

USER'S MANUAL

KS32C65100
32-Bit RISC
Microprocessor
Revision 0

KS32C65100

32-BIT RISC

MICROPROCESSORS

USER'S MANUAL

Revision 0



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.

KS32C65100 RISC Microprocessors
User's Manual, Revision 0
Publication Number: 20-32-C65100-0599

© 1999 Samsung Electronics

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

Samsung Electronics' Microprocessor business has been awarded full ISO-14001 certification (BSI Certificate No. FM24653). 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: (02) 760-6530, (0331) 209-6530
FAX: (02) 760-6547
Home-Page URL: [Http://www.samsungsemi.com](http://www.samsungsemi.com)

Printed in the Republic of Korea

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

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

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

Preface

The *KS32C65100 RISC Microprocessor User's Manual* is designed for application designers and programmers who are using the KS32C65100 RISC Microprocessor 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 RISC Microprocessor's architecture, programming model, instruction set, memory structure, and special function registers. It has five chapters:

Chapter 1 Product Overview
Chapter 2 Programmer's Model
Chapter 3 Instruction Set

Chapter 4 Address Spaces
Chapter 5 Special Function Registers

Chapter 1, "Product Overview," is a high-level introduction to KS17C80064/C80013/F80013 with general product descriptions, as well as detailed information about individual pin characteristics and pin circuit types.

Chapter 2, "Programmer's Model," describes the important feature of the KS17C80064/C80013/F80013 programming environment.

Chapter 3, "Instruction Set," describes the features and conventions of the instruction set used for all KS17-series RISC Microprocessors. Several summary tables are presented for orientation and reference. Detailed descriptions of each instruction are presented in a standard format. Each instruction description includes one or more practical examples of how to use the instruction when writing an application program.

Chapter 4, "Address Spaces," describes program and data memory spaces, the internal register file, and register addressing.

Chapter 5, "Special Function Registers," contains overview tables for all mapped system and peripheral control register values, as well as detailed one-page descriptions in a standardized format. You can use these easy-to-read, alphabetically organized, register descriptions as a quick-reference source when writing programs.

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 KS17-series RISC Microprocessor 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, and 5. Later, you can reference the information in Part I as necessary.

Part II "hardware Descriptions," has detailed information about specific hardware components of the KS17C80064/C80013/F80013 RISC Microprocessor. Also included in Part II are electrical, mechanical, and MTP. It has 14 chapters:

Chapter 6 System Reset and Power Mode
Chapter 7 Clock Circuits
Chapter 8 Interrupts
Chapter 9 I/O Ports
Chapter 10 Real Timer
Chapter 11 Basic Timer & Watchdog Timer
Chapter 12 16-Bit Timers

Chapter 13 PWM
Chapter 14 Remocon Receive
Chapter 15 4-Bit Analog-to-Digital Converter
Chapter 16 On-Screen Display (OSD)
Chapter 17 Electrical Data
Chapter 18 Mechanical Data
Chapter 19 KS17F80013 MTP

Two order forms are included at the back of this manual to facilitate customer order for KS17C80064/C80013/F80013 RISC Microprocessors: the Mask ROM Order Form, and the Mask Option Selection Form.

You can photocopy these forms, fill them out, and then forward them to your local Samsung Sales Representative.

Table of Contents

Chapter 1 Product Overview

Introduction.....	1-1
Features	1-2
Block Diagram	1-4
Pin Assignment.....	1-5
Pin Description	1-6
KS32C65100 Special Function Registers.....	1-11

Chapter 2 Programmer's Model

Overview	2-1
Processor Operating States	2-1
Switching State.....	2-1
Memory Formats	2-1
Big-Endian Format.....	2-2
Little-Endian Format	2-2
Instruction Length	2-3
Operating Modes	2-3
Registers	2-3
The Program Status Registers	2-7
Exceptions.....	2-10
FIQ.....	2-11
IRQ.....	2-12
Interrupt Latencies	2-14
Reset.....	2-14

Chapter 3 Instruction Set

Instruction Set Summay.....	3-1
Format Summary	3-1
Instruction Summary	3-2
The Condition Field	3-3
Branch and Exchange (BX).....	3-4
Instruction Cycle Times	3-4
Assembler Syntax	3-4
Using R15 As An Operand.....	3-4
Branch and Branch With Link (B, BL)	3-6
The Link Bit	3-6
Instruction Cycle Times	3-6
Assembler Syntax	3-7

Table of Contents (Continued)

Chapter 3 Instruction Set(Continued)

Data Processing.....	3-8
CPSR Flags	3-9
Shifts.....	3-10
Immediate Operand Rotates.....	3-14
Writing To R15	3-14
Using R15 As An Operand.....	3-14
TEQ, TST, CMP and CMN Opcodes	3-14
Instruction Cycle Times	3-14
Assembler Syntax	3-15
PSR Transfer (MRS, MSR).....	3-16
Operand Restrictions.....	3-16
Reserved Bits	3-18
Instruction Cycle Times	3-18
Assembler Syntax	3-19
Multiply and Multiply-Accumulate (MUL, MLA).....	3-20
CPSR Flags	3-21
Instruction Cycle Times	3-21
Assembler Syntax	3-21
Multiply Long and Multiply-Accumulate Long (MULL,MLAL).....	3-22
Operand Restrictions	3-22
CPSR Flags	3-23
Instruction Cycle Times	3-23
Assembler Syntax	3-23
Single Data Transfer (LDR, STR).....	3-24
Offsets and Auto-Indexing	3-25
Shifted Register Offset	3-25
Bytes and Words	3-25
Use of R15	3-27
Restriction On The Use of Base Register.....	3-27
Data Aborts	3-27
Instruction Cycle Times	3-27
Assembler Syntax	3-28
Halfword and Signed Data Transfer (LDRH/STRH/LDRSB/LDRSH).....	3-30
Offsets and Auto-Indexing	3-32
Halfword Load and Stores	3-32
Signed Byte and Halfword Loads	3-32
Endianness and Byte/Halfword Selection.....	3-32
Use of R15	3-33
Data Aborts	3-33
Instruction Cycle Times	3-33
Assembler Syntax	3-34

Table of Contents (Continued)

Chapter 3 Instruction Set(Continued)

Block Data Transfer (LDM, STM)	3-36
The Register List	3-36
Addressing Modes	3-37
Address Alignment	3-37
Use of The S Bit.....	3-39
Use of R15 As The Base	3-39
Inclusion of The Base In The Register List.....	3-40
Data Aborts	3-40
Instruction Cycle Times	3-40
Assembler Syntax	3-41
Single Data Swap (SWP).....	3-43
Bytes and Words	3-43
Use of R15	3-43
Data Aborts	3-44
Instruction Cycle Times	3-44
Assembler Syntax	3-44
Software Interrupt (SWI).....	3-45
Return From The Supervisor	3-45
Comment Field.....	3-45
Instruction Cycle Times	3-45
Assembler Syntax	3-46
Coprocessor Data Operations (CDP)	3-47
Coprocessor Instructions	3-47
The Coprocessor Fields.....	3-48
Instruction Cycle Times	3-48
Assembler Syntax	3-48
Coprocessor Data Transfers (LDC, STC).....	3-49
The Coprocessor Fields.....	3-49
Addressing Modes	3-50
Address Alignment	3-50
Use of R15	3-50
Data Aborts	3-50
Instruction Cycle Times	3-50
Assembler Syntax	3-51
Coprocessor Register Transfers (MRC, MCR).....	3-52
The Coprocessor Fields.....	3-52
Transfers To R15	3-53
Transfers From R15	3-53
Instruction Cycle Times	3-53
Assembler Syntax	3-53

Table of Contents (Continued)

Chapter 3 Instruction Set(Continued)

Undefined Instruction	3-54
Instruction Cycle Times	3-54
Assembler Syntax	3-54
Instruction Set Examples	3-55
Using The Conditional Instructions.....	3-55
Pseudo-Random Binary Sequence Generator	3-57
Multiplication by Constant Using The Barrel Shifter	3-57
Loading A Word From An Unknown Alignment	3-59
Thumb Instruction Set Format.....	3-60
Format Summary	3-60
OP code Summary	3-61
Format 1: Move Shifted Register	3-63
Format 1: Move Shifted Register	3-63
Format 2: Add/Subtract.....	3-64
Format 3: Move/Compare/Add/Subtract Immediate	3-65
Format 4: ALU Operations	3-66
Format 5: Hi-Register Operations/Branch Exchange	3-68
Format 6: PC-Relative Load	3-71
Format 7: Load/Store With Register Offset	3-72
Format 8: Load/Store Sign-Extended Byte/Halfword	3-74
Format 9: Load/Store With Immediate Offset.....	3-76
Format 10: Load/Store Halfword	3-78
Format 11: SP-Relative Load/Store	3-79
Format 12: Load Address.....	3-80
Format 13: Add Offset To Stack Pointer	3-82
Format 14: Push/Pop Registers	3-83
Format 15: Multiple Load/Store.....	3-85
Format 16: Conditional Branch	3-86
Format 17: Software Interrupt	3-88
Format 18: Unconditional Branch.....	3-89
Format 19: Long Branch With Link.....	3-90
Instruction Set Examples	3-92
Multiplication by A Constant Using Shifts and Adds	3-92
General Purpose Signed Divide.....	3-93
Division by A Constant	3-95

Table of Contents (Continued)

Chapter 4 System Manager

Overview	4-1
System Manager Registers (SMR)	4-1
System Register Address Configuration Register (SYSCFG)	4-4
ROM Control Register	4-6
SRAM Control Registers	4-11
DRAM Control Registers	4-14
DRAM Refresh Control Register	4-17
Extra Bank Access Control Registers	4-21
A.C Electrical Characteristics	4-24

Chapter 5 Cache Controller

Overview	5-1
Cache Operation	5-3
Cache Control Registers	5-5

Chapter 6 Derasterizer

Overview	6-1
Rotation	6-1
Shift Control Register (SFTCON)	6-2

Chapter 7 General ADC

Overview	7-1
Functions	7-1
SAR (Successive Approximation Register) A/D Converter Operation	7-2
Comparator (COMP) and DAC (Digital To Analog Converter)	7-2
Special Register	7-3

Chapter 8 Timer

Overview	8-1
Timer Control Register	8-2
Timer Count Value Register	8-3

Table of Contents (Continued)

Chapter 9 DMA

Overview	9-1
DMA Operation.....	9-2
Data Transfers Mode	9-3
General DMA Control Register	9-5
GDMA Source/Destination Address Register	9-8
CDMA Control Register	9-9

Chapter 10 Parallel Port Interface

Overview	10-1
KS32C65100 PPIC Operating Modes	10-2
PPIC Special Registers.....	10-5
Parallel Port Data Register	10-5
Parallel Port Status Register.....	10-6
Parallel Port ACK Width Register	10-9
Parallel Port Control Register	10-10
Parallel Port Interrupt Event Registers (PPINTEN, PPINTPND).....	10-14

Chapter 11 UART

Overview	11-1
UART Operation.....	11-2
UART Special Registers	11-6
Timing Diagrams	11-15

Chapter 12 Tone Generator

Overview	12-1
Tone Generator Data Register (TONDATA).....	12-1

Chapter 13 Watchdog Timer

Overview	13-1
Watchdog Timer Counter Register	13-2
Watchdog Timer Control Register	13-3

Table of Contents (Continued)

Chapter 14 I/O Ports

Overview	14-1
I/O Port Special Registers.....	14-1
I/O Port Mode Register.....	14-3
Input Port Mode Register.....	14-3
Output A Port Mode Register.....	14-4
Output B Port Mode Register.....	14-4
I/O Port Data Register	14-5
Input Port Data Register	14-5
Output A Port Data Register	14-6
Output B Port Data Register	14-6
Test Control Register.....	14-7
External Interrupt Control Register	14-8
Test Pin Setting	14-8

Chapter 15 Interrupt Controller

Overview	15-1
Interrupt Sources	15-2
Special Register	15-3

Chapter 16 LF Motor

Overview	16-1
Special Function Register.....	16-1
Line Feed Motor Control Register	16-1
Line Feed Motor Phase Control Register	16-3
Line Feed Timer Register	16-4
LFCON Each Control Register.....	16-5
Phase State and Current Table For Full/Half/Quarter Step Mode	16-6

Chapter 17 CR Control

Overview	17-1
Special Function Register.....	17-3
CR_PWM Timer.....	17-8
Encoder Counter	17-9
Interrupt Interval Counter	17-10
Suggestions For Carrier Motor Drive F/W Design	17-11

Table of Contents (Continued)

Chapter 18 CR Fire

Overview	18-1
Special Function Register	18-2
Position & Fire Control Register	18-2
CR Position and Fire Control Register	18-3
Suggestions For F/W Design	18-4

Chapter 19 Print Head

Overview	19-1
Special Function Register	19-1
Print Head Control Register	19-1
Fire Enable Timer/Observation Register	19-3
Fire Window Timer/Observation Register	19-3
Fire Strobe Delay Timer/Observation Register	19-4
Pre-Heat Pulse Timer/Observation Register	19-5
Pre-Heat Delay Timer/Observation Register	19-5
Printhead Observation Register	19-6
Front and Back End Delay Counter Register	19-7
Print Head Data Word Register	19-8
Dot Counter Register	19-9
Dot Counter Control Observation Register	19-9

Chapter 20 HDMA

Overview	20-1
HDMA Special Registers	20-1
HEAD DMA Control Register	20-1
HDMA Source Address Register	20-4
HDMA Transfer Count Register	20-4
HDMA Source/Match ADR Register	20-5

Table of Contents (Continued)

Chapter 21 Image Processor

Overview	21-1
Image Processor Special Registers	21-2
Sensor Shift Clock Control Register	21-2
Sensor SI Clock Control Register	21-3
Sensor R (GB) Led Control Register.....	21-4
IWIN Control Register	21-4
Changed IWIN Control Register	21-5
MAG/RED Ratio Control Register.....	21-5
LAT (Local Adaptive Threshold) Control Register	21-6
ADC Control Register	21-6
Operation Control Register	21-6
SRAM Control Register	21-8
SRAM Data Register	21-8
Motor Term Control Register	21-9
Motor Phase Control Register.....	21-9
Black Shading Correction Factor Register	21-10
Reduction & Magnification.....	21-12
Digital Shading Correction.....	21-14
Gamma Correction.....	21-15
Binarization	21-15
ADC Control.....	21-16
Motor Control	21-17
Register Read/Write.....	21-18
DMA Output	21-20

Chapter 22 Real Time Clock

Overview	22-1
Leap Year Generator	22-2
System Power Operation (+5v).....	22-2
Backup Battery Operation.....	22-2
Real Time Clock Registers	22-3
RTCCON Register.....	22-3
BCDSEC Counter Register.....	22-4
BCDMIN Counter Register	22-4
BCD HOUR Counter Register.....	22-5
BCDDAY Counter Register.....	22-5
BCDDATE Counter Register.....	22-6
BCDMON Counter Register.....	22-6
BCDYEAR Counter Register	22-7

Table of Contents (Concluded)

Chapter 23 Clock Save and PLL Control

Overview	23-1
Registers	23-1
CLKSAVCON Register	23-2
PLLCON Register	23-2

Chapter 24 LSU Control

Introduction	24-1
Main Input/Output Signals	24-2
Special Register	24-3
LSU_CON Control Register	24-3
V_Window Start/End Time Register	24-4
LD_ON Pre/Post Time Register	24-4
V_Window Counter Observation Register	24-5
LSU Motor Clock Generation Counter Register	24-5

Chapter 25 Printer Interface Controller

Overview	25-1
Page Image Data Fetch Operation	25-2
Print Operation	25-4
PIFC Special Registers	25-5
PDMA and Engine Interface Status Register	25-5
Video Control Register	25-6
Pattern Control Register	25-8
Printer Dma Control Register	25-11
Top Margin Register	25-13
Left Margin Register	25-14
Pixel Count Register	25-14
Queue 0/1 Start Address Registers	25-15
Queue 0/1 Transfer Count Registers	25-16
F- θ Lens Compensation Control Register	25-17
F- θ Compensation Table Start Address	25-18
F- θ Compensation Table Data Register	25-19
Toner Counter Setting Register	25-19
Toner Count Register	25-20

Table of Contents (Concluded)

Chapter 26 Variable Image Scailing

Overview	26-1
Algorithm.....	26-2
Example of VIS Operation	26-3
Halftoning.....	26-4
Special Register	26-5

Chapter 27 PWM Timer Control

Introduction.....	27-1
Main Input/Output Signals	27-1
Special Function Register.....	27-2

Chapter 28 Mechanical Data

Package Dimensions	28-1
--------------------------	------

Chapter 29 Evaluation Board

Introduction.....	29-1
System Requirements	29-1
Board Components.....	29-2
Bootng Systemt.....	29-4
EmbeddedICE Unit Installation	29-5
EmbeddedICE Unit.....	29-5
Connecting KS32C65100 Evaluation Board and PC	29-5
Powering Up The Board and EmbeddedICE	29-6
Debug Application With EmbeddedICE	29-6
Switch and Jumpers Description	29-7

List of Figures

Figure Number	Title	Page Number
1-1	KS32C65100 Block Diagram	1-4
1-2	Pin Assignments.....	1-5
2-1	Big-Endian Addresses of Bytes within Words	2-2
2-2	Little-Endian Addresses of Bytes within Words	2-2
2-3	Register Organization in ARM State	2-4
2-4	Register Organization in THUMB State	2-5
2-5	Mapping of THUMB State Registers onto ARM State Registers.....	2-6
2-6	Program Status Register Format	2-7
3-1	ARM Instruction Set Format	3-1
3-2	Branch and Exchange Instructions	3-4
3-3	Branch Instructions.....	3-6
3-4	Data Processing Instructions	3-8
3-5	ARM Shift Operations.....	3-10
3-6	Logical Shift Left	3-10
3-7	Logical Shift Right	3-11
3-8	Arithmetic Shift Right	3-11
3-9	Rotate Right	3-12
3-10	Rotate Right Extended	3-12
3-11	PSR Transfer	3-17
3-12	Multiply Instructions.....	3-20
3-13	Multiply Long Instructions	3-22
3-14	Single Data Transfer Instructions.....	3-24
3-15	Little-Endian Offset Addressing	3-26
3-16	Halfword and Signed Data Transfer with Register Offset	3-30
3-17	Halfword and Signed Data Transfer with Immediate Offset and Auto-Indexing	3-31
3-18	Block Data Transfer Instructions.....	3-36
3-19	Post-Increment Addressing.....	3-37
3-20	Pre-Increment Addressing	3-38
3-21	Post-Decrement Addressing	3-38
3-22	Pre-Decrement Addressing.....	3-39
3-23	Swap Instruction.....	3-43
3-24	Software Interrupt Instruction.....	3-45
3-25	Coprocessor Data Operation Instruction	3-47
3-26	Coprocessor Data Transfer Instructions.....	3-49
3-27	Coprocessor Register Transfer Instructions	3-52
3-28	Undefined Instruction	3-54
3-29	THUMB Instruction Set Formats.....	3-60

List of Figures (Continued)

Figure Number	Title	Page Number
3-30	Format 1.....	3-63
3-31	Format 2.....	3-64
3-32	Format 3.....	3-65
3-33	Format 4.....	3-66
3-34	Format 5.....	3-68
3-35	Format 6.....	3-71
3-36	Format 7.....	3-72
3-37	Format 8.....	3-74
3-38	Format 9.....	3-76
3-39	Format 10.....	3-78
3-40	Format 11.....	3-79
3-41	Format 12.....	3-80
3-42	Format 13.....	3-82
3-43	Format 14.....	3-83
3-44	Format 15.....	3-85
3-45	Format 16.....	3-86
3-46	Format 17.....	3-88
3-47	Format 18.....	3-89
3-48	Format 19.....	3-90
4-1	System Memory Map (Default Map After Reset).....	4-2
4-2	System Memory Map.....	4-3
4-3	Special Function Register Address Configuration Register	4-4
4-4	ROM Control Register (ROMCON)	4-6
4-5	The Byte Swap Operation of BTU and the Positions of Data in Memory	4-8
4-6	Simple ROM Access Timing.....	4-10
4-7	Page Mode ROM Access Timing.....	4-10
4-8	SRAM Control Registers.....	4-11
4-9	External Address Bus Generation (ADDR[21:0])	4-12
4-11	SRAM Read Timing.....	4-13
4-12	SRAM Write Timing	4-13
4-13	DRAM Control Registers (DRAMCON0 - DRAMCON1)	4-15
4-14	DRAM Bank Read Timing (Page Mode)	4-16
4-15	DRAM Bank Write Timing (Page Mode)	4-16
4-16	DRAM Refresh Control & Memory Configuration Register (DRAM Refresh Control).....	4-17
4-17	Self Refresh Mode Entry Process by nRESET	4-18
4-18	Self Refresh Mode Entry Process by Software.....	4-19
4-19	DRAM Refresh Timing.....	4-20
4-20	Special I/O Address Map	4-21
4-21	Extra Bank Control Registers (ExtCntr 0, 1, 2, 3).....	4-22
4-22	Extra Bank Read Timing (tcoh = 1, tacc = 4, tcas = 1, tacs = 2).....	4-23
4-23	Extra Bank Write Timing	4-23
4-24	An Example of System Manager Register Settings.....	4-25

List of Figures (Continued)

Figure Number	Title	Page Number
5-1	Cache Memory Configuration	5-2
5-2	CS-bit Status Diagram.....	5-3
5-3	Write Buffer Configuration.....	5-4
5-4	Non-Cacheable Area Register	5-5
6-1	Shift Control Register	6-2
6-2	Rotation Configuration.....	6-2
7-1	Functional Block Diagram of General ADC.....	7-1
7-2	ADC Control Register (ADCCON).....	7-4
7-3	ADC Data Register (ADCDATA).....	7-5
8-1	16-Bit Timer Block Diagram	8-1
8-2	Timer Control Register	8-2
8-3	Timer Count Value Register	8-3
8-4	Timer Programming Sequence.....	8-3
9-1	GDMA/CDMA Unit Block Diagram.....	9-1
9-2	External DMA Requests @ Single Mode	9-3
9-3	External DAM Requests @ Block Mode	9-4
9-4	External DMA Requests @ Demand Mode.....	9-4
9-5	GDMA Control Register.....	9-7
9-6	GDMA Source/Destination Address Register	9-8
9-7	GDMA Transfer Count Register	9-8
9-8	CDMA Control Register	9-11
9-9	CDMA Source/Destination Address Register	9-12
9-10	CDMA Transfer Count Register	9-12
10-1	Real Timer Block Diagram	10-1
10-2	Real Time Clock Control Register (RTCON).....	10-2
11-1	UART Block Diagram	11-1
11-2	UART Block Diagram	11-2
11-3	UART Data Transmission Process.....	11-4
11-4	UART Data Reception Process.....	11-5
11-5	UART Line Control Register (ULCON0, 1, 2)	11-7
11-6	UART Control Register (UCON0,1,2)	11-9
11-7	UART Status Register (USTAT0,1,2).....	11-11
11-8	UART Transmit Buffer Register (UTXBUF0,1,2)	11-12
11-9	UART Receive Buffer Register (URXBUF0, 1, 2)	11-13
11-10	UART Baud Rate Divisor Register (UBRDIV0,1, 2).....	11-14
11-11	Interrupt-Based Serial I/O Timing Diagram (Tx and Rx)	11-15

List of Figures (Continued)

Figure Number	Title	Page Number
12-1	16-Bit Timer Block Diagram.....	12-2
12-2	Timer Control Register	12-5
12-3	Timer Data Registers (T0DATA, T1DATA, and T2DATA)	12-6
12-4	Timer Count Registers (T0CNT, T1CNT and, T2CNT).....	12-7
13-1	PWM Control Register (PWMCON).....	13-3
13-2	PWM Data Register (PWM0, PWM1)	13-4
13-3	Block Diagram of 14-bit PWM Output Unit.....	13-6
14-1	Bi-directional Port Mode Register (GIOPMOD)	14-3
14-2	Input Port Mode Register (GIPMOD)	14-3
14-3	Output Port Mode Register (GOPAMOD)	14-4
14-4	Output Port Mode Register (GOPBMOD)	14-4
14-5	Bi-directional Port Data Register (GIOPD)	14-5
14-6	Input Port Data Register (GIPD)	14-5
14-7	Output Port A Data Register (GOPAD)	14-6
14-8	Output Port B Data Register (GOPBD)	14-6
14-9	Test Control Register (TSTCON).....	14-7
14-10	External Interrupt Control Register (INTCON).....	14-8
15-1	Interrupt Mode Register	15-3
15-2	Interrupt Pending Register	15-4
15-3	Interrupt Mask Register	15-5
16-1	LF Motor Control Register	16-2
16-2	LF Motor Phase Control Register.....	16-3
16-3	LF Motor Timer Register.....	16-4
16-4	LFCON Register	16-5
17-1	Carrier Motor Control Register	17-3
17-2	Basic Timer Base Register	17-4
17-3	Pre-step Timer Base Register.....	17-4
17-4	CR State Control Register	17-5
17-5	CRSREG Register.....	17-6
17-5	PWM Counter Base Register.....	17-8
17-6	Encoder Cycle Register	17-9
17-7	Interrupt Interval Value Register	17-10
18-1	Position & Fire Control Register.....	18-2
18-2	CR Count Register	18-3

List of Figures (Continued)

Figure Number	Title	Page Number
19-1	Print Head Control Register	19-2
19-2	Fire Enable Timer/Observation Register	19-3
19-3	Fire Window Timer/Observation Register	19-3
19-4	Fire Strobe Delay Timer/Observation Register	19-4
19-5	Pre-Heat Pulse Timer/Observation Register	19-5
19-6	Pre-Heat Delay Timer/Observation Register	19-5
19-7	PrintHead Observation Register	19-6
19-8	Td Delay Counter Register	19-7
19-9	Print Head Data Word Register	19-8
19-10	Dot Counter Register	19-9
19-11	Dot Counter Control Observation Register	19-9
20-1	HDMA Control Register	20-3
20-2	HDMA Source Address	20-4
20-3	HDMA Transfer Count Register	20-4
20-4	HDMA Source/Match Address	20-5
21-1	Image Processor Block Diagram	21-1
21-2	Sensor Shift Clock Control Register	21-2
21-3	Sensor SI Clock Control Register	21-3
21-4	Sensor R(GB) LED Control Register	21-4
21-5	IWIN Control Register	21-4
21-6	CHANGED_IWIN Control Register	21-5
21-7	Mag/Red Ratio Control Register	21-5
21-8	LAT Control Register	21-6
21-9	ADC Control Register	21-6
21-10	Operation Control Register	21-7
21-11	SRAM Control Register	21-8
21-12	SRAM Data Register	21-8
21-13	Motor Term Control Register	21-9
21-14	Motor Phase Control Register	21-9
21-15	Block Shading Correction Factor Register	21-10
21-16	Restart & SCAN_ON Timing Diagram	21-10
21-17	Reduction Pixel Clock Timing Diagram	21-12
21-18	Magnification Pixel Clock Timing Diagram	21-13
21-19	Shading Correction Block Diagram	21-14
21-20	Gamma Correction Block Diagram	21-15
21-21	ADC Control Timing Diagram (CANON)	21-16
21-22	ADC Control Timing Diagram (DYNA)	21-17
21-23	Motor Interrupt/Phase Timing Diagram	21-17
21-24	Register Read/Write Timing Diagram	21-18
21-25	Timing Diagram SRAM Read/Write by Register	21-19

List of Figures (Continued)

Figure Number	Title	Page Number
22-1	Real Time Clock Block Diagram	22-1
22-2	RTCCON Register	22-3
22-3	BCDSEC Counter Register	22-4
22-4	BCDMIN Counter Register	22-4
22-5	BCDHOURL Counter Register	22-5
22-6	BCDDAY Counter Register	22-5
22-7	BCDDATE Counter Register	22-6
22-8	BCDMON Counter Register	22-6
22-9	BCDYEAR Counter Register	22-7
23-1	Clock Save Block Diagram	23-1
23-2	CLKSAVCON	23-2
23-3	PLLCON	23-2
24-1	LSU Control	24-1
24-2	LSU_CON Control Register	24-3
24-3	V-Window Time Start/End Register	24-4
24-4	LDON_Pre/Post Time Register	24-4
24-5	V-Window Counter Observation Register	24-5
24-6	LSU CLK Counter Base/Observation Register	24-5
25-1	Queued Operation for End-of-Page (EOP)	25-2
25-2	Queued Operation for Page Under-run (PUR)	25-3
25-3	Protocol Diagram (PIFC and Printer Engine)	25-4
25-4	PDMA and Engine Interface Status Register (STATUS)	25-5
25-5	Video Control Register (VCON)	25-7
25-6	Pattern Control Register (PCON)	25-10
25-7	Printer DMA Control Register (PDMACON)	25-12
25-8	Top Margin Register (TOP)	25-13
25-9	Page Layout	25-13
25-10	Left Margin Register (LFT)	25-14
25-12	Queue 0/1 Start Address Registers (QSAR0, QSAR1)	25-15
25-13	Queue 0/1 Transfer Count Registers (QTCR0, QTCR1)	25-16
25-14	F-θ Compensation Control Register (FTCON)	25-17
25-15	F-θ Compensation Table Start Address (FSADDR)	25-18
25-16	F-θ Compensation Table Data Register (FDATA)	25-19
25-17	Toner Counter Setting Register (TCVAL)	25-19
25-18	Toner Count Register (TNCNT)	25-20
25-19	Test Pattern Duration (TPVAL)	25-20
25-20	Test Pattern Width (TPON)	25-21

List of Figures (Continued)

Figure Number	Title	Page Number
26-1	VIS Algorithm Description	26-2
26-2	Examples of VIS's Internal Operation	26-3
26-4	VIS Status Register (VISSR)	26-5
26-5	VIS Control Register (VISCON)	26-6
26-6	VIS Data Size Registers (DstSize, SrcSize)	26-6
26-7	VIS Data Registers (SrcReg, DstReg)	26-7
26-7	VIS Data Registers (SrcReg, DstReg)	26-8
27-1	PWM_CON Control Register	27-2
27-2	PWM Pre-Scaler Counter Base/Observation Register	27-2
27-3	PWM Cycle Time Base/Observation Register	27-3
27-4	PWM On Time Base/Observation Register	27-3
28-1	208-QFP-2828 Package Dimensions	28-1
29-1	Evaluation Board	29-3
29-3	Connection to Embedded ICE	29-4
29-4	Evaluation Board Schematic 1	29-8
29-5	Evaluation Board Schematic 2	29-9
29-6	Evaluation Board Schematic 3	29-10
29-7	Evaluation Board Schematic 2	29-11
29-8	Evaluation Board Schematic 2	29-12

List of Tables

Table Number	Title	Page Number
1-1	Pin Description	1-6
1-2	KS32C65100 Special Function Registers	1-11
2-1	PSR Mode bit Values	2-9
2-2	Exception Entry/Exit.....	2-11
2-3	Exception Vectors	2-13
3-1	The ARM Instruction Set	3-2
3-2	Condition Code Summary	3-3
3-3	ARM Data Processing Instructions.....	3-9
3-4	Incremental Cycle Times	3-14
3-5	Assembler Syntax Descriptions	3-23
3-6	Addressing Mode Names.....	3-41
3-7	THUMB Instruction Set Opcodes	3-61
3-7	THUMB Instruction Set Opcodes (Continued).....	3-62
3-8	Summary of Format 1 Instructions.....	3-63
3-9	Summary of Format 2 Instructions.....	3-64
3-10	Summary of Format 3 Instructions.....	3-65
3-11	Summary of Format 4 Instructions.....	3-66
3-12	Summary of Format 5 Instructions.....	3-68
3-13	Summary of PC-Relative Load Instruction.....	3-71
3-36	Summary of Format 7 Instructions.....	3-72
3-15	Summary of Format 8 Instructions.....	3-74
3-16	Summary of Format 9 Instructions.....	3-76
3-17	Halfword Data Transfer Instructions.....	3-78
3-18	SP-Relative Load/Store Instructions	3-79
3-19	Load Address	3-80
3-20	The ADD SP Instructions.....	3-82
3-21	PUSH and POP Instructions.....	3-83
3-22	The Multiple Load/Store Instructions.....	3-85
3-23	The Conditional Branch Instructions	3-86
3-24	The SWI Instruction.....	3-88
3-25	Summary of Branch Instruction	3-89
3-26	The BL Instruction	3-91
4-1	The Relations Between Physical Address and Address in Instructions	4-9
6-1	Set 1 Register Values after a Reset.....	6-4

List of Tables (Continued)

Table Number	Title	Page Number
9-1	Difference Between GDMA and CDMA.....	9-3
12-1	Timer Control Register Description	12-3
12-2	Timer Data Registers Description	12-6
12-3	Timer Count Registers description	12-7
13-1	PWM0 and PWM1 Control and Data Registers.....	13-5
13-2	PWM Output (Stretch) Values for Extension Registers PWM0 and PWM1	13-5
14-1	I/O Port Mode Configuration and Settings.....	14-1
15-1	Interrupt Sources	15-2
29-1	Jumper Description.....	29-7
29-2	Switch Description	29-7

1

PRODUCT OVERVIEW

INTRODUCTION

Samsung's KS32C65100 16/32-bit RISC micro controller is designed to provide a cost-effective and high performance micro controller solution for ink-jet/laser-jet printers and MFP.

An outstanding feature of the KS32C65100 is its CPU core, a 16/32-bit RISC processor (ARM7TDMI) designed by Advanced RISC Machines, Ltd. The ARM7TDMI core is a low-power, general purpose microprocessor macro-cell that was developed for use in application-specific and custom-specific integrated circuits. Its simple, elegant, and fully static design is particularly suitable for cost-sensitive and power sensitive applications.

The KS32C65100 was developed using an ARM7TDMI core, 0.35um CMOS standard cells, and a memory compiler. Most of the on-chip function blocks were designed using an HDL synthesiser.

The integrated on-chip functions that are described in this document include:

- 2KB Instruction/data cache and controller
- PLL (Phase Locked Loop)
- Clock save control
- DMA control (3 channel)
- Interrupt control
- UART (3 channel)
- 16-bit Timer (3 channel)
- PWM timer (3 channel)
- Watch dog timer
- A/D converter (8/10-bit, 3 channel)
- General I/O port control
- Scan image control
- Scan motor control
- Tone generator
- Real time clock
- Parallel Port Interface control
- Print head control
- Carrier motor control
- Paper motor control
- Laser Printer Interface control
- Laser engine control
- S/W assistant function (rotator)

FEATURES

Architecture

- Fully 16/32-bit RISC architecture
- Efficient and powerful ARM7TDMI CPU core
- Cost-effective JTAG-based debug solution

System Manager

- 16-bit external bus support for ROM.
8/16-bit external bus support for SRAM, DRAM (Fast Page, EDO) and external I/O
- Programmable access cycle (2 ~ 7 wait cycles)
- Support idle mode for low power consumption

Unified Instruction/Data cache

- Two way set associative cache with 2KB
- LRU (Least Recently Used)
- Four depth write buffer

PLL Frequency Synthesiser

- Input freq. range: 10MHz ~ 40MHz
- Jitter: ± 150 ps
- External loop filter: 820 pF

DMA (Direct Memory Access) Controller

- 3-channel DMA Controller
- Memory-to-print block with decompression
- Memory-to-memory, memory-to-parallel port, parallel-to-memory, UART-to-memory, memory-to-UART, IP-to-memory, memory-to-IP data, I/O-to-memory, memory-to-I/O data transfers without CPU intervention.
- Initiated by software or external DMA request.
- Increments or decrements source or 8-bit, 16-bit or 32-bit data transfer

Interrupts

- 31 interrupt sources (external: 3)
- Normal or fast interrupt modes (IRQ, FIQ)
- Level or edge selectable 3 external interrupt

UART (SIO)

- Three channel UART (Serial I/O) with DMA based or interrupt based operation; supports 5-bit, 6-bit, 7-bit, or 8-bit serial data transmit/receive
- Programmable baud rate
- Infra-red (IR) Tx/Rx support (IrDA)

General Timers

- Three programmable 16-bit timers

Watch Dog Timer

- 16-bit timer useful for periodic reset or interrupts

PWM Timers

- Three programmable 16-bit PWM timers with each prescaler

General ADC

- Three input 8/10-bit ADC with analog MUX (Max. conversion rate: 650KSPS @25MHz)
- ADC clock: MCLK/2 or MCLK/4

Scan Image Control

- Minimum scan line time: up to 2 ms
- Supports 200dpi or 300dpi, include 8 bit ADC
- 25-200% reduction/magnification,
- White shading & gamma correction
- LAT/EDF(2X3) Binarization, 256 Gray

Scan Motor Control

- Programmable 16 bit interval counter with interrupt
- Output phase and control signals at same time with counter interrupt

RTC (Real Time Clock) Unit

- 32.768kHz clock
- The data includes second, minute, hour, date, day, month and year

Parallel Port Interface Controller

- DMA-based or interrupt-based operation
- Supports IEEE 1284 standard communication modes (compatibility mode, nibble mode, bytes mode, and ECP mode)
- Supports ECP protocol with or without Run-Length Encoding (RLE)
- Automatic handshaking mode for any forward or reverse protocol with software/DMA

Ink Head Control

- Supports both daVinci and Babbage print head
- Printing data and fire control

Carrier Motor Control

- Supports two kind of Motor (DC and stepper)
- Motor speed calculation and compensation
- Support Full/Half/Quarter step mode for stepper

Carrier Position and Fire Control

- Fire control up to 2400 dpi, and position control by 600 dpi in DC motor mode
- Fire and position control up to 9600 dpi in step motor mode
- Carrier position interrupt for easy position control

Paper Motor Control

- Support two kind of motor driver (Uni-/Bi-polar)
- Support Full/Half/Quarter step mode

Derasterizer

- 16×16 bit block rotates by 90/270 degree
- Rotates 13 half words with selectable direction

Laser Printer Video Data Control

- Cost-effective, high-performance, DMA-based Laser printer engine interface
- Dedicated DMA for fast data transfer between page memory and the printer engine
- Consecutive zero string (blank data) output for banded bit maps(no memory access required)
- Queuing operation to facilitate smooth switching between data blocks of banded page memory
- Pixel chopping mode for fine-edged image printing
- Video data/boundary polarity definition
- Support for 2x or 4x image expansion
- Dot counter to accumulate printing dot
- Generates test pattern for laser engine

Laser Engine Control

- Controls LSU on/off data and motor clock
- Controls LSU interface signals
- Programmable external clock for LSU motor

