OR of V and MSB of result. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** \* ESLA8T instruction can be executed only when the T flag is set.

Otherwise, No operation is performed.

**Examples:** ESLA8 C

ESLA8T B

**# of Words:** 1



## ESLC/ESLCT\* – Arithmetic 1-bit Left Shift Accumulator w/ Carry

**Format:** ESLC(T) An

**Operation:**  $A_n \leftarrow A_n \ll 1$ ,  $A_n[0] \leftarrow C$

This instruction shifts the value of one of 16-bit Accumulator ( $A_n$ ) to 1-bit left with carry : i.e. the carry bit is shifted into LSB of  $A_n$  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.

**V:** Set if overflow is generated. Reset if not.

**N:** Exclusive OR of V and MSB of result. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** \* ESLCT instruction can be executed only when the T flag is set.

Otherwise, No operation is performed.

**Example s:** ESLC A

ESLCT C

**# of Words:** 1

## ESRA <sup>(1)</sup>/ESRAT <sup>\*</sup> – Arithmetic 1-bit Right Shift Accumulator

**Format:** ESRA(T) An

**Operation:** An <- An >>1

This instruction shifts the value of one of 16-bit Accumulator (An) 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.

**V:** Set if overflow is generated. Reset if not.

**N:** Exclusive OR of V and MSB of result. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** \* ESLRT instruction can be executed only when the T flag is set.

Otherwise, No operation is performed.

**Examples:** ESRA A

ESRAT B

**# of Words:** 1

## ESRA <sup>(2)</sup> – Arithmetic 1-bit Right Shift Multiplier Accumulator

**Format:** ESRA Mi

**Operation:** MAi ← MAi >>1

This instruction shifts one of the 36-bit Multiplier Accumulator MAi to 1-bit right, and stores the result back into the same Multiplier Accumulator.

**Flags:** **VMi<sup>\*</sup>**: 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 Mi.

**Examples:** ESRA MA1

**# of Words:** 1

## ESRA8/ESRA8T<sup>\*</sup> – Arithmetic 8-bit Right Shift Accumulator

**Format:** ESRA8(T) An

**Operation:**  $An \leftarrow An \gg 8$

This instruction shifts the value of one of 16-bit Accumulator (An) 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.

**V:** Set if overflow is generated. Reset if not.

**N:** Exclusive OR of V and MSB of result. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** \* ESRA8T instruction can be executed only when the T flag is set.

Otherwise, No operation is performed.

**Examples:** ESRA8 D

ESRA8T B

**# of Words:** 1

## ESRC/ESRCT<sup>\*</sup> – Arithmetic 1-bit Right Shift Accumulator w/ Carry

**Format:** ESRC(T) An

**Operation:**  $An \leftarrow An \gg 1$ ,  $An[15] \leftarrow C$

This instruction shifts the value of one of 16-bit Accumulator (An) to 1-bit right with carry : i.e. the carry bit is shifted into MSB of An 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.

**V:** Set if overflow is generated. Reset if not.

**N:** Exclusive OR of V and MSB of result. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:** \* ESRCT instruction can be executed only when the T flag is set.

Otherwise, No operation is performed.

**Examples:** ESRC A

ESRCT B

**# of Words:** 1

## ESUB <sup>(1)</sup> – Subtract Accumulator

**Format:**       ESUB An, <op>  
                  <op>: #simm:16  
                  Am  
                  mg / mgx

**Operation:**    An <- An - <op>  
  
This instruction subtracts <op> value from the value of one of 16-bit Accumulator (An), and stores the result back into the same Accumulator. If <op> is immediate value, it is first right adjusted and sign-extended to 16-bit value. If <op> is 16-bit register, it is zero-extended.

**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. Refer to Chapter 2 for more detailed explanation about this convention.

**Notes:**        –

**Examples:**    SUB A, #0486h  
                  ESUB B, C  
                  ESUB D, RPO

**# of Words:**   1  
                  2 when <op> is : #simm:16

## ESUB<sup>(2)</sup> – Subtract Accumulator w/ One Parallel Move

**Format:**

1. ESUB Ai, Mi, <dest>, <src>  
 <dest>, <src>: mgx, @rps  
     Mi, @rps  
     @rpd, mga  
     @rpd, P
2. ESUB Ai, Ci, Cj, @rps

**Operation:**

1.  $A_i \leftarrow A_i - M_i$ ,  $\langle \text{dest} \rangle \leftarrow \langle \text{src} \rangle$
2.  $A_i \leftarrow A_i - C_i$ ,  $C_j \leftarrow @rps$

This instruction subtracts higher 16-bit part of Multiplier Accumulator MA<sub>i</sub> (MA0 or MA1 register) or the value of 16-bit Accumulator C<sub>i</sub> (C or D register) from the values of 16-bit Accumulator A<sub>i</sub> (A or B register), and stores the result back into the same accumulator A<sub>i</sub>. This instruction also stores a source operand from memory or register to the destination register or memory location.

**Flags:**

- C:** Set if carry is generated by addition. Reset if not.
- Z:** Set if result is zero by addition. Reset if not.
- V:** Set if overflow is generated by addition. Reset if not.
- N:** Exclusive OR of V and MSB of result by addition. Refer to Chapter 2 for more detailed explanation about this convention.
- VMi<sup>\*</sup>:** Set if result is overflowed to guard-bits. Reset if not.

**Notes:** \* VM<sub>i</sub> denotes for VM0 or VM1 according to M<sub>i</sub> if <dest> is M<sub>i</sub>.

**Examples:**

```

ESUB A, MA0, X0, @RP0+S1
ESUB B, MA1, MA1, @RP1+S0
ESUB B, MA0, @RP3+D1, A
ESUB A, C, C, @RP2+S1
  
```

**# of Words:** 1

## ESUB<sup>(3)</sup> – Subtract Multiplier Accumulator

**Format:**        ESUB Mi, <op>  
                  <op>: P / PSH

**Operation:**    MAi <- MAi - <op>  
                  This instruction subtracts <op> value from the values of 36-bit Multiplier Accumulator MAi, and stores the result back into the same Multiplier Accumulator MAi. The "PSH" means 16-bit arithmetic right shifted P register value.

**Flags:** **VMi**<sup>\*</sup>: 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 Mi.

**Examples:**     ESUB MA0, P  
                  ESUB MA1, PSH

**# of Words:**    1



## ESUB<sup>(4)</sup> – Subtract Multiplier Accumulator w/One Parallel Move

**Format:**       ESUB Mi, P <dest>,<src>  
                  <dest>,<src>: mgx, @rps  
                  An, @rps  
                  @rpd, mga  
                  @rpd, P

**Operation:**   MAi <- MAi - P, <dest> <- <src>  
                  This instruction subtracts the value of the Product register P from the value of 36-bit Multiplier Accumulator MAi, and stores the result back into the same Multiplier Accumulator MAi. This instruction also stores source operand from memory or register to destination register or memory.

**Flags:** **VMi**<sup>\*</sup>: 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 Mi.

**Examples:**    ESUB MA0, P,   Y0, @RP1+S1  
                  ESUB MA1, P,   C, @RP2+S0  
                  ESUB MA1, P,   @RP0+D0, B

**# of Words:**  1

## ETST – Test

**Format:** ETST cct, <op>  
<op> : T  
EC3

**Operation:** if (cct is true)  
                  <op> <- 1  
          else  
                  <op> <- 0

This instruction sets the T flag of MSR0 register or EC[3] output port of CalmMAC16 to 1 if condition specified in cct field is evaluated to truth. Else, resets <op>. This instruction must be executed before executing the conditional instructions or branch instruction with EC3 as a condition code.

**Flags:** T : Set/reset according to the condition

**Notes:** –

**Examples:** ETST GT, EC3  
              ETST NEG, T

**# of Words:** 1

## NOTES

# 22 PROGRAM MEMORY ACCESS SPEED

## OVERVIEW

The FMCON.0 had better be set logic "1" when the CPU clock is under 10 MHz. It will be helped to reduction current consumption. In order to enable Y-Data ROM, FMCON.[1] is set to "1".

