S3CK318/FK318

CalmRISC 8-Bit CMOS

MICROCONTROLLER

USER'S MANUAL

Revision 1

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.

"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.

S3CK318/FK318 8-Bit CMOS Microcontroller
User's Manual, Revision 1
Publication Number: 21-S3-CK318/FK318-042004

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

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

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

Printed in the Republic of Korea

Preface

The S3CK318/FK318 *Microcontroller User's Manual* is designed for application designers and programmers who are using the S3CK318/FK318 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 5	Hardware Stack
Chapter 2	Address Spaces	Chapter 6	Exceptions
Chapter 3	Register	Chapter 7	Instruction Set
Chapter 4	Memory Map		

Chapter 1, "Product Overview," is a high-level introduction to S3CK318/FK318 with general product descriptions, as well as detailed information about individual pin characteristics and pin circuit types.

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

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

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

Chapter 5, "Hardware Stack," describes the S3CK318/FK318 hardware stack structure in detail.

Chapter 6, "Exception," describes the S3CK318/FK318 exception 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, 6 and 7. Later, you can reference the information in Part I as necessary.

Part II "hardware Descriptions," has detailed information about specific hardware components of the S3CK318/FK318 microcontroller. Also included in Part II are electrical, mechanical. It has 17 chapters:

Chapter 8	Clock Circuit	Chapter 17	LCD Controller/Driver
Chapter 9	RESET and Power-Down	Chapter 18	10-Bit Analog-to-Digital Converter
Chapter 10	I/O Ports	Chapter 19	D/A Converter
Chapter 11	Basic Timer/Watchdog Timer	Chapter 20	Frequency Counter
Chapter 12	Watch Timer	Chapter 21	Electrical Data
Chapter 13	16-bit Timer 0	Chapter 22	Mechanical Data
Chapter 14	8-bit Timer 1	Chapter 23	S3FK318 Flash MCU
Chapter 15	Serial I/O Interface	Chapter 24	Development Tools
Chapter 16	Battery Level Detector		

One order form is included at the back of this manual to facilitate customer order for S3CK318/FK318 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

Chapter 2 Address Spaces

Chapter 3 Register

Chapter 4 Memory Map

Table of Contents (Continued)

Chapter 5 Hardware Stack

Chapter 6 Exceptions

Chapter 7 Instruction Set

Table of Contents (Continued)

Part II — Hardware Descriptions

Chapter 8 Clock Circuit

Chapter 9 RESET and Power-Down

Chapter 10 I/O Ports

Chapter 11 Basic Timer/Watchdog Timer

Chapter 12 Watch Timer

Chapter 13 16-Bit Timer 0

Table of Contents (Continued)

Chapter 14 8-Bit Timer 1

Chapter 15 Serial I/O Interface

Chapter 16 Battery Level Detector

Chapter 17 LCD Controller/Driver

Chapter 18 10-Bit Analog-to-Digital Converter

Table of Contents (Continued)

Chapter 19 D/A Converter

Chapter 20 Frequency Counter

Chapter 21 Electrical Data

Chapter 22 Mechanical Data

Chapter 23 S3FK318 Flash MCU

Table of Contents (Continued)

Chapter 24 Development Tools

List of Figures

List of Figures (Continued)

List of Figures (Continued)

List of Figures (Continued)

Figure Number **Title** **Page Number**

List of Tables

List of Tables (Continued)

List of Programming Tips

Description **Page Number**

Chapter 6: Exceptions

List of Instruction Descriptions

List of Instruction Descriptions (Continued)

NOTIFICATION OF REVISIONS

ORIGINATOR:	Samsung Electronics, LSI Development Group, Ki-Heung, South Korea
PRODUCT NAME:	S3CK318/FK318 8-bit CMOS Microcontroller
DOCUMENT NAME:	S3CK318/FK318 User's Manual, Revision 1
DOCUMENT NUMBER:	21-S3-CK318/FK318-042004
EFFECTIVE DATE:	April, 2004

SUMMARY:

As a result of additional product testing and evaluation, some specifications published in the S3CK318/FK318 User's Manual, Revision 0, have been changed. These changes for S3CK318/FK318 microcontroller, which are described in detail in the *Revision Descriptions* section below, are related to the followings:

— Chapter 21. Electrical Data

DIRECTIONS:

Please note the changes in your copy (copies) of the S3CK318/FK318 User's Manual, Revision 0. Or, simply attach the *Revision Descriptions* of the next page to S3CK318/FK318 User's Manual, Revision 0.

REVISION HISTORY

Revision	Date	Remark
0	November, 2003	Preliminary Spec for internal release only. First edition. Reviewed by Finechips.
1	April, 2004	Second edition. Reviewed by Finechips.

REVISION DESCRIPTIONS

1. OPERATING VOLTAGE (PAGE 21-1)

The Operating voltage at 8MHz X-tal is changed 2.4V to 2.3V.

The Operating voltage at 2MHz X-tal is changed 2.0V to 1.95V.

Table 21-2. D.C. Electrical Characteristics

(T_A = – 25 °C to + 85 °C, V_{DD} = 1.95 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Operating voltage	V_{DD}	fx = 8 MHz	2.3	–	3.6	V
		fx = 2 MHz	1.95	–	3.6	

2. CHARACTERISTICS OF BATTERY LEVEL DETECT CIRCUIT (PAGE 21-8)

Table 21-9. Characteristics of Battery Level Detect Circuit

(T_A = 25 °C)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Operating voltage of BLD	V_{DDBLD}		1.95	–	3.6	V
Voltage of BLD	V_{BLD}	BLDCON.4–.2 = 100b	1.95	2.2	2.45	
		BLDCON.4–.2 = 101b	2.15	2.4	2.65	
		BLDCON.4–.2 = 110b	2.3	2.6	2.9	

1 PRODUCT OVERVIEW

OVERVIEW

The S3CK318/FK318 single-chip CMOS microcontroller is designed for high performance using Samsung's new 8-bit CPU core, CalmRISC.

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

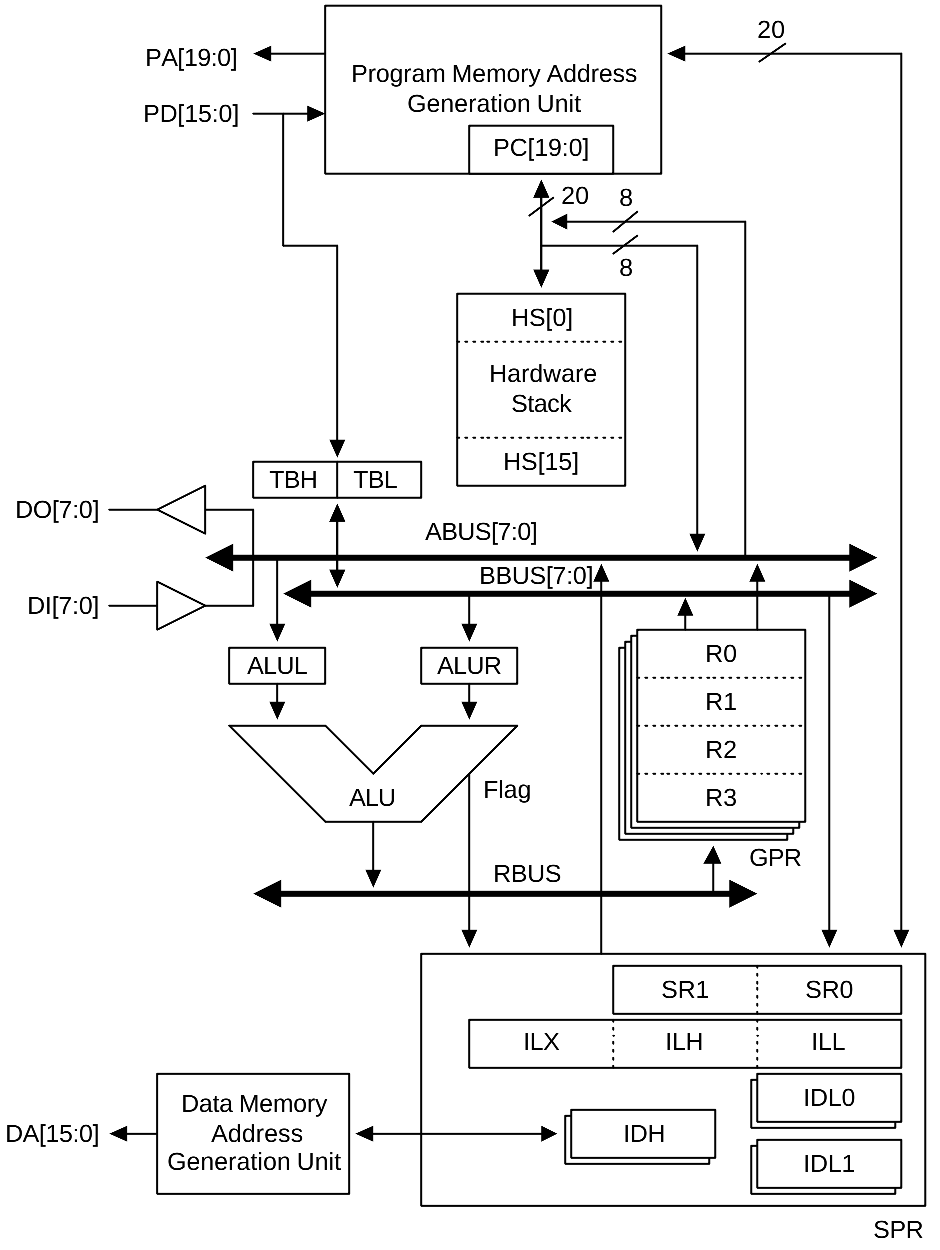


Figure 1-1. Top Block Diagram

The CalmRISC building blocks consist of:

— An 8-bit ALU

— 16 general purpose registers (GPR)

— 11 special purpose registers (SPR)

— 16-level hardware stack

— Program memory address generation unit

— Data memory address generation unit

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

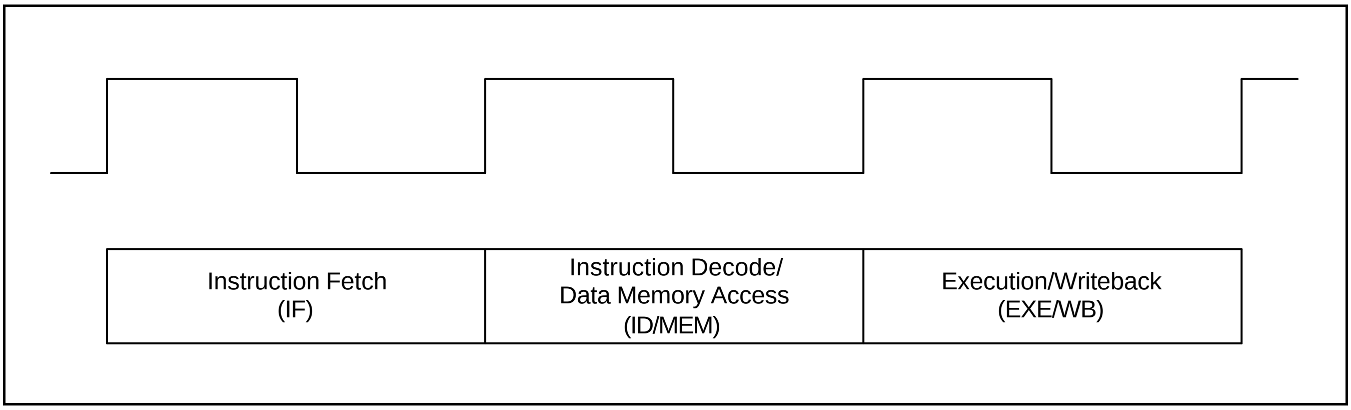


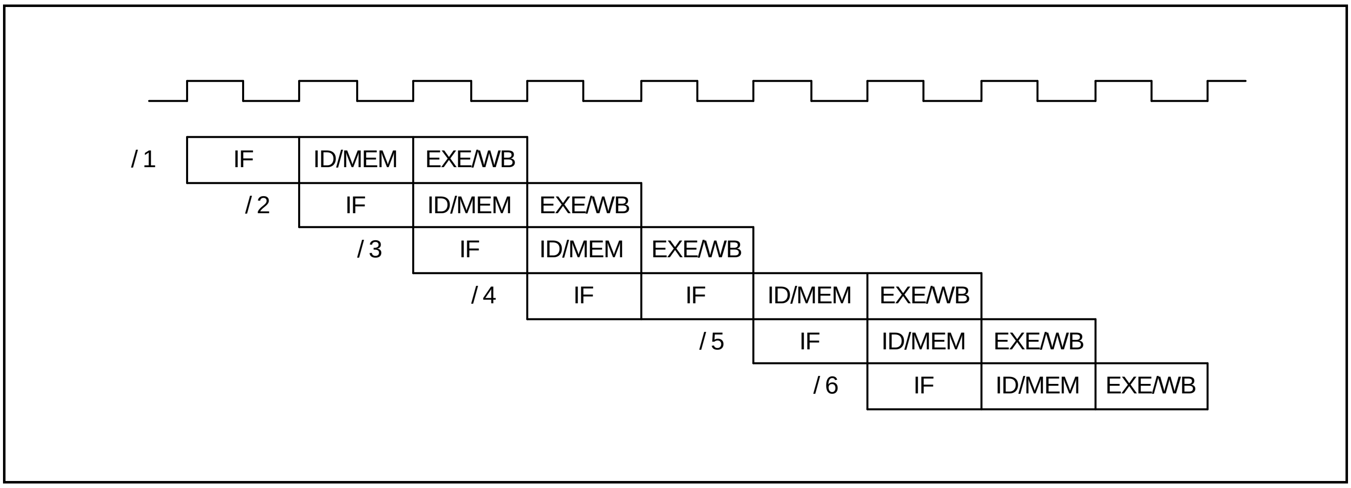
Figure 1-2. CalmRISC Pipeline Diagram

CalmRISC has a 3-stage pipeline as described below:

As can be seen in the pipeline scheme, CalmRISC adopts a register-memory instruction set. In other words, data memory where *R* is a GPR can be one operand of an ALU instruction as shown below:

The first stage (or cycle) is the Instruction fetch stage (IF for short), where the instruction pointed by the program counter, PC[19:0] , is read into the Instruction Register (IR for short). The second stage is the Instruction Decode and Data Memory Access stage (ID/MEM for short), where the fetched instruction (stored in IR) is decoded and data memory access is performed, if necessary. The final stage is the Execute and Write-back stage (EXE/WB), where the required ALU operation is executed and the result is written back into the destination registers.

Since CalmRISC instructions are pipelined, the next instruction fetch is not postponed until the current instruction is completely finished but is performed immediately after completing the current instruction fetch. The pipeline stream of instructions is illustrated in the following diagram.



/ 1	IF	ID/MEM	EXE/WB						
/ 2		IF	ID/MEM	EXE/WB					
/ 3			IF	ID/MEM	EXE/WB				
/ 4				IF	IF	ID/MEM	EXE/WB		
/ 5						IF	ID/MEM	EXE/WB	
/ 6							IF	ID/MEM	EXE/WB

Figure 1-3. CalmRISC Pipeline Stream Diagram

Most CalmRISC instructions are 1-word instructions, while same branch instructions such as long "call" and "jp" instructions are 2-word instructions. In Figure 1-3, the instruction, *I4*, is a long branch instruction, and it takes two clock cycles to fetch the instruction. As indicated in the pipeline stream, the number of clocks per instruction (CPI) is 1 except for long branches, which take 2 clock cycles per instruction.

FEATURES

CPU

- CalmRISC core (8-bit RISC architecture)

Memory

- ROM: 4K-word (8K-byte)
- RAM: 256-byte (excluding LCD data RAM)

Stack

- Size: maximum 16 word-level

36 I/O Pins

- Input only: 2 pins
- I/O: 10 pins
- 24 I/O pins are share with COM/SEG pins

Basic Timer

- Overflow signal makes a system reset
- Watchdog function

16-bit Timer/Counter 0

- Programmable 16-bit timer
- Interval, capture, PWM mode
- Match/capture, overflow interrupt

8-bit Timer/Counter 1

- Programmable 8-bit timer
- Match interrupt generator

Watch Timer

- Real-time and interval time measurement
- Clock generation for LCD
- Four frequency outputs for buzzer sound (0.47/0.94/1.87/3.75 kHz at 75 kHz)

LCD Controller/Driver

- Up to 20 segment pins
- 3, 4, and 8 common selectable
- Internal resistor circuit for LCD bias
- LCD Contrast Control

24-bit Frequency Counter (FC)

- Level = 300 mVpp (Min.)
- AMF input range = 0.5 to 5 MHz or 5 to 30 MHz
- FMF input range = 30 to 130 MHz
- Operating voltage: 2.0 V to 3.6 V
- Counter measurement function (16/32/64 msec)

Battery Level Detector

- Programmable detection voltage (2.2 V, 2.4V, 2.6 V)

8-Bit Serial I/O Interface

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

A/D Converter

- Four analog input channels
- 25 µs conversion speed at 8 MHz
- 10-bit conversion resolution
- Operating voltage: 2.0 V to 3.6 V

D/A Converter

- One analog output channel
- 9-bit conversion resolution (R-2R)
- Operating voltage: 2.0 V to 3.6 V

Oscillation Sources

- Crystal, ceramic, RC for main clock
- Crystal for sub clock
- Main clock frequency: 0.4 – 8 MHz
- Sub clock frequency: 32.8 – 100 kHz
- CPU clock divider circuit (divided by 1, 2, 4, 8, 16, 32, 64, or 128)

Two Power-Down Modes

- Idle (only CPU clock stops)
- Stop (System clock stops)

Interrupts

- 2 Vectors, 14 interrupts

Instruction Execution Times

- 125 ns at 8 MHz (main clock)
- 13.3 µs at 75 kHz (sub clock)

Operating Temperature Range

- - 25 °C to 85 °C

Operating Voltage Range

- 1.95 V to 3.6 V at 2 MHz (2MIPS)
- 2.3 V to 3.6 V at 8 MHz (8MIPS)

Package Type

- 44-QFP-1010B

BLOCK DIAGRAM

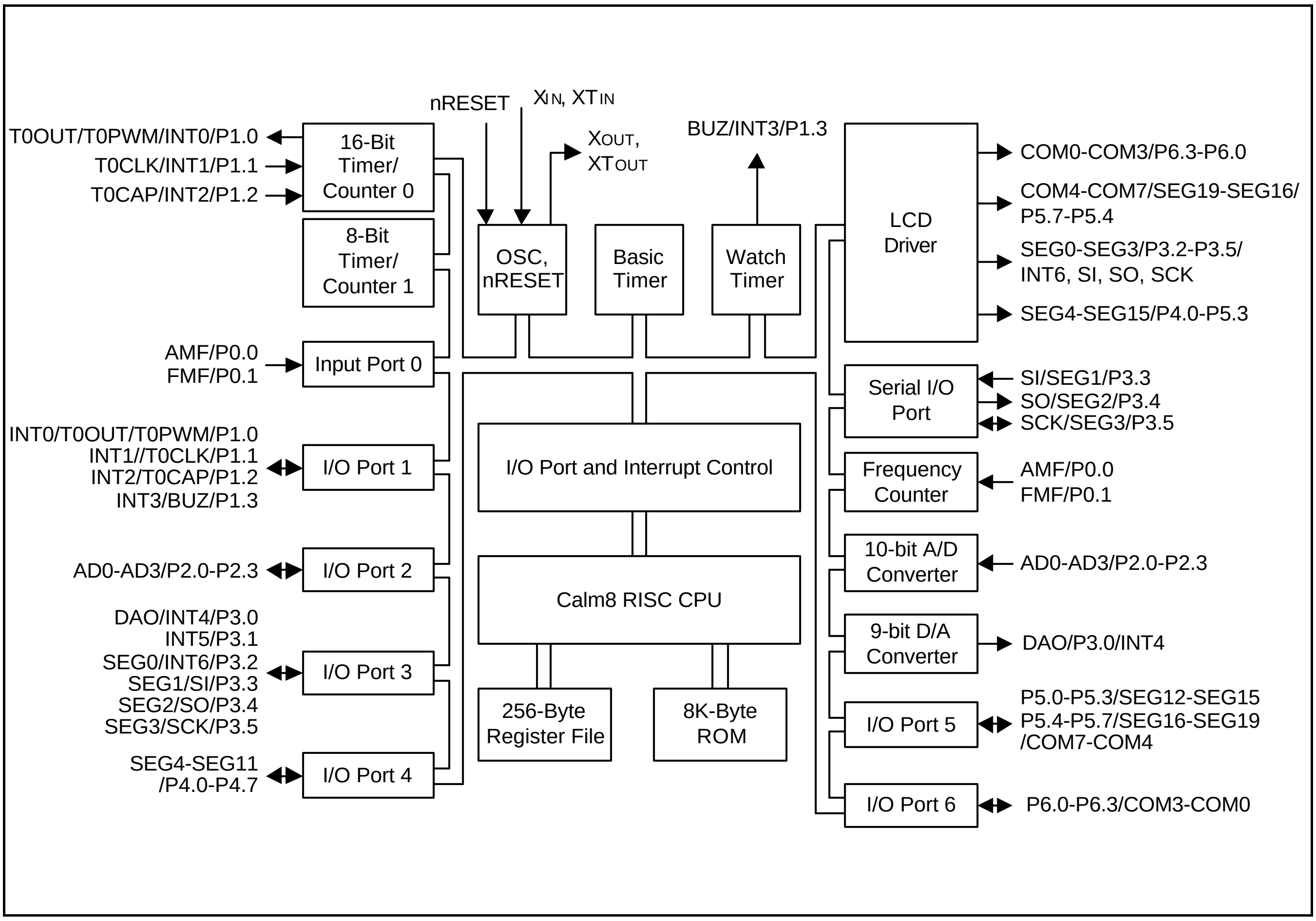


Figure 1-4. Block Diagram

PIN ASSIGNMENT

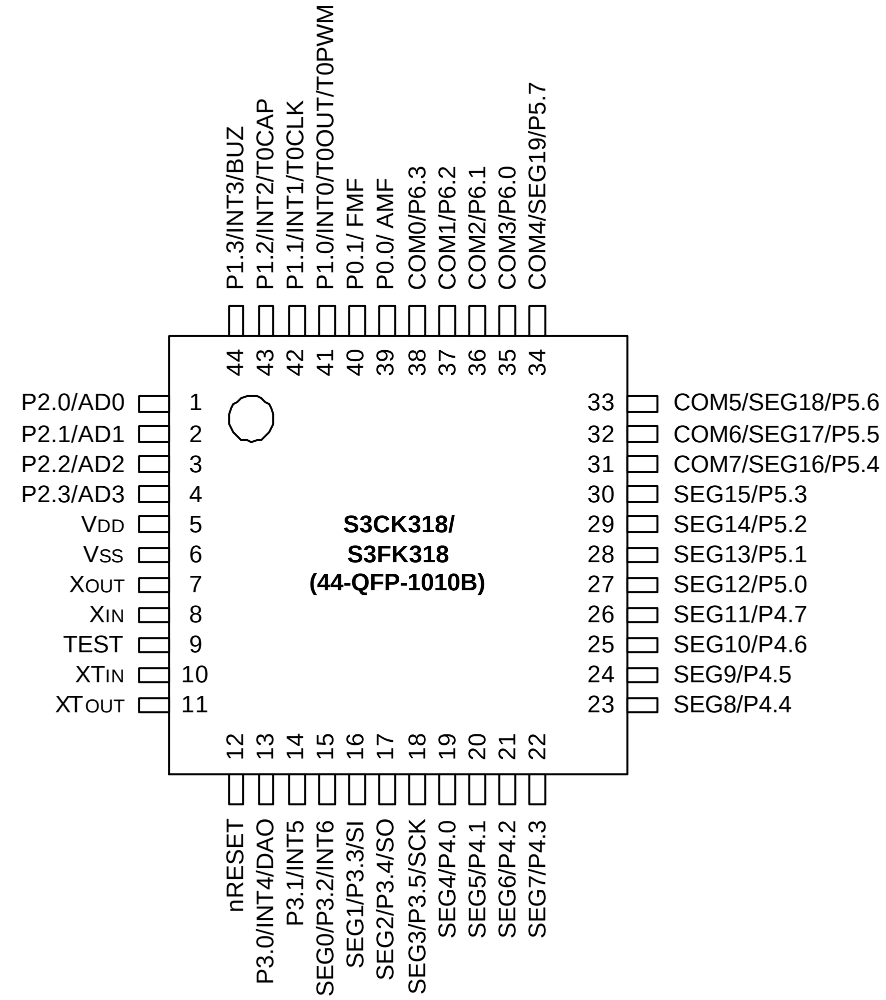


Figure 1-5. Pin Assignment (44-QFP-1010B)

PIN DESCRIPTIONS

Table 1-1. Pin Descriptions

Pin Names	Pin Type	Pin Description	Circuit Type	Pin Numbers	Share Pins
P0.0 P0.1	I	Input port with bit programmable pins; software assignable pull-up resistors.	B-4	39 40	AMF FMF
P1.0 P1.1 P1.2 P1.3	I/O	I/O port with bit programmable pins; Schmitt trigger input or output mode selected by software; Open-drain output mode can be selected by software; software assignable pull-up resistors.	E-4	41 42 43 44	INT0/T0OUT/T0PWM INT1/T0CLK INT2/T0CAP INT3/BUZ
P2.0 P2.1 P2.2 P2.3	I/O	I/O port with bit programmable pins; Schmitt trigger input or output mode selected by software; Open-drain output mode can be selected by software; software assignable pull-up resistors.	F-16A	1 2 3 4	AD0 AD1 AD2 AD3
P3.0 P3.1	I/O	I/O port with bit programmable pins; Schmitt trigger input or output mode selected by software; Open-drain output mode can be selected by software; software assignable pull-up resistors.	E-5 E-4	13 14	INT4/DAO INT5
P3.2 P3.3 P3.4 P3.5			H-32A	15 16 17 18	SEG0/INT6 SEG1/SI SEG2/SO SEG3/SCK
P4.0–P4.7	I/O	I/O port with bit programmable pins; Input or output mode selected by software; Open-drain output mode can be selected by software; software assignable pull-up resistors.	H-32	19–26	SEG4–SEG11
P5.0–P5.3 P5.4–P5.7	I/O	Have the same characteristic as port 4	H-32	27–30 31–34	SEG12–SEG15 SEG16–SEG19/ COM7–COM4
P6.0–P6.3	I/O	Have the same characteristic as port 4	H-32	35–38	COM3–COM0

Table 1-1. Pin Descriptions (Continued)

Pin Names	Pin Type	Pin Description	Circuit Type	Pin Numbers	Share Pins
V_{DD}, V_{SS}	–	Main power supply and Ground	–	5, 6	–
X_{OUT}, X_{IN}	–	Main oscillator pins	–	7, 8	–
XT_{OUT}, XT_{IN}	–	Sub oscillator pins	–	11, 10	–
TEST	I	Test signal input (must be connected to Vss)	–	9	–
nRESET	I	System reset pin	B	12	–
INT0–INT3 INT4 INT5 INT6	I/O	External interrupt input pins	E-4 E-5 E-4 H-32A	41–44 13 14 15	P1.0–P1.3 P3.0/DAO P3.1 P3.2/SEG0
SI, SO, SCK	I/O	Serial I/O interface clock and data signal	H-32A	16–18	P3.3–P3.5/ SEG1–SEG3
AMF FMF	I	External AM/FM frequency inputs	B-4	39 40	P0.0 P0.1
T0OUT/T0PWM	I/O	Timer 0 output and PWM output	E-4	41	P1.0/INT0
T0CLK	I/O	Timer 0 External clock input		42	P1.1/INT1
T0CAP	I/O	Timer 0 Capture input		43	P1.2/INT2
BUZ	I/O	Four frequency output for buzzer sound with main clock or sub clock		44	P1.3/INT3
AD0–AD3	I/O	A/D converter analog input channels	F-16A	1–4	P2.0–P2.3
DAO	I/O	D/A converter analog output channel	E-5	13	P3.0/INT4
COM0–COM3 COM4–COM7	I/O	LCD common signal output	H-32	38–35 34–31	P6.3–P6.0 P5.7–P5.4/ SEG19–SEG16
SEG0 SEG1–SEG3 SEG4–SEG11 SEG12–SEG19	I/O	LCD segment signal output	H-32A H-32A H-32 H-32	15 16–18 19–26 27–34	P3.2/INT6 P3.3–3.5/ SI,SO,SCK P4.0–P4.7 P5.0–P5.7

PIN CIRCUITS

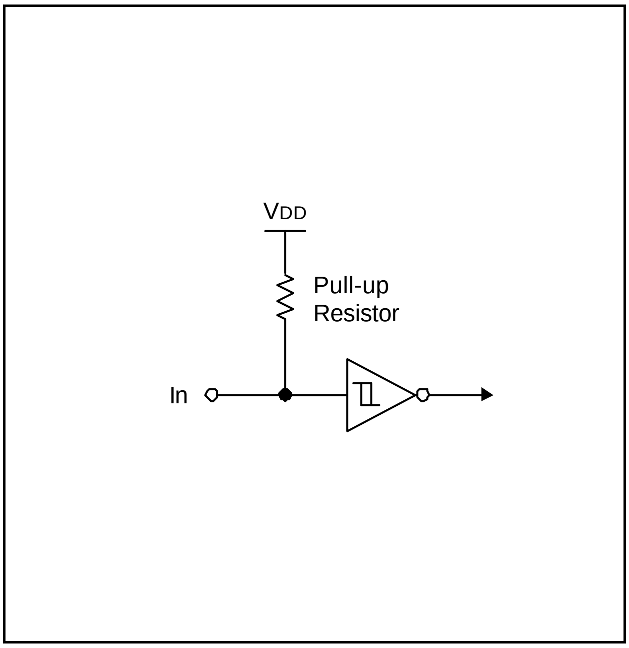


Figure 1-6. Pin Circuit Type B (nRESET)

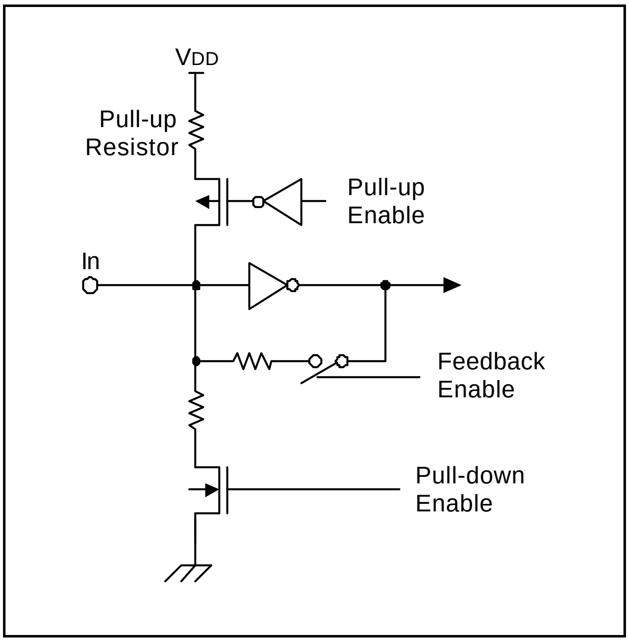


Figure 1-7. Pin Circuit Type B-4 (P0)

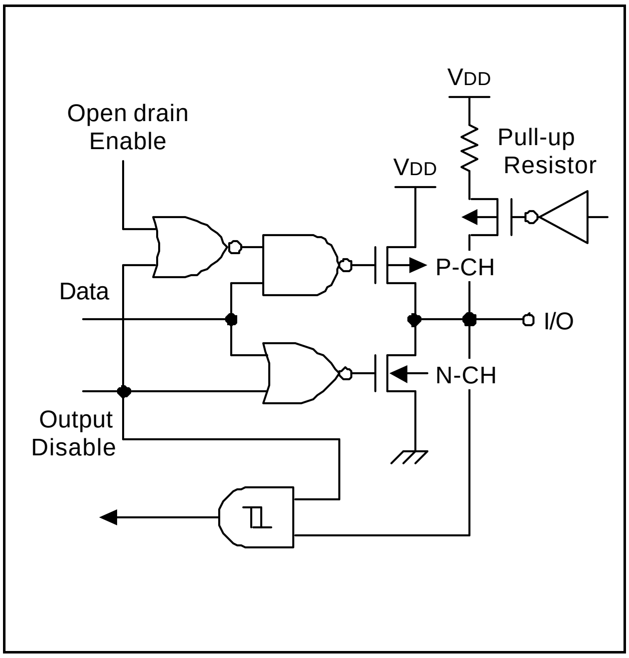


Figure 1-8. Pin Circuit Type E-4 (P1, P3.1)

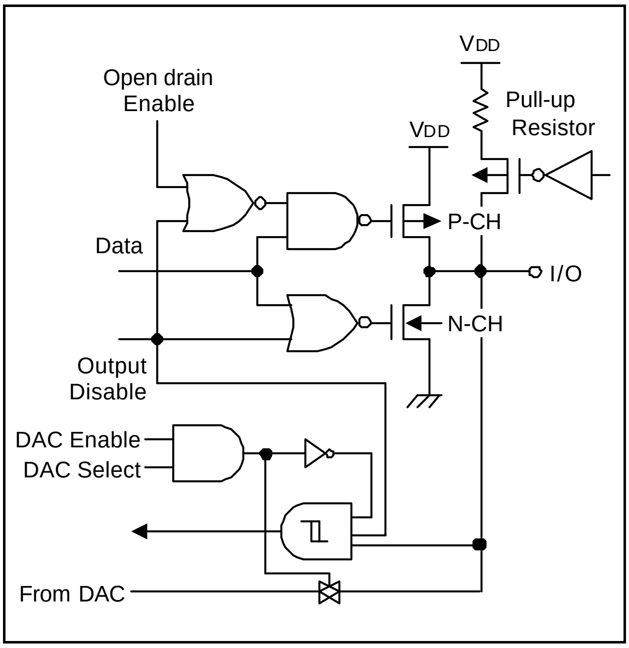


Figure 1-9. Pin Circuit Type E-5 (P3.0)

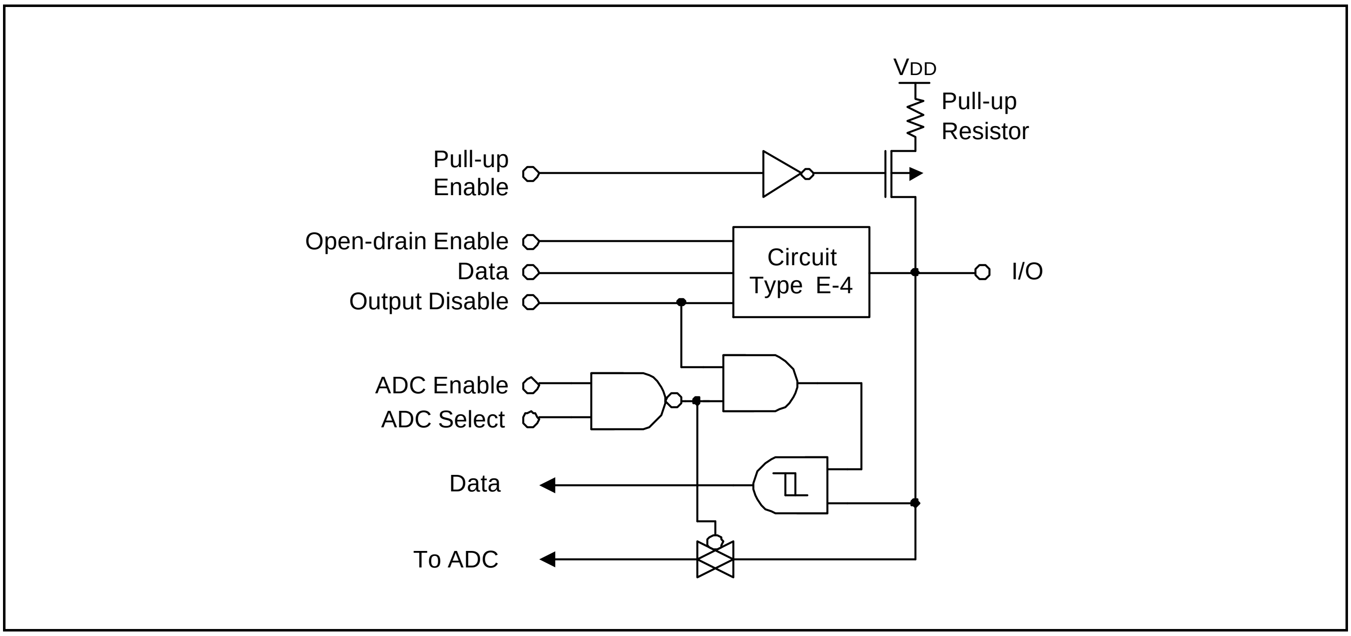


Figure 1-10. Pin Circuit Type F-16A (P2)

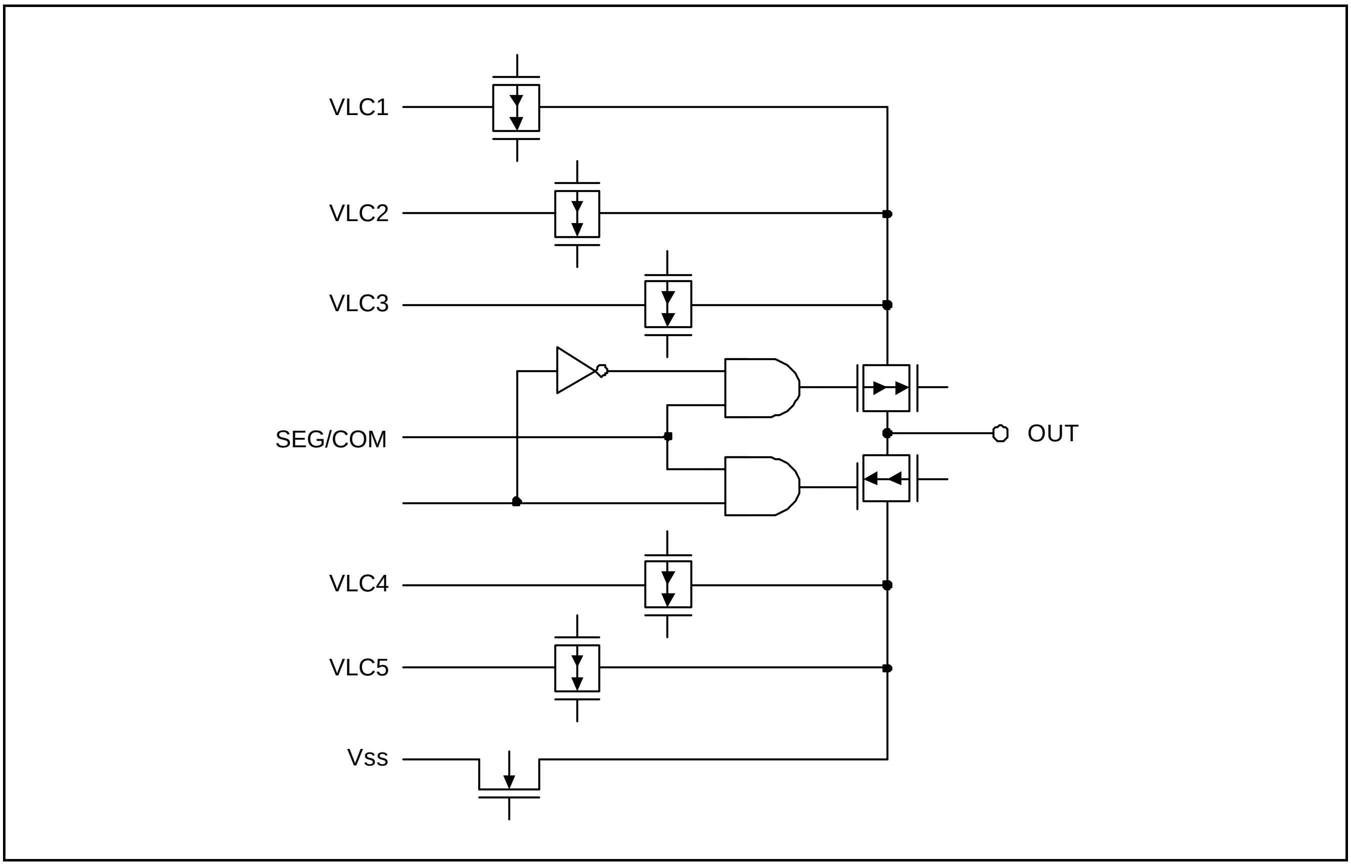


Figure 1-11. Pin Circuit Type H-23

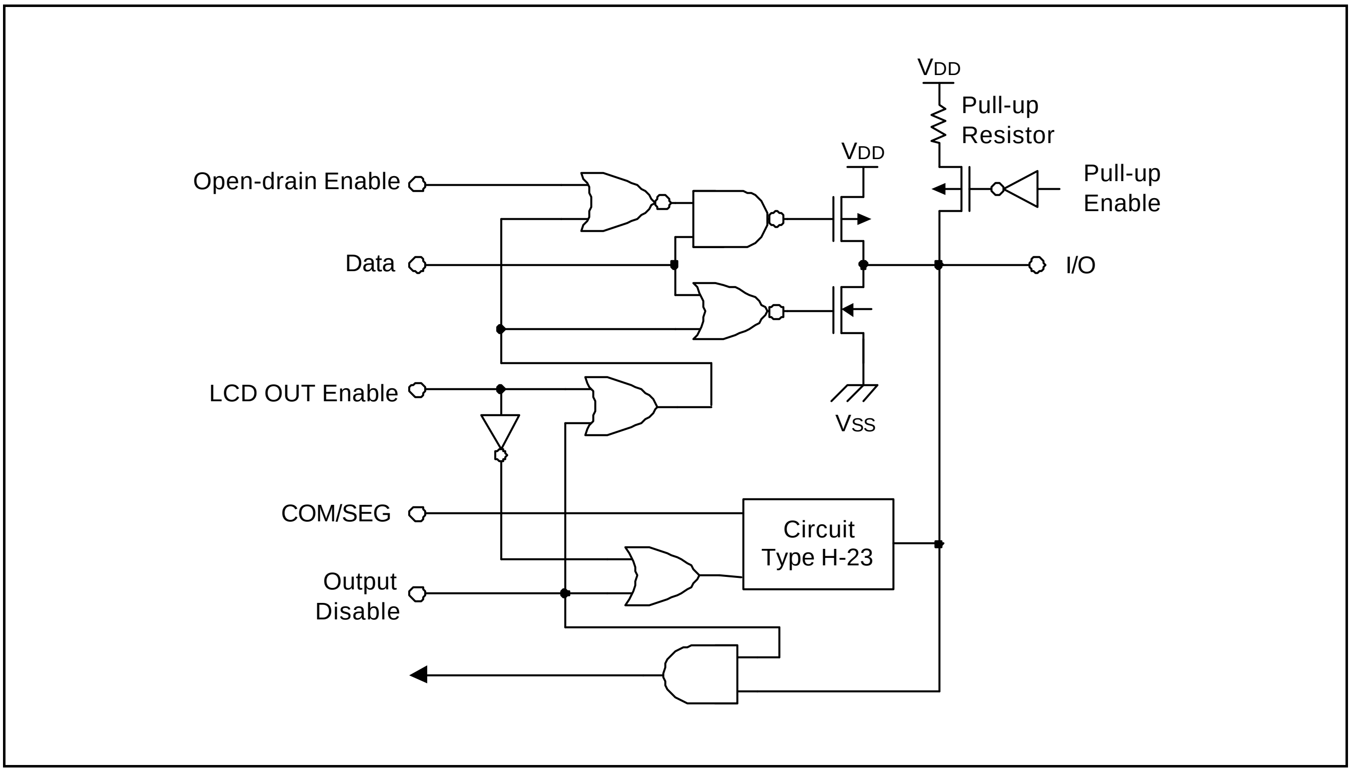


Figure 1-12. Pin Circuit Type H-32 (P4, P5, and P6)

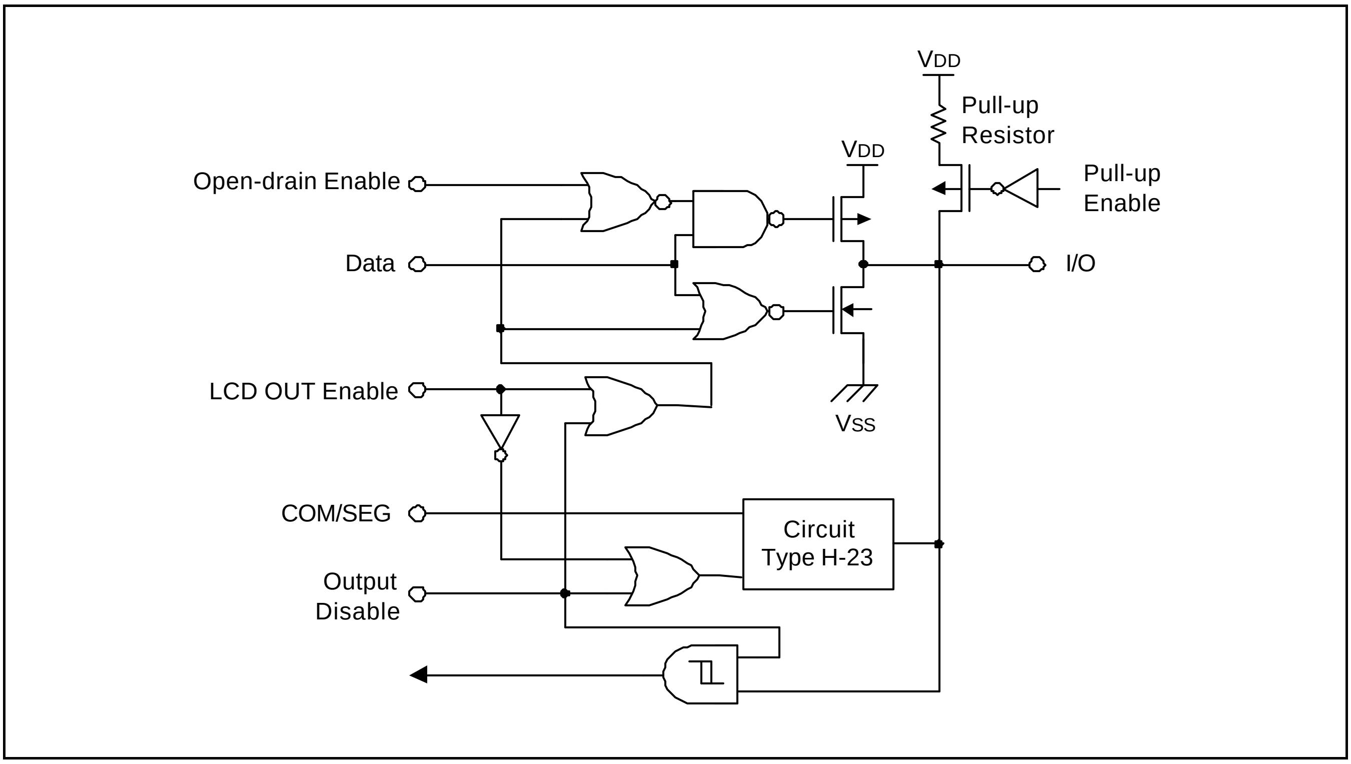


Figure 1-13. Pin Circuit Type H-32A (P3.2–P3.5)

NOTES

2 ADDRESS SPACES

OVERVIEW

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

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

PROGRAM MEMORY (ROM)

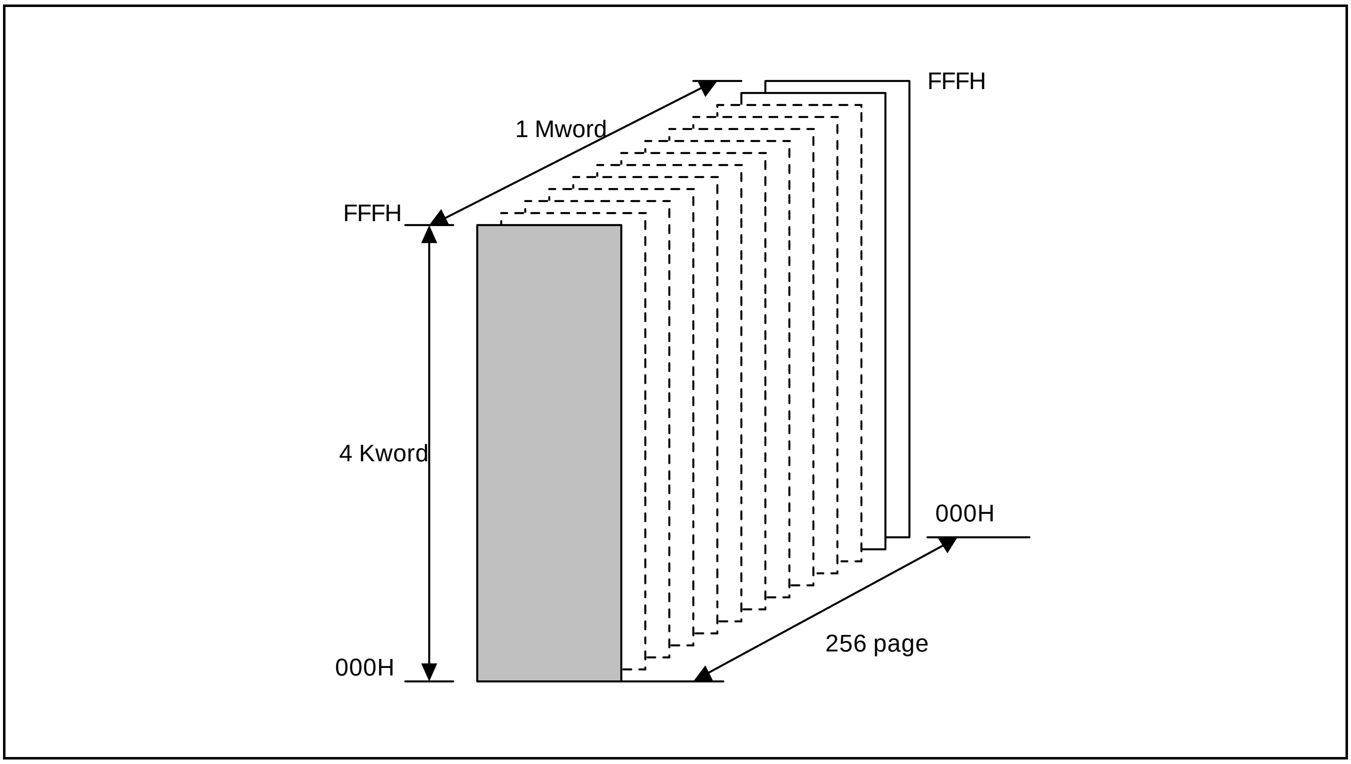


Figure 2-1. Program Memory Organization

For example, if PC[19:0] = 5F79AH, the page index pointed to by PC is 5FH and the offset in the page is 79AH. If the current PC[19:0] = 5EFFFH and the instruction pointed to by the current PC, i.e., the instruction at the address 5EFFFH is *not* a branch instruction, the next PC becomes 5E000H, *not* 5F000H. In other words, the instruction sequence wraps around at the page boundary, unless the instruction at the boundary (in the above example, at 5EFFFH) is a long branch instruction. The only way to change the program page is by long branches (LCALL, LLNK, and LJP), where the absolute branch target address is specified. For example, if the current PC[19:0] = 047ACH (the page index is 04H and the offset is 7ACH) and the instruction pointed to by the current PC, i.e., the instruction at the address 047ACH, is "LJP A507FH" (jump to the program address A507FH), then the next PC[19:0] = A507FH, which means that the page and the offset are changed to A5H and 07FH, respectively. On the other hand, the short branch instructions cannot change the page indices.