Operating Voltage Range

- Internal logic: 3 to 3.6 Volts
- PAD: 4.75 to 5.25 Volts

Operating Frequency

- Up to 33MHz,

Package Type

- 208 pin QFP

BLOCK DIAGRAM

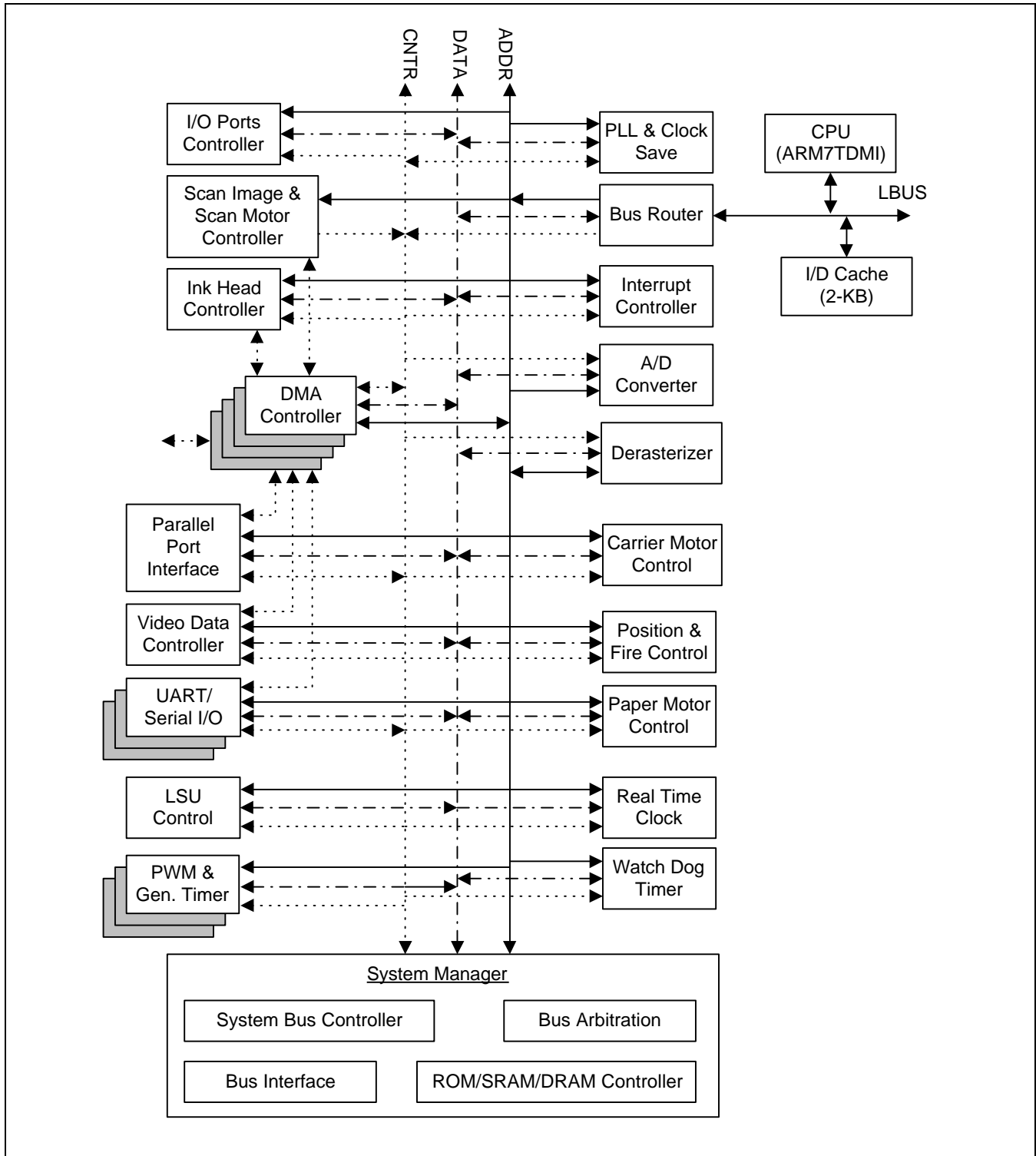


Figure 1-1. KS32C65100 Block Diagram

PIN ASSIGNMENT

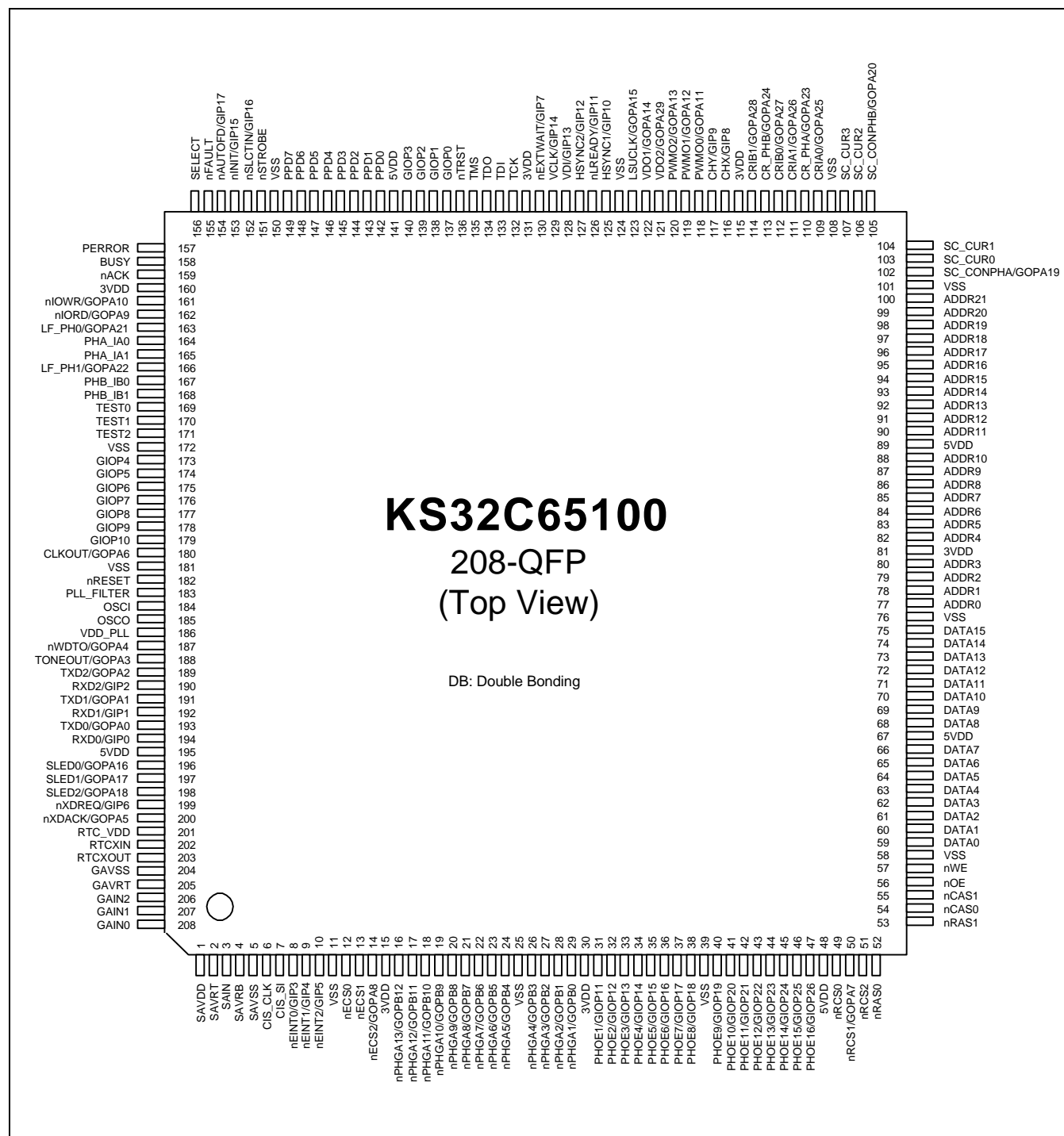


Figure 1-2. Pin Assignment

PIN DESCRIPTION

Table 1-1. Pin Description

Signal	Pin No.	I/O Type	Description
OSCI	184	I7	KS32C65100 master clock input.
OSCO	185	O7	KS32C65100 master clock output.
PLL_FILTER	183	I5	PLL filter
nRESET	182	I4	Not reset. nRESET is the global reset input for the KS32C65100. For a system reset, nRESET must be held to low level for at least 65 machine cycles.
nSLCTIN/GIP[16]	152	I1	Not select information. This input signal is used by parallel port interface to request 'on-line' status information.
nSTROBE	151	I1	Not strobe. The nSTROBE input indicates when valid data is on parallel port data bus, PPD[7:0]
nAUTOFD/GIP[17]	154	I1	Not auto feed. The nAUTOFD input indicates whether data on the parallel port data bus, PPD[7:0], is an auto feed command. Otherwise, the bus signals are interpreted as data only.
nINIT/GIP[15]	153	I1	Not initialization. The nINIT input signal initializes the parallel port's input control.
nACK	159	I1	Not parallel port acknowledge. The nACK output signal is issued whenever a transfer on the parallel port data bus is completed.
BUSY	158	O1	Parallel port busy. The BUSY output signal indicates that the KS32C65100 parallel port is currently busy.
SELECT	156	O1	Parallel port select. The SELECT output signal indicates whether the device connected to the KS32C65100 parallel port is 'on-line' or 'off-line'.
PERROR	157	O1	Parallel port paper error. PERROR output indicates that a problem exists with the paper in the ink-jet printer. It could indicate that the printer has a paper jam or that the printer is out of paper.
nFAULT	155	O1	Not fault. The nFAULT output indicates that an error condition exists with the printer. This signal can be used to indicate that the printer is out of ink or to inform the user that the printer is not turned on.
PPD[7:0]	142~149	I/O2	Parallel port data bus. This 8-bit, tri-state bus is used to exchange data between the KS32C65100 and an external host(peripheral).
SAVRT	2	I6	Top reference voltage for IP ADC
SAIN	3	I6	Analog input for IP ADC
SAVRB	4	I6	Bottom reference voltage for IP ADC

Table 1-1. Pin Description (Continued)

Signal	Pin No.	I/O Type	Description
CIS_CLK	6	O1	CIS shift clock
CIS_SI	7	O1	CIS latch signal
PHA_IA0	164	O1	Line feed motor phase signal A
PHA_IA1	165	O1	Line feed motor phase signal AZ
PHB_IB0	167	O1	Line feed motor phase signal B
PHB_IB1	168	O1	Line feed motor phase signal BZ
LF_PH0/GOPA[21]	163	O1	Line feed motor control signal 0
LF_PH1/GOPA[22]	166	O1	Line feed motor control signal 1
CR_PHA/GOPA[23]	110	O1	Direction control line for phase A
CR_PHB/GOPA[24]	113	O1	Direction control line for phase B
CRIA0/GOPA[25]	109	O1	Current control line 0 for phase A
CRIA1/GOPA[26]	111	O1	Current control line 1 for phase A
CRIB0/GOPA[27]	112	O1	Current control line 0 for phase B
CRIB1/GOPA[28]	114	O1	Current control line 1 for phase B
CHX/GIP[8]	116	I3	Encode sensor
CHY/GIP[9]	117	I3	Encode sensor
ADDR[21:0]	77~80, 82~88, 90~100	O5	Address bus. The 22bit address bus, ADDR[21:0], covers the full 4M half-words address range of each ROM/SRAM, DRAM, and external I/O bank
DATA[15:0]	59~66, 68~75	I/O3	External bi-directional 16-bit data bus.
nRAS[1:0]	52,53	O1	Not row address strobe for DRAM. The KS32C65100 supports up to two DRAM banks. One nRAS output is provided for each bank.
nCAS[1:0]	54,55	O1	Not column address strobe for DRAM. The two nCAS outputs indicate the byte selections whenever a DRAM bank is accessed.
nOE	56	O1	Not output enable. Whenever a memory access occurs, the nOE output controls the output enable port of the specific memory device.
nWE	57	O6	Not write enable. Whenever a memory access occurs, the nWE output controls the write enable port of the specific memory device.
nPHGA[13:1]/ GOPB[12:0]	16~24, 26~29	O1	Gate control line for print head.
PHOE[16:1]/ GIOP[26:11]	31~38, 40~47	I/O1	Drain control line for print head.

Table 1-1. Pin Description (Continued)

Signal	Pin No.	I/O Type	Description
RXD0/GIP[0]	194	I1	Receive data input for the UART0. RXD0 is the UART0 channel's input signal for receiving serial data.
RXD1/GIP[1]	192	I1	Receive data input for the UART1. RXD1 is the UART1 channel's input signal for receiving serial data.
RXD2/GIP[2]	190	I1	Receive data input for the UART2. RXD2 is the UART2 channel's input signal for receiving serial data.
nEINT0/GIP[3]	8	I3	External interrupt request input nEINT0.
nEINT1/GIP[4]	9	I3	External interrupt request input nEINT1.
nEINT2/GIP[5]	10	I3	External interrupt request input nEINT2.
nXDREQ/GIP[6]	199	I3	External DMA request.
TXD0/GOPA[0]	193	O1	Transmit data output for the UART0. TXD0 is the UART0 channel's output for transmitting serial data.
TXD1/GOPA[1]	191	O1	Transmit data output for the UART1. TXD1 is the UART1 channel's output for transmitting serial data.
TXD2/GOPA[2]	189	O1	Transmit data output for the UART2. TXD2 is the UART2 channel's output for transmitting serial data.
nXDACK/GOPA[5]	200	O1	External DMA acknowledge. This active low output signal is generated whenever a DMA transfer is completed.
TONEOUT/GOPA[3]	188	O1	Tone generator output.
nWDTO/GOPA[4]	187	P3	Reset out by watch dog timer.
nIOWR/GOPA[10]	161	O1	External output write strobe
nIORD/GOPA[9]	162	O1	External output read strobe
CLKOUT/GOPA[6]	180	O1	Clock for external chip
nECS2/GOPA[8]	14	O1	External memory chip select 2.
TCK	132	I2	JTAG TCK interface in MDS mode.
TMS	135	I2	JTAG TMS interface in MDS mode.
TDI	133	I2	JTAG TDI interface in MDS mode.
nTRST	136	I2	JTAG nTRST interface in MDS mode.
TDO	134	O1	JTAG TDO interface in MDS mode.
GIOP[10:0]	137~140, 173~179	I/O4	General I/O port.
TEST0	169	I2	Test 0 pin. At normal operation this pin must be connected to GND.
TEST1	170	I2	Test 1 pin. At normal operation this pin must be connected to GND.
TEST2	171	I2	Test 2 pin. At normal operation this pin must be connected to GND.

Table 1-1. Pin Description (Continued)

Signal	Pin No.	I/O Type	Description
nECS[1:0]	12,13	O1	Not external chip select. Three I/O banks are provided for external memory-mapped I/O operations. Each I/O bank contains up to 4M half-word. The nECS signals indicate that an external I/O bank is selected.
nRCS[2]	51	O2	Not ROM/SRAM chip select. The KS32C65100 can access up to three external ROM/SRAM banks. nRCS[0] corresponds to ROM/SRAM bank 0, nRCS[1] to bank 1, and nRCS[2] to bank 2. By controlling the nRCS signals, CPU addresses can be mapped into the physical memory banks.
nRCS[1]/GOPA[7]	50	O1	
nRCS[0]	49	O1	
SC_CONPHA/ GOPA[19]	102	O1	Scan motor control/Bi-phase
SC_CONPHB/ GOPA[20]	105	O1	Scan motor control/Bi-phase
SC_CUR[3:0]	103, 104, 106, 107	O1	Scan motor bi-current/uni-phase
PWMO[2:0]/ GOPA[13:11]	118~120	O1	PWM out signal
VDO2/GOPA[29]	121	O4	Video out from PIFC
VDO1/GOPA[14]	122	O5	Video out from LSU control
LSU_CLK/ GOPA[15]	123	O1	Clock for LSU motor
nHSYNC1/GIP[10]	125	I1	HSYNC1
nLREADY/GIP[11]	126	I1	LSU ready
nHSYNC2/GIP[12]	127	I1	HSYNC2
VDI/GIP[13]	128	I2	Video data input from RET
VCLK/GIP[14]	129	I2	External video clock
nEXTWAIT/GIP[7]	130	I3	External wait
RTCXIN	202	I7	RTC oscillator clock input.
RTCXOUT	203	O7	RTC oscillator clock output.
SLED[2:0]/ GOPA[18:16]	196~198	O1	CIS LED signals
GAVRT	205	I5	Top reference voltage for general ADC
GAIN[2:0]	206~208	I5	Analog inputs for general ADC
RTC_VDD	201		RTC VDD.

Table 1-1. Pin Description (Continued)

Signal	Pin No.	I/O Type	Description
VDD_PLL	186		PLL power (3.3V).
SAVDD	1		Analog power for scan ADC and general ADC (3.3V).
SAVSS	5		Scan ADC ground.
GAVSS	204		General ADC ground
3VDD	15, 30, 81, 115, 131, 160		3.3V internal power. Externally connected to the 3.3V regulator.
5VDD	48, 67, 89, 141, 195		5V I/O power. Externally connected to the VCC board plane.
VSS	11, 25, 39, 58, 76, 101, 108, 124, 150, 172, 181		System ground. Externally connected to the ground board plane.

Pin Type	Pad Type	Resistor/Drive	Description
I1	PHIL	- / -	TTL schmitt trigger level input buffer
I2	PHIT	- / -	TTL level input buffer
I3	PHILU50	50K/ -	TTL schmitt trigger level input buffer with pull-up resistor
I4	PHIS	- / -	CMOS schmitt trigger level input
I5	PICA	- / -	Analog normal input pad with seperate bulk bias
I6	PICA10	10/ -	Analog normal input pad with resistor & seperate bulk bias
I7		- / -	Master clock input.
O1	PHOB4	- /4mA	Normal output buffer
O2	PHOT4	- /4mA	Tri-State output buffer
O3	PHOD4	- /4mA	Open drain output buffer
O4	PHOB4SM	- /4mA	Normal output buffer with medium slew-rate
O5	PHOB8SM	- /8mA	Normal output buffer with medium slew-rate
O6	PHOT4SM	- /4mA	Tri-State output buffer with medium slew-rate
O7		- / -	Master clock output.
I/O1	PHBTU50T4	50K/4mA	TTL level with pull-up resistor and Tri-State output
I/O2	PHBLU50T4SM	50K/4mA	TTL schmitt trigger level input with pull-up resistor and Tri-State output with medium slew-rate
I/O3	PHBLU50T8SM	50K/8mA	TTL schmitt trigger level input with pull-up resistor and Tri-State output with medium slew-rate
I/O4	PHBLT4	- /4mA	TTL schmitt trigger level input and Tri-State output.

KS32C65100 SPECIAL FUNCTION REGISTERS

Table 1-2. KS32C65100 Special Function Registers

Group	Register	Offset	R/W	Description	Reset Value
System Manager	SYSCFG	0x0000	R/W	System register address configuration register	0x1001
	ROMCON	0x1000	R/W	ROM control register	0x02003002
	SRAMCON0	0x1004	R/W	SRAM control register 0	0x000007fc
	SRAMCON1	0x1008	R/W	SRAM control register 1	0x000007fc
	EXTCON0	0x100c	R/W	I/O bank0 control register	0x00000000
	EXTCON1	0x1010	R/W	I/O bank1 control register	0x00000000
	EXTCON2	0x1014	R/W	I/O bank2 control register	0x00000000
	EXTCON3	0x1018	R/W	I/O bank3 control register	0x00000000
	DRAMCON0	0x101c	R/W	DRAM control register 0	0x00000000
	DRAMCON1	0x1020	R/W	DRAM control register 1	0x00000000
	REFCON	0x1024	R/W	DRAM refresh control	0x00000001
Cache	CACHNAB0	0x0004	R/W	Non-cacheable area begin 0	0x00000000
	CACHNAE0	0x0008	R/W	Non-cacheable area end 0	0x00000000
	CACHNAB1	0x000c	R/W	Non-cacheable area begin 1	0x00000000
	CACHNAE1	0x0010	R/W	Non-cacheable area end 1	0x00000000
Derasterizer	SFTCON	0x5004	R/W	Shift control register	0x04
	DRAST0	0x4800	R/W	Derasterizer data register 0	0xFFFF
	DRAST1	0x4804	R/W	Derasterizer data register 1	0xFFFF
	DRAST2	0x4808	R/W	Derasterizer data register 2	0xFFFF
	DRAST3	0x480c	R/W	Derasterizer data register 3	0xFFFF
	DRAST4	0x4810	R/W	Derasterizer data register 4	0xFFFF
	DRAST5	0x4814	R/W	Derasterizer data register 5	0xFFFF
	DRAST6	0x4818	R/W	Derasterizer data register 6	0xFFFF
	DRAST7	0x481c	R/W	Derasterizer data register 7	0xFFFF
	DRAST8	0x4820	R/W	Derasterizer data register 8	0xFFFF
	DRAST9	0x4824	R/W	Derasterizer data register 9	0xFFFF
	DRAST10	0x4828	R/W	Derasterizer data register 10	0xFFFF
	DRAST11	0x482c	R/W	Derasterizer data register 11	0xFFFF
	DRAST12	0x4830	R/W	Derasterizer data register 12	0xFFFF
	DRAST13	0x4834	R/W	Derasterizer data register 13	0xFFFF
	DRAST14	0x4838	R/W	Derasterizer data register 14	0xFFFF
	DRAST15	0x483c	R/W	Derasterizer data register 15	0xFFFF

Table 1-2. KS32C65100 Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
Timer	TCON	0x3000	R/W	System timers control register	0x000
	TBCNT0	0x3004	R/W	Timer base/count register 0	0xFFFF
	TBCNT1	0x3008	R/W	Timer base/count register 1	0xFFFF
	TBCNT2	0x301c	R/W	Timer base/count register 2	0xFFFF
DMA	DMACON0	0x8800	R/W	CDMA control register	0x00000
	DMASRC0	0x8804	R/W	CDMA source address register	0xFFFFFFFF
	DMADST0	0x8808	R/W	CDMA destination address register	0xFFFFFFFF
	DMACNT0	0x880c	R/W	CDMA transfer count register	0xFFFFFFFF
	DMACON1	0x9000	R/W	GDMA control register	0x0000
	DMASRC1	0x9004	R/W	GDMA source address register	0xFFFFFFFF
	DMADST1	0x9008	R/W	GDMA destination address register	0xFFFFFFFF
	DMACNT1	0x900c	R/W	GDMA transfer count register	0xFFFFFFFF
Parallel port	PPDATA	0x8000	R/W	Parallel port data register	0x100
	PPSTAT	0x8004	R/W	Parallel port status register	0x7e8
	PPACKWTH	0x8008	R/W	Parallel port acknowledge width register	0XXX
	PPCON	0x800c	R/W	Parallel port control register	0x0000
	PPINTEN	0x8010	R/W	Parallel port enable interrupt event register	0x000
	PPINTPND	0x8014	R/W	Parallel port interrupt pending register	0x000
HDMA	HDCON	0x7800	R/W	HDMA control register	0x0000000
	HDSAR	0x7804	R/W	HDMA source address register	0x0000000
	HDTCR	0x780c	R/W	HDMA transfer count register	0x000000
	HDSAR0	0x7814	R/W	HDMA source address register 0	0x0000000
	HDMAR0	0x7818	R/W	HDMA match address register 0	0x0000000
	HDSAR1	0x781c	R/W	HDMA source address register 1	0x0000001
	HDMAR1	0x7820	R/W	HDMA match address register 1	0x0000000
Tone Generator	TONDATA	0x3804	R/W	Tone generator data & control register	0x0ff
Watchdog Timer	WTCON	0x4000	R/W	Watch dog timer control register	0x21
	WTCNT	0x4004	R/W	Watch dog timer count register	0x0003

Table 1-2. KS32C65100 Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
I/O Ports	GIOPMOD	0x2800	R/W	Bi-Directional port mode register	0xffff800
	GIPMOD	0x2804	R/W	Input port mode register	0x00000
	GOPAMOD	0x2808	R/W	Output port mode register	0x00000000
	GOPBMOD	0x280c	R/W	Output port mode register	0x0000
	GIOPD	0x2810	R/W	Bi-Directional port data register	0x00000000
	GIPD	0x2814	R/W	Input port data register	0xXXXX
	GOPAD	0x2818	R/W	Output port data register	0x00000000
	GOPBD	0x281c	R/W	Output port data register	0x0000
	TSTCON	0x2820	R/W	Test control register	0x00600
	INTCON	0x2824	R/W	External interrupt control register	0x000
Interrupt Controller	INTMOD	0x2000	R/W	Interrupt mode register	0x00000000
	INTPND	0x2004	R/W	Interrupt pending register	0x00000000
	INTMSK	0x2008	R/W	Interrupt mask register	0x00000000
Real Time Clock	RTCCON	0xc840	R/W	RTC control register	0x0
	BCDSEC	0xc870	R/W	RTC second register	0xXX
	BCDMIN	0xc874	R/W	RTC minute register	0xXX
	BCD HOUR	0xc878	R/W	RTC hour register	0xXX
	BCDDAY	0xc87c	R/W	RTC day register	0xXX
	BCDDATE	0xc880	R/W	RTC date register	0xX
	BCDMON	0xc884	R/W	RTC month register	0xXX
	BCDYEAR	0xc888	R/W	RTC year register	0xXX
Clock Save	CLKSAVCON	0x1800	R/W	Clock save control register	0x0
	PLLCON	0x1804	W	PLL control register	0x00000

Table 1-2. KS32C65100 Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
UART	ULCON0	0xb000	R/W	UART Ch-0 line control register	0x00
	ULCON1	0xb800	R/W	UART Ch-1 line control register	0x00
	ULCON2	0xc000	R/W	UART Ch-2 line control register	0x00
	UCON0	0xb004	R/W	UART Ch-0 control register	0x00
	UCON1	0xb804	R/W	UART Ch-1 control register	0x00
	UCON2	0xc004	R/W	UART Ch-2 control register	0x00
	USTAT0	0xb008	R	UART Ch-0 status register	0x00
	USTAT1	0xb808	R	UART Ch-1 status register	0x00
	USTAT2	0xc008	R	UART Ch-2 status register	0x00
	UTXBUF0	0xb00c	W	UART Ch-0 transmit buffer register	0x00
	UTXBUF1	0xb80c	W	UART Ch-1 transmit buffer register	0x00
	UTXBUF2	0xc00c	W	UART Ch-2 transmit buffer register	0x00
	URXBUF0	0xb010	R	UART Ch-0 receive buffer register	0x00
	URXBUF1	0xb810	R	UART Ch-1 receive buffer register	0x00
	URXBUF2	0xc010	R	UART Ch-2 receive buffer register	0x00
	UBRDIV0	0xb014	R/W	Baud rate divisor register 0	0x0000
	UBRDIV1	0xb814	R/W	Baud rate divisor register 1	0x0000
	UBRDIV2	0xc014	R/W	Baud rate divisor register 2	0x0000
Printer Interface Controller	STATUS	0xa000	R/W	Status register	0x00
	VCON	0xa004	R/W	Video control register	0x00000000
	PCON	0xa008	R/W	Pattern control register	0x00000000
	PDMACON	0xa00c	R/W	PDMA control register	0x00
	TOP	0xa010	R/W	TOP margin register	0x0000
	LFT	0xa014	R/W	LEFT margin register	0x0000
	PXL	0xa018	R/W	Pixel count register	0x0000
	QSAR0	0xa01c	R/W	Q0 start address register	0x00000000
	QTCR0	0xa020	R/W	Q0 transfer address register	0x00000000
	QSAR1	0xa024	R/W	Q1 start address register	0x00000000
	QTCR1	0xa028	R/W	Q1 transfer address register	0x00000000
	FTCON	0xa02c	R/W	F-θ control register	0x0
	FSADDR	0xa030	R/W	F-θ start register	0x00
	FDATA	0xa034	R/W	F-θ data register	0xeffb
	TCVAL	0xa038	R/W	Toner counter set value register	0x00000000
	TNCNT	0xa03c	R/W	Tone count value register	0x00000000
	TPVAL	0xa040	R/W	Test pattern period value register	0x00
	TPON	0xa044	R/W	Test pattern on length register	0x00

Table 1-2. KS32C65100 Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
LSU	LSUCON	0xd000	R/W	LSU_CON control register	0x0000
	VWIN_STR	0xd004	R/W	V_Window time start register	0x00000
	VWIN_END	0xd008	R/W	V_Window time end register	0x00000
	LDPON_Pre	0xd00c	R/W	LDON Pre_On time register	0x00000
	LDPON_Post	0xd010	R/W	LDON Post_On time register	0x00000
	VCNT_OBS	0xd014	R/W	V_Window counter observation register	0x00000
	LSUCK_CNT	0xd018	R/W	LSU Motor Clock counter base & observation register	0x00000000
GADC	ADCCON	0xd800	R/W	ADC control register	0xa0
	ADCDATA	0xd804	R	ADC data register	0xXXX
Image Processing	SEN_CLK	0x9800	R/W	Sensor shift signal period register	0x00818
	SI_TERM	0x9804	R/W	Sensor SI signal period register	0x09c4
	RLED	0x9808	R/W	Sensor R led signal period register	0x00000960
	GLED	0x980c	R/W	Sensor G led signal period register	0x00000960
	BLED	0x9810	R/W	Sensor B led signal period register	0x00000960
	IWIN	0x9814	R/W	EAI Image area register	0x000006b8
	CHANGED_IWIN	0x9818	R/W	Magnified/reduced pixels num. register	0x06b8
	RATIO	0x981c	R/W	Magnified/reduced ratio register	0x10080
	LAT	0x9820	R/W	Local adaptive register	0xdc7f40
	ADC	0x9824	R/W	IP ADC control register	0x005
	OPERATION	0x9828	R/W	Operation control register	0x000
	RAM_CTRL	0x982c	R/W	IP inner SRAM control register	0x70000
	RAM_DATA	0x9830	R/W	IP inner SRAM data register	0x00
	MOTOR_TERM	0x9834	R/W	Motor signal period register	0x0000
	MOTOR_PHASE	0x9838	R/W	Motor signal phase register	0x00f
	BLACK	0x983c	R/W	Black shading correction value register	0x00
LF Motor	LFCR	0x5800	R/W	Line feed motor control register	0x0800
	LFPCR	0x5804	R/W	Line feed motor phase control register	0x3c0
	LFTBR	0x5808	R/W	Line feed motor timer base register	0x0000
	LFTOR	0x580c	R	Line feed motor timer observation register	0x1e0d
	LFTCBR	0x5810	R/W	Line feed motor timer compare base register	0x0000
	LFTCOR	0x5814	R	LF motor timer compare observation register	0x0000
	LFCON	0x5818	R/W	LF step each control register	0x0000

Table 1-2. KS32C65100 Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
Carrier Motor Control	CMCR	0x6000	R/W	Carrier motor control register	0x204
	BTB1R	0x6004	R/W	Basic timer base register 1	0xFFFF
	BTB2R	0x6008	R/W	Basic timer base register 2	0xFFFF
	PSTBR	0x600c	R/W	CR_Step_INT counter & pre-step counter base register	0x000
	CRSCR	0x6010	R/W	CR state control register	0x603f
	PWMOBS	0x6014	R	PWM counter observation register	0x0000
	PWMCYL	0x6018	R/W	PWM cycle time base register	0x0000
	PWMONT	0x601c	R/W	PWM on time base register	0x0000
	ECDTIM	0x6020	R	PWM on time base register	0x020292
	ECDVAL	0x6024	R	Encoder cycle value register	0x000000
	INTTIM	0x6028	R	Interval counter observation register	0x0000
	INTVAL	0x602c	R	Interrupt interval value register	0x0000
	CRSREG	0x6030	R/W	CR step each control register	0x000000
Fire & Position Control	PFCR	0x6820	R/W	Position & Fire control register	0x0080d0
	CPCR	0x6824	R/W	Carrier position counter register	0x0001
	PSPR	0x6828	R/W	Print start position register	0x0ffff
	PSCR	0x682c	R/W	Print slice counter register	0x0000
	PIR	0x6830	R/W	Position interrupt register	0xffff
Print Head	PHCR	0xa000	R/W	Print head control register	0x000000
	FETR	0x7004	R/W	Fire enable timer register	0x00
	FETOR	0x7008	R	Fire enable timer observation register	0x00
	FWTR	0x700c	R/W	Fire window timer register	0x000
	FWTOR	0x7010	R	Fire window timer observation register	0x000
	FSDTR	0x7014	R/W	Fire strobe delay timer register	0x000
	FSDT0OR	0x7018	R	Fire delay strobe timer 0 observation register	0x000
	FSDT1OR	0x701c	R	Fire delay strobe timer 1 observation register	0x000
	FSDT2OR	0x7020	R/W	Fire delay strobe timer 2 observation register	0x000
	FSDT3OR	0x7024	R/W	Fire delay strobe timer 3 observation register	0x000

Table 1-2. KS32C65100 Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
Print Head	PHPTR	0x7028	R/W	Pre-heat pulse timer register	0x00
	PHPTOR	0x702c	R	Pre-heat pulse timer observation register	0x00
	PHDTR	0x7030	R/W	Pre-heat delay timer register	0x00
	PHDTOR	0x7034	R	Pre-heat delay timer observation register	0x00
	PHOR	0x7038	R	Print head observation register	0x00000000
	TDCR	0x703c	R/W	Td delay counter register	0x00
	PHDW0R	0x7040	R/W	Print head data word 0 register	0x0000
	PHDW1R	0x7044	R/W	Print head data word 1 register	0x0000
	PHDW2R	0x7048	R/W	Print head data word 2 register	0x0000
	PHDW3R	0x704c	R/W	Print head data word 3 register	0x0000
	PHDW4R	0x7050	R/W	Print head data word 4 register	0x0000
	PHDW5R	0x7054	R/W	Print head data word 5 register	0x0000
	PHDW6R	0x7058	R/W	Print head data word 6 register	0x0000
	PHDW7R	0x705c	R/W	Print head data word 7 register	0x0000
	PHDW8R	0x7060	R/W	Print head data word 8 register	0x0000
	PHDW9R	0x7064	R/W	Print head data word 9 register	0x0000
	PHDW10R	0x7068	R/W	Print head data word 10 register	0x0000
	PHDW11R	0x706c	R/W	Print head data word 11 register	0x0000
	PHDW12R	0x7070	R/W	Print head data word 12 register	0x0000
	DCBR	0x7074	R/W	Dot counter black register	0x00000000
	DCYR	0x7078	R/W	Dot counter yellow register	0x00000000
	DCCR	0x707c	R/W	Dot counter cyan register	0x00000000
	DCMR	0x7080	R/W	Dot counter magenta register	0x00000000
	DCCOR	0x7084	R	Dot counter control observation register	0x000
PWM	PWMCONR	0xe000	R/W	PWM_CON control register	0x0
	PWM_PRSC	0xe004	R/W	PWM Pre-Scaler counter base value register	0x00000000
	PWM_CYT0	0xe008	R/W	PWM0 cycle time & observation register	0x00000000
	PWM_ONT0	0xe00c	R/W	PWM0 on time & observation register	0x00000000
	PWM_CYT1	0xe010	R/W	PWM1 cycle time & observation register	0x00000000
	PWM_ONT1	0xe014	R/W	PWM1 on time & observation register	0x00000000
	PWM_CYT2	0xe018	R/W	PWM2 cycle time & observation register	0x00000000
	PWM_ONT2	0xe01c	R/W	PWM2 on time & observation register	0x00000000

Table 1-2. KS32C65100 Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
VIS	VISSR	0xa800	R	VIS Status register	0x0
	VISCON	0xa804	R/W	VIS control register	0x0
	DstSize	0xa808	R/W	Destination image data size register	0xFFFF
	SrcSize	0xa80c	R/W	Source image data size register	0xFFFF
	SrcReg	0xa810	R/W	Source image data register	0xFF
	DstReg	0xa814	R	Destination image data register	0xFFFF
	RefIn	0xa818	R/W	Reference data register	0xFFFF
	PixIn	0xa81c	R/W	Source image pixel data register	0xFFFF
	HftReg	0xa820	R	Halftone image data register	0xFFFF

2

PROGRAMMER'S MODEL

OVERVIEW

KS32C65100 was developed using the advanced ARM7TDMI core designed by advanced RISC machines, Ltd.

PROCESSOR OPERATING STATES

From the programmer's point of view, the ARM7TDMI can be in one of two states:

- ARM state which executes 32-bit, word-aligned ARM instructions.
- THUMB state which operates with 16-bit, halfword-aligned THUMB instructions. In this state, the PC uses bit 1 to select between alternate halfword.

NOTE

Transition between these two states does not affect the processor mode or the contents of the registers.

SWITCHING STATE

Entering THUMB State

Entry into THUMB state can be achieved by executing a BX instruction with the state bit (bit 0) set in the operand register.

Transition to THUMB state will also occur automatically on return from an exception (IRQ, FIQ, UNDEF, ABORT, SWI etc.), if the exception was entered with the processor in THUMB state.

Entering ARM State

Entry into ARM state happens:

- On execution of the BX instruction with the state bit clear in the operand register.
- On the processor taking an exception (IRQ, FIQ, RESET, UNDEF, ABORT, SWI etc.). In this case, the PC is placed in the exception mode's link register, and execution commences at the exception's vector address.

MEMORY FORMATS

ARM7TDMI views memory as a linear collection of bytes numbered upwards from zero. Bytes 0 to 3 hold the first stored word, bytes 4 to 7 the second and so on. ARM7TDMI can treat words in memory as being stored either in Big-Endian or Little-Endian format.

NOTE

The KS32C65100 is configured to the big-endian format.

BIG-ENDIAN FORMAT

In Big-Endian format, the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte. Byte 0 of the memory system is therefore connected to data lines 31 through 24.

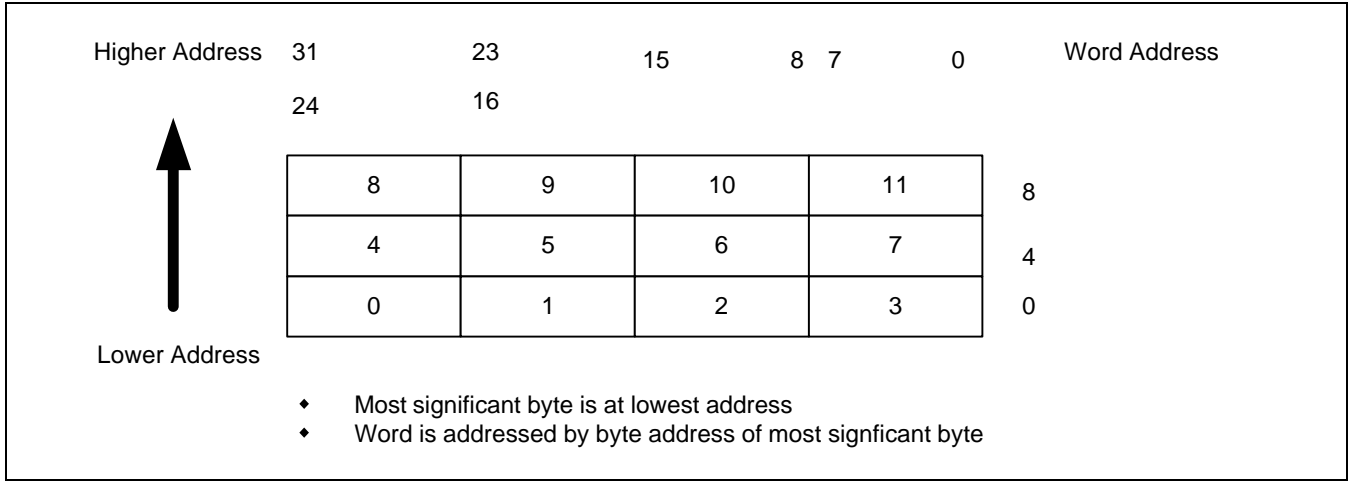


Figure 2-1. Big-Endian Addresses of Bytes within Words

The data locations in the external memory are different with Figure 2-1 in the KS32C6200. Please refer to the chapter 4, system manager.

LITTLE-ENDIAN FORMAT

In Little-Endian format, the lowest numbered byte in a word is considered the word's least significant byte, and the highest numbered byte the most significant. Byte 0 of the memory system is therefore connected to data lines 7 through 0.

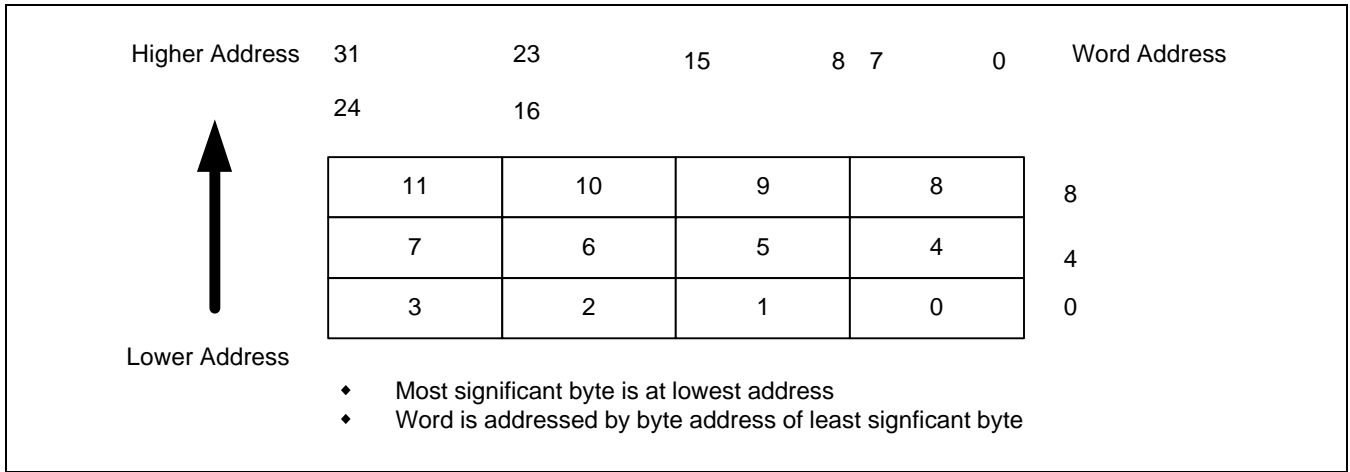


Figure 2-2. Little-Endian Addresses of Bytes within Words

INSTRUCTION LENGTH

Instructions are either 32 bits long (in ARM state) or 16 bits long (in THUMB state).