**3F0078H**

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
Reset Value	–	–	–	–	–	–	0	0
Read/Write	–	–	–	–	–	–	R/W	R/W

.7–.2

**Bits 7–2**

–	Not used
---	----------

.1

**Y-Data ROM Control Bit**

0	Disable Y-Data ROM
1	Enable Y-Data ROM

.0

**Flash Memory Accessing Speed Selection Bit**

0	When fxx is more than 10 MHz
1	When fxx is under 10 MHz

## NOTES

# 23

## ELECTRICAL DATA

### OVERVIEW

Table 23-1. Absolute Maximum Ratings

( $T_A = 25^\circ\text{C}$ )

Parameter	Symbol	Conditions	Rating	Unit
Supply voltage	$V_{DD}$	–	– 0.3 to + 4.6	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 All I/O pins active	– 18 – 60	mA
Output current low	$I_{OL}$	One I/O pin active Total pin current	+ 30 + 100	mA
Operating temperature	$T_A$	–	– 25 to + 85	$^\circ\text{C}$
Storage temperature	$T_{STG}$	–	– 65 to + 150	$^\circ\text{C}$

Table 23-2. D.C. Electrical Characteristics

( $T_A = -25^\circ\text{C}$  to  $+85^\circ\text{C}$ ,  $V_{DD} = 2.0\text{ V}$  to  $3.6\text{ V}$ )

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Operating Voltage	$V_{DD}$	$f_x = 32\text{MHz}$	3.0	–	3.6	V
		$f_x = 4\text{MHz}$	2.0	–	3.6	
Input high voltage	$V_{IH1}$	Ports 0 – 9, nRESET	$0.8 V_{DD}$	–	$V_{DD}$	V
	$V_{IH2}$	$X_N$ and $X_{tIN}$	$V_{DD}-0.1$	–	$V_{DD}$	
Input low voltage	$V_{IL1}$	Ports 0 – 9, nRESET	–	–	$0.2 V_{DD}$	V
	$V_{IL2}$	$X_N$ and $X_{tIN}$	–	–	0.1	

Table 23-2. D.C. Electrical Characteristics (Continued)

(T<sub>A</sub> = -25°C to +85°C, V<sub>DD</sub> = 2.0 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Output high voltage	V <sub>OH</sub>	V <sub>DD</sub> = 3.0 V to 3.6 V I <sub>OH</sub> = -1 mA Ports 0-9	V <sub>DD</sub> -1.0	-	-	V
Output low voltage	V <sub>OL</sub>	V <sub>DD</sub> = 3.0 V to 3.6 V I <sub>OL</sub> = 15 mA Ports 0-9	-	-	1.0	V
Input high leakage current	I <sub>LH1</sub>	V <sub>I</sub> = V <sub>DD</sub> All input pins except I <sub>LH2</sub>	-	-	3	μA
	I <sub>LH2</sub>	V <sub>IN</sub> = V <sub>DD</sub> X <sub>IN</sub> , X <sub>OUT</sub> , XT <sub>IN</sub>	-	-	20	
Input low leakage current	I <sub>LIL1</sub>	V <sub>IN</sub> = 0 V All input pins except I <sub>LIL2</sub>	-	-	-3	
	I <sub>LIL2</sub>	V <sub>IN</sub> = 0 V X <sub>IN</sub> , X <sub>OUT</sub> , XT <sub>IN</sub>	-	-	-20	
Output high leakage current	I <sub>LOH</sub>	V <sub>OUT</sub> = V <sub>DD</sub> All Output pins	-	-	3	
Output low leakage current	I <sub>LOL</sub>	V <sub>OUT</sub> = 0 V All Output pins	-	-	-3	
Pull-up resistor	RL1	V <sub>IN</sub> = 0 V; V <sub>DD</sub> = 3.3 V; T <sub>A</sub> = 25°C Ports 0-9	3	60	90	kΩ
	RL2	V <sub>IN</sub> = 0 V; V <sub>DD</sub> = 3.3 V; T <sub>A</sub> = 25°C nRESET	150	300	450	
LCD Voltage Dividing Resistor	R <sub>LCD</sub>	T <sub>A</sub> = 25 °C	30	50	70	kΩ
VLCD-COMi   Voltage Drop (i = 0-7)	V <sub>DC</sub>	-15 μA per common pin	-	-	120	mV
VLCD-SEGx   Voltage Drop (x = 0-39)	V <sub>DS</sub>	-15 μA per common pin	-	-	120	
Middle Output Voltage <sup>(1)</sup>	V <sub>LC2</sub>	V <sub>DD</sub> = 2.4V to 3.6V, 1/5 bias LCD clock = 0Hz, V <sub>LC1</sub> = V <sub>DD</sub>	0.8V <sub>DD</sub> -0.2	0.8V <sub>DD</sub>	0.8V <sub>DD</sub> +0.2	V
	V <sub>LC3</sub>		0.6V <sub>DD</sub> -0.2	0.6V <sub>DD</sub>	0.6V <sub>DD</sub> +0.2	
	V <sub>LC4</sub>		0.4V <sub>DD</sub> -0.2	0.4V <sub>DD</sub>	0.4V <sub>DD</sub> +0.2	
	V <sub>LC5</sub>		0.2V <sub>DD</sub> -0.2	0.2V <sub>DD</sub>	0.2V <sub>DD</sub> +0.2	

**NOTE:** It is middle output voltage when LCD controller/driver is 1/8 duty and 1/5 bias.

Table 23-2. D.C. Electrical Characteristics (Continued)

(T<sub>A</sub> = -25°C to +85°C, V<sub>DD</sub> = 2.0 V to 3.6 V)

Parameter	Symbol	Conditions		Min	Typ	Max	Units
Supply Current (1)	I <sub>DD1</sub> (2)	V <sub>DD</sub> = 3.3 V ± 10% Crystal oscillator	20 MHz 4 MHz	–	16.0 4.0	32.0 8.0	mA
		f <sub>x</sub> =f <sub>OUT</sub> , 2.048 MHz X <sub>tal</sub>	32.768 MHz	–	25.0	50.0	
	I <sub>DD2</sub> (2)	Idle mode V <sub>DD</sub> = 3.3 V ± 10% Crystal oscillator	20 MHz 4 MHz	–	1.0 0.4	2.0 0.8	mA
	I <sub>DD3</sub> (3)	V <sub>DD</sub> = 3.3 V ± 10% 32 kHz crystal oscillator		–	60	100	µA
	I <sub>DD4</sub> (3)	Idle mode; V <sub>DD</sub> = 3.3 V ± 10% 32 kHz crystal oscillator	T <sub>A</sub> = -5 °C to + 85 °C	–	6.0	20	µA
I <sub>DD5</sub>	Stop mode; V <sub>DD</sub> = 3.3 V ± 10% T <sub>A</sub> = 25 °C	X <sub>TIN</sub> = 0V	25 °C		0.2	2.0	µA
			-25°C~ 85°C		–	10	

**NOTES:**

- Supply current does not include current drawn through internal pull-up resistors, PWM, PLL, or external output current loads.
- I<sub>DD1</sub> and I<sub>DD2</sub> include power consumption through sub clock oscillation and main oscillator is in normal mode.
- I<sub>DD3</sub> and I<sub>DD4</sub> are current when main clock oscillation stops and the sub clock is used in normal mode.
- Every value in this table is measured when bits 1-0 of the clock control register (CLKCON.1 -.0) is set to 11B.
- If the f<sub>out</sub> (PLL's output clock) is used for the system clock, the current consumption is added by 0.2mA/1MHz at V<sub>DD</sub> = 3.3V and the current through PLL block to I<sub>DD1</sub>.
- The current is added a little when Y-ROM is enabled.