Suppose the current PC is 6FFFEH and its instruction is "JR 5H" (jump to the program address PC + 5H). Then the next instruction address is 6F003H, *not* 70003H. In other words, the branch target address calculation also wraps around with respect to a page boundary. This situation is illustrated below:

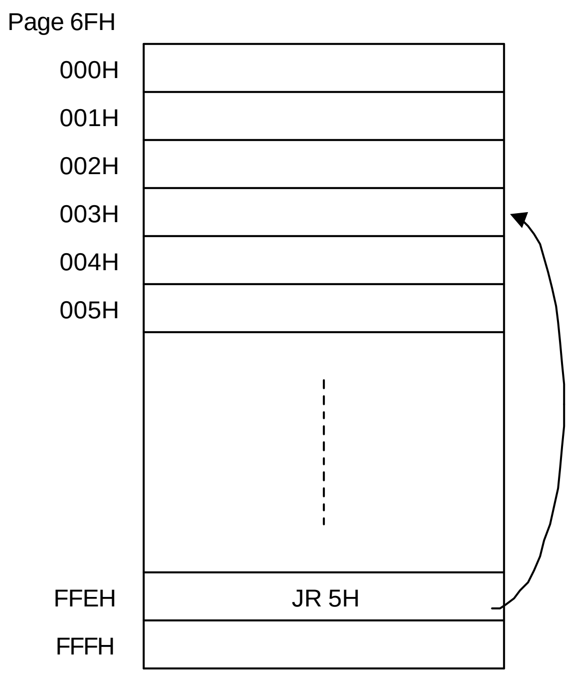
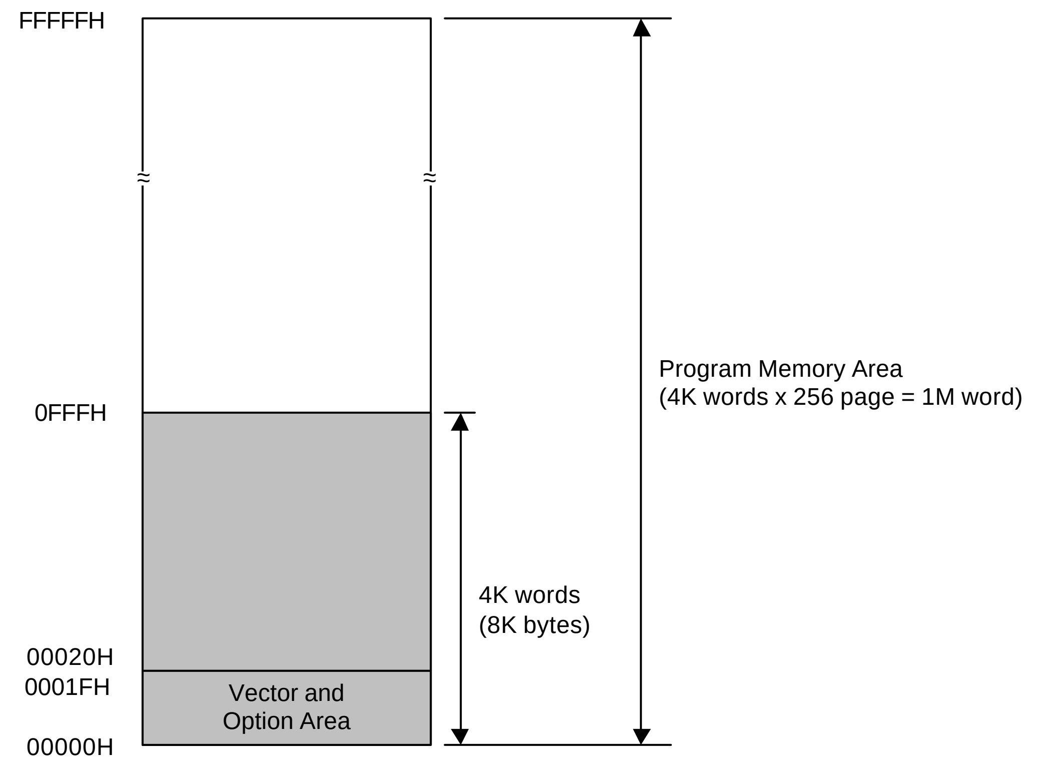


Figure 2-2. Relative Jump Around Page Boundary

Programmers do not have to manually calculate the offset and insert extra instructions for a jump instruction across page boundaries. The compiler and the assembler for CalmRISC are in charge of producing appropriate codes for it.



NOTE: For S3CK318, total size of program memory area is 4K words (8K bytes).

Figure 2-3. Program Memory Layout

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

The Program memory size of S3CK318 is 4K word (8K byte), so from the address 00020H to 00FFFH are the program memory area.

ROM CODE OPTION (RCOD_OPT)

Just after power on, the ROM data located at 0001EH and 0001FH is used as the ROM code option.
S3CK318 has ROM code options like the Reset value of Basic timer and Watchdog timer enable.

For example, if you program as below:

RCOD_OPT1EH, 0x0000

RCOD_OPT1FH, 0xbfff

— fxx/32 is used as Reset value of basic timer (by bit.14, 13, 12)

— Watchdog timer is enabled (by bit.11)

If you don't program any values in these option areas, then the default value is "1".

In these cases, the address 0001EH would be the value of "FFFFH".

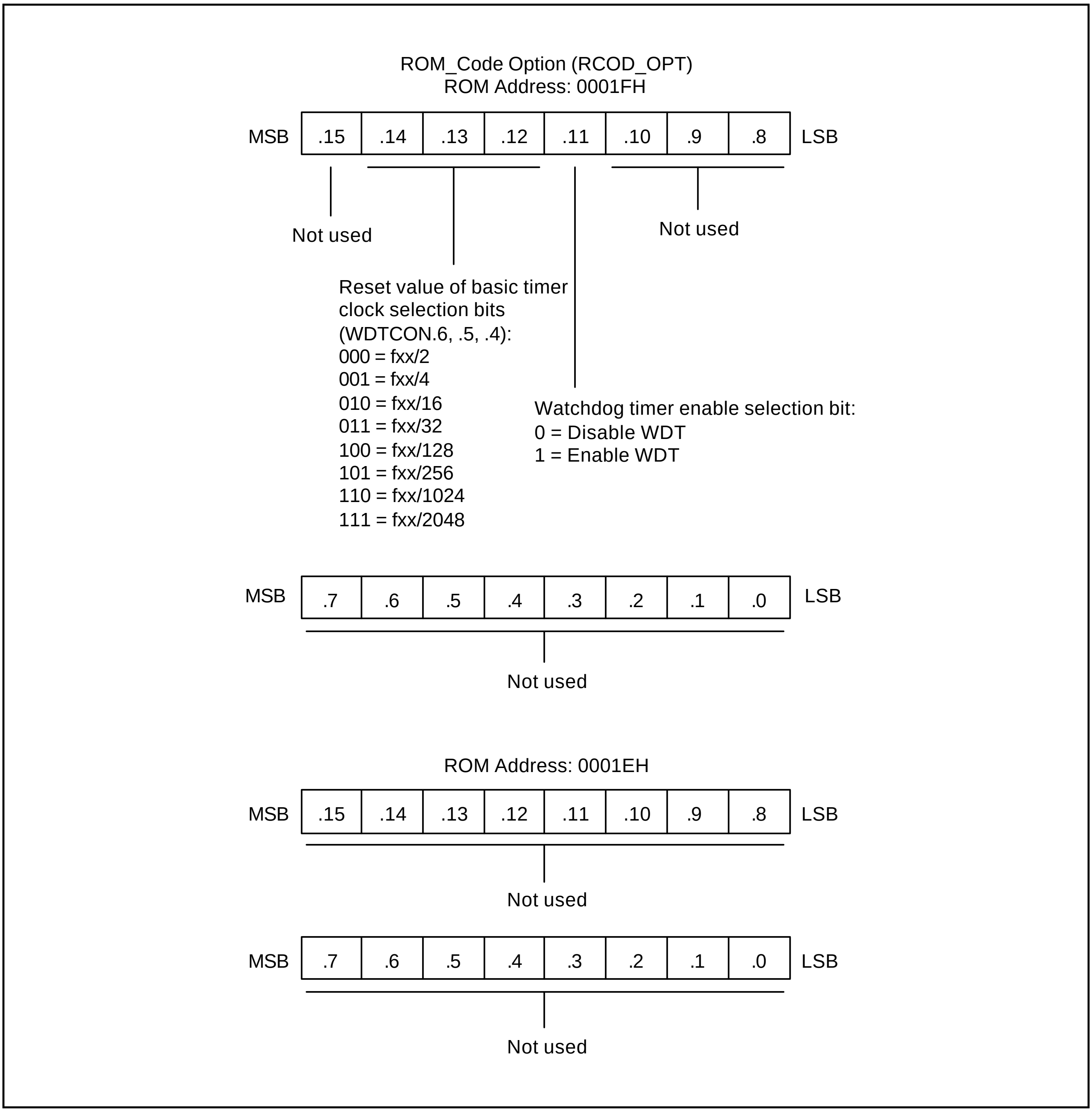


Figure 2-4. ROM Code Option (RCOD_OPT)

DATA MEMORY ORGANIZATION

The total data memory address space is 64K bytes, addressed by *DA[15:0],* and divided into 256 pages, Each page consists of 256 bytes as shown below.

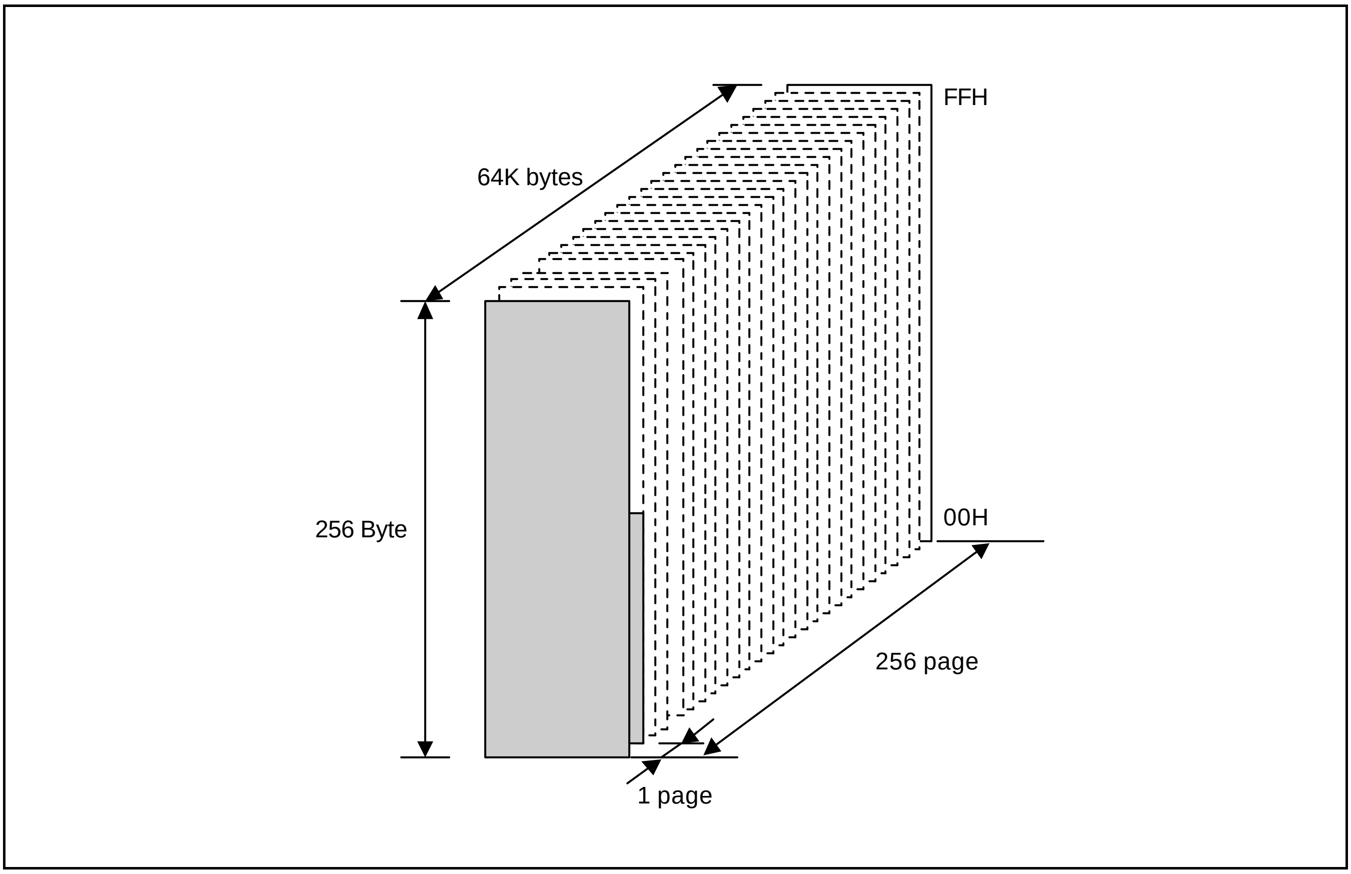


Figure 2-5. Data Memory Map of CalmRISC8

The data memory page is indexed by SPR and IDH. In data memory index addressing mode, 16-bit data memory address is composed of two 8-bit SPRs, IDH[7:0] and IDL0[7:0] (or IDH[7:0] and IDL1[7:0]). IDH[7:0] points to a page index, and IDL0[7:0] (or IDL1[7:0]) represents the page offset. In data memory direct addressing mode, an 8-bit direct address, adr[7:0], specifies the offset of the page pointed to by IDH[7:0] (See the details for direct addressing mode in the instruction sections). Unlike the program memory organization, data memory address does *not* wrap around. In other words, data memory index addressing with modification performs an addition or a subtraction operation on the whole 16-bit address of IDH[7:0] and IDL0[7:0] (or IDL1[7:0]) and updates IDH[7:0] and IDL0[7:0] (or IDL1[7:0]) accordingly. Suppose IDH[7:0] is 0FH and IDL0[7:0] is FCH and the modification on the index registers, IDH[7:0] and IDL0[7:0], is increment by 5H, then, after the modification (i.e., 0FFCH + 5 = 1001H), IDH[7:0] and IDL0[7:0] become 10H and 01H, respectively.

The S3CK318 has 256 bytes of data register address from 0080H to 017FH.
The area from 0000H to 007FH is for peripheral control, and LCD RAM area is from 0180H to 0193H.

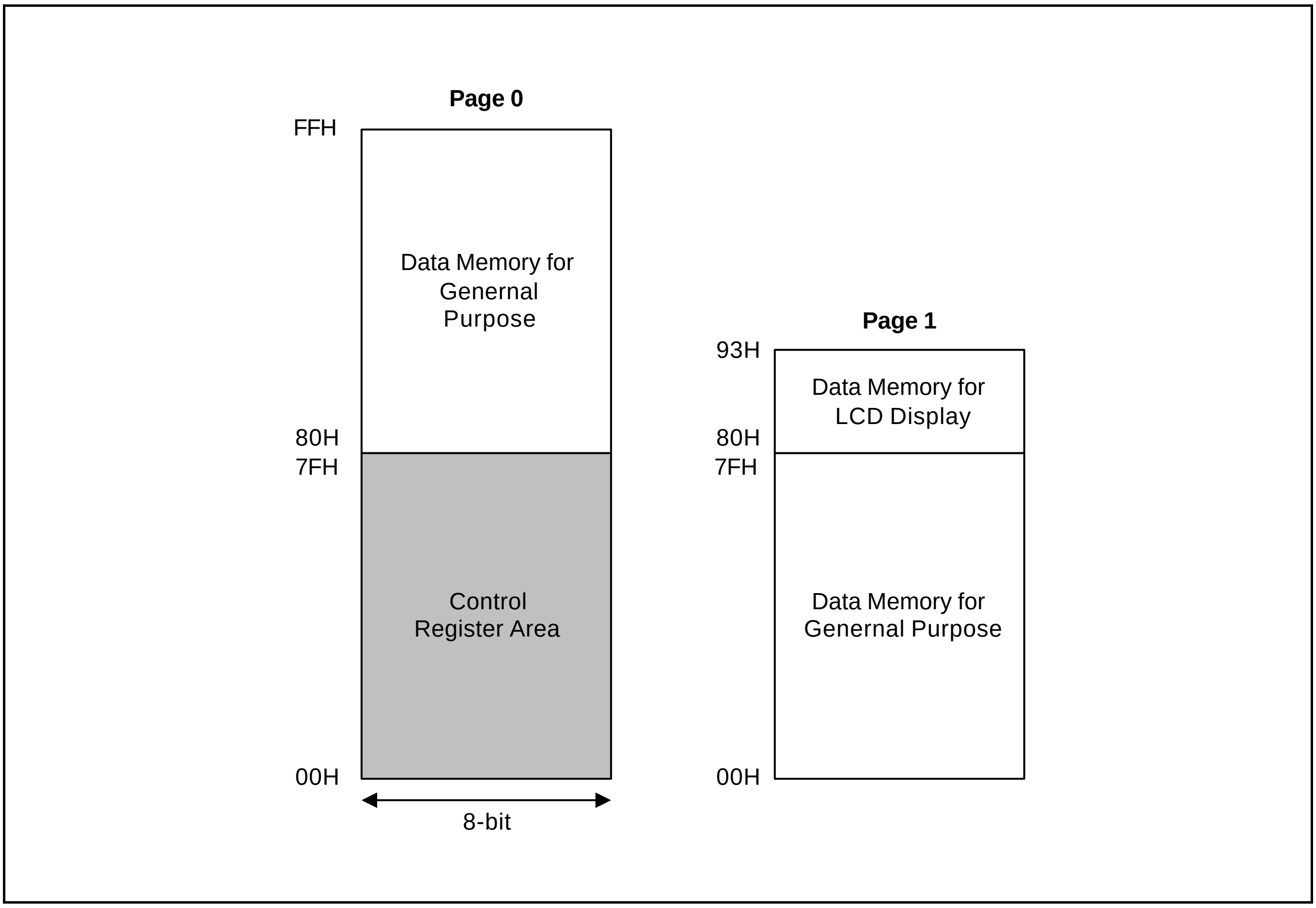

Page 0
FFH
Data Memory for Genernal Purpose
80H
7FH
Control Register Area
00H
8-bit
Page 1
93H
Data Memory for LCD Display
80H
7FH
Data Memory for Genernal Purpose
00H

Figure 2-6. Data Memory Map of S3CK318

NOTES

3 REGISTERS

OVERVIEW

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

Table 3-1. General and Special Purpose Registers

Registers		Mnemonics	Description	Reset Value
General Purpose Registers (GPR)		*R0*	General Register 0	Unknown
		R1	General Register 1	Unknown
		R2	General Register 2	Unknown
		R3	General Register 3	Unknown
Special Purpose Registers (SPR)	Group 0 (SPR0)	IDL0	Lower Byte of Index Register 0	Unknown
		IDL1	Lower Byte of Index Register 1	Unknown
		IDH	Higher Byte of Index Register	Unknown
		SR0	Status Register 0	00H
	Group 1 (SPR1)	ILX	Instruction Pointer Link Register for Extended Byte	Unknown
		ILH	Instruction Pointer Link Register for Higher Byte	Unknown
		ILL	Instruction Pointer Link Register for Lower Byte	Unknown
		SR1	Status Register 1	Unknown

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

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

INDEX REGISTERS: IDH, IDL0 AND IDL1

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

LINK REGISTERS: ILX, ILH AND ILL

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

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

STATUS REGISTER 0: SR0

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

Table 3-2. Status Register 0 configuration

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

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

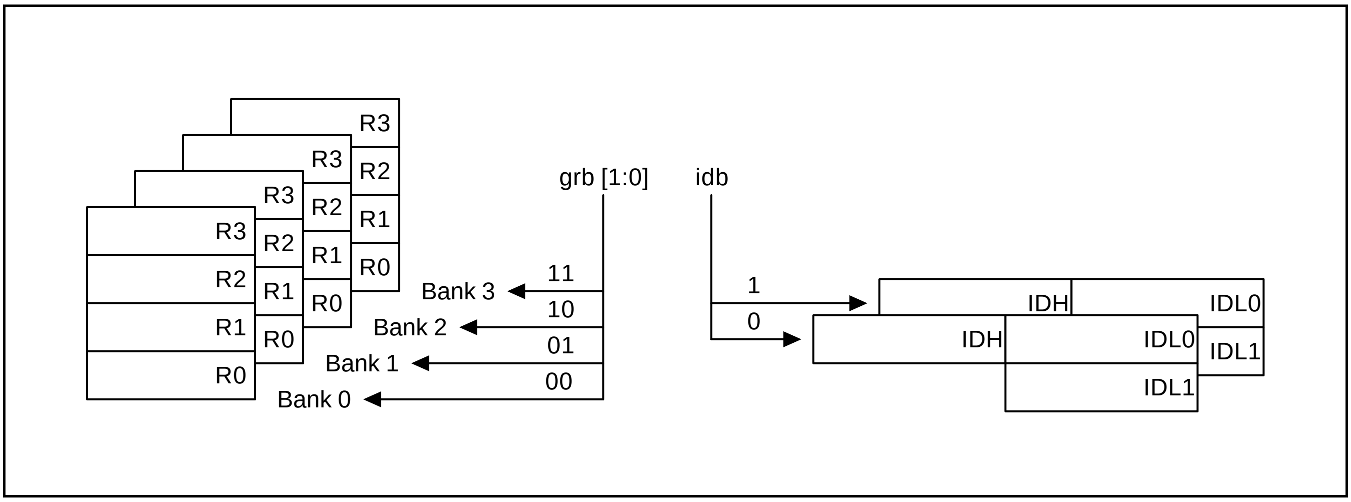


Figure 3-1. Bank Selection by Setting of GRB Bits and IDB Bit

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

STATUS REGISTER 1: SR1

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

Table 3-3. Status Register 1: SR1

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

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

NOTE

When an interrupt occurs, SR0 and SR1 are not saved by hardware, so SR0, and SR1 register values must be saved by software.

4 MEMORY MAP

OVERVIEW

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

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

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

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

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

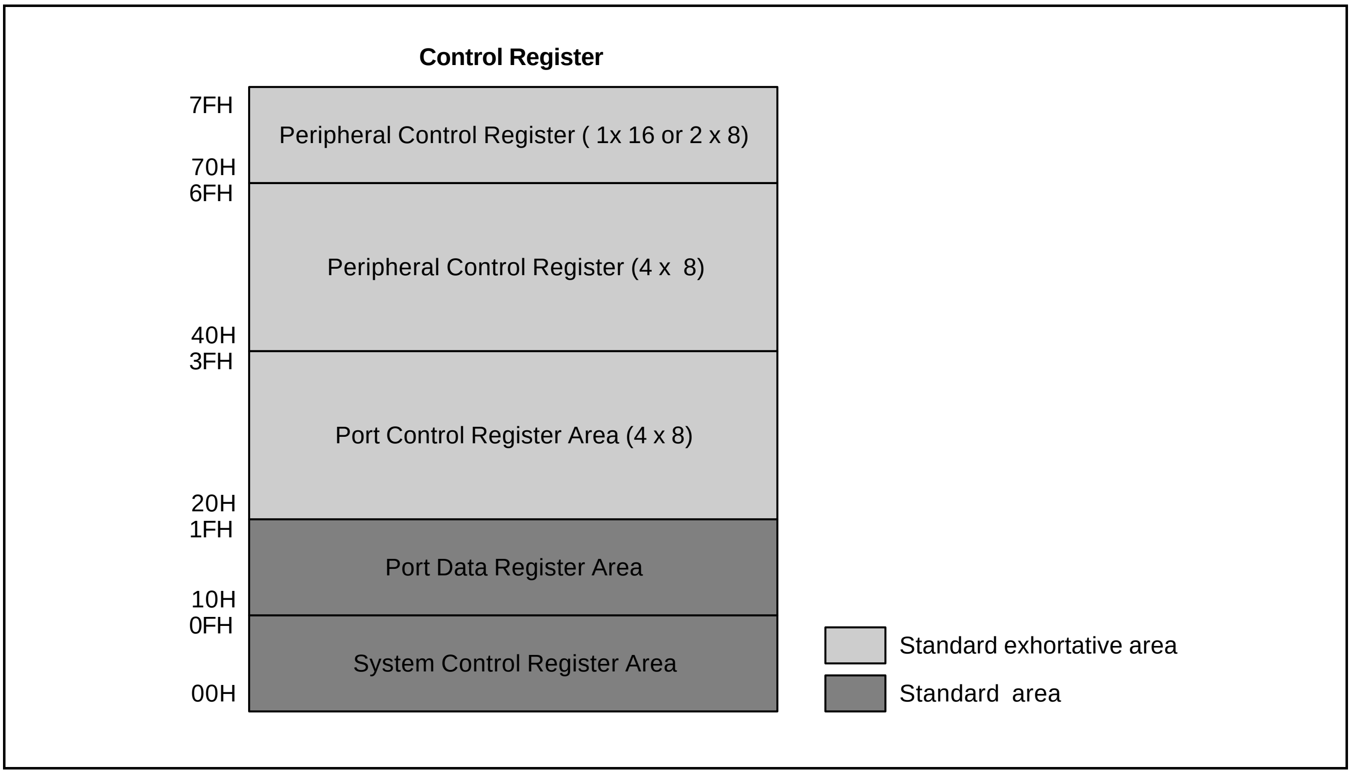

Control Register
7FH
70H
6FH
40H
3FH
20H
1FH
10H
0FH
00H
Peripheral Control Register (1x 16 or 2 x 8)
Peripheral Control Register (4 x 8)
Port Control Register Area (4 x 8)
Port Data Register Area
System Control Register Area
Standard exhortative area
Standard area

Figure 4-1. Memory Map Area

Table 4-1. Registers

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
Locations 17H–1FH are not mapped.					
Port 6 data register	P6	22	16H	00H	R/W
Port 5 data register	P5	21	15H	00H	R/W
Port 4 data register	P4	20	14H	00H	R/W
Port 3 data register	P3	19	13H	00H	R/W
Port 2 data register	P2	18	12H	00H	R/W
Port 1 data register	P1	17	11H	00H	R/W
Port 0 data register	P0	16	10H	00H	R
Locations 0EH–0FH are not mapped.					
Watchdog timer control register	WDTCON	13	0DH	X0H	R/W
Basic timer counter	BTCNT	12	0CH	00H	R
Interrupt ID register 1	IIR1	11	0BH	–	R/W
Interrupt priority register 1	IPR1	10	0AH	–	R/W
Interrupt mask register 1	IMR1	9	09H	00H	R/W
Interrupt request register 1	IRQ1	8	08H	–	R
Interrupt ID register 0	IIR0	7	07H	–	R/W
Interrupt priority register 0	IPR0	6	06H	–	R/W
Interrupt mask register 0	IMR0	5	05H	00H	R/W
Interrupt request register 0	IRQ0	4	04H	–	R
Oscillator control register	OSCCON	3	03H	00H	R/W
Power control register	PCON	2	02H	04H	R/W
Locations 00H–01H are not mapped.					

NOTES:

1. All the unused and unmapped registers and bits read "0".
2. '–' means undefined.
3. If you want to clear the bit of IRQx, then write the number that you want to clear to IIRx. For example, when clear IRQ0.4 then LD Rx, #04H and LD IIR0, Rx.

Table 4-1. Registers (continued)

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
Locations 4BH–4FH are not mapped.					
Timer 1 counter	T1CNT	74	4AH	00H	R
Timer 1 data register	T1DATA	73	49H	FFH	R/W
Timer 1 control register	T1CON	72	48H	00H	R/W
Locations 45H–47H are not mapped.					
Timer 0 counter (low byte)	T0CNTL	68	44H	00H	R
Timer 0 counter (high byte)	T0CNTH	67	43H	00H	R
Timer 0 data register (low byte)	T0DATAL	66	42H	FFH	R/W
Timer 0 data register (high byte)	T0DATAH	65	41H	FFH	R/W
Timer 0 control register	T0CON	64	40H	00H	R/W
Locations 37H–3FH are not mapped					
Port 6 control register	P6CON	54	36H	00H	R/W
Port 5 control register (low byte)	P5CONL	53	35H	00H	R/W
Port 5 control register (high byte)	P5CONH	52	34H	00H	R/W
Locations 32H–33H are not mapped					
Port 4 control register (low byte)	P4CONL	49	31H	00H	R/W
Port 4 control register (high byte)	P4CONH	48	30H	00H	R/W
Location 2FH is not mapped.					
Port 3 control register C	P3CONC	45	2EH	00H	R/W
Port 3 control register B	P3CONB	45	2DH	00H	R/W
Port 3 control register A	P3CONA	44	2CH	00H	R/W
Locations 2AH–2BH are not mapped					
Port 2 pull-up control register	P2PUR	41	29H	00H	R/W
Port 2 control register	P2CON	40	28H	00H	R/W
Location 27H is not mapped					
Port 1 interrupt edge selection register	P1EDGE	38	26H	00H	R/W
Port 1 pull-up control register	P1PUR	37	25H	00H	R/W
Port 1 control register	P1CON	36	24H	00H	R/W
Locations 21H–23H are not mapped					
Port 0 pull-up control register	P0PUR	32	20H	00H	R/W

NOTES:

1. All unused and unmapped registers and bits read "0".
2. '–' means undefined.

Table 4-1. Registers (continued)

Register Name	Mnemonic	Decimal	Hex	Reset	R/W
Locations 7DH–7FH are not mapped					
Frequency Counter 2 (high byte)	FCNT2	124	7CH	00H	R
Frequency Counter 1 (mid byte)	FCNT1	123	7BH	00H	R
Frequency Counter 0 (low byte)	FCNT0	122	7AH	00H	R
Frequency counter mode register	FCMOD	121	79H	00H	R/W
Frequency counter control register	FCCON	120	78H	00H	R/W
Location 77H is not mapped					
D/A converter data register (low byte)	DADATAL	118	76H	00H	R/W
D/A converter data register (high byte)	DADATAH	117	75H	00H	R/W
D/A converter control register	DACON	116	74H	00H	R/W
Locations 72H–73H are not mapped					
Battery level detector control register	BLDCON	113	71H	00H	R/W
Watch timer control register	WTCON	112	70H	00H	R/W
Locations 64H–6FH are not mapped					
LCD Port Control Register 2	LPOT2	99	63H	00H	R/W
LCD Port Control Register 1	LPOT1	98	62H	00H	R/W
LCD Port Control Register 0	LPOT0	97	61H	00H	R/W
LCD mode control register	LMOD	96	60H	00H	R/W
Location 5FH is not mapped					
A/D Converter data register (low byte)	ADDATAL	94	5EH	–	R
A/D Converter data register (high byte)	ADDATAH	93	5DH	–	R
A/D Converter control register	ADCON	92	5CH	00H	R/W
Location 5BH is not mapped					
Serial I/O data register	SIODATA	90	5AH	00H	R/W
Serial I/O pre-scale register	SIOPS	89	59H	00H	R/W
Serial I/O control register	SIOCON	88	58H	00H	R/W
Locations 50H–57H are not mapped					

NOTES

1. All unused and unmapped registers and bits read "0".
2. "–" means undefined.

5 HARDWARE STACK

OVERVIEW

The hardware stack in CalmRISC has two usages:

- To save and restore the return PC[19:0] on LCALL, CALLS, RET, and IRET instructions.
- Temporary storage space for registers on PUSH and POP instructions.

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

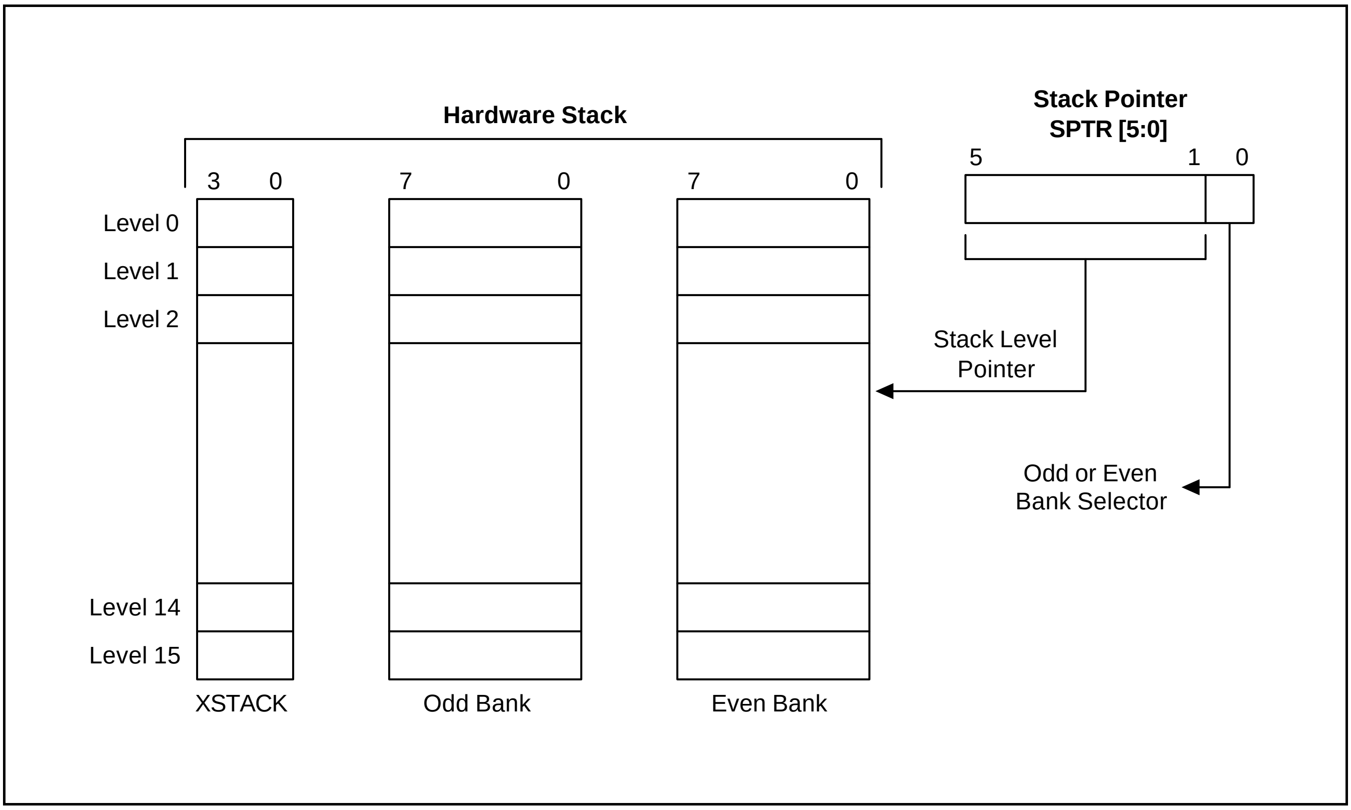


Figure 5-1. Hardware Stack

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

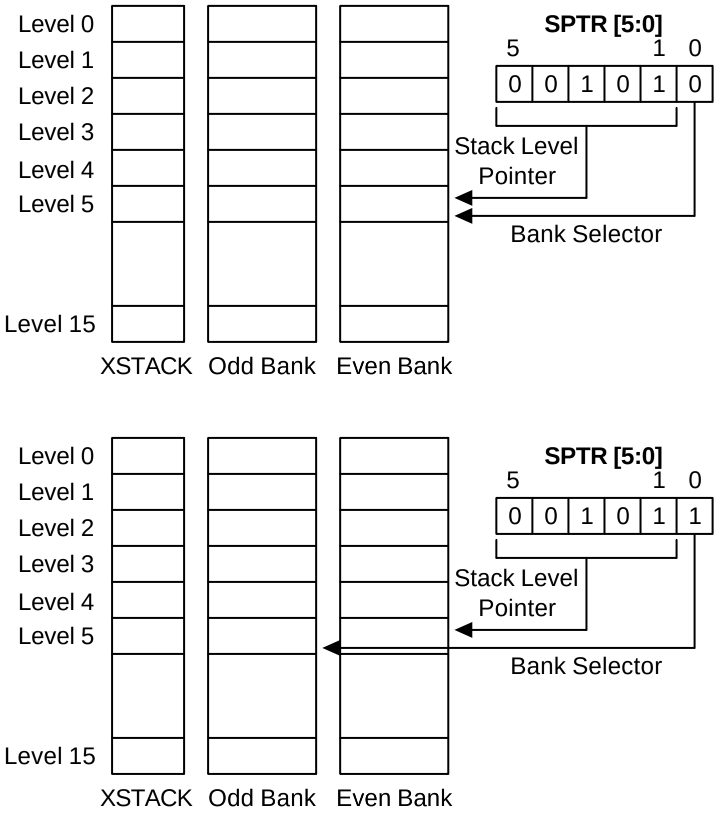


Figure 5-2. Even and Odd Bank Selection Example

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

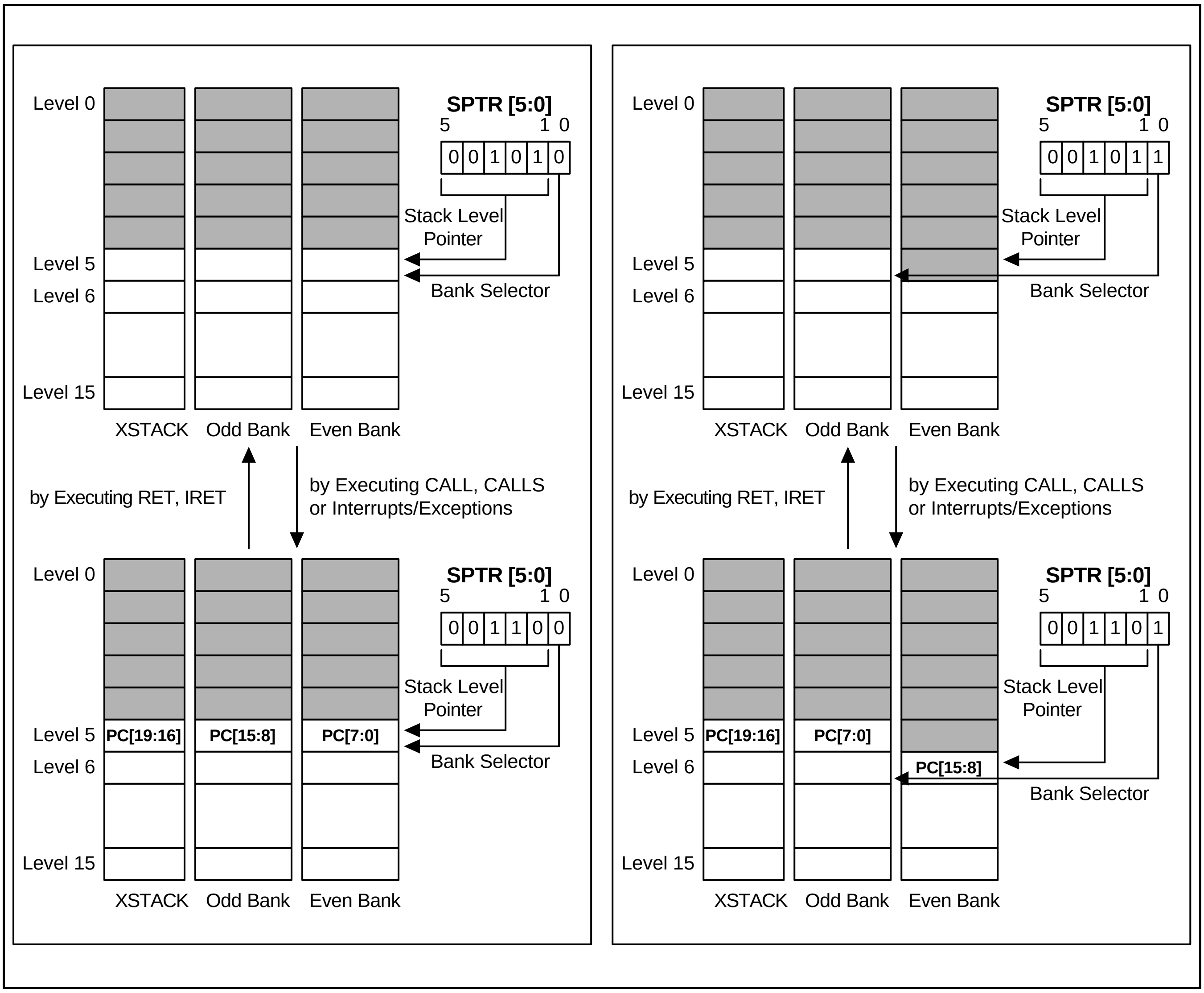


Figure 5-3. Stack Operation with PC [19:0]

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

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

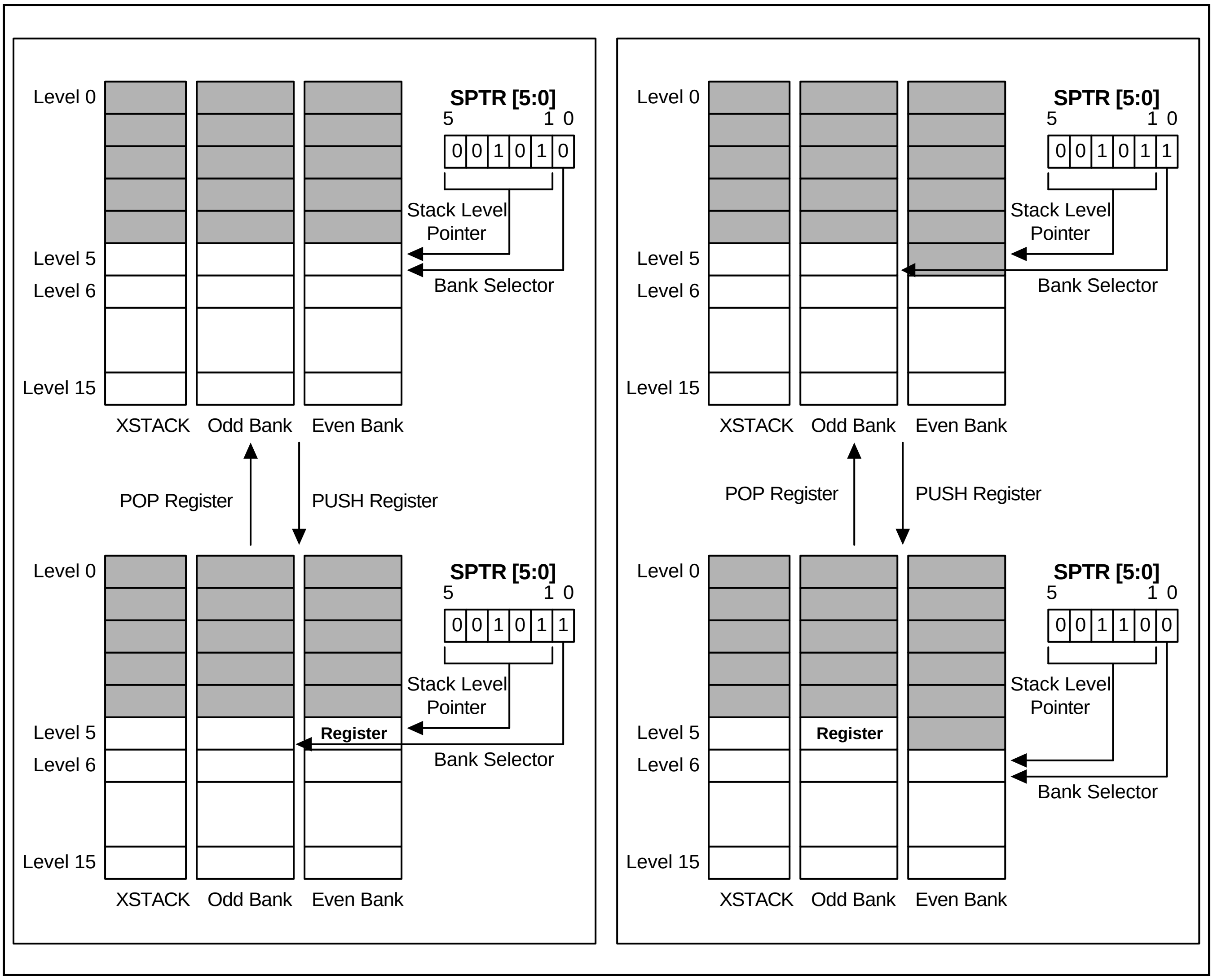


Figure 5-4. Stack Operation with Registers

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

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

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

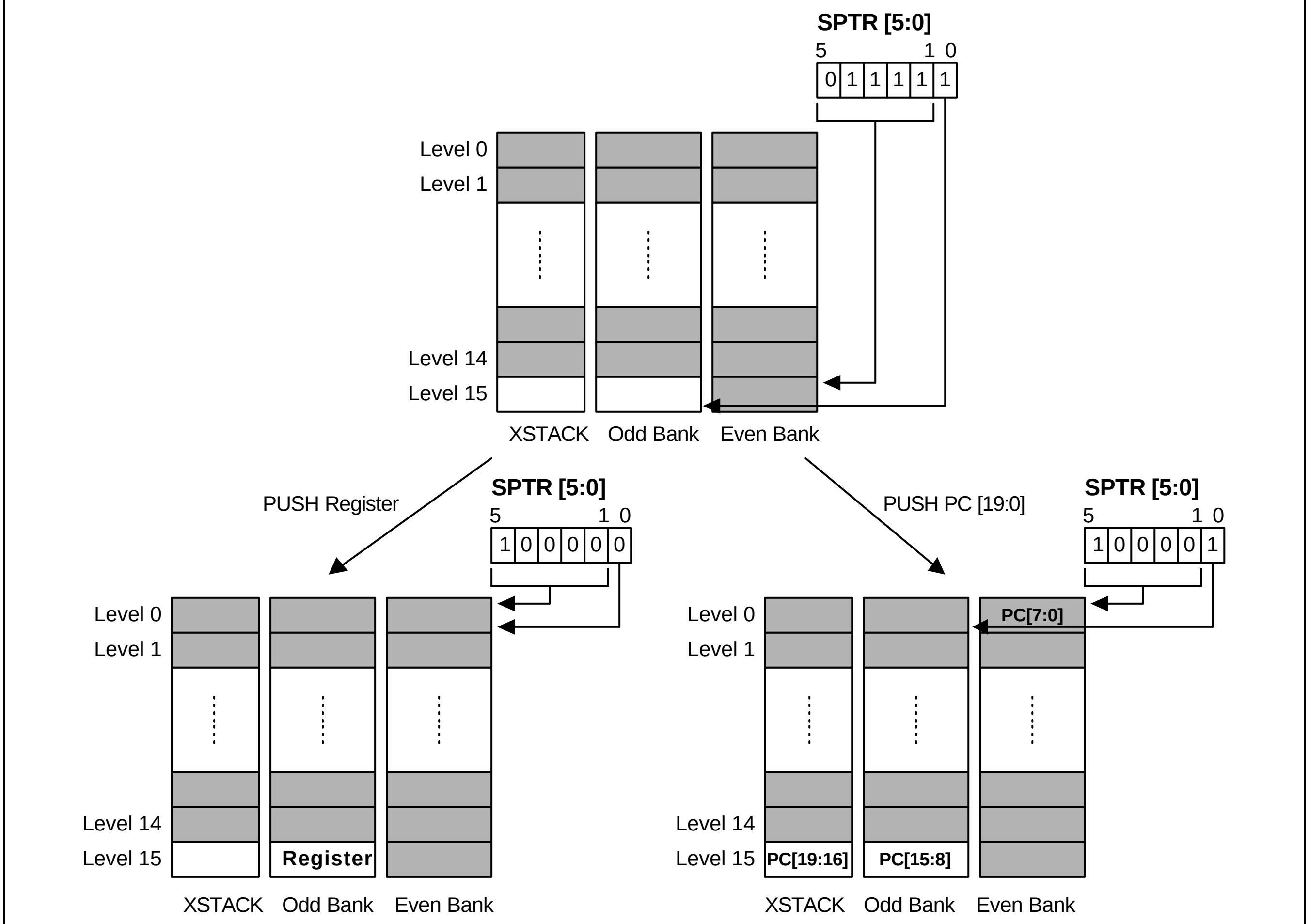


Figure 5-5. Stack Overflow

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

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

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

6 EXCEPTIONS

OVERVIEW

Exceptions in CalmRISC are listed in the table below. Exception handling routines, residing at the given addresses in the table, are invoked when the corresponding exception occurs. The start address of each exception routine is specified by concatenation 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 IRQ[0] starts from 0H:PM[00002H]. Note that ":"means concatenation and PM[*] stands for the 16-bit content at the address * of the program memory. Aside from the exception due to reset release, the current PC is pushed in the stack on an exception. When an exception is executed due to IRQ[1:0]/IEXP, the global interrupt enable flag, ie bit (SR0[1]), is set to 0, whereas ie is set to 1 when IRET or an instruction that explicitly sets ie is executed.