Data Types

ARM7TDMI supports byte (8-bit), half-word (16-bit) and word (32-bit) data types. Words must be aligned to four-byte boundaries and half words to two-byte boundaries.

OPERATING MODES

ARM7TDMI supports seven modes of operation:

- User (usr): The normal ARM program execution state
- FIQ (fiq): Designed to support a data transfer or channel process
- IRQ (irq): Used for general-purpose interrupt handling
- Supervisor (svc): Protected mode for the operating system
- Abort mode (abt): Entered after a data or instruction prefetch abort
- System (sys): A privileged user mode for the operating system
- Undefined (und): Entered when an undefined instruction is executed

Mode changes may be made under software control, or may be brought about by external interrupts or exception processing. Most application programs will execute in user mode. The non-user modes-known as privileged modes-are entered in order to service interrupts or exceptions, or to access protected resources.

REGISTERS

ARM7TDMI has a total of 37 registers-31 general-purpose 32-bit registers and six status registers - but these cannot all be seen at once. The processor state and operating mode dictate which registers are available to the programmer.

The ARM State Register Set

In ARM state, 16 general registers and one or two status registers are visible at any one time. In privileged (non-user) modes, mode-specific banked registers are switched in. Figure 2-3 shows which registers are available in each mode: the banked registers are marked with a shaded triangle.

The ARM state register set contains 16 directly accessible registers: R0 to R15. All of these except R15 are general-purpose, and may be used to hold either data or address values. In addition to these, there is a seventeenth register used to store status information

- Register 14 is used as the subroutine link register. This receives a copy of R15 when a branch and link (BL) instruction is executed. At all other times it may be treated as a general-purpose register. The corresponding banked registers R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are similarly used to hold the return values of R15 when interrupts and exceptions arise, or when branch and link instructions are executed within interrupt or exception routines.
- Register 15 [31:2] holds the Program Counter (PC). In ARM state, bits [1:0] of R15 are zero and bits contain the PC. In THUMB state, bit [0] is zero and bits [31:1] contain the PC.
- Register 16 and is the CPSR (Current Program Status Register). This contains condition code flags the current mode bits. FIQ mode has seven banked registers mapped to R8-14 (R8_fiq-R14_fiq). In ARM state, many FIQ handlers do not need to save any registers. User, IRQ, Supervisor, abort and undefined each have two banked registers mapped to R13 and R14, allowing each of these modes to have a private stack pointer and link registers.

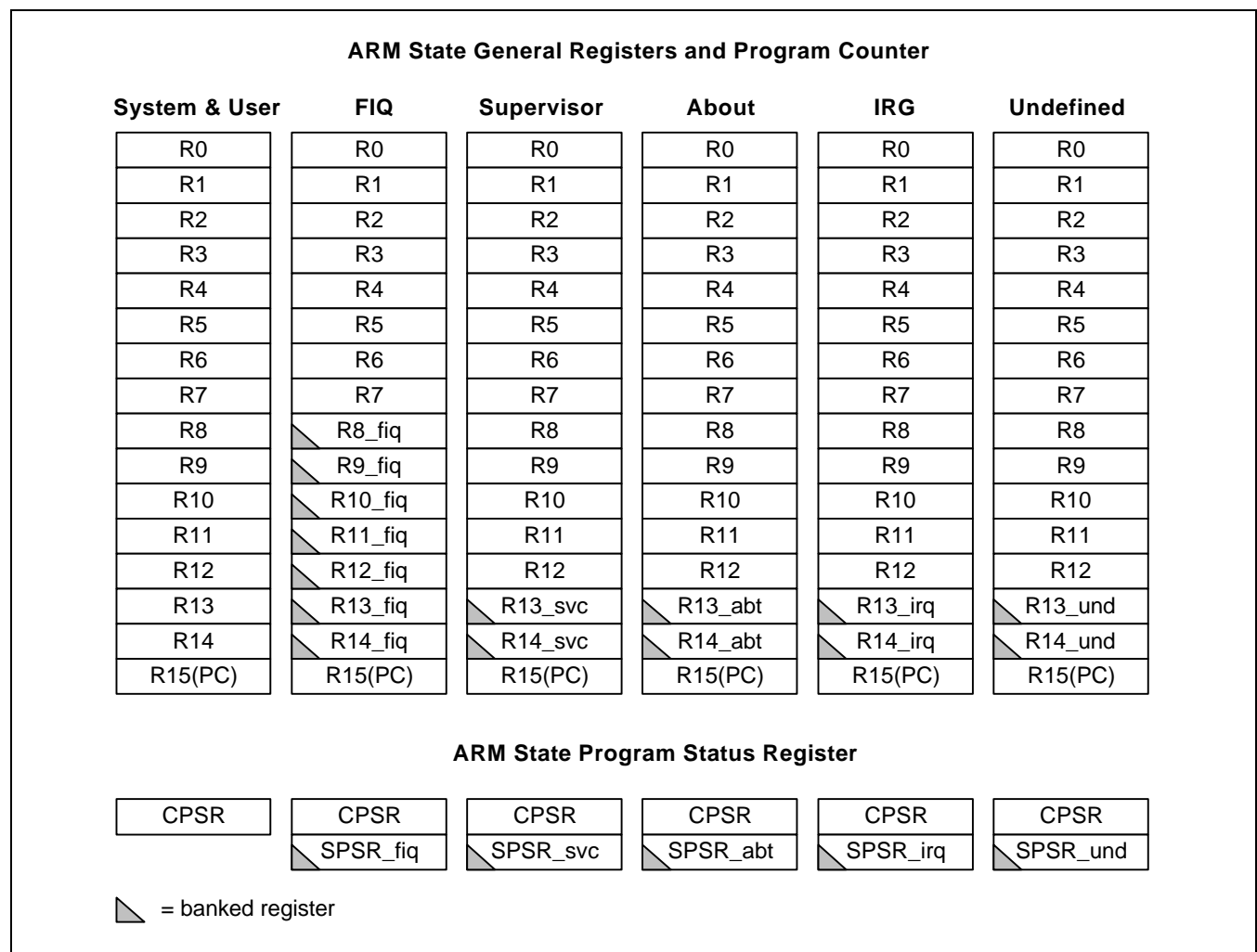


Figure 2-3. Register Organization in ARM State

The THUMB State Register Set

The THUMB state register set is a subset of the ARM state set. The programmer has direct access to eight general registers, R0-R7, as well as the Program Counter (PC), a stack pointer register (SP), a link register (LR), and the CPSR. There are banked Stack Pointers, Link Registers and Saved Process Status Registers (SPSRs) for each privileged mode. This is shown in Figure 2-4.

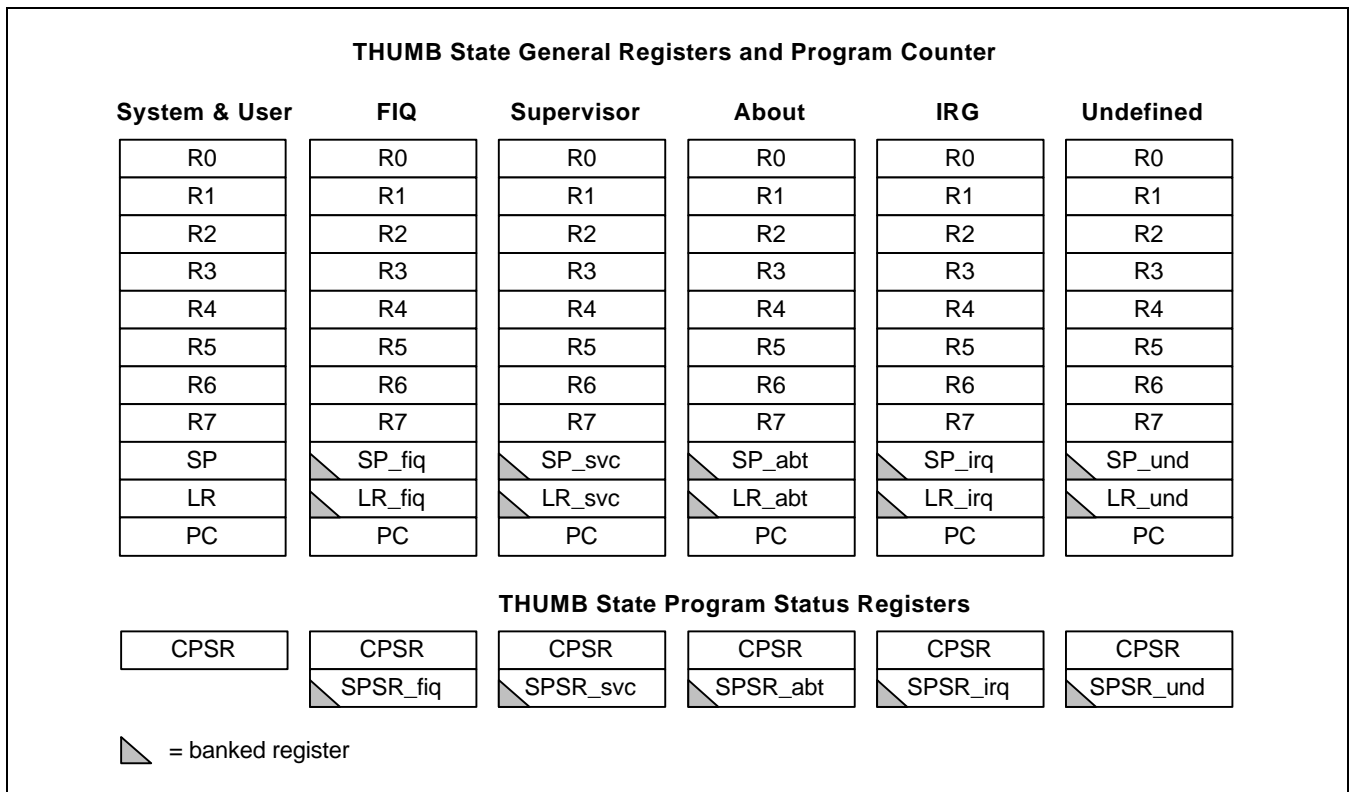


Figure 2-4. Register Organization in THUMB State

The relationship between ARM and THUMB state registers

The THUMB state registers relate to the ARM state registers in the following way:

- THUMB state R0-R7 and ARM state R0-R7 are identical
- THUMB state CPSR and SPSRs and ARM state CPSR and SPSRs are identical
- THUMB state SP maps onto ARM state R13
- THUMB state LR maps onto ARM state R14
- The THUMB state program counter maps onto the ARM state program counter (R15)

This relationship is shown in Figure 2-5.

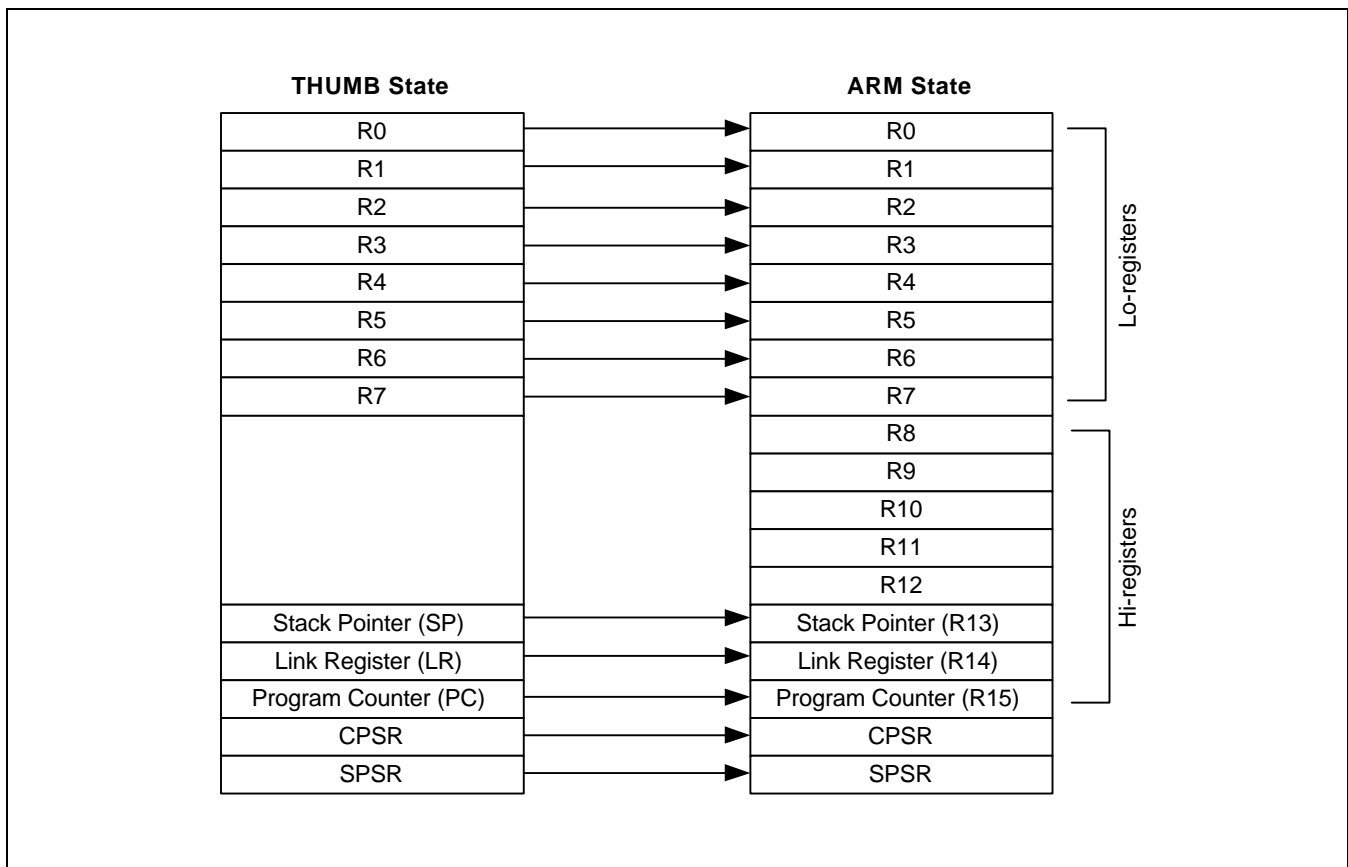


Figure 2-5. Mapping of THUMB State Registers onto ARM State Registers

Accessing Hi-Registers in THUMB State

In THUMB state, registers R8-R15 (the Hi registers) are not part of the standard register set. However, the assembly language programmer has limited access to them, and can use them for fast temporary storage.

A value may be transferred from a register in the range R0-R7 (a Lo register) to a Hi register, and from a Hi register to a Lo register, using special variants of the MOV instruction. Hi register values can also be compared against or added to Lo register values with the CMP and ADD instructions. For more information, refer to Figure 3-34.

THE PROGRAM STATUS REGISTERS

The ARM7TDMI contains a Current Program Status Register (CPSR), plus five Saved Program Status Registers (SPSRs) for use by exception handlers. These register's functions are:

- Hold information about the most recently performed ALU operation
- Control the enabling and disabling of interrupts
- Set the processor operating mode

The arrangement of bits is shown in Figure 2-6.

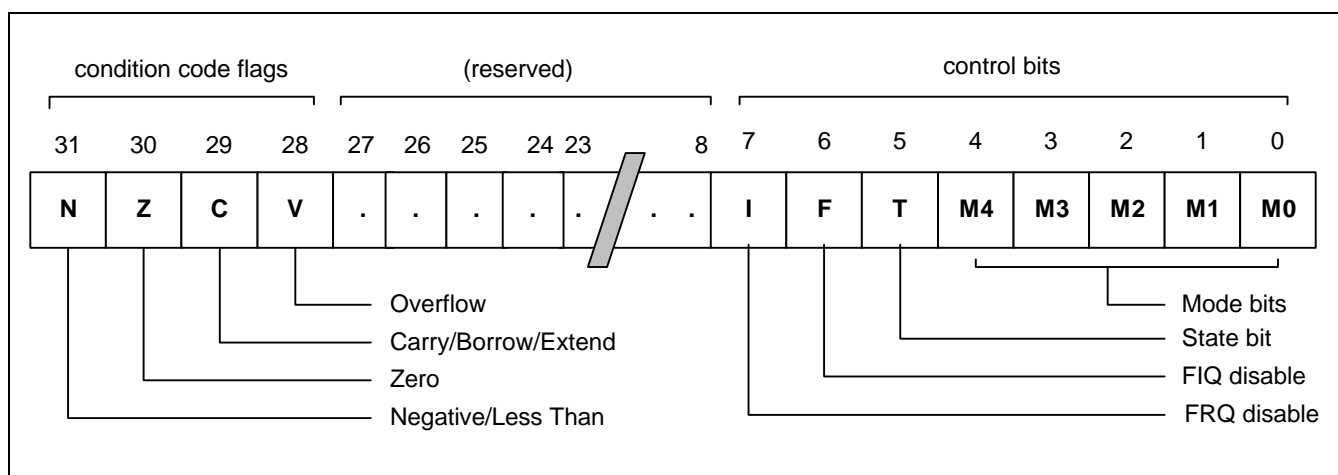


Figure 2-6. Program Status Register Format

The Condition Code Flags

The N, Z, C and V bits are the condition code flags. These may be changed as a result of arithmetic and logical operations, and may be tested to determine whether an instruction should be executed.

In ARM state, all instructions may be executed conditionally: see Table 3-2 for details.

In THUMB state, only the Branch instruction is capable of conditional execution: see Figure 3-46 for details.

The Control Bits

The bottom 8 bits of a PSR (incorporating I, F, T and M[4:0]) are known collectively as the control bits. These will change when an exception arises. If the processor is operating in a privileged mode, they can also be manipulated by software.

The T bit	<p>This reflects the operating state. When this bit is set, the processor is executing in THUMB state, otherwise it is executing in ARM state. This is reflected on the TBIT external signal.</p> <p>Note that the software must never change the state of the TBIT in the CPSR. If this happens, the processor will enter an unpredictable state.</p> <p>Interrupt disable bits The I and F bits are the interrupt disable bits. When set, these disable the IRQ and FIQ interrupts respectively.</p>
The mode bits	<p>The M4, M3, M2, M1 and M0 bits (M[4:0]) are the mode bits. These determine the processor's operating mode, as shown in Table 2-1. Not all combinations of the mode bits define a valid processor mode. Only those explicitly described shall be used. The user should be aware that if any illegal value is programmed into the mode bits, then the processor will enter an unrecoverable state. If this occurs, reset should be applied.</p>
M[4:0],	
Reserved bits	<p>The remaining bits in the PSRs are reserved. When changing a PSR's flag or control bits, you must ensure that these unused bits are not altered. Also, your program should not rely on them containing specific values, since in future processors they may read as one or zero.</p>

Table 2-1. PSR Mode bit Values

M[4:0]	Mode	Visible THUMB State Registers	Visible ARM State Registers
10000	User	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR
10001	FIQ	R7..R0, LR_fiq, SP_fiq PC, CPSR, SPSR_fiq	R7..R0, R14_fiq..R8_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	R7..R0, LR_irq, SP_irq PC, CPSR, SPSR_irq	R12..R0, R14_irq..R13_irq, PC, CPSR, SPSR_irq
10011	Supervisor	R7..R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc	R12..R0, R14_svc..R13_svc, PC, CPSR, SPSR_svc
10111	Abort	R7..R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt	R12..R0, R14_abt..R13_abt, PC, CPSR, SPSR_abt
11011	Undefined	R7..R0 LR_und, SP_und, PC, CPSR, SPSR_und	R12..R0, R14_und..R13_und, PC, CPSR
11111	System	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR

EXCEPTIONS

Exceptions arise whenever the normal flow of a program has to be halted temporarily, for example to service an interrupt from a peripheral. Before an exception can be handled, the current processor state must be preserved so that the original program can resume when the handler routine has finished.

It is possible for several exceptions to arise at the same time. If this happens, they are dealt with in a fixed order. see exception priorities on page 2-14.

Action on Entering an Exception

When handling an exception, the ARM7TDMI:

1. Preserves the address of the next instruction in the appropriate link register. If the exception has been entered from ARM state, then the address of the next instruction is copied into the link register (that is, current PC + 4 or PC + 8 depending on the exception. See Table 2-2 on for details). If the exception has been entered from THUMB state, then the value written into the link register is the current PC offset by a value such that the program resumes from the correct place on return from the exception. This means that the exception handler need not determine which state the exception was entered from. For example, in the case of SWI, MOVS PC, R14_svc will always return to the next instruction regardless of whether the SWI was executed in ARM or THUMB state.
2. Copies the CPSR into the appropriate SPSR
3. Forces the CPSR mode bits to a value which depends on the exception
4. Forces the PC to fetch the next instruction from the relevant exception vector It may also set the interrupt disable flags to prevent otherwise unmanageable nestings of exceptions. If the processor is in THUMB state when an exception occurs, it will automatically switch into ARM state when the PC is loaded with the exception vector address.

Action on Leaving an Exception

On completion, the exception handler:

1. Moves the link register, minus an offset where appropriate, to the PC. (The offset will vary depending on the type of exception.)
2. Copies the SPSR back to the CPSR
3. Clears the interrupt disable flags, if they were set on entry

NOTE

An explicit switch back to THUMB state is never needed, since restoring the CPSR from the SPSR automatically sets the T bit to the value it held immediately prior to the exception.

Exception Entry/Exit Summary

Table 2-2 summarises the PC value preserved in the relevant R14 on exception entry, and the recommended instruction for exiting the exception handler.

Table 2-2. Exception Entry/Exit

	Return Instruction	Previous State		Notes
		ARM R14_x	THUMB R14_x	
BL	MOV PC, R14	PC + 4	PC + 2	1
SWI	MOVS PC, R14_svc	PC + 4	PC + 2	1
UDEF	MOVS PC, R14_und	PC + 4	PC + 2	1
FIQ	SUBS PC, R14_fiq, #4	PC + 4	PC + 2	2
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 2	2
PABT	SUBS PC, R14_abt, #4	PC + 4	PC + 2	1
DABT	SUBS PC, R14_abt, #8	PC + 8	PC + 2	3
RESET	NA	-	-	4

NOTES:

1. Where PC is the address of the BL/SWI/Undefined Instruction fetch which had the prefetch abort.
2. Where PC is the address of the instruction which did not get executed since the FIQ or IRQ took priority.
3. Where PC is the address of the Load or Store instruction which generated the data abort.
4. The value saved in R14_svc upon reset is unpredictable.

FIQ

The FIQ (Fast Interrupt Request) exception is designed to support a data transfer or channel process, and in ARM state has sufficient private registers to remove the need for register saving (thus minimising the overhead of context switching).

FIQ is externally generated by taking the **nFIQ** input LOW. This input can except either synchronous or asynchronous transitions, depending on the state of the **ISYNC** input signal. When **ISYNC** is LOW, **nFIQ** and **nIRQ** are considered asynchronous, and a cycle delay for synchronization is incurred before the interrupt can affect the processor flow.

Irrespective of whether the exception was entered from ARM or Thumb state, a FIQ handler should leave the interrupt by executing

SUBS PC,R14_fiq,#4

FIQ may be disabled by setting the CPSR's F flag (but note that this is not possible from user mode). If the F flag is clear, ARM7TDMI checks for a LOW level on the output of the FIQ synchroniser at the end of each instruction.

IRQ

The IRQ (Interrupt Request) exception is a normal interrupt caused by a low level on the nIRQ input. IRQ has a lower priority than FIQ and is masked out when a FIQ sequence is entered. It may be disabled at any time by setting the I bit in the CPSR, though this can only be done from a privileged (non-User) mode.

Irrespective of whether the exception was entered from ARM or Thumb state, an IRQ handler should return from the interrupt by executing

```
SUBS PC,R14_irq,#4
```

Abort

An abort indicates that the current memory access cannot be completed. It can be signalled by the external abort input. ARM7TDMI checks for the abort exception during memory access cycles.

There are two types of abort:

- Prefetch abort: occurs during an instruction prefetch.
- Data abort: occurs during a data access.

If a prefetch abort occurs, the prefetched instruction is marked as invalid, but the exception will not be taken until the instruction reaches the head of the pipeline. If the instruction is not executed - for example because a branch occurs while it is in the pipeline - the abort does not take place.

If a data abort occurs, the action taken depends on the instruction type:

- Single data transfer instructions (LDR, STR) write back modified base registers: the Abort handler must be aware of this.
- The swap instruction (SWP) is aborted as though it had not been executed.
- Block data transfer instructions (LDM, STM) complete. If write-back is set, the base is updated. If the instruction would have overwritten the base with data (i.e. it has the base in the transfer list), the overwriting is prevented. All register overwriting is prevented after an abort is indicated, which means in particular that R15 (always the last register to be transferred) is preserved in an aborted LDM instruction.

The abort mechanism allows the implementation of a demand paged virtual memory system. In such a system the processor is allowed to generate arbitrary addresses. When the data at an address is unavailable, the Memory Management Unit (MMU) signals an abort. The abort handler must then work out the cause of the abort, make the requested data available, and retry the aborted instruction. The application program needs no knowledge of the amount of memory available to it, nor is its state in any way affected by the abort.

After fixing the reason for the abort, the handler should execute the following irrespective of the state (ARM or Thumb):

```
SUBS PC,R14_abt,#4 ;      for a prefetch abort, or  
SUBS PC,R14_abt,#8 ;      for a data abort
```

This restores both the PC and the CPSR, and retries the aborted instruction.

Software Interrupt

The software interrupt instruction (SWI) is used for entering supervisor mode, usually to request a particular supervisor function. A SWI handler should return by executing the following irrespective of the state (ARM or Thumb):

```
MOV PC,R14_svc
```

This restores the PC and CPSR, and returns to the instruction following the SWI.

NOTE

nFIQ, nIRQ, ISYNC, LOCK, BIGEND, and ABORT pins exist only in the ARM7TDMI CPU core.

Undefined Instruction

When ARM7TDMI comes across an instruction which it cannot handle, it takes the undefined instruction trap. This mechanism may be used to extend either the THUMB or ARM instruction set by software emulation.

After emulating the failed instruction, the trap handler should execute the following irrespective of the state (ARM or Thumb):

```
MOVS PC, R14_und
```

This restores the CPSR and returns to the instruction following the undefined instruction.

Exception Vectors

The following table shows the exception vector addresses.

Table 2-3. Exception Vectors

Address	Exception	Mode on Entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Exception Priorities

When multiple exceptions arise at the same time, a fixed priority system determines the order in which they are handled:

Highest priority:

1. Reset
2. Data abort
3. FIQ
4. IRQ
5. Prefetch abort

Lowest priority:

6. Undefined instruction, software interrupt.

Not All Exceptions Can Occur at Once:

Undefined Instruction and software Interrupt are mutually exclusive, since they each correspond to particular (non-overlapping) decodings of the current instruction.

If a data abort occurs at the same time as a FIQ, and FIQs are enabled (i.e. the CPSR's F flag is clear), ARM7TDMI enters the data abort handler and then immediately proceeds to the FIQ vector. A normal return from FIQ will cause the data abort handler to resume execution. Placing data abort at a higher priority than FIQ is necessary to ensure that the transfer error does not escape detection. The time for this exception entry should be added to worst-case FIQ latency calculations.

INTERRUPT LATENCIES

The worst case latency for FIQ, assuming that it is enabled, consists of the longest time the request can take to pass through the synchroniser ($T_{syncmax}$ if asynchronous), plus the time for the longest instruction to complete (T_{ldm} , the longest instruction is an LDM which loads all the registers including the PC), plus the time for the data abort entry (T_{exc}), plus the time for FIQ entry (T_{fiq}). At the end of this time ARM7TDMI will be executing the instruction at 0x1C.

$T_{syncmax}$ is 3 processor cycles, T_{ldm} is 20 cycles, T_{exc} is 3 cycles, and T_{fiq} is 2 cycles. The total time is therefore 28 processor cycles. This is just over 1.4 microseconds in a system which uses a continuous 20MHz processor clock. The maximum IRQ latency calculation is similar, but must allow for the fact that FIQ has higher priority and could delay entry into the IRQ handling routine for an arbitrary length of time. The minimum latency for FIQ or IRQ consists of the shortest time the request can take through the synchroniser ($T_{syncmin}$) plus T_{fiq} . This is 4 processor cycles.

RESET

When the nRESET signal goes LOW, ARM7TDMI abandons the executing instruction and then continues to fetch instructions from incrementing word addresses.

When nRESET goes HIGH again, ARM7TDMI:

1. Overwrites R14_svc and SPSR_svc by copying the current values of the PC and CPSR into them. The value of the saved PC and SPSR is not defined.
2. Forces M[4:0] to 10011 (supervisor mode), sets the I and F bits in the CPSR, and clears the CPSR's T bit.
3. Forces the PC to fetch the next instruction from address 0x00.
4. Execution resumes in ARM state.

3 INSTRUCTION SET

INSTRUCTION SET SUMMARY

This chapter describes the ARM instruction set and the THUMB instruction set in the ARM7TDMI core.

FORMAT SUMMARY

The ARM instruction set formats are shown below.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
Cond	0	0	1	Opcode				S	Rn				Rd				Operand2												Data processing/ PSR Transfer					
Cond	0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm				Multiply					
Cond	0	0	0	0	1	U	A	S	RdHi				RnLo				Rn				1	0	0	1	Rm				Multiply Long					
Cond	0	0	0	1	0	B	0	0	Rn				Rd				0	0	0	0	1	0	0	1	Rm				Single data swap					
Cond	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn				Branch and exchange					
Cond	0	0	0	P	U	0	W	L	Rn				Rd				0	0	0	0	1	S	H	1	Rm				Halfword data transfer: register offset					
Cond	0	0	0	P	U	1	W	L	Rn				Rd				Offset				1	S	H	1	Offset				Halfword data transfer: immediate offset					
Cond	0	1	1	P	U	B	W	L	Rn				Rd				Offset												Single data transfer					
Cond	0	1	1																											1				Undefined
Cond	1	0	0	P	U	S	W	L	Rn				Register List																Block data transfer					
Cond	1	0	1	L	Offset																										Branch			
Cond	1	1	0	P	U	N	W	L	Rn				CRd				CP#				Offset								Coprocessor data transfer					
Cond	1	1	1	0	CP Opc				CRn				CRd				CP#				CP#				0	CRm				Coprocessor data Operation				
Cond	1	1	1	0	CP Opc				L	CRn				Rd				CP#				CP#				1	CRm				Coprocessor data Transfer			
Cond	1	1	1	1	Ignored by processor																										Software Interrupt			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			

Figure 3-1. ARM Instruction Set Format

Some instruction codes are not defined but do not cause the undefined instruction trap to be taken, for instance a multiply instruction with bit 6 changed to a 1. These instructions should not be used, as their action may change in future ARM implementations.

INSTRUCTION SUMMARY

Table 3-1. The ARM Instruction Set

Mnemonic	Instruction	Action
ADC	Add with carry	$Rd = Rn + Op2 + \text{Carry}$
ADD	Add	$Rd = Rn + Op2$
AND	AND	$Rd = Rn \text{ and } Op2$
B	Branch	$R15 = \text{address}$
BIC	Bit clear	$Rd = Rn \text{ and not } Op2$
BL	Branch with link	$R14 = R15, R15 = \text{address}$
BX	Branch and exchange	$R15 = Rn, T \text{ bit} = Rn[0]$
CDP	Coprocessor data processing	(Coprocessor-specific)
CMN	Compare negative	$CPSR \text{ flags} = Rn + Op2$
CMP	Compare	$CPSR \text{ flags} = Rn - Op2$
EOR	Exclusive OR	$Rd = (Rn \text{ and not } Op2) \text{ or } (Op2 \text{ and not } Rn)$
LDC	Load coprocessor from memory	Coprocessor load
LDM	Load multiple registers	Stack manipulation (Pop)
LDR	Load register from memory	$Rd = (\text{address})$
MCR	Move CPU register to coprocessor register	$cRn = rRn \{<op>cRm\}$
MLA	Multiply accumulate	$Rd = (Rm * Rs) + Rn$
MOV	Move register or constant	$Rd = Op2$
MRC	Move from coprocessor register to CPU register	$Rn = cRn \{<op>cRm\}$
MRS	Move PSR status/flags to register	$Rn = PSR$
MSR	Move register to PSR status/flags	$PSR = Rm$
MUL	Multiply	$Rd = Rm * Rs$
MVN	Move negative register	$Rd = 0xFFFFFFFF \text{ EOR } Op2$
ORR	OR	$Rd = Rn \text{ or } Op2$
RSB	Reverse subtract	$Rd = Op2 - Rn$
RSC	Reverse subtract with carry	$Rd = Op2 - Rn - 1 + \text{Carry}$
SBC	Subtract with carry	$Rd = Rn - Op2 - 1 + \text{Carry}$
STC	Store coprocessor register to memory	$\text{address} = CRn$
STM	Store Multiple	Stack manipulation (push)
STR	Store register to memory	$<\text{address}> = Rd$
SUB	Subtract	$Rd = Rn - Op2$
SWI	Software Interrupt	OS call
SWP	Swap register with memory	$Rd = [Rn], [Rn] := Rm$
TEQ	Test bitwise equality	$CPSR \text{ flags} = Rn \text{ EOR } Op2$
TST	Test bits	$CPSR \text{ flags} = Rn \text{ AND } Op2$

THE CONDITION FIELD

In ARM state, all instructions are conditionally executed according to the state of the CPSR condition codes and the instruction's condition field. This field (bits 31:28) determines the circumstances under which an instruction is to be executed. If the state of the C, N, Z and V flags fulfils the conditions encoded by the field, the instruction is executed, otherwise it is ignored.

There are sixteen possible conditions, each represented by a two-character suffix that can be appended to the instruction's mnemonic. For example, a branch (B in assembly language) becomes BEQ for "Branch if Equal", which means the branch will only be taken if the Z flag is set.

In practice, fifteen different conditions may be used: these are listed in Table 3-2. The sixteenth (1111) is reserved, and must not be used.

In the absence of a suffix, the condition field of most instructions is set to "Always" (suffix AL). This means the instruction will always be executed regardless of the CPSR condition codes.

Table 3-2. Condition Code Summary

Code	Suffix	Flags	Meaning
0000	EQ	Z set	Equal
0001	NE	Z clear	Not equal
0010	CS	C set	Unsigned higher or same
0011	CC	C clear	Unsigned lower
0100	MI	N set	Negative
0101	PL	N clear	Positive or zero
0110	VS	V set	Overflow
0111	VC	V clear	No overflow
1000	HI	C set and Z clear	Unsigned higher
1001	LS	C clear or Z set	Unsigned lower or same
1010	GE	N equals V	Greater or equal
1011	LT	N not equal to V	Less than
1100	GT	Z clear AND (N equals V)	Greater than
1101	LE	Z set OR (N not equal to V)	Less than or equal
1110	AL	(ignored)	Always

BRANCH AND EXCHANGE (BX)

This instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.

This instruction performs a branch by copying the contents of a general register, Rn, into the program counter, PC. The branch causes a pipeline flush and refill from the address specified by Rn. This instruction also permits the instruction set to be exchanged. When the instruction is executed, the value of Rn[0] determines whether the instruction stream will be decoded as ARM or THUMB instructions.

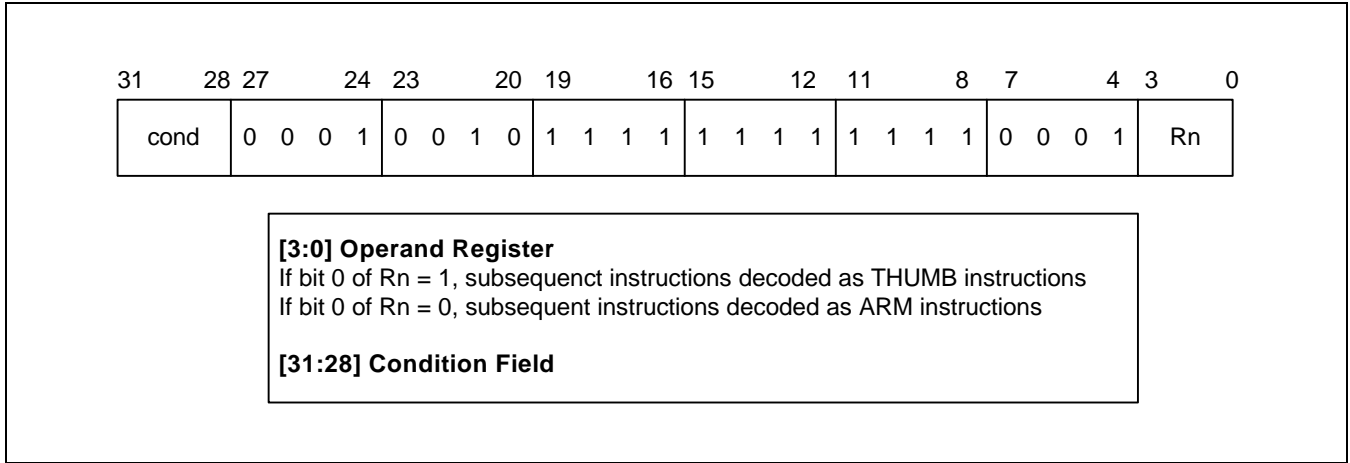


Figure 3-2. Branch and Exchange Instructions

INSTRUCTION CYCLE TIMES

The BX instruction takes 2S + 1N cycles to execute, where S and N are defined as sequential (S-cycle) and non-sequential (N-cycle), respectively.

ASSEMBLER SYNTAX

BX - branch and exchange. Items in {} are optional. Items in <> must be present.

BX {cond} Rn

{cond} Two character condition mnemonic. See Table 3-2.
Rn is an expression evaluating to a valid register number.

USING R15 AS AN OPERAND

If R15 is used as an operand, the behaviour is undefined.

Examples

```

ADR      R0, Into_THUMB + 1      ; Generate branch target address
                                   ; and set bit 0 high - hence
                                   ; arrive in THUMB state.
BX       R0                      ; Branch and change to THUMB
                                   ; state.
CODE16                                       ; Assemble subsequent code as
Into_THUMB                                ; THUMB instructions
.
.
.
ADR R5, Back_to_ARM                  ; Generate branch target to word aligned address
                                   ; - hence bit 0 is low and so change back to ARM state.
BX R5                                ; Branch and change back to ARM state.
.
.
.
ALIGN                                     ; Word align
CODE32                                  ; Assemble subsequent code as ARM instructions
Back_to_ARM

```


BRANCH AND BRANCH WITH LINK (B, BL)

The instruction is only executed if the condition is true. The various conditions are defined Table 3-2. The instruction encoding is shown in Figure 3-3, below

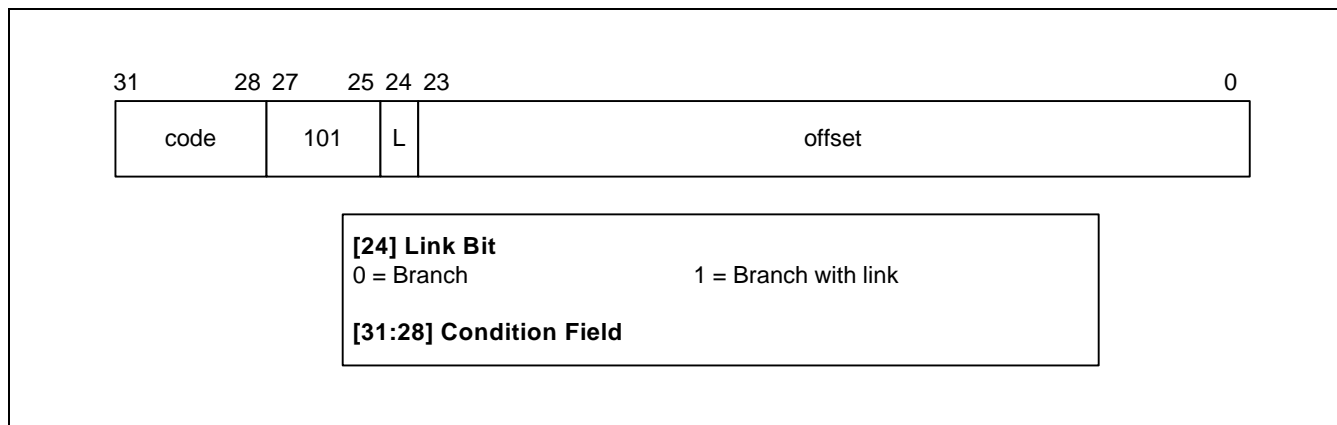


Figure 3-3. Branch Instructions

Branch instructions contain a signed 2's complement 24 bit offset. This is shifted left two bits, sign extended to 32 bits, and added to the PC. The instruction can therefore specify a branch of +/- 32Mbytes. The branch offset must take account of the prefetch operation, which causes the PC to be 2 words (8 bytes) ahead of the current instruction.