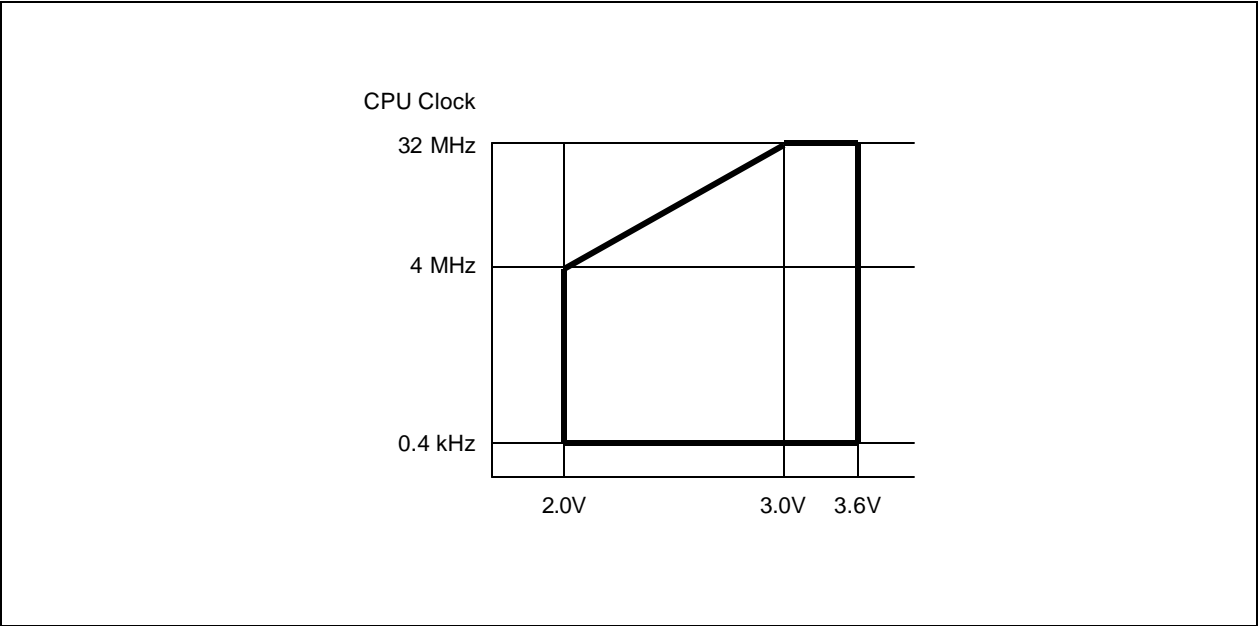
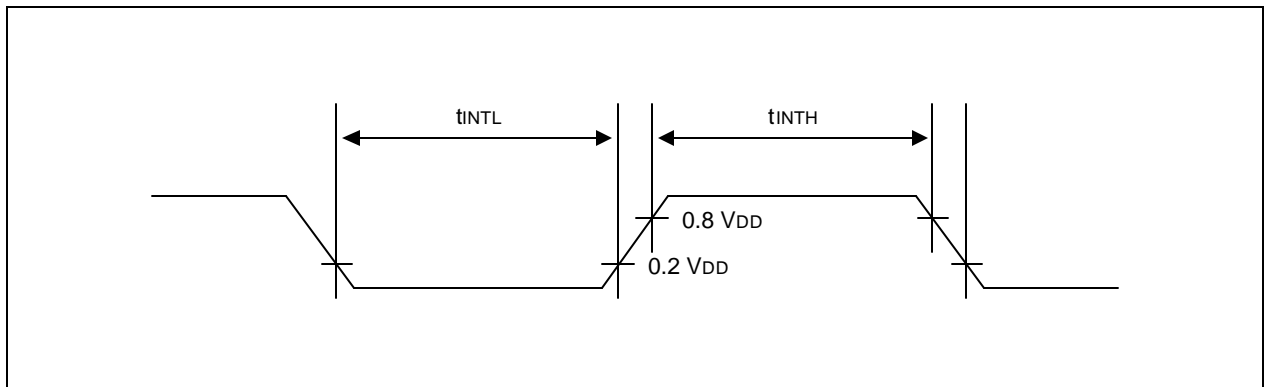


Figure 23-1. Operating Voltage Range

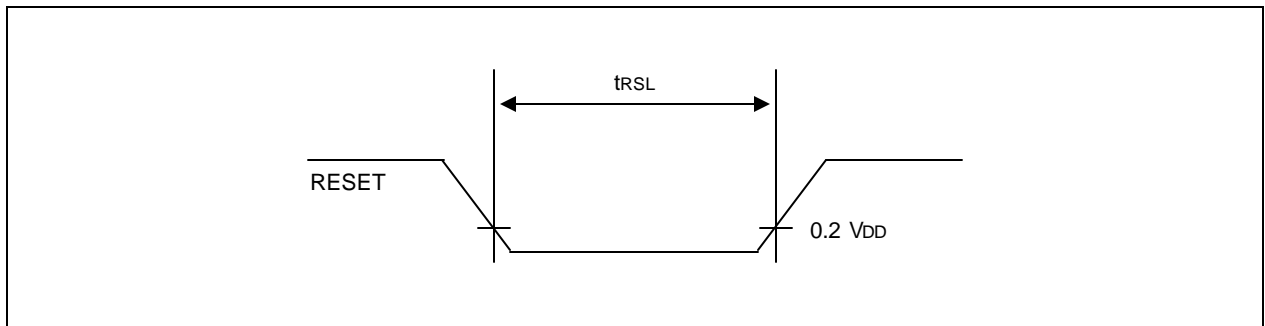
**Table 23-3. A.C. Electrical Characteristics**

( $T_A = -25\text{ }^\circ\text{C}$  to  $+85\text{ }^\circ\text{C}$ ,  $V_{DD} = 2.0\text{ V}$  to  $3.6\text{ V}$ )

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Interrupt input high, low width	$t_{INTH}$ , $t_{INTL}$	P0.0 – P0.3, P4.4 – P4.7; $V_{DD} = 3.3\text{ V}$	200	–	–	ns
RESET input low width	$t_{RSL}$	$V_{DD} = 3.3\text{ V}$	10	–	–	us



**Figure 23-2. Input Timing for External Interrupts (Ports 0, Ports 4)**

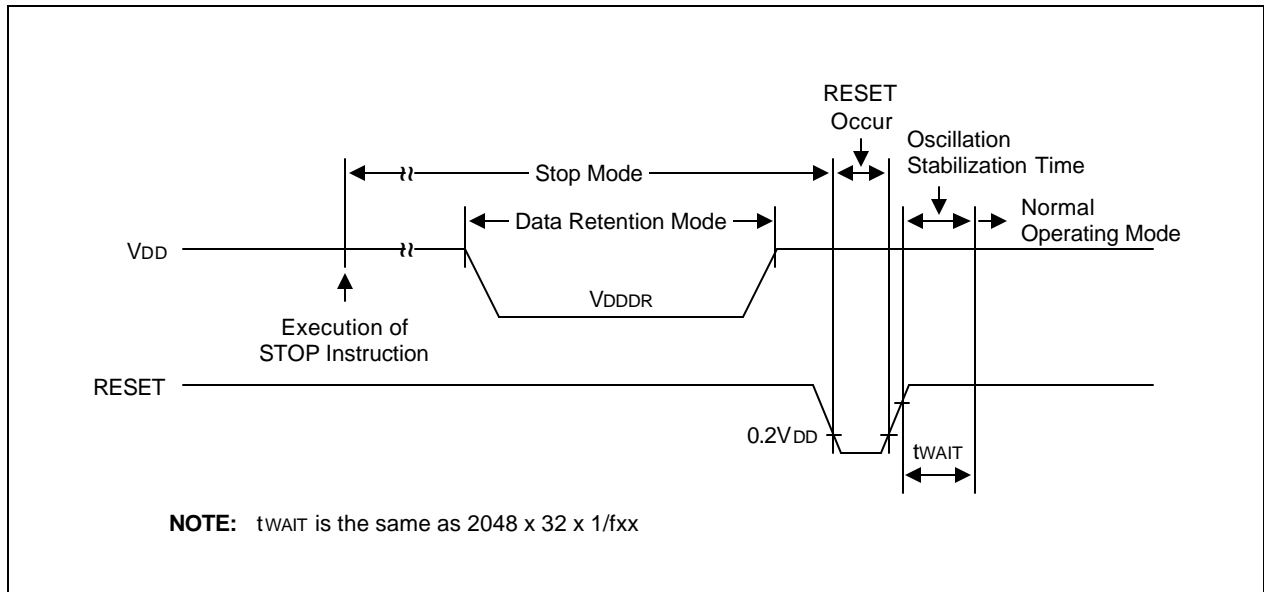


**Figure 23-3. Input Timing for nRESET**

**Table 23-4. Data Retention Supply Voltage in Stop Mode**

( $T_A = -25\text{ }^\circ\text{C}$  to  $+85\text{ }^\circ\text{C}$ )

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	$V_{DDDR}$		2.0	–	3.6	V
Data retention supply current	$I_{DDDR}$	$V_{DDDR} = 1.5\text{ V}$	–	–	2	$\mu\text{A}$