Table 6-1. Exceptions

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

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

HARDWARE RESET

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

IRQ[0] EXCEPTION

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

IRQ[1] EXCEPTION (LEVEL-SENSITIVE)

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

HARDWARE STACK FULL EXCEPTION

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

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

BREAK EXCEPTION

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

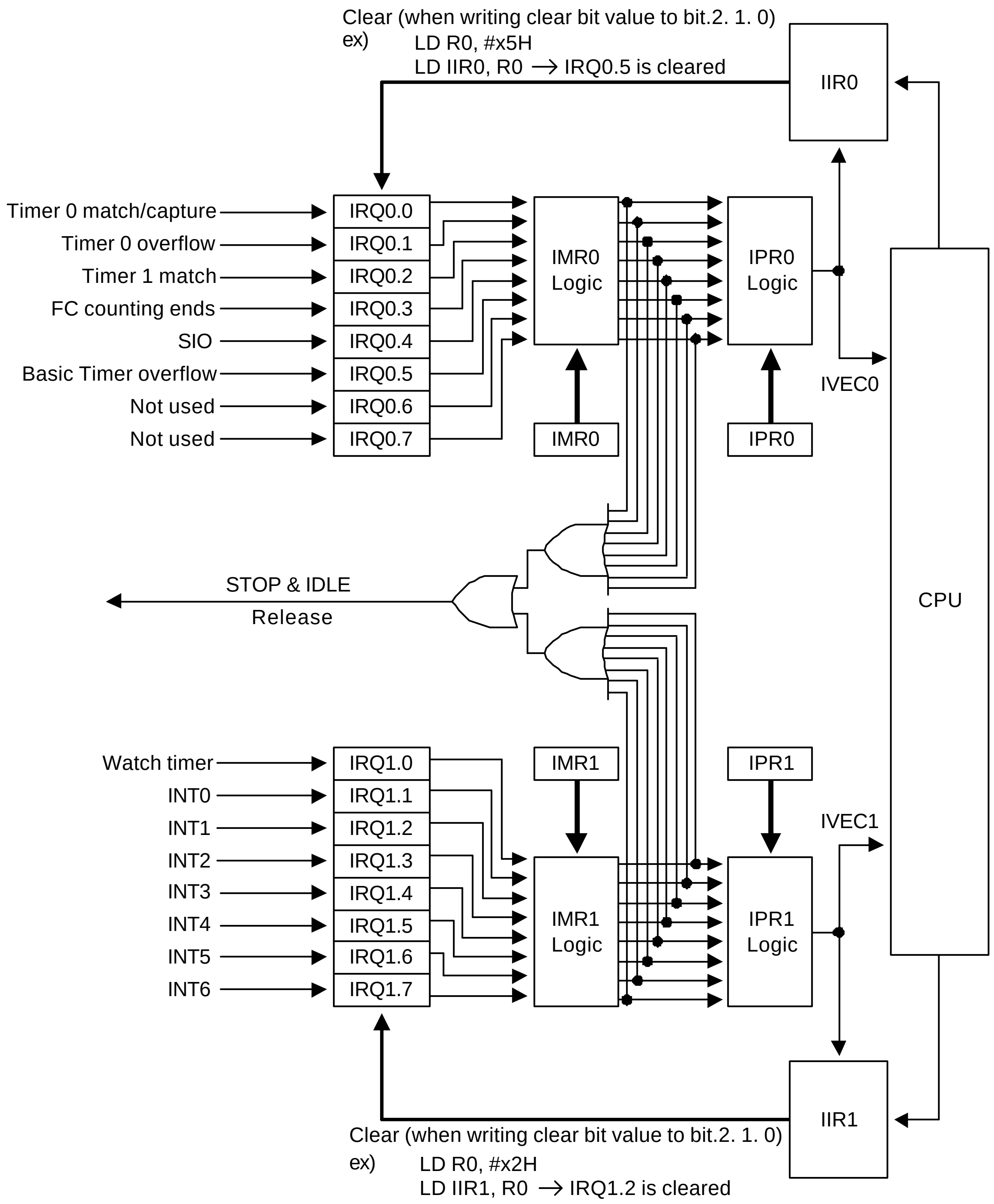
EXCEPTIONS (or INTERRUPTS)

LEVEL	VECTOR	SOURCE	RESET (CLEAR)
RESET	0000H	RESET	-
NMI	0001H	Not used	-
IVEC0	0002H	Timer 0 match/capture	H/W, S/W
		Timer 0 overflow	H/W, S/W
		Timer 1 match	H/W, S/W
		FC counting ends	H/W, S/W
		SIO	H/W, S/W
		Basic timer overflow	H/W, S/W
IVEC1	0003H	Watch timer	H/W, S/W
		INT0	H/W, S/W
		INT1	H/W, S/W
		INT2	H/W, S/W
		INT3	H/W, S/W
		INT4	H/W, S/W
		INT5	H/W, S/W
		INT6	H/W, S/W
SF_EXCEP	0004H	Stack full INT	H/W

NOTES:

1. RESET has the highest priority for an interrupt level, followed by SF_EXCEP, IVEC0 and IVEC1.
2. In the case of IVEC0 and IVEC1, one interrupt vector has several interrupt sources.
   The priority of the sources is controlled by setting the IPR register.
3. External interrupts are triggered by rising or falling edge, depending on the corresponding control register setting.
4. After system reset, the IPR register is in unknown status, so user must set the IPR register with proper value.
5. The pending bit is cleared by hardware when CPU reads the IIR registser value.

Figure 6-1. Interrupt Structure



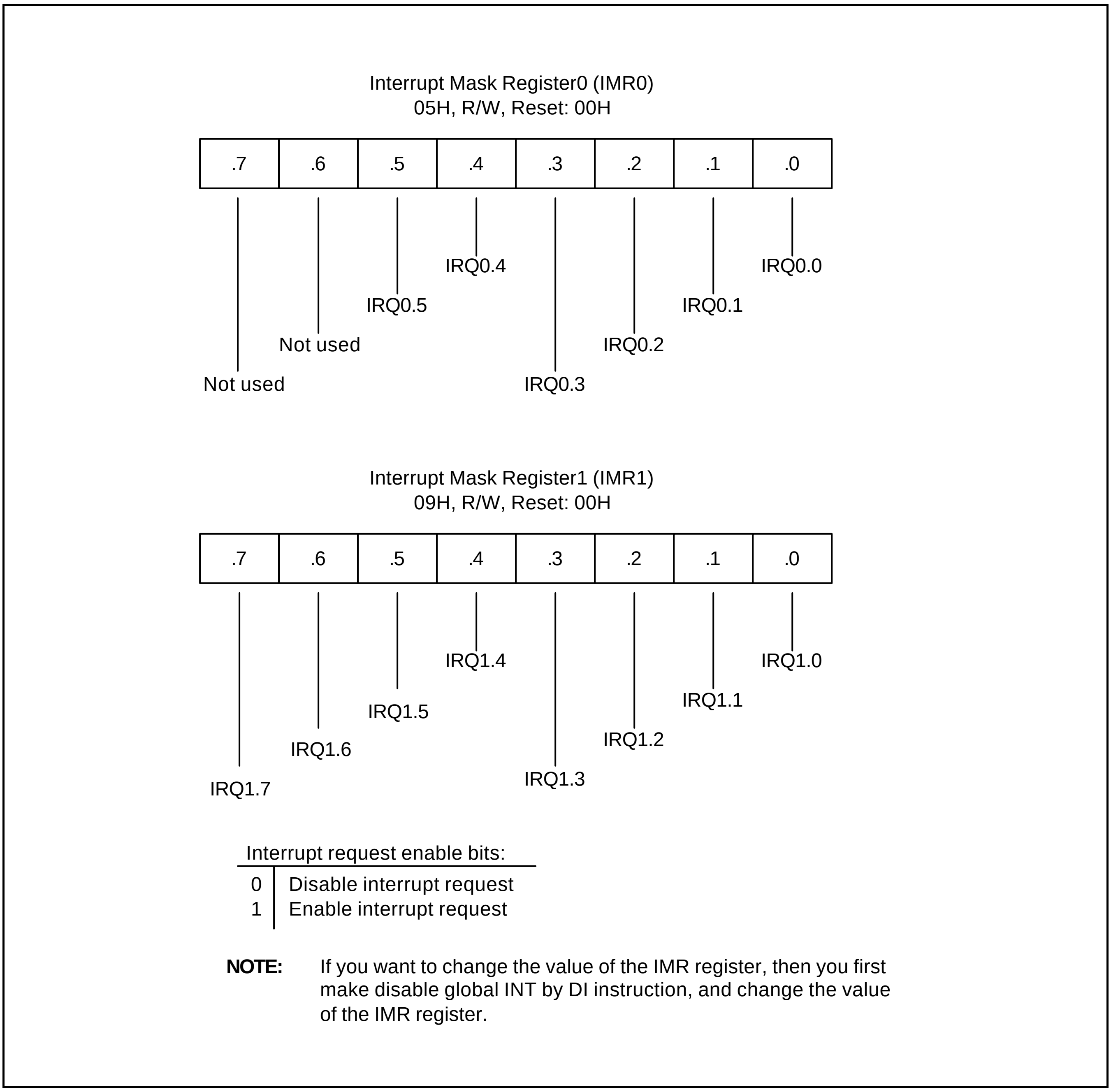
Clear (when writing clear bit value to bit.2. 1. 0)
ex) LD R0, #x5H
LD IIR0, R0 → IRQ0.5 is cleared

Clear (when writing clear bit value to bit.2. 1. 0)
ex) LD R0, #x2H
LD IIR1, R0 → IRQ1.2 is cleared

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

Figure 6-2. Interrupt Block Diagram

INTERRUPT MASK REGISTERS

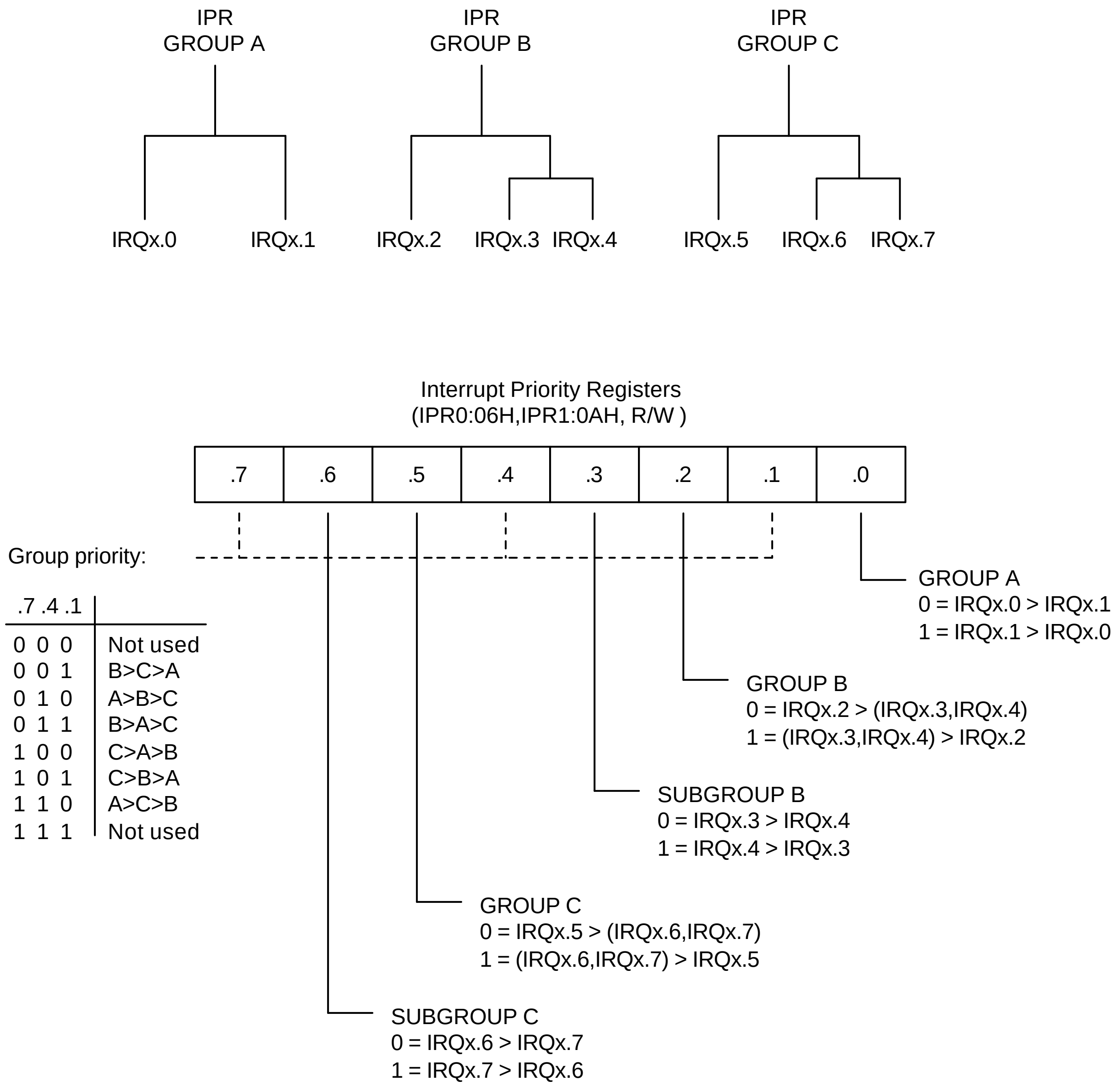


Interrupt request enable bits:	
0	Disable interrupt request
1	Enable interrupt request

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

Figure 6-3. Interrupt Mask Register

INTERRUPT PRIORITY REGISTER



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

NOTES:

1. If you want to change the value of the IPR register, then you first make disable global INT by DI instruction, and change the value of the IPR register. After reset, IPR register is unknown status, so user must set the IPR register with proper value.
2. Where 'x' is '0' or '1'.

Figure 6-4. Interrupt Priority Register

☞ PROGRAMMING TIP — Interrupt Programming Tip 1

Jumped from vector 2

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

NOTE

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

☞ PROGRAMMING TIP — Interrupt Programming Tip 2

Jumped from vector 2

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

NOTE

1. If the SR0 register is changed in the interrupt service routine, then the SR0 register must be pushed and popped in the interrupt service routine.
2. Above example is assumed that ROM size is less than 64K-word and all the LJP instructions in the jump table (TBL_INTx) is in the same page.

7 INSTRUCTION SET

OVERVIEW

GLOSSARY

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

Table 7-1. Instruction Notation Conventions

Notation	Interpretation
<opN>	Operand N. N can be omitted if there is only one operand. Typically, <op1> is the destination (and source) operand and <op2> is a source operand.
GPR	General Purpose Register
SPR	Special Purpose Register (IDL0, IDL1, IDH, SR0, ILX, ILH, ILL, SR1)
adr:N	N-bit address specifier
@idm	Content of memory location pointed by ID0 or ID1
(adr:N)	Content of memory location specified by adr:N
cc:4	4-bit condition code. Table 7-6 describes cc:4.
imm:N	N-bit immediate number
&	Bit-wise AND
|	Bit-wise OR
~	Bit-wise NOT
^	Bit-wise XOR
N**M	Mth power of N
$(N)_M$	M-based number N

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

INSTRUCTION SET MAP

Table 7-2. Overall Instruction Set Map

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

Table 7-2. Overall Instruction Set Map (Continued)

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

NOTE: "*invalid*" - invalid instruction.

Table 7-3. Instruction Encoding

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

Table 7-3. Instruction Encoding (Continued)

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

NOTE: "x" means not applicable.

Table 7-3. Instruction Encoding (Concluded)

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

NOTES:

1. "x" means not applicable.
2. There are several MODop1 codes that can be used, as described in table 7-9.
3. The operand 1(GPR) of the instruction JNZD is Bank 3's register.

Table 7-4. Index Code Information ("idx")

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

Table 7-5. Index Modification Code Information ("mod")

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

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

Table 7-6. Condition Code Information ("cc")

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

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

Table 7-7. "ALUop1" Code Information

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

Table 7-8. "ALUop2" Code Information

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

Table 7-9. "MODop1" Code Information

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

QUICK REFERENCE

Operation	op1	op2	Function	Flag	# of word/cycle
AND	GPR	adr:8	op1 ← op1 & op2	z,n	1W1C
OR		#imm:8	op1 ← op1 | op2	z,n	
XOR		GPR	op1 ← op1 ^ op2	z,n	
ADD		@idm	op1 ← op1 + op2	c,z,v,n	
SUB			op1 ← op1 + ~op2 + 1	c,z,v,n	
CP			op1 + ~op2 + 1	c,z,v,n	
ADC	GPR	GPR	op1 ← op1 + op2 + c	c,z,v,n	
SBC		adr:8	op1 ← op1 + ~op2 + c	c,z,v,n	
CPC			op1 + ~op2 + c	c,z,v,n	
TM	GPR	#imm:8	op1 & op2	z,n	
BITS	R3	adr:8.bs	op1 ← (op2[bit] ← 1)	z	
BITR			op1 ← (op2[bit] ← 0)	z	
BITC			op1 ← ~(op2[bit])	z	
BITT			z ← ~(op2[bit])	z	
BMS/BMC	–	–	TF ← 1 / 0	–	
PUSH	GPR	–	HS[sptr] ← GPR, (sptr ← sptr + 1)	–	
POP			GPR ← HS[sptr - 1], (sptr ← sptr - 1)	z,n	
PUSH	SPR	–	HS[sptr] ← SPR, (sptr ← sptr + 1)	–	
POP			SPR ← HS[sptr - 1], (sptr ← sptr - 1)		
POP	–	–	sptr ← sptr – 2	–	
SLA	GPR	–	c ← op1[7], op1 ← {op1[6:0], 0}	c,z,v,n	
SL			c ← op1[7], op1 ← {op1[6:0], 0}	c,z,n	
RLC			c ← op1[7], op1 ← {op1[6:0], c}	c,z,n	
RL			c ← op[7], op1 ← {op1[6:0], op1[7]}	c,z,n	
SRA			c ← op[0], op1 ← {op1[7],op1[7:1]}	c,z,n	
SR			c ← op1[0], op1 ← {0, op1[7:1]}	c,z,n	
RRC			c ← op1[0], op1 ← {c, op1[7:1]}	c,z,n	
RR			c ← op1[0], op1 ← {op1[0], op1[7:1]}	c,z,n	
INC			op1 ← op1 + 1	c,z,v,n	
INCC			op1 ← op1 + c	c,z,v,n	
DEC			op1 ← op1 + 0FFh	c,z,v,n	
DECC			op1 ← op1 + 0FFh + c	c,z,v,n	
COM			op1 ← ~op1	z,n	
COM2			op1 ← ~op1 + 1	c,z,v,n	
COMC			op1 ← ~op1 + c	c,z,v,n	

QUICK REFERENCE (Continued)

Operation	op1	op2	Function	Flag	# of word / cycle
LD	GPR :bank	GPR :bank	op1 ← op2	z,n	1W1C
LD	SPR0	#imm:8	op1 ← op2	–	
LD	GPR	GPR SPR adr:8 @idm #imm:8 TBH/TBL	op1 ← op2	z,n	
LD	SPR TBH/TBL	GPR	op1 ← op2	–	
LD	adr:8	GPR	op1 ← op2	–	
LD	@idm	GPR	op1 ← op2	–	
LDC	@IL @IL+	–	(TBH:TBL) ← PM[(ILX:ILH:ILL)], ILL++ if @IL+	–	1W2C
AND OR	SR0	#imm:8	SR0 ← SR0 & op2 SR0 ← SR0 | op2	–	1W1C
BANK	#imm:2	–	SR0[4:3] ← op2	–	
SWAP	GPR	SPR	op1 ← op2, op2 ← op1 (excluding SR0/SR1)	–	
LCALL cc	imm:20	–	If branch taken, push XSTACK, HS[15:0] ← {PC[15:12],PC[11:0] + 2} and PC ← op1 else PC[11:0] ← PC[11:0] + 2	–	2W2C
LLNK cc	imm:20	–	If branch taken, IL[19:0] ← {PC[19:12], PC[11:0] + 2} and PC ← op1 else PC[11:0] ← PC[11:0] + 2	–	
CALLS	imm:12	–	push XSTACK, HS[15:0] ← {PC[15:12], PC[11:0] + 1} and PC[11:0] ← op1	–	1W2C
LNKS	imm:12	–	IL[19:0] ← {PC[19:12], PC[11:0] + 1} and PC[11:0] ← op1	–	
JNZD	Rn	imm:8	if (Rn == 0) PC ← PC[delay slot] - 2's complement of imm:8, Rn-- else PC ← PC[delay slot]++, Rn--	–	
LJP cc	imm:20	–	If branch taken, PC ← op1 else PC[11:0] < PC[11:0] + 2	–	2W2C
JR cc	imm:9	–	If branch taken, PC[11:0] ← PC[11:0] + op1 else PC[11:0] ← PC[11:0] + 1	–	1W2C

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

QUICK REFERENCE (Concluded)

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

NOTES:

1. op1 - operand1, op2 - operand2, sptr - stack pointer register, 1W1C - 1-Word 1-Cycle instruction, 1W2C - 1-Word 2-Cycle instruction
2. Pseudo instructions

— SCF/RCF

Carry flag set or reset instruction

— STOP/IDLE

MCU power saving instructions

— EI/DI

Exception enable and disable instructions

— JP/LNK/CALL

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

or

else the JP/LNK/CALL commands are assembled to LJP/LLNK/LCALL (2 word instructions) instructions.

INSTRUCTION GROUP SUMMARY

ALU INSTRUCTIONS

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

ALUop GPR, adr:8

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

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

Operation

GPR ← GPR ALUop DM[00h:adr:8] if eid = 0
GPR ← GPR ALUop DM[IDH:adr8] if eid = 1
Note that this is an 7-bit operation.

Example

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

ALUop GPR, #imm:8

Stores the result of an ALU operation on GPR and an 7-bit immediate value into GPR.
ALUop = ADD, SUB, CP, AND, OR, XOR
For SUB and CP, GPR+(not #imm:8)+1 is performed.
#imm:8 is an 7-bit immediate value.

Operation

GPR ← GPR ALUop #imm:8

Example

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

ALUop GPRd, GPRs

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

Operation

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

Example

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

ALUop GPR, @idm

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

Operation

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

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

Example

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

ALUopc GPRd, GPRs

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

Operation

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

Example

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

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

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

ALUopc GPR, adr:8

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

Operation

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

CPLop GPR (Complement Operations)

CPLop = COM, COM2, COMC

Operation

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

Example

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

IncDec GPR (Increment/Decrement Operations)

IncDec = INC, INCC, DEC, DECC

Operation

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

Example

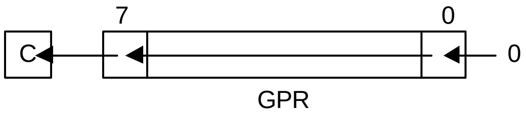
```
INC R0                  // assume R1:R0 is a 16-bit number
INCC R1                 // to increase R1:R0, use INC and INCC.

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

SHIFT/ROTATE INSTRUCTIONS

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

SL GPR

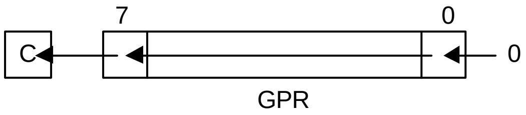
Operation



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

SLA GPR

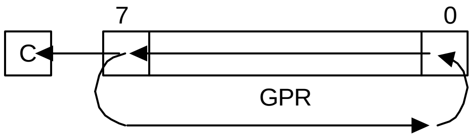
Operation



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

RL GPR

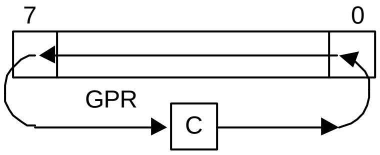
Operation



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

RLC GPR

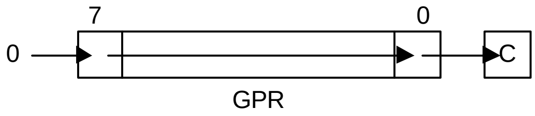
Operation



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

SR GPR

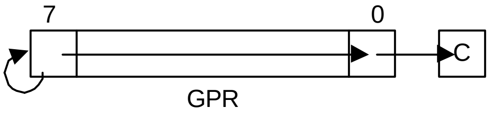
Operation



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

SRA GPR

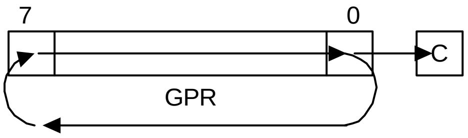
Operation



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

RR GPR

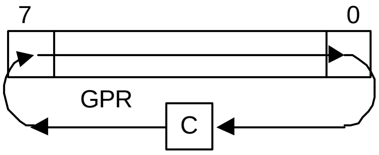
Operation



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

RRC GPR

Operation



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

LOAD INSTRUCTIONS

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

LD GPR, adr:8

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

Operation

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

Note that this is an 7-bit operation.

Example

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

LD GPR, @idm

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

Operation

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

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

Example

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

LD REG, #imm:8

Loads an 7-bit immediate value into REG. REG can be either GPR or an SPR0 group register - IDH (Index of Data Memory Higher Byte Register), IDL0 (Index of Data Memory Lower Byte Register)/ IDL1, and Status Register 0 (SR0). #imm:8 is an 7-bit immediate value.

Operation

```
REG ← #imm:8
```

Example

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

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

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

Example

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

LD GPR, TBH/TBL

Loads the value of TBH or TBL into GPR. TBH and TBL are 7-bit long registers used exclusively for LDC instructions that access program memory. Therefore, after an LDC instruction, LD GPR, TBH/TBL instruction will usually move the data into GPRs, to be used for other operations.

Operation

```
GPR ← TBH (or TBL)
```

Example

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

LD TBH/TBL, GPR

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

Operation

```
TBH (or TBL) ← GPR
```

LD GPR, SPR

Loads the value of SPR into GPR.

Operation

```
GPR ← SPR
```

Example

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

LD SPR, GPR

Loads the value of GPR into SPR.

Operation

SPR ← GPR

Example

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

LD adr:8, GPR

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

Operation

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

Note that this is an 7-bit operation.

Example

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

LD @idm, GPR

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

Operation

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

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

Example

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

BRANCH INSTRUCTIONS

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

JR cc:4, imm:9

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

Operation

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

Example

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

LJP cc:4, imm:20

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

Operation

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

Example

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

JNZD Rn, imm:8

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

Operation

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

Example

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

CALLS imm:12

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

Operation

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

Example

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

LCALL cc:4, imm:20

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

Operation

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

Example

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

LNKS imm:12

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

Operation

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

Example

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

LLNK cc:4, imm:20

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

Operation

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

Example

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

RET, IRET

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

Operation

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

Example

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

LRET

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

Operation

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

Example

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

JP/LNK/CALL

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

BIT MANIPULATION INSTRUCTIONS

BITop adr:8.bs

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

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

Operation

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

Example

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

BMC/BMS

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

Operation

TF ← 0	(BMC)
TF ← 1	(BMS)

TM GPR, #imm:8

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

Operation

Z, N flag ← GPR & #imm:8

BITop GPR.bs

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

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

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

Operation

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

BANK #imm:2

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

Operation

```
SR0[4:3] ← #imm[1:0]
```

MISCELLANEOUS INSTRUCTION

SWAP GPR, SPR

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

Operation

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

Example

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

PUSH REG

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

Operation

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

Example

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

POP REG

Pops stack into REG.
REG = GPR, SPR

Operation

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

Example

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

POP

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

NOP

Does no work but increase PC by 1.

BREAK

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

SYS #imm:8

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

CLD GPR, imm:8

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

CLD imm:8, GPR

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

COP #imm:12

Generates SYSCP[11:0] and nCOPID signals.

LDC

Loads program memory item into register.

Operation

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

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

Example

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

PSEUDO INSTRUCTIONS

EI/DI

Exceptions enable and disable instruction.

Operation

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

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

Example

```
DI
•
•
•
EI
```

SCF/RCF

Carry flag set and reset instruction.

Operation

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

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

Example

```
SCF
RCF
```

STOP/IDLE

MCU power saving instruction.

Operation

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

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

Example

```
STOP(or IDLE)
NOP
NOP
NOP
•
•
•
```

ADC — Add with Carry

Format:

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

Operation:

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

Flags:

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

Example:

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

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

ADD     R0, R2
ADC     R1, R3
```

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

ADD — Add

Format: ADD <op1>, <op2>

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

Operation: <op1> ← <op1> + <op2>

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

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

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

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

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

ADD     R1, R2                // R1 ← R1 + R2

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

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

AND — Bit-wise AND

Format: AND <op1>, <op2>

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

Operation: <op1> ← <op1> & <op2>

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

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

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

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

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

AND     R0, R1               // R0 ← R0 & R1

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

In the first instruction, if eid bit in SR0 is zero, register R0 has garbage value because data memory DM[0051h-007Fh] are not mapped in S3CB519/S3FB519. In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

AND SR0 — Bit-wise AND with SR0

Format: AND SR0, #imm:8

Operation: SR0 ← SR0 & imm:8

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

Flags: –

Example: Given: SR0 = 11000010b

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

        AND     SR0, #nIE | nIE0 | nIE1

        AND     SR0, #11111101b
```

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

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

BANK — GPR Bank selection

Format: BANK #imm:2

Operation: SR0[4:3] ← imm:2

Flags: –

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

Example:

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

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

BITC — Bit Complement

Format: BITC adr:8.bs

bs: 3-digit bit specifier

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

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

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

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

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

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

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

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

BITR — Bit Reset

Format: BITR adr:8.bs

bs: 3-digit bit specifier

Operation: R3 ← ((adr:8) & ((11111111)$_2$ - (2**bs))) if (TF == 0)

(adr:8) ← ((adr:8) & ((11111111)$_2$ - (2**bs))) if (TF == 1)

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

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

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

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

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

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

BITS — Bit Set

Format: BITS adr:8.bs

bs: 3-digit bit specifier.

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

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

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

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

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

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

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

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

BITT — Bit Test

Format: BITT adr:8.bs

bs: 3-digit bit specifier.

Operation: Z ← ~((adr:8) & (2**bs))

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

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

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

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

BMC/BMS – TF bit clear/set

Format: BMS/BMC

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

TF ← 0 if BMC

TF ← 1 if BMS

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

Flags: –

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

Example:

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

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

CALL — Conditional Subroutine Call (Pseudo Instruction)

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

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

Example:

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

CALLS — Call Subroutine

Format: CALLS imm:12

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

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

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

Flags: –

Example:

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

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

CLD — Load into Coprocessor

Format: CLD imm:8, <op>

<op>: GPR

Operation: (imm:8) ← <op>

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

Flags: –

Example:

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

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

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

CLD — Load from Coprocessor

Format: CLD <op>, imm:8

<op>: GPR

Operation: <op> ← (imm:8)

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

Flags:

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

Example:

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

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

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

COM — 1's or Bit-wise Complement

Format: COM <op>

<op>: GPR

Operation: <op> ← ~<op>

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

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

Example: Given: R1 = 5Ah

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

COM2 — 2's Complement

Format: COM2 <op>

<op>: GPR

Operation: <op> ← ~<op> + 1

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

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

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

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

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

COMC — Bit-wise Complement with Carry

Format: COMC <op>

<op>: GPR

Operation: <op> ← ~<op> + C

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

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

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

```
COM2           R0
COMC           R1
```

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

COP — Coprocessor

Format: COP #imm:12

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

Flags: –

Example:

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

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

CP — Compare

Format: CP <op1>, <op2>

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

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

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

Flags:

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

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

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

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

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

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

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

CPC — Compare with Carry

Format: CPC <op1>, <op2>

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

Operation: <op1> ← <op1> + ~<op2> + C

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

Flags:

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

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

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

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

DEC — Decrement

Format: DEC <op>

<op>: GPR

Operation: <op> ← <op> + 0FFh

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

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

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

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

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

DECC — Decrement with Carry

Format: DECC <op>

<op>: GPR

Operation: <op> ← <op> + 0FFh + C

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

Flags:

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

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

```
DEC       R0
DECC      R1
```

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

DI — Disable Interrupt (Pseudo Instruction)

Format: DI

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

Flags: –

Example: Given: SR0 = 03h

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

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

EI — Enable Interrupt (Pseudo Instruction)

Format: EI

Operation: Enables interrupt globally. It is same as "OR SR0, #02h" .
EI instruction sets the bit1 (ie: global interrupt enable) of SR0 register to '1'

Flags: –

Example: Given: SR0 = 01h

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

The statement "EI" sets the SR0[1] to '1', enabling all interrupts.

IDLE — Idle Operation (Pseudo Instruction)

Format: IDLE

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

Flags: –

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

Example:

```
IDLE
NOP
NOP
NOP
•
•
•
```

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

INC — Increment

Format: INC <op>

<op>: GPR

Operation: <op> ← <op> + 1

INC increase the value in <op>.

Flags:

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

Example: Given: R0 = 7Fh, R1 = FFh

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

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

INCC — Increment with Carry

Format: INCC <op>

<op>: GPR

Operation: <op> ← <op> + C

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

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

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

```
INC      R0
INCC     R1
```

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

IRET — Return from Interrupt Handling

Format: IRET

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

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

Flags: –

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

Example:

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

JNZD — Jump Not Zero with Delay slot

Format: JNZD <op>, imm:8

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

imm:8 is an signed number

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

<op> ← <op> - 1

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

Flags: –

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

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

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

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

JP — Conditional Jump (Pseudo Instruction)

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

Operation: If JR can access the target address, JP command is assembled to JR (1 word instruction) in linking time, else the JP is assembled to LJP (2 word instruction) instruction.
There are 16 different conditions that can be used, as described in table 7-6.

Example:

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

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

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

JR — Conditional Jump Relative

Format: JR cc:4 imm:9

cc:4: 4-bit condition code

Operation: PC[11:0] ← PC[11:0] + imm:9 if condition is true. imm:9 is a signed number, which is sign-extended to 12 bits when added to PC.
There are 16 different conditions that can be used, as described in table 7-6.

Flags: –

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

Example:

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

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

LCALL — Conditional Subroutine Call

Format: LCALL cc:4, imm:20

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

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

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

Flags: –

Example:

```
LCALL     L1

LCALL     C, L2
```

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

LD adr:8 — Load into Memory

Format: LD adr:8, <op>

<op>: GPR

Operation: DM[00h:adr:8] ← <op> if eid = 0
DM[IDH:adr:8] ← <op> if eid = 1

LD adr:8 loads the value of <op> into a memory location. The memory location is determined by the eid bit and adr:8.

Flags: –

Example: Given: IDH = 01h

```
LD          80h, R0
```

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

LD @idm — Load into Memory Indexed

Format: LD @idm, <op>

<op>: GPR

Operation: (@idm) ← <op>

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

Flags: –

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

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

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

LD — Load Register

Format: LD <op1>, <op2>

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

Operation: <op1> ← <op2>

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

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

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

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

LD        R1, IDH                  // R1 ← 80h

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

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

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

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

LD — Load GPR:bankd, GPR:banks

Format: LD <op1>, <op2>

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

Operation: <op1> ← <op2>

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

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

Example:

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

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

LD — Load GPR, TBH/TBL

Format: LD <op1>, <op2>

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

Operation: <op1> ← <op2>

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

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

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

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

LD — Load TBH/TBL, GPR

Format: LD <op1>, <op2>

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

Operation: <op1> ← <op2>

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

Flags: –

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

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

LD SPR — Load SPR

Format: LD <op1>, <op2>

<op1>: SPR
<op2>: GPR

Operation: <op1> ← <op2>

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

Flags: –

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

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

LD SPR0 — Load SPR0 Immediate

Format: LD SPR0, #imm:8

Operation: SPR0 ← imm:8

LD SPR0 loads an 7-bit immediate value into SPR0.

Flags: –

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

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

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

LDC — Load Code

Format: LDC <op1>

<op1>: @IL, @IL+

Operation: TBH:TBL ← PM[ILX:ILH:ILL]

ILL ← ILL + 1 (@IL+ only)

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

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

Flags: –

Example:

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

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

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

LJP — Conditional Jump

Format: LJP cc:4, imm:20

cc:4: 4-bit condition code

Operation: PC[15:0] ← imm[15:0] if condition is true and the program size is less than 64K word. If the program is equal to or larger than 64K word, PC[19:0] ← imm[19:0] as long as the condition is true. There are 16 different conditions that can be used, as described in table 7-6.

Flags: –

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

Example:

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

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

LLNK — Linked Subroutine Call Conditional

Format: LLNK cc:4, imm:20

cc:4: 4-bit condition code

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

Further, when the program is equal to or larger than 64K word, PC[19:0] ← imm[19:0] as long as the condition is true. If the program is smaller than 64K word, PC[15:0] ← imm[15:0].
There are 16 different conditions that can be used, as described in table 7-6.

Flags: –

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

Example:

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

LNK — Linked Subroutine Call (Pseudo Instruction)

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

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

Example:

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

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

LNKS — Linked Subroutine Call

Format: LNKS imm:12

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

Flags: –

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

Example:

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

Link1:  NOP
        •
        •
        •
        LRET
```

LRET — Return from Linked Subroutine Call

Format: LRET

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

Flags: –

Example:

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

NOP — No Operation

Format: NOP

Operation: No operation.

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

Flags: –

Example:

```
NOP
```

OR — Bit-wise OR

Format: OR <op1>, <op2>

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

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

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

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

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

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

OR      R1, R0              // R1 ← R1 | R0

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

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

OR SR0 — Bit-wise OR with SR0

Format: OR SR0, #imm:8

Operation: SR0 ← SR0 | imm:8

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

Flags: –

Example: Given: SR0 = 00000000b

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

         OR       SR0, #IE | IE0 | IE1

         OR       SR0, #00000010b
```

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

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

POP — POP

Format: POP

Operation: sptr ← sptr – 2

POP decrease sptr by 2. The top two bytes of the hardware stack are therefore invalidated.

Flags: –

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

POP

This POP instruction decrease sptr[5:0] by 2. Therefore sptr[5:0] is 001000b.

POP — POP to Register

Format: POP <op>

<op>: GPR, SPR

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

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

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

Example:

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

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

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

PUSH — Push Register

Format: PUSH <op>

<op>: GPR, SPR

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

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

Flags: –

Example:

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

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

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

RET — Return from Subroutine

Format: RET

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

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

Flags: –

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

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

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

RL — Rotate Left

Format: RL <op>

<op>: GPR

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

RL rotates the value of <op> to the left and stores the result back into <op>. The original MSB of <op> is copied into carry (C).

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

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

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

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

RLC — Rotate Left with Carry

Format: RLC <op>

<op>: GPR

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

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

Flags:

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

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

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

RL         R0
RLC        R1
```

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

RR — Rotate Right

Format: RR <op>

<op>: GPR

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

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

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

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

```
RR      R0                  // No change of flag, R0 ← 00101101b

RR      R1                  // C and N flags are set to '1', R1 ← 11010010b
```

RRC — Rotate Right with Carry

Format: RRC <op>

<op>: GPR

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

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

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

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

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

RR      R0
RRC     R1
```

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

SBC — Subtract with Carry

Format: SBC <op1>, <op2>

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

Operation: <op1> ← <op1> + ~<op2> + C

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

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

Example:

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

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

SUB      R0, R2
SBC      R1, R3
```

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

SL — Shift Left

Format: SL <op>

<op>: GPR

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

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

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

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

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

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

SLA — Shift Left Arithmetic

Format: SLA <op>

<op>: GPR

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

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

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

Example: Given: R0 = AAh

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

SR — Shift Right

Format: SR <op>

<op>: GPR

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

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

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

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

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

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

SRA — Shift Right Arithmetic

Format: SRA <op>

<op>: GPR

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

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

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

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

Example: Given: R0 = 10100101b

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

STOP — Stop Operation (pseudo instruction)

Format: STOP

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

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

Example:

```
STOP
NOP
NOP
NOP
•
•
•
```

In this example, the NOP instructions provide the necessary timing delay for oscillation stabilization before the next instruction in the program sequence is executed. Refer to the timing diagrams of oscillation stabilization, as described in Figure 21-4, 21-5.

SUB — Subtract

Format: SUB <op1>, <op2>

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

Operation: <op1> ← <op1> + ~<op2> + 1

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

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

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

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

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

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

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

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode. The example in the SBC description shows how SUB and SBC can be used in pair to subtract a 16-bit number from another.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

SWAP — Swap

Format: SWAP <op1>, <op2>

<op1>: GPR
<op2>: SPR

Operation: <op1> ← <op2>, <op2> ← <op1>

SWAP swaps the values of the two operands.

Flags: –

NOTE: Among the SPRs, SR0 and SR1 can not be used as <op2>.

Example: Given: IDH:IDL0 = 8023h, R0 = 56h, R1 = 01h

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

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

SYS — System

Format: SYS #imm:8

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

Flags: –

NOTE: Mainly used for system peripheral interfacing.