Branches beyond +/- 32Mbytes must use an offset or absolute destination which has been previously loaded into a register. In this case the PC should be manually saved in R14 if a branch with link type operation is required.

THE LINK BIT

Branch with Link (BL) writes the old PC into the link register (R14) of the current bank. The PC value written into R14 is adjusted to allow for the prefetch, and contains the address of the instruction following the branch and link instruction. Note that the CPSR is not saved with the PC and R14[1:0] are always cleared.

To return from a routine called by Branch with Link use MOV PC,R14 if the link register is still valid or LDM Rn!,{..PC} if the link register has been saved onto a stack pointed to by Rn.

INSTRUCTION CYCLE TIMES

Branch and branch with Link instructions take $2S + 1N$ incremental cycles, where S and N are defined as sequential (S-cycle) and internal (I-cycle).

ASSEMBLER SYNTAX

Items in {} are optional. Items in <> must be present.

B{L}{cond} <expression>

{L} Used to request the branch with link form of the instruction. If absent, R14 will not be affected by the instruction.

{cond} A two-character mnemonic as shown in Table 3-2. If absent then AL (ALways) will be used.

<expression> The destination. The assembler calculates the offset.

Examples

here	BAL	here	; Assembles to 0xEAFFFFFEE (note effect of PC offset).
	B	there	; Always condition used as default.
	CMP	R1,#0	; Compare R1 with zero and branch to fred
			; if R1 was zero, otherwise continue.
	BEQ	fred	; Continue to next instruction.
	BL	sub+ROM	; Call subroutine at computed address.
	ADDS	R1,#1	; Add 1 to register 1, setting CPSR flags
			; on the result then call subroutine if
	BLCC	sub	; the C flag is clear, which will be the
			; case unless R1 held 0xFFFFFFFF.

DATA PROCESSING

The data processing instruction is only executed if the condition is true. The conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-4.

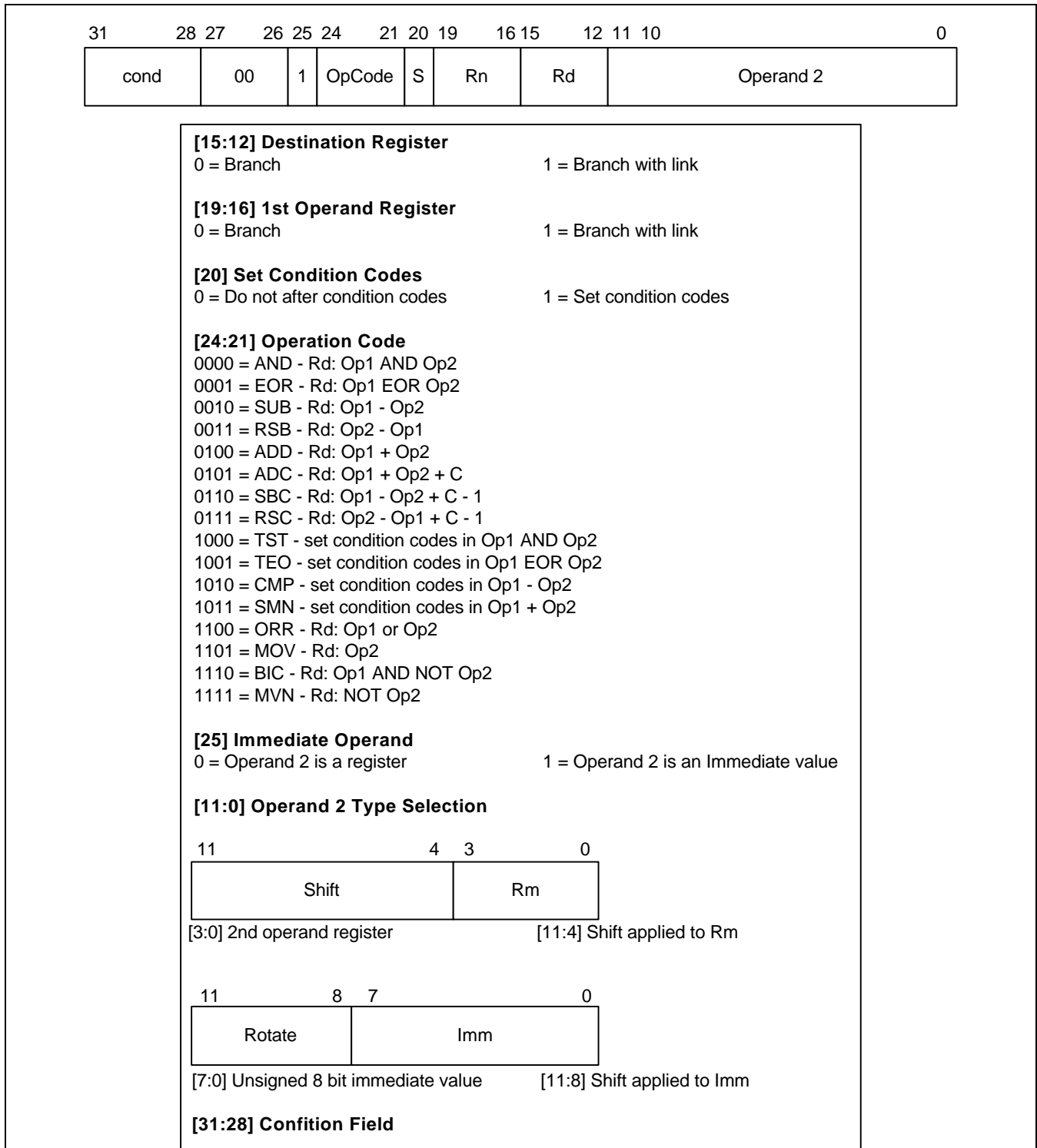


Figure 3-4. Data Processing Instructions

The instruction produces a result by performing a specified arithmetic or logical operation on one or two operands. The first operand is always a register (Rn).

The second operand may be a shifted register (Rm) or a rotated 8 bit immediate value (Imm) according to the value of the I bit in the instruction. The condition codes in the CPSR may be preserved or updated as a result of this instruction, according to the value of the S bit in the instruction.

Certain operations (TST, TEQ, CMP, CMN) do not write the result to Rd. They are used only to perform tests and to set the condition codes on the result and always have the S bit set. The instructions and their effects are listed in Table 3-3.

CPSR FLAGS

The data processing operations may be classified as logical or arithmetic. The logical operations (AND, EOR, TST, TEQ, ORR, MOV, BIC, MVN) perform the logical action on all corresponding bits of the operand or operands to produce the result. If the S bit is set (and Rd is not R15, see below) the V flag in the CPSR will be unaffected, the C flag will be set to the carry out from the barrel shifter (or preserved when the shift operation is LSL #0), the Z flag will be set if and only if the result is all zeros, and the N flag will be set to the logical value of bit 31 of the result.

Table 3-3. ARM Data Processing Instructions

Assembler Mnemonic	Op-Code	Action
AND	0000	Operand1 AND operand 2
EOR	0001	Operand1 EOR operand2
SUB	0010	Operand1 - operand2
RSB	0011	Operand2 operand1
ADD	0100	Operand1 + operand2
ADC	0101	Operand1 + operand2 + carry
SBC	0110	Operand1 - operand2 + carry - 1
RSC	0111	Operand2 - operand1 + carry - 1
TST	1000	As AND, but result is NOT written
TEQ	1001	As EOR, but result is NOT written
CMP	1010	As SUB, but result is NOT written
CMN	1011	As ADD, but result is NOT written
ORR	1100	Operand1 OR operand2
MOV	1101	Operand2 (operand1 is ignored)
BIC	1110	Operand1 AND NOT operand2 (Bit clear)
MVN	1111	NOT operand2 (operand1 is ignored)

The arithmetic operations (SUB, RSB, ADD, ADC, SBC, RSC, CMP, CMN) treat each operand as a 32 bit integer (either unsigned or 2's complement signed, the two are equivalent). If the S bit is set (and Rd is not R15) the V flag in the CPSR will be set if an overflow occurs into bit 31 of the result; this may be ignored if the operands were considered unsigned, but warns of a possible error if the operands were 2's complement signed. The C flag will be set to the carry out of bit 31 of the ALU, the Z flag will be set if and only if the result was zero, and the N flag will be set to the value of bit 31 of the result (indicating a negative result if the operands are considered to be 2's complement signed).

SHIFTS

When the second operand is specified to be a shifted register, the operation of the barrel shifter is controlled by the Shift field in the instruction. This field indicates the type of shift to be performed (logical left or right, arithmetic right or rotate right). The amount by which the register should be shifted may be contained in an immediate field in the instruction, or in the bottom byte of another register (other than R15). The encoding for the different shift types is shown in Figure 3-5.

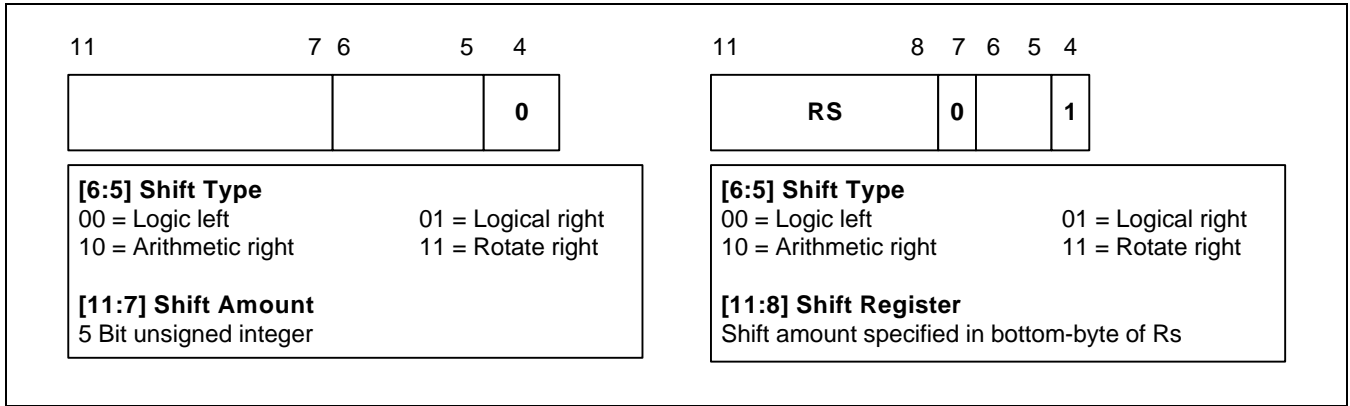


Figure 3-5. ARM Shift Operations

Instruction Specified Shift Amount

When the shift amount is specified in the instruction, it is contained in a 5 bit field which may take any value from 0 to 31. A logical shift left (LSL) takes the contents of Rm and moves each bit by the specified amount to a more significant position. The least significant bits of the result are filled with zeros, and the high bits of Rm which do not map into the result are discarded, except that the least significant discarded bit becomes the shifter carry output which may be latched into the C bit of the CPSR when the ALU operation is in the logical class (see above). For example, the effect of LSL #5 is shown in Figure 3-6.

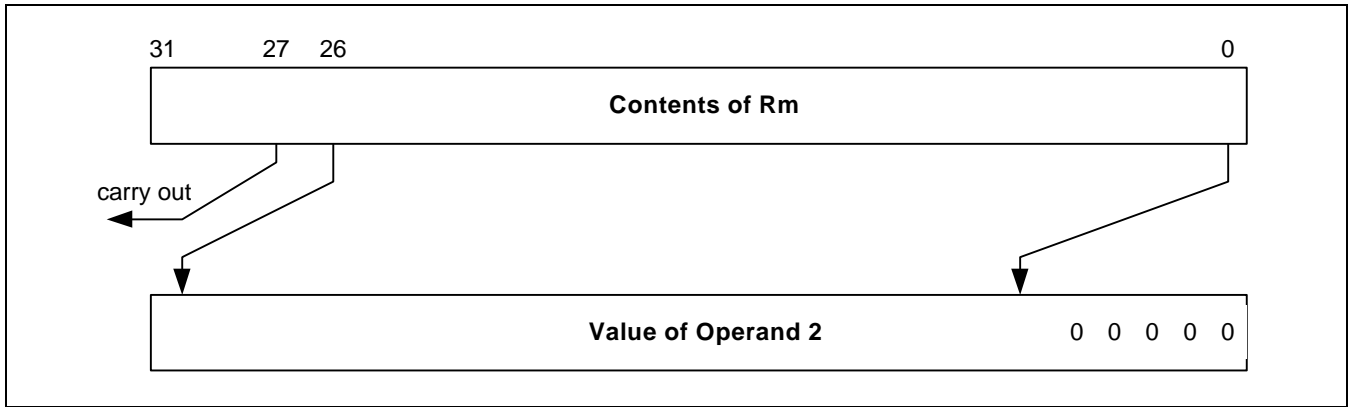


Figure 3-6. Logical Shift Left

NOTE

LSL #0 is a special case, where the shifter carry out is the old value of the CPSR C flag. The contents of Rm are used directly as the second operand. A logical shift right (LSR) is similar, but the contents of Rm are moved to less significant positions in the result. LSR #5 has the effect shown in Figure 3-7.

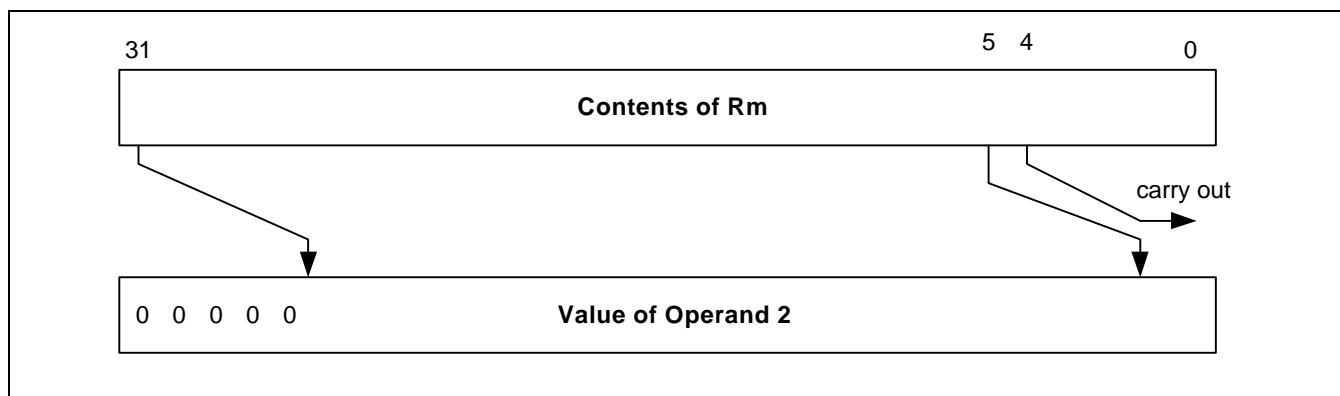


Figure 3-7. Logical Shift Right

The form of the shift field which might be expected to correspond to LSR #0 is used to encode LSR #32, which has a zero result with bit 31 of Rm as the carry output. Logical shift right zero is redundant as it is the same as logical shift left zero, so the assembler will convert LSR #0 (and ASR #0 and ROR #0) into LSL #0, and allow LSR #32 to be specified.

An arithmetic shift right (ASR) is similar to logical shift right, except that the high bits are filled with bit 31 of Rm instead of zeros. This preserves the sign in 2's complement notation. For example, ASR #5 is shown in Figure 3-8.

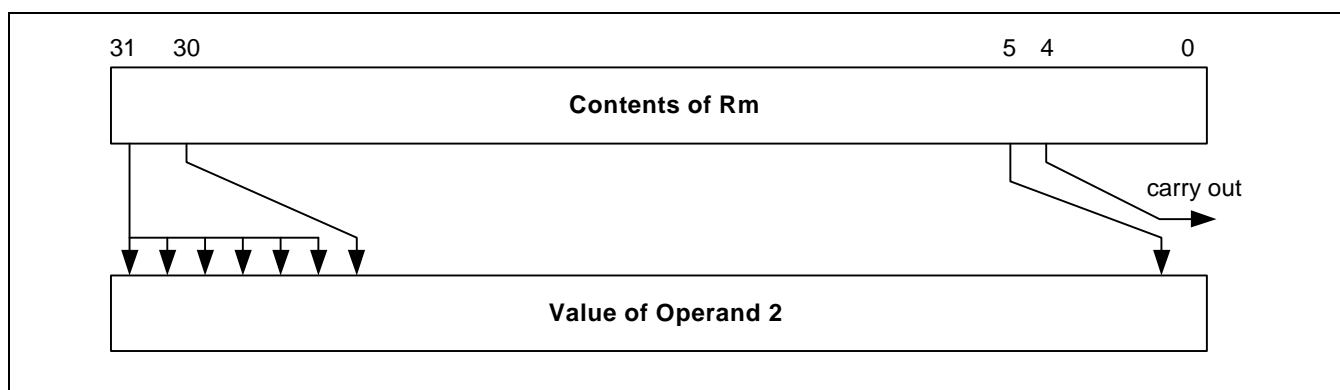


Figure 3-8. Arithmetic Shift Right

The form of the shift field which might be expected to give ASR #0 is used to encode ASR #32. Bit 31 of Rm is again used as the carry output, and each bit of operand 2 is also equal to bit 31 of Rm. The result is therefore all ones or all zeros, according to the value of bit 31 of Rm.

Rotate right (ROR) operations reuse the bits which “overshoot” in a logical shift right operation by reintroducing them at the high end of the result, in place of the zeros used to fill the high end in logical right operations. For example, ROR #5 is shown in Figure 3-9. The form of the shift field which might be expected to give ROR #0 is used to encode a special function of the barrel shifter, rotate right extended (RRX). This is a rotate right by one bit position of the 33 bit quantity formed by appending the CPSR C flag to the most significant end of the contents of Rm as shown in Figure 3-10.

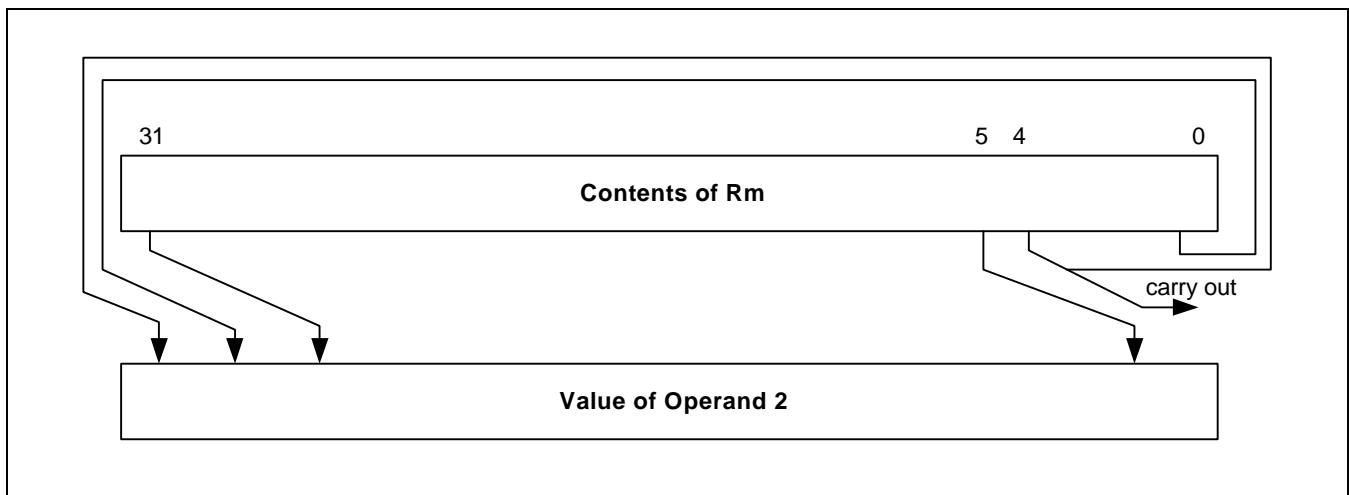


Figure 3-9. Rotate Right

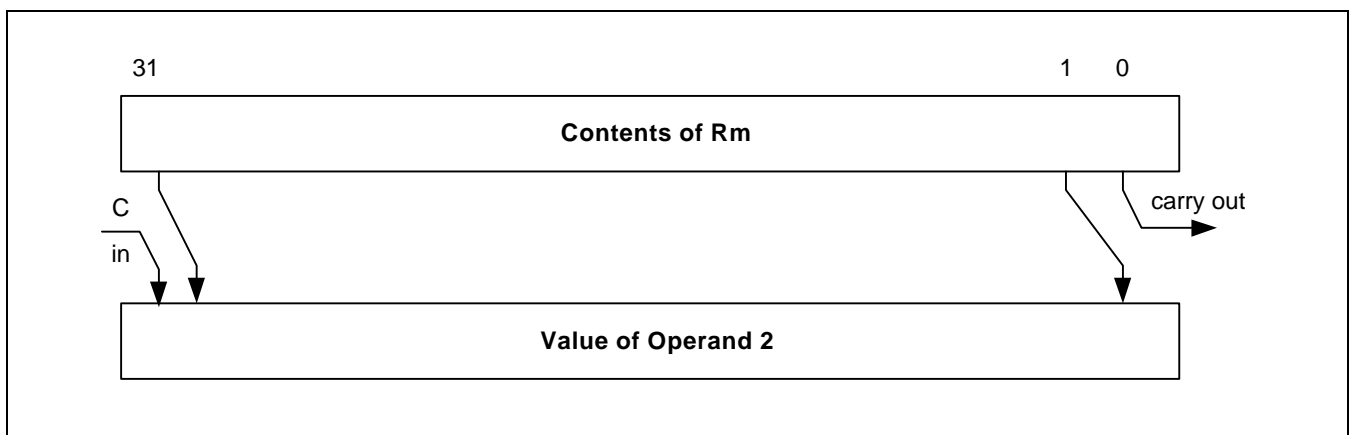


Figure 3-10. Rotate Right Extended

Register Specified Shift Amount

Only the least significant byte of the contents of Rs is used to determine the shift amount. Rs can be any general register other than R15.

If this byte is zero, the unchanged contents of Rm will be used as the second operand, and the old value of the CPSR C flag will be passed on as the shifter carry output.

If the byte has a value between 1 and 31, the shifted result will exactly match that of an instruction specified shift with the same value and shift operation.

If the value in the byte is 32 or more, the result will be a logical extension of the shift described above:

1. LSL by 32 has result zero, carry out equal to bit 0 of Rm.
2. LSL by more than 32 has result zero, carry out zero.
3. LSR by 32 has result zero, carry out equal to bit 31 of Rm.
4. LSR by more than 32 has result zero, carry out zero.
5. ASR by 32 or more has result filled with and carry out equal to bit 31 of Rm.
6. ROR by 32 has result equal to Rm, carry out equal to bit 31 of Rm.
7. ROR by n where n is greater than 32 will give the same result and carry out as ROR by n-32; therefore repeatedly subtract 32 from n until the amount is in the range 1 to 32 and see above.

NOTE

The zero in bit 7 of an instruction with a register controlled shift is compulsory; a one in this bit will cause the instruction to be a multiply or undefined instruction.

IMMEDIATE OPERAND ROTATES

The immediate operand rotate field is a 4 bit unsigned integer which specifies a shift operation on the 8 bit immediate value. This value is zero extended to 32 bits, and then subject to a rotate right by twice the value in the rotate field. This enables many common constants to be generated, for example all powers of 2.

WRITING TO R15

When Rd is a register other than R15, the condition code flags in the CPSR may be updated from the ALU flags as described above.

When Rd is R15 and the S flag in the instruction is not set the result of the operation is placed in R15 and the CPSR is unaffected.

When Rd is R15 and the S flag is set the result of the operation is placed in R15 and the SPSR corresponding to the current mode is moved to the CPSR. This allows state change which atomically restore both PC and CPSR. This form of instruction should not be used in User mode.

USING R15 AS AN OPERAND

If R15 (the PC) is used as an operand in a data processing instruction the register is used directly.

The PC value will be the address of the instruction, plus 8 or 12 bytes due to instruction prefetching. If the shift amount is specified in the instruction, the PC will be 8 bytes ahead. If a register is used to specify the shift amount the PC will be 12 bytes ahead.

TEQ, TST, CMP AND CMN OPCODES

NOTE

TEQ, TST, CMP and CMN do not write the result of their operation but do set flags in the CPSR. An assembler should always set the S flag for these instructions even if this is not specified in the mnemonic.

The TEQP form of the TEQ instruction used in earlier ARM processors must not be used: the PSR transfer operations should be used instead.

The action of TEQP in the ARM7TDMI is to move SPSR_<mode> to the CPSR if the processor is in a privileged mode and to do nothing if in User mode.

INSTRUCTION CYCLE TIMES

Data processing instructions vary in the number of incremental cycles taken as follows:

Table 3-4. Incremental Cycle Times

Processing Type	Cycles
Normal data processing	1S
Data processing with register specified shift	1S + 1I
Data processing with PC written	2S + 1N
Data processing with register specified shift and PC written	2S + 1N + 1I

NOTE: S, N and I are as defined sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle) respectively .

ASSEMBLER SYNTAX

- MOV,MVN (single operand instructions).
<opcode>{cond}{S} Rd,<Op2>
- CMP,CMN,TEQ,TST (instructions which do not produce a result).
<opcode>{cond} Rn,<Op2>
- AND,EOR,SUB,RSB,ADD,ADC,SBC,RSC,ORR,BIC
<opcode>{cond}{S} Rd,Rn,<Op2>

<Op2>	Rm{,<shift>} or,<#expression>
{cond}	A two-character condition mnemonic. See Table 3-2.
{S}	Set condition codes if S present (implied for CMP, CMN, TEQ, TST).
Rd, Rn and Rm	Expressions evaluating to a register number.
<#expression>	If this is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.
<shift>	<Shiftname> <register> or <shiftname> #expression, or RRX (rotate right one bit with extend).
<shiftname>s	ASL, LSL, LSR, ASR, ROR. (ASL is a synonym for LSL, they assemble to the same code.)

Examples

ADDEQ	R2, R4, R5	; If the Z flag is set make R2: = R4 + R5
TEQS	R4, #3	; Test R4 for equality with 3.
		; (The S is in fact redundant as the
		; assembler inserts it automatically.)
SUB	R4, R5, R7, LSR R2	; Logical right shift R7 by the number in
		; the bottom byte of R2, subtract result
		; from R5, and put the answer into R4.
MOV	PC, R14	; Return from subroutine.
MOVS	PC, R14	; Return from exception and restore CPSR
		; from SPSR_mode.

PSR TRANSFER (MRS, MSR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.

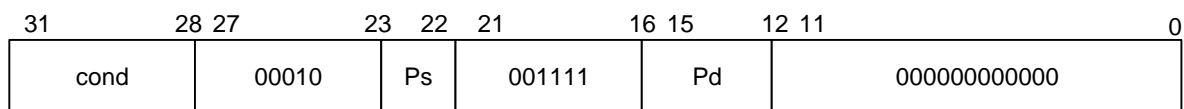
The MRS and MSR instructions are formed from a subset of the Data Processing operations and are implemented using the TEQ, TST, CMN and CMP instructions without the S flag set. The encoding is shown in Figure 3-11.

These instructions allow access to the CPSR and SPSR registers. The MRS instruction allows the contents of the CPSR or SPSR_<mode> to be moved to a general register. The MSR instruction allows the contents of a general register to be moved to the CPSR or SPSR_<mode> register.

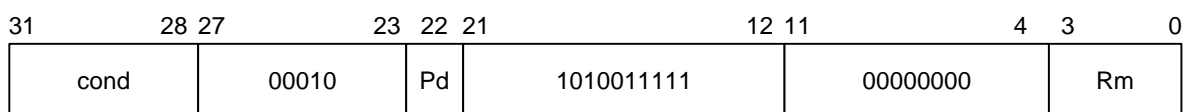
The MSR instruction also allows an immediate value or register contents to be transferred to the condition code flags (N,Z,C and V) of CPSR or SPSR_<mode> without affecting the control bits. In this case, the top four bits of the specified register contents or 32 bit immediate value are written to the top four bits of the relevant PSR.

OPERAND RESTRICTIONS

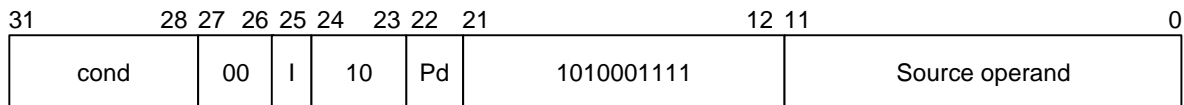
- In user mode, the control bits of the CPSR are protected from change, so only the condition code flags of the CPSR can be changed. In other (privileged) modes the entire CPSR can be changed.
- Note that the software must never change the state of the T bit in the CPSR. If this happens, the processor will enter an unpredictable state.
- The SPSR register which is accessed depends on the mode at the time of execution. For example, only SPSR_fiq is accessible when the processor is in FIQ mode.
- You must not specify R15 as the source or destination register.
- Also, do not attempt to access an SPSR in User mode, since no such register exists.

MRS (Transfer PSR Contents to a Register)**[15:12] Destination Register****[22] Source PSR**

0 = CPSR 1 = SPSR_<current mode>

[31:28] Condition Field**MRS (Transfer Register Contents to PSR)****[3:0] Source Register****[22] Destination PSR**

0 = CPSR 1 = SPSR_<current mode>

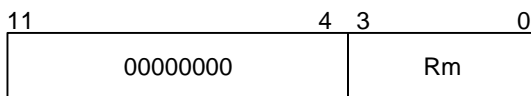
[31:28] Condition Field**MRS (transfer register contents or immediate value to PSR flag bits only)****[22] Destination PSR**

0 = CPSR 1 = SPSR_<current mode>

[25] Immediate Operand

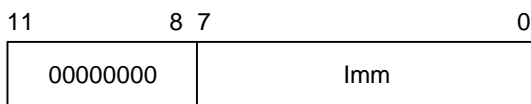
0 = Source operand is a register

1 = SPSR_<current mode>

[11:0] Source Operand

[3] Source register

[11:4] Source operand is an immediate value



[7:0] Unsigned 8 bit immediate value

[11:8] Shift applied to Imm

[31:28] Confition Field**Figure 3-11. PSR Transfer**

RESERVED BITS

Only twelve bits of the PSR are defined in ARM7TDMI (N,Z,C,V,I,F, T & M[4:0]); the remaining bits are reserved for use in future versions of the processor. Refer to Figure 2-6 for a full description of the PSR bits.

To ensure the maximum compatibility between ARM7TDMI programs and future processors, the following rules should be observed:

- The reserved bits should be preserved when changing the value in a PSR.
- Programs should not rely on specific values from the reserved bits when checking the PSR status, since they may read as one or zero in future processors.

A read-modify-write strategy should therefore be used when altering the control bits of any PSR register; this involves transferring the appropriate PSR register to a general register using the MRS instruction, changing only the relevant bits and then transferring the modified value back to the PSR register using the MSR instruction.

Examples

The following sequence performs a mode change:

MRS	R0,CPSR	; Take a copy of the CPSR.
BIC	R0,R0,#0x1F	; Clear the mode bits.
ORR	R0,R0,#new_mode	; Select new mode
MSR	CPSR,R0	; Write back the modified CPSR.

When the aim is simply to change the condition code flags in a PSR, a value can be written directly to the flag bits without disturbing the control bits. The following instruction sets the N, Z, C and V flags:

MSR	CPSR_flg,#0xF0000000	; Set all the flags regardless of their previous state
		; (does not affect any control bits).

No attempt should be made to write an 8 bit immediate value into the whole PSR since such an operation cannot preserve the reserved bits.

INSTRUCTION CYCLE TIMES

PSR transfers take 1S incremental cycles, where S is defined as sequential (S-cycle).

ASSEMBLER SYNTAX

- MRS - transfer PSR contents to a register
MRS{cond} Rd,<psr>
- MSR - transfer register contents to PSR
MSR{cond} <psr>,Rm
- MSR - transfer register contents to PSR flag bits only
MSR{cond} <psrf>,Rm

The most significant four bits of the register contents are written to the N,Z,C & V flags respectively.

- MSR - transfer immediate value to PSR flag bits only
MSR{cond} <psrf>,<#expression>

The expression should symbolise a 32 bit value of which the most significant four bits are written to the N,Z,C and V flags respectively.

Key:

{cond}	Two-character condition mnemonic. See Table 3-2.
Rd and Rm	Expressions evaluating to a register number other than R15
<psr>	CPSR, CPSR_all, SPSR or SPSR_all. (CPSR and CPSR_all are synonyms as are SPSR and SPSR_all)
<psrf>	CPSR_flg or SPSR_flg
<#expression>	Where this is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.

Examples

In user mode the instructions behave as follows:

MSR	CPSR_all,Rm	; CPSR[31:28] ← Rm[31:28]
MSR	CPSR_flg,Rm	; CPSR[31:28] ← Rm[31:28]
MSR	CPSR_flg,#0xA0000000	; CPSR[31:28] ← 0xA (set N, C; clear Z, V)
MRS	Rd,CPSR	; Rd[31:0] ← CPSR[31:0]

In privileged modes the instructions behave as follows:

MSR	CPSR_all,Rm	; CPSR[31:0] ← Rm[31:0]
MSR	CPSR_flg,Rm	; CPSR[31:28] ← Rm[31:28]
MSR	CPSR_flg,#0x50000000	; CPSR[31:28] ← 0x5 (set Z, V; clear N, C)
MSR	SPSR_all,Rm	; SPSR_<mode>[31:0] ← Rm[31:0]
MSR	SPSR_flg,Rm	; SPSR_<mode>[31:28] ← Rm[31:28]
MSR	SPSR_flg,#0xC0000000	; SPSR_<mode>[31:28] ← 0xC (set N, Z; clear C, V)
MRS	Rd,SPSR	; Rd[31:0] ← SPSR_<mode>[31:0]

MULTIPLY AND MULTIPLY-ACCUMULATE (MUL, MLA)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-12.

The multiply and multiply-accumulate instructions use an 8 bit Booth's algorithm to perform integer multiplication.

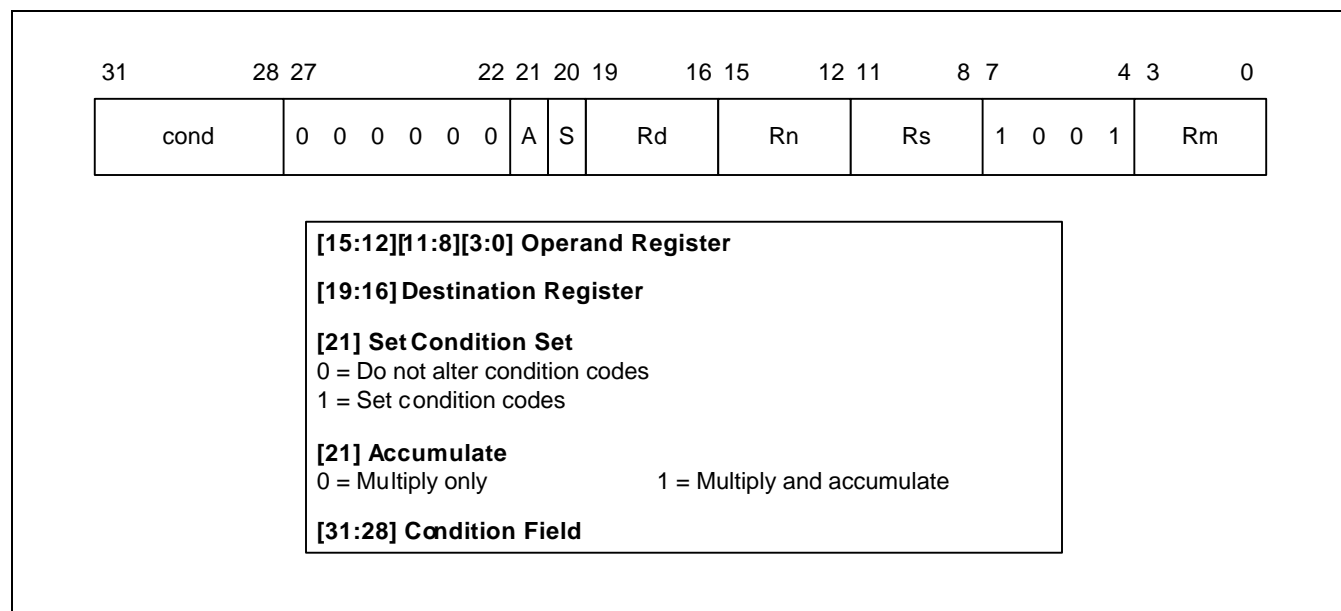


Figure 3-12. Multiply Instructions

The multiply form of the instruction gives $Rd = Rm * Rs$. Rn is ignored, and should be set to zero for compatibility with possible future upgrades to the instruction set. The multiply-accumulate form gives $Rd = Rm * Rs + Rn$, which can save an explicit ADD instruction in some circumstances. Both forms of the instruction work on operands which may be considered as signed (2's complement) or unsigned integers.

The results of a signed multiply and of an unsigned multiply of 32 bit operands differ only in the upper 32 bits-the low 32 bits of the signed and unsigned results are identical. As these instructions only produce the low 32 bits of a multiply, they can be used for both signed and unsigned multiplies.

For example consider the multiplication of the operands:

Operand A	Operand B	Result
0xFFFFFFFF6	0x0000001	0xFFFFFFFF38

If the Operands Are Interpreted as Signed

Operand A has the value -10, operand B has the value 20, and the result is -200 which is correctly represented as 0xFFFFFFFF38.

If the Operands Are Interpreted as Unsigned

Operand A has the value 4294967286, operand B has the value 20 and the result is 85899345720, which is represented as 0x13FFFFFF38, so the least significant 32 bits are 0xFFFFFFFF38.

Operand Restrictions

The destination register Rd must not be the same as the operand register Rm . $R15$ must not be used as an operand or as the destination register.

All other register combinations will give correct results, and Rd, Rn and Rs may use the same register when required.

CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N (Negative) and Z (Zero) flags are set correctly on the result (N is made equal to bit 31 of the result, and Z is set if and only if the result is zero). The C (Carry) flag is set to a meaningless value and the V (oVerflow) flag is unaffected.

INSTRUCTION CYCLE TIMES

MUL takes 1S + mI and MLA 1S + (m+1)I cycles to execute, where S and I are defined as sequential (S-cycle) and internal (I-cycle), respectively.

m: The number of 8 bit multiplier array cycles is required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs. Its possible values are as follows

1. If bits [32:8] of the multiplier operand are all zero or all one.
2. If bits [32:16] of the multiplier operand are all zero or all one.
3. If bits [32:24] of the multiplier operand are all zero or all one.
4. In all other cases.

ASSEMBLER SYNTAX

MUL{cond}{S} Rd,Rm,Rs
MLA{cond}{S} Rd,Rm,Rs,Rn

{cond} Two-character condition mnemonic. See Table 3-2.