**Figure 23-4. Stop Mode Release Timing When Initiated by a nRESET**

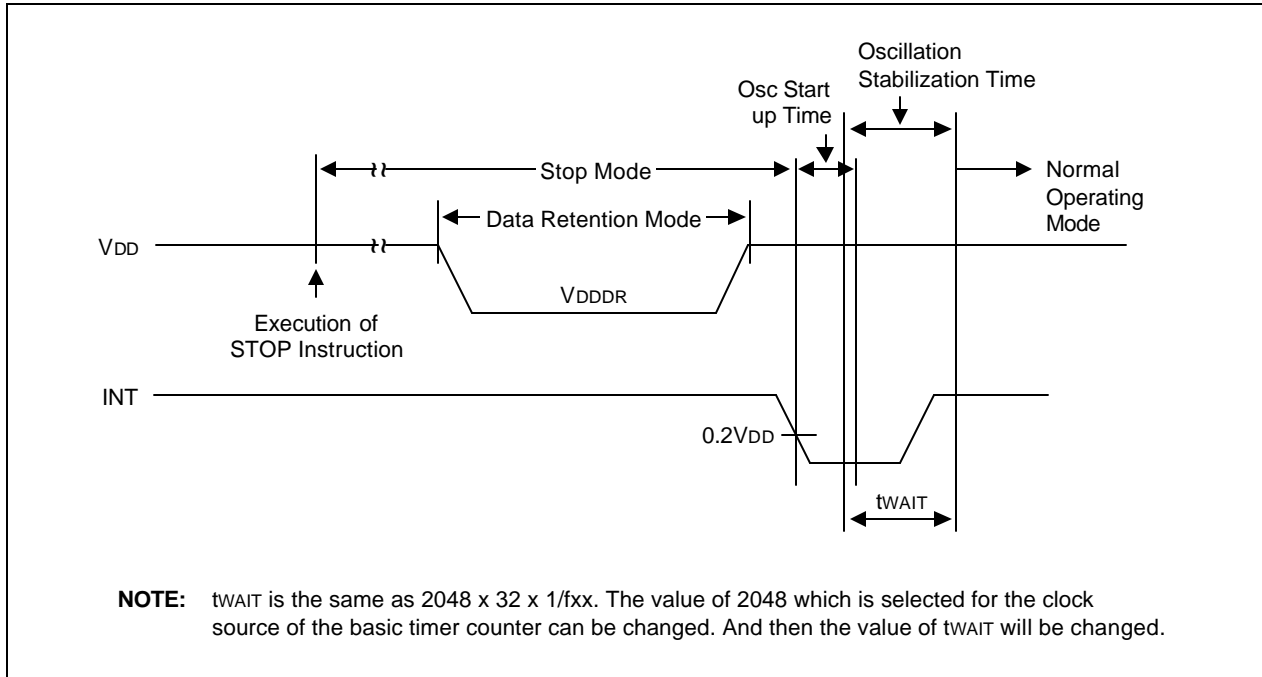


Figure 23-5. Stop Mode(main) Release Timing Initiated by Interrupts

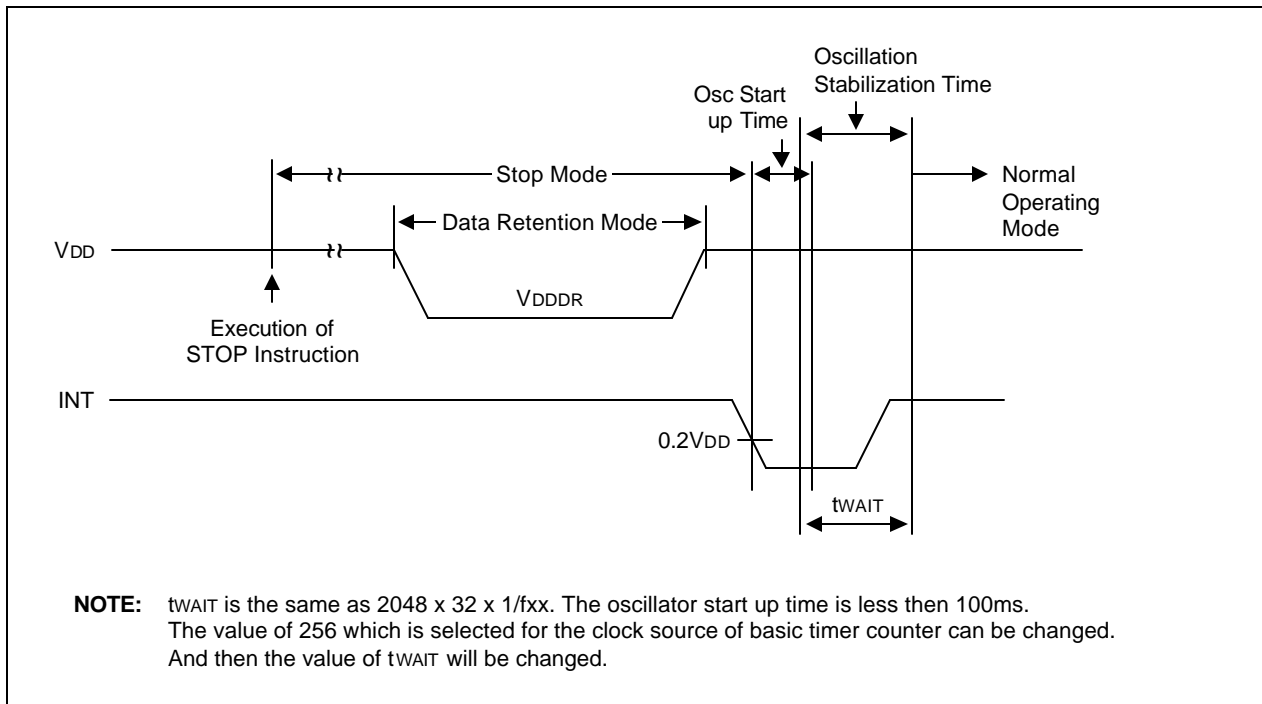
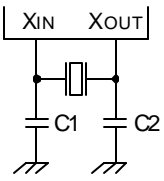
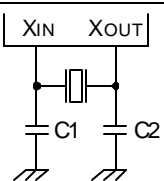
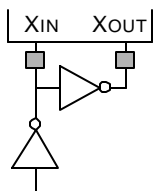


Figure 23-6. Stop Mode(sub) Release Timing Initiated by Interrupts

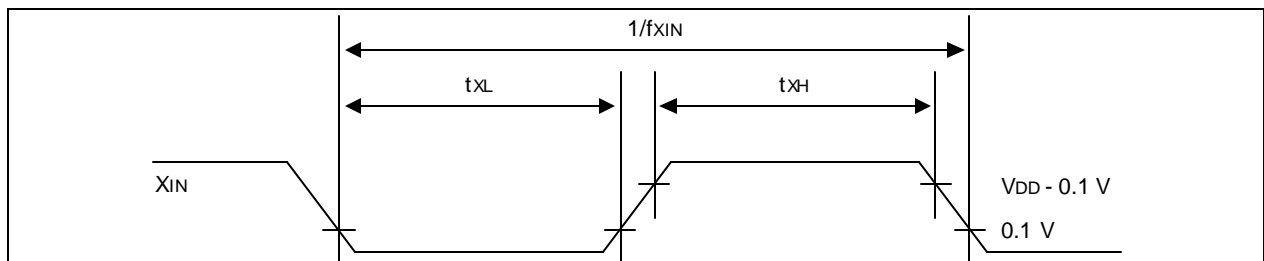
Table 23-5. Main Oscillator Characteristics

(T<sub>A</sub> = -25 °C to +85 °C, V<sub>DD</sub> = 2.0 V to 3.6 V)

Oscillator	Clock Configuration	sParameter	Test Condition	Min	Typ	Max	Unit
Ceramic Oscillator		Oscillation frequency <sup>(1)</sup>	–	0.4	–	20	MHz
		Stabilization time <sup>(2)</sup>	Stabilization occurs when V <sub>DD</sub> is equal to the minimum oscillator voltage range.	–	–	4	ms
Crystal Oscillator		Oscillation frequency <sup>(1)</sup>	–	0.4	–	20	MHz
		Stabilization time <sup>(2)</sup>	V <sub>DD</sub> =2.0V to 3.6V	–	–	30	ms
External Clock		X <sub>N</sub> input frequency <sup>(1)</sup>	–	0.4	–	20	MHz
		X <sub>N</sub> input high and low level width (t <sub>XH</sub> , t <sub>XL</sub> )	–	62.0	–	1250	ns

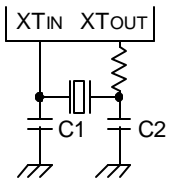
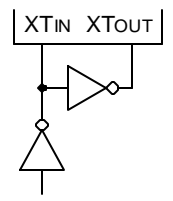
**NOTES:**

- Oscillation frequency and X<sub>N</sub> input frequency data are for oscillator characteristics only.
- Stabilization time is the interval required for oscillating stabilization after a power-on occurs, or when stop mode is terminated.