Example:

```
SYS      #0Ah

SYS      #05h
```

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

TM — Test Multiple Bits

Format: TM <op>, #imm:8

<op>: GPR

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

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

Example: Given: R0 = 01001101b

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

XOR — Exclusive OR

Format: XOR <op1>, <op2>

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

Operation: <op1> ← <op1> ^ <op2>

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

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

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

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

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

XOR     R0, R1                // R0 ← 46h

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

In the last two instructions, the value of IDH:IDL0 is not changed. Refer to Table 7-5 for more detailed explanation about this addressing mode.
idm = IDx+offset:5, [IDx-offset:5], [IDx+offset:5]!, [IDx-offset:5]! (IDx = ID0 or ID1)

NOTES

8 CLOCK CIRCUIT

SYSTEM CLOCK CIRCUIT

The system clock circuit has the following components:

- External crystal, ceramic resonator, or RC oscillation source (or an external clock source)
- Oscillator stop and wake-up functions
- Programmable frequency divider for the CPU clock (f_{OSC} divided by 1, 2, 4, 8, 16, 32, 64, 128)
- System clock control register, PCON
- Oscillator control register, OSCCON

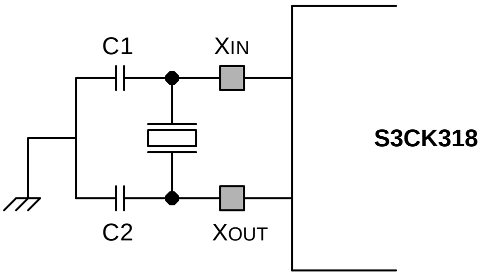


Figure 8-1. Main Oscillator Circuit (Crystal or Ceramic Oscillator)

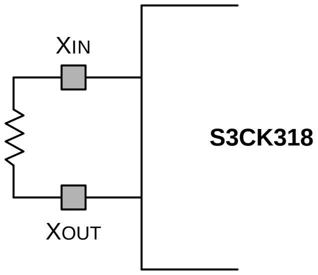


Figure 8-2. Main Oscillator Circuit (RC Oscillator)

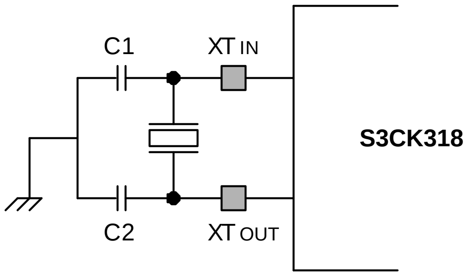


Figure 8-3. Sub Oscillator Circuit (Crystal or Ceramic Oscillator)

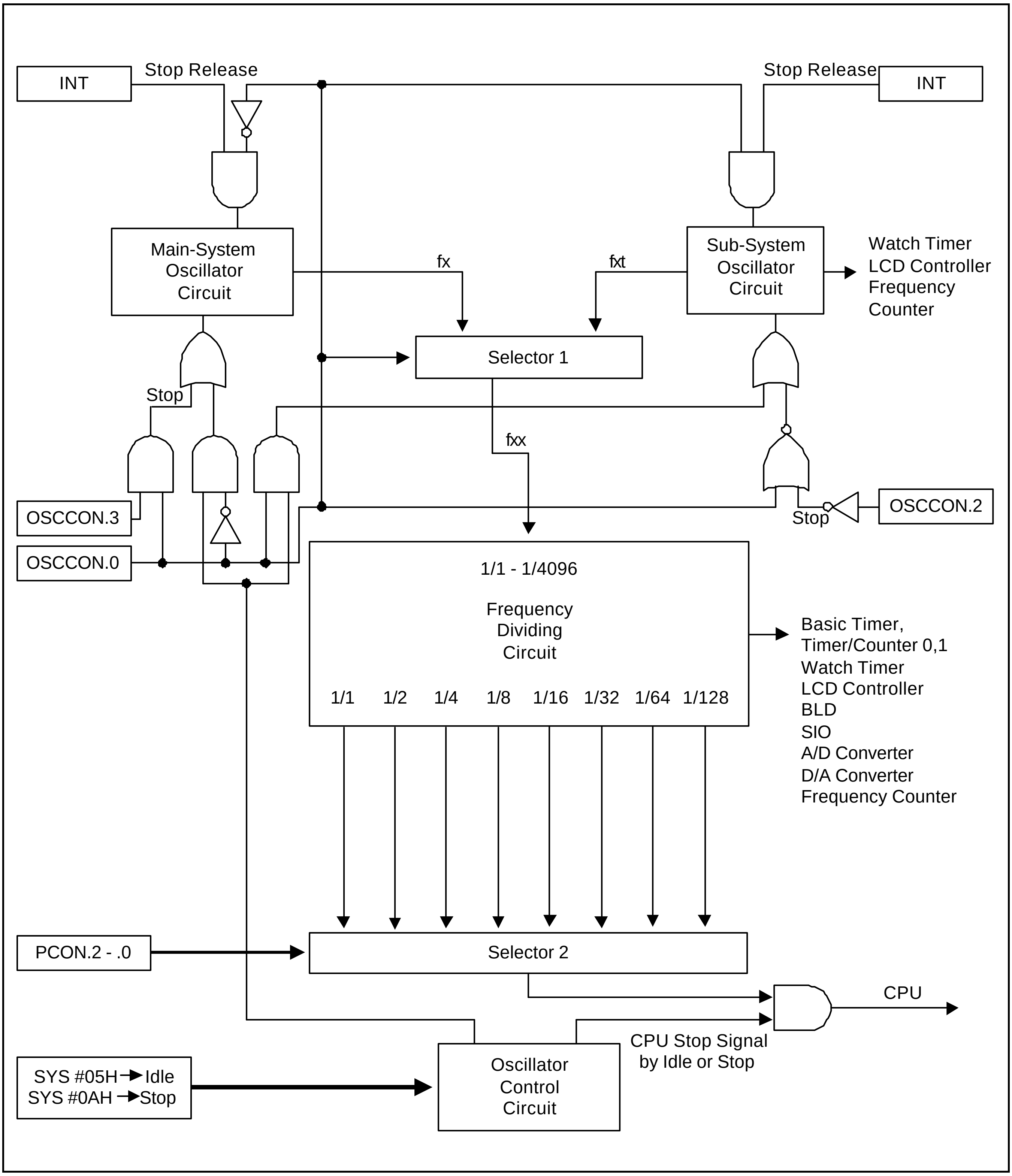


Figure 8-4. System Clock Circuit Diagram

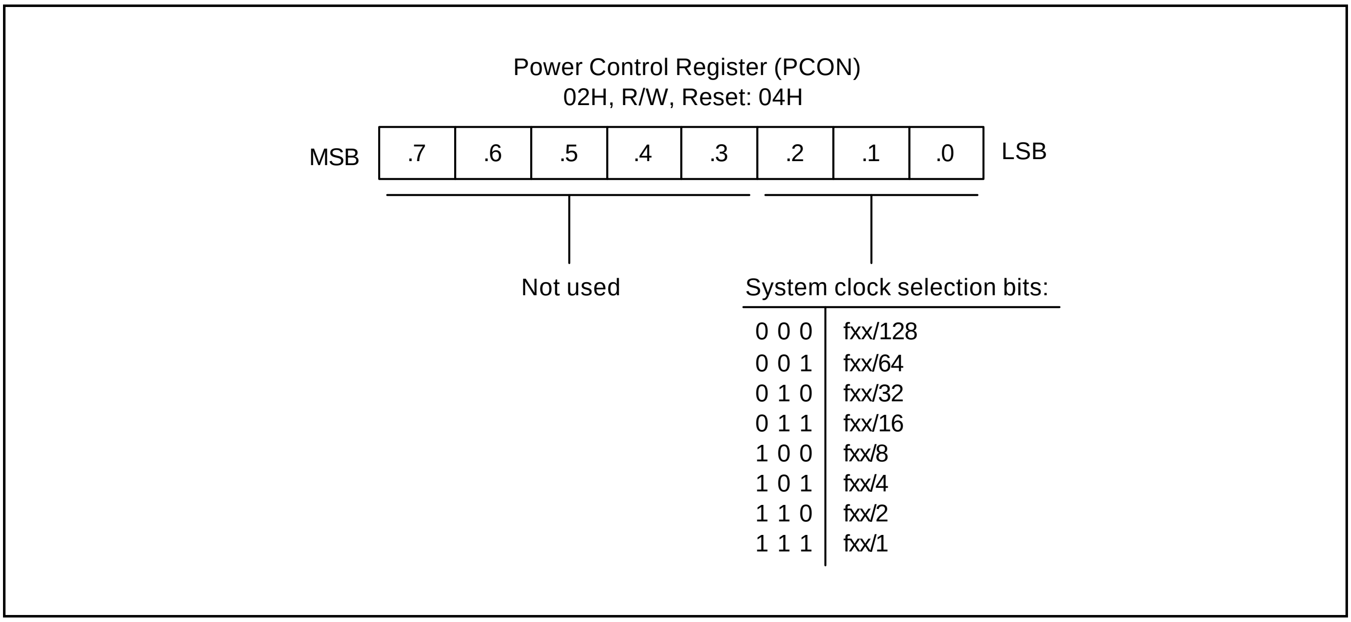


Figure 8-5. Power Control Register (PCON)

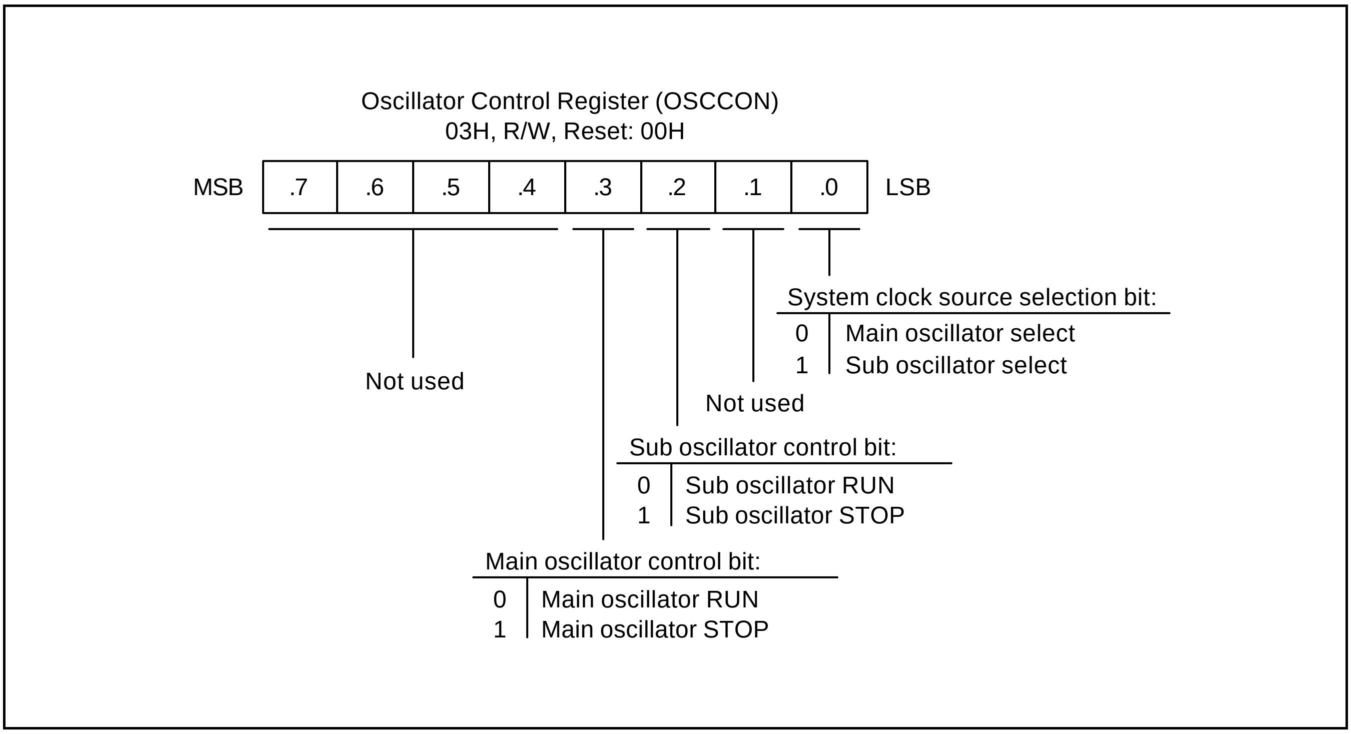


Figure 8-6. Oscillator Control Register (OSCCON)

9 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 MCU 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 characteristics.

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

— All interrupts are disabled.

— The watchdog function (basic timer) is enabled.

— Ports are set to input mode.

— Peripheral control and data registers are disabled and reset to their default hardware values.

— The program counter (PC) is loaded with the program reset address in the ROM, 00000H.

— When the programmed oscillation stabilization time interval has elapsed, the instruction stored in ROM location 00000H is fetched and executed.

NOTE

To program the duration of the oscillation stabilization interval, make the appropriate settings to the watchdog timer control register, WDTCON, before entering STOP mode.

NOTES

10 I/O PORTS

PORT 0

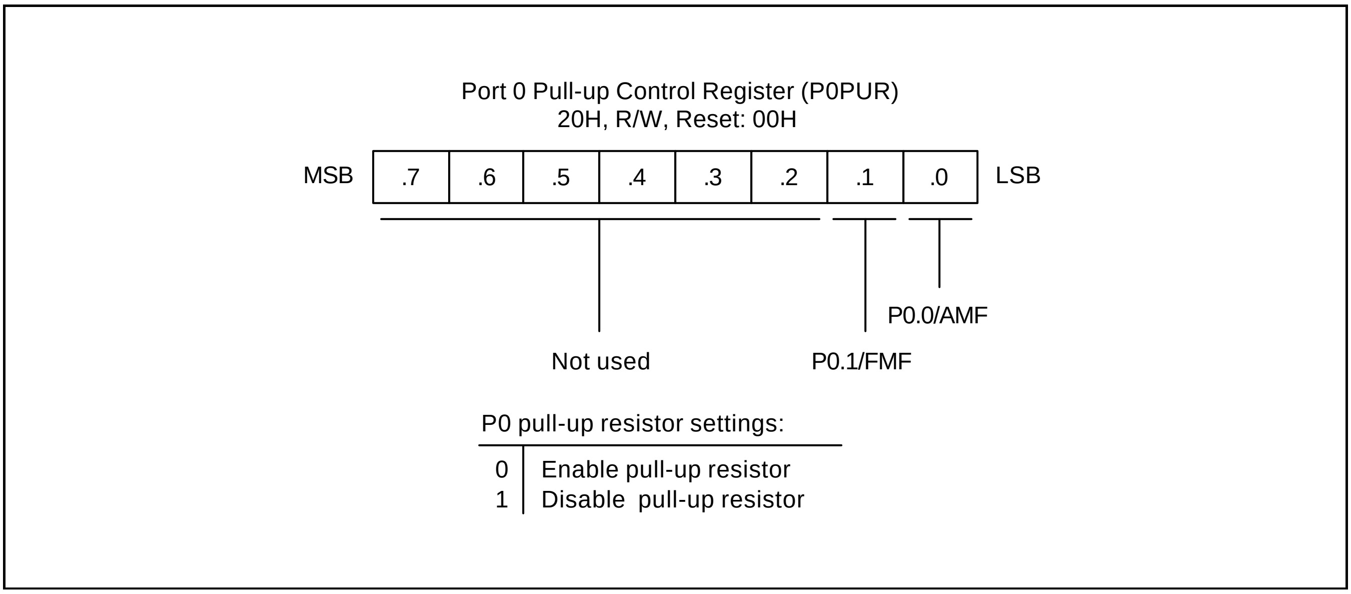
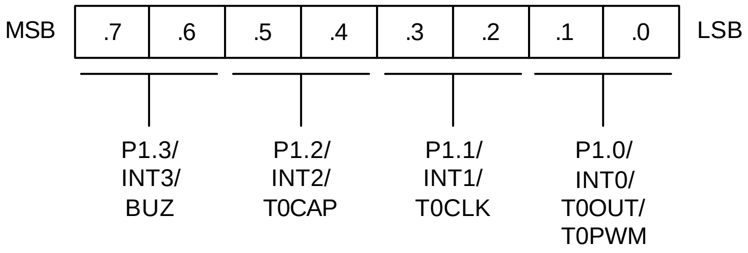

Port 0 Pull-up Control Register (P0PUR)
20H, R/W, Reset: 00H
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
P0.0/AMF
Not used
P0.1/FMF
P0 pull-up resistor settings:
0 Enable pull-up resistor
1 Disable pull-up resistor

Figure 10-1. Port 0 Pull-up Control Register (P0PUR)

PORT 1

Port 1 Control Register (P1CON)
24H, R/W, Reset: 00H

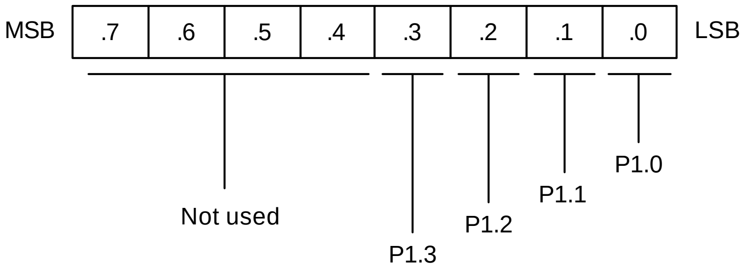


P1CON bit-pair pin configuration settings:

0 0	Schmitt trigger input mode (T0CLK, T0CAP)
0 1	Output mode, open-drain
1 0	Alternative function (T0OUT/T0PWM, BUZ)
1 1	Output mode, push-pull

Figure 10-2. Port 1 Control Register (P1CON)

Port 1 Pull-up Control Register (P1PUR)
25H, R/W, Reset: 00H



P1 pull-up resistor settings:

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

NOTE: A pull-up resistor of port 1 is automatically disabled when the corresponding pin is selected as push-pull output or alternative function.

Figure 10-3. Port 1 Pull-up Control Register (P1PUR)

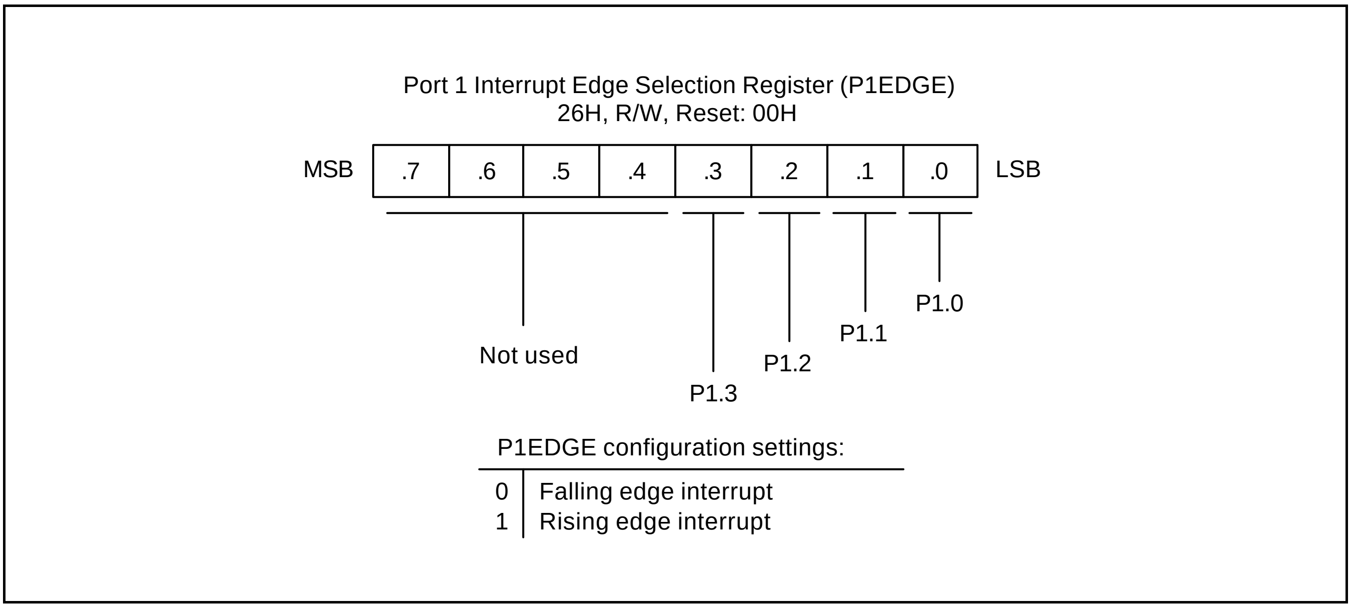
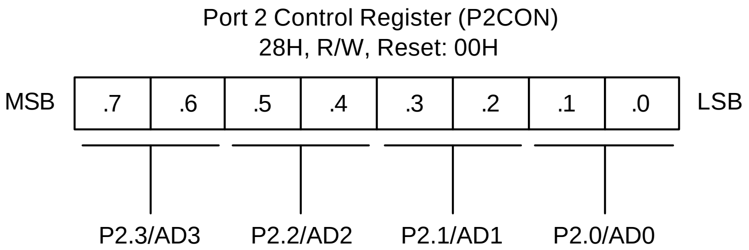


Figure 10-4. Port 1 Interrupt Edge Selection Register (P1EDGE)

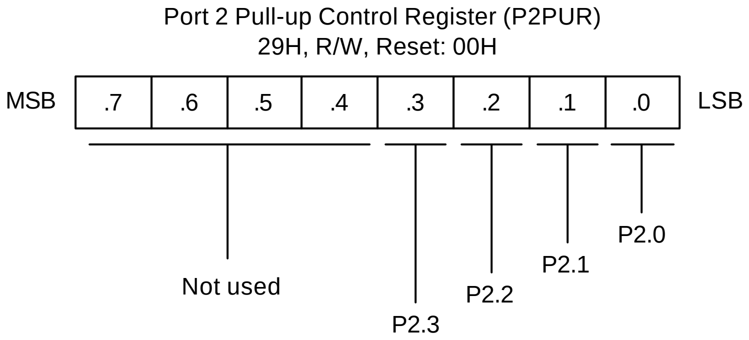
PORT 2



P2CON bit-pair pin configuration settings:

0	0	Schmitt trigger input mode
0	1	Output mode, open-drain
1	0	Alternative function (AD0, AD1, AD2, AD3)
1	1	Output mode, push-pull

Figure 10-5. Port 2 Control Register (P2CON)



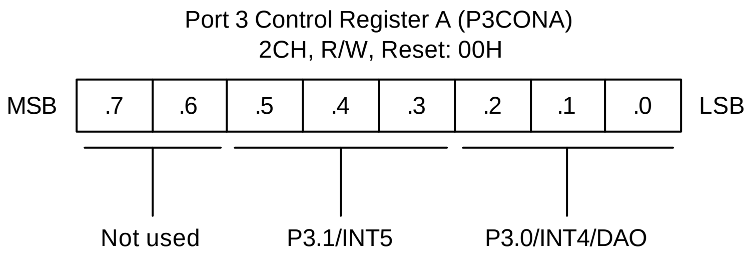
P2 pull-up resistor settings:

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

NOTE: A pull-up resistor of port 2 is automatically disabled when the corresponding pin is selected as push-pull output or alternative function.

Figure 10-6. Port 2 Pull-up Control Register (P2PUR)

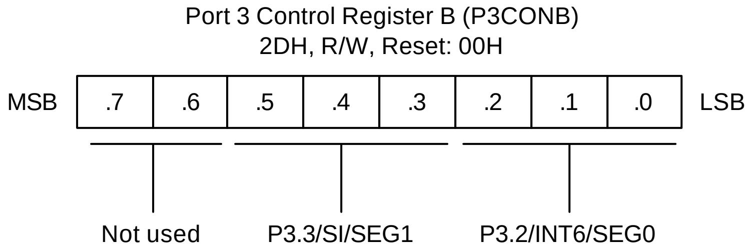
PORT 3



P3CONA bits pin configuration settings:

0 0 0	Schmitt trigger input mode; pull-up; interrupt on falling edge
0 0 1	Schmitt trigger input mode; interrupt on rising edge
0 1 0	Schmitt trigger input mode; interrupt on rising or falling edge
0 1 1	Output mode; push-pull
1 0 0	Output mode; open-drain
1 0 1	Alternative function (DAO)

Figure 10-7. Port 3 Control Register A (P3CONA)

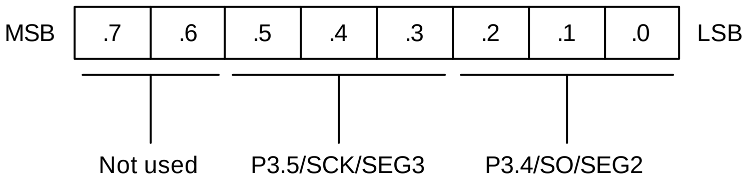


P3CONB bits pin configuration settings:

0 0 0	Schmitt trigger input mode; pull-up; interrupt on falling edge (SI)
0 0 1	Schmitt trigger input mode; interrupt on rising edge (SI)
0 1 0	Schmitt trigger input mode; interrupt on rising or falling edge (SI)
0 1 1	Output mode; push-pull
1 0 0	Output mode; open-drain
1 0 1	Not available

Figure 10-8. Port 3 Control Register B (P3CONB)

Port 3 Control Register C (P3CONC)
2EH, R/W, Reset: 00H

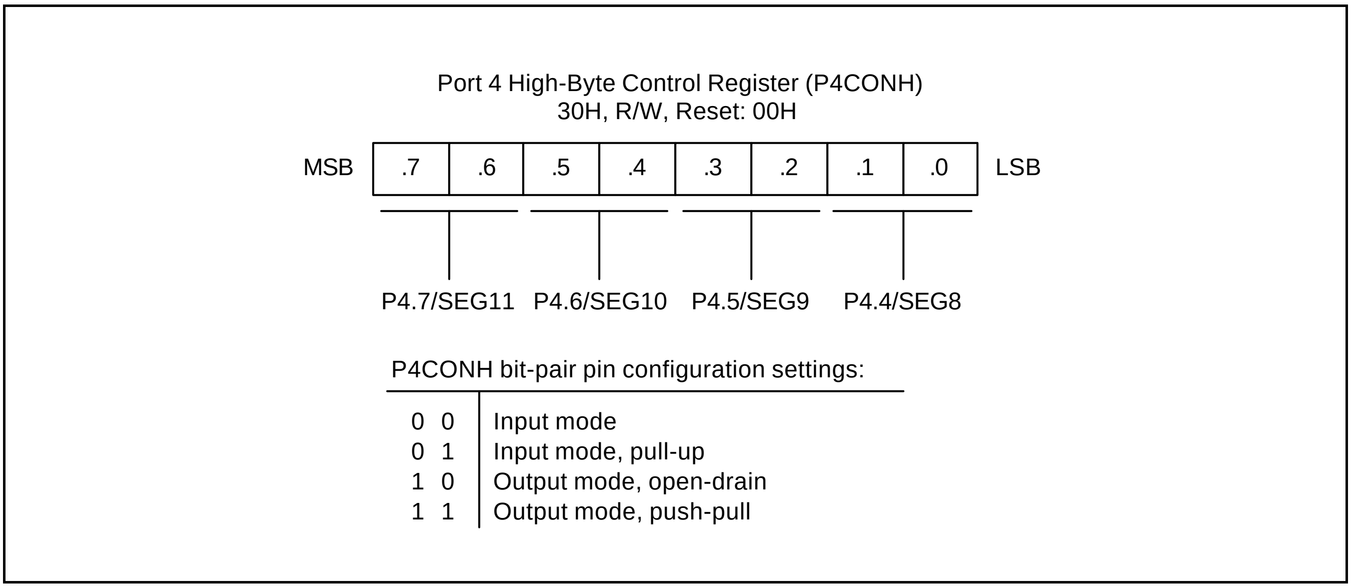


P3CONC bits pin configuration settings:

0 0 0	Schmitt trigger input mode; pull-up (SCK)
0 0 1	Schmitt trigger input mode (SCK)
0 1 0	Not available
0 1 1	Output mode; push-pull
1 0 0	Output mode; open-drain
1 0 1	Alternative function 1 (SO, SCK output: push-pull)
1 1 0	Alternative function 2 (SO, SCK output: open-drain)

Figure 10-9. Port 3 Control Register C (P3CONC)

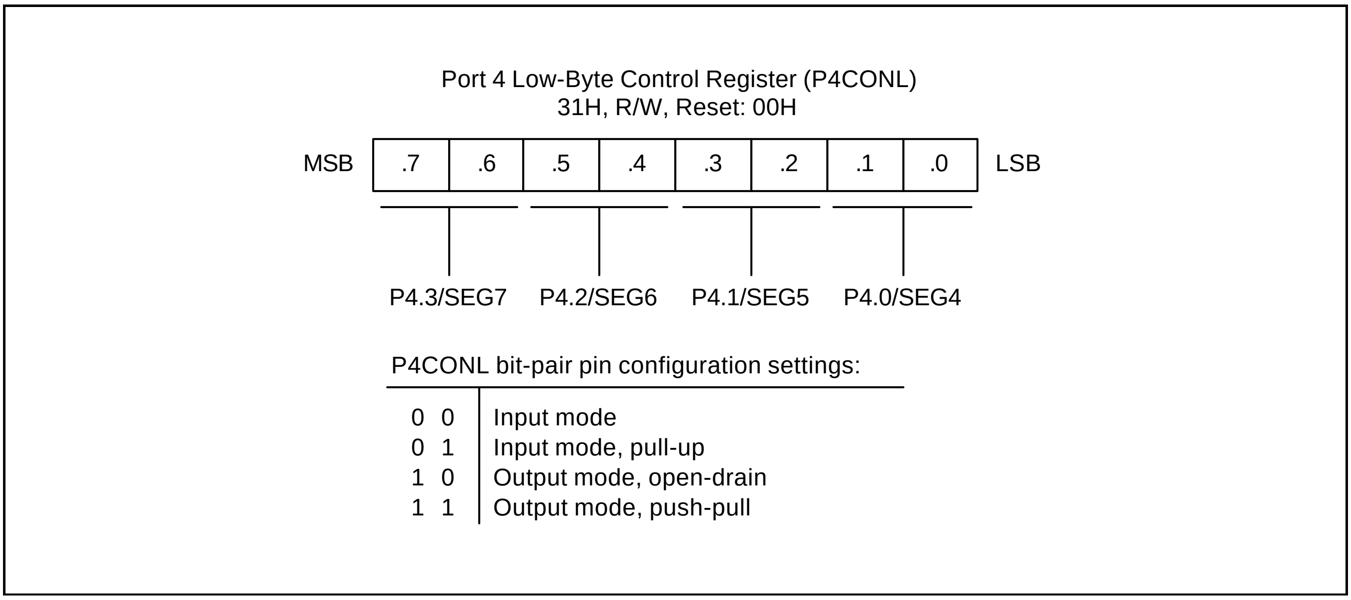
PORT 4



P4CONH bit-pair pin configuration settings:

0 0	Input mode
0 1	Input mode, pull-up
1 0	Output mode, open-drain
1 1	Output mode, push-pull

Figure 10-10. Port 4 High-Byte Control Register (P4CONH)

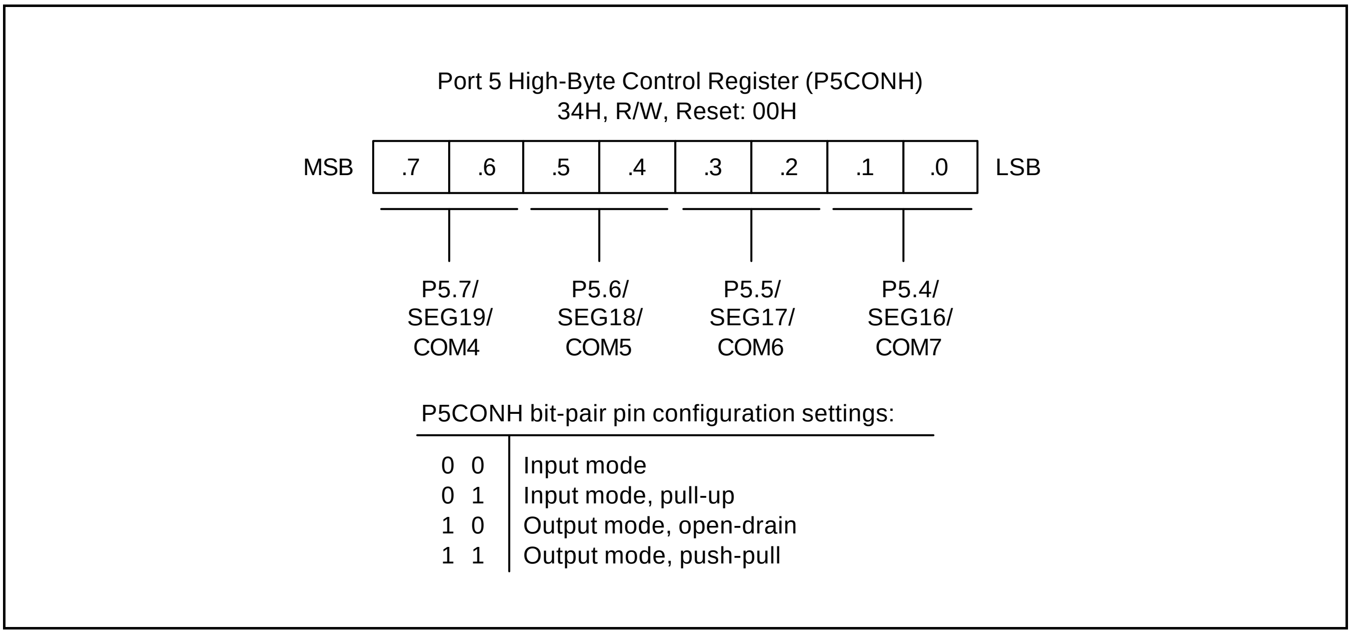


P4CONL bit-pair pin configuration settings:

0 0	Input mode
0 1	Input mode, pull-up
1 0	Output mode, open-drain
1 1	Output mode, push-pull

Figure 10-11. Port 4 Low-Byte Control Register (P4CONL)

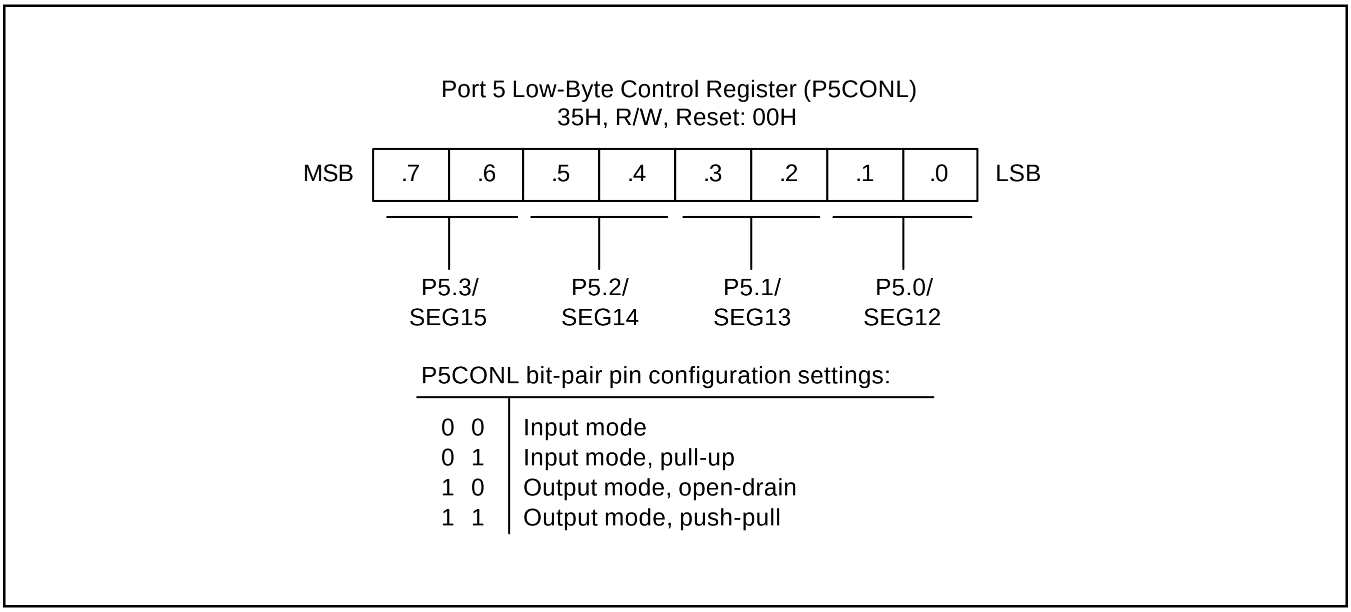
PORT 5



P5CONH bit-pair pin configuration settings:

0 0	Input mode
0 1	Input mode, pull-up
1 0	Output mode, open-drain
1 1	Output mode, push-pull

Figure 10-12. Port 5 High-Byte Control Register (P5CONH)

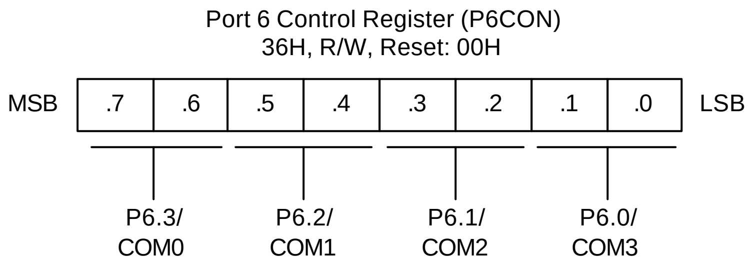


P5CONL bit-pair pin configuration settings:

0 0	Input mode
0 1	Input mode, pull-up
1 0	Output mode, open-drain
1 1	Output mode, push-pull

Figure 10-13. Port 5 Low-Byte Control Register (P5CONL)

PORT 6



P6CON bit-pair pin configuration settings:

0 0	Input mode
0 1	Input mode, pull-up
1 0	Output mode, open-drain
1 1	Output mode, push-pull

Figure 10-14. Port 6 Control Register (P6CON)

NOTES

11 BASIC TIMER/WATCHDOG TIMER

OVERVIEW

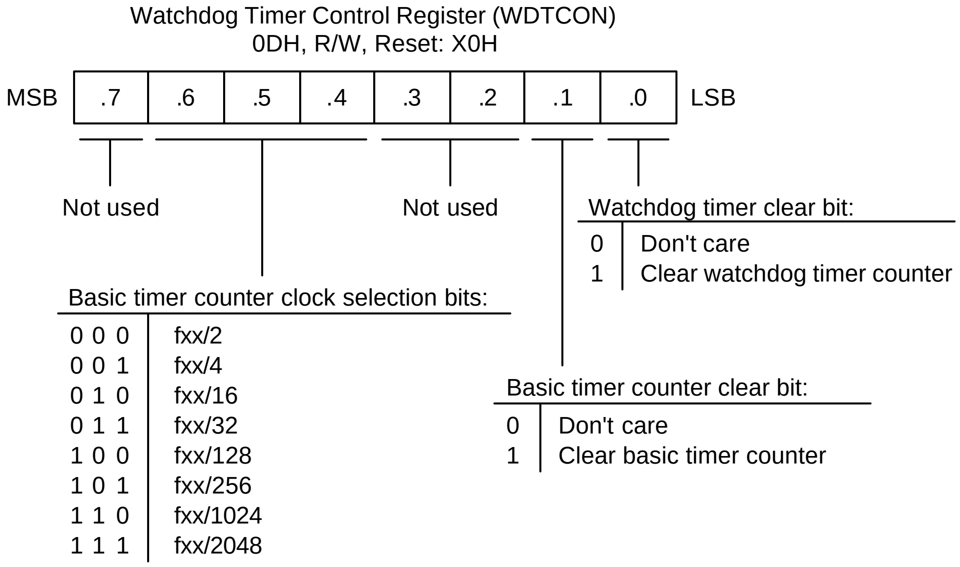
WDTCON controls basic timer clock selection and watchdog timer clear bit.

Basic timer is used in two different ways**:**

- As a clock source to watchdog timer to provide an automatic reset mechanism in the event of a system malfunction (When watchdog function is enabled in ROM code option)
- To signal the end of the required oscillation stabilization interval after a reset or stop mode release.

The reset value of basic timer clock selection bits is decided by the ROM code option. (see the section on ROM code option for details). After reset, programmer can select the basic timer input clock using WDTCON.

When watchdog function is enabled by the ROM code option, programmer must set WDTCON.0 periodically within every 2048 × basic timer input clock time to prevent system reset.



Watchdog Timer Control Register (WDTCON)
0DH, R/W, Reset: X0H

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

.7: Not used

.6-.4: Basic timer counter 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: Not used

.1: Basic timer counter clear bit:

0	Don't care
1	Clear basic timer counter

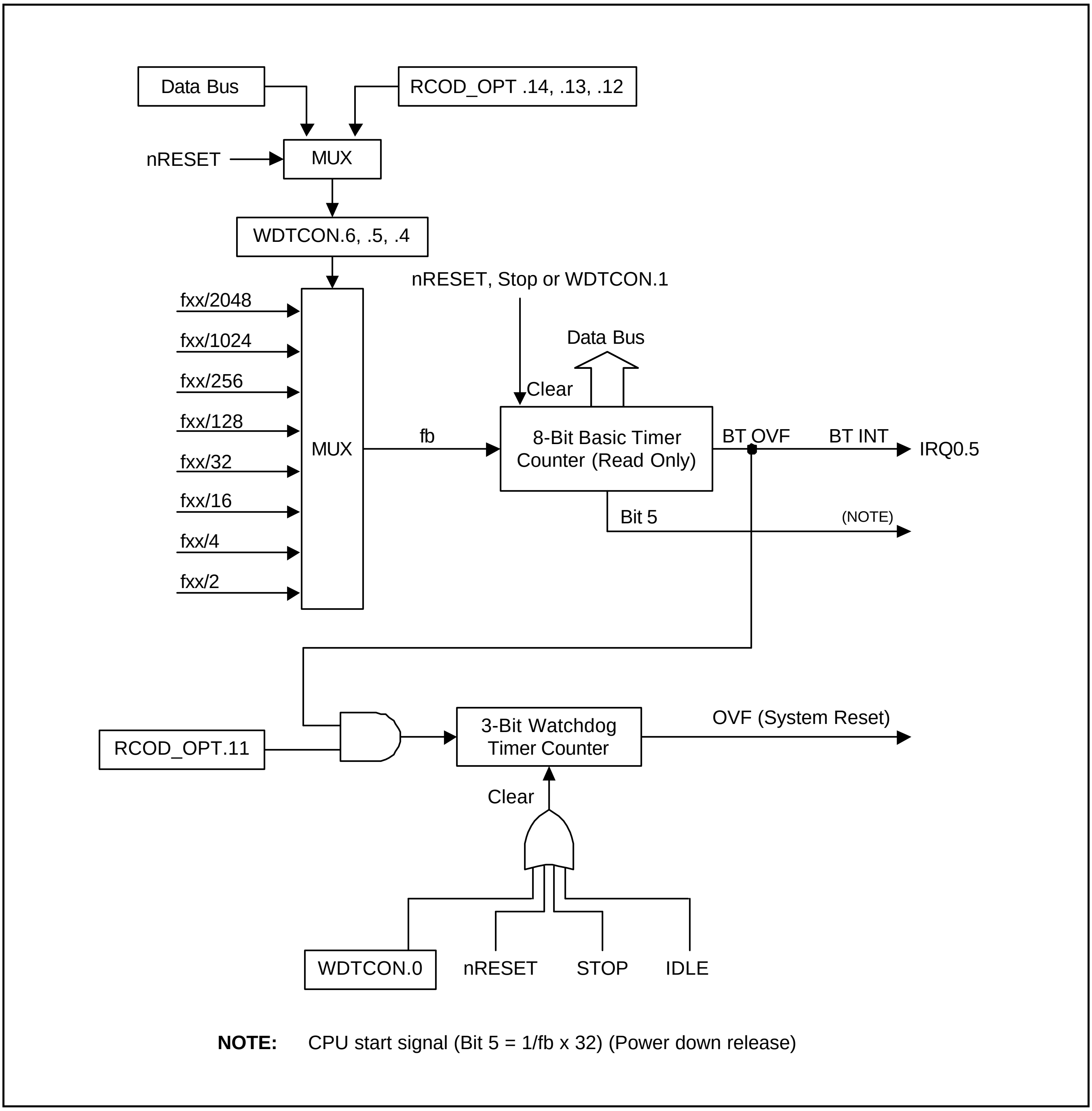
.0: Watchdog timer clear bit:

0	Don't care
1	Clear watchdog timer counter

NOTE: Under the reset operation, basic timer input clock source is effected by RCOD_OPT.14-.12.
After reset, it can be selected basic timer clock source by writing appropriate value to WDTCON.6-.4.

Figure 11-1. Watchdog Timer Control Register (WDTCON)

BLOCK DIAGRAM



NOTE: CPU start signal (Bit 5 = 1/fb x 32) (Power down release)

Figure 11-2. Basic Timer & Watchdog Timer Functional Block Diagram

12 WATCH TIMER

OVERVIEW

The source of watch timer is fx/60 (main osc.) or fxt (sub osc.). The interval of watch timer interrupt can be selected by WTCON.3-2.

Table 12-1. Watch Timer Control Register (WTCON): 8-Bit R/W

Bit Name	Values		Function	Address
WTCON.7–.6	–		Not used.	70H
WTCON.5–.4	0	0	0.47 kHz buzzer (BUZ) signal output	
	0	1	0.94 kHz buzzer (BUZ) signal output	
	1	0	1.87 kHz buzzer (BUZ) signal output	
	1	1	3.75 kHz buzzer (BUZ) signal output	
WTCON.3–.2	0	0	Set watch timer interrupt to 1 sec.	
	0	1	Set watch timer interrupt to 0.1 sec.	
	1	0	Set watch timer interrupt to 0.5 sec.	
	1	1	Set watch timer interrupt to 50 msec.	
WTCON.1	0		Select fx/60 as the watch timer clock.	
	1		Select fxt (sub osc) as the watch timer clock.	
WTCON.0	0		Disable watch timer: clear frequency dividing circuits.	
	1		Enable watch timer.	

NOTES:
1. The main clock frequency (fx) is assumed to be 4.5 MHz.
2. The watch timer clock frequency (fw) is assumed to be 75 kHz.