{S} Set condition codes if S present

Rd, Rm, Rs and Rn Expressions evaluating to a register number other than R15.

Examples

MUL	R1,R2,R3	; R1: = R2*R3
MLAEQS	R1,R2,R3,R4	; Conditionally R1: = R2*R3+R4, Setting condition codes.

MULTIPLY LONG AND MULTIPLY-ACCUMULATE LONG (MULL,MLAL)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-13.

The multiply long instructions perform integer multiplication on two 32 bit operands and produce 64 bit results. signed and unsigned multiplication each with optional accumulate give rise to four variations.

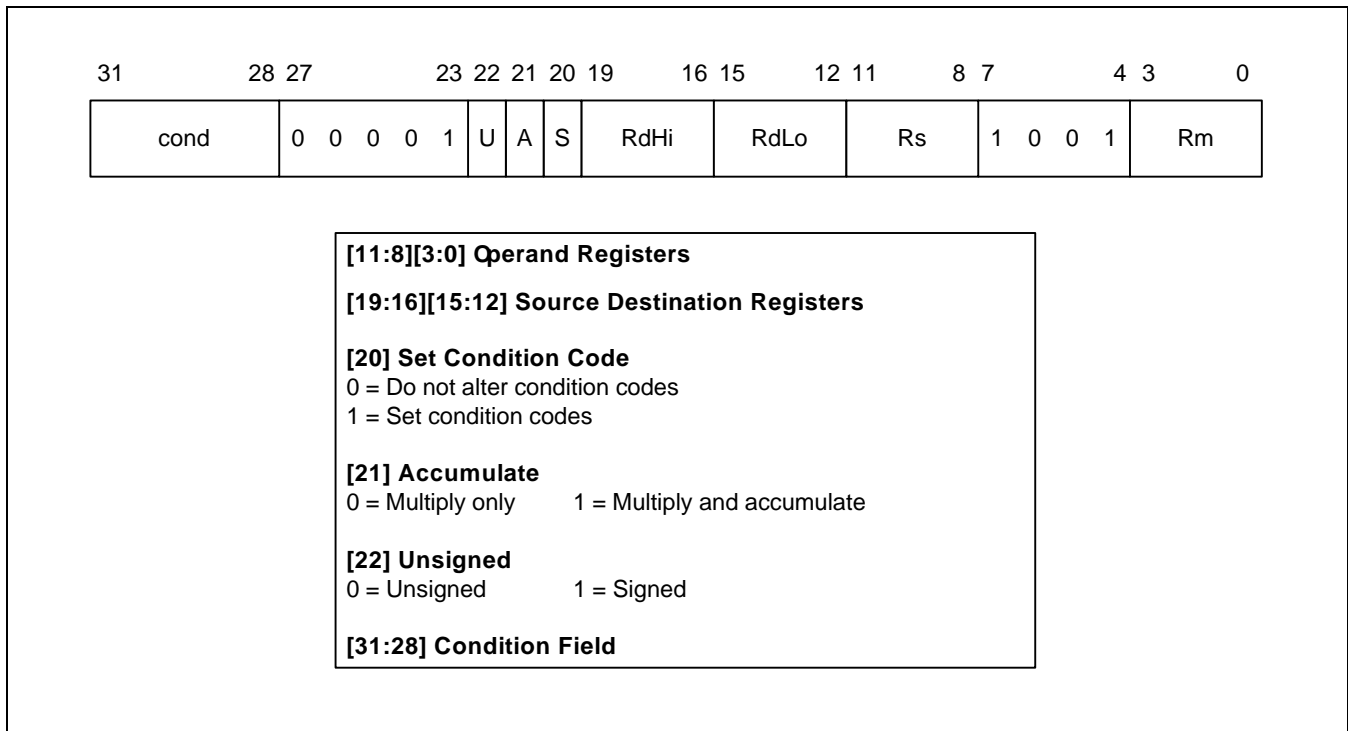


Figure 3-13. Multiply Long Instructions

The multiply forms (UMULL and SMULL) take two 32 bit numbers and multiply them to produce a 64 bit result of the form RdHi, RdLo: = Rm * Rs. The lower 32 bits of the 64 bit result are written to RdLo, the upper 32 bits of the result are written to RdHi.

The multiply-accumulate forms (UMLAL and SMLAL) take two 32 bit numbers, multiply them and add a 64 bit number to produce a 64 bit result of the form RdHi, RdLo: = Rm * Rs + RdHi, RdLo. The lower 32 bits of the 64 bit number to add is read from RdLo. The upper 32 bits of the 64 bit number to add is read from RdHi. The lower 32 bits of the 64 bit result are written to RdLo. The upper 32 bits of the 64 bit result are written to RdHi.

The UMULL and UMLAL instructions treat all of their operands as unsigned binary numbers and write an unsigned 64 bit result. The SMULL and SMLAL instructions treat all of their operands as two's-complement signed numbers and write a two's-complement signed 64 bit result.

OPERAND RESTRICTIONS

- R15 must not be used as an operand or as a destination register.
- RdHi, RdLo, and Rm must all specify different registers.

CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N and Z flags are set correctly on the result (N is equal to bit 63 of the result, Z is set if and only if all 64 bits of the result are zero). Both the C and V flags are set to meaningless values.

INSTRUCTION CYCLE TIMES

MULL takes $1S + (m+1)I$ and MLAL $1S + (m+2)I$ cycles to execute, where m is the number of 8 bit multiplier array cycles required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs.

Its possible values are as follows:

For Signed Instructions SMULL, SMLAL:

- If bits [31:8] of the multiplier operand are all zero or all one.
- If bits [31:16] of the multiplier operand are all zero or all one.
- If bits [31:24] of the multiplier operand are all zero or all one.
- In all other cases.

For Unsigned Instructions UMULL, UMLAL:

- If bits [31:8] of the multiplier operand are all zero.
- If bits [31:16] of the multiplier operand are all zero.
- If bits [31:24] of the multiplier operand are all zero.
- In all other cases.

S and I are defined as sequential (S-cycle) and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

Table 3-5. Assembler Syntax Descriptions

Mnemonic	Description	Purpose
UMULL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned multiply long	$32 \times 32 = 64$
UMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned multiply & accumulate long	$32 \times 32 + 64 = 64$
SMULL{cond}{S} RdLo,RdHi,Rm,Rs	Signed multiply long	$32 \times 32 = 64$
SMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Signed multiply & accumulate long	$32 \times 32 + 64 = 64$

{cond} Two-character condition mnemonic. See Table 3-2.

{S} Set condition codes if S present

RdLo, RdHi, Rm, Rs Expressions evaluating to a register number other than R15.

Examples

```

UMULL    R1,R4,R2,R3      ; R4,R1: = R2*R3
UMLALS   R1,R5,R2,R3      ; R5,R1: = R2*R3+R5,R1 also setting condition codes

```

SINGLE DATA TRANSFER (LDR, STR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-14.

The single data transfer instructions are used to load or store single bytes or words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register.

The result of this calculation may be written back into the base register if auto-indexing is required.

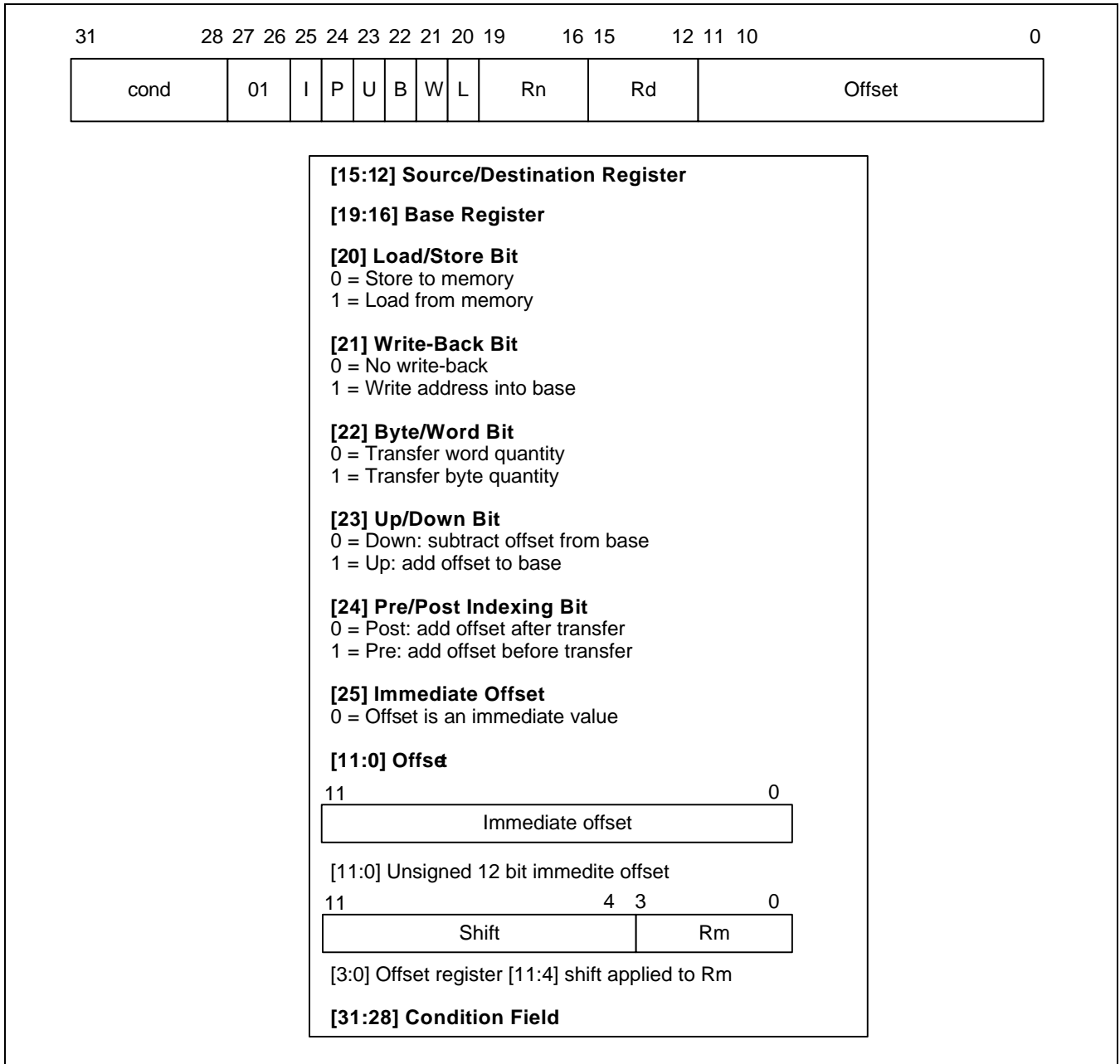


Figure 3-14. Single Data Transfer Instructions

OFFSETS AND AUTO-INDEXING

The offset from the base may be either a 12 bit unsigned binary immediate value in the instruction, or a second register (possibly shifted in some way). The offset may be added to ($U = 1$) or subtracted from ($U = 0$) the base register R_n . The offset modification may be performed either before (pre-indexed, $P = 1$) or after (post-indexed, $P = 0$) the base is used as the transfer address.

The W bit gives optional auto increment and decrement addressing modes. The modified base value may be written back into the base ($W = 1$), or the old base value may be kept ($W = 0$). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base. The only use of the W bit in a post-indexed data transfer is in privileged mode code, where setting the W bit forces non-privileged mode for the transfer, allowing the operating system to generate a user address in a system where the memory management hardware makes suitable use of this hardware.

SHIFTED REGISTER OFFSET

The 8 shift control bits are described in the data processing instructions section. However, the register specified shift amounts are not available in this instruction class. See Figure 3-5.

BYTES AND WORDS

This instruction class may be used to transfer a byte ($B = 1$) or a word ($B = 0$) between an ARM7TDMI register and memory.

The action of LDR(B) and STR(B) instructions is influenced by the BIGEND control signal of ARM7TDMI core. The two possible configurations are described below.

NOTE

The KS32C65100 is configured to the big-endian format.

Little-Endian Configuration

A byte load (LDRB) expects the data on data bus inputs 7 through 0 if the supplied address is on a word boundary, on data bus inputs 15 through 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register, and the remaining bits of the register are filled with zeros. Please see Figure 2-2.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) will normally use a word aligned address. However, an address offset from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 0 to 7. This means that half-words accessed at offsets 0 and 2 from the word boundary will be correctly loaded into bits 0 through 15 of the register. Two shift operations are then required to clear or to sign extend the upper 16 bits.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

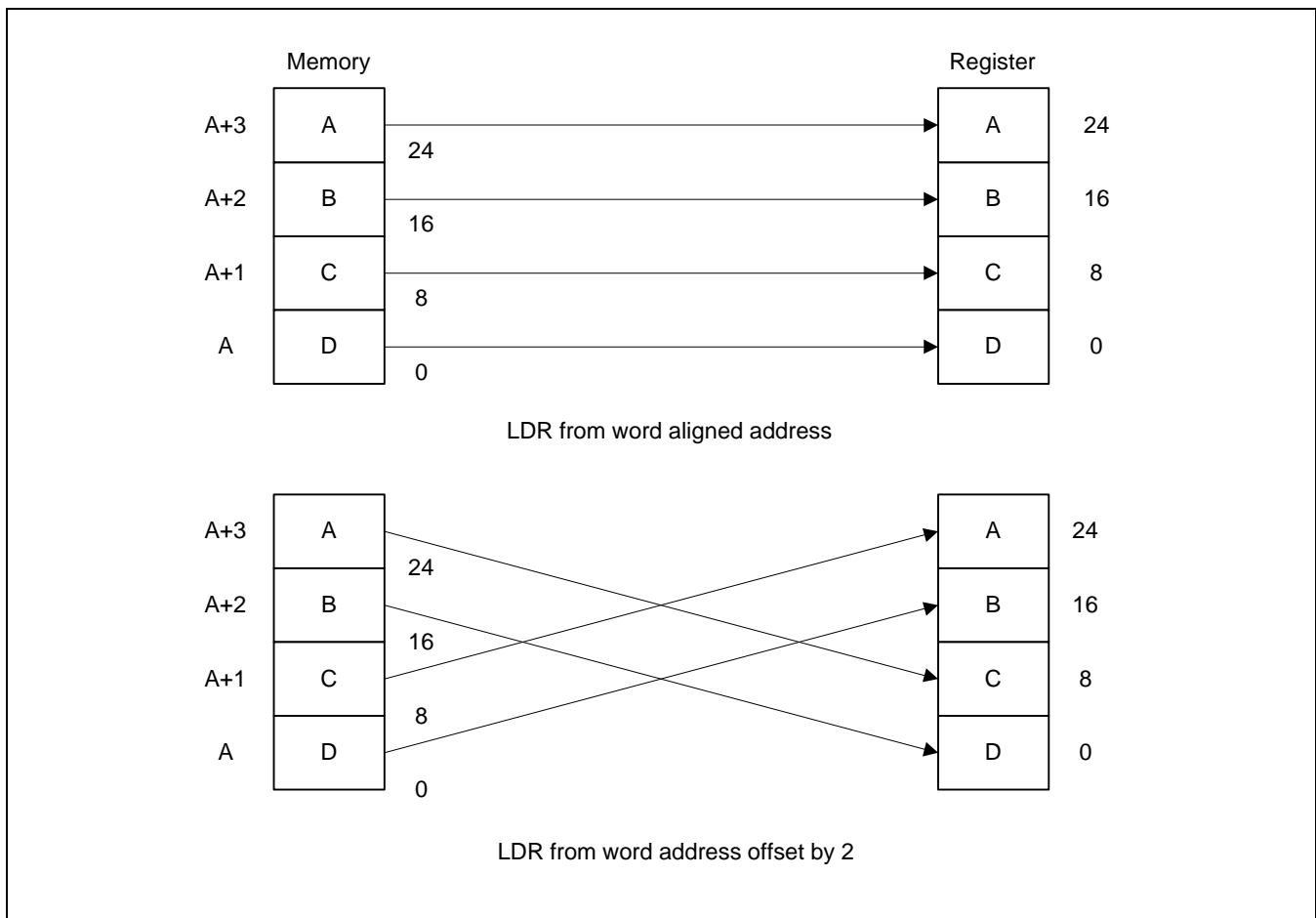


Figure 3-15. Little-Endian Offset Addressing

Big-Endian Configuration

A byte load (LDRB) expects the data on data bus inputs 31 through 24 if the supplied address is on a word boundary, on data bus inputs 23 through 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register and the remaining bits of the register are filled with zeros. please see Figure 2-1.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) should generate a word aligned address. An address offset of 0 or 2 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 31 through 24. This means that half-words accessed at these offsets will be correctly loaded into bits 16 through 31 of the register. A shift operation is then required to move (and optionally sign extend) the data into the bottom 16 bits. An address offset of 1 or 3 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 15 through 8.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

USE OF R15

Write-back must not be specified if R15 is specified as the base register (Rn). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 must not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a register store (STR) instruction, the stored value will be address of the instruction plus 12.

RESTRICTION ON THE USE OF BASE REGISTER

When configured for late aborts, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

After an abort, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

Example:

```
LDR      R0,[R1],R1
```

Therefore a post-indexed LDR or STR where Rm is the same register as Rn should not be used.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory the required data may be absent from main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the data abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Normal LDR instructions take $1S + 1N + 1I$ and LDR PC take $2S + 2N + 1I$ incremental cycles, where S, N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STR instructions take 2N incremental cycles to execute.

ASSEMBLER SYNTAX

<LDR|STR>{cond}{B}{T} Rd,<Address>

where:

LDR	Load from memory into a register						
STR	Store from a register into memory						
{cond}	Two-character condition mnemonic. See Table 3-2.						
{B}	If B is present then byte transfer, otherwise word transfer						
{T}	If T is present the W bit will be set in a post-indexed instruction, forcing non-privileged mode for the transfer cycle. T is not allowed when a pre-indexed addressing mode is specified or implied.						
Rd	An expression evaluating to a valid register number.						
Rn and Rm	Expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining. In this case base write-back should not be specified.						
<Address>can be:							
1	<p>An expression which generates an address: The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.</p>						
2	<p>A pre-indexed addressing specification:</p> <table> <tr> <td>[Rn]</td><td>offset of zero</td></tr> <tr> <td>[Rn,<#expression>]{!}</td><td>offset of <expression> bytes</td></tr> <tr> <td>[Rn,{+/-}Rm{,<shift>}]</td><td>offset of +/- contents of index register, shifted by <shift></td></tr> </table>	[Rn]	offset of zero	[Rn,<#expression>]{!}	offset of <expression> bytes	[Rn,{+/-}Rm{,<shift>}]	offset of +/- contents of index register, shifted by <shift>
[Rn]	offset of zero						
[Rn,<#expression>]{!}	offset of <expression> bytes						
[Rn,{+/-}Rm{,<shift>}]	offset of +/- contents of index register, shifted by <shift>						
3	<p>A post-indexed addressing specification:</p> <table> <tr> <td>[Rn,<#expression></td><td>offset of <expression> bytes</td></tr> <tr> <td>[Rn,{+/-}Rm{,<shift>}]</td><td>offset of +/- contents of index register, shifted as by <shift>.</td></tr> </table>	[Rn,<#expression>	offset of <expression> bytes	[Rn,{+/-}Rm{,<shift>}]	offset of +/- contents of index register, shifted as by <shift>.		
[Rn,<#expression>	offset of <expression> bytes						
[Rn,{+/-}Rm{,<shift>}]	offset of +/- contents of index register, shifted as by <shift>.						
<shift>	General shift operation (see data processing instructions) but you cannot specify the shift amount by a register.						
{!}	Writes back the base register (set the W bit) if ! is present.						

Examples

STR	R1,[R2,R4]!	; Store R1 at R2+R4 (both of which are registers)
		; and write back address to R2.
STR	R1,[R2],R4	; Store R1 at R2 and write back R2+R4 to R2.
LDR	R1,[R2,#16]	; Load R1 from contents of R2+16, but don't write back.
LDR	R1,[R2,R3,LSL#2]	; Load R1 from contents of R2+R3*4.
LDREQB	R1,[R6,#5]	; Conditionally load byte at R6+5 into
		; R1 bits 0 to 7, filling bits 8 to 31 with zeros.
STR	R1,PLACE	; Generate PC relative offset to address PLACE.
PLACE		

HALFWORD AND SIGNED DATA TRANSFER (LDRH/STRH/LDRSB/LDRSH)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-16.

These instructions are used to load or store half-words of data and also load sign-extended bytes or half-words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register. The result of this calculation may be written back into the base register if auto-indexing is required.

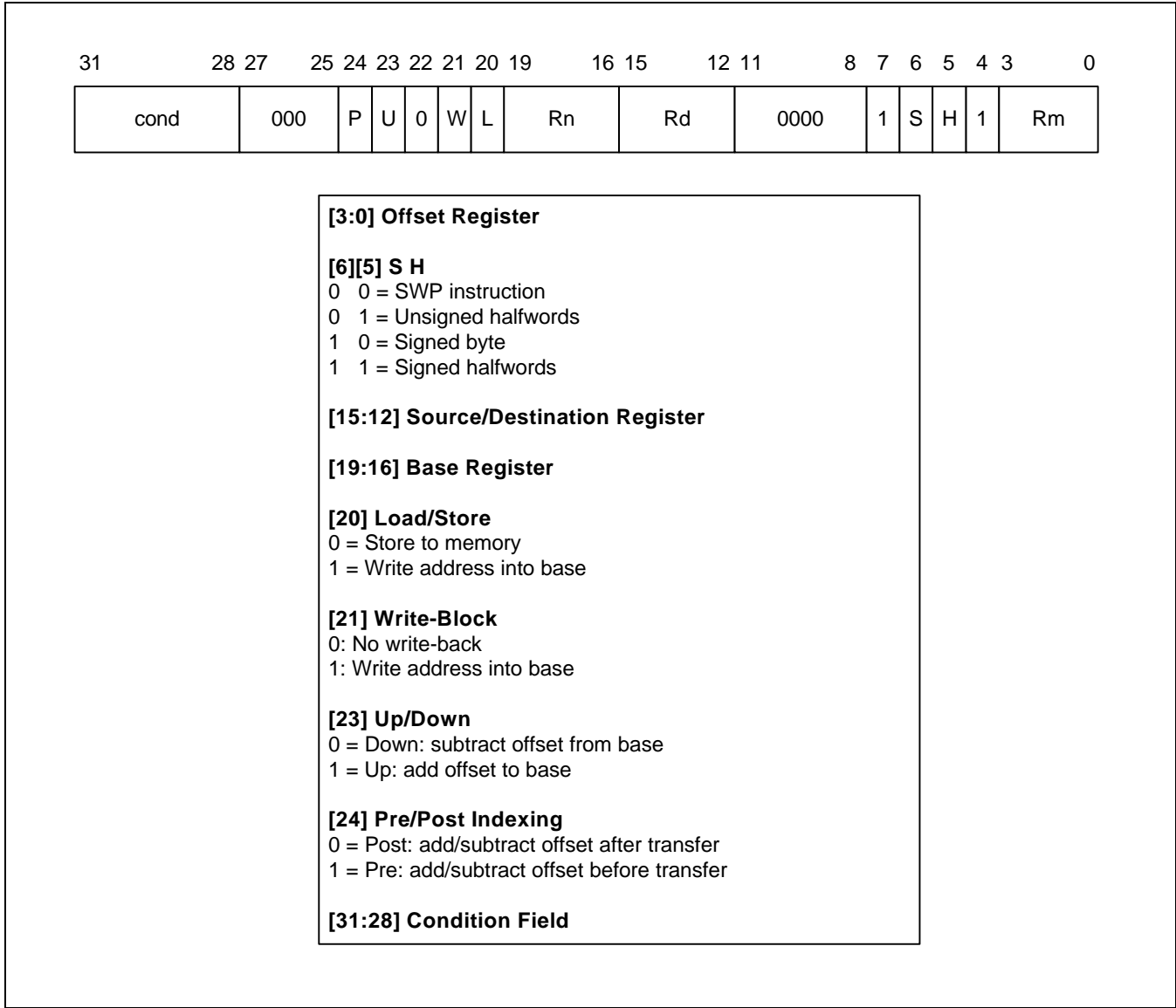


Figure 3-16. Halfword and Signed Data Transfer with Register Offset

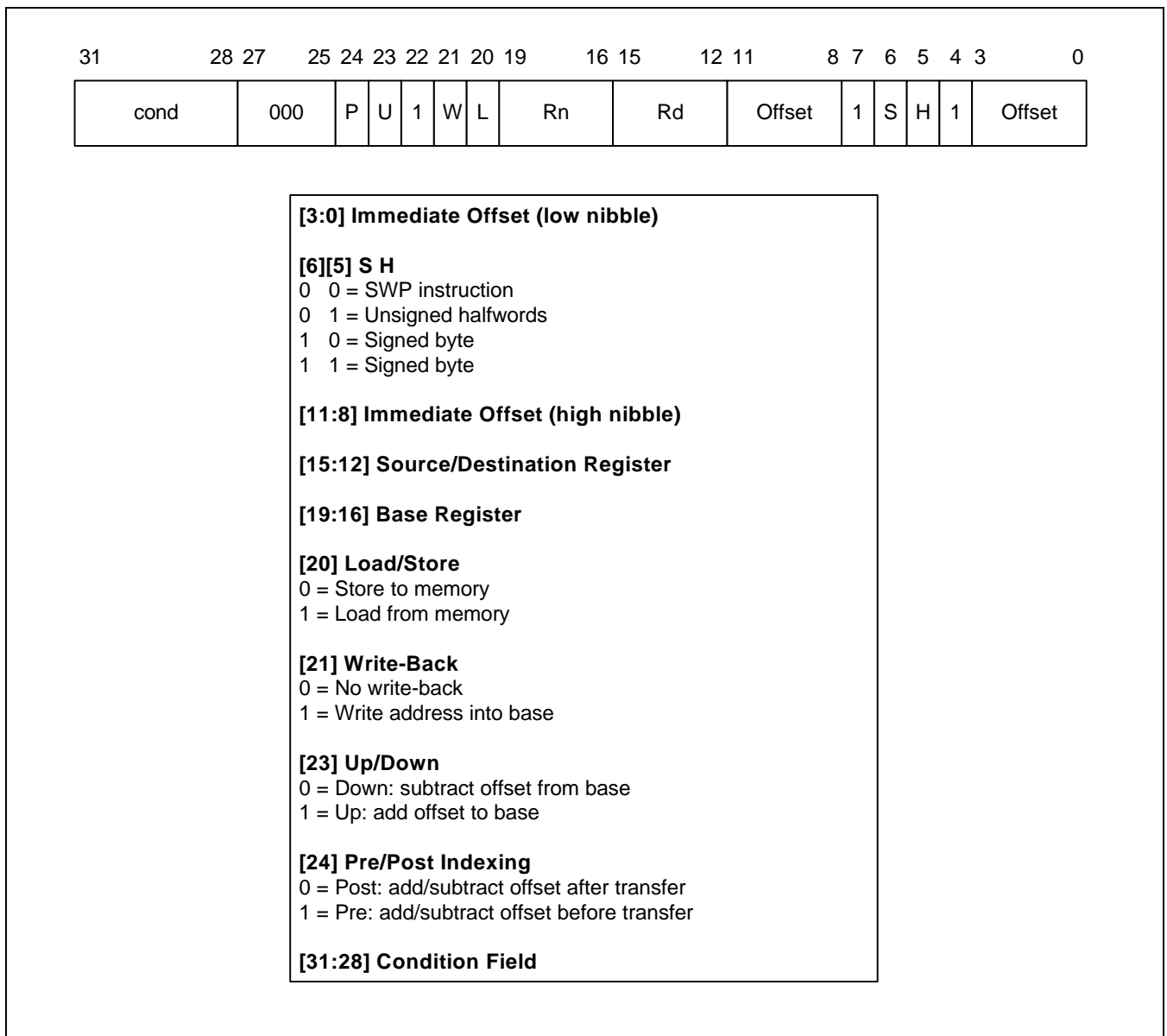


Figure3-17. Halfword and Signal Data Transfer with Immediate Offset and Auto-Indexing

OFFSETS AND AUTO-INDEXING

The offset from the base may be either a 8-bit unsigned binary immediate value in the instruction, or a second register. The 8-bit offset is formed by concatenating bits 11 to 8 and bits 3 to 0 of the instruction word, such that bit 11 becomes the MSB and bit 0 becomes the LSB. The offset may be added to ($U = 1$) or subtracted from ($U = 0$) the base register R_n . The offset modification may be performed either before (pre-indexed, $P = 1$) or after (post-indexed, $P = 0$) the base register is used as the transfer address.

The W bit gives optional auto-increment and decrement addressing modes. The modified base value may be written back into the base ($W = 1$), or the old base may be kept ($W = 0$). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained if necessary by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base. The Write-back bit should not be set high ($W = 1$) when post-indexed addressing is selected.

HALFWORD LOAD AND STORES

Setting $S = 0$ and $H = 1$ may be used to transfer unsigned Half-words between an ARM7TDMI register and memory.

The action of LDRH and STRH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the section below.

SIGNED BYTE AND HALFWORD LOADS

The S bit controls the loading of sign-extended data. When $S = 1$ the H bit selects between Bytes ($H = 0$) and Half-words ($H = 1$). The L bit should not be set low (Store) when signed ($S = 1$) operations have been selected.

The LDRSB instruction loads the selected Byte into bits 7 to 0 of the destination register and bits 31 to 8 of the destination register are set to the value of bit 7, the sign bit.

The LDRSH instruction loads the selected Half-word into bits 15 to 0 of the destination register and bits 31 to 16 of the destination register are set to the value of bit 15, the sign bit.

The action of the LDRSB and LDRSH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the following section.

ENDIANNESS AND BYTE/HALFWORD SELECTION

Little-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 7 through to 0 if the supplied address is on a word boundary, on data bus inputs 15 through to 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bit of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-2.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 15 through to 0 if the supplied address is on a word boundary and on data bus inputs 31 through to 16 if it is a halfword boundary, ($A[1]=1$). The supplied address should always be on a halfword boundary. If bit 0 of the supplied address is HIGH then the ARM7TDMI will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH) the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned, if bit 0 of the address is HIGH this will cause unpredictable behaviour.

Big-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 31 through to 24 if the supplied address is on a word boundary, on data bus inputs 23 through to 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bit of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-1.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 31 through to 16 if the supplied address is on a word boundary and on data bus inputs 15 through to 0 if it is a halfword boundary, ($A[1]=1$). The supplied address should always be on a halfword boundary. If bit 0 of the supplied address is HIGH then the ARM7TDMI will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH) the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned, if bit 0 of the address is HIGH this will cause unpredictable behaviour.

NOTE

The KS32C651000 is configured to the big-endian format.

USE OF R15

Write-back should not be specified if R15 is specified as the base register (R_n). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 should not be specified as the register offset (R_m).

When R15 is the source register (R_d) of a Half-word store (STRH) instruction, the stored address will be address of the instruction plus 12.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory the required data may be absent from the main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the data abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Normal LDR(H, SH, SB) instructions take $1S + 1N + 1I$. LDR(H, SH, SB) PC take $2S + 2N + 1I$ incremental cycles. S, N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STRH instructions take $2N$ incremental cycles to execute.

ASSEMBLER SYNTAX

<LDR|STR>{cond}<H|SH|SB> Rd,<address>

LDR	Load from memory into a register
STR	Store from a register into memory
{cond}	Two-character condition mnemonic. See Table 3-2..
H	Transfer halfword quantity
SB	Load sign extended byte (only valid for LDR)
SH	Load sign extended halfword (only valid for LDR)
Rd	An expression evaluating to a valid register number.

<address> can be:

- 1 An expression which generates an address:
The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.
- 2 A pre-indexed addressing specification:

[Rn]	offset of zero
[Rn,<#expression>]{!}	offset of <expression> bytes
[Rn,{+/-}Rm]{!}	offset of +/- contents of index register
- 3 A post-indexed addressing specification:

[Rn],<#expression>	offset of <expression> bytes
[Rn},{+/-}Rm	offset of +/- contents of index register.
- 4 Rn and Rm are expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining. In this case base write-back should not be specified.
- {!} Writes back the base register (set the W bit) if ! is present.

Examples

LDRH	R1,[R2,-R3]!	; Load R1 from the contents of the halfword address ; contained in R2-R3 (both of which are registers) ; and write back address to R2
STRH	R3,[R4,#14]	; Store the halfword in R3 at R14+14 but don't write back.
LDRSB	R8,[R2],#-223	; Load R8 with the sign extended contents of the byte ; address contained in R2 and write back R2-223 to R2.
LDRNESH	R11,[R0]	; Conditionally load R11 with the sign extended contents ; of the halfword address contained in R0.
HERE		; Generate PC relative offset to address FRED.
STRH	R5, [PC,#(FRED-HERE-8)];	Store the halfword in R5 at address FRED
FRED		

BLOCK DATA TRANSFER (LDM, STM)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-18.

Block data transfer instructions are used to load (LDM) or store (STM) any subset of the currently visible registers. They support all possible stacking modes, maintaining full or empty stacks which can grow up or down memory, and are very efficient instructions for saving or restoring context, or for moving large blocks of data around main memory.

THE REGISTER LIST

The instruction can cause the transfer of any registers in the current bank (and non-user mode programs can also transfer to and from the user bank, see below). The register list is a 16 bit field in the instruction, with each bit corresponding to a register. A 1 in bit 0 of the register field will cause R0 to be transferred, a 0 will cause it not to be transferred; similarly bit 1 controls the transfer of R1, and so on.

Any subset of the registers, or all the registers, may be specified. The only restriction is that the register list should not be empty.

Whenever R15 is stored to memory the stored value is the address of the STM instruction plus 12.

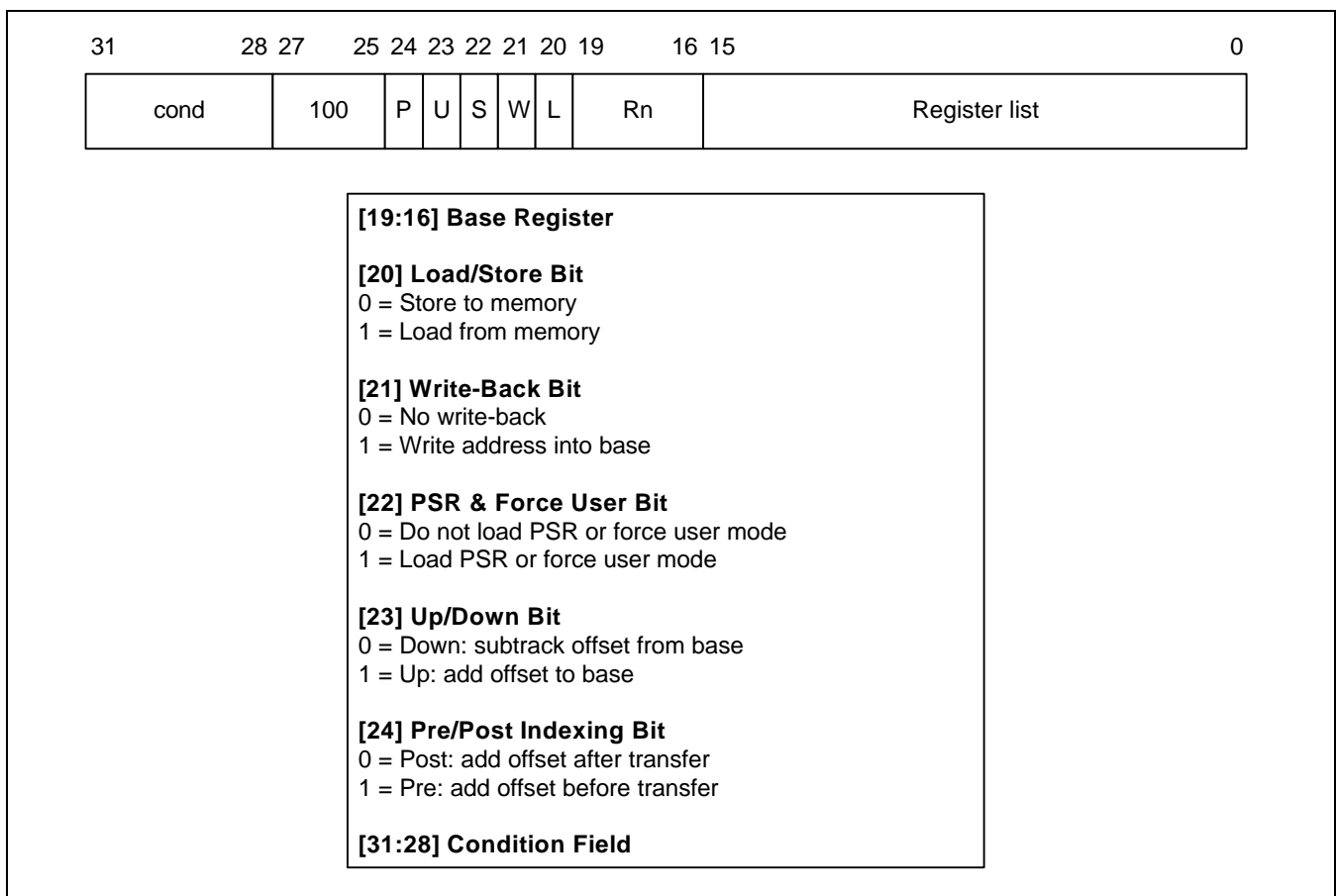


Figure 3-18. Block Data Transfer Instructions

ADDRESSING MODES

The transfer addresses are determined by the contents of the base register (R_n), the pre/post bit (P) and the up/down bit (U). The registers are transferred in the order lowest to highest, so R_{15} (if in the list) will always be transferred last. The lowest register also gets transferred to/from the lowest memory address. By way of illustration, consider the transfer of R_1 , R_5 and R_7 in the case where $R_n = 0x1000$ and write back of the modified base is required ($W = 1$). Figure 3.19-22 show the sequence of register transfers, the addresses used, and the value of R_n after the instruction has completed.

In all cases, had write back of the modified base not been required ($W = 0$), R_n would have retained its initial value of $0x1000$ unless it was also in the transfer list of a load multiple register instruction, when it would have been overwritten with the loaded value.

ADDRESS ALIGNMENT

The address should normally be a word aligned quantity and non-word aligned addresses do not affect the instruction. However, the bottom 2 bits of the address will appear on $A[1:0]$ and might be interpreted by the memory system.