Figure 23-7. Clock Timing Measurement at X<sub>IN</sub>

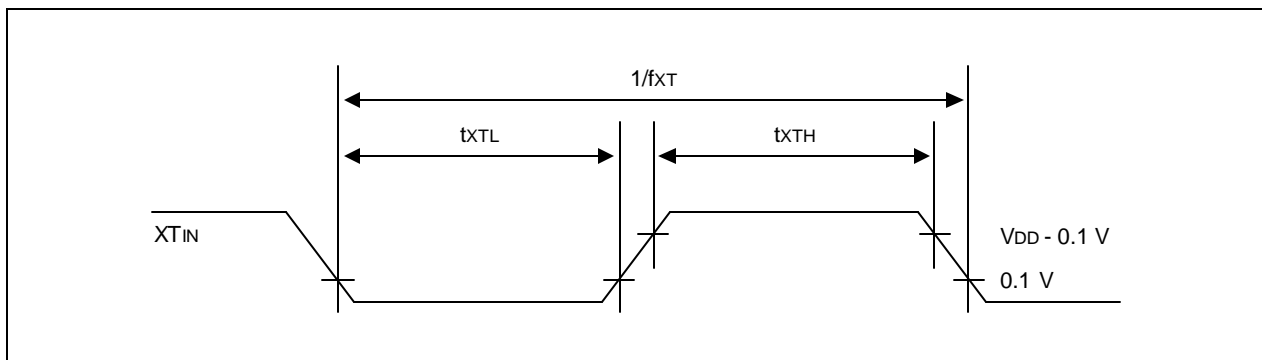
**Table 23-6. Sub Oscillator Frequency**

( $T_A = -25\text{ }^\circ\text{C}$  to  $+85\text{ }^\circ\text{C}$ ,  $V_{DD} = 2.0\text{ V}$  to  $3.6\text{ V}$ )

Oscillator	Clock Configuration	Parameter	Test Condition	Min	Typ	Max	Unit
Crystal Oscillator		Oscillation frequency <sup>(1)</sup>	–	32	32.768	35	kHz
		Stabilization time <sup>(2)</sup>	$V_{DD} = 3.0\text{V}$ to $3.6\text{V}$	–	1.0	2	s
			$V_{DD} = 2.0\text{V}$ to $3.6\text{V}$	–	3.0	10	s
External Clock		$XT_N$ input frequency	–	32	–	100	kHz
		$XT_N$ input high and low level width ( $t_{XTH}$ , $t_{XTL}$ )	–	5	–	15	us

**NOTES:**

- Oscillation frequency and  $XT_{in}$  input frequency data are for oscillator characteristics only.
- Stabilization time is the interval required for oscillating stabilization after a power-on occurs .



**Figure 23-8. Clock Timing Measurement at  $XT_{IN}$**

Table 23-7. BLD Electrical Characteristics

(T<sub>A</sub> = -25 °C, V<sub>DD</sub> = 2.0 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
BLD Voltage	V <sub>B0</sub>	BLDCON.4-2 = 001B	2.2	2.45	2.7	V
	V <sub>B1</sub>	BLDCON.4-2 = 111B	2.45	2.70	2.95	
BLD Circuit Response Time	t <sub>B</sub>	f <sub>W</sub> = 32.768 kHz	-	-	1.0	mS
BLD Operating Current	I <sub>BL</sub>	-	-	50	100	μA

Table 23-8. PLL Electrical Characteristics

(T<sub>A</sub> = -25 °C to +85 °C, V<sub>DD</sub> = 3.0 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Units
Input Clock Frequency	f <sub>in</sub>	-	-	2.048	-	MHz
Output Clock Frequency	f <sub>vco</sub>	-	16.38	-	32.768	
Output Clock Duty	-	-	40	-	60	%
Settling Time	T <sub>d</sub>	V <sub>DD</sub> = 3.3V	-	-	600	μS
Accuracy	-	-	-	-	1	%

Table 23-9. 10-Bit A/D Converter Electrical Characteristics

(T<sub>A</sub> = -25 °C to +85 °C, V<sub>DD</sub> = 3.0 V to 3.6 V)

Parameter	Symbol	Condition	Min	Typ	Max	Units
Resolution	-	-	-	10	-	bit
Total Accuracy	-	V <sub>DD</sub> = 3.3V ADC clock = 2MHz	-	-	± 3	LSB
Integral Linearity Error	ILE		-	-	±2	
Differential Linearity Error	DLE		-	-	± 1	
Offset Error of Top	EOT		-	± 1	± 3	
Offset Error of Bottom	EOB		-	± 1	± 3	
Conversion Time (1)	T <sub>CON</sub>	-	25	-	-	μS
Analog Input Voltage	V <sub>IAN</sub>	-	V <sub>SS</sub>	-	V <sub>DD</sub>	V
Analog Input Impedance	R <sub>AN</sub>	-	2	1000	-	MΩ
Analog Input Current	I <sub>ADIN</sub>	V <sub>DD</sub> = 3.3V	-	-	10	μA
Analog Block Current	I <sub>ADC</sub>		-	1	3	mA

**NOTES:**

1. Conversion time<sup>1</sup> is the time required from the moment a conversion operation starts until it ends.
2. I<sub>ADC</sub> is an operating current during A/D conversion

**Table 23-10. 14-bit ADC/DAC Electrical Characteristics**(T<sub>A</sub> = 0 °C to +70 °C, V<sub>DD</sub> = 3.0 V to 3.6 V, V<sub>SS</sub> = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Units
ADC Operating Current	I <sub>ADC</sub>	V <sub>DD</sub> = 3.3 V, f <sub>s</sub> = 8 kHz	–	1.5	3	mA
DAC Operating Current	I <sub>DAC</sub>	V <sub>DD</sub> = 3.3 V, f <sub>s</sub> = 8 kHz	–	1.5	3	mA
Sampling Frequency	f <sub>s</sub>	-	3.6	8	11	kHz
Resolution	–	Input sine wave: 1 kHz Measurement Bandwidth: 20 Hz – 4 kHz f <sub>s</sub> = 8 kHz	–	14	–	bits
Offset Error	–		–	–	±20	mV
Signal-to-(Noise + THD) Ratio	–		70	75	–	dB
ADC Input Voltage Range	–	V <sub>DD</sub> = 3.3V	–	–	1.8	V <sub>pp</sub>
DAC Output Voltage Range	–	V <sub>DD</sub> = 3.3V	–	–	1.8	



**NOTES**

# 24 MECHANICAL DATA

## OVERVIEW

The S3CC11B/FC11B microcontroller is currently available in a 100-pin QFP and TQFP package.