WATCH TIMER CIRCUIT DIAGRAM

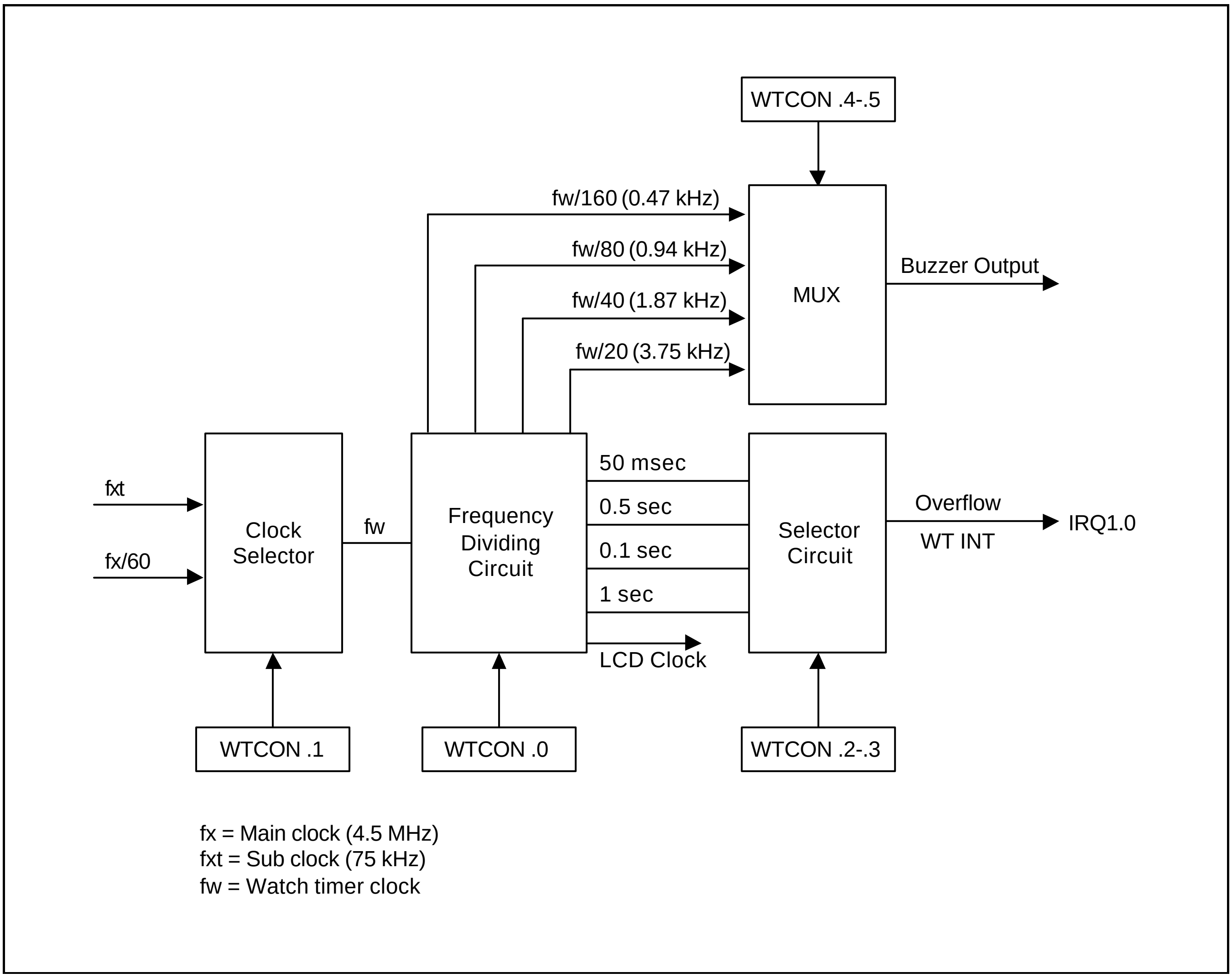


Figure 12-1. Watch Timer Circuit Diagram

13 16-BIT TIMER 0

OVERVIEW

The 16-bit timer 0 is an 16-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 T0OUT pin)

— Capture input mode with a rising or falling edge trigger at the T0CAP pin

— PWM mode (T0PWM)

Timer 0 has the following functional components:

— Clock frequency divider (fxx divided by 4096, 512, 64, 8, or 1) with multiplexer

— External clock input pin (T0CLK)

— 16-bit counter (T0CNTH/L), 16-bit comparator, and 16-bit reference data register (T0DATAH/L)

— I/O pins for capture input (T0CAP), or PWM or match output (T0PWM, T0OUT)

— Timer 0 overflow interrupt (IRQ0.1) and match/capture interrupt (IRQ0.0) generation

— Timer 0 control register, T0CON (40H, read/write)

FUNCTION DESCRIPTION

Timer 0 Interrupts (IRQ0.0, IRQ0.1)

The timer 0 module can generate two interrupts, the timer 0 overflow interrupt (T0OVF), and the timer 0 match/capture interrupt (T0INT). T0OVF is interrupt level IRQ0.1. T0INT belongs to interrupt level IRQ0.0.

Interval Timer Function

In interval timer mode, a match signal is generated and T0OUT is toggled when the counter value is identical to the value written to the T0 reference data register, T0DATAH/L. The match signal generates a timer 0 match interrupt (T0INT) and clears the counter.

If, for example, you write the value 0010H to T0DATAH/L and 04H to T0CON, the counter will increment until it reaches 0010H. At this point, the T0 interrupt request is generated, 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 FFFFH, and then repeat the incrementing from 0000H. Whenever an overflow is occurred, 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 Low level as long as the reference data value is less than or equal to (≤) the counter value and then pulse is held to High level for as long as the data value is greater than (>) the counter value. One pulse width is equal to t_{CLK}

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 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 1 control register, P1CON, (24H). When P1CON.5–.4 is 00, the T0CAP input or normal input is selected .When P1CON.5–.4 is set to 11, normal output is selected.

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 T0DATAH/L, 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.

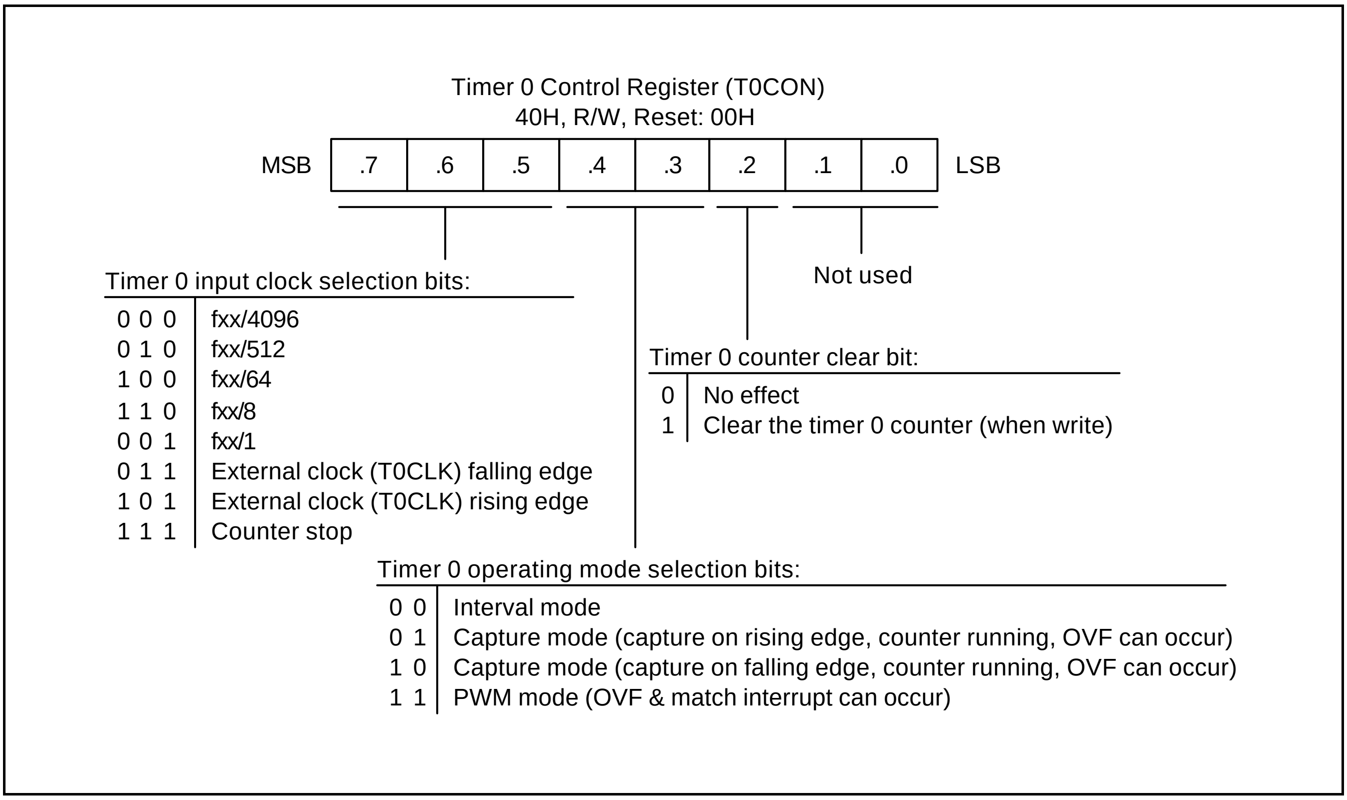
TIMER 0 CONTROL REGISTER (T0CON)

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, T0CNTH/L

T0CON is located at address 40H, and is read/written addressable.

A reset clears T0CON to '00H'. This sets timer 0 to normal interval timer mode, and selects an input clock frequency of fxx/4096. To disable the counter operation, please set T0CON.7–.5 to 111B. You can clear the timer 0 counter at any time during normal operation by writing a "1" to T0CON.2.



Timer 0 input clock selection bits:

0 0 0	fxx/4096
0 1 0	fxx/512
1 0 0	fxx/64
1 1 0	fxx/8
0 0 1	fxx/1
0 1 1	External clock (T0CLK) falling edge
1 0 1	External clock (T0CLK) rising edge
1 1 1	Counter stop

Timer 0 counter clear bit:

0	No effect
1	Clear the timer 0 counter (when write)

Timer 0 operating mode selection bits:

0 0	Interval mode
0 1	Capture mode (capture on rising edge, counter running, OVF can occur)
1 0	Capture mode (capture on falling edge, counter running, OVF can occur)
1 1	PWM mode (OVF & match interrupt can occur)

Figure 13-1. Timer 0 Control Register (T0CON)

BLOCK DIAGRAM

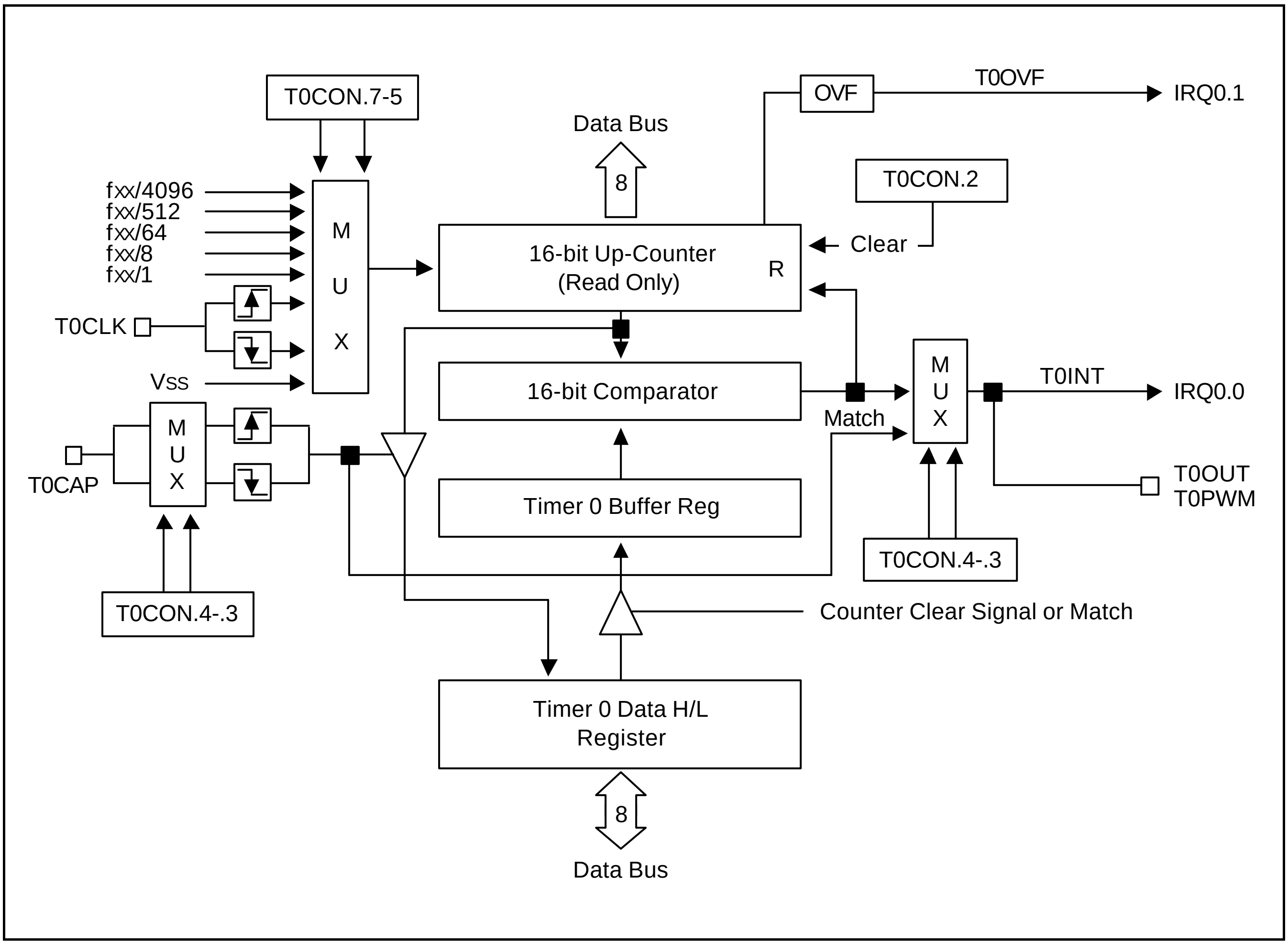


Figure 13-2. Timer 0 Functional Block Diagram

Timer 0 Counter High-Byte Register (T0CNTH)
43H, Read only, Reset: 00H

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

Timer 0 Counter Low-Byte Register (T0CNTL)
44H, Read only, Reset: 00H

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

Timer 0 Data High-Byte Register (T0DATAH)
41H, R/W, Reset: FFH

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

Timer 0 Data Low-Byte Register (T1DATAL)
42H, R/W, Reset: FFH

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

Figure 13-3. Timer 0 Counter and Data Register (T0CNTH/L, T0DATAH/L)

NOTES

14 8-BIT TIMER 1

OVERVIEW

The 8-bit timer 1 is an 8-bit general-purpose timer. Timer 1 has the interval timer mode by using the appropriate T1CON setting.

Timer 1 has the following functional components:

— Clock frequency divider (fxx divided by 1024, 256, 64, 8, or 1) with multiplexer

— 8-bit counter (T1CNT), 8-bit comparator, and 8-bit reference data register (T1DATA)

— Timer 1 interrupt (IRQ0.2) generation

— Timer 1 control register, T1CON (48H, read/write)

FUNCTION DESCRIPTION

Interval Timer Function

The timer 1 module can generate an interrupt, the timer 1 match interrupt (T1INT). T1INT belongs to interrupt level IRQ0.2.

In interval timer mode, a match signal is generated when the counter value is identical to the values written to the T1 reference data registers, T1DATA. The match signal generates a timer 1 match interrupt (T1INT) and clears the counter.

If, for example, you write the value 10H to T1DATA and 0CH to T1CON, the counter will increment until it reaches 10H. At this point, the T1 interrupt request is generated, the counter value is reset, and counting resumes.

TIMER 1 CONTROL REGISTER (T1CON)

You use the timer 1 control register, T1CON, to

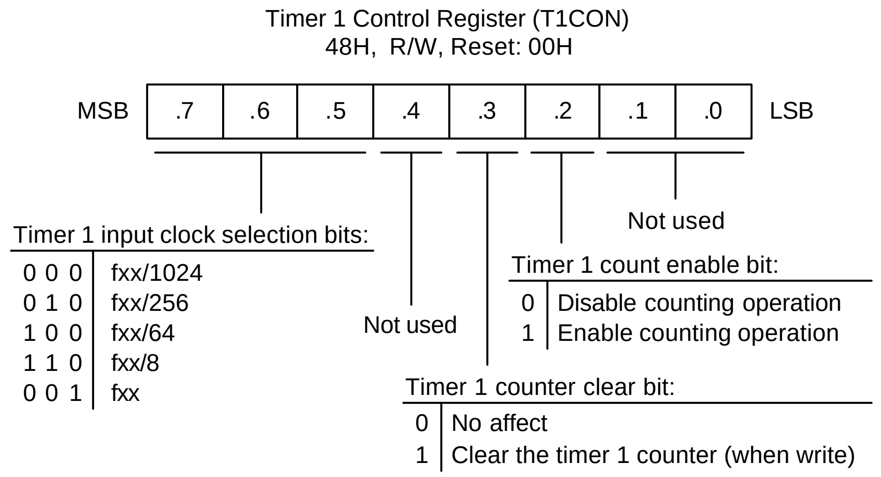
— Enable the timer 1 operating (interval timer)

— Select the timer 1 input clock frequency

— Clear the timer 1 counter, T1CNT

T1CON is located, at address 48H, and is read/written addressable.

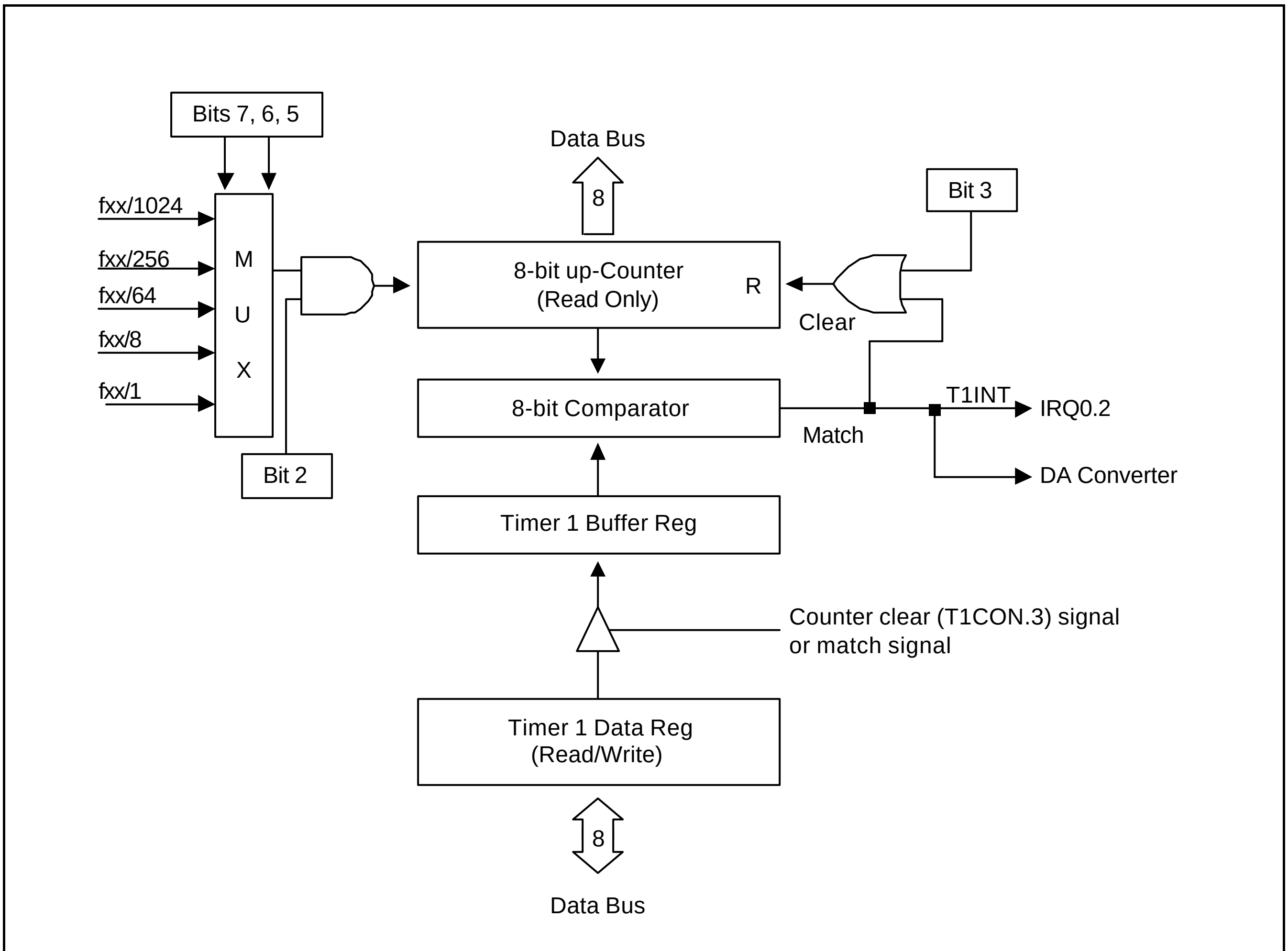
A reset clears T1CON to "00H". This sets timer 1 to disable interval timer mode, and selects an input clock frequency of fxx/1024. You can clear the timer 1 counter at any time during normal operation by writing a "1" to T1CON.3


Timer 1 Control Register (T1CON)
48H, R/W, Reset: 00H
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Timer 1 input clock selection bits:
0 0 0 fxx/1024
0 1 0 fxx/256
1 0 0 fxx/64
1 1 0 fxx/8
0 0 1 fxx
Not used
Not used
Timer 1 count enable bit:
0 Disable counting operation
1 Enable counting operation
Timer 1 counter clear bit:
0 No affect
1 Clear the timer 1 counter (when write)

NOTE: For normal operation T1CON.2 bit must be set 1.

Figure 14-1. Timer 1 Control Register (T1CON)

BLOCK DIAGRAM



NOTE: T1CON.3 bit is cleared automatically.

Figure 14-2. Timer 1 Functional Block Diagram

Timer 1 Counter (T1CNT)
4AH, Read only, Reset: 00H

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

Figure 14-3. Timer 1 Counter (T1CNT)

Timer 1 Data Register (T1DATA)
49H, R/W, Reset: FFH

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

Figure 14-4. Timer 1 Data Register (T1DATA)

15 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, SCK, SI) by loading the appropriate value to the P3CONB and P3CONC registers, 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.

SIO CONTROL REGISTER (SIOCON)

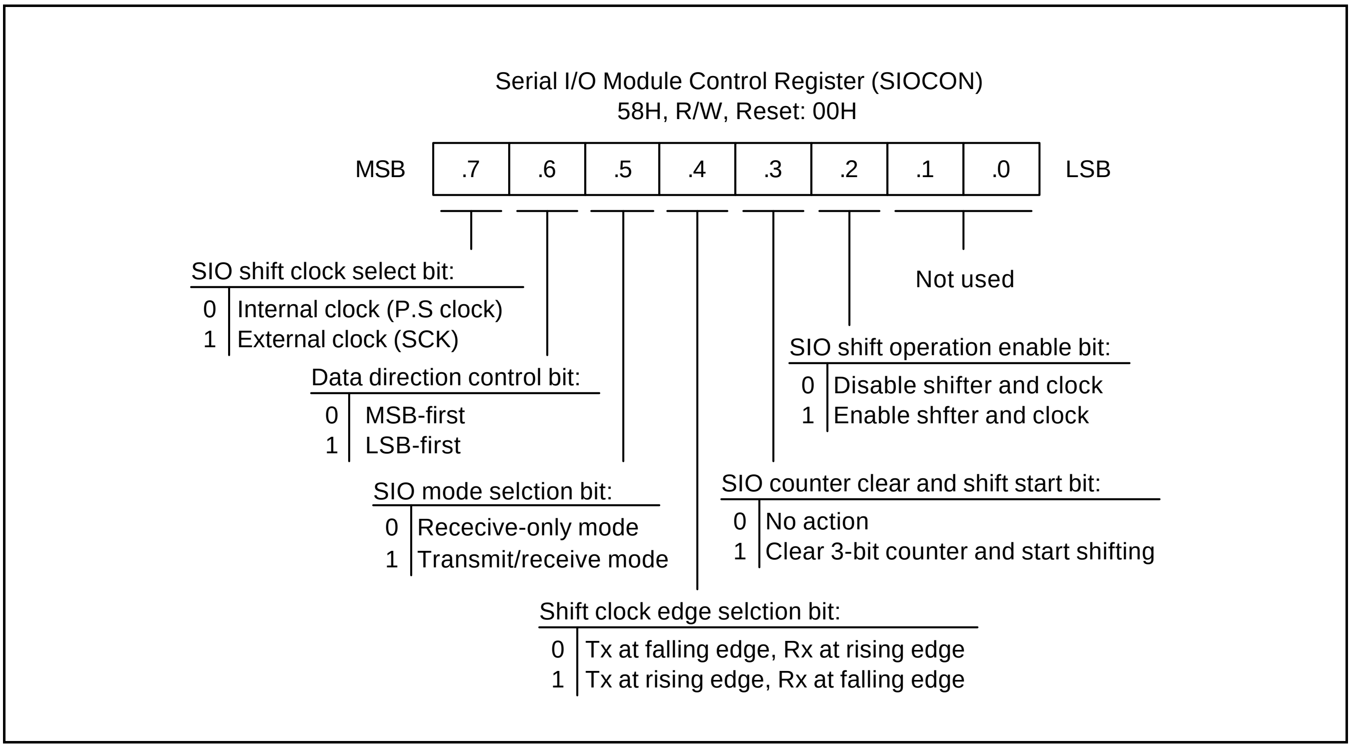

Serial I/O Module Control Register (SIOCON)
58H, R/W, Reset: 00H
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
SIO shift clock select bit:
0 Internal clock (P.S clock)
1 External clock (SCK)
Data direction control bit:
0 MSB-first
1 LSB-first
SIO mode selction bit:
0 Rececive-only mode
1 Transmit/receive mode
Shift clock edge selction bit:
0 Tx at falling edge, Rx at rising edge
1 Tx at rising edge, Rx at falling edge
SIO counter clear and shift start bit:
0 No action
1 Clear 3-bit counter and start shifting
SIO shift operation enable bit:
0 Disable shifter and clock
1 Enable shfter and clock
Not used

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

SIO PRE-SCALER REGISTER (SIOPS)

The value stored in the SIO pre-scaler registers, SIOPS, lets you determine the SIO clock rate (baud rate) as follows:

Baud rate = Input clock(fxx/4)/(Pre-scaler value + 1), or, SCK input clock

where fxx is a selected clock.

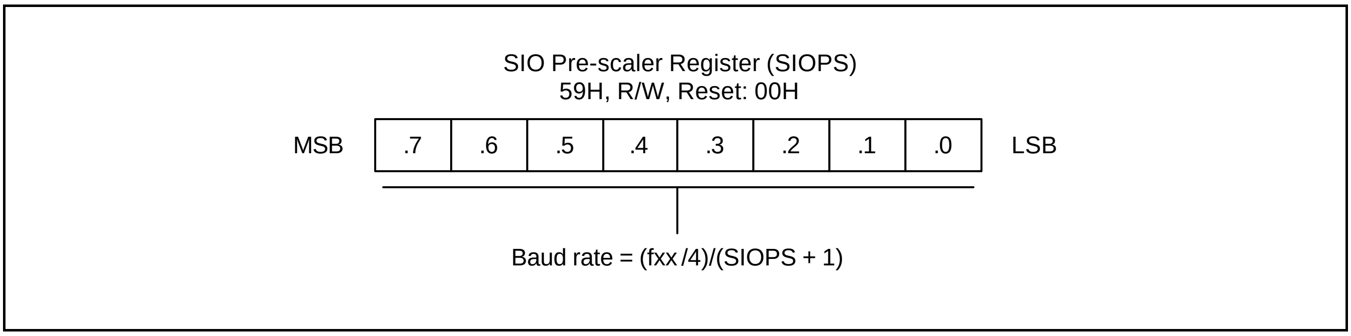

SIO Pre-scaler Register (SIOPS)
59H, R/W, Reset: 00H
MSB
.7
.6
.5
.4
.3
.2
.1
.0
LSB
Baud rate = (fxx /4)/(SIOPS + 1)

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

BLOCK DIAGRAM

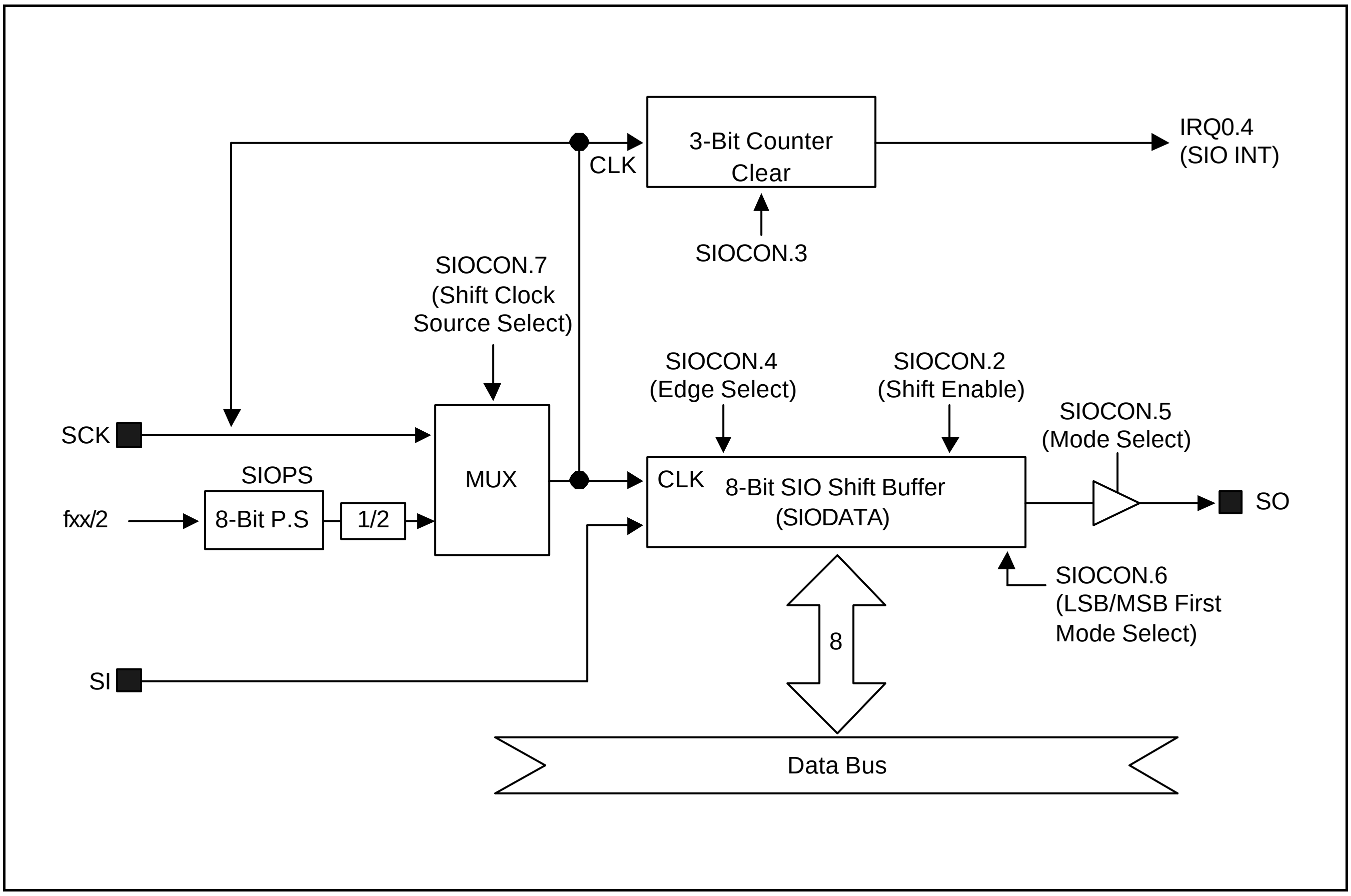


Figure 15-3. SIO Function Block Diagram

SERIAL I/O TIMING DIAGRAM

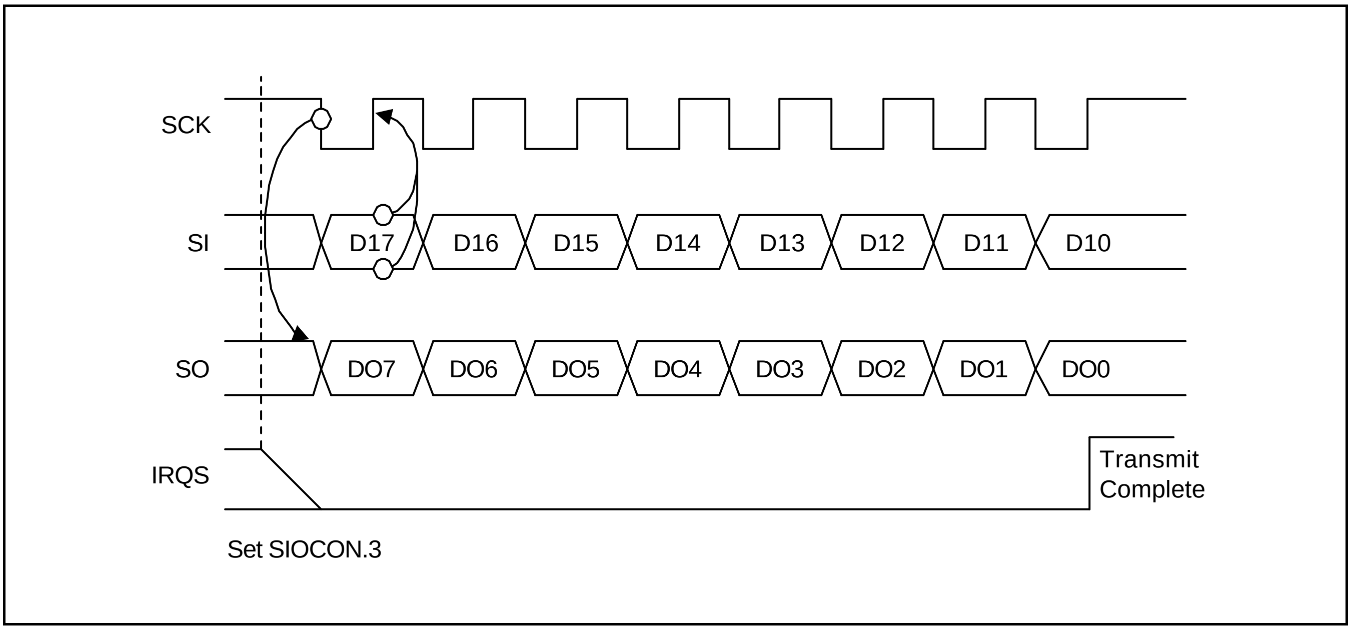


Figure 15-4. Serial I/O Timing in Transmit/Receive Mode (Tx at falling, SIOCON.4 = 0)

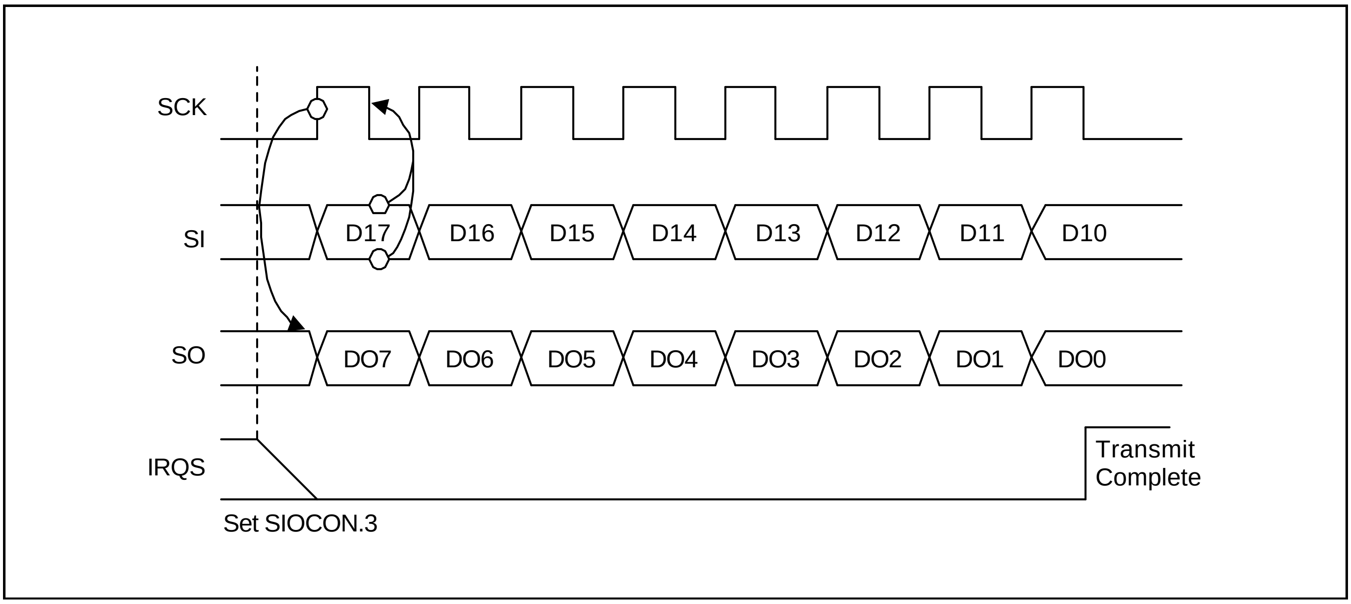


Figure 15-5. Serial I/O Timing in Transmit/Receive Mode (Tx at rising, SIOCON.4 = 1)

16 BATTERY LEVEL DETECTOR

OVERVIEW

The S3CK318/FK318 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 3 kinds of voltage.

2.2 V, 2.4 V or 2.6 V (VDD 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.

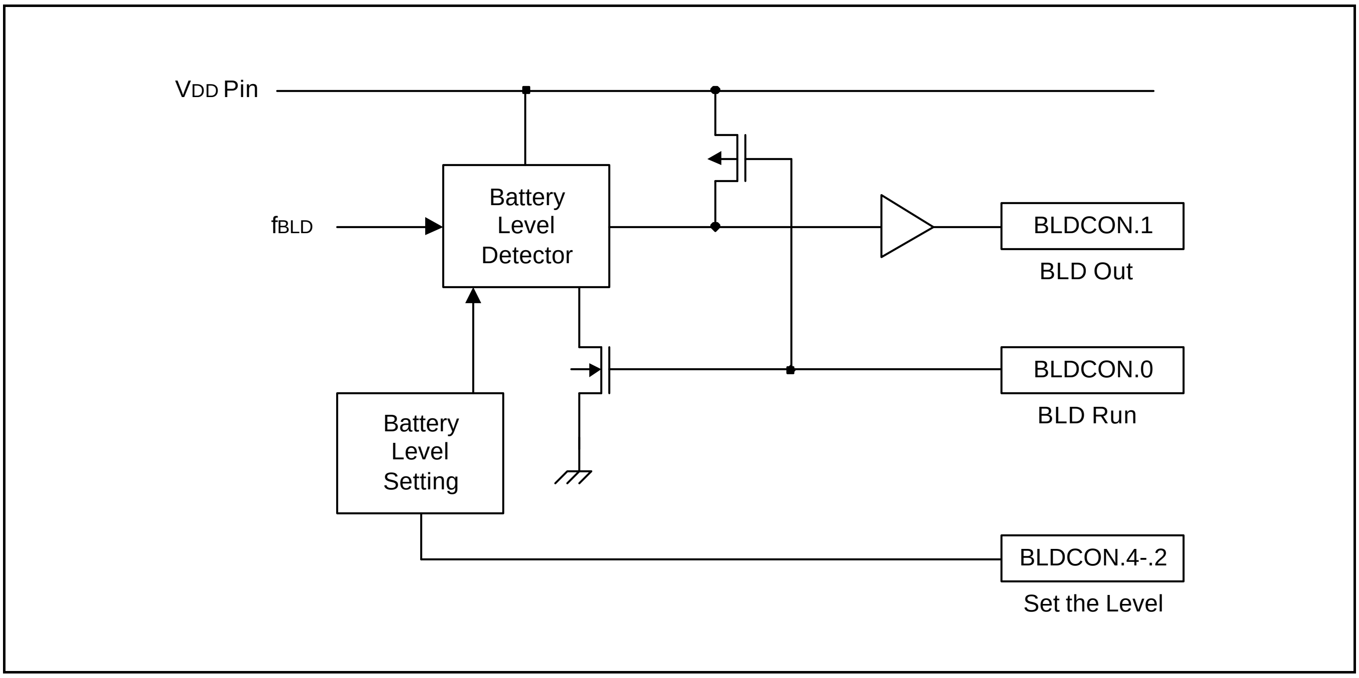
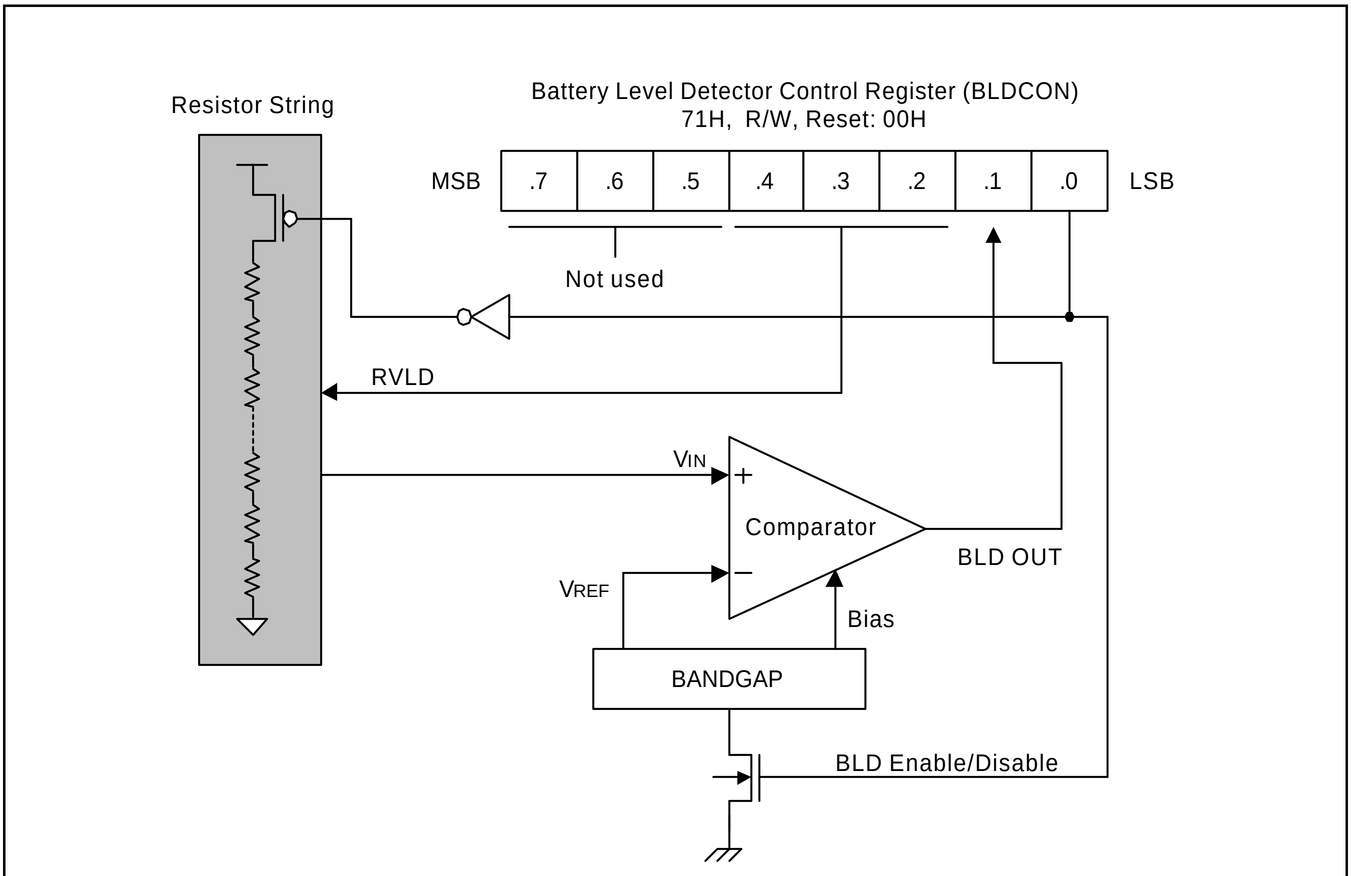


Figure 16-1. Block Diagram for Battery Level Detect

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 3 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 3 levels.



NOTES:
1. The Reset value of BLDCON is #00H.
2. VREF is about 1 volt.

Figure 16-2. Battery Level Detector Circuit and Control Register

Table 16-1. BLDCON Value and Detection Level

VLDCON .4–.2	V_{BLD}
1 0 0	2.2 V
1 0 1	2.4 V
1 1 0	2.6 V
1 1 1	–

17 LCD CONTROLLER/DRIVER

OVERVIEW

The S3CK318/FK318 microcontroller can directly drive an up-to-128-dot (16 segments × 8 commons) LCD panel. Its LCD block has the following components:

- LCD controller/driver
- Display RAM for storing display data
- 4 common/segment output pins (COM4/SEG19–COM7/SEG16)
- 16 segment output pins (SEG0–SEG15)
- 4 common output pins (COM0–COM3)
- Internal resistor circuit for LCD bias

To use the LCD controller/driver, the watch timer must be enabled by setting WTCON.0 to "1" because fLCD for LCD controller/driver clock source is supplied by the watch timer.

When a sub block is selected as the watch timer clock source and watch timer is enabled, the LCD display can be enabled even during stop and idle modes.

The LCD clock source speed, duty, bias LCD contrast level, and display on or off are determined by bit settings in the LCD control register, LMOD, at address 60H.

Data written to the LCD display RAM can be transferred to the segment signal pins automatically without program control.