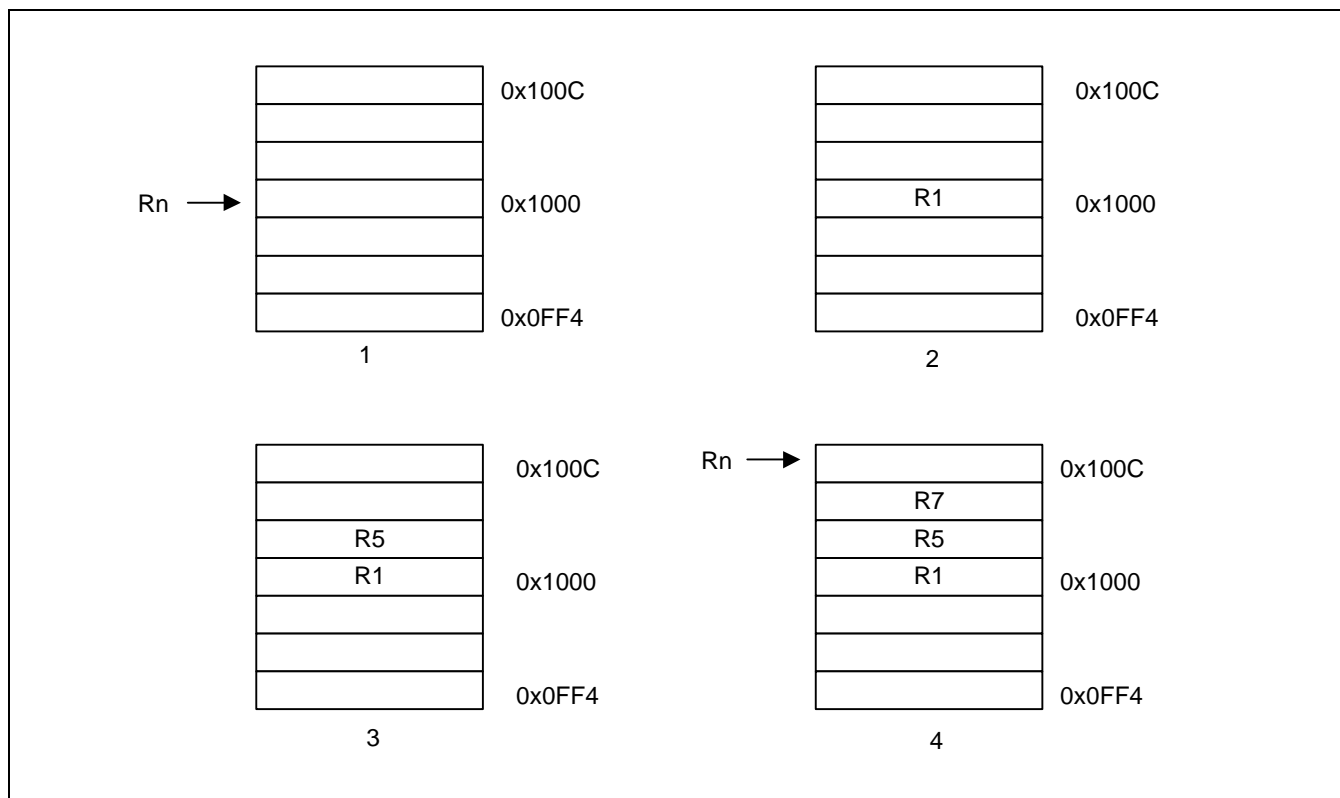


Figure 3-19. Post-Increment Addressing

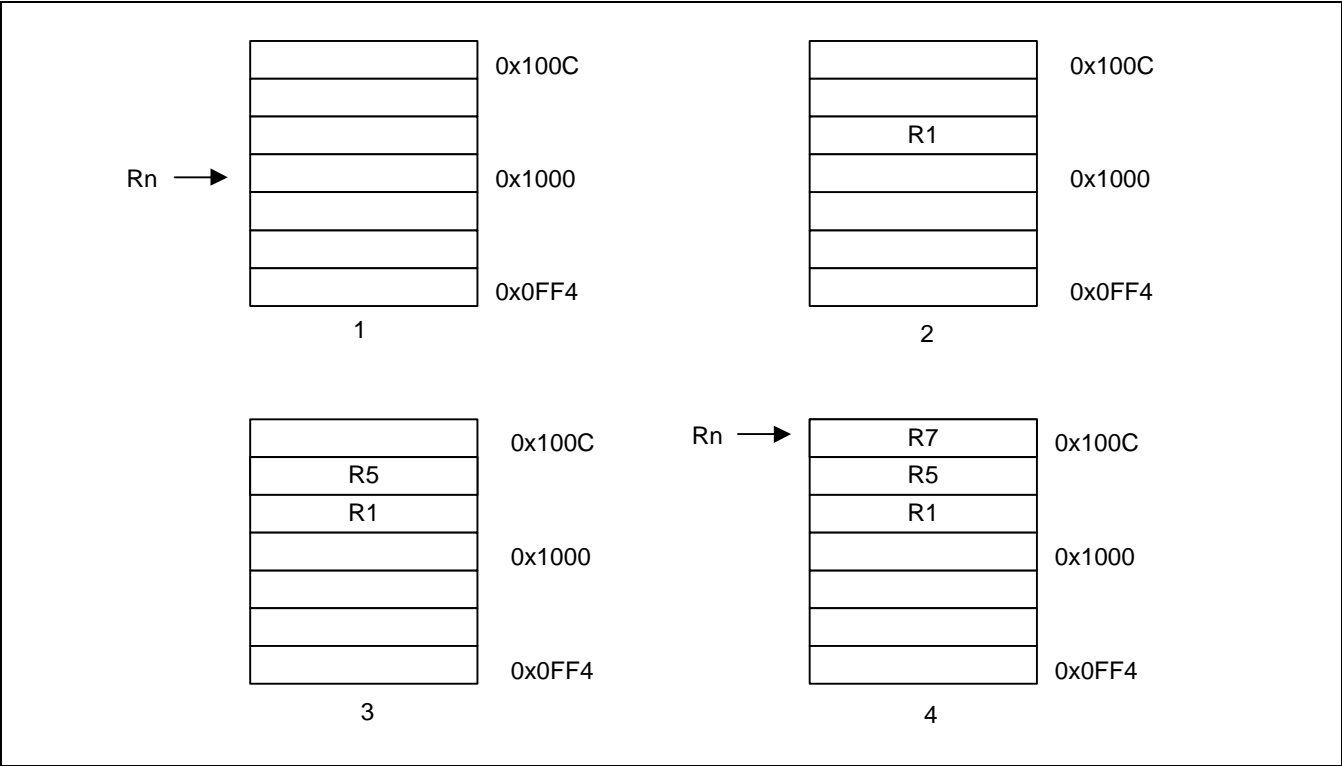


Figure 3-20. Pro-Increment Addressing

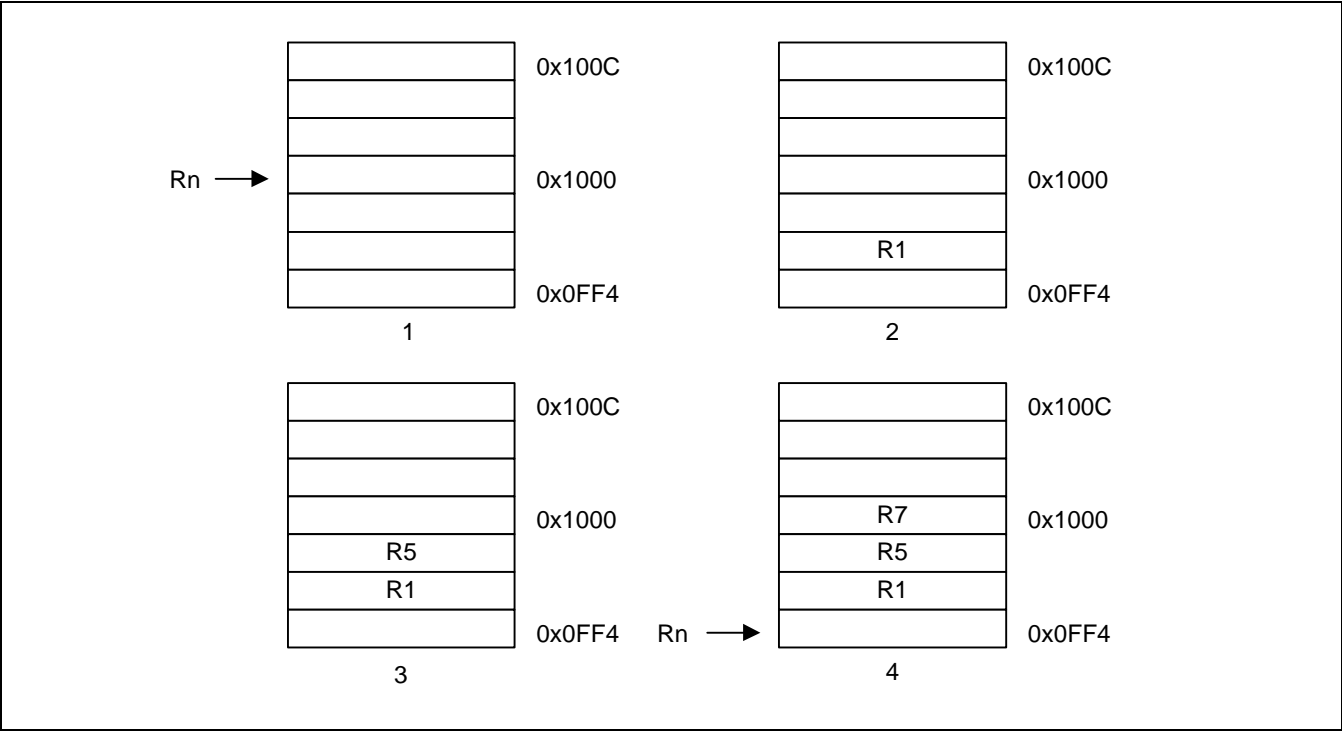


Figure 3-21. Post-Decrement Addressing

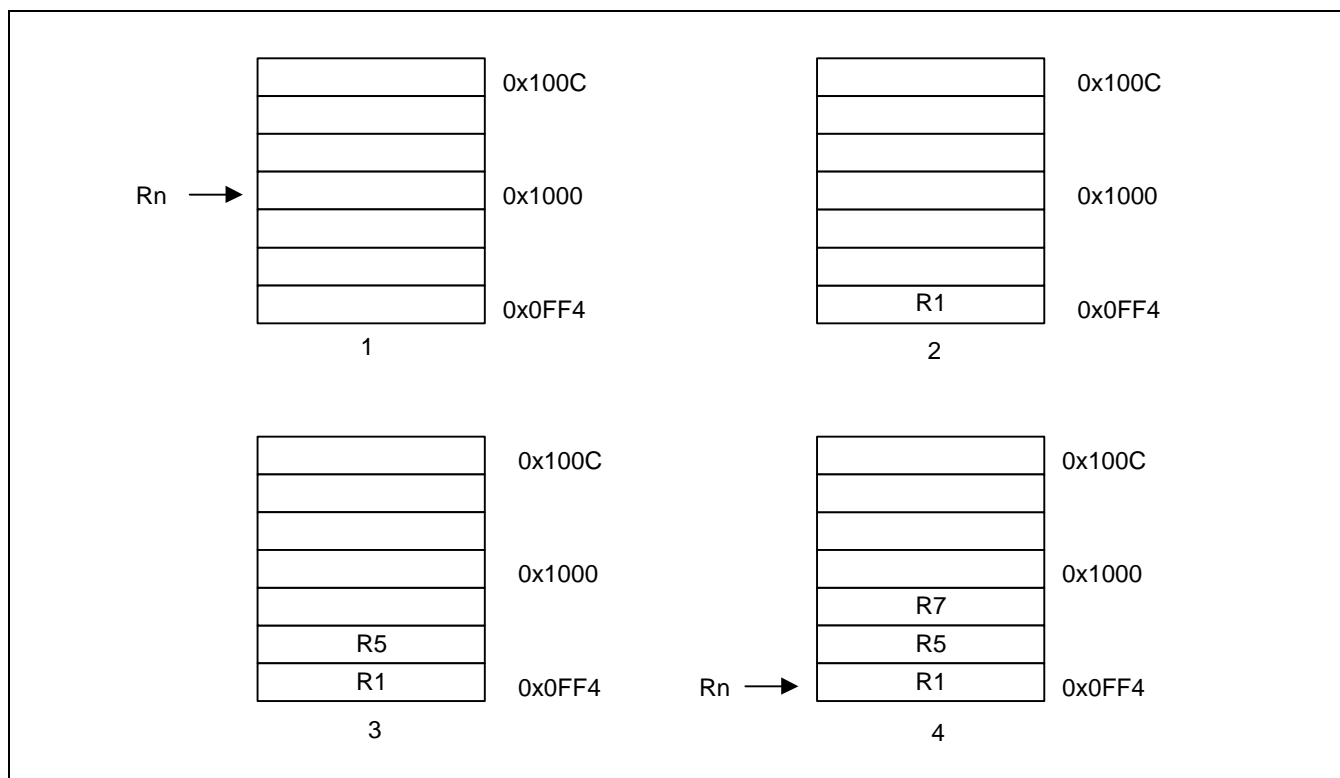


Figure 3-22. Pre-Decrement Addressing

USE OF THE S BIT

When the S bit is set in a LDM/STM instruction its meaning depends on whether or not R15 is in the transfer list and on the type of instruction. The S bit should only be set if the instruction is to execute in a privileged mode.

LDM with R15 in Transfer List and S Bit Set (Mode Changes)

If the instruction is a LDM then SPSR_<mode> is transferred to CPSR at the same time as R15 is loaded.

STM with R15 in Transfer List and S Bit Set (User Bank Transfer)

The registers transferred are taken from the user bank rather than the bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

R15 not in List and S Bit Set (User Bank Transfer)

For both LDM and STM instructions, the user bank registers are transferred rather than the register bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

When the instruction is LDM, care must be taken not to read from a banked register during the following cycle (inserting a dummy instruction such as MOV R0, R0 after the LDM will ensure safety).

USE OF R15 AS THE BASE

R15 should not be used as the base register in any LDM or STM instruction.

INCLUSION OF THE BASE IN THE REGISTER LIST

When write-back is specified, the base is written back at the end of the second cycle of the instruction. During a STM, the first register is written out at the start of the second cycle. A STM which includes storing the base, with the base as the first register to be stored, will therefore store the unchanged value, whereas with the base second or later in the transfer order, will store the modified value. A LDM will always overwrite the updated base if the base is in the list.

DATA ABORTS

Some legal addresses may be unacceptable to a memory management system, and the memory manager can indicate a problem with an address by taking the ABORT signal HIGH. This can happen on any transfer during a multiple register load or store, and must be recoverable if ARM7TDMI is to be used in a virtual memory system.

Aborts during STM Instructions

If the abort occurs during a store multiple instruction, ARM7TDMI takes little action until the instruction completes, whereupon it enters the data abort trap. The memory manager is responsible for preventing erroneous writes to the memory. The only change to the internal state of the processor will be the modification of the base register if write-back was specified, and this must be reversed by software (and the cause of the abort resolved) before the instruction may be retried.

Aborts during LDM Instructions

When ARM7TDMI detects a data abort during a load multiple instruction, it modifies the operation of the instruction to ensure that recovery is possible.

- Overwriting of registers stops when the abort happens. The aborting load will not take place but earlier ones may have overwritten registers. The PC is always the last register to be written and so will always be preserved.
- The base register is restored, to its modified value if write-back was requested. This ensures recoverability in the case where the base register is also in the transfer list, and may have been overwritten before the abort occurred.

The data abort trap is taken when the load multiple has completed, and the system software must undo any base modification (and resolve the cause of the abort) before restarting the instruction.

INSTRUCTION CYCLE TIMES

Normal LDM instructions take $nS + 1N + 1I$ and LDM PC takes $(n+1)S + 2N + 1I$ incremental cycles, where S,N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STM instructions take $(n-1)S + 2N$ incremental cycles to execute, where n is the number of words transferred.

ASSEMBLER SYNTAX

<LDM|STM>{cond}<FD|ED|FA|EA|IA|IB|DA|DB> Rn{!}, <Rlist>{^}

where:

{cond}	Two character condition mnemonic. See Table 3-2.
Rn	An expression evaluating to a valid register number
<Rlist>	A list of registers and register ranges enclosed in {} (e.g. {R0,R2-R7,R10}).
{!}	If present requests write-back (W = 1), otherwise W = 0.
{^}	If present set S bit to load the CPSR along with the PC, or force transfer of user bank when in privileged mode.

Addressing Mode Names

There are different assembler mnemonics for each of the addressing modes, depending on whether the instruction is being used to support stacks or for other purposes. The equivalence between the names and the values of the bits in the instruction are shown in the following table 3-6.

Table 3-6. Addressing Mode Names

Name	Stack	Other	L bit	P bit	U bit
Pre-increment load	LDMED	LDMIB	1	1	1
Post-increment load	LDMFD	LDMIA	1	0	1
Pre-decrement load	LDMEA	LDMDB	1	1	0
Post-decrement load	LDMFA	LDMDA	1	0	0
Pre-increment store	STMFA	STMIB	0	1	1
Post-increment store	STMEA	STMIA	0	0	1
Pre-decrement store	STMFD	STMDB	0	1	0
Post-decrement store	STMED	STMDA	0	0	0

FD, ED, FA, EA define pre/post indexing and the up/down bit by reference to the form of stack required. The F and E refer to a "full" or "empty" stack, i.e. whether a pre-index has to be done (full) before storing to the stack. The A and D refer to whether the stack is ascending or descending. If ascending, a STM will go up and LDM down, if descending, vice-versa.

IA, IB, DA, DB allow control when LDM/STM are not being used for stacks and simply mean increment after, increment before, decrement after, decrement before.

Examples

LDMFD	SP!,{R0,R1,R2}	; Unstack 3 registers.
STMIA	R0,{R0-R15}	; Save all registers.
LDMFD	SP!,{R15}	; R15 ← (SP), CPSR unchanged.
LDMFD	SP!,{R15}^	; R15 ← (SP), CPSR ← SPSR_mode
		; (allowed only in privileged modes).
STMFD	R13,{R0-R14}^	; Save user mode regs on stack
		; (allowed only in privileged modes).

These instructions may be used to save state on subroutine entry, and restore it efficiently on return to the calling routine:

STMED	SP!,{R0-R3,R14}	; Save R0 to R3 to use as workspace
		; and R14 for returning.
BL	somewhere	; This nested call will overwrite R14
LDMED	SP!,{R0-R3,R15}	; Restore workspace and return.

SINGLE DATA SWAP (SWP)

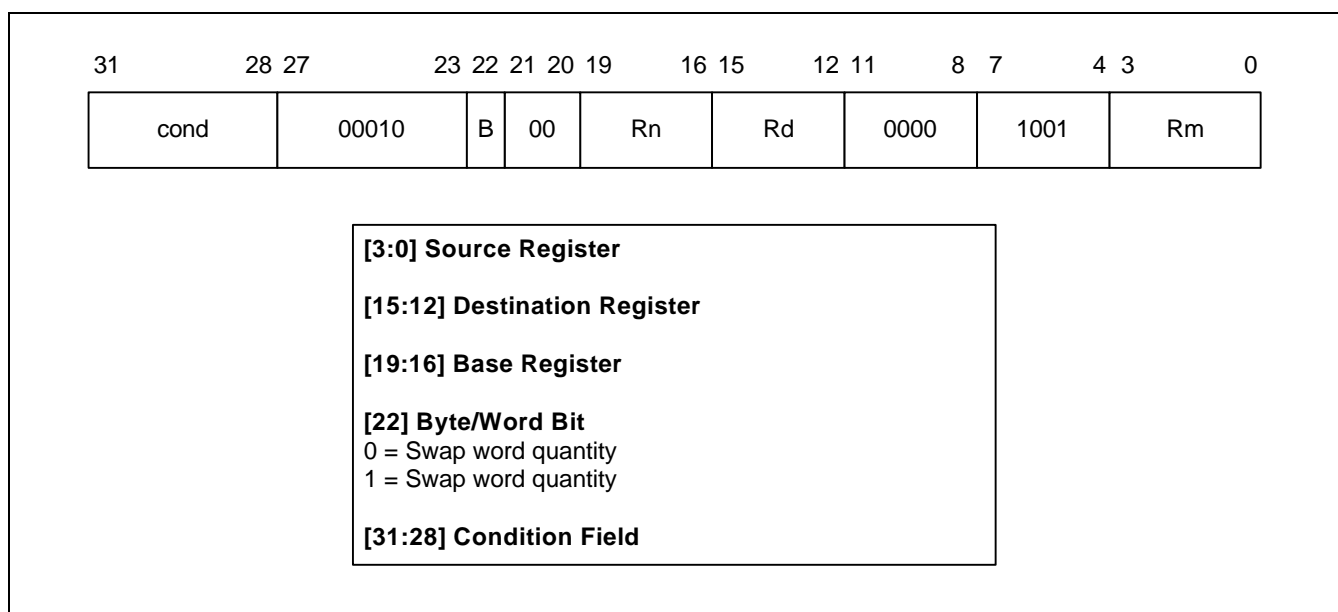


Figure 3-23. Swap Instruction

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-23.

The data swap instruction is used to swap a byte or word quantity between a register and external memory. This instruction is implemented as a memory read followed by a memory write which are "locked" together (the processor cannot be interrupted until both operations have completed, and the memory manager is warned to treat them as inseparable). This class of instruction is particularly useful for implementing software semaphores.

The swap address is determined by the contents of the base register (Rn). The processor first reads the contents of the swap address. Then it writes the contents of the source register (Rm) to the swap address, and stores the old memory contents in the destination register (Rd). The same register may be specified as both the source and destination.

The LOCK output goes HIGH for the duration of the read and write operations to signal to the external memory manager that they are locked together, and should be allowed to complete without interruption. This is important in multi-processor systems where the swap instruction is the only indivisible instruction which may be used to implement semaphores; control of the memory must not be removed from a processor while it is performing a locked operation.

BYTES AND WORDS

This instruction class may be used to swap a byte (B = 1) or a word (B = 0) between an ARM7TDMI register and memory. The SWP instruction is implemented as a LDR followed by a STR and the action of these is as described in the section on single data transfers. In particular, the description of Big and little Endian configuration applies to the SWP instruction.

USE OF R15

Do not use R15 as an operand (Rd, Rn or Rs) in a SWP instruction.

DATA ABORTS

If the address used for the swap is unacceptable to a memory management system, the memory manager can flag the problem by driving ABORT HIGH. This can happen on either the read or the write cycle (or both), and in either case, the data abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Swap instructions take $1S + 2N + 1I$ incremental cycles to execute, where S, N and I are defined as sequential (S-cycle), non-sequential, and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

<SWP>{cond}{B} Rd,Rm,[Rn]

{cond} Two-character condition mnemonic. See Table 3-2.

{B} If B is present then byte transfer, otherwise word transfer

Rd,Rm,Rn Expressions evaluating to valid register numbers

Examples

SWP	R0,R1,[R2]	; Load R0 with the word addressed by R2, and ; store R1 at R2.
SWPB	R2,R3,[R4]	; Load R2 with the byte addressed by R4, and ; store bits 0 to 7 of R3 at R4.
SWPEQ	R0,R0,[R1]	; Conditionally swap the contents of the ; word addressed by R1 with R0.

SOFTWARE INTERRUPT (SWI)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-24, below.

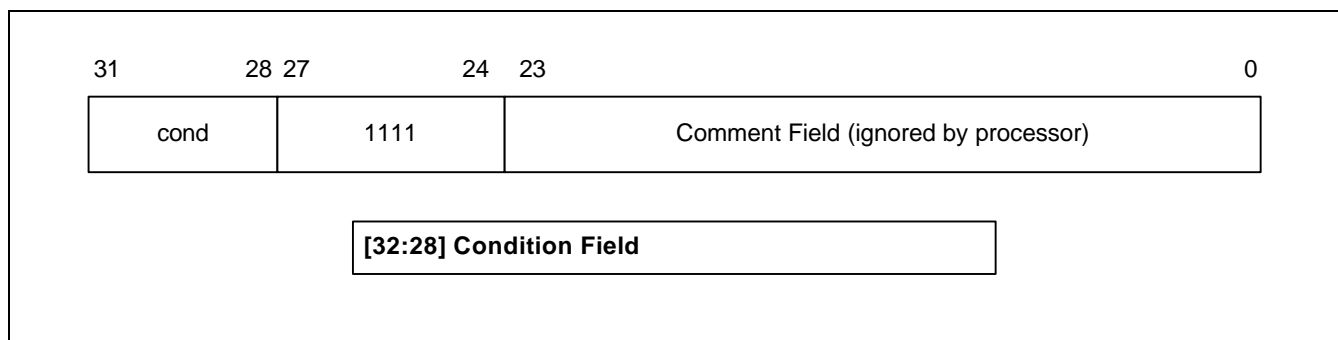


Figure 3-24. Software Interrupt Instruction

The software interrupt instruction is used to enter supervisor mode in a controlled manner. The instruction causes the software interrupt trap to be taken, which effects the mode change. The PC is then forced to a fixed value (0x08) and the CPSR is saved in SPSR_svc. If the SWI vector address is suitably protected (by external memory management hardware) from modification by the user, a fully protected operating system may be constructed.

RETURN FROM THE SUPERVISOR

The PC is saved in R14_svc upon entering the software interrupt trap, with the PC adjusted to point to the word after the SWI instruction. MOVS PC,R14_svc will return to the calling program and restore the CPSR.

Note that the link mechanism is not re-entrant, so if the supervisor code wishes to use software interrupts within itself it must first save a copy of the return address and SPSR.

COMMENT FIELD

The bottom 24 bits of the instruction are ignored by the processor, and may be used to communicate information to the supervisor code. For instance, the supervisor may look at this field and use it to index into an array of entry points for routines which perform the various supervisor functions.

INSTRUCTION CYCLE TIMES

Software interrupt instructions take $2S + 1N$ incremental cycles to execute, where S and N are defined as sequential (S-cycle) and non-sequential (N-cycle).

ASSEMBLER SYNTAX

SWI{cond} <expression>

{cond} Two character condition mnemonic, Table 3-2.

<expression> Evaluated and placed in the comment field (which is ignored by ARM7TDMI).

Examples

```

SWI      ReadC           ; Get next character from read stream.
SWI      Writel+"k"      ; Output a "k" to the write stream.
SWINE    0               ; Conditionally call supervisor with 0 in comment field.

```

Supervisor code

The previous examples assume that suitable supervisor code exists, for instance:

```

0x08 B Supervisor           ; SWI entry point
EntryTable                  ; Addresses of supervisor routines
DCD ZeroRtn
DCD ReadCRtn
DCD WritelRtn
...
Zero      EQU 0
ReadC     EQU 256
Writel    EQU 512

Supervisor                  ; SWI has routine required in bits 8-23 and data (if any) in
                           ; bits 0-7. Assumes R13_svc points to a suitable stack
STMFD     R13,{R0-R2,R14}   ; Save work registers and return address.
LDR       R0,[R14,#-4]      ; Get SWI instruction.
BIC       R0,R0,#0xFF000000 ; Clear top 8 bits.
MOV       R1,R0,LSR#8       ; Get routine offset.
ADR       R2,EntryTable     ; Get start address of entry table.
LDR       R15,[R2,R1,LSL#2] ; Branch to appropriate routine.
WritelRtn                  ; Enter with character in R0 bits 0-7.
...
LDMFD     R13,{R0-R2,R15}^  ; Restore workspace and return,
                           ; restoring processor mode and flags.

```

COPROCESSOR DATA OPERATIONS (CDP)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-25.

This class of instruction is used to tell a coprocessor to perform some internal operation. No result is communicated back to ARM7TDMI, and it will not wait for the operation to complete. The coprocessor could contain a queue of such instructions awaiting execution, and their execution can overlap other activity, allowing the coprocessor and ARM7TDMI to perform independent tasks in parallel.

COPROCESSOR INSTRUCTIONS

The KS32C65100, unlike some other ARM-based processors, does not have an external coprocessor interface. It does not have a on-chip coprocessor also.

So then all coprocessor instructions will cause the undefined instruction trap to be taken on the KS32C65100. These coprocessor instructions can be emulated by the undefined trap handler. Even though external coprocessor can not be connected to the KS32C65100, the coprocessor instructions are still described here in full for completeness. (Remember that any external coprocessor described in this section is a software emulation.)

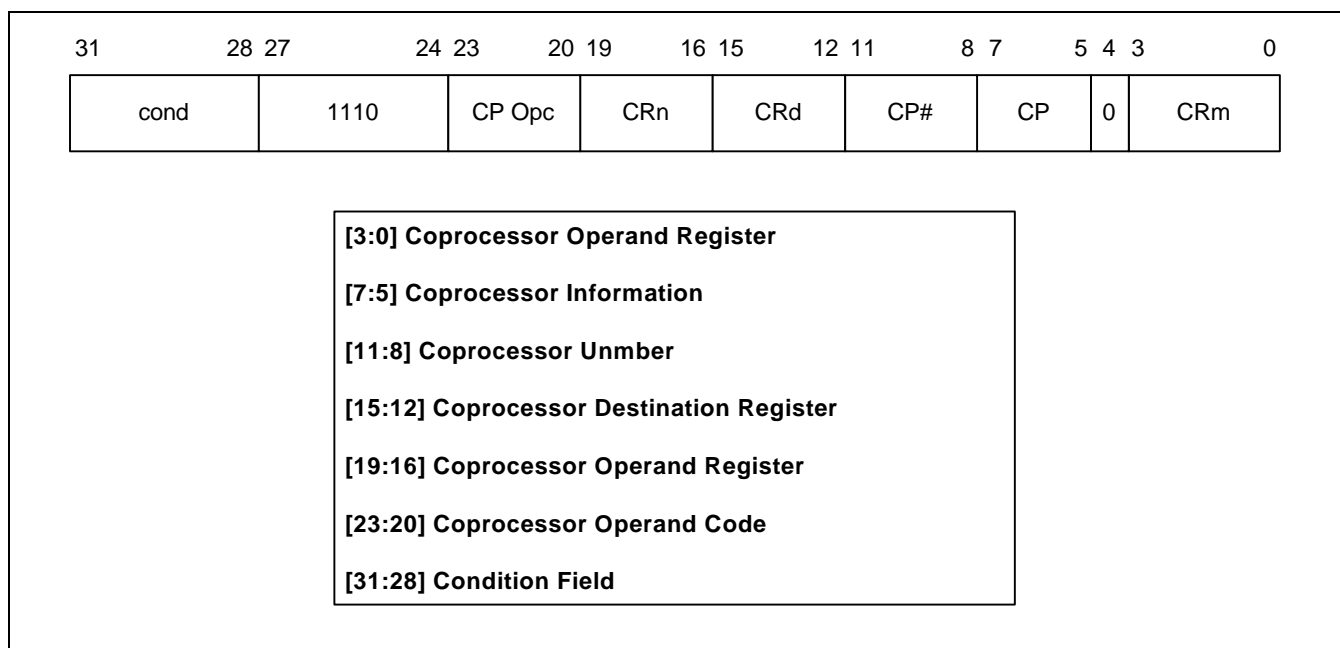


Figure 3-25. Coprocessor Data Operation Instruction

THE COPROCESSOR FIELDS

Only bit 4 and bits 24 to 31 are significant to ARM7TDMI. The remaining bits are used by coprocessors. The above field names are used by convention, and particular coprocessors may redefine the use of all fields except CP# as appropriate. The CP# field is used to contain an identifying number (in the range 0 to 15) for each coprocessor, and a coprocessor will ignore any instruction which does not contain its number in the CP# field.

The conventional interpretation of the instruction is that the coprocessor should perform an operation specified in the CP Opc field (and possibly in the CP field) on the contents of CRn and CRm, and place the result in CRd.

INSTRUCTION CYCLE TIMES

Coprocessor data operations take 1S + bI incremental cycles to execute, where *b* is the number of cycles spent in the coprocessor busy-wait loop.

S and I are defined as sequential (S-cycle) and internal (I-cycle).

ASSEMBLER SYNTAX

CDP{cond} p#,<expression1>,cd,cn,cm{,<expression2>}

{cond}	Two character condition mnemonic. See Table 3-2.
p#	The unique number of the required coprocessor
<expression1>	Evaluated to a constant and placed in the CP Opc field
cd, cn and cm	Evaluate to the valid coprocessor register numbers CRd, CRn and CRm respectively
<expression2>	Where present is evaluated to a constant and placed in the CP field

Examples

CDP	p1,10,c1,c2,c3	; Request coproc 1 to do operation 10
		; on CR2 and CR3, and put the result in CR1.
CDPEQ	p2,5,c1,c2,c3,2	; If Z flag is set request coproc 2 to do operation 5 (type 2)
		; on CR2 and CR3, and put the result in CR1.

COPROCESSOR DATA TRANSFERS (LDC, STC)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-26. This class of instruction is used to load (LDC) or store (STC) a subset of a coprocessor's registers directly to memory. ARM7TDMI is responsible for supplying the memory address, and the coprocessor supplies or accepts the data and controls the number of words transferred.

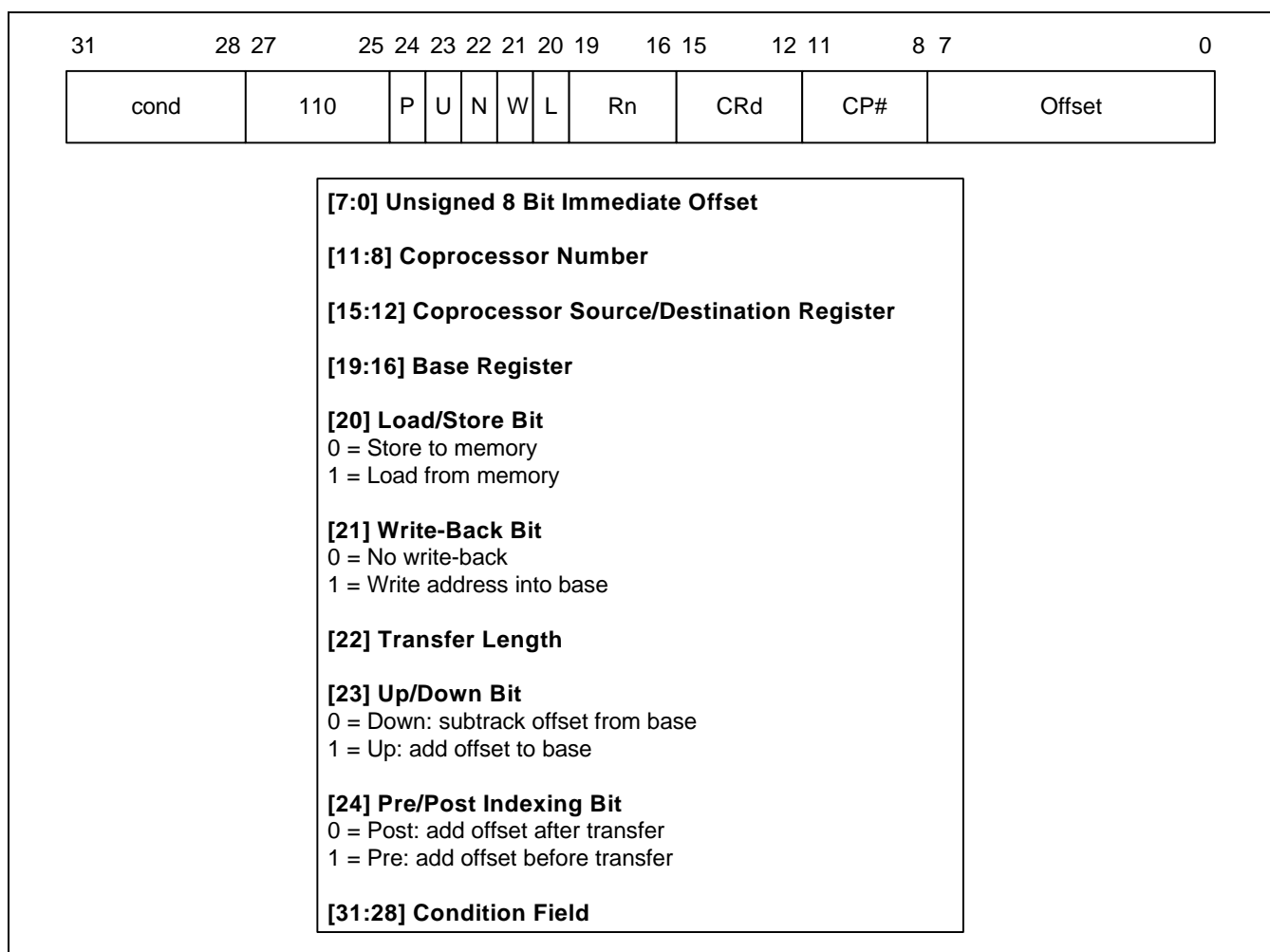


Figure 3-26. Coprocessor Data Transfer Instructions

THE COPROCESSOR FIELDS

The CP# field is used to identify the coprocessor which is required to supply or accept the data, and a coprocessor will only respond if its number matches the contents of this field.

The CRd field and the N bit contain information for the coprocessor which may be interpreted in different ways by different coprocessors, but by convention CRd is the register to be transferred (or the first register where more than one is to be transferred), and the N bit is used to choose one of two transfer length options. For instance N = 0 could select the transfer of a single register, and N = 1 could select the transfer of all the registers for context switching.

ADDRESSING MODES

ARM7TDMI is responsible for providing the address used by the memory system for the transfer, and the addressing modes available are a subset of those used in single data transfer instructions. Note, however, that the immediate offsets are 8 bits wide and specify word offsets for coprocessor data transfers, whereas they are 12 bits wide and specify byte offsets for single data transfers.

The 8 bit unsigned immediate offset is shifted left 2 bits and either added to ($U = 1$) or subtracted from ($U = 0$) the base register (R_n); this calculation may be performed either before ($P = 1$) or after ($P = 0$) the base is used as the transfer address. The modified base value may be overwritten back into the base register (if $W = 1$), or the old value of the base may be preserved ($W = 0$). Note that post-indexed addressing modes require explicit setting of the W bit, unlike LDR and STR which always write-back when post-indexed.

The value of the base register, modified by the offset in a pre-indexed instruction, is used as the address for the transfer of the first word. The second word (if more than one is transferred) will go to or come from an address one word (4 bytes) higher than the first transfer, and the address will be incremented by one word for each subsequent transfer.

ADDRESS ALIGNMENT

The base address should normally be a word aligned quantity. The bottom 2 bits of the address will appear on $A[1:0]$ and might be interpreted by the memory system.

USE OF R15

If R_n is R15, the value used will be the address of the instruction plus 8 bytes. Base write-back to R15 must not be specified.

DATA ABORTS

If the address is legal but the memory manager generates an abort, the data trap will be taken. The write-back of the modified base will take place, but all other processor state will be preserved. The coprocessor is partly responsible for ensuring that the data transfer can be restarted after the cause of the abort has been resolved, and must ensure that any subsequent actions it undertakes can be repeated when the instruction is retried.

INSTRUCTION CYCLE TIMES

Coprocessor data transfer instructions take $(n-1)S + 2N + B$ incremental cycles to execute, where:

n : The number of words transferred.

B : The number of cycles spent in the coprocessor busy-wait loop.

S , N and I are defined as sequential (S -cycle), non-sequential (N -cycle), and internal (I -cycle), respectively.

ASSEMBLER SYNTAX

<LDC|STC>{cond}{L} p#,cd,<Address>

LDC	Load from memory to coprocessor
STC	Store from coprocessor to memory
{L}	When present perform long transfer (N=1), otherwise perform short transfer (N=0)
{cond}	Two character condition mnemonic. See Table 3-2..
p#	The unique number of the required coprocessor
cd	An expression evaluating to a valid coprocessor register number that is placed in the CRd field

<Address> can be:

- 1 An expression which generates an address:
The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated
- 2 A pre-indexed addressing specification:

[Rn]	offset of zero
[Rn,<#expression>]{!}	offset of <expression> bytes
- 3 A post-indexed addressing specification:

[Rn],<#expression >	offset of <expression> bytes
{!}	write back the base register (set the W bit) if! is present
Rn	is an expression evaluating to a valid ARM7TDMI register number.

NOTE

If Rn is R15, the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining.

Examples

LDC	p1,c2,table	; Load c2 of coproc 1 from address
		; table, using a PC relative address.
STCEQL	p2,c3,[R5,#24]!	; Conditionally store c3 of coproc 2
		; into an address 24 bytes up from R5,
		; write this address back to R5, and use
		; long transfer option (probably to store multiple words).

NOTE

Although the address offset is expressed in bytes, the instruction offset field is in words. The assembler will adjust the offset appropriately.

COPROCESSOR REGISTER TRANSFERS (MRC, MCR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.. The instruction encoding is shown in Figure 3-27.

This class of instruction is used to communicate information directly between ARM7TDMI and a coprocessor. An example of a coprocessor to ARM7TDMI register transfer (MRC) instruction would be a FIX of a floating point value held in a coprocessor, where the floating point number is converted into a 32 bit integer within the coprocessor, and the result is then transferred to ARM7TDMI register. A FLOAT of a 32 bit value in ARM7TDMI register into a floating point value within the coprocessor illustrates the use of ARM7TDMI register to coprocessor transfer (MCR).

An important use of this instruction is to communicate control information directly from the coprocessor into the ARM7TDMI CPSR flags. As an example, the result of a comparison of two floating point values within a coprocessor can be moved to the CPSR to control the subsequent flow of execution.