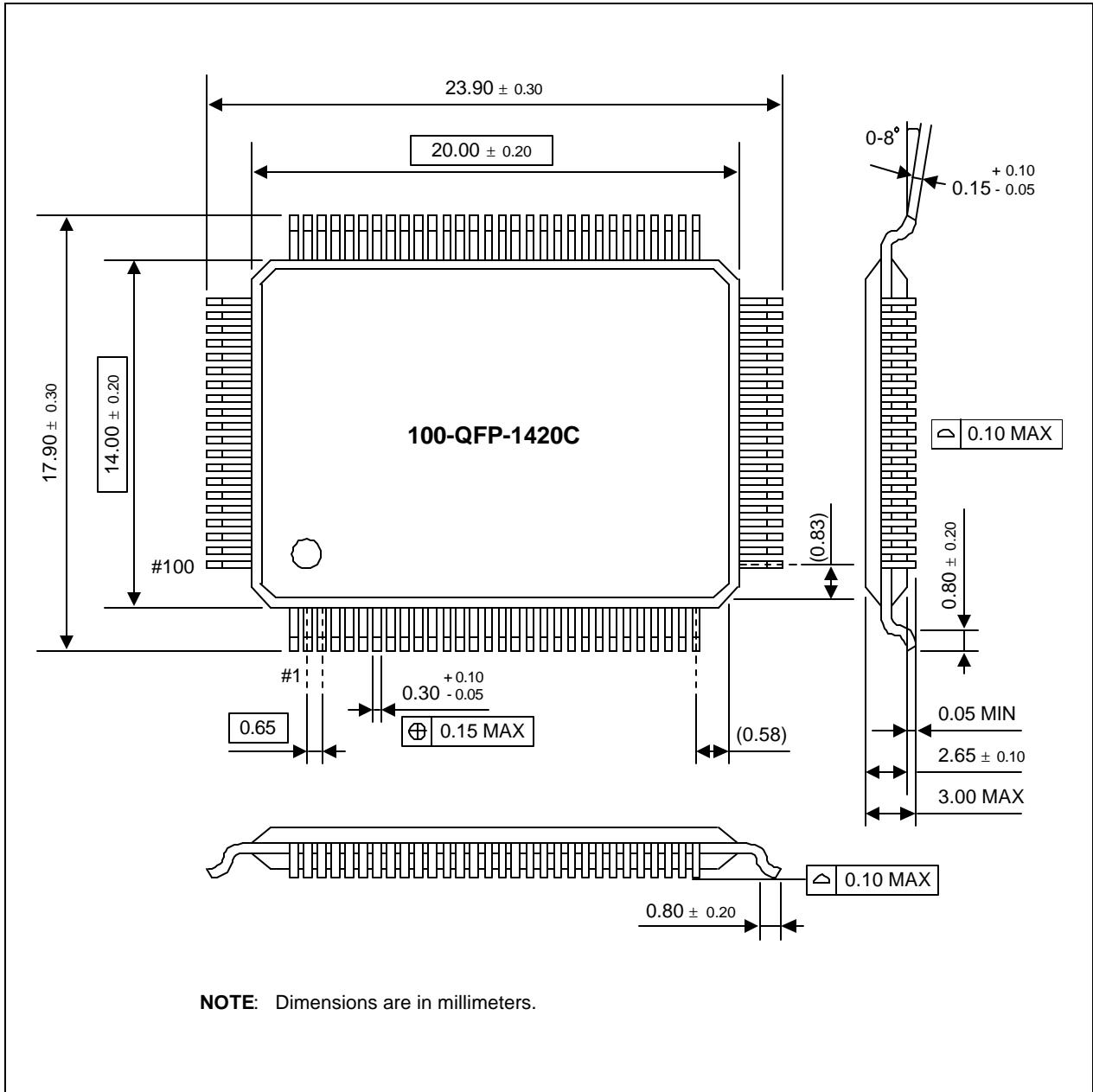


Figure 24-1. 100-QFP -1420C Package Dimensions

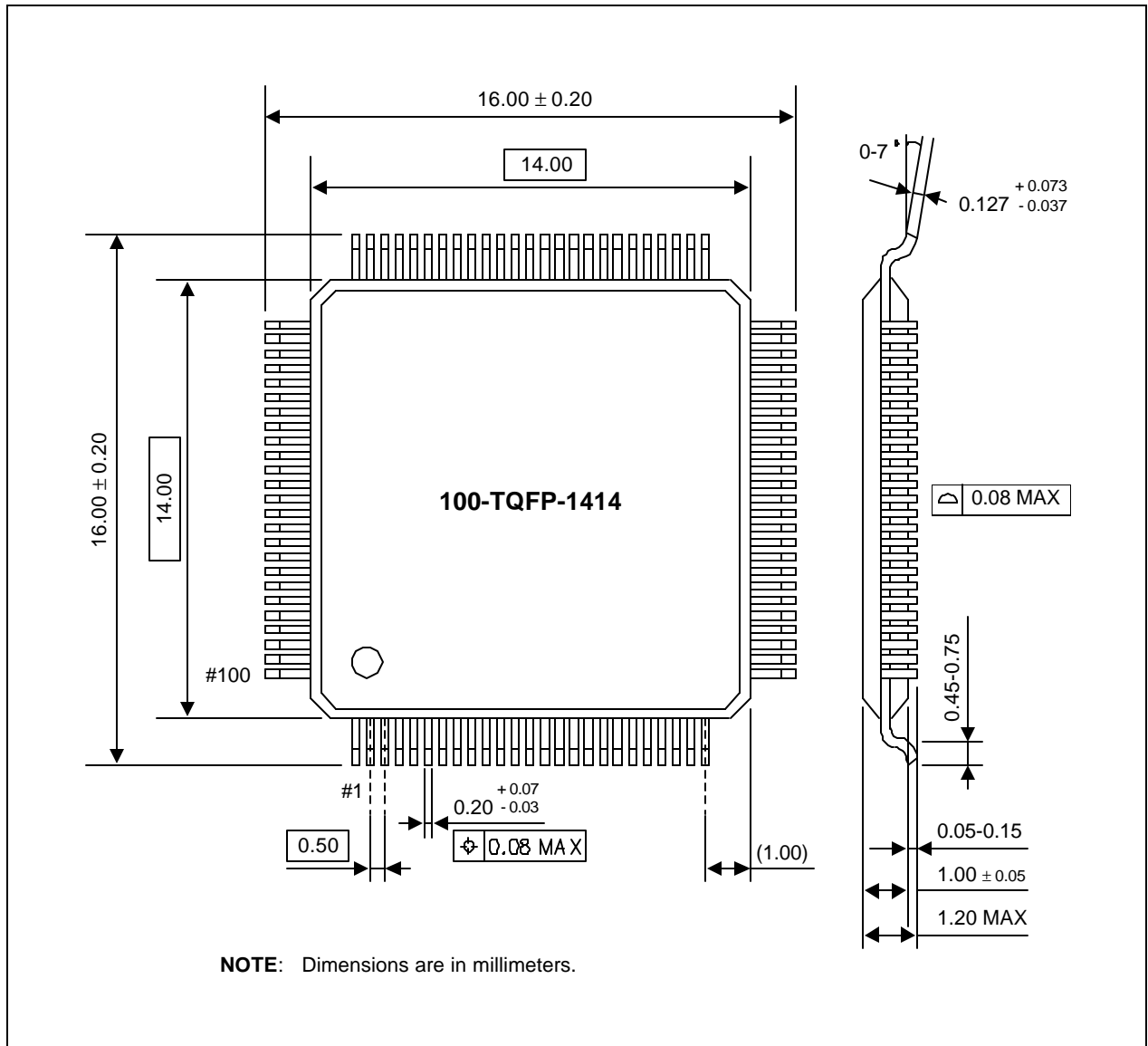


Figure 24-2. 100-TQFP-1414 Package Dimensions

**NOTES**

# 25

## S3FC11B FLASH MCU

### OVERVIEW

The S3FC11B single-chip CMOS microcontroller is the FLASH ROM version of the S3CC11B microcontroller. It has an on-chip FLASH ROM instead of masked ROM. The FLASH ROM is accessed by serial data formats.

The S3FC11B is fully compatible with S3CC11B, both in function and in electrical characteristics. Because of its simple programming requirements, the S3FC11B is ideal for use as an evaluation for the S3CC11B.

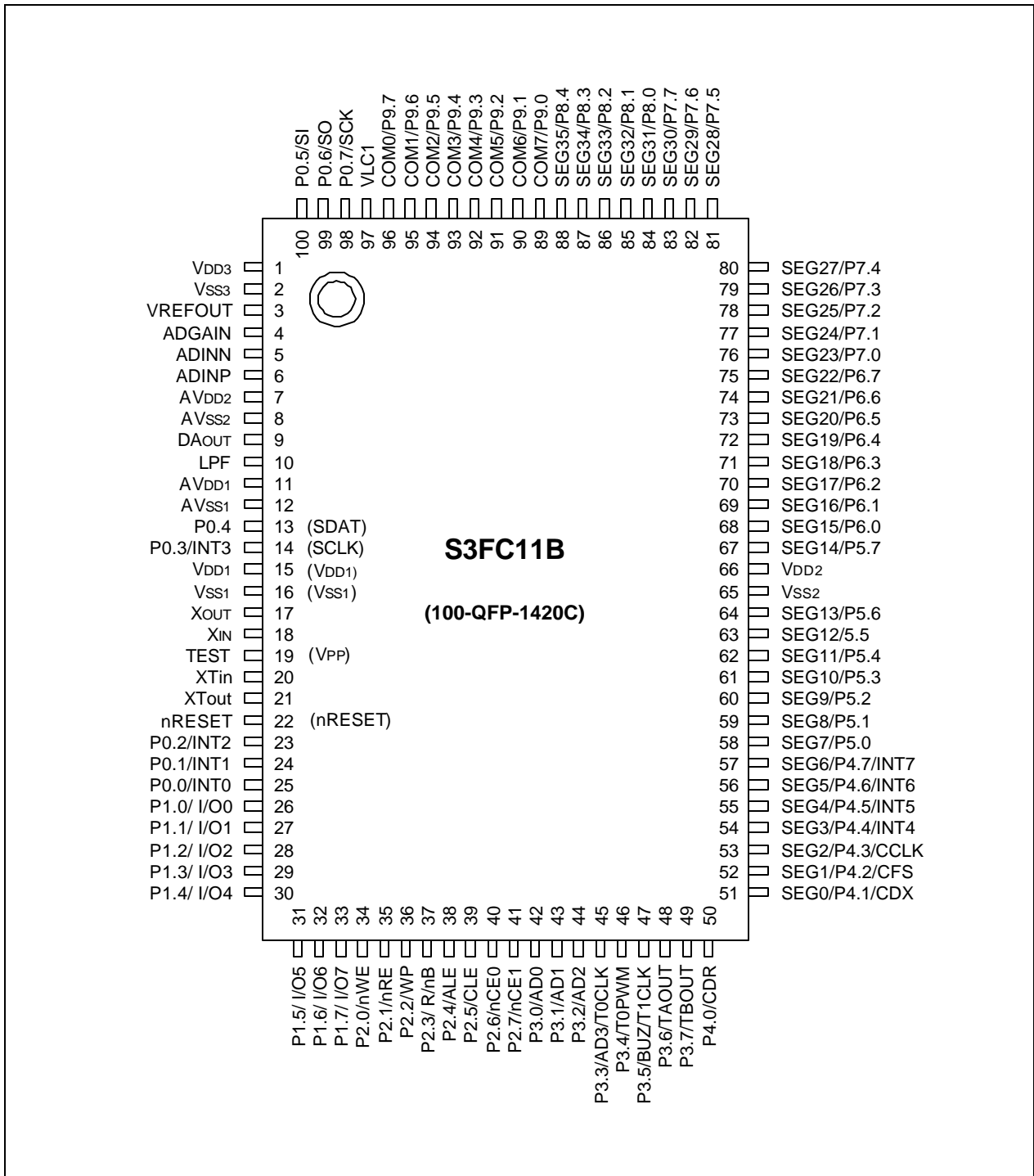


Figure 25-1. S3FC11B Pin Assignments (100-QFP-1420C)