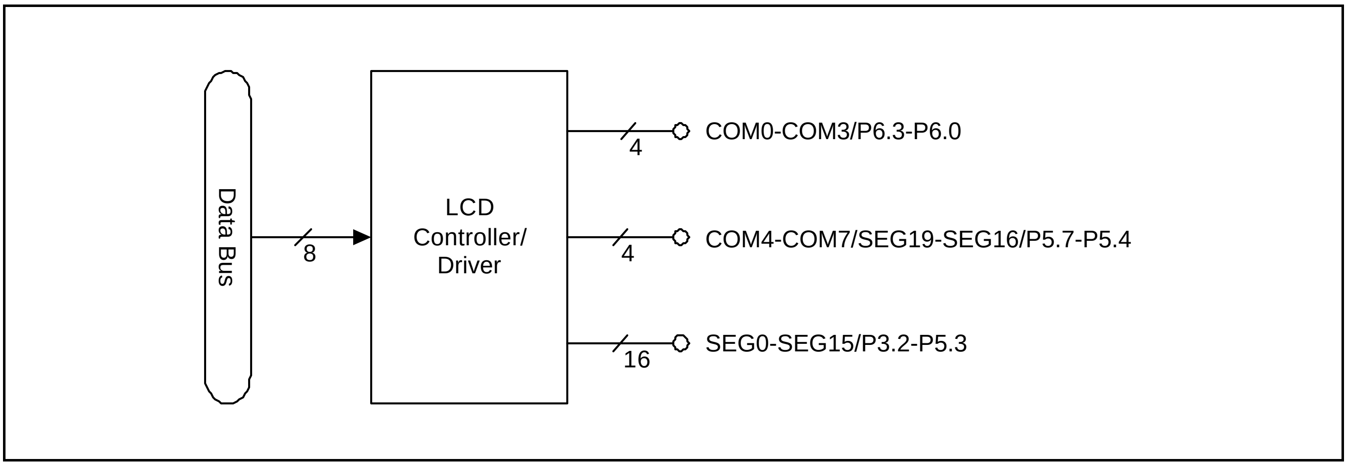


Figure 17-1. LCD Function Diagram

LCD CIRCUIT DIAGRAM

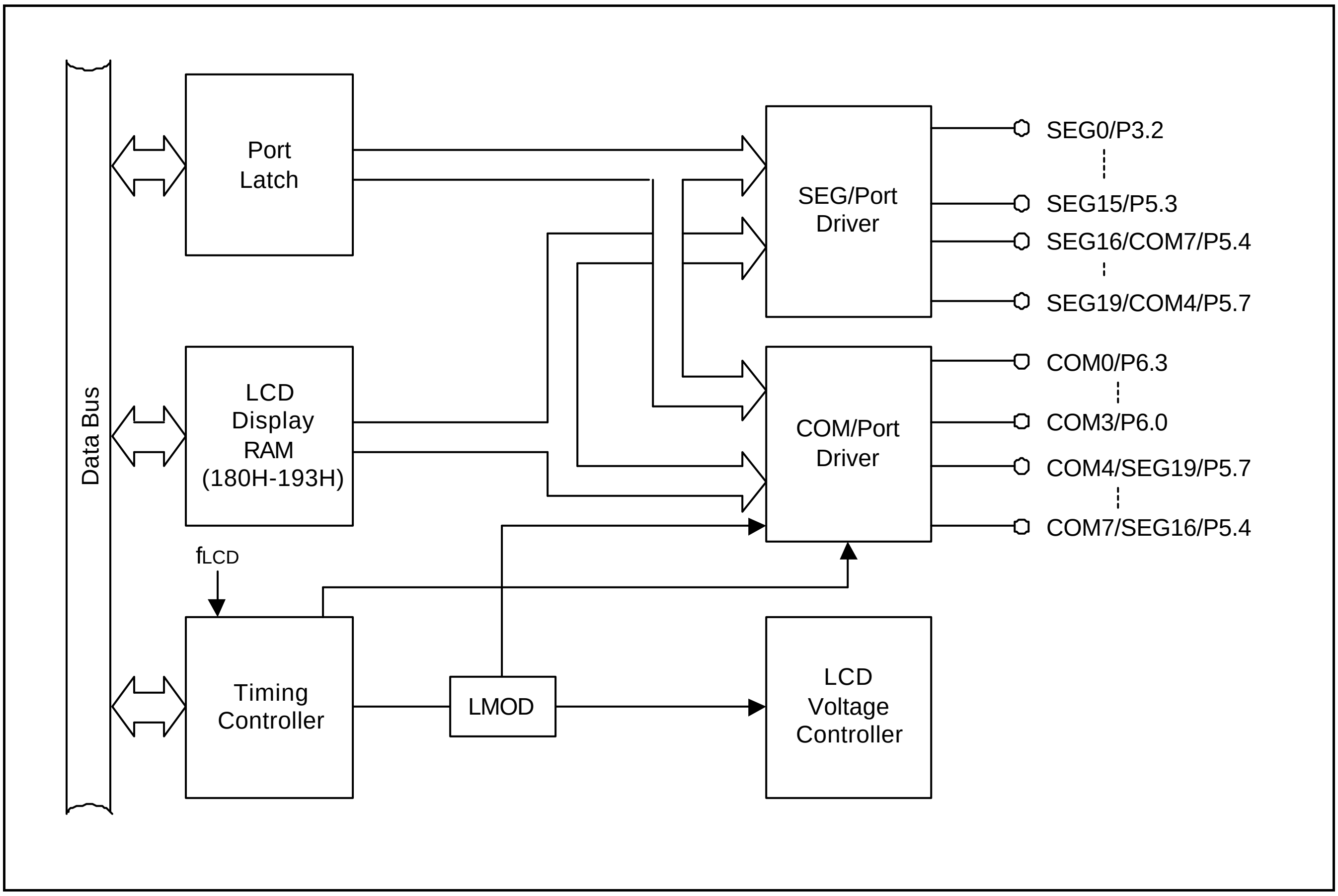


Figure 17-2. LCD Circuit Diagram

LCD RAM ADDRESS AREA

RAM addresses of page 1 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–SEG19, 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.

1/8 Duty Mode

COM	Bit	SEG0	SEG1	SEG2	SEG3	------	SEG13	SEG14	SEG15
COM0 COM1 COM2 COM3 COM4 COM5 COM6 COM7	.0 .1 .2 .3 .4 .5 .6 .7	180H	181H	182H	183H	------	18DH	18EH	18FH

1/4, 1/3 Duty Mode

COM	Bit	SEG0	SEG1	SEG2	SEG3	------	SEG17	SEG18	SEG19
COM0 COM1 COM2 COM3	.0 .1 .2 .3	180H	181H	182H	183H	------	191H	192H	193H
	.4 .5 .6 .7								

Figure 17-3. LCD Display Data RAM Organization

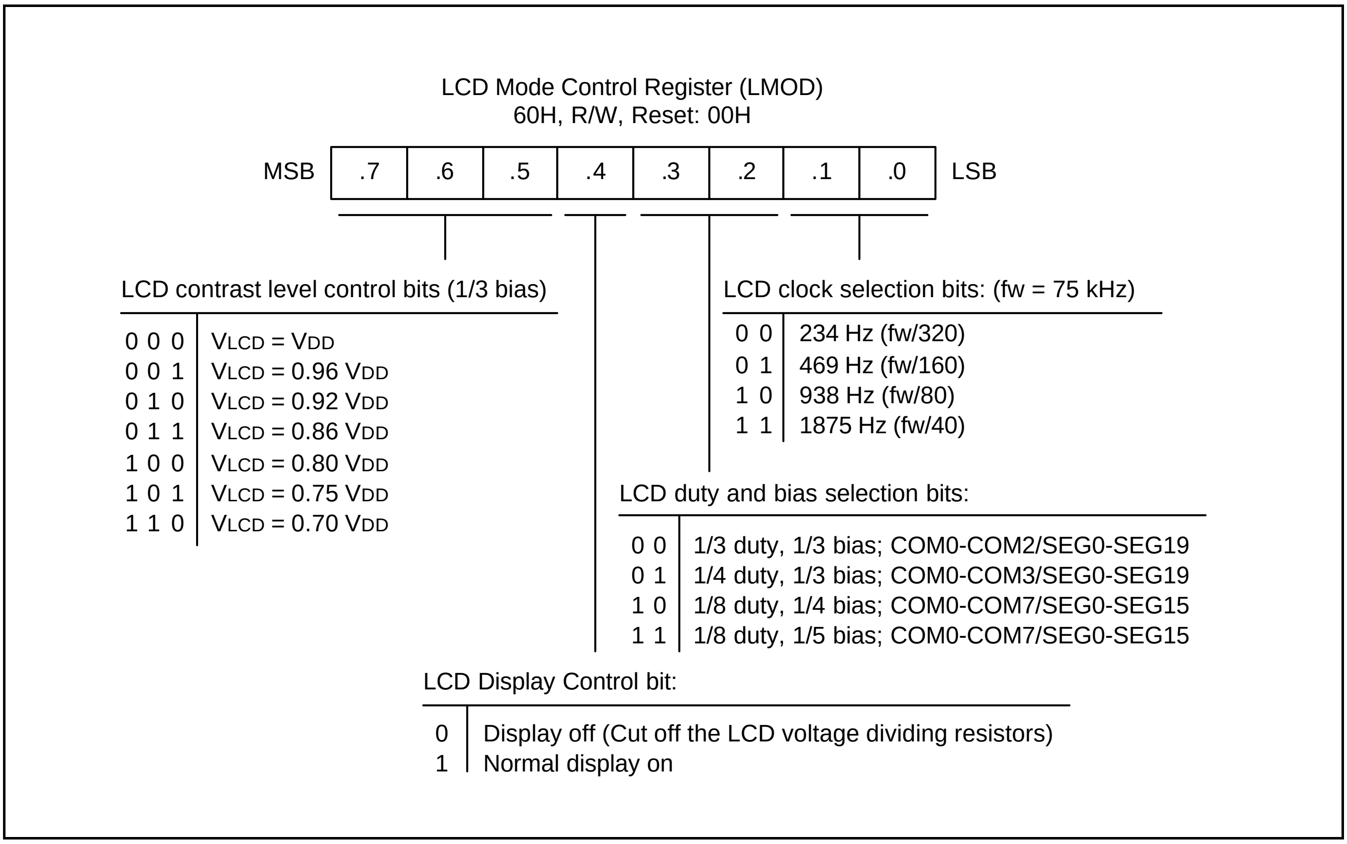
LCD MODE CONTROL REGISTER (LMOD)

The LCD mode control register for the LCD controller/driver is called LMOD. LMOD is located at address 60H. It has the following control functions:

- LCD duty selection
- LCD bias selection
- LCD clock selection
- LCD contrast level control
- LCD display control

The LCD mode control register, LMOD is used to turn the LCD display on/off, to select duty and bias, to select LCD clock, and to select dividing resistors for bias. Following a reset all LMOD values are cleared to "0". This turns off the LCD display, select 1/3 bias and 1/3 duty, and select VDD for LCD panel power (V_{LCD}).

The LCD clock signal determines the frequency of COM signal scanning of each segment output. This is also referred as the frame frequency. Since the LCD clock signal is generated by the watch timer clock (fw), the watch timer must be enabled when the LCD display is turned on.



LCD Mode Control Register (LMOD)
60H, R/W, Reset: 00H

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

LCD contrast level control bits (1/3 bias)

0 0 0	$V_{LCD} = V_{DD}$
0 0 1	$V_{LCD} = 0.96\ V_{DD}$
0 1 0	$V_{LCD} = 0.92\ V_{DD}$
0 1 1	$V_{LCD} = 0.86\ V_{DD}$
1 0 0	$V_{LCD} = 0.80\ V_{DD}$
1 0 1	$V_{LCD} = 0.75\ V_{DD}$
1 1 0	$V_{LCD} = 0.70\ V_{DD}$

LCD clock selection bits: (fw = 75 kHz)

0 0	234 Hz (fw/320)
0 1	469 Hz (fw/160)
1 0	938 Hz (fw/80)
1 1	1875 Hz (fw/40)

LCD duty and bias selection bits:

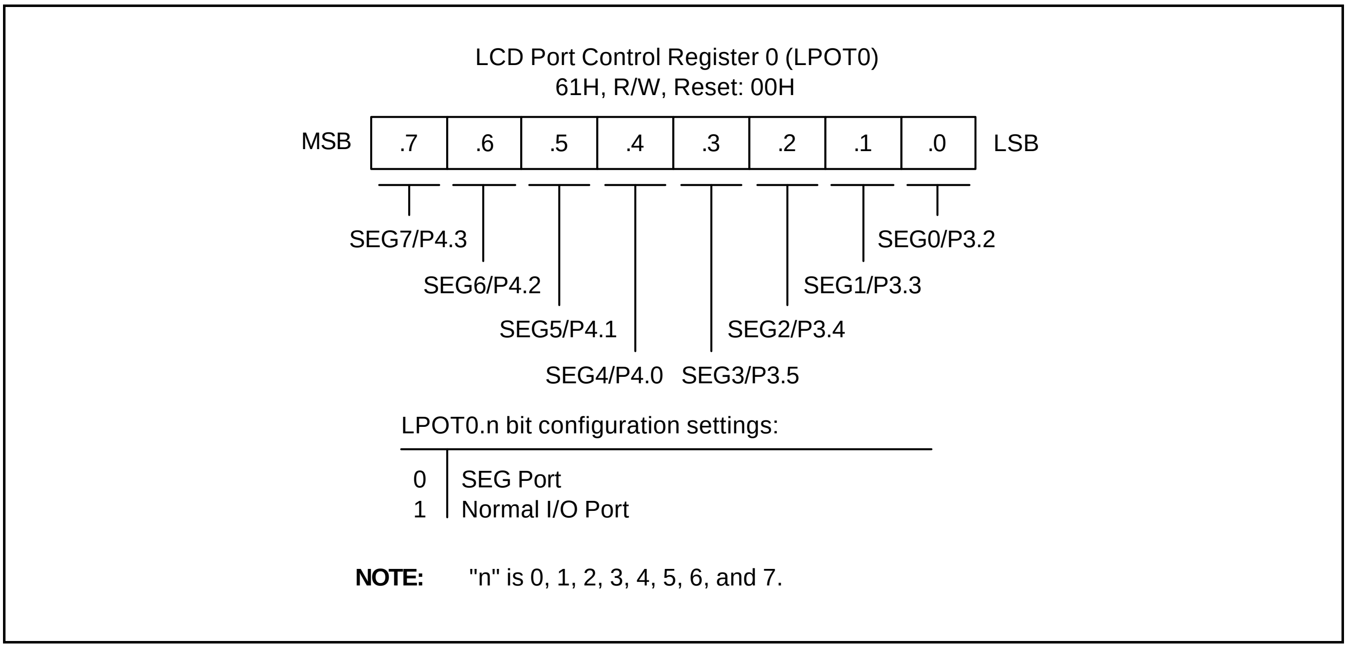
0 0	1/3 duty, 1/3 bias; COM0-COM2/SEG0-SEG19
0 1	1/4 duty, 1/3 bias; COM0-COM3/SEG0-SEG19
1 0	1/8 duty, 1/4 bias; COM0-COM7/SEG0-SEG15
1 1	1/8 duty, 1/5 bias; COM0-COM7/SEG0-SEG15

LCD Display Control bit:

0	Display off (Cut off the LCD voltage dividing resistors)
1	Normal display on

Figure 17-4. LCD Mode Control Register (LMOD)

LCD PORT CONTROL REGISTERS (LPOT0, LPOT1, LPOT2)

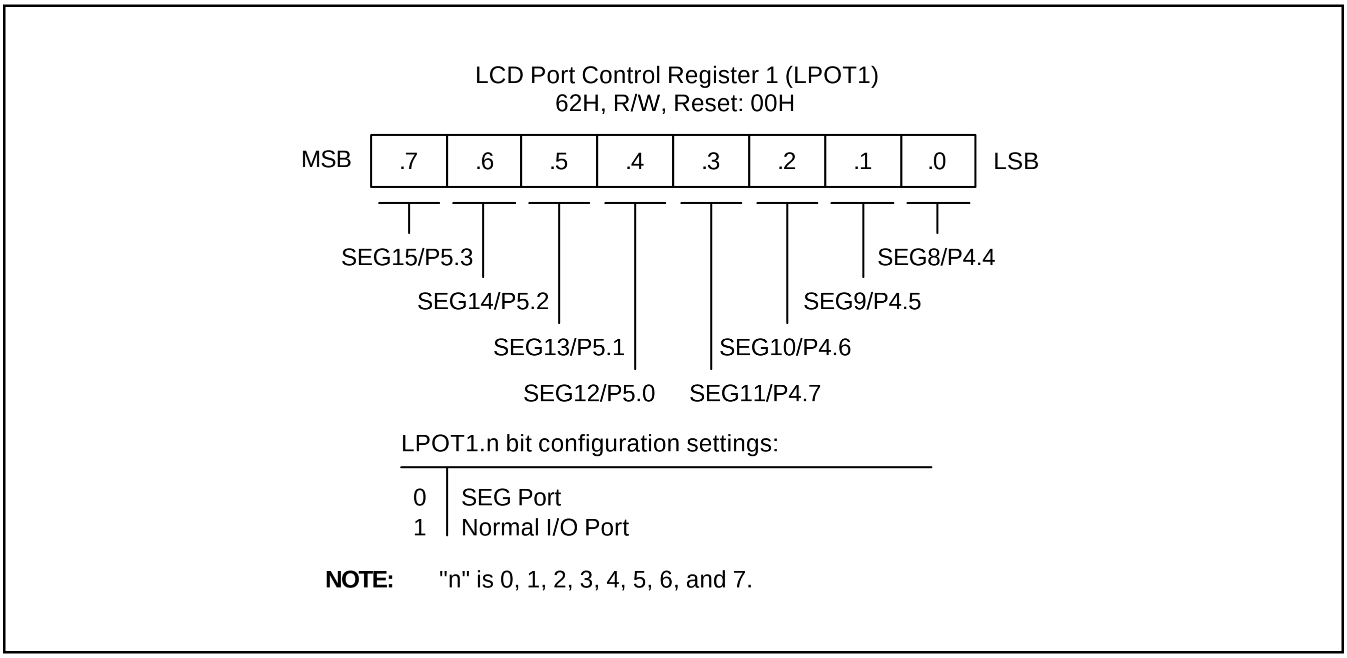


LPOT0.n bit configuration settings:

0	SEG Port
1	Normal I/O Port

NOTE: "n" is 0, 1, 2, 3, 4, 5, 6, and 7.

Figure 17-5. LCD Port Control Register 0 (LPOT0)

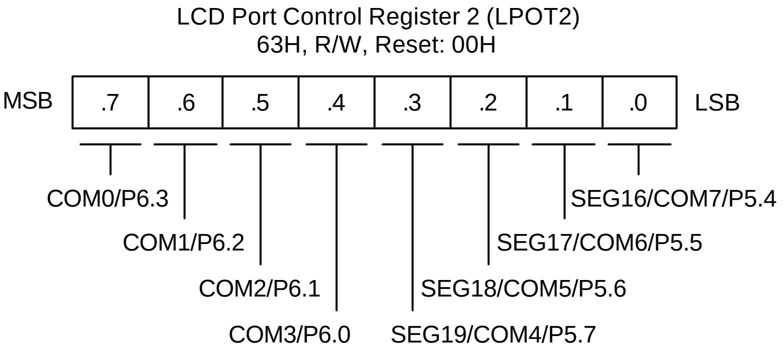


LPOT1.n bit configuration settings:

0	SEG Port
1	Normal I/O Port

NOTE: "n" is 0, 1, 2, 3, 4, 5, 6, and 7.

Figure 17-6. LCD Port Control Register 1 (LPOT1)



LPOT2.n bit configuration settings:

0	SEG or COM Port
1	Normal I/O Port

NOTE: "n" is 0, 1, 2, 3, 4, 5, 6, and 7.

Figure 17-7. LCD Port Control Register 2 (LPOT2)

LCD VOLTAGE DIVIDING RESISTORS

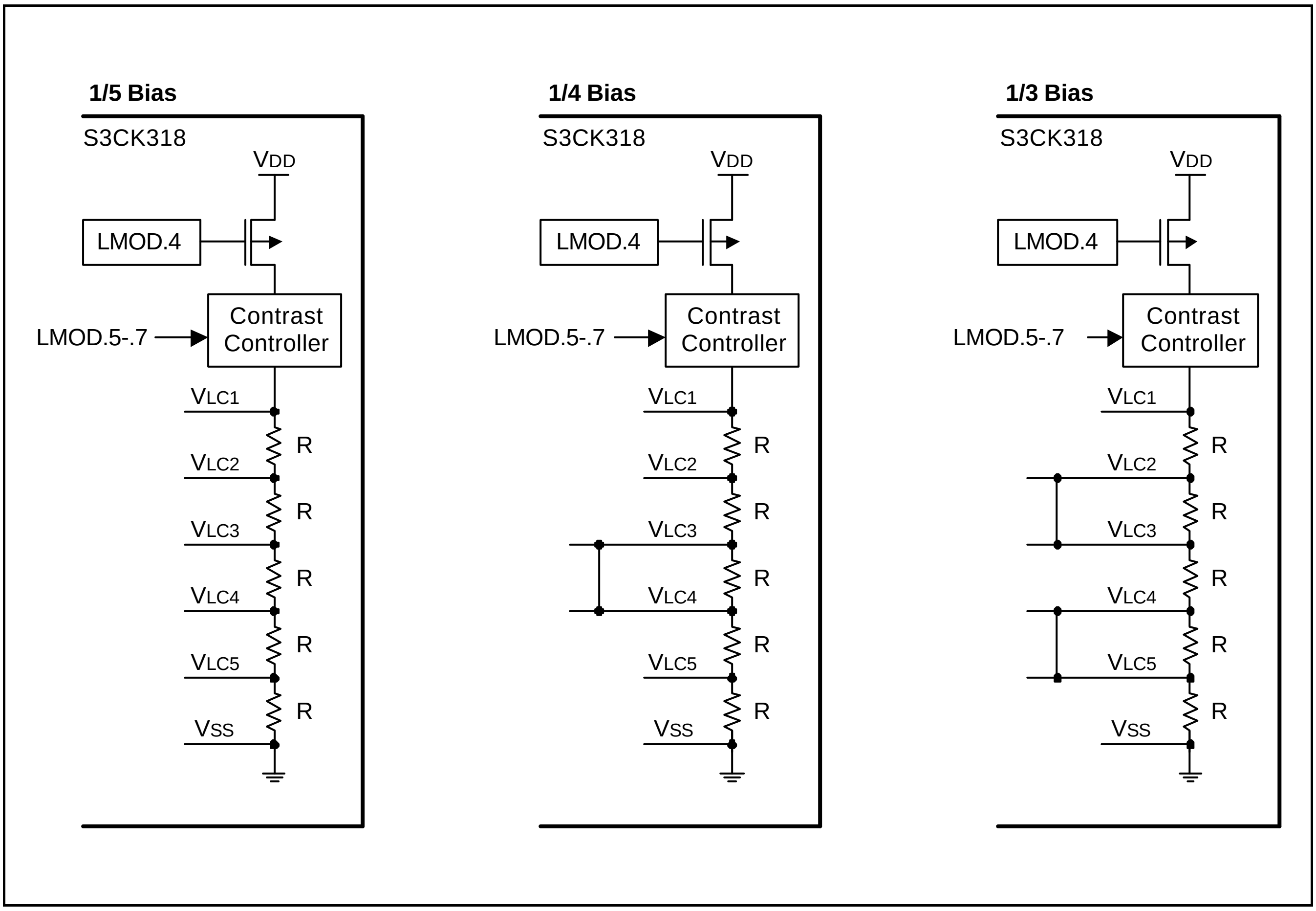


Figure 17-8. LCD Bias Circuit Connection

COMMON (COM) SIGNALS

The common signal output pin selection (COM pin selection) varies according to the selected duty cycle.

— In 1/3 duty mode, COM0–COM2 pins are selected.

— In 1/4 duty mode, COM0–COM3 pins are selected.

— In 1/8 duty mode, COM0–COM7 pins are selected.

When 1/3 duty is selected by setting LMOD.3–LMOD.2 to "00", COM3–COM7 (P6.0, P5.7–P5.4) can be used for I/O ports, and when 1/4 duty is selected by setting LMOD.3–LMOD.2 to "01", COM4–COM7 (P5.7–P5.4) can be used for I/O ports.

SEGMENT (SEG) SIGNALS

The 20 LCD segment signal pins are connected to corresponding display RAM locations at 80H–93H of page 1. Bits of the display RAM are synchronized with the common signal output pins.

When the bit value of a display RAM location is "1", a 'select' signal (Display ON) is sent to the corresponding segment pin.
When the display bit is "0", a 'non-select' signal (Display Off) is sent to the corresponding segment pin.

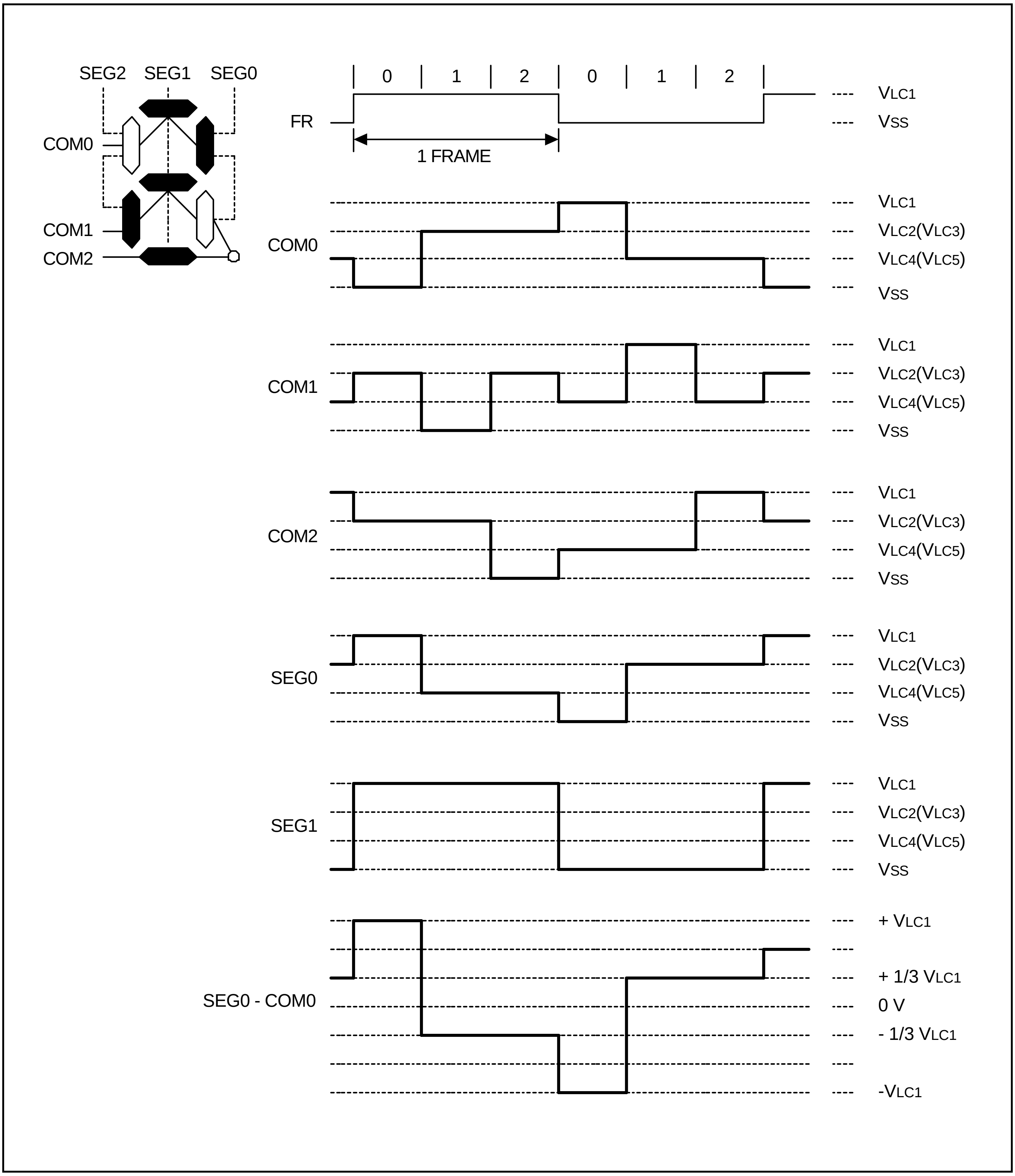


Figure 17-9. LCD Signal Waveforms (1/3 Duty, 1/3 Bias)

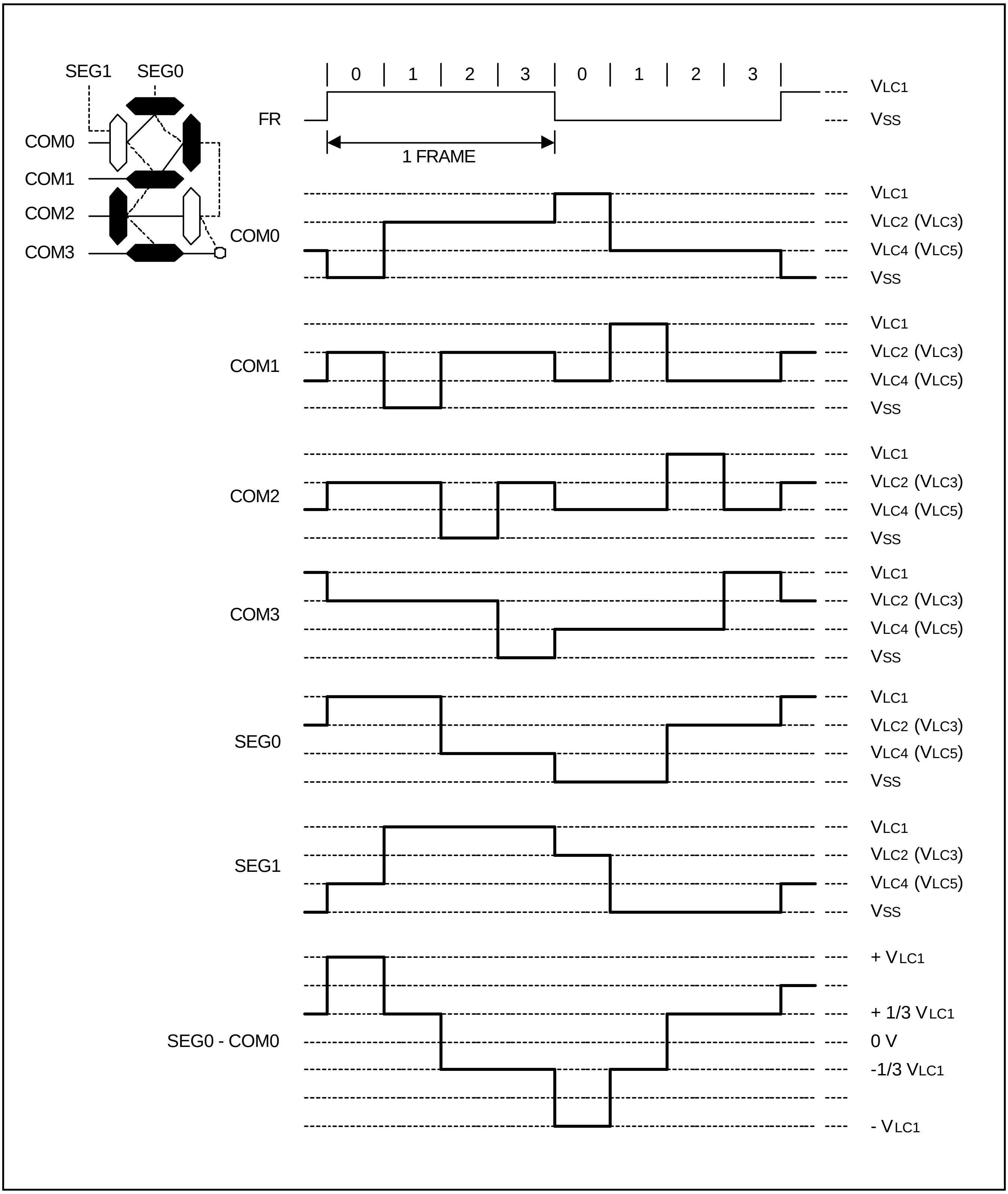


Figure 17-10. LCD Signal Waveforms (1/4 Duty, 1/3 Bias)

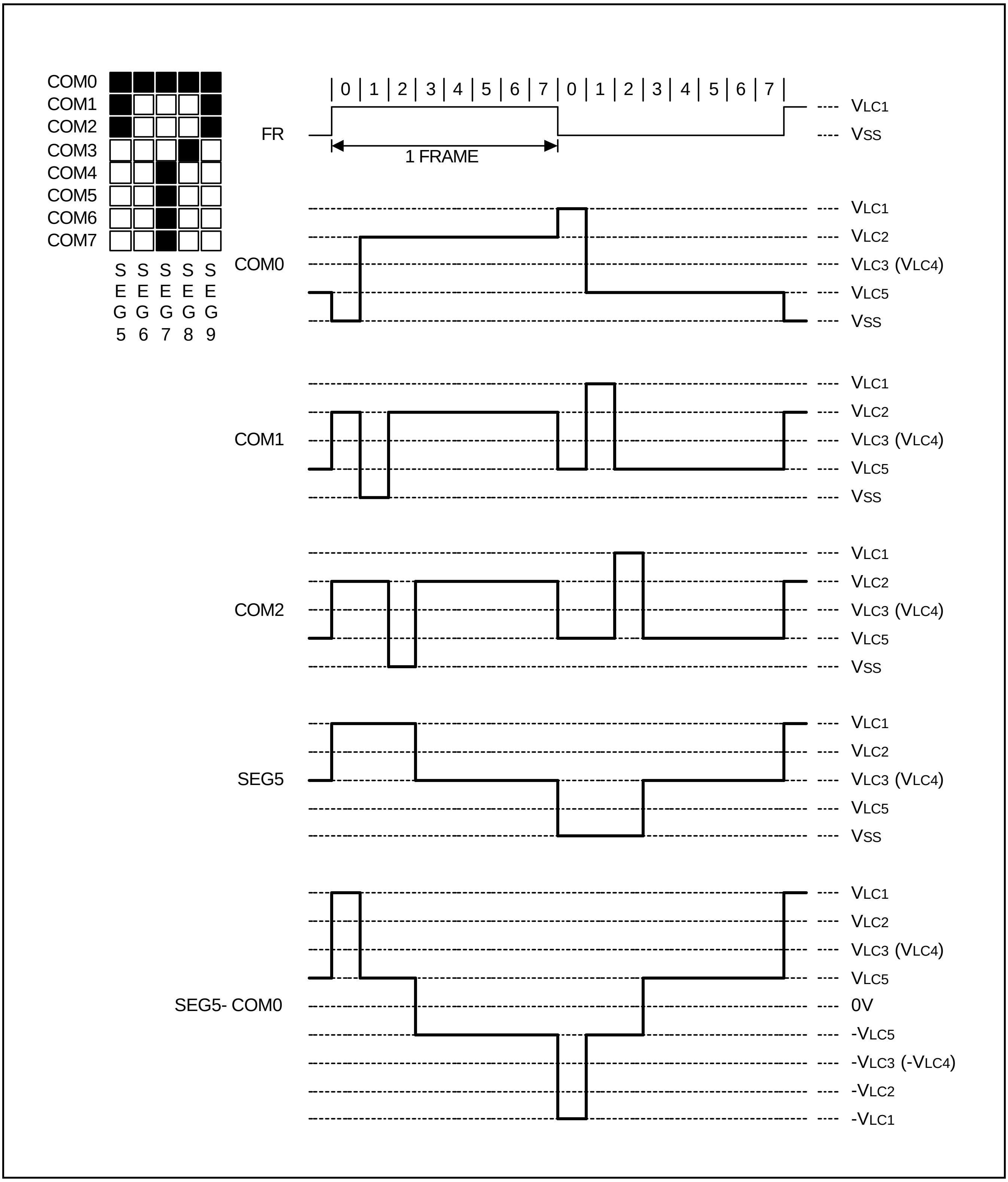


Figure 17-11. LCD Signal Waveforms (1/8 Duty, 1/4 Bias)

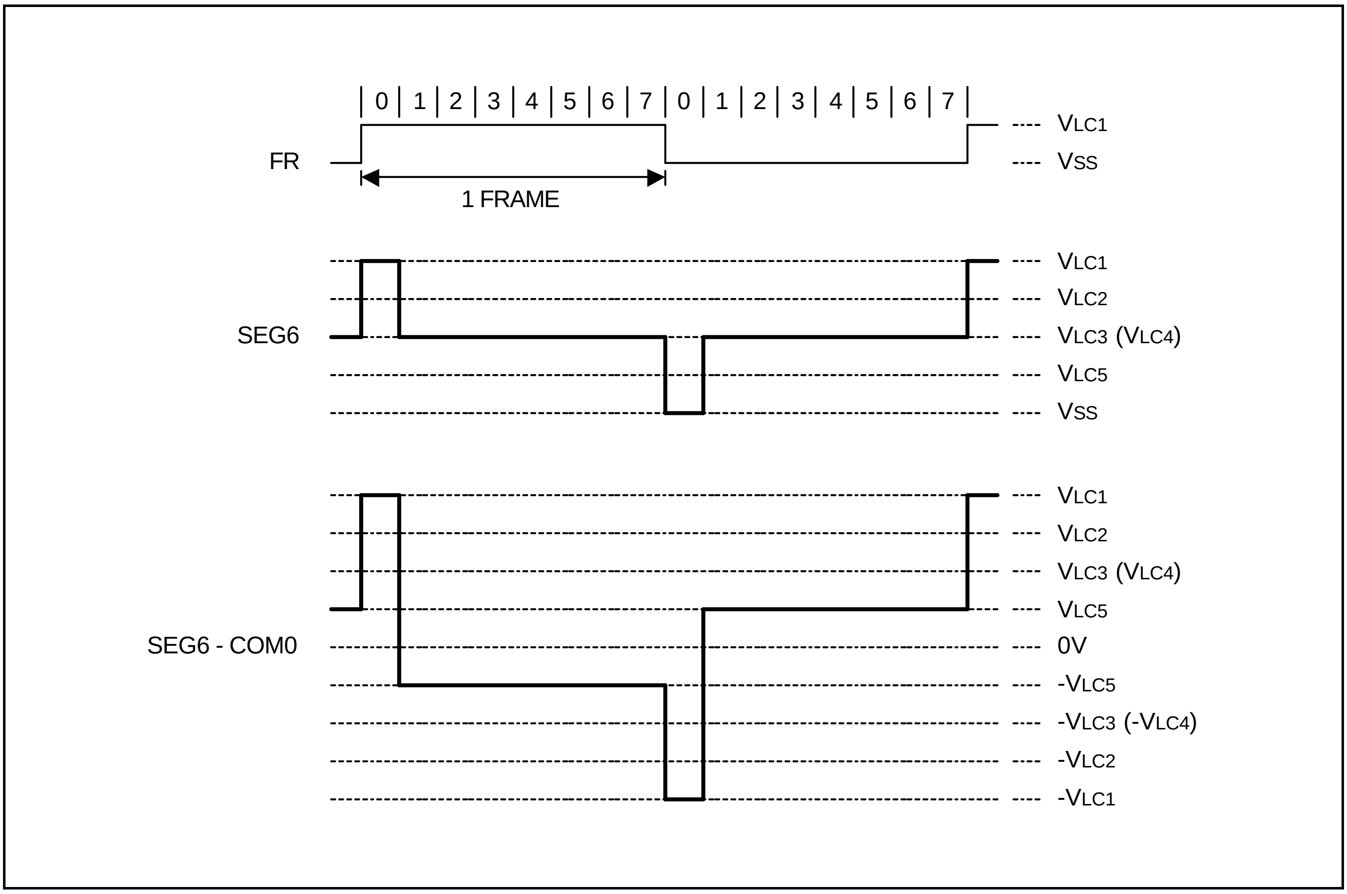


Figure 17-11. LCD Signal Waveforms (1/8 Duty, 1/4 Bias) (Continued)

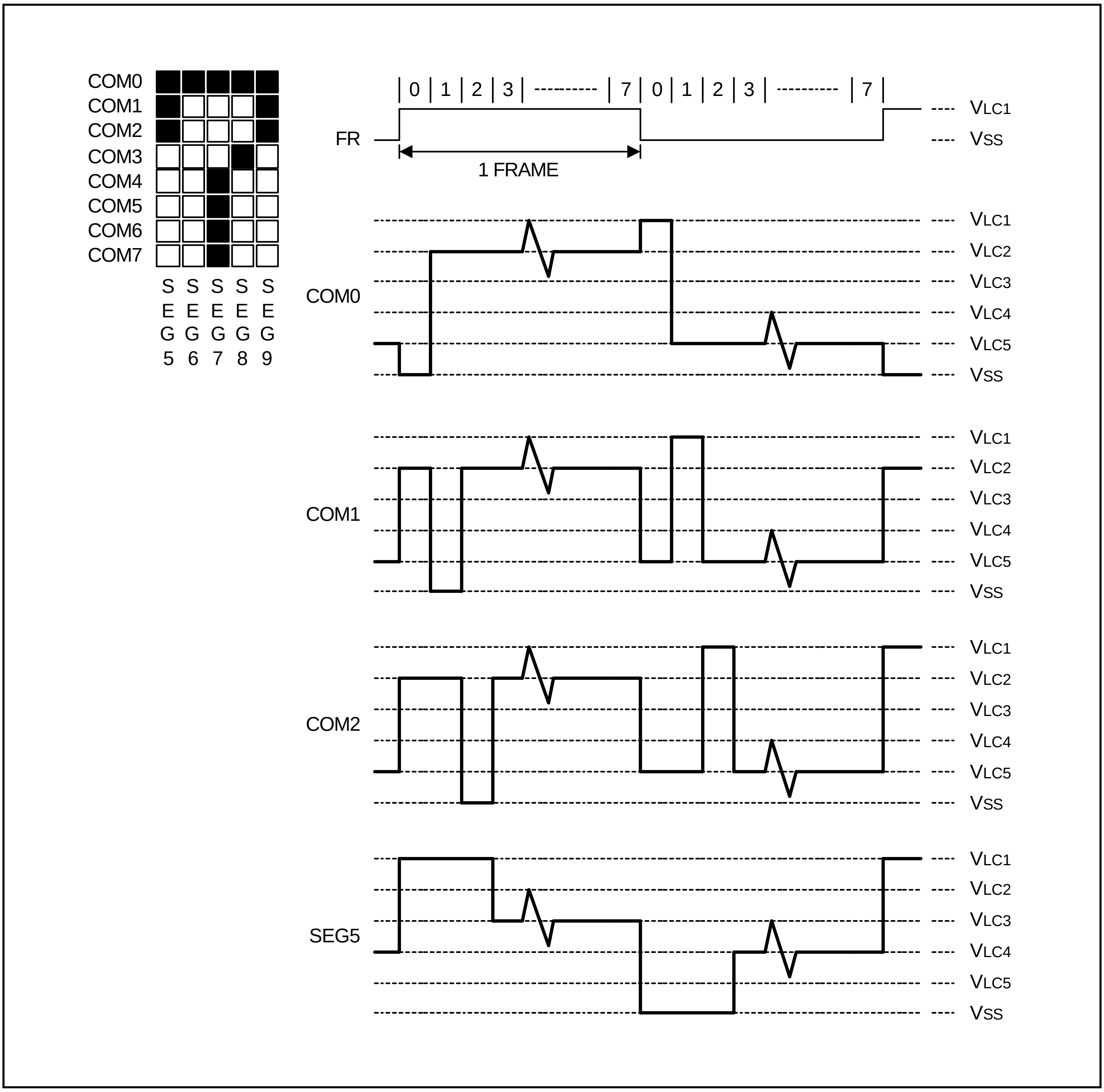


Figure 17-12. LCD Signal Waveforms (1/8 Duty, 1/5 Bias)

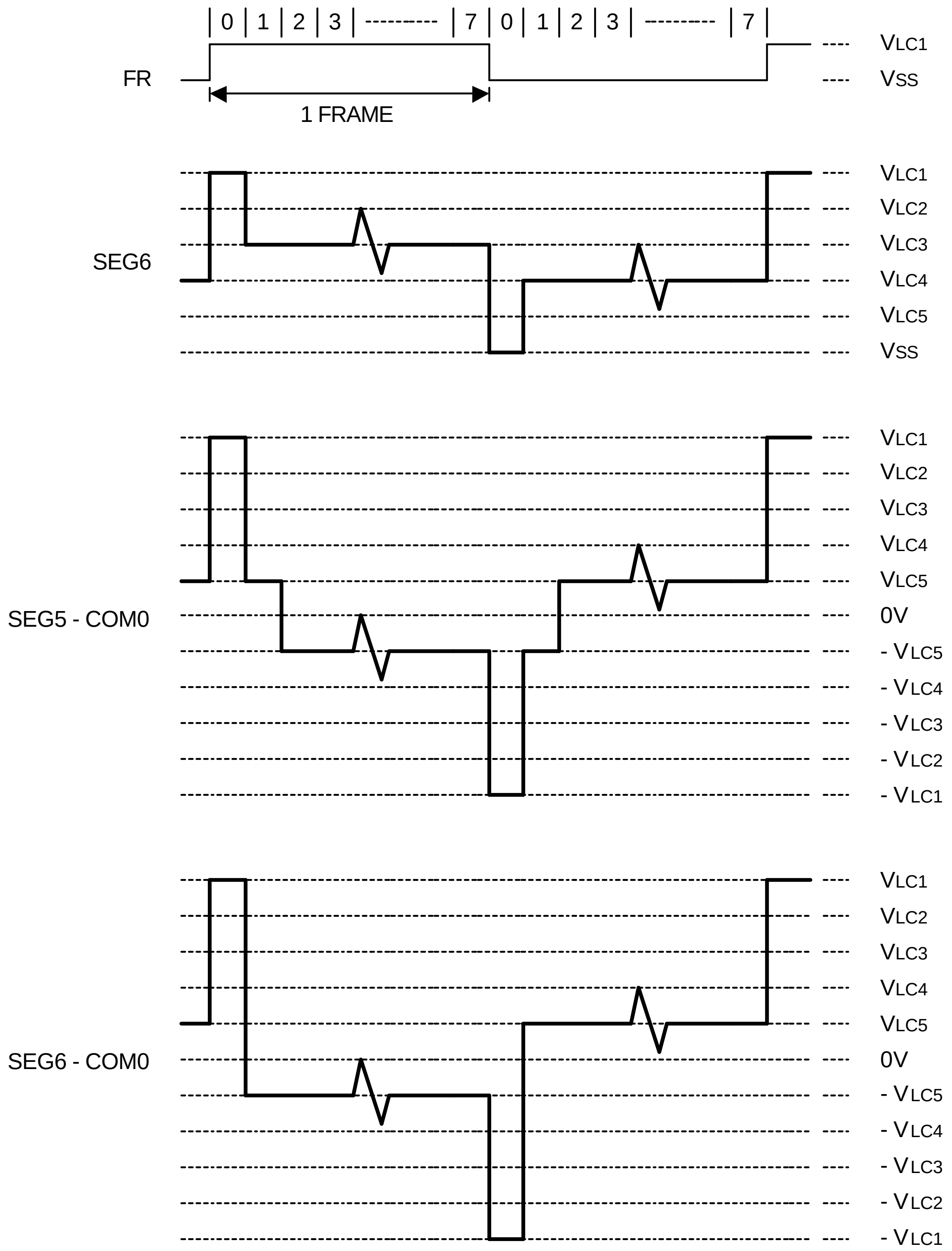


Figure 17-12. LCD Signal Waveforms (1/8 Duty, 1/5 Bias) (Continued)

18 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 four input channels to equivalent 10-bit digital values. The analog input level must lie between the V_{DD} and V_{SS} values. The A/D converter has the following components:

- Analog comparator with successive approximation logic
- D/A converter logic (resistor string type)
- ADC control register (ADCON)
- Four multiplexed analog data input pins (AD0–AD3)
- 10-bit A/D conversion data output register (ADDATAH/ADDATAL)
- 4-bit digital input port (Alternately, I/O port)

FUNCTION DESCRIPTION

To initiate an analog-to-digital conversion procedure, at first you must set with alternative function for ADC input enable at port 2, 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 ADCON.4–.5 to select one of the four analog input pins (AD0–AD3) and set the conversion start or disable bit, ADCON.0. The read-write ADCON register is located in address 5CH. The pins witch are not used for ADC can be used for normal I/O.

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 (ADCON.5–.4) in the ADCON register. To start the A/D conversion, you should set the enable bit, ADCON.0. When a conversion is completed, ADCON.3, the end-of-conversion (EOC) bit is automatically set to 1 and the result is dumped into the ADDATAH/ADDATAL register where it can be read. The A/D converter then enters an idle state. Remember to read the contents of ADDATAH/ADDATAL 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 8 MHz fxx clock frequency, one clock cycle is 1 us. Each bit conversion requires 4 clocks, the conversion rate is calculated as follows:

4 clocks/bit × 10-bit + set-up time = 50 clocks, 50 clock × 1 us = 50 us at 1 MHz (8 MHz/8)

Note that A/D converter needs at least 25µs for conversion time.