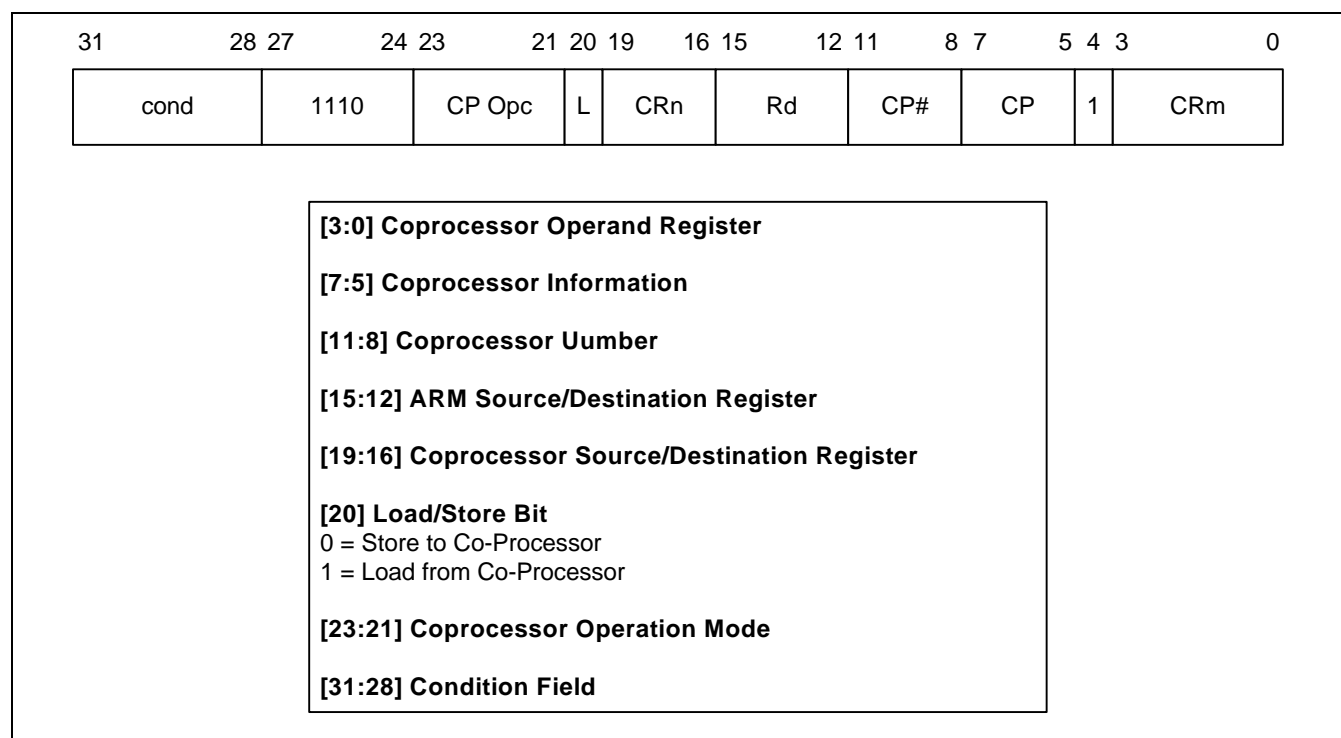


Figure 3-27. Coprocessor Register Transfer Instructions

THE COPROCESSOR FIELDS

The CP# field is used, as for all coprocessor instructions, to specify which coprocessor is being called upon.

The CP Opc, CRn, CP and CRm fields are used only by the coprocessor, and the interpretation presented here is derived from convention only. Other interpretations are allowed where the coprocessor functionality is incompatible with this one. The conventional interpretation is that the CP Opc and CP fields specify the operation the coprocessor is required to perform, CRn is the coprocessor register which is the source or destination of the transferred information, and CRm is a second coprocessor register which may be involved in some way which depends on the particular operation specified.

TRANSFERS TO R15

When a coprocessor register transfer to ARM7TDMI has R15 as the destination, bits 31, 30, 29 and 28 of the transferred word are copied into the N, Z, C and V flags respectively. The other bits of the transferred word are ignored, and the PC and other CPSR bits are unaffected by the transfer.

TRANSFERS FROM R15

A coprocessor register transfer from ARM7TDMI with R15 as the source register will store the PC+12.

INSTRUCTION CYCLE TIMES

MRC instructions take $1S + (b+1)I + 1C$ incremental cycles to execute, where S, I and C are defined as sequential (S-cycle), internal (I-cycle), and coprocessor register transfer (C-cycle), respectively. MCR instructions take $1S + bI + 1C$ incremental cycles to execute, where b is the number of cycles spent in the coprocessor busy-wait loop.

ASSEMBLER SYNTAX

<MCR|MRC>{cond} p#,<expression1>,Rd,cn,cm{,<expression2>}

MRC Move from coprocessor to ARM7TDMI register (L=1)

MCR Move from ARM7TDMI register to coprocessor (L=0)
{cond} Two character condition mnemonic. See Table 3-2

p# The unique number of the required coprocessor

<expression1> Evaluated to a constant and placed in the CP Opc field

Rd An expression evaluating to a valid ARM7TDMI register number
cn and cm Expressions evaluating to the valid coprocessor register numbers CRn and CRm respectively

<expression2> Where present is evaluated to a constant and placed in the CP field

Examples

MRC	p2,5,R3,c5,c6	; Request coproc 2 to perform operation 5
		; on c5 and c6, and transfer the (single
		; 32-bit word) result back to R3.
MCR	p6,0,R4,c5,c6	; Request coproc 6 to perform operation 0
		; on R4 and place the result in c6.
MRCEQ	p3,9,R3,c5,c6,2	; Conditionally request coproc 3 to
		; perform operation 9 (type 2) on c5 and
		; c6, and transfer the result back to R3.

UNDEFINED INSTRUCTION

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction format is shown in Figure 3-28.



Figure 3-28. Undefined Instruction

If the condition is true, the undefined instruction trap will be taken.

Note that the undefined instruction mechanism involves offering this instruction to any coprocessors which may be present, and all coprocessors must refuse to accept it by driving CPA and CPB HIGH.

INSTRUCTION CYCLE TIMES

This instruction takes 2S + 1I + 1N cycles, where S, N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle).

ASSEMBLER SYNTAX

The assembler has no mnemonics for generating this instruction. If it is adopted in the future for some specified use, suitable mnemonics will be added to the assembler. Until such time, this instruction must not be used.

INSTRUCTION SET EXAMPLES

The following examples show ways in which the basic ARM7TDMI instructions can combine to give efficient code. None of these methods saves a great deal of execution time (although they may save some), mostly they just save code.

USING THE CONDITIONAL INSTRUCTIONS

Using Conditionals for Logical OR

CMP	Rn,#p		
BEQ	Label		
CMP	Rm,#q		
BEQ	Label		

; If Rn=p OR Rm=q THEN GOTO Label.

This can be replaced by

CMP	Rn,#p		
CMPNE	Rm,#q		
BEQ	Label		

; If condition not satisfied try other test.

Absolute Value

TEQ	Rn,#0		
RSBMI	Rn,Rn,#0		

; Test sign
; and 2's complement if necessary.

Multiplication by 4, 5 or 6 (Run Time)

MOV	Rc,Ra,LSL#2		
CMP	Rb,#5		
ADDCS	Rc,Rc,Ra		
ADDHI	Rc,Rc,Ra		

; Multiply by 4,
; Test value,
; Complete multiply by 5,
; Complete multiply by 6.

Combining Discrete and Range Tests

TEQ	Rc,#127		
CMPNE	Rc,#"-1		
MOVLS	Rc,#"		

; Discrete test,
; Range test
; IF Rc ≤ "-1 OR Rc = ASCII(127)
; THEN Rc = ".

Division and Remainder

A number of divide routines for specific applications are provided in source form as part of the ANSI C library provided with the ARM cross development Toolkit, available from your supplier. A short general purpose divide routine follows.

	MOV	Rcnt,#1	; Enter with numbers in Ra and Rb.
			; Bit to control the division.
Div1	CMP	Rb,#0x80000000	; Move Rb until greater than Ra.
	CMPCC	Rb,Ra	
	MOVCC	Rb,Rb,ASL#1	
	MOVCC	Rcnt,Rcnt,ASL#1	
	BCC	Div1	
	MOV	Rc,#0	
Div2	CMP	Ra,Rb	; Test for possible subtraction.
	SUBCS	Ra,Ra,Rb	; Subtract if ok,
	ADDCS	Rc,Rc,Rcnt	; Put relevant bit into result
	MOVS	Rcnt,Rcnt,LSR#1	; Shift control bit
	MOVNE	Rb,Rb,LSR#1	; Halve unless finished.
	BNE	Div2	; Divide result in Rc, remainder in Ra.

Overflow Detection in the ARM7TDMI

1. Overflow in unsigned multiply with a 32-bit result

UMULL	Rd,Rt,Rm,Rn	; 3 to 6 cycles
TEQ	Rt,#0	; +1 cycle and a register
BNE	overflow	

2. Overflow in signed multiply with a 32-bit result

SMULL	Rd,Rt,Rm,Rn	; 3 to 6 cycles
TEQ	Rt,Rd ASR#31	; +1 cycle and a register
BNE	overflow	

3. Overflow in unsigned multiply accumulate with a 32 bit result

UMLAL	Rd,Rt,Rm,Rn	; 4 to 7 cycles
TEQ	Rt,#0	; +1 cycle and a register
BNE	overflow	

4. Overflow in signed multiply accumulate with a 32 bit result

SMLAL	Rd,Rt,Rm,Rn	; 4 to 7 cycles
TEQ	Rt,Rd, ASR#31	; +1 cycle and a register
BNE	overflow	

5. Overflow in unsigned multiply accumulate with a 64 bit result

UMULL	RI,Rh,Rm,Rn	; 3 to 6 cycles
ADDS	RI,RI,Ra1	; Lower accumulate
ADC	Rh,Rh,Ra2	; Upper accumulate
BCS	overflow	; 1 cycle and 2 registers

6. Overflow in signed multiply accumulate with a 64 bit result

SMULL	RI,Rh,Rm,Rn	; 3 to 6 cycles
ADDS	RI,RI,Ra1	; Lower accumulate
ADC	Rh,Rh,Ra2	; Upper accumulate
BVS	overflow	; 1 cycle and 2 registers

NOTE

Overflow checking is not applicable to unsigned and signed multiplies with a 64-bit result, since overflow does not occur in such calculations.

PSEUDO-RANDOM BINARY SEQUENCE GENERATOR

It is often necessary to generate (pseudo-) random numbers and the most efficient algorithms are based on shift generators with exclusive-OR feedback rather like a cyclic redundancy check generator. Unfortunately the sequence of a 32 bit generator needs more than one feedback tap to be maximal length (i.e. $2^{32}-1$ cycles before repetition), so this example uses a 33 bit register with taps at bits 33 and 20. The basic algorithm is newbit: = bit 33 or bit 20, shift left the 33 bit number and put in newbit at the bottom; this operation is performed for all the newbits needed (i.e. 32 bits). The entire operation can be done in 5 S cycles:

		; Enter with seed in Ra (32 bits),
		; Rb (1 bit in Rb lsb), uses Rc.
TST	Rb,Rb,LSR#1	; Top bit into carry
MOVS	Rc,Ra,RRX	; 33 bit rotate right
ADC	Rb,Rb,Rb	; Carry into lsb of Rb
EOR	Rc,Rc,Ra,LSL#12	; (involved!)
EOR	Ra,Rc,Rc,LSR#20	; (similarly involved!) new seed in Ra, Rb as before

MULTIPLICATION BY CONSTANT USING THE BARREL SHIFTER**Multiplication by 2^n (1,2,4,8,16,32..)**

MOV Ra, Rb, LSL #n

Multiplication by 2^{n+1} (3,5,9,17..)

ADD Ra,Ra,Ra,LSL #n

Multiplication by 2^{n-1} (3,7,15..)

RSB Ra,Ra,Ra,LSL #n

Multiplication by 6

```

ADD    Ra,Ra,Ra,LSL #1    ; Multiply by 3
MOV    Ra,Ra,LSL#1        ; and then by 2

```

Multiply by 10 and add in extra number

```

ADD    Ra,Ra,Ra,LSL#2      ; Multiply by 5
ADD    Ra,Rc,Ra,LSL#1      ; Multiply by 2 and add in next digit

```

General recursive method for $Rb := Ra * C$, C a constant:

1. If C even, say $C = 2^n * D$, D odd:

```

D=1:      MOV  Rb,Ra,LSL #n
D<>1:     {Rb := Ra*D}
MOV       Rb,Rb,LSL #n

```

2. If $C \bmod 4 = 1$, say $C = 2^n * D + 1$, D odd, $n > 1$:

```

D=1:      ADD  Rb,Ra,Ra,LSL #n
D<>1:     {Rb := Ra*D}
ADD       Rb,Ra,Rb,LSL #n

```

3. If $C \bmod 4 = 3$, say $C = 2^n * D - 1$, D odd, $n > 1$:

```

D=1:      RSB  Rb,Ra,Ra,LSL #n
D<>1:     {Rb := Ra*D}
RSB       Rb,Ra,Rb,LSL #n

```

This is not quite optimal, but close. An example of its non-optimality is multiply by 45 which is done by:

```

RSB      Rb,Ra,Ra,LSL#2    ; Multiply by 3
RSB      Rb,Ra,Rb,LSL#2    ; Multiply by  $4*3-1 = 11$ 
ADD      Rb,Ra,Rb,LSL# 2   ; Multiply by  $4*11+1 = 45$ 

```

rather than by:

```

ADD      Rb,Ra,Ra,LSL#3    ; Multiply by 9
ADD      Rb,Rb,Rb,LSL#2    ; Multiply by  $5*9 = 45$ 

```

LOADING A WORD FROM AN UNKNOWN ALIGNMENT

BIC	Rb,Ra,#3	; Enter with address in Ra (32 bits) uses
LDMIA	Rb,{Rd,Rc}	; Rb, Rc result in Rd. Note d must be less than c e.g. 0,1
AND	Rb,Ra,#3	; Get word aligned address
MOVS	Rb,Rb,LSL#3	; Get 64 bits containing answer
MOVNE	Rd,Rd,LSR Rb	; Correction factor in bytes
RSBNE	Rb,Rb,#32	; ...now in bits and test if aligned
ORRNE	Rd,Rd,Rc,LSL Rb	; Produce bottom of result word (if not aligned)
		; Get other shift amount
		; Combine two halves to get result

THUMB INSTRUCTION SET FORMAT

The thumb instruction sets are 16-bit versions of ARM instruction sets (32-bit format). The ARM instructions are reduced to 16-bit versions, Thumb instructions, at the cost of versatile functions of the ARM instruction sets. The thumb instructions are decompressed to the ARM instructions by the Thumb decompressor inside the ARM7TDMI core.

As the Thumb instructions are compressed ARM instructions, the Thumb instructions have the 16-bit format instructions and have some restrictions. The restrictions by 16-bit format is fully notified for using the Thumb instructions.

FORMAT SUMMARY

The THUMB instruction set formats are shown in the following figure.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	Op		Offset					Rs		Rd			Move shifted register	
2	0	0	0	1	1	I	op	Rn/offset8			Rs		Rd			Add/subtract	
3	0	0	1	Op		Rd		Offset8									Move/compare/add
4	0	1	0	0	0	0	Op			Rs		Rd			/subtract immediate		
5	0	1	0	0	0	1	Op	H ₁	H ₁	Rs/Hs		Rd/Hd			ALU operations		
6	0	1	0	0	1	Rd		Word8									Hi register operations
7	0	1	0	1	L	B	0	Ro			Rb		Rd		/branch exchange		
8	0	1	0	1	H	S	1	Ro			Rb		Rd		PC-relative load		
9	0	1	1	B	L	Offset5					Rb		Rd		Load/store with register offset		
10	1	0	0	0	L	Offset5					Rb		Rd		Load/store sign-extended byte/halfword		
11	1	0	0	1	L	Rd		Word8									Load/store with immediate offset
12	1	0	1	0	S _p	Rd		Word8									Load/store halfword
13	1	0	1	1	0	0	0	0	S	SWord7							SP-relative load/store
14	1	0	1	1	L	1	0	R	Rlist								Load address
15	1	1	0	0	L	Rb		Rlist									Add offset to stack pointer
16	1	1	0	1	cond					Softset8							Push/pop registers
17	1	1	0	1	1	1	1	1	Value8								Multiple load/store
18	1	1	1	0	0	Offset11											Conditional branch
19	1	1	1	1	H	Offset											Software interrupt
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Long branch with link

Figure 3-29. THUMB Instruction Set Formats

OPCODE SUMMARY

The following table summaries the THUMB instruction set. For further information about a particular instruction please refer to the sections listed in the right-most column.

Table 3-7. THUMB Instruction Set Opcodes

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
ADC	Add with carry	4	-	4
ADD	Add	4	4	4 ⁽¹⁾
AND	AND	4	-	4
ASR	Arithmetic shift right	4	-	4
B	Unconditional branch	4	-	-
Bxx	Conditional branch	4	-	-
BIC	Bit clear	4	-	4
BL	Branch and link	-	-	-
BX	Branch and exchange	4	4	-
CMN	Compare negative	4	-	4
CMP	Compare	4	4	4
EOR	EOR	4	-	4
LDMIA	Load multiple	4	-	-
LDR	Load word	4	-	-
LDRB	Load byte	4	-	-
LDRH	Load halfword	4	-	-
LSL	Logical shift left	4	-	4
LDSB	Load sign-extended byte	4	-	-
LDSH	Load sign-extended halfword	4	-	--
LSR	Logical shift right	4	-	44
MOV	Move register	4	-	4 ⁽²⁾
MUL	Multiply	4	4	4
MVN	Move negative register	4	-	4
NEG	Negate	4	-	4
ORR	OR	4	-	4
POP	Pop registers	4	-	-
PUSH	Push registers	4	-	-
ROR	Rotate right	4	-	4
SBC	Subtract with carry	4	-	4
STMIA	Store multiple	4	-	-

Table 3-7. THUMB Instruction Set Opcodes (Continued)

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
STR	Store word	4	-	-
STRB	Store byte	4	-	-
STRH	Store half-word	4	-	-
SWI	Software interrupt	-	-	-
SUB	Subtract	4	-	4
TST	Test bits	4	-	4

NOTES:

1. The condition codes are unaffected by the format 5, 12 and 13 versions of this instruction.
2. The condition codes are unaffected by the format 5 version of this instruction.

FORMAT 1: MOVE SHIFTED REGISTER

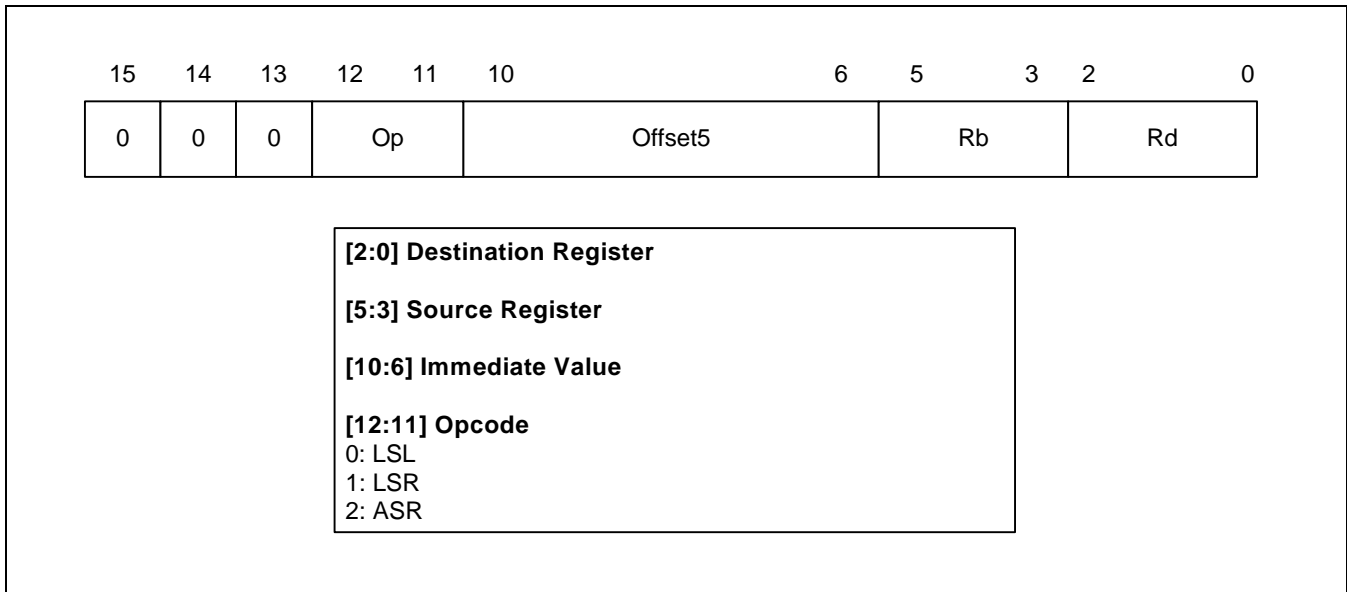


Figure 3-30. Format 1

OPERATION

These instructions move a shifted value between Lo registers. The THUMB assembler syntax is shown in Table 3-8.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-8. Summary of Format 1 Instructions

OP	THUMB Assembler	ARM Equivalent	Action
00	LSL Rd, Rs, #Offset5	MOV _S Rd, Rs, LSL #Offset5	Shift Rs left by a 5-bit immediate value and store the result in Rd.
01	LSR Rd, Rs, #Offset5	MOV _S Rd, Rs, LSR #Offset5	Perform logical shift right on Rs by a 5-bit immediate value and store the result in Rd.
10	ASR Rd, Rs, #Offset5	MOV _S Rd, Rs, ASR #Offset5	Perform arithmetic shift right on Rs by a 5-bit immediate value and store the result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-8. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

LSR R2, R5, #27 ; Logical shift right the contents
; of R5 by 27 and store the result in R2.
; Set condition codes on the result.

FORMAT 2: ADD/SUBTRACT

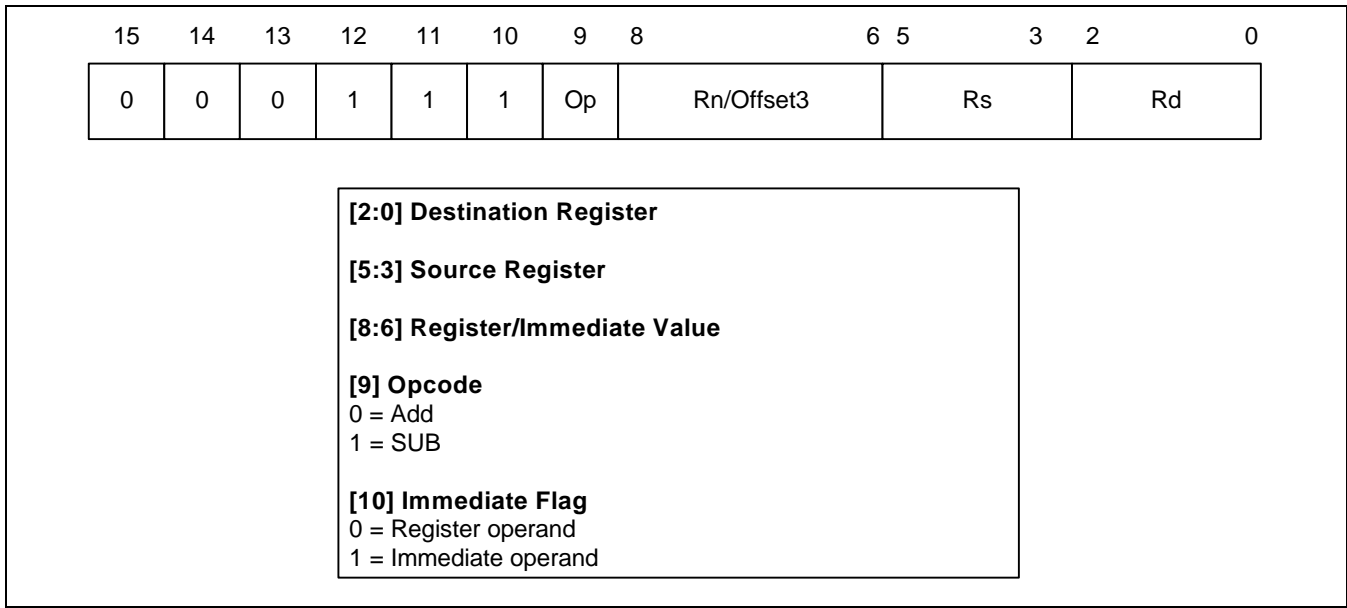


Figure 3-31. Format 2

OPERATION

These instructions allow the contents of a Lo register or a 3-bit immediate value to be added to or subtracted from a Lo register. The THUMB assembler syntax is shown in Table 3-9.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-9. Summary of Format 2 Instructions

OP	I	THUMB Assembler	ARM Equivalent	Action
0	0	ADD Rd, Rs, Rn	ADDS Rd, Rs, Rn	Add contents of Rn to contents of Rs. Place result in Rd.
0	1	ADD Rd, Rs, #Offset3	ADDS Rd, Rs, #Offset3	Add 3-bit immediate value to contents of Rs. Place result in Rd.
1	0	SUB Rd, Rs, Rn	SUBS Rd, Rs, Rn	Subtract contents of Rn from contents of Rs. Place result in Rd.
1	1	SUB Rd, Rs, #Offset3	SUBS Rd, Rs, #Offset3	Subtract 3-bit immediate value from contents of Rs. Place result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-9. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

ADD	R0, R3, R4	; R0: = R3 + R4 and set condition codes on the result.
SUB	R6, R2, #6	; R6: = R2 - 6 and set condition codes.

FORMAT 3: MOVE/COMPARE/ADD/SUBTRACT IMMEDIATE

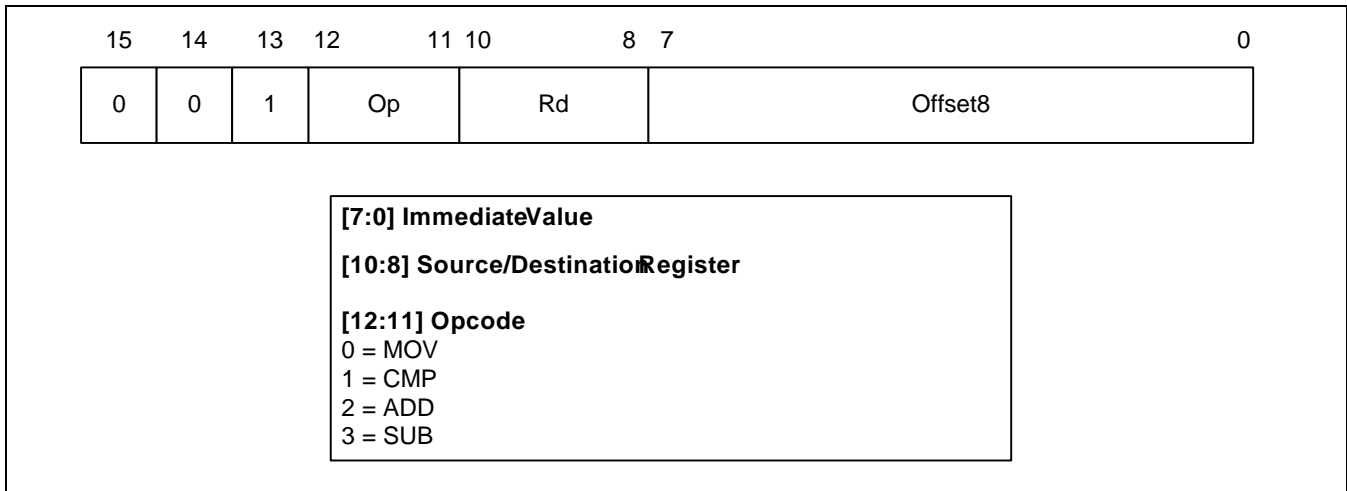


Figure 3-32. Format 3

OPERATIONS

The instructions in this group perform operations between a Lo register and an 8-bit immediate value. The THUMB assembler syntax is shown in Table 3-10.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-10. Summary of Format 3 Instructions

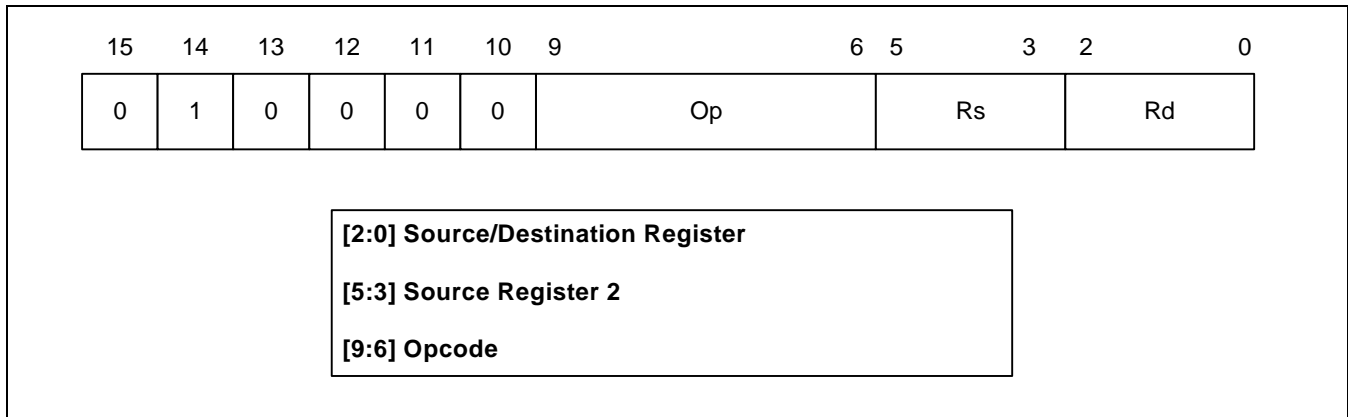
OP	THUMB Assembler	ARM Equivalent	Action
00	MOV Rd, #Offset8	MOVS Rd, #Offset8	Move 8-bit immediate value into Rd.
01	CMP Rd, #Offset8	CMP Rd, #Offset8	Compare contents of Rd with 8-bit immediate value.
10	ADD Rd, #Offset8	ADDS Rd, Rd, #Offset8	Add 8-bit immediate value to contents of Rd and place the result in Rd.
11	SUB Rd, #Offset8	SUBS Rd, Rd, #Offset8	Subtract 8-bit immediate value from contents of Rd and place the result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-10. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

MOV	R0, #128	; R0: = 128 and set condition codes
CMP	R2, #62	; Set condition codes on R2 - 62
ADD	R1, #255	; R1: = R1 + 255 and set condition codes
SUB	R6, #145	; R6: = R6 - 145 and set condition codes

FORMAT 4: ALU OPERATIONS**Figure 3-33. Format 4****OPERATION**

The following instructions perform ALU operations on a Lo register pair.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-11. Summary of Format 4 Instructions

OP	THUMB Assembler	ARM Equivalent	Action
0000	AND Rd, Rs	ANDS Rd, Rd, Rs	Rd: = Rd AND Rs
0001	EOR Rd, Rs	EORS Rd, Rd, Rs	Rd: = Rd EOR Rs
0010	LSL Rd, Rs	MOVS Rd, Rd, LSL Rs	Rd: = Rd << Rs
0011	LSR Rd, Rs	MOVS Rd, Rd, LSR Rs	Rd: = Rd >> Rs
0100	ASR Rd, Rs	MOVS Rd, Rd, ASR Rs	Rd: = Rd ASR Rs
0101	ADC Rd, Rs	ADCS Rd, Rd, Rs	Rd: = Rd + Rs + C-bit
0110	SBC Rd, Rs	SBCS Rd, Rd, Rs	Rd: = Rd - Rs - NOT C-bit
0111	ROR Rd, Rs	MOVS Rd, Rd, ROR Rs	Rd: = Rd ROR Rs
1000	TST Rd, Rs	TST Rd, Rs	Set condition codes on Rd AND Rs
1001	NEG Rd, Rs	RSBS Rd, Rs, #0	Rd = - Rs
1010	CMP Rd, Rs	CMP Rd, Rs	Set condition codes on Rd - Rs
1011	CMN Rd, Rs	CMN Rd, Rs	Set condition codes on Rd + Rs
1100	ORR Rd, Rs	ORRS Rd, Rd, Rs	Rd: = Rd OR Rs
1101	MUL Rd, Rs	MULS Rd, Rs, Rd	Rd: = Rs * Rd
1110	BIC Rd, Rs	BICS Rd, Rd, Rs	Rd: = Rd AND NOT Rs
1111	MVN Rd, Rs	MVNS Rd, Rs	Rd: = NOT Rs

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-11. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

EOR	R3, R4	; R3: = R3 EOR R4 and set condition codes
ROR	R1, R0	; Rotate Right R1 by the value in R0, store
		; the result in R1 and set condition codes
NEG	R5, R3	; Subtract the contents of R3 from zero,
		; store the result in R5. Set condition codes ie R5 = - R3
CMP	R2, R6	; Set the condition codes on the result of R2 - R6
MUL	R0, R7	; R0: = R7 * R0 and set condition codes

FORMAT 5: HI-REGISTER OPERATIONS/BRANCH EXCHANGE

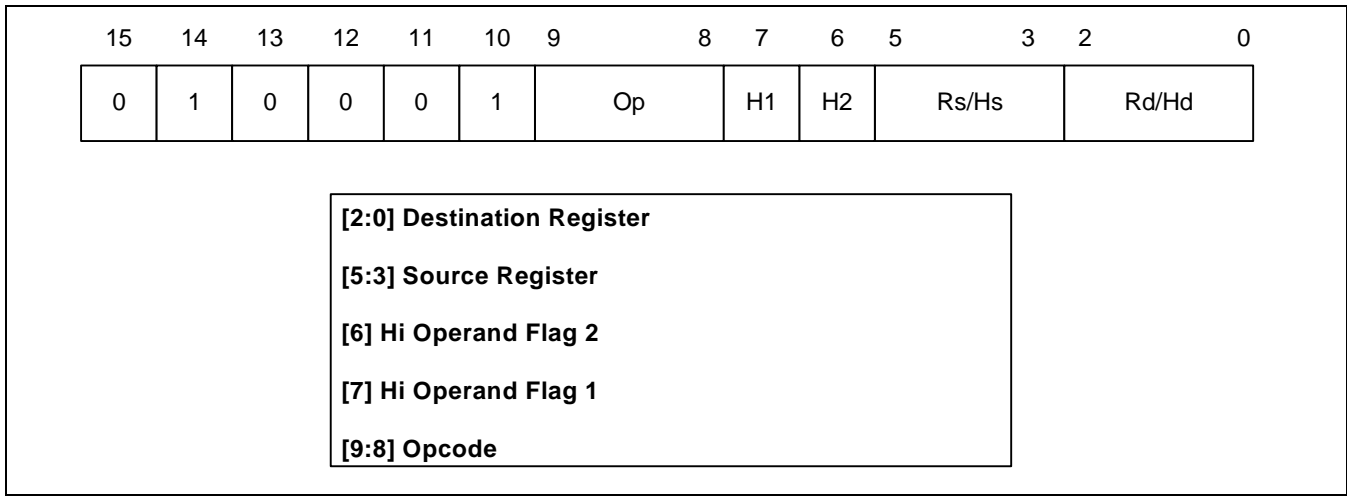


Figure 3-34. Format 5

OPERATION

There are four sets of instructions in this group. The first three allow ADD, CMP and MOV operations to be performed between Lo and Hi registers, or a pair of Hi registers. The fourth, BX, allows a branch to be performed which may also be used to switch processor state. The THUMB assembler syntax is shown in Table 3-12.

NOTE

In this group only CMP (Op = 01) sets the CPSR condition codes.

The action of H1 = 0, H2 = 0 for Op = 00 (ADD), Op = 01 (CMP) and Op = 10 (MOV) is undefined, and should not be used.

Table 3-12. Summary of Format 5 Instructions

OP	H1	H2	THUMB Assembler	ARM Equivalent	Action
00	0	1	ADD Rd, Hs	ADD Rd, Rd, Hs	Add a register in the range 8-15 to a register in the range 0-7.
00	1	0	ADD Hd, Rs	ADD Hd, Hd, Rs	Add a register in the range 0-7 to a register in the range 8-15.
00	1	1	ADD Hd, Hs	ADD Hd, Hd, Hs	Add two registers in the range 8-15
01	0	1	CMP Rd, Hs	CMP Rd, Hs	Compare a register in the range 0-7 with a register in the range 8-15. Set the condition code flags on the result.
01	1	0	CMP Hd, Rs	CMP Hd, Rs	Compare a register in the range 8-15 with a register in the range 0-7. Set the condition code flags on the result.
01	1	1	CMP Hd, Hs	CMP Hd, Hs	Compare two registers in the range 8-15. Set the condition code flags on the result.
10	0	1	MOV Rd, Hs	MOV Rd, Hs	Move a value from a register in the range 8-15 to a register in the range 0-7.
10	1	0	MOV Hd, Rs	MOV Hd, Rs	Move a value from a register in the range 0-7 to a register in the range 8-15.
10	1	1	MOV Hd, Hs	MOV Hd, Hs	Move a value between two registers in the range 8-15.
11	0	0	BX Rs	BX Rs	Perform branch (plus optional state change) to address in a register in the range 0-7.
11	0	1	BX Hs	BX Hs	Perform branch (plus optional state change) to address in a register in the range 8-15.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-12. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

THE BX INSTRUCTION

BX performs a branch to a routine whose start address is specified in a Lo or Hi register.

Bit 0 of the address determines the processor state on entry to the routine:

Bit 0 = 0 Causes the processor to enter ARM state.
 Bit 0 = 1 Causes the processor to enter THUMB state.

NOTE

The action of H1 = 1 for this instruction is undefined, and should not be used.

Examples

Hi-Register Operations

ADD	PC, R5	; PC: = PC + R5 but don't set the condition codes.
CMP	R4, R12	; Set the condition codes on the result of R4 - R12.
MOV	R15, R14	; Move R14 (LR) into R15 (PC)
		; but don't set the condition codes,
		; eg. return from subroutine.

Branch and Exchange

ADR	R1,outofTHUMB	; Switch from THUMB to ARM state.
MOV	R11,R1	; Load address of outofTHUMB into R1.
BX	R11	; Transfer the contents of R11 into the PC.
		; Bit 0 of R11 determines whether
		; ARM or THUMB state is entered, ie. ARM state here.
...		
ALIGN		
CODE32		
outofTHUMB		
		; Now processing ARM instructions...

USING R15 AS AN OPERAND

If R15 is used as an operand, the value will be the address of the instruction + 4 with bit 0 cleared. Executing a BX PC in THUMB state from a non-word aligned address will result in unpredictable execution.

FORMAT 6: PC-RELATIVE LOAD

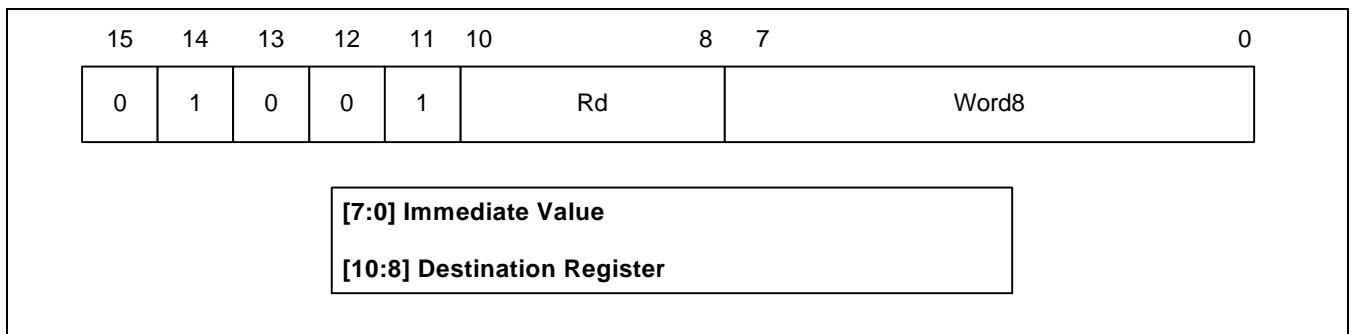


Figure 3-35. Format 6

OPERATION

This instruction loads a word from an address specified as a 10-bit immediate offset from the PC. The THUMB assembler syntax is shown below.