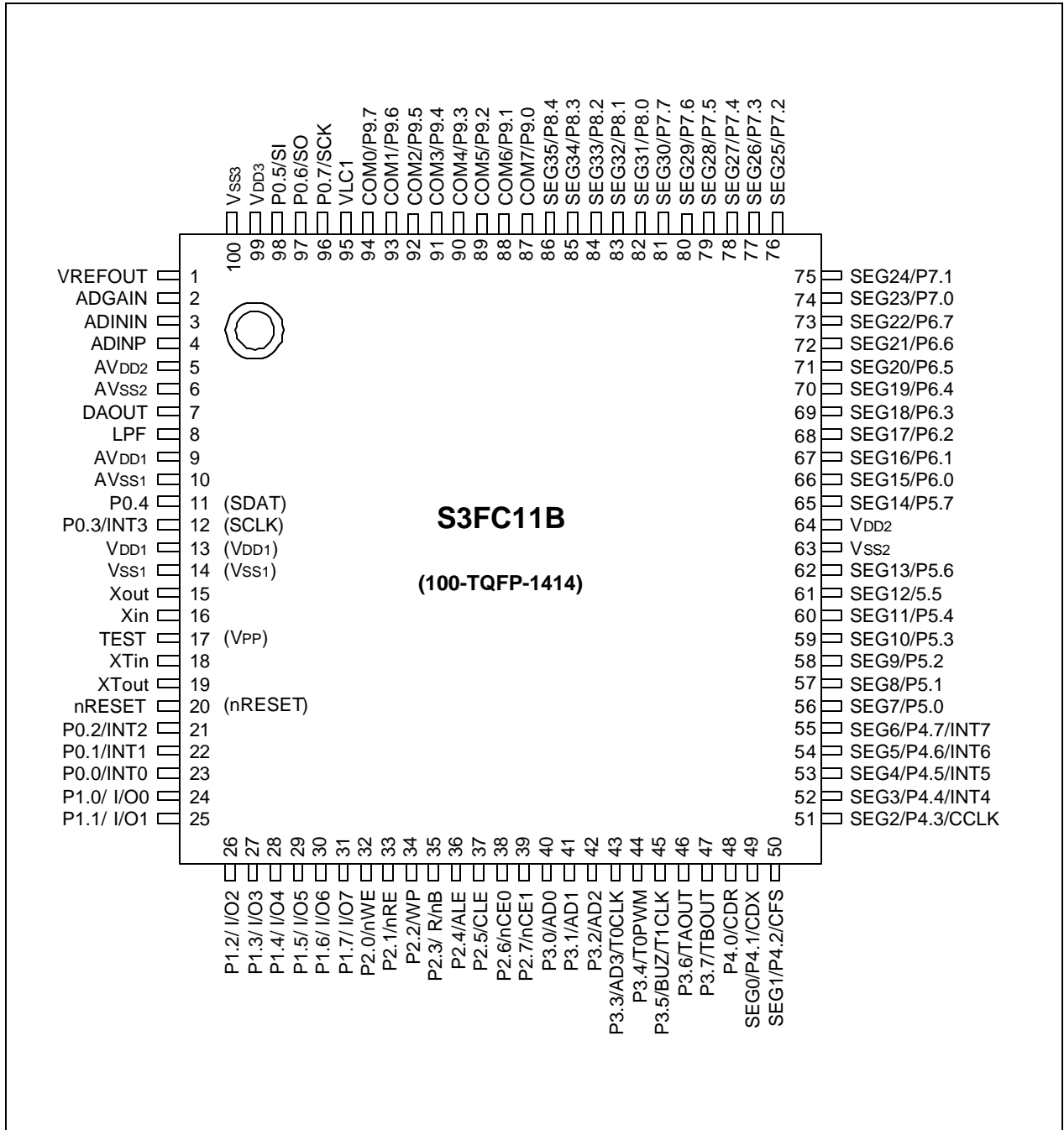


Figure 25-2. S3FC11B Pin Assignments (100-TQFP-1414)



Table 25-1. Descriptions of Pins Used to Read/Write the FLASH ROM

During Programming			
Pin Name	Pin No.	I/O	Function
SDAT (P0.4)	100 QFP: 13 100 TQFP: 11	I/O	Serial data pin. Output port when reading and input port when writing. Can be assigned as a Input/push-pull output port.
SCLK (P0.3)	100 QFP: 14 100 TQFP: 12	I/O	Serial clock pin. Input only pin.
V <sub>PP</sub> (TEST)	100 QFP: 19 100 TQFP: 17	I	Power supply pin for FLASH ROM cell writing (indicates that FLASH enters into the writing mode). When 12.5 V is applied, FLASH is in writing mode and when 3.3 V is applied, FLASH is in reading mode. When FLASH is operating, hold GND.
RESET (nRESET)	100 QFP: 22 100 TQFP: 20	I	Chip initialization
V <sub>DD1</sub> /V <sub>SS1</sub> (V <sub>DD1</sub> /V <sub>SS1</sub> )	100 QFP: 15/16 100 TQFP: 13/14	I	Logic power supply pin. V <sub>DD</sub> should be tied to 3.3 V during programming.

# 26 DEVELOPMENT TOOLS

## OVERVIEW

Samsung provides a powerful and easy-to-use development support system in turnkey form. The development support system is configured with a host system, debugging tools, and support software. For the host system, any standard computer that operates with windows95/98/NT/XP as its operating system can be used. One type of debugging tool including hardware and software is provided: the effective cost and powerful in-circuit emulator, InvisibleMDS, for CalmRISC16. Samsung also offers support software that includes debugger, Compiler, Assembler, and a program for setting options.

### **CalmSHINE: IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)**

CalmRISC16 Samsung Host Interface for In-circuit Emulator, CalmSHINE, is a multi window based debugger for CalmRISC16. CalmSHINE provides pull-down, pop-up and tool-bar menus, mouse support, function/hot keys, syntax highlight, tool-tip, drag-and-drop and context-sensitive hyper-linked help. It has an advanced, multiple-windowed user interface that emphasizes ease of use. Each window can be sized, moved, scrolled, highlighted, added or removed, docked or undocked completely.

### **IN-CIRCUIT EMULATOR**

The evaluation chip of CalmRISC16 has a basic debugging utility block. Using this block, evaluation chip directly interfaces with host through only communication wire. So, InvisibleMDS offers simple and powerful debugging environment.

### **CalmRISC16 C-COMPILER: CalmCC16**

The CalmRISC16 Compiler offers the standard features of the C language, plus many extensions for MCU applications, such as interrupt handling in C and data placement controls, designed to take full advantage of CalmRISC16 facilities. It conforms to the ANSI specification. It supports standard library of functions applicable to MCU systems. The standard library also conforms to the ANSI standard. It generates highly size-optimized code for CalmRISC16 by fully utilizing CalmRISC16 architecture. It is available in a Windows version integrated with the CalmSHINE.

### **CalmRISC16 RELOCATABLE ASSEMBLER: Calm8ASM**

The CalmRISC16 Assembler is a relocatable assembler for Samsung's CalmRISC16 MCU and its MAC1616 and MAC2424 coprocessors. It translates a source file containing assembly language statements into a relocatable machine object code file in Samsung format. It runs on WINDOWS95 compatible operating systems. It supports macros and conditional assembly. It produces the relocatable object code only, so the user should link object files. Object files can be linked with other object files and loaded into memory.

### **CalmRISC16 LINKER: Calm8LINK**

The CalmRISC16 Linker combines Samsung object format files and library files and generates absolute, machine-code executable hex programs or binary files for CalmRISC16 MCU and its MAC1616 and MAC2424 coprocessors. It generates the map file, which shows the physical addresses to which each section and symbol is bounded, start addresses of each section and symbol, and size of them. It runs on WINDOWS95 compatible operating systems.