A/D CONVERTER CONTROL REGISTER (ADCON)

The A/D converter control register, ADCON, is located at address 5CH. It has three functions:

- — Analog input pin selection (bits 5 and 4)
- — 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 ADCON.4–.5 bits. And the pins not used for analog input can be used for normal I/O function.

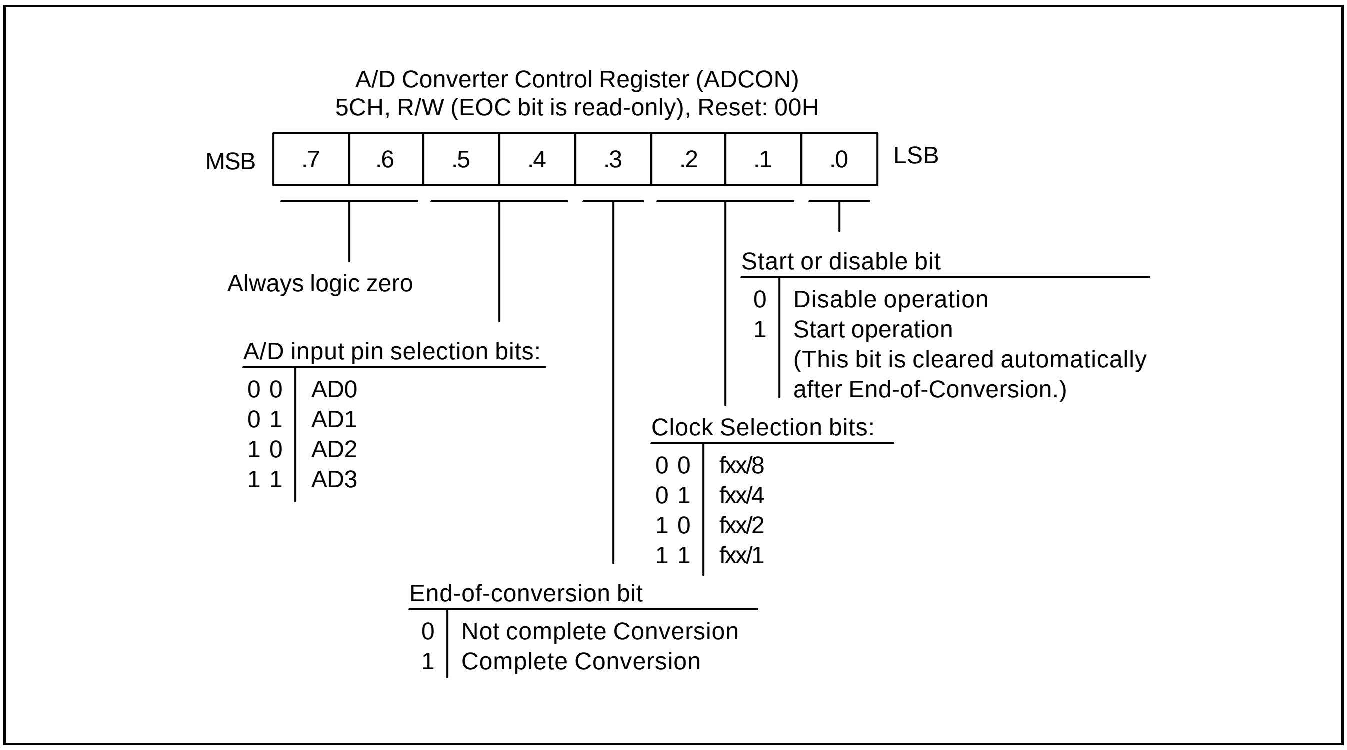


Figure 18-1. A/D Converter Control Register (ADCON)

A/D Converter Data Register (ADDATAH/ADDATAL)
5DH/5EH, Read Only

MSB	.9	.8	.7	.6	.5	.4	.3	.2	LSB (ADDATAH)
MSB	.1	.0	—	—	—	—	—	—	LSB (ADDATAL)

Figure 18-2. A/D Converter Data Register (ADDATAH/ADDATAL)

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

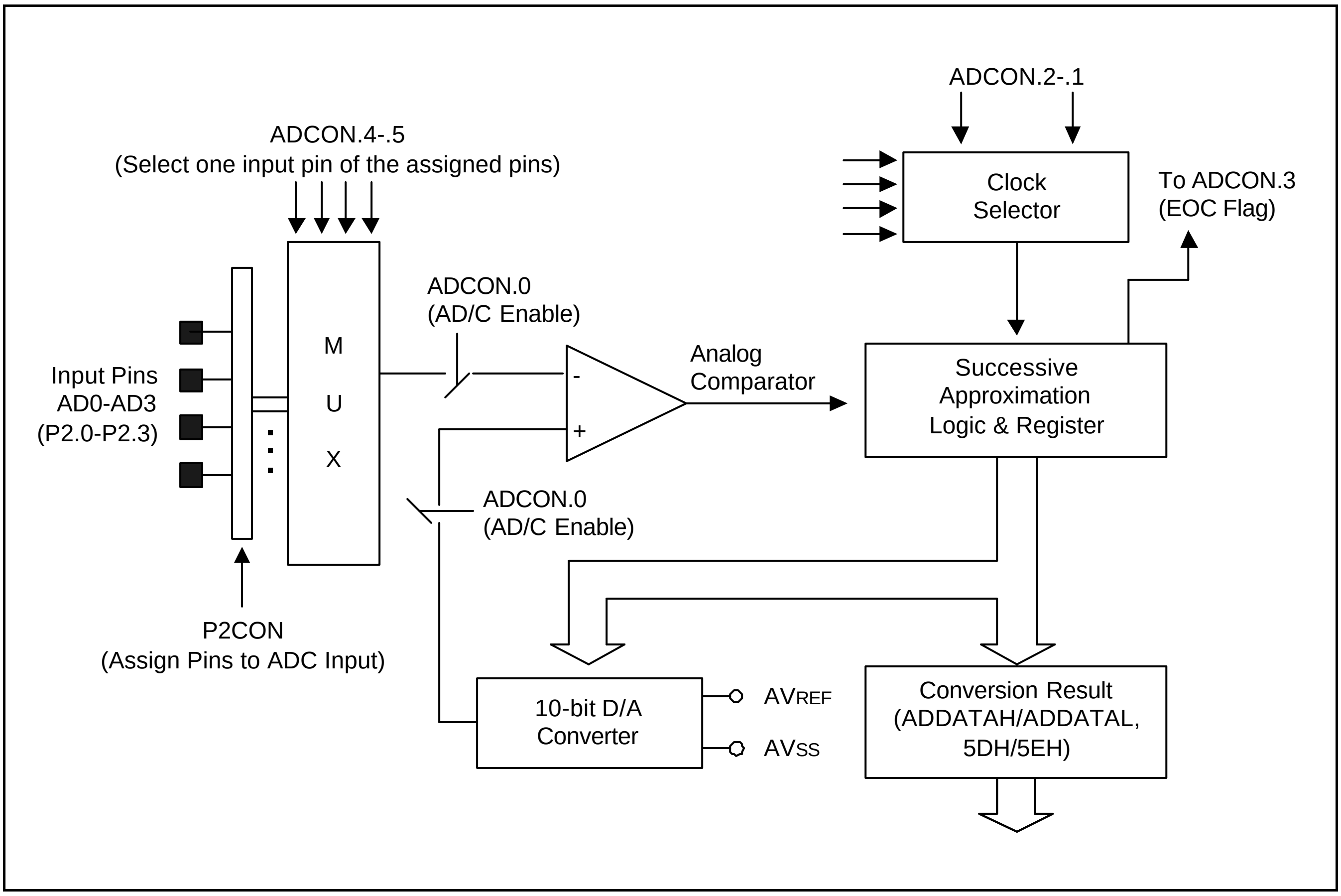


Figure 18-3. A/D Converter Functional Block Diagram

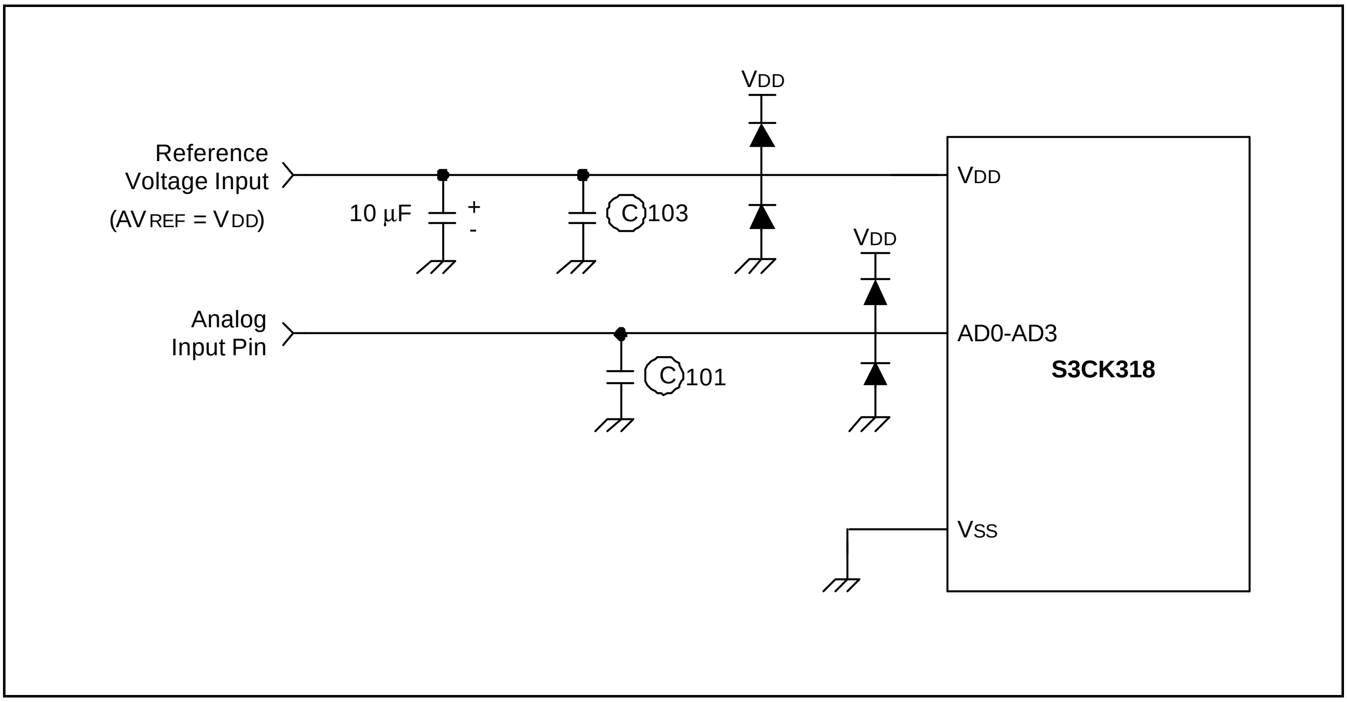


Figure 18-4. Recommended A/D Converter Circuit for Highest Absolute Accuracy

19 D/A CONVERTER

OVERVIEW

The 9-bit D/A Converter (DAC) module uses successive approximation logic to convert 9-bit digital values to equivalent analog levels between V_{DD} $(1 - \frac{1}{512})$ and $V_{SS.}$

This D/A Converter consists of R–2R array structure. The D/A Converter has the following components:

- R–2R array structure
- Digital-to-analog converter control register (DACON)
- Digital-to-analog converter data register (DADATAH/DADATAL)
- Digital-to-analog converter output pin (DAO)

FUNCTION DESCRIPTION

To initiate a digital-to-analog conversion procedure, at first you must set with alternative function (P3CONA.2–.0) and set the digital-to-analog converter enable bit (DACON.0).

The DACON register is located at the RAM address 74H. You should write the digital value calculated to digital-to-analog converter data register (DADATAH/DADATAL).

NOTE

If the chip enters to power-down mode, STOP or IDLE, in conversion process, there will be current path in D/A Converter block. So. It is necessary to cut off the current path before the instruction execution enters power-down mode.

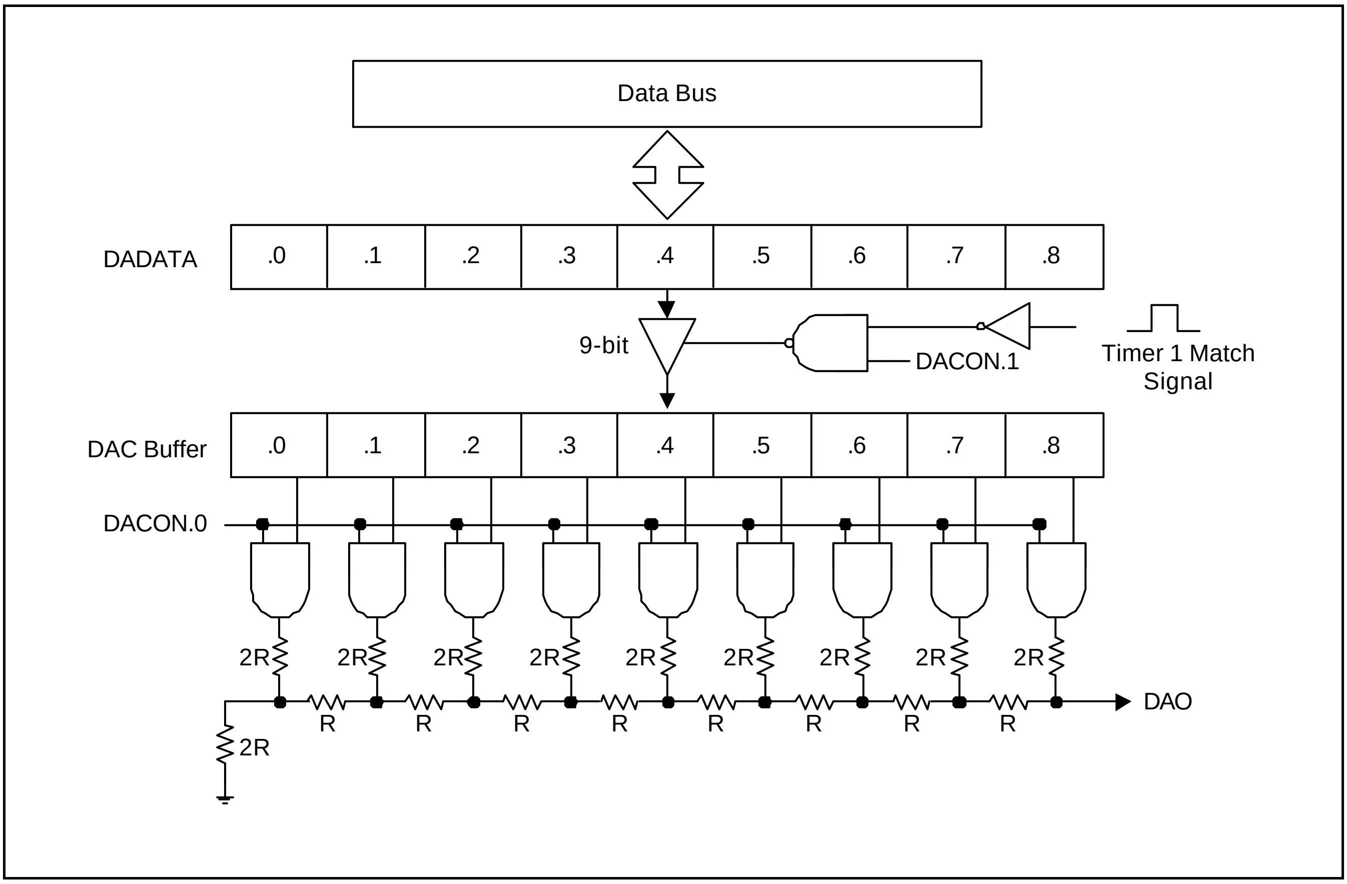
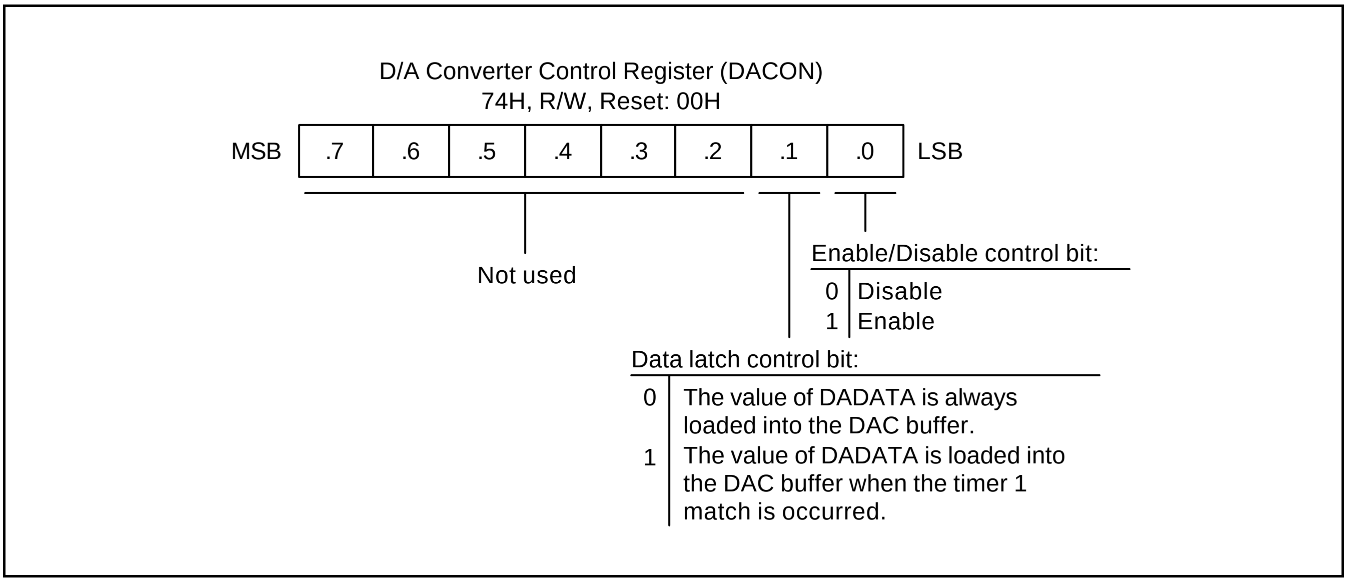


Figure 19-1. DAC Circuit Diagram



D/A Converter Control Register (DACON)
74H, R/W, Reset: 00H

Not used

Enable/Disable control bit:

0	Disable
1	Enable

Data latch control bit:

0	The value of DADATA is always loaded into the DAC buffer.
1	The value of DADATA is loaded into the DAC buffer when the timer 1 match is occurred.

Figure 19-2. Digital to Analog Converter Control Register (DACON)

D/A CONVERTER DATA REGISTER (DADATAH/DADATAL)

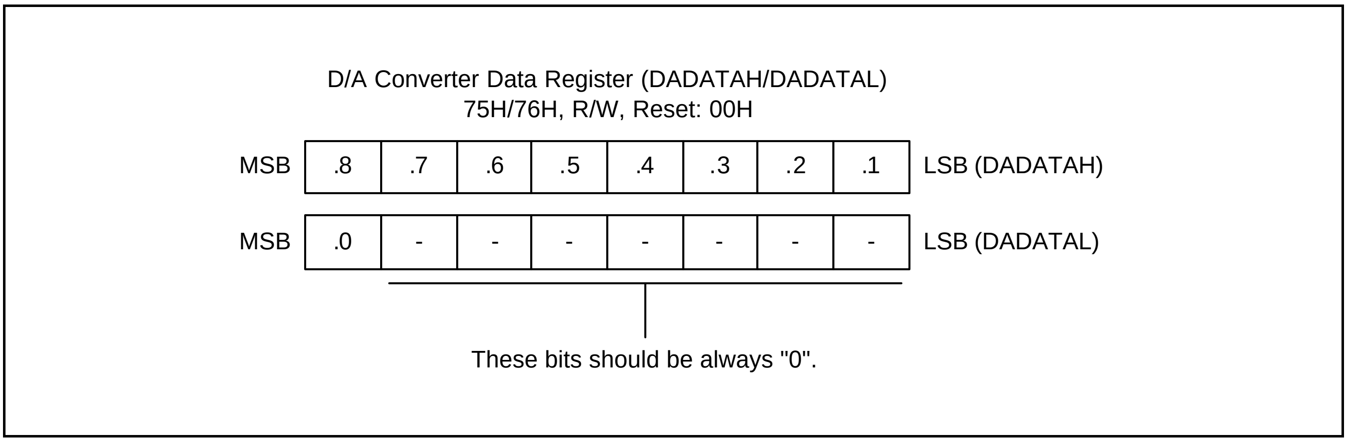
The DAC DATA register, DADATAH/DADATAL is located at the RAM address, 75H–76H. DADATAH/DADATAL specifies the digital data to generate analog voltage. A reset initializes the DADATAH/DADATAL value to "00H". The D/A converter output value, V_{DAO}, is calculated by the following formula.

$$V_{DAO} = V_{DD} \text{ ´ } \frac{n}{512} \qquad \text{(n = 0–511, DADATAH/DADATAL value)}$$

Table 19-1. DADATA Setting to Generate Analog Voltage

DADATAH.7	DADATAH.6	DADATAH.5	DADATAH.4	DADATAH.3	DADATAH.2	DADATAH.1	DADATAH.0	DADATAL.7	V_{DAO}
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	$V_{DD}/2^1$
0	1	0	0	0	0	0	0	0	$V_{DD}/2^2$
0	0	1	0	0	0	0	0	0	$V_{DD}/2^3$
0	0	0	1	0	0	0	0	0	$V_{DD}/2^4$
0	0	0	0	1	0	0	0	0	$V_{DD}/2^5$
0	0	0	0	0	1	0	0	0	$V_{DD}/2^6$
0	0	0	0	0	0	1	0	0	$V_{DD}/2^7$
0	0	0	0	0	0	0	1	0	$V_{DD}/2^8$
0	0	0	0	0	0	0	0	1	$V_{DD}/2^9$

NOTE: These are the values determined by setting just one-bit of DADATA.0–DADATA.8. Other values of DAO can be obtained with superimposition.



D/A Converter Data Register (DADATAH/DADATAL)
75H/76H, R/W, Reset: 00H

MSB	.8	.7	.6	.5	.4	.3	.2	.1	LSB (DADATAH)
MSB	.0	-	-	-	-	-	-	-	LSB (DADATAL)

These bits should be always "0".

Figure 19-3. D/A Converter Data Register (DADATAH/DADATAL)

NOTES

20 FREQUENCY COUNTER

OVERVIEW

The S3CK318 uses a frequency counter (FC) to counter the analog frequencies, FM, AM, SW, or digital pules. The FC block consists of a 1/16 divider, gate control circuit, FC control register (FCCON) and a 24-bit binary counter. The gate control circuit, which controls the frequency counting time, is programmed using the FCMOD register. Four different gate times can be selected using FCMOD register settings.

During gate time, the 24-bit FC counts the input frequency at the FMF or AMF pins. The FMF or AMF pin input signal for the 24-bit counter is selected using FCCON register settings.

When the FMF pin input signal is selected, the signal is divided by 16. When the AMF pin input signal is selected, the signal is directly connected or divided by 16 to the FC.

P0.1/FMF and P0.0/AMF signals also can be used normal input port and count external events.

By setting FCMOD register, the gate is opened for 16-ms, 32-ms or 64-ms periods. During the open period of the gate, input frequency is counted by the 24-bit counter. When the gate is closed, the counting operation is complete, and an interrupt is generated.

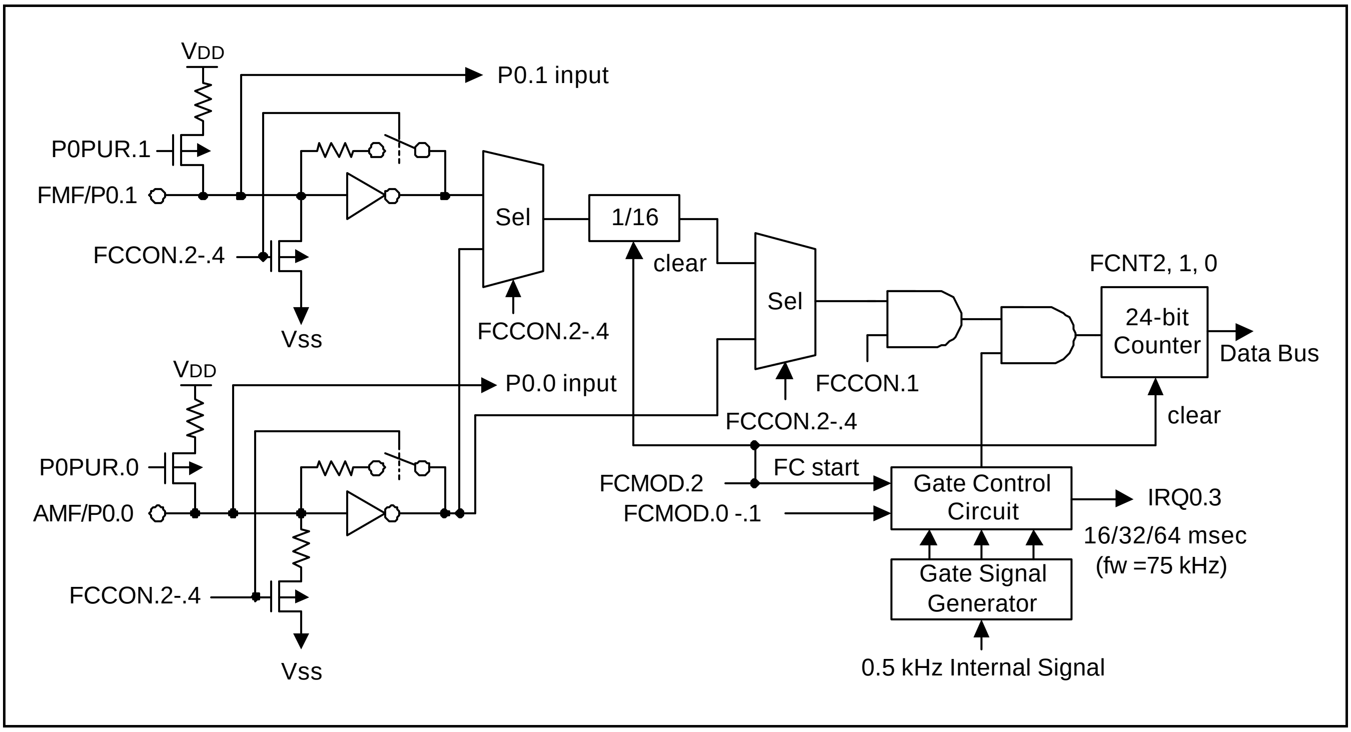
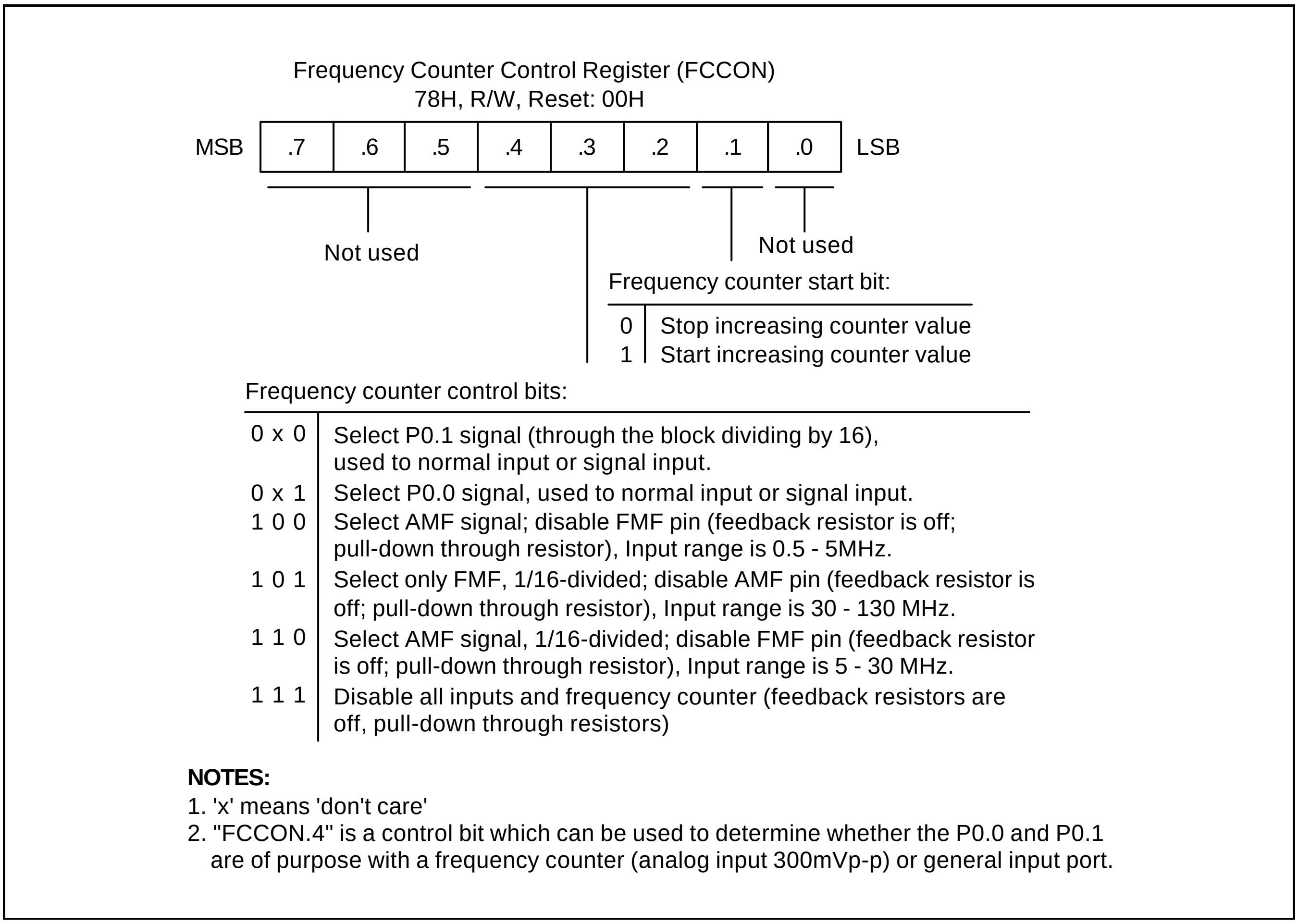


Figure 20-1. Frequency Counter Block Diagram

FC CONTORL REGISTER (FCCON)

The FC control register (FCCON) is a 8-bit register that is used to stop/start frequency counting and select a signal of FMF or AMF.



Frequency Counter Control Register (FCCON)
78H, R/W, Reset: 00H

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

Frequency counter start bit:

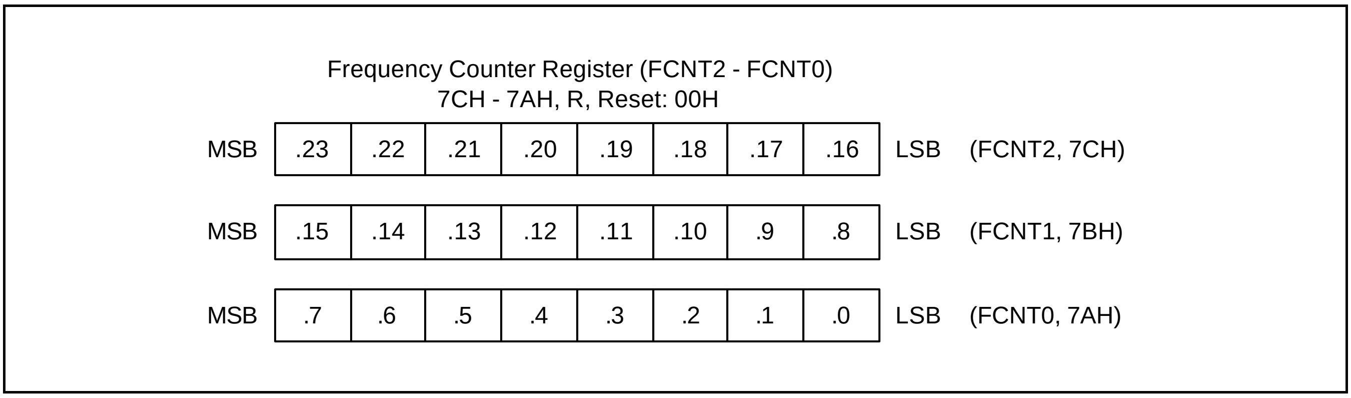
0	Stop increasing counter value
1	Start increasing counter value

Frequency counter control bits:

0 x 0	Select P0.1 signal (through the block dividing by 16), used to normal input or signal input.
0 x 1	Select P0.0 signal, used to normal input or signal input.
1 0 0	Select AMF signal; disable FMF pin (feedback resistor is off; pull-down through resistor), Input range is 0.5 - 5MHz.
1 0 1	Select only FMF, 1/16-divided; disable AMF pin (feedback resistor is off; pull-down through resistor), Input range is 30 - 130 MHz.
1 1 0	Select AMF signal, 1/16-divided; disable FMF pin (feedback resistor is off; pull-down through resistor), Input range is 5 - 30 MHz.
1 1 1	Disable all inputs and frequency counter (feedback resistors are off, pull-down through resistors)

NOTES:
1. 'x' means 'don't care'
2. "FCCON.4" is a control bit which can be used to determine whether the P0.0 and P0.1 are of purpose with a frequency counter (analog input 300mVp-p) or general input port.

Figure 20-2. Frequency Counter Control Register (FCCON)



Frequency Counter Register (FCNT2 - FCNT0)
7CH - 7AH, R, Reset: 00H

MSB	.23	.22	.21	.20	.19	.18	.17	.16	LSB	(FCNT2, 7CH)
MSB	.15	.14	.13	.12	.11	.10	.9	.8	LSB	(FCNT1, 7BH)
MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB	(FCNT0, 7AH)

Figure 20-3. Frequency Counter Register (FCNT2–FCNT0)

Table 20-1. Frequency Counter Control Register (FCCON) Bit Settings in Normal Operating Mode

FCCON.4	FCCON.3	FCCON.2	FMF's Feedback Resistor	AMF's Feedback Resistor	FMF's Pull-down	AMF's Pull-down
0	x	x	Off	Off	Off	Off
1	0	0	Off	On	On	Off
	0	1	On	Off	Off	On
	1	0	Off	On	On	Off
	1	1	Off	Off	On	On

Table 20-2. Frequency Counter Control Register (FCCON) Bit Settings in Power-Down Mode

FCCON.4	FCCON.3	FCCON.2	FMF's Feedback Resistor	AMF's Feedback Resistor	FMF's Pull-down	AMF's Pull-down
0	x	x	Off	Off	Off	Off
1	x	x	Off	Off	On	On

NOTE: 'x' means 'don't care'.

INPUT PIN CONFIGURATION FOR AN AC FREQUENCY COUNT

Because the FMF and AMF pins have built-in AC amplifiers, DC component of the input signal must be stripped off by the external capacitor.

When the FMF or AMF pin is selected for AC frequency counter function, the voltage level of the corresponding pin is increased to approximately 1/2 V_{DD} after a sufficiently long time. If the pin voltage does not increase, the AC amplifier exceeds its operating range and maybe cause an FC malfunction. To prevent this from occurring, you should program a sufficiently long time delay interval before starting the count operation.

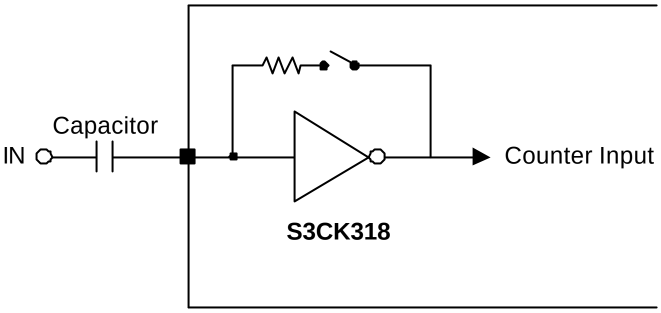
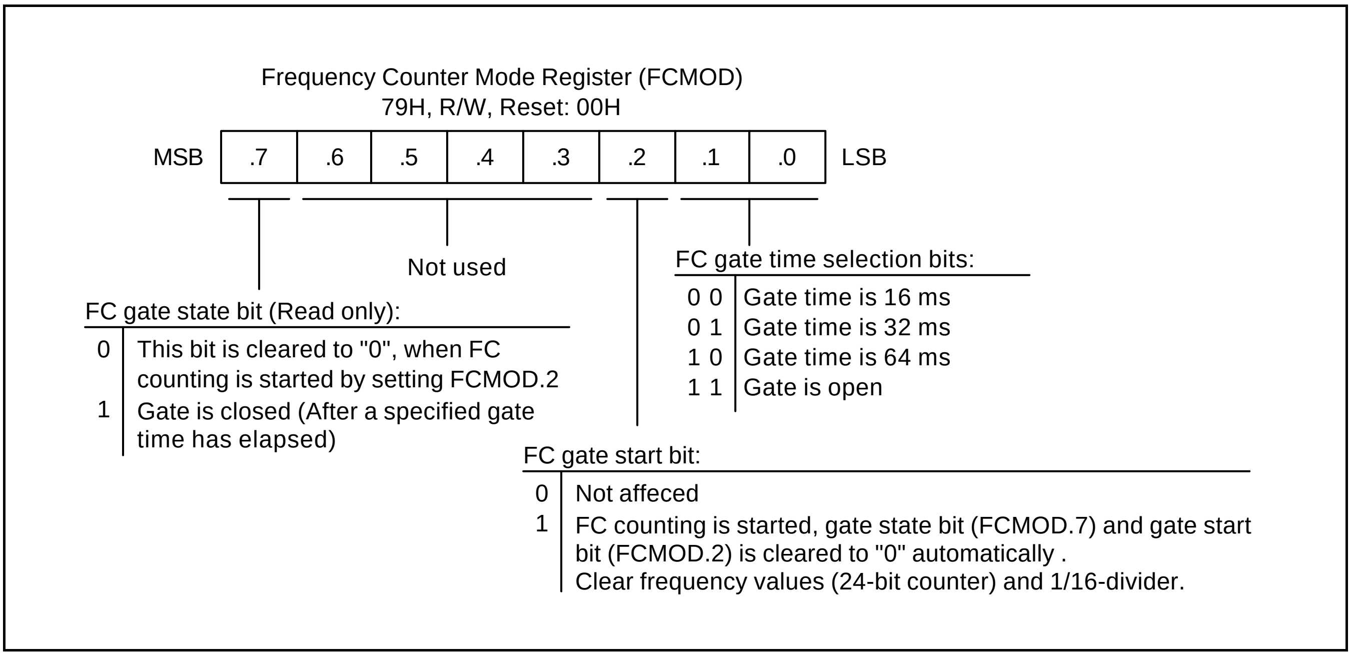


Figure 20-4. FMF and AMF Pin Configuration

FC MODE REGISTER (FCMOD)

FCMOD is a 8-bit register which can be used to select the gate time, and check whether a FC counting operation has been completed or not.



FC gate state bit (Read only):

0 This bit is cleared to "0", when FC counting is started by setting FCMOD.2

1 Gate is closed (After a specified gate time has elapsed)

FC gate time selection bits:

0 0	Gate time is 16 ms
0 1	Gate time is 32 ms
1 0	Gate time is 64 ms
1 1	Gate is open

FC gate start bit:

0 Not affeced

1 FC counting is started, gate state bit (FCMOD.7) and gate start bit (FCMOD.2) is cleared to "0" automatically .
Clear frequency values (24-bit counter) and 1/16-divider.

Figure 20-5. Frequency Counter Mode Register (FCMOD)

GATE TIMES

When you write a value to FCMOD, the FC gate is opened for a 16-millisecond, 32-millisecond, or 64-millisecond interval, setting with a rising clock edge. When the gate is open, the frequency at the AMF or FMF pin is counted by the 24-bit counter. When the gate closes, the FCMOD.7 is set to "1". An interrupt is then generated and the FC interrupt request bit (IRQ0.3) is set.

Figure 20-6 shows gate timings with a 0.5 kHz internal clock.

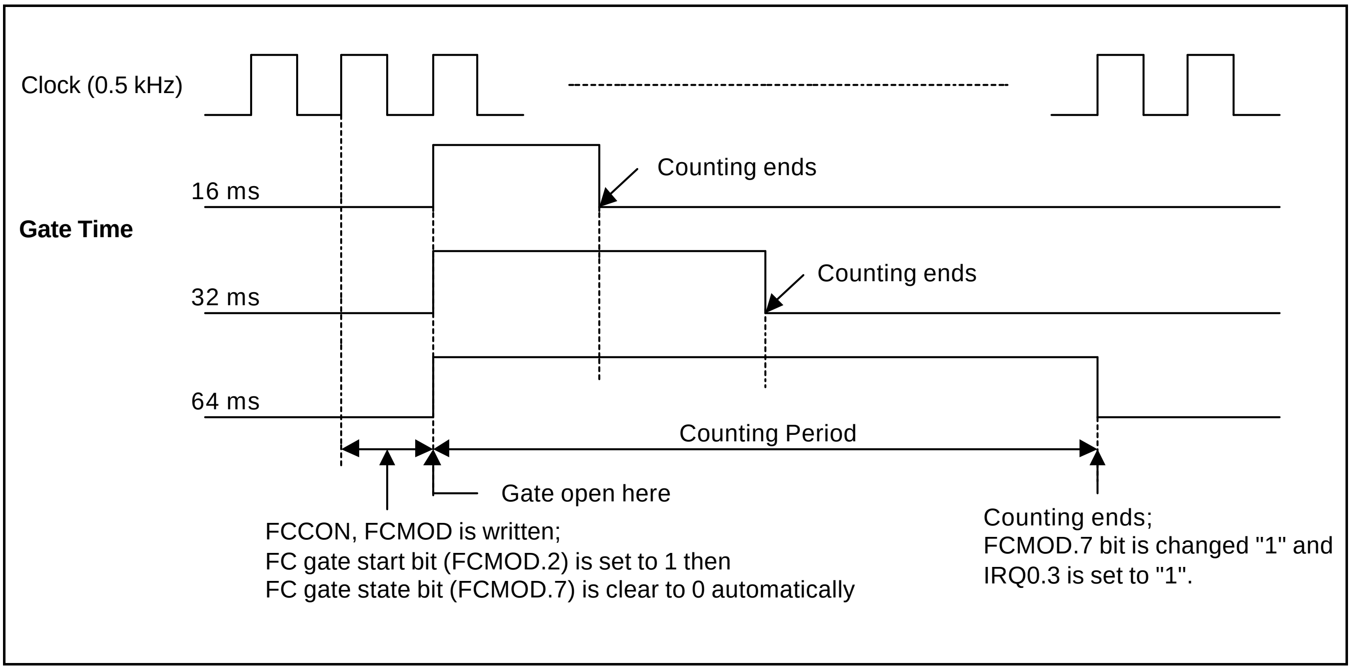


Figure 20-6. Gate Timing (16, 32, or 64-ms)

Selecting "Gate Remains Open"

If you select "gate remain open" (FCMOD.0 and FCMOD.1 = '11'), the FC counts the input signal during the open period of the gate. The gate closes the next time a value is written to FCMOD.

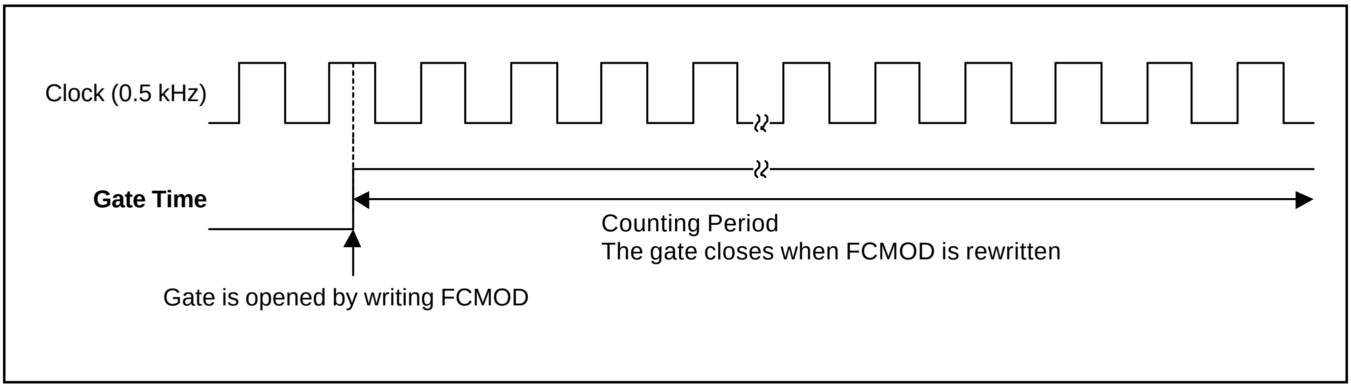
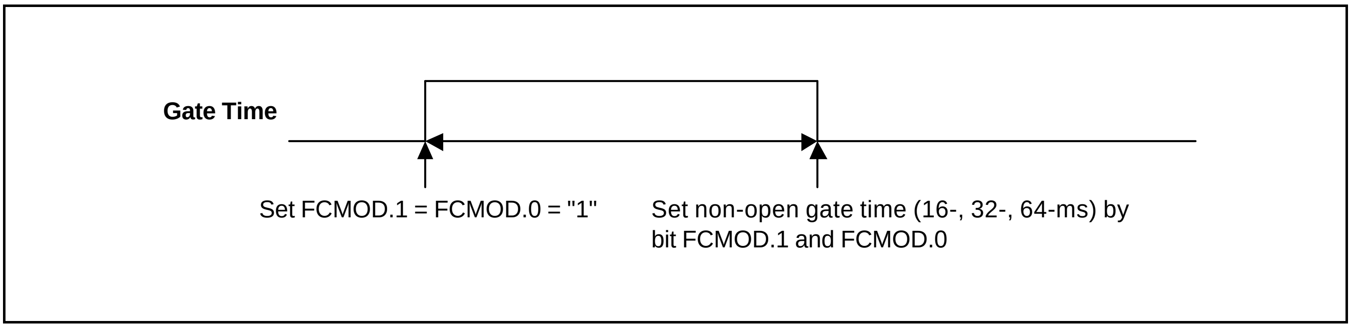

Clock (0.5 kHz)
Gate Time
Counting Period
The gate closes when FCMOD is rewritten
Gate is opened by writing FCMOD

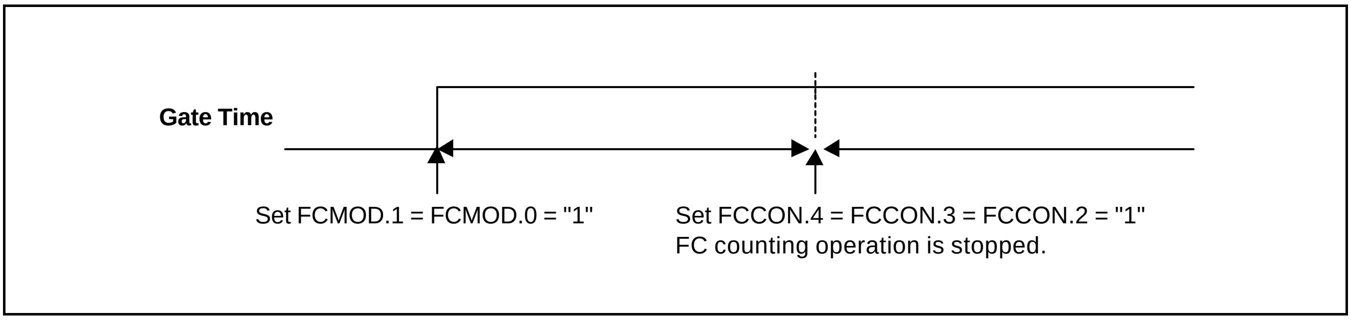
Figure 20-7. Gate Timing (When Open)

When you select "gate remains open" as the gating time, you can control the opening and closing of the gate in one of two ways:

— Set the gate time to a specific interval (16-ms, 32-ms, or 64-ms) by setting bits FCMOD.1 and FCMOD.0.