Table 3-13. Summary of PC-Relative Load Instruction

THUMB Assembler	ARM Equivalent	Action
LDR Rd, [PC, #Imm]	LDR Rd, [R15, #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the PC. Load the word from the resulting address into Rd.

NOTE: The value specified by #Imm is a full 10-bit address, but must always be word-aligned (i.e. with bits 1:0 set to 0), since the assembler places #Imm >> 2 in field Word 8. The value of the PC will be 4 bytes greater than the address of this instruction, but bit 1 of the PC is forced to 0 to ensure it is word aligned.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
LDR R3,[PC,#844]           ; Load into R3 the word found at the
                           ; address formed by adding 844 to PC.
                           ; bit[1] of PC is forced to zero.
                           ; Note that the THUMB opcode will contain
                           ; 211 as the Word8 value.
```

FORMAT 7: LOAD/STORE WITH REGISTER OFFSET

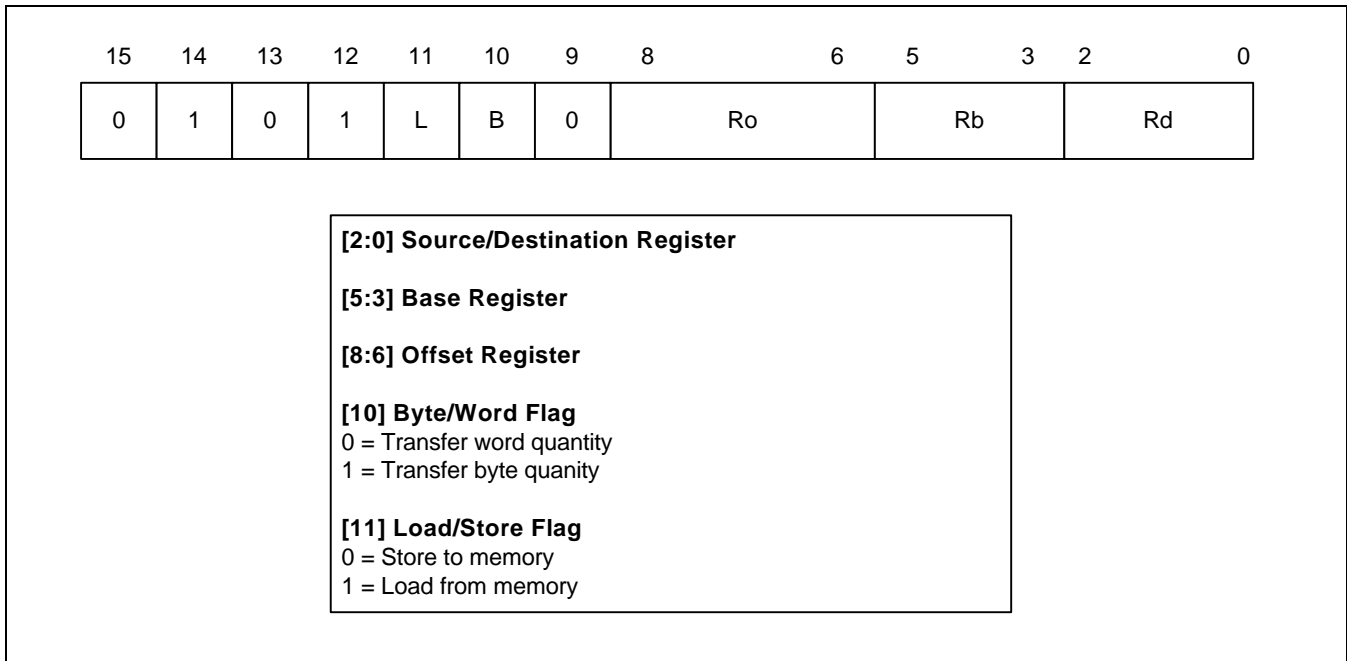


Figure 3-36. Format 7

OPERATION

These instructions transfer byte or word values between registers and memory. Memory addresses are pre-indexed using an offset register in the range 0-7. The THUMB assembler syntax is shown in Table 3-14.

Table 3-36. Summary of Format 7 Instructions

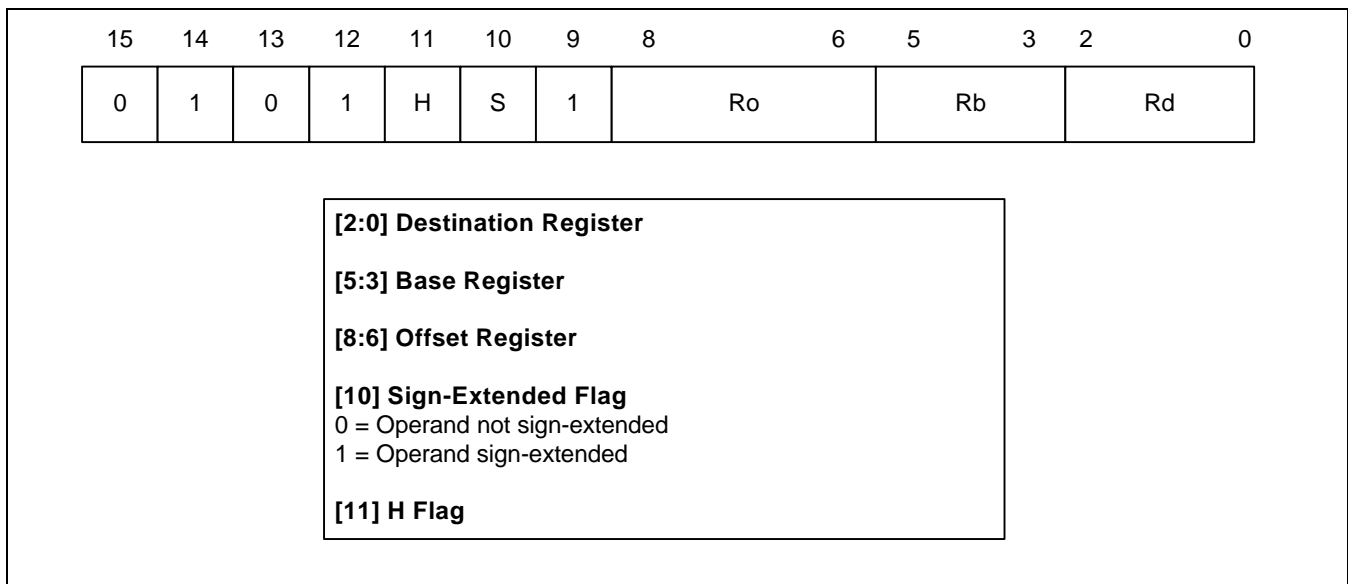
L	B	THUMB Assembler	ARM Equivalent	Action
0	0	STR Rd, [Rb, Ro]	STR Rd, [Rb, Ro]	Pre-indexed word store: Calculate the target address by adding together the value in Rb and the value in Ro. store the contents of Rd at the address.
0	1	STRB Rd, [Rb, Ro]	STRB Rd, [Rb, Ro]	Pre-indexed byte store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the byte value in Rd at the resulting address.
1	0	LDR Rd, [Rb, Ro]	LDR Rd, [Rb, Ro]	Pre-indexed word load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the contents of the address into Rd.
1	1	LDRB Rd, [Rb, Ro]	LDRB Rd, [Rb, Ro]	Pre-indexed byte load: Calculate the source address by adding together the value in Rb and the value in Ro. load the byte value at the resulting address.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-14. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

STR	R3, [R2,R6]	; Store word in R3 at the address
		; formed by adding R6 - R2.
LDRB	R2, [R0,R7]	; Load into R2 the byte found at
		; the address formed by adding R7 - R0.

FORMAT 8: LOAD/STORE SIGN-EXTENDED BYTE/HALFWORD**Figure 3-37. Format 8****OPERATION**

These instructions load optionally sign-extended bytes or halfwords, and store halfwords. The THUMB assembler syntax is shown below.

Table 3-15. Summary of Format 8 Instructions

S	H	THUMB Assembler	ARM Equivalent	Action
0	0	STRH Rd, [Rb, Ro]	STRH Rd, [Rb, Ro]	Store halfword: Add Ro to base address in Rb. Store bits 0-15 of Rd at the resulting address.
0	1	LDRH Rd, [Rb, Ro]	LDRH Rd, [Rb, Ro]	Load halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to 0.
1	0	LDSB Rd, [Rb, Ro]	LDRSB Rd, [Rb, Ro]	Load sign-extended byte: Add Ro to base address in Rb. Load bits 0-7 of Rd from the resulting address, and set bits 8-31 of Rd to bit 7.
1	1	LDSH Rd, [Rb, Ro]	LDRSH Rd, [Rb, Ro]	Load sign-extended halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to bit 15.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-15. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

STRH	R4, [R3, R0]	; Store the lower 16 bits of R4 at the
		; address formed by adding R0 - R3.
LDSB	R2, [R7, R1]	; Load into R2 the sign extended byte
		; found at the address formed by adding R1 - R7.
LDSH	R3, [R4, R2]	; Load into R3 the sign extended halfword
		; found at the address formed by adding R2 - R4.

FORMAT 9: LOAD/STORE WITH IMMEDIATE OFFSET

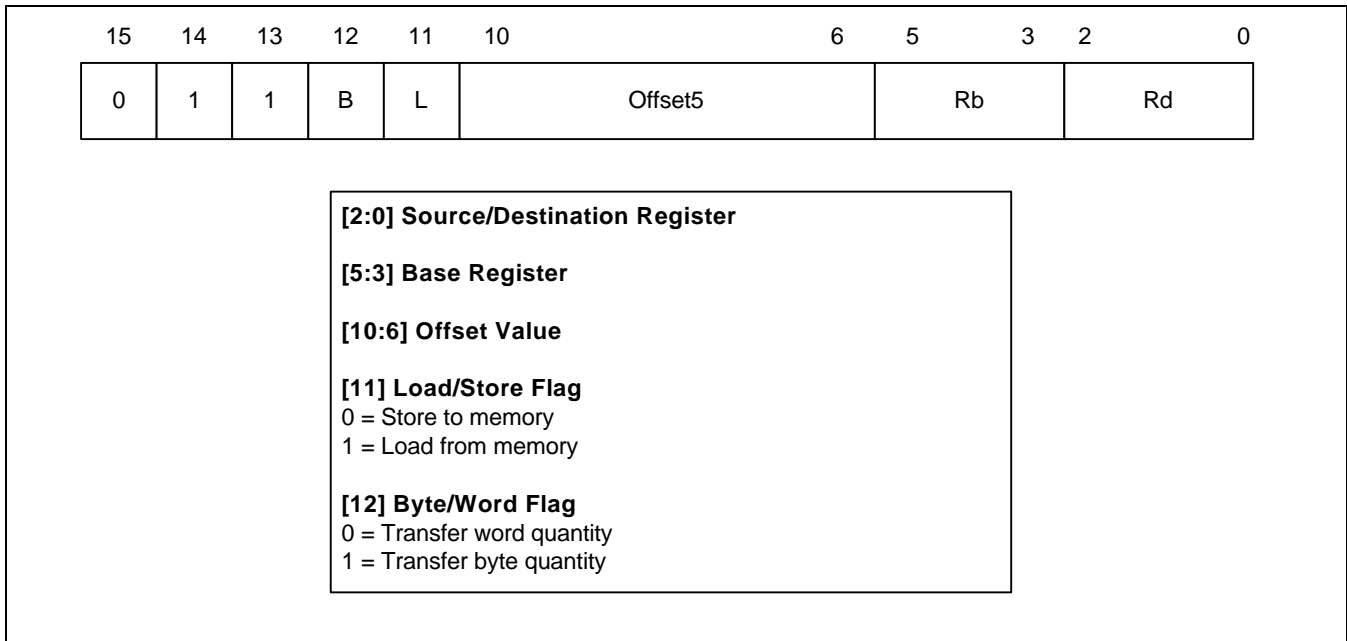


Figure 3-38. Format 9

OPERATION

These instructions transfer byte or word values between registers and memory using an immediate 5 or 7-bit offset. The THUMB assembler syntax is shown in Table 3-16.

Table 3-16. Summary of Format 9 Instructions

S	H	THUMB Assembler	ARM Equivalent	Action
0	0	STR Rd, [Rb, #Imm]	STR Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the contents of Rd at the address.
0	1	LDR Rd, [Rb, #Imm]	LDR Rd, [Rb, #Imm]	Calculate the source address by adding together the value in Rb and Imm. Load Rd from the address.
1	0	STRB Rd, [Rb, #Imm]	STRB Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the byte value in Rd at the address.
1	1	LDRB Rd, [Rb, #Imm]	LDRB Rd, [Rb, #Imm]	Calculate source address by adding together the value in Rb and Imm. Load the byte value at the address into Rd.

NOTE: For word accesses (B = 0), the value specified by #Imm is a full 7-bit address, but must be word-aligned (ie with bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Offset5 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-16. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

LDR	R2, [R5,#116]	; Load into R2 the word found at the ; address formed by adding 116 - R5. ; Note that the THUMB opcode will ; contain 29 as the Offset5 value.
STRB	R1, [R0,#13]	; Store the lower 8 bits of R1 at the ; address formed by adding 13 - R0. ; Note that the THUMB opcode will ; contain 13 as the Offset5 value.

FORMAT 10: LOAD/STORE HALFWORD

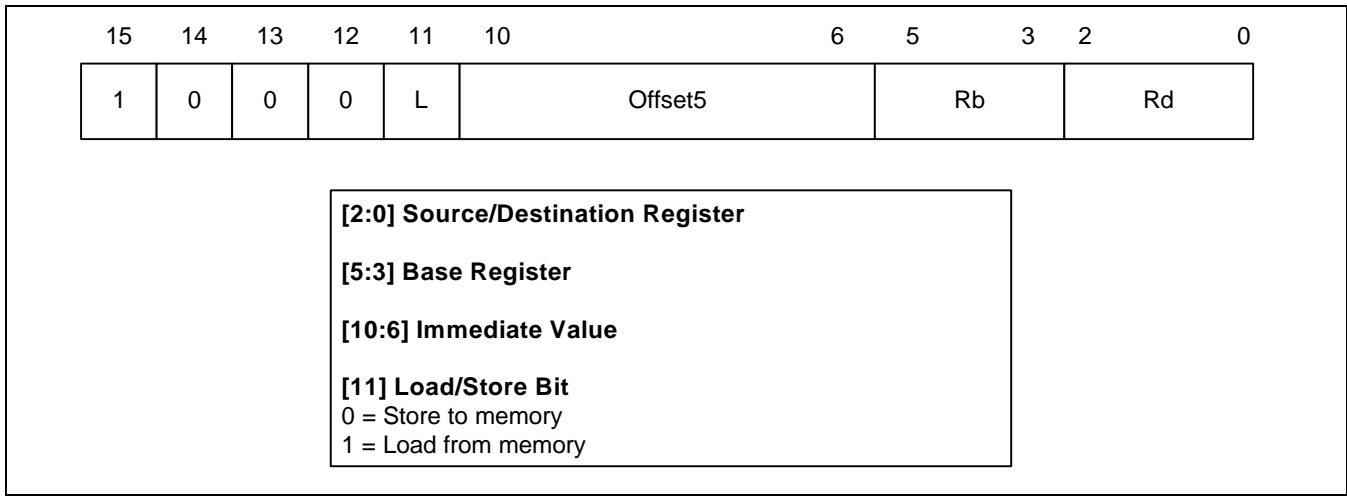


Figure 3-39. Format 10

OPERATION

These instructions transfer halfword values between a Lo register and memory. Addresses are pre-indexed, using a 6-bit immediate value. The THUMB assembler syntax is shown in Table 3-17.

Table 3-17. Halfword Data Transfer Instructions

L	THUMB Assembler	ARM Equivalent	Action
0	STRH Rd, [Rb, #Imm]	STRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb and store bits 0-15 of Rd at the resulting address.
1	LDRH Rd, [Rb, #Imm]	LDRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb. Load bits 0-15 from the resulting address into Rd and set bits 16-31 to zero.

NOTE: #Imm is a full 6-bit address but must be halfword-aligned (ie with bit 0 set to 0) since the assembler places #Imm >> 1 in the Offset5 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-17. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

STRH

R6, [R1, #56]

; Store the lower 16 bits of R4 at the address formed by

; adding 56 R1. Note that the THUMB opcode will contain

; 28 as the Offset5 value.

LDRH

R4, [R7, #4]

; Load into R4 the halfword found at the address formed by

; adding 4 to R7. Note that the THUMB opcode will

; 2 as the Offset5 value.

FORMAT 11: SP-RELATIVE LOAD/STORE

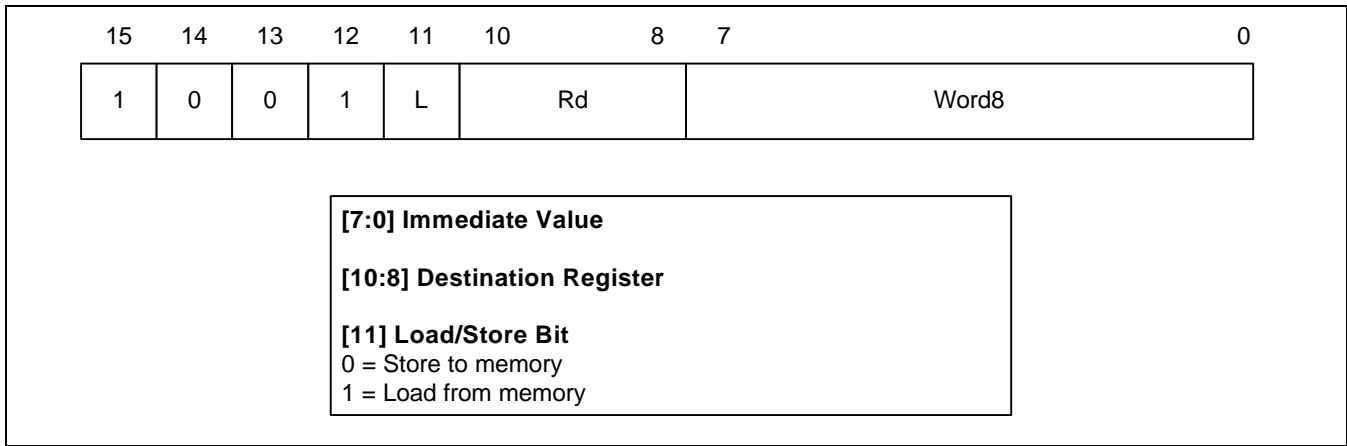


Figure 3-40. Format 11

OPERATION

The instructions in this group perform an SP-relative load or store. The THUMB assembler syntax is shown in the following table.

Table 3-18. SP-Relative Load/Store Instructions

L	THUMB Assembler	ARM Equivalent	Action
0	STR Rd, [SP, #Imm]	STR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Store the contents of Rd at the resulting address.
1	LDR Rd, [SP, #Imm]	LDR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Load the word from the resulting address into Rd.

NOTE: The offset supplied in #Imm is a full 10-bit address, but must always be word-aligned (ie bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Word8 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-18. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

STR R4, [SP,#492] ; Store the contents of R4 at the address
; formed by adding 492 to SP (R13).
; Note that the THUMB opcode will contain
; 123 as the Word8 value.

FORMAT 12: LOAD ADDRESS

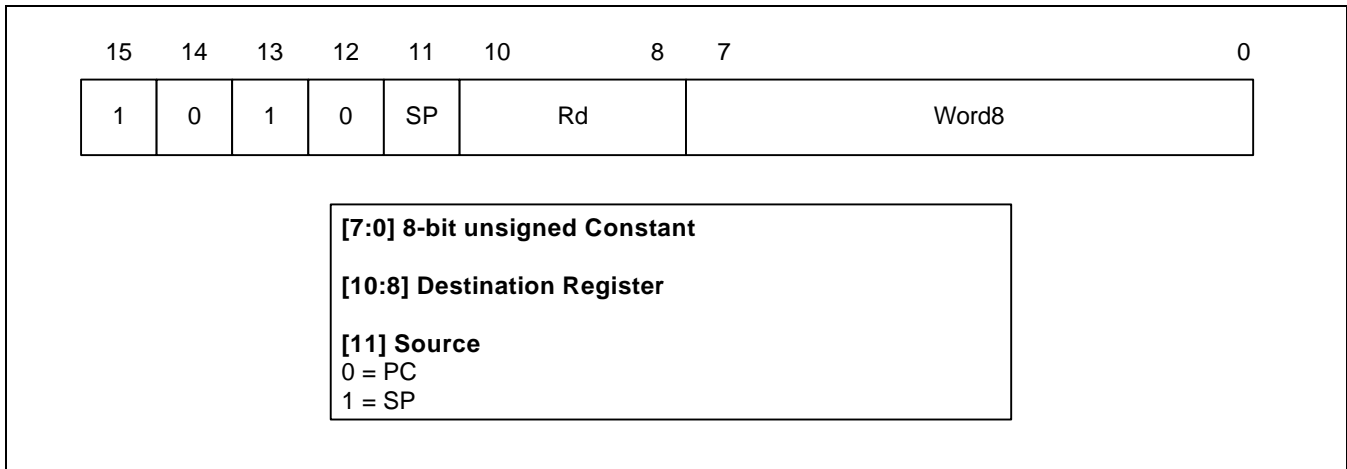


Figure 3-41. Format 12

OPERATION

These instructions calculate an address by adding an 10-bit constant to either the PC or the SP, and load the resulting address into a register. The THUMB assembler syntax is shown in the following table.

Table 3-19. Load Address

SP	THUMB Assembler	ARM Equivalent	Action
0	ADD Rd, PC, #Imm	ADD Rd, R15, #Imm	Add #Imm to the current value of the program counter (PC) and load the result into Rd.
1	ADD Rd, SP, #Imm	ADD Rd, R13, #Imm	Add #Imm to the current value of the stack pointer (SP) and load the result into Rd.

NOTE: The value specified by #Imm is a full 10-bit value, but this must be word-aligned (ie with bits 1:0 set to 0) since the assembler places #Imm >> 2 in field Word 8.

Where the PC is used as the source register (SP = 0), bit 1 of the PC is always read as 0. The value of the PC will be 4 bytes greater than the address of the instruction before bit 1 is forced to 0.

The CPSR condition codes are unaffected by these instructions.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-19. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

ADD	R2, PC, #572	; R2: = PC + 572, but don't set the ; condition codes. bit[1] of PC is forced to zero. ; Note that the THUMB opcode will ; contain 143 as the Word8 value.
ADD	R6, SP, #212	; R6: = SP (R13) + 212, but don't ; set the condition codes. ; Note that the THUMB opcode will ; contain 53 as the Word 8 value.

FORMAT 13: ADD OFFSET TO STACK POINTER

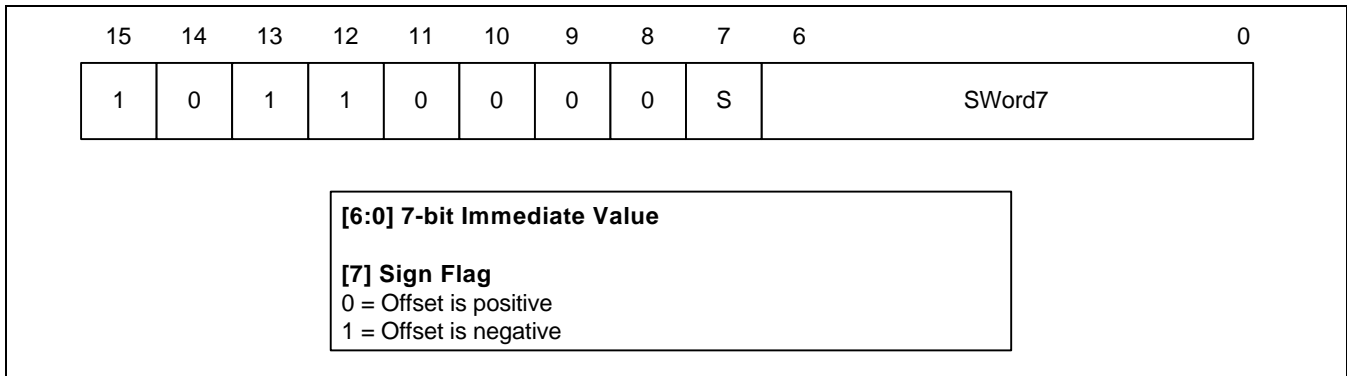


Figure 3-42. Format 13

OPERATION

This instruction adds a 9-bit signed constant to the stack pointer. The following table shows the THUMB assembler syntax.

Table 3-20. The ADD SP Instructions

S	THUMB Assembler	ARM Equivalent	Action
0	ADD SP, #Imm	ADD R13, R13, #Imm	Add #Imm to the stack pointer (SP).
1	ADD SP, #-Imm	SUB R13, R13, #Imm	Add #-Imm to the stack pointer (SP).

NOTE: The offset specified by #Imm can be up to -/+ 508, but must be word-aligned (i.e. with bits 1:0 set to 0) since the assembler converts #Imm to an 8-bit sign + magnitude number before placing it in field SWord7. The condition codes are not set by this instruction.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-20. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

ADDSP, #268

ADDSP, #-104

; SP (R13): = SP + 268, but don't set the condition codes.

; Note that the THUMB opcode will

; contain 67 as the Word7 value and S = 0.

; SP (R13): = SP - 104, but don't set the condition codes.

; Note that the THUMB opcode will contain

; 26 as the Word7 value and S = 1.

FORMAT 14: PUSH/POP REGISTERS

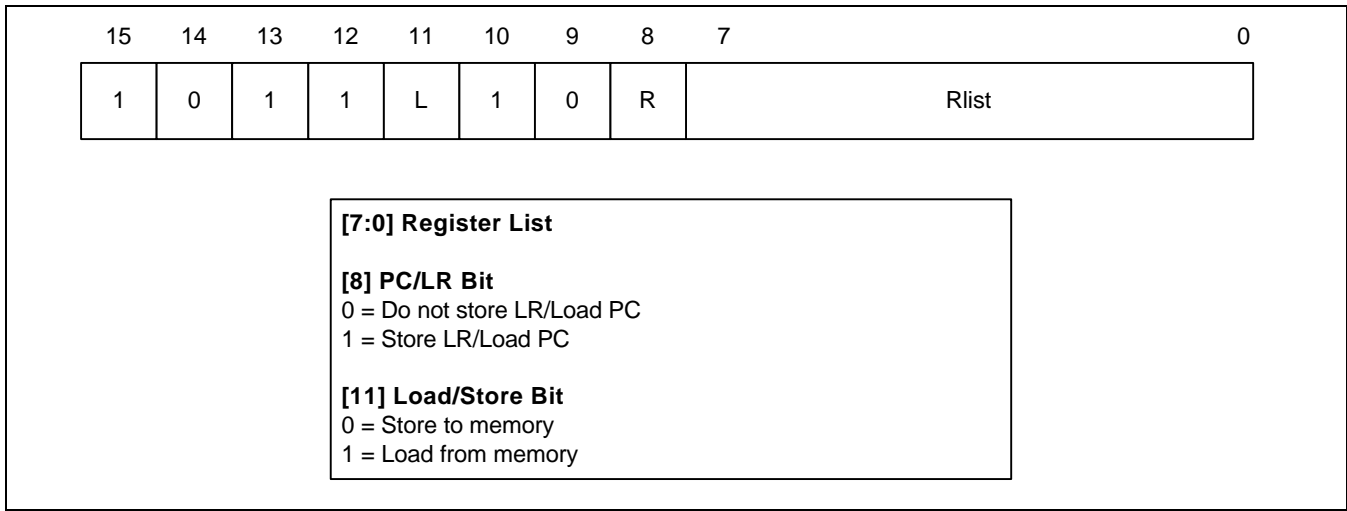


Figure 3-43. Format 14

OPERATION

The instructions in this group allow registers 0-7 and optionally LR to be pushed onto the stack, and registers 0-7 and optionally PC to be popped off the stack. The THUMB assembler syntax is shown in Table 3-21.

NOTE

The stack is always assumed to be full descending.

Table 3-21. PUSH and POP Instructions

S	H	THUMB Assembler	ARM Equivalent	Action
0	0	PUSH {Rlist}	STMDB R13!, {Rlist}	Push the registers specified by Rlist onto the stack. Update the stack pointer.
0	1	PUSH {Rlist, LR}	STMDB R13!, {Rlist, R14}	Push the Link Register and the registers specified by Rlist (if any) onto the stack. Update the stack pointer.
1	0	POP {Rlist}	LDMIA R13!, {Rlist}	Pop values off the stack into the registers specified by Rlist. Update the stack pointer.
1	1	POP {Rlist, PC}	LDMIA R13!, {Rlist, R15}	Pop values off the stack and load into the registers specified by Rlist. Pop the PC off the stack. Update the stack pointer.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-21. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

PUSH	{R0-R4,LR}	;	Store R0, R1, R2, R3, R4 and R14 (LR) at
		;	the stack pointed to by R13 (SP) and update R13.
		;	Useful at start of a sub-routine to
		;	save workspace and return address.
POP	{R2,R6,PC}	;	Load R2, R6 and R15 (PC) from the stack
		;	pointed to by R13 (SP) and update R13.
		;	Useful to restore workspace and return from sub-routine.

FORMAT 15: MULTIPLE LOAD/STORE

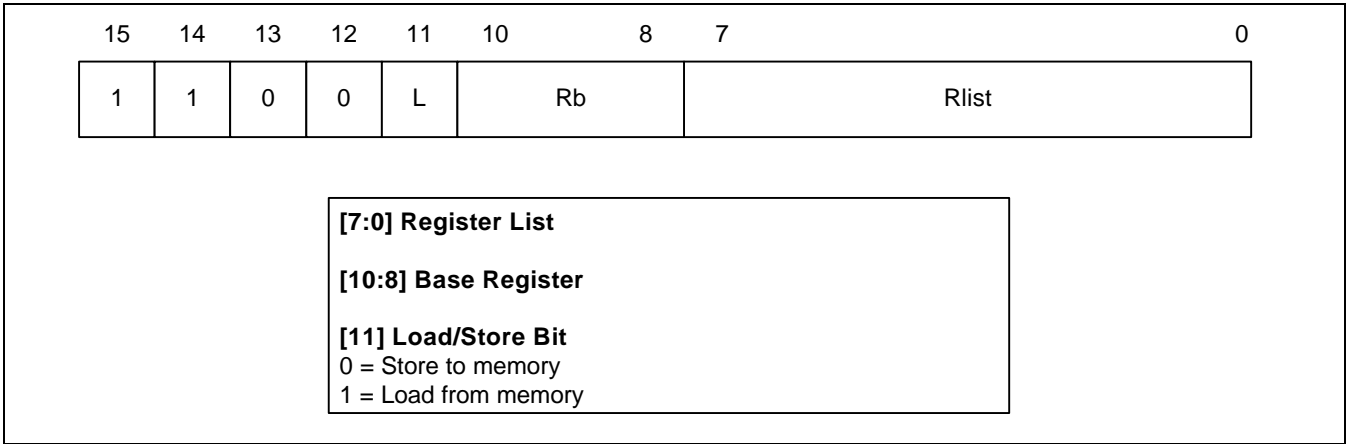


Figure 3-44. Format 15

OPERATION

These instructions allow multiple loading and storing of Lo registers. The THUMB assembler syntax is shown in the following table.

Table 3-22. The Multiple Load/Store Instructions

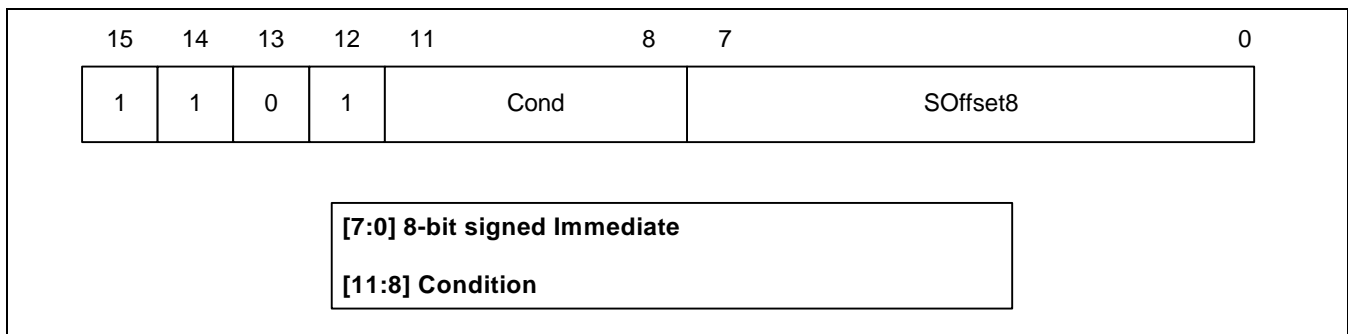
L	THUMB Assembler	ARM Equivalent	Action
0	STMIA Rb!, {Rlist}	STMIA Rb!, {Rlist}	Store the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.
1	LDMIA Rb!, {Rlist}	LDMIA Rb!, {Rlist}	Load the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-22. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

STMIA R0!, {R3-R7} ; Store the contents of registers R3-R7
; starting at the address specified in
; R0, incrementing the addresses for each word.
; Write back the updated value of R0.

FORMAT 16: CONDITIONAL BRANCH**Figure 3-45. Format 16****OPERATION**

The instructions in this group all perform a conditional Branch depending on the state of the CPSR condition codes. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

The THUMB assembler syntax is shown in the following table.

Table 3-23. The Conditional Branch Instructions

Code	THUMB Assembler	ARM Equivalent	Action
0000	BEQ label	BEQ label	Branch if Z set (equal)
0001	BNE label	BNE label	Branch if Z clear (not equal)
0010	BCS label	BCS label	Branch if C set (unsigned higher or same)
0011	BCC label	BCC label	Branch if C clear (unsigned lower)
0100	BMI label	BMI label	Branch if N set (negative)
0101	BPL label	BPL label	Branch if N clear (positive or zero)
0110	BVS label	BVS label	Branch if V set (overflow)
0111	BVC label	BVC label	Branch if V clear (no overflow)
1000	BHI label	BHI label	Branch if C set and Z clear (unsigned higher)
1001	BLS label	BLS label	Branch if C clear or Z set (unsigned lower or same)
1010	BGE label	BGE label	Branch if N set and V set, or N clear and V clear (greater or equal)
1011	BLT label	BLT label	Branch if N set and V clear, or N clear and V set (less than)
1100	BGT label	BGT label	Branch if Z clear, and either N set and V set or N clear and V clear (greater than)
1101	BLE label	BLE label	Branch if Z set, or N set and V clear, or N clear and V set (less than or equal)

NOTES:

- While label specifies a full 9-bit two's complement address, this must always be halfword-aligned (i.e. with bit 0 set to 0) since the assembler actually places label >> 1 in field SOffset8.

- 2. Cond = 1110 is undefined, and should not be used.
Cond = 1111 creates the SWI instruction: see .

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-23. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

	CMP R0, #45	; Branch to "over" if R0 > 45.
	BGT over	; Note that the THUMB opcode will contain
	...	; the number of halfwords to offset.
	...	
over	...	; Must be halfword aligned.
	...	

FORMAT 17: SOFTWARE INTERRUPT

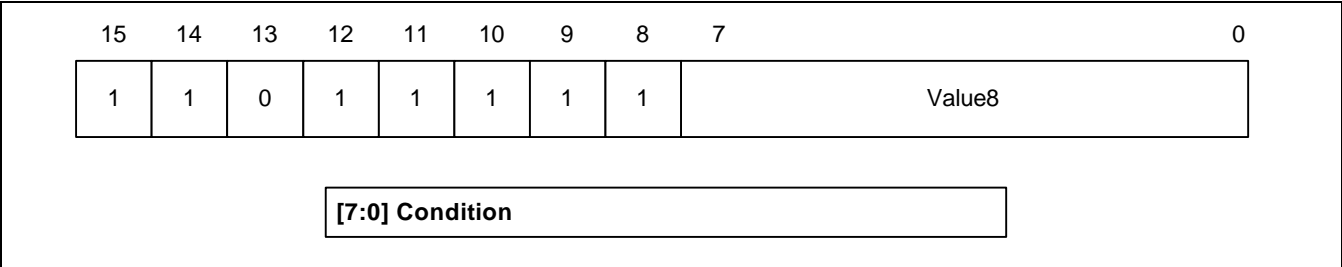


Figure 3-46. Format 17

OPERATION

The SWI instruction performs a software interrupt. On taking the SWI, the processor switches into ARM state and enters Supervisor (SVC) mode.

The THUMB assembler syntax for this instruction is shown below.

Table 3-24. The SWI Instruction

THUMB Assembler	ARM Equivalent	Action
SWI Value 8	SWI Value 8	Perform Software Interrupt: Move the address of the next instruction into LR, move CPSR to SPSR, load the SWI vector address (0x8) into the PC. Switch to ARM state and enter SVC mode.

NOTE: Value8 is used solely by the SWI handler; it is ignored by the processor.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-24. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

SWI 18

; Take the software interrupt exception.
; Enter supervisor mode with 18 as the
; requested SWI number.

FORMAT 18: UNCONDITIONAL BRANCH

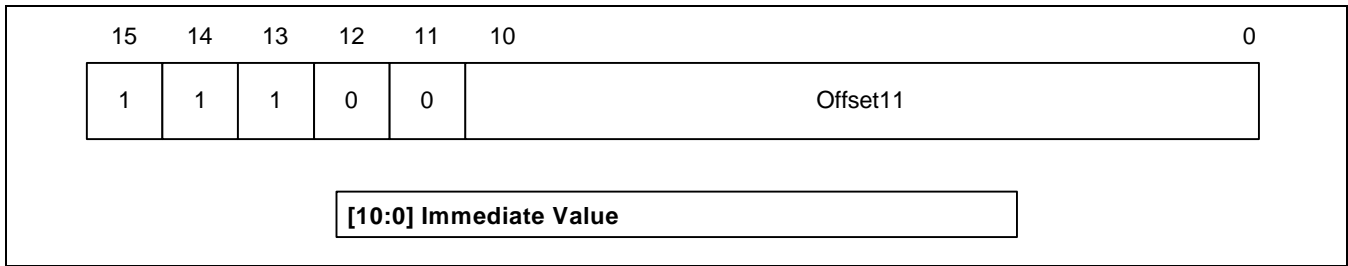


Figure 3-47. Format 18

OPERATION

This instruction performs a PC-relative Branch. The THUMB assembler syntax is shown below. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

Table 3-25. Summary of Branch Instruction

THUMB Assembler	ARM Equivalent	Action
B label	BAL label (halfword offset)	Branch PC relative +/- Offset11 << 1, where label is PC +/- 2048 bytes.

NOTE: The address specified by label is a full 12-bit two’s complement address, but must always be halfword aligned (i.e. bit 0 set to 0), since the assembler places label >> 1 in the Offset11 field.

Examples

here	B here	; Branch onto itself. Assembles to 0xE7FE.
		; (Note effect of PC offset).
	B jimmy	; Branch to 'jimmy'.
	...	; Note that the THUMB opcode will contain the number of
		; half-word to offset.
jimmy	...	; Must be halfword aligned.

15	14	13	12	11	10	0
1	1	1	1	H	Offset	

[10:0] Long Branch and Link Offset High/low

[11] Low/High Offset Bit
 0 = Offset high
 1 = Offset low

OPERATION

The assembler splits the 23-bit two's complement half-word offset specified by the label into two 11-bit halves, ignoring bit 0 (which must be 0), and creates two THUMB instructions.

In the first instruction the Offset field contains the upper 11 bits of the target address. This is shifted left by 12 bits and added to the current PC address. The resulting address is placed in LR.

In the second instruction the Offset field contains an 11-bit representation lower half of the target address. This is shifted left by 1 bit and added to LR. LR, which now contains the full 23-bit address, is placed in PC, the address of the instruction following the BL is placed in LR and bit 0 of LR is set.

3-90

SAMSUNG
ELECTRONICS

INSTRUCTION CYCLE TIMES

This instruction format does not have an equivalent ARM instruction.

Table 3-26. The BL Instruction

H	THUMB Assembler	ARM Equivalent	Action
0	BL label	none	LR: = PC + OffsetHigh << 12
1			Temp: = next instruction address PC: = LR