### EMULATION PROBE BOARD CONFIGURATION

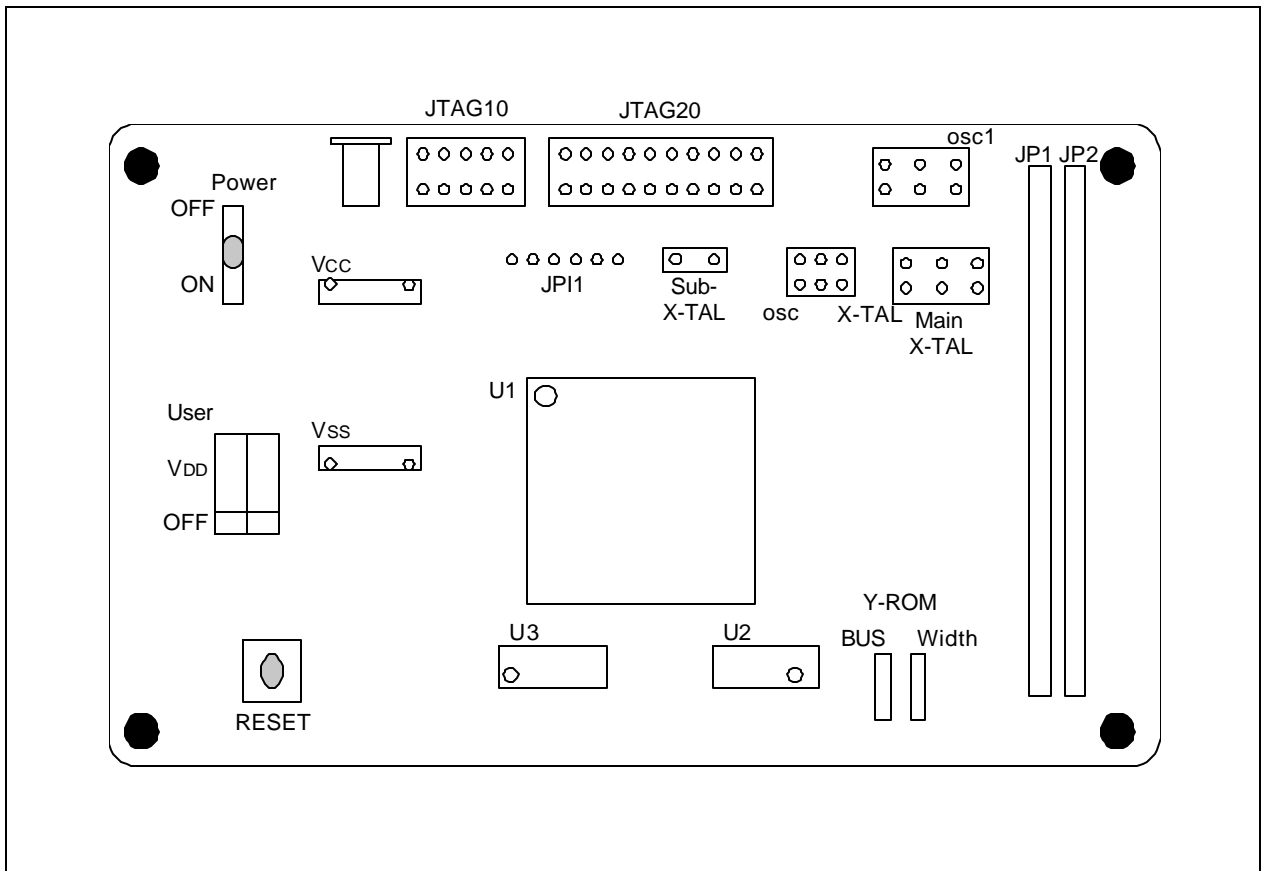


Figure 26-1. Emulation Probe Board Configuration

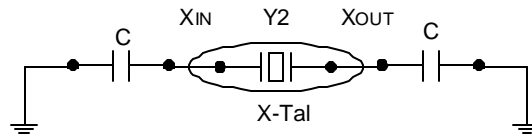
**20-pin/normal Pitch (2.54mm) = JTAG**

Pin No.	Pin Name	Pin No.	Pin Name
1	V <sub>DD</sub>	11	ETDO_TXD
3	ENJRST_UINIT	13	NC
5	ETDI_RXD	15	NC
7	ETMS	17	EOCLK
9	ETCK_MCLK	19	V <sub>DD</sub>
2,4,6,8,10,12,14,16,18,20	GND		

**NOTE:** JTAG (10-pin) is not used.

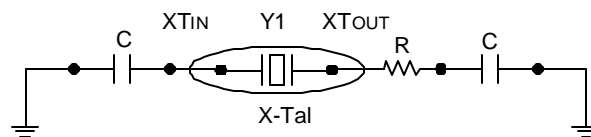
### USE CLOCK SETTING FOR EXTERNAL CLOCK MODE

Proper crystal and capacitors for main clock should be inserted into pin socket on the IE Board as follows;



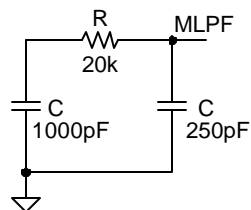
### SUB CLOCK SETTING

For sub-clock mode a crystal, 32.768 kHz and capacitors should be inserted into pin socket on the IE Board as follows;

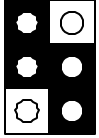
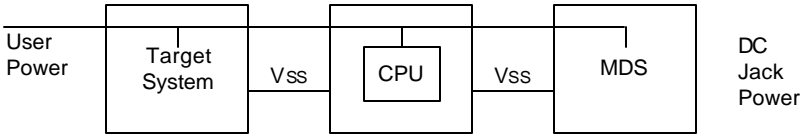
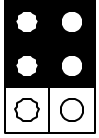
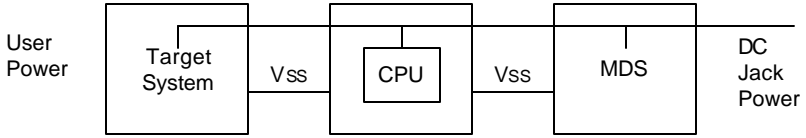


**NOTE:** The value of resistor is 0 k $\Omega$ .

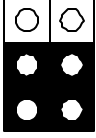
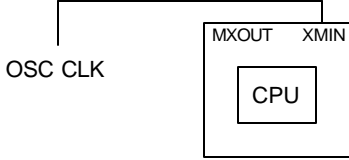
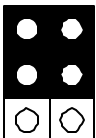
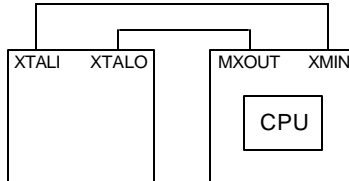
### THE LOWPASS FILTER FOR PLL



**POWER SELECTION**

JP10 State	Description
<p>USER _VDD      POWER _VDD</p>  <p>OFF      OFF</p>	<p><b>Same Power Source from Target System</b></p> 
<p>USER _VDD      POWER _VDD</p>  <p>OFF      OFF</p>	<p><b>Same Power Source from DC Jack</b></p> 

**CLOCK SELECTION**

U1 State	Description
<p>X-TAL</p>  <p>OSC</p>	<p>OSC is used to clock source for evaluation chip</p> 
<p>X-TAL</p>  <p>OSC</p>	<p>X-TAL is used to clock source for evaluation chip</p> 

## JP1, JP2 PIN ASSIGNMENT

JP1	Function	JP1	Function	JP2	Function	JP2	Function
1	VDD	2	GND	1	MP4_1	2	MP4_2
3	NC	4	NC	3	MP4_3	4	MP4_4
5	NC	6	NC	5	MP4_5	6	MP4_6
7	NC	8	AGND	7	MP4_7	8	MP5_0
9	NC	10	NC	9	MP5_1	10	MP5_2
11	NC	12	AGND	11	MP5_3	12	MP5_4
13	MP0_4	14	MP0_3	13	MP5_5	14	MP5_6
15	VDD	16	GND	15	GND	16	VDD
17	MXOUT	18	MXIN	17	MP5_7	18	MP6_0
19	MTEST	20	MXTIN	19	MP6_1	20	MP6_2
21	MXTOUT	22	MRESETB	21	MP6_3	22	MP6_4
23	MP0_2	24	MP0_1	23	MP6_5	24	MP6_6
25	MP0_0	26	MP1_0	25	MP6_7	26	MP7_0
27	MP1_1	28	MP1_2	27	MP7_1	28	MP7_2
29	MP1_3	30	MP1_4	29	MP7_3	30	MP7_4
31	MP1_5	32	MP1_6	31	MP7_5	32	MP7_6
33	MP1_7	34	MP2_0	33	MP7_7	34	MP8_0
35	MP2_1	36	MP2_2	35	MP8_1	36	MP8_2
37	MP2_3	38	MP2_4	37	MP8_3	38	MP8_4
39	MP2_5	40	MP2_6	39	MP9_0	40	MP9_1
41	MP2_7	42	MP3_0	41	MP9_2	42	MP9_3
43	MP3_1	44	MP3_2	43	MP9_4	44	MP9_5
45	MP3_3	46	MP3_4	45	MP9_6	46	MP9_7
47	MP3_5	48	MP3_6	47	MVLC1	48	MP0_7
49	MP3_7	50	MP4_0	49	MP0_6	50	MP0_5

## JP11 PIN ASSIGNMENT

Pin No.	1	2	3	4	5	6
Pin Name	AV <sub>SS2</sub>	DAOUT	ADINP	ADINN	ADGAIN	AV <sub>DD2</sub>

**NOTES**