Gate Time
Set FCMOD.1 = FCMOD.0 = "1"
Set non-open gate time (16-, 32-, 64-ms) by
bit FCMOD.1 and FCMOD.0

— Disable FC operation by setting bits FCCON.4–.2 to "111b". This method lets the gate remain open, and stops the counting operation.


Gate Time
Set FCMOD.1 = FCMOD.0 = "1"
Set FCCON.4 = FCCON.3 = FCCON.2 = "1"
FC counting operation is stopped.

Gate Time Errors

A gate time error occurs whenever the gate signals are not synchronized to the interval instruction clock. That is, the FC does not start counter operation until a rising edge of the gate signal is detected, even though the counter start instruction (setting bit FCMOD.2) has been executed. Therefore, there is a maximum 16-ms timing error (see Figure 20-8).

After you have executed the FC start instruction, you can check the gate state at any time. Please note, however that the FC does not actually start its counting operation until stabilization time for the gate control signal has elapsed.

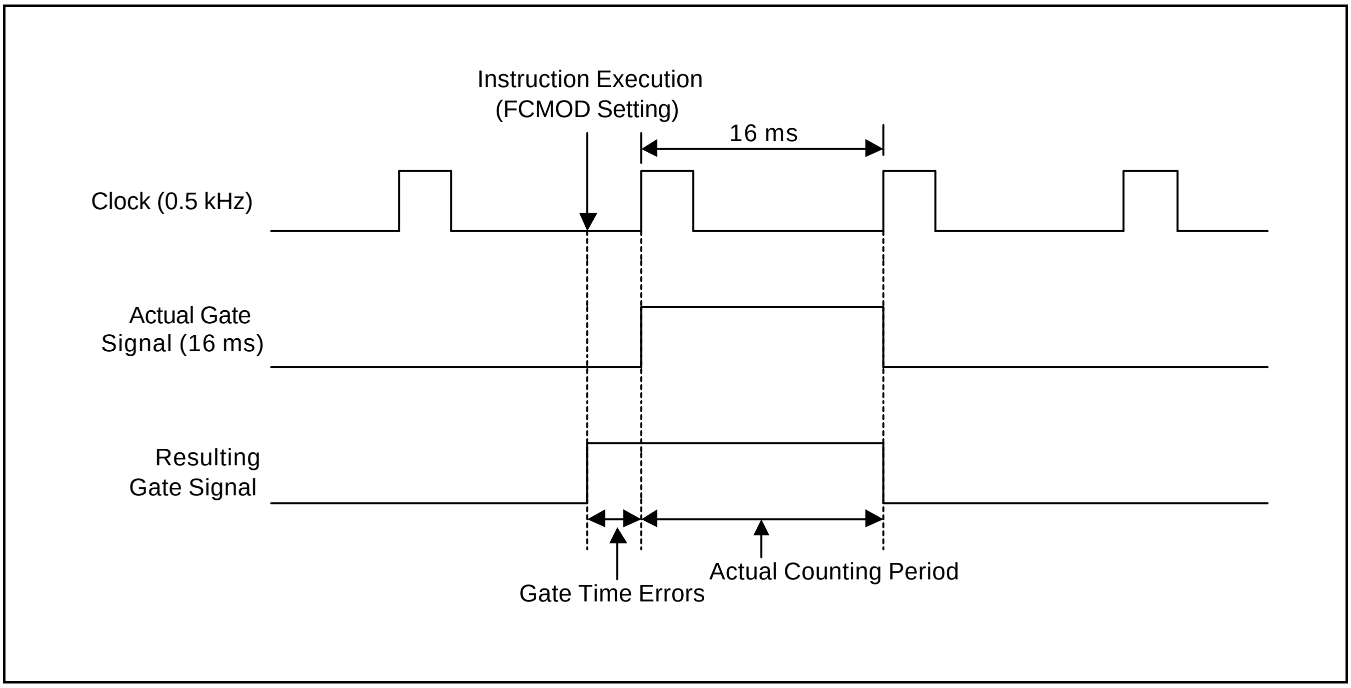


Figure 20-8. Gate Timing (16-ms Error)

Counting Errors

The frequency counter counts the rising edges of the input signal in order to determine the frequency. If the input signal is High level when the gate is open, one additional pulse is counted. When the gate is close, however, counting is not affected by the input signal status. In other words, the counting error is "+1, 0".

FREQUENCY COUNTER (FC) OPERATION

FCCON register bits 4-2 are used to select the input pin and the register bit 1 is use to start or stop FC increasing a counter value. The counting operation of FC is started by setting FCMOD.2 to "1". You stop the counting operation by setting FCCON.4–.2 to "111b". The FCNT2–0 retains its previous value until FCCON and FCMOD register values are specified.

Setting bit FCMOD.2 starts the frequency counting operation. Counting continues as long as the gate is open. The 24-bit counter value is automatically cleared to 000000H after it overflow (at FFFFFFH), and continues counting from zero. A reset operation clears the counter to zero.

FCNT2	FCNT2.7	FCNT2.6	FCNT2.5	FCNT2.4	FCNT2.3	FCNT2.2	FCNT2.1	FCNT2.0
FCNT1	FCNT1.7	FCNT1.6	FCNT1.5	FCNT1.4	FCNT1.3	FCNT1.2	FCNT1.1	FCNT1.0
FCNT0	FCNT0.7	FCNT0.6	FCNT0.5	FCNT0.4	FCNT0.3	FCNT0.2	FCNT0.1	FCNT0.0

When the specified gate open time has elapsed, the gate closes in order to complete the counter operation. At this time, the FC counting ends interrupt request bit (IRQ0.3) is automatically set to "1" and an interrupt is generated. The corresponding IRQ bit must be cleared to "0" by software when the interrupt is serviced (Refer to chapter 6 to clear IRQ bit). The FCMOD.7 is set to "1" at the same time the gate is closed. Since the FCMOD.7 is cleaned to "0" when FC operation start, you can check the FCMOD.7 to determine when FC operation stops (that is, when the specified gate open time has elapsed).

The frequency applied to FMF or AMF pin is counted while the gate is open. The relationship between the count value (N) and input frequencies f_{FMF} and f_{AMF} is shown below.

— FMF (divided by 16 before counting) pin input frequency is

$$f_{FMF} = \frac{N\ (DEC)}{T_G} \times 16$$

when T_G = gate time (16 ms, 32 ms, 64 ms)

— AMF (divided by 16 before counting) pin input frequency is

$$f_{AMF} = \frac{N\ (DEC)}{T_G} \times 16$$

when T_G = gate time (16 ms, 32 ms, 64 ms)

— AMF pin input frequency is

$$f_{AMF} = \frac{N\ (DEC)}{T_G}$$

when T_G = gate time (16 ms, 32 ms, 64 ms)

Table 20-3 shows the range of frequency that you can apply to the AMF and FMF pins.

Table 20-3. FC Frequency Characteristics

Pin	Voltage Level	Frequency Range
AMF	300 m V_{PP} (min)	0.5 MHz to 5 MHz
AMF (divided by 16)	300 m V_{PP} (min)	5 MHz to 30 MHz
FMF (divided by 16)	300 m V_{PP} (min)	30 MHz to 130 MHz

FC DATA CALCULATION

Selecting the FMF pin for FC Input

First, divide the signal at the FMF pin by 16, and then apply this value to the frequency counter. This means that the frequency counter value is equal to one-sixteenth of the input signal frequency.

FMF input frequency (f_{FMF}): 80 MHz
Gate time (T_G): 32 ms
FC counter value (N):

$$
\begin{aligned}
N &= (f_{FMF}/16) \times T_G \\
&= 80 \times 10^6/16 \times 32 \times 10^{-3} \\
&= 160000 \\
&= 027100\text{H}
\end{aligned}
$$

Bin	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0
Hex	0				2				7				1				0				0			
FCNT	FCNT2								FCNT1								FCNT0							

Selecting the AMF (divided by 16) Pin for FC Input

First, divided the signal at the AMF pin 16, and then apply this value to the frequency counter. This means that the frequency counter value is equal to one-sixteenth of the input signal frequency.

AMF input frequency (f_{AMF}): 20 MHz

Gate time (T_G): 16 ms

FC counter value (N):

$$N = (f_{AMF}/16) \times T_G$$
$$= 20 \times 10^6/16 \times 16 \times 10^{-3}$$
$$= 20000$$
$$= 004E20H$$

Bin	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0
Hex	0				0				4				E				2				0			
FCNT	FCNT2								FCNT1								FCNT0							

Selecting the AMF Pin for FC Input

The signal at AMF pin is directly input to the frequency counter.

AMF input frequency (f_{AMF}): 450 kHz

Gate time (T_G): 64 ms

FC counter value (N):

$$N = (f_{AMF}) \times T_G$$
$$= 450 \times 10^3 \times 64 \times 10^{-3}$$
$$= 28800$$
$$= 007080H$$

Bin	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0
Hex	0				0				7				0				8				0			
FCNT	FCNT2								FCNT1								FCNT0							

21 ELECTRICAL DATA

Table 21-1. Absolute Maximum Ratings

(T_A = 25 °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	– 18	mA
		All I/O pins active	– 60	
Output current low	I_{OL}	One I/O pin active	+ 30	mA
		Total pin current for port	+ 100	
Operating temperature	T_A	–	– 25 to + 85	°C
Storage temperature	T_{STG}	–	– 65 to + 150	°C

Table 21-2. D.C. Electrical Characteristics

(T_A = – 25 °C to + 85 °C, V_{DD} = 1.95 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Operating voltage	V_{DD}	fx = 8 MHz	2.3	–	3.6	V
		fx = 2 MHz	1.95	–	3.6	
Input high voltage	V_{IH1}	All input pins except V_{IH2}, V_{IH3}	0.7V_{DD}	–	V_{DD}	V
	V_{IH2}	P1, P2, P3, and nRESET	0.8V_{DD}			
	V_{IH3}	X_{IN}, XT_{IN}	V_{DD}–0.1			
Input low voltage	V_{IL1}	All input pins except V_{IL2}, V_{IL3}	–	–	0.3V_{DD}	V
	V_{IL2}	P1, P2, P3, and nRESET			0.2V_{DD}	
	V_{IL3}	X_{IN}, XT_{IN}			0.1	
Output high voltage	V_{OH}	V_{DD} = 2.7 V to 3.6 V; I_{OH} = –1 mA, all output pins	V_{DD}–1.0	–	–	V
Output low voltage	V_{OL}	V_{DD} = 2.7 V to 3.6 V; I_{OL} = 8 mA, all output pins	–	–	1.0	

Table 21-2. D.C. Electrical Characteristics (Continued)

(T_A = – 25 °C to + 85 °C, V_{DD} = 1.95 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Input high leakage current	I_{LIH1}	$V_{IN} = V_{DD}$ All input pins except I_{LIH2}	–	–	3	uA
	I_{LIH2}	$V_{IN} = V_{DD}$; X_{IN}, XT_{IN}	–	–	20	
Input low leakage current	I_{LIL1}	V_{IN} = 0 V All input pins except I_{LIL2}	–	–	–3	
	I_{LIL2}	V_{IN} = 0 V; X_{IN}, XT_{IN}, nRESET			–20	
Output high leakage current	I_{LOH}	$V_{OUT} = V_{DD}$ All I/O pins and output pins	–	–	3	uA
Output low leakage current	I_{LOL}	V_{OUT} = 0 V All I/O pins and output pins	–	–	–3	
Oscillator feedback resistors	R_{OSC1}	V_{DD} = 3.0 V, T_A = 25 °C $X_{IN} = V_{DD}$, X_{OUT} = 0 V	600	1700	3000	kΩ
	R_{OSC2}	V_{DD} = 3.0 V, T_A = 25 °C $XT_{IN} = V_{DD}$, XT_{OUT} = 0 V	3000	6000	9000	
Pull-up resistor	R_{L1}	V_{IN} = 0 V; V_{DD} = 3 V ± 10% Port 0,1,2,3,4,5,6 T_A = 25 °C	40	70	140	
	R_{L2}	V_{IN} = 0 V; V_{DD} = 3 V ± 10% T_A = 25 °C, nRESET only	200	400	800	
LCD voltage dividing resistor	R_{LCD}	T_A = 25 °C	40	55	90	
LCD driving voltage (resistor bias)	VLCD	–	2.5	–	3.6	V
Middle output voltage	V_{LC1}	V_{DD} = 2.7 V to 3.6 V LCD clock = 0 Hz T_A = 25 °C	V_{DD}-0.2	V_{DD}	V_{DD}+0.2	V
	V_{LC2}		0.8V_{DD}-0.2	0.8V_{DD}	0.8V_{DD}+ 0.2	
	V_{LC3}		0.6V_{DD}-0.2	0.6V_{DD}	0.6V_{DD}+ 0.2	
	V_{LC4}		0.4V_{DD}-0.2	0.4V_{DD}	0.4V_{DD}+ 0.2	
	V_{LC5}		0.2V_{DD}-0.2	0.2V_{DD}	0.2V_{DD}+ 0.2	

Table 21-2. D.C. Electrical Characteristics (Concluded)

(T_A = – 25 °C to + 85 °C, V_{DD} = 1.95 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
COM output voltage deviation	V_{DC}	$V_{DD} = V_{LC2}$ = 3 V (V_{LCD}–COMi) Io = ± 15 µA , (i = 0–7)	–	± 60	± 120	mV
SEG output voltage deviation	V_{DS}	$V_{DD} = V_{LC2}$ = 3 V (V_{LCD}–SEGi) Io = ± 15 µA , (i = 0–19)	–	± 60	± 120	
Supply current [1]	I_{DD1} [2]	Main operating: FC enable V_{DD} = 3 V ± 10 % 4.5 MHz crystal oscillator	–	2.5	5	mA
	I_{DD2} [2]	Main operating: V_{DD} = 3 V ± 10 % 4.5 MHz crystal oscillator		2	4	
	I_{DD3} [2]	Idle mode: V_{DD} = 3 V ± 10 % 4.5 MHz crystal oscillator	–	0.3	1.0	
	I_{DD4} [3]	Sub operating: main-osc stop V_{DD} = 3 V ± 10 % 75 kHz crystal oscillator	–	40	80	uA
	I_{DD5} [3]	Sub idle mode: main-osc stop V_{DD} = 3 V ± 10 % 75 kHz crystal oscillator	–	10	20	
	I_{DD6} [4]	Main stop mode : sub-osc stop V_{DD} = 3 V ± 10 %, T_A = 25 °C	–	0.2	2	

NOTES:

1. Supply current does not include current drawn through internal pull-up resistors, LCD voltage dividing resistors, voltage level detector, ADC, DAC, FC or external output current loads.
2. I_{DD1}, I_{DD2}, and I_{DD3} includes a power consumption of sub oscillator.
3. I_{DD4} and I_{DD5} are the current when the main clock oscillation stop and the sub clock is used.
4. I_{DD6} is the current when the main and sub clock oscillation stop.
5. Every value in this table is measured when bits 2-0 of the power control register (PCON.2–.0) is set to 111B.

Table 21-3. A.C. Electrical Characteristics

(T_A = – 25 °C to + 85 °C, V_{DD} = 1.95 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Interrupt input high, low width	t_{INTH}, t_{INTL}	P1.0–P1.3, P3.0–P3.2 V_{DD} = 3 V	–	200	–	ns
nRESET input low width	t_{RSL}	V_{DD} = 3 V ± 10 %	10	–	–	µs

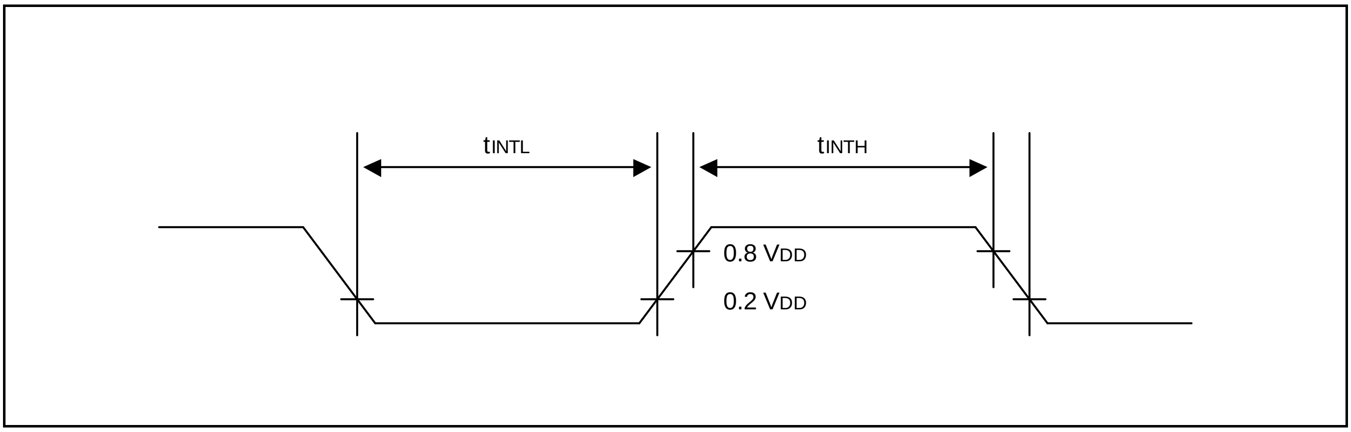


Figure 21-1. Input Timing for External Interrupts (P1, P3.0–P3.2)

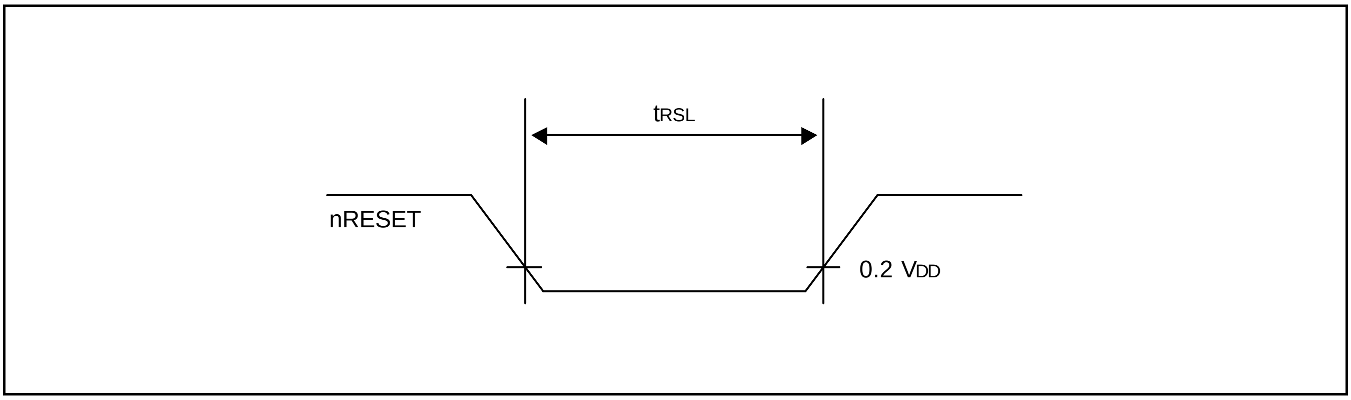


Figure 21-2. Input Timing for RESET

Table 21-4. Input/Output Capacitance

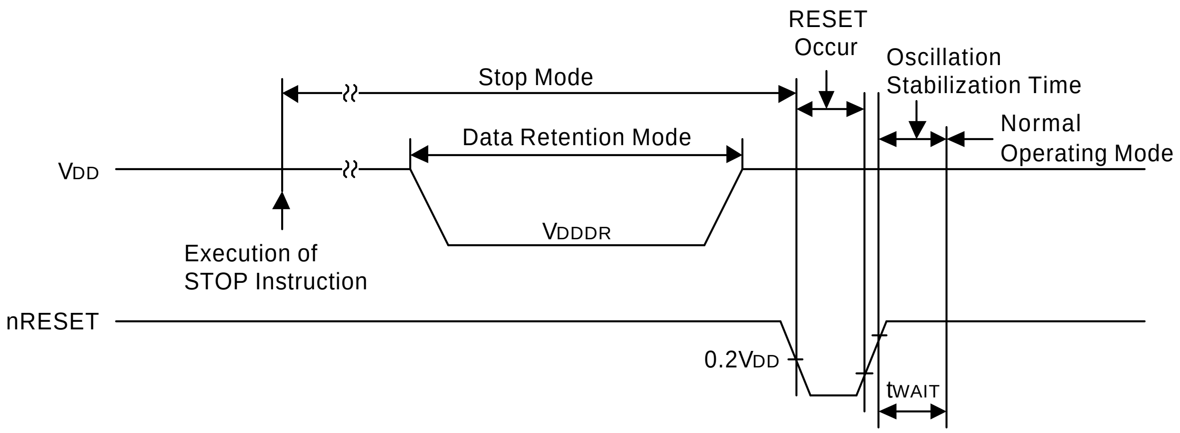
(T_A = –25 °C to + 85 °C, V_{DD} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Input capacitance	C_{IN}	f = 1 MHz; unmeasured pins are returned to V_{SS}.	–	–	10	pF
Output capacitance	C_{OUT}					
I/O capacitance	C_{IO}					

Table 21-5. Data Retention Supply Voltage in Stop Mode

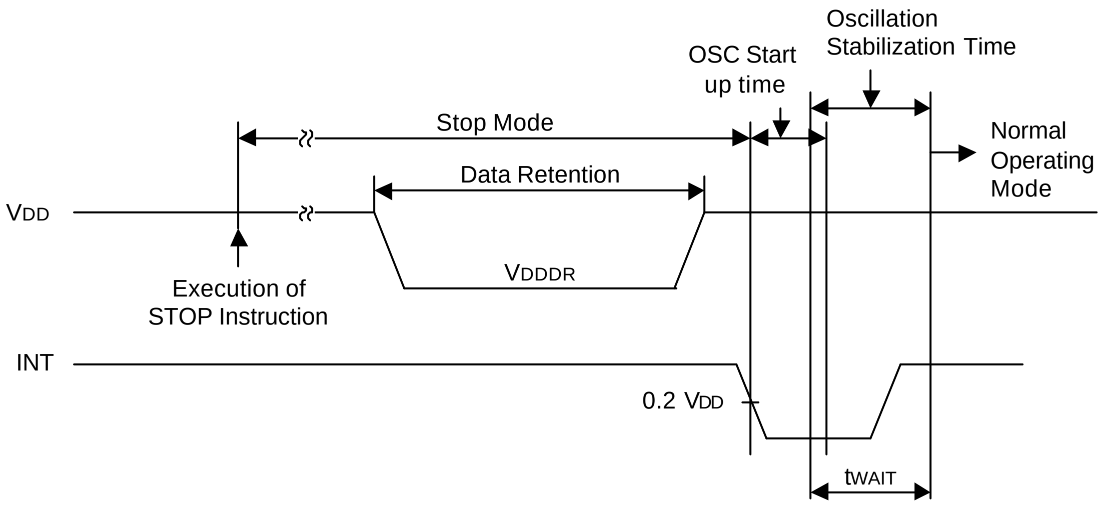
(T_A = –25 °C to + 85 °C)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	V_{DDDR}	–	1.95	–	3.6	V
Data retention supply current	I_{DDDR}	V_{DDDR} = 1.95 V	–	–	2	uA



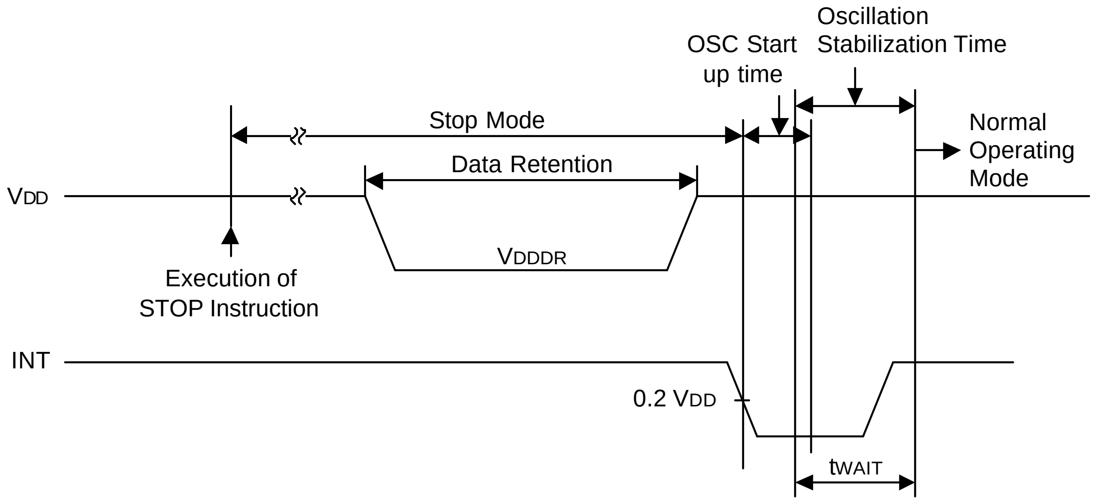
NOTE: tWAIT is same as 2048 x 32 x 1/fxx when RCOD-OPT .14-.12 = "111b".
That is, tWAIT is decided by RCOD-OPT .14-.12 when system Reset is generated.

Figure 21-3. Stop Mode Release Timing When Initiated by a RESET



NOTE: t_{WAIT} is decided by WDTCON register setting. When bit 6, 5, 4 of WDTCON is "110"b, t_{WAIT} is 1024 x 32 x 1/fxx.

Figure 21-4. Stop Mode (Main) Release Timing Initiated by Interrupts



NOTE: t_{WAIT} is same as 256 x 32 x 1/fxx. The oscillator strat up time is less than 100 ms. The value of 256 which is selected for the clock source of basic timer must be kept within this value.

Figure 21-5. Stop Mode (Sub) Release Timing Initiated by Interrupts

Table 21-6. A/D Converter Electrical Characteristics

(T_A = – 25 °C to 85 °C, V_{DD} = 2.0 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Resolution			–	10	–	bit
Total Accuracy		V_{DD} = 3.072 V	–	–	± 3	LSB
Integral Linearity Error	ILE	V_{SS} = 0 V		–	± 2	
Differential Linearity Error	DLE	fxx = 8MHz		–	± 1	
Offset Error of Top	EOT			± 1	± 3	
Offset Error of Bottom	EOB			± 0.5	± 2	
Conversion Time [(1)]	T_{CON}	10-bit resolution 50 × fxx/4, fxx = 8MHz	25	–	–	µs
Analog Input Voltage	V_{IAN}	–	V_{SS}	–	V_{DD}	V
Analog Input Impedance	R_{AN}	–	2	–	–	MΩ
Analog Input Current	I_{ADIN}	V_{DD} = 3 V	–	–	10	µA
Analog Block Current [(2)]	I_{ADC}	V_{DD} = 3 V	–	0.5	1.5	mA
		V_{DD} = 3 V When power down mode		100	500	nA

NOTES:

1. 'Conversion time' is the period between start and end of conversion operation.
2. I_{ADC} is an operating current during A/D conversion.

Table 21-7. D/A Converter Electrical Characteristics

(T_A = – 25 °C to 85 °C, V_{DD} = 2.0 V to 3.6 V, V_{SS} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Resolution	–	V_{DD} = 3.072 V	–	–	9	bit
Absolute Accuracy	–		– 3	–	3	LSB
Differential Linearity Error	DLE		– 2	–	2	LSB
Setup Time	t_{SU}		–	–	5	µs
Output Resistance	R_O		6	9	12	kΩ

Table 21-8. Characteristics of Frequency Counter

(T_A = – 25 °C to + 85 °C, V_{DD} = 2.0 V to 3.6 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Units
Input voltage (peak to peak)	V_{IN}	AMF/FMF mode, sine wave input	0.3	–	V_{DD}	V
Frequency	f_{AMF}	AMF mode, sine wave input; V_{IN} = 300mV_{P-P} FCCON.4–.2 = 100b	0.5	–	5	MHz
		AMF mode, sine wave input; V_{IN} = 300mV_{P-P} FCCON.4–.2 = 110b	5		30	
	f_{FMF}	FMF mode, sine wave input; V_{IN} = 300mV_{P-P} FCCON.4–.2 = 101b	30		130	

Table 21-9. Characteristics of Battery Level Detect Circuit

(T_A = 25 °C)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Operating voltage of BLD	V_{DDBLD}		1.95	–	3.6	V
Voltage of BLD	V_{BLD}	BLDCON.4–.2 = 100b	1.95	2.2	2.45	
		BLDCON.4–.2 = 101b	2.15	2.4	2.65	
		BLDCON.4–.2 = 110b	2.3	2.6	2.9	
Current consumption	I_{BLD}	BLD on V_{DD} = 3.0 V	–	60	100	µA
		V_{DD} = 2.0 V		40	80	
Hysteresys voltage of BLD	ΔV	BLDCON.4–.2 = 100b, 101b, 110b	–	10	100	mV
BLD circuit response time	T_B	fw = 75 kHz	–	–	1	ms

Table 21-10. LCD Contrast Level Characteristics

(T_A = – 25 °C to 85 °C, V_{DD} = 2.5 V to 3.6 V)

Parameter	Symbol	Conditions	LMOD Value	Min	Typ	Max	Unit
LCD drive voltage	V_{LC1}	Connect a 1Mohm	000	Typ.	V_{DD}	Typ.	V
		load resistor	001	x 0.9	0.96 V_{DD}	x 1.1	
		between V_{SS} and V_{LC1}	010		0.92 V_{DD}		
		(No panel load)	011		0.86 V_{DD}		
			100		0.80 V_{DD}		
			101		0.75 V_{DD}		
			110		0.70 V_{DD}		

Table 21-11. Synchronous SIO Electrical Characteristics

(T_A = – 25 °C to + 85 °C, V_{DD} = 1.95 V to 3.6 V, V_{SS} = 0 V, fxx = 8 MHz oscillator)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
SCK Cycle time	T_{CYC}	–	250	–	–	ns
Serial Clock High Width	T_{SCKH}	–	75	–	–	
Serial Clock Low Width	T_{SCKL}	–	75	–	–	
Serial Output data delay time	T_{OD}	–	–	–	65	
Serial Input data setup time	T_{ID}	–	50	–	–	
Serial Input data Hold time	T_{IH}	–	125	–	–	

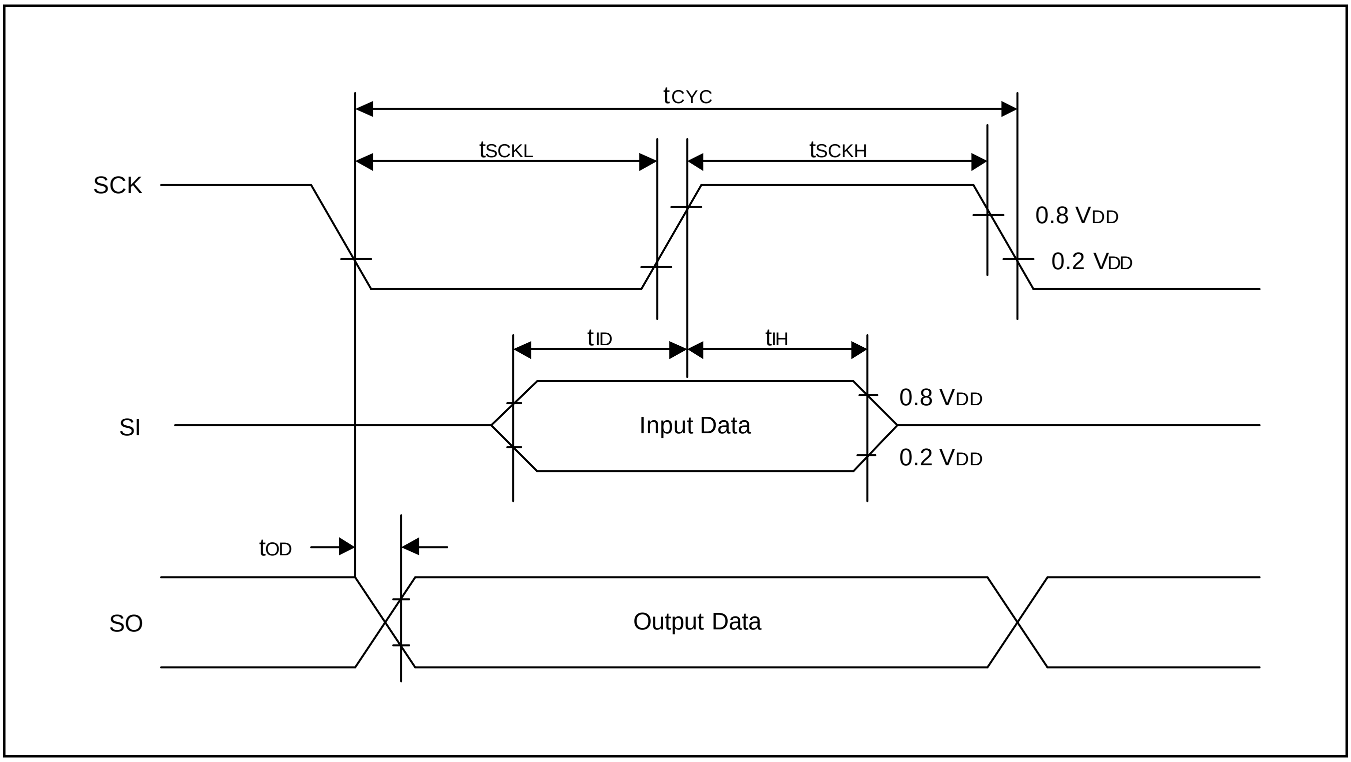
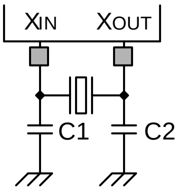
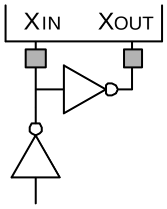
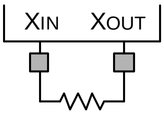


Figure 21-6. Serial Data Transfer Timing

Table 21-12. Main Oscillator Frequency ($f_{OSC}1$)

(T_A = – 25 °C to + 85 °C V_{DD} = 1.95 V to 3.6 V)

Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
Crystal/Ceramic		V_{DD} = 1.95 V – 3.6 V	0.4	–	2.0	MHz
		V_{DD} = 2.3 V – 3.6 V			8.0	
External clock		V_{DD} = 1.95 V – 3.6 V	0.4	–	2.0	MHz
		V_{DD} = 2.3 V – 3.6 V			8.0	
RC		V_{DD} = 3 V	0.4	–	1	MHz

NOTE: Oscillation frequency and Xin input frequency data are for oscillator characteristics only.

Table 21-13. Main Oscillator Clock Stabilization Time (T_{ST1})

(T_A = – 25 °C to + 85 °C, V_{DD} = 1.95 V to 3.6 V)

Oscillator	Test Condition	Min	Typ	Max	Unit
Crystal	V_{DD} = 1.95 V to 3.6 V	–	–	30	ms
Ceramic	V_{DD} = 1.95 V to 3.6 V	–	–	10	ms
External clock	XIN input high and low level width (t_{XH}, t_{XL})	62.5	–	1250	ns

NOTE: Oscillation stabilization time (T_{ST1}) is the time required for the CPU clock to return to its normal oscillation frequency after a power-on occurs, or when Stop mode is ended by a nRESET signal.

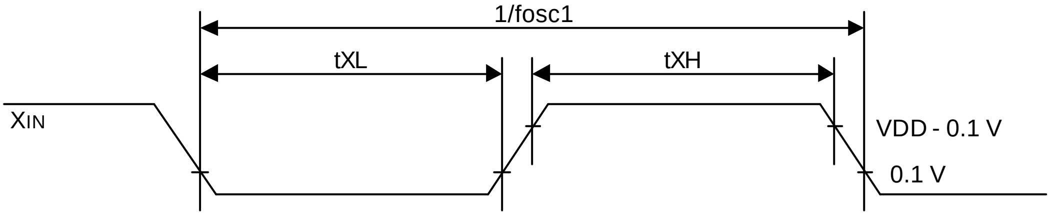
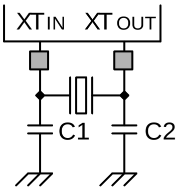
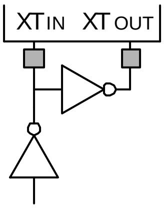


Figure 21-7. Clock Timing Measurement at X_{IN}

Table 21-14. Sub Oscillator Frequency (f_{OSC2})

(T_A = – 25 °C to + 85 °C, V_{DD} = 1.95 V to 3.6 V)

Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
Crystal		–	32	–	100	kHz
External clock		–	32	–	100	kHz

NOTE: Oscillation frequency and Xtin input frequency data are for oscillator characteristics only.

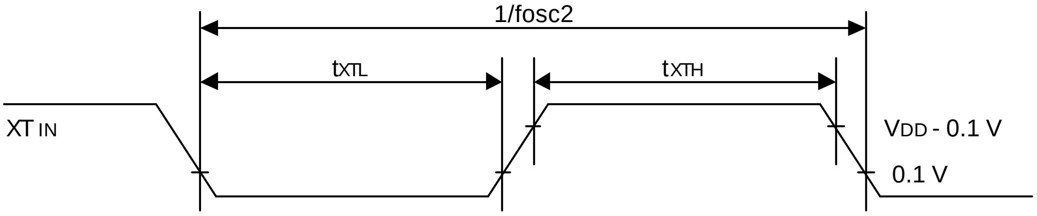


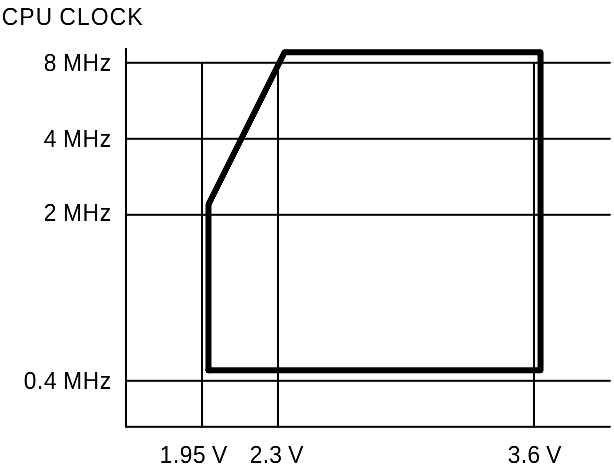
Figure 21-8. Clock Timing Measurement at XT_{IN}

Table 21-15. Sub Oscillator (Crystal) Start up Time (t_{ST2})

(T_A = – 25 °C to + 85 °C, V_{DD} = 1.95 V to 3.6 V)

Oscillator	Test Condition	Min	Typ	Max	Unit
Normal drive	V_{DD} = 1.95 V to 3.6 V	–	–	10	sec
External clock	XT_{IN} input high and low level width (t_{XTH}, t_{XTL})	5	–	15	us

NOTE: Oscillator stabilization time (t_{ST2}) is the time required for the oscillator to it's normal oscillation when stop mode is released by interrupts.



Supply Voltage (V)

Minimum instruction clock = oscillator frequency

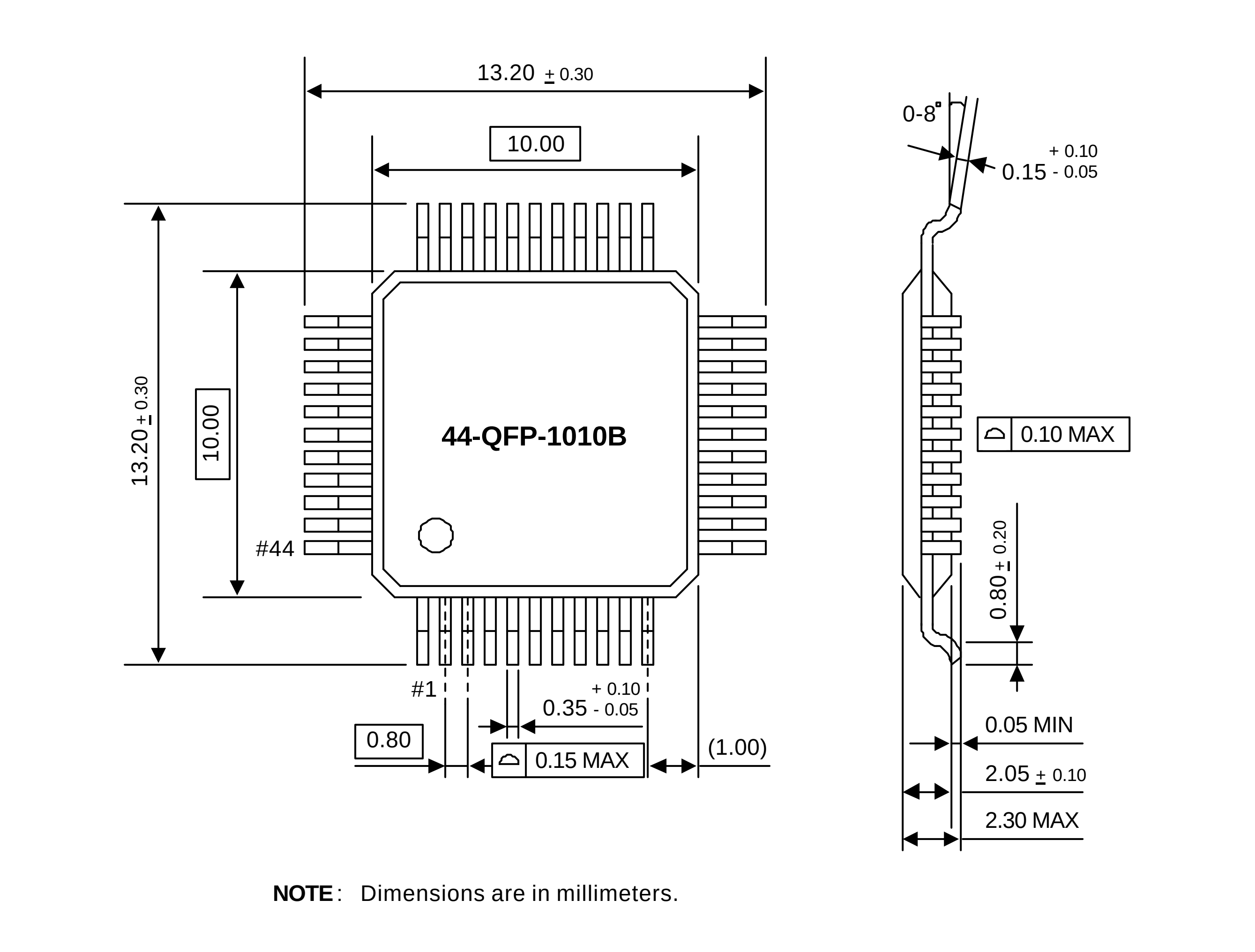
Figure 21-9. Operating Voltage Range

NOTES

22 MECHANICAL DATA

OVERVIEW

The S3CK318/FK318 is available in 44-QFP-1010B package.



NOTE: Dimensions are in millimeters.

Figure 22-1. 44-Pin QFP Package Dimensions (44-QFP-1010B)

NOTES

S3CK SERIES MASK ROM ORDER FORM

Product description:

Device Number: S3CK__________- ___________ (write down the ROM code number)

Product Order Form: □ Package □ Pellet □ Wafer Package Type: __________

Package Marking (Check One):

□ Standard	□ Custom A (Max 10 chars)	□ Custom B (Max 10 chars each line)
SEC @ YWW Device Name	@ YWW Device Name □	@ YWW □ □

@ : Assembly site code, Y : Last number of assembly year, WW : Week of assembly

Delivery Dates and Quantities:

Deliverable	Required Delivery Date	Quantity	Comments
ROM code	–	Not applicable	See ROM Selection Form
Customer sample			
Risk order			See Risk Order Sheet

Please answer the following questions:

☞ **For what kind of product will you be using this order?**

□ New product □ Upgrade of an existing product

□ Replacement of an existing product □ Other

If you are replacing an existing product, please indicate the former product name

( )

☞ **What are the main reasons you decided to use a Samsung microcontroller in your product?**

Please check all that apply.

□ Price □ Product quality □ Features and functions

□ Development system □ Technical support □ Delivery on time

□ Used same micom before □ Quality of documentation □ Samsung reputation

Mask Charge (US$ / Won): ____________________________

Customer Information:

Company Name: ___________________ Telephone number _________________________

Signatures: ________________________ __________________________________

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)

(Person placing the order) (Technical Manager)

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)

S3CK SERIES
REQUEST FOR PRODUCTION AT CUSTOMER RISK

Customer Information:

Company Name: ____

Department: ____

Telephone Number: __________________ Fax: ____________________

Date: __________________

Risk Order Information:

Device Number: S3CK________- ________ (write down the ROM code number)

Package: Number of Pins: __________ Package Type: ____________________

Intended Application: ____

Product Model Number: ____

Customer Risk Order Agreement:

We hereby request SEC to produce the above named product in the quantity stated below. We believe our risk order product to be in full compliance with all SEC production specifications and, to this extent, agree to assume responsibility for any and all production risks involved.

Order Quantity and Delivery Schedule:

Risk Order Quantity: ____________________ PCS

Delivery Schedule:

Delivery Date (s)	Quantity	Comments

Signatures: ______________________________ ______________________________

(Person Placing the Risk Order) (SEC Sales Representative)

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)

S3FK SERIES FLASH FACTORY WRITING ORDER FORM

Product Description:

Device Number: S3FK________-________ (write down the ROM code number)

Product Order Form: □ Package □ Pellet □ Wafer

If the product order form is package: Package Type: ______________________

Package Marking (Check One):

□ Standard

□ Custom A
(Max 10 chars)

□ Custom B
(Max 10 chars each line)

SEC @ YWW
Device Name

@ YWW
Device Name
□

@ YWW
□
□

@ : Assembly site code, Y : Last number of assembly year, WW : Week of assembly

Delivery Dates and Quantity:

ROM Code Release Date	Required Delivery Date of Device	Quantity

Please answer the following questions:

☞ **What is the purpose of this order?**

□ New product development

□ Upgrade of an existing product

□ Replacement of an existing microcontroller

□ Other

If you are replacing an existing microcontroller, please indicate the former microcontroller name

( )

☞ **What are the main reasons you decided to use a Samsung microcontroller in your product?**

Please check all that apply.

□ Price

□ Development system

□ Used same micom before

□ Product quality

□ Technical support

□ Quality of documentation

□ Features and functions

□ Delivery on time

□ Samsung reputation

Customer Information:

Company Name: ____________________ Telephone number ___________________________

Signatures: __________________________ _____________________________________

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)

(Person placing the order) (Technical Manager)

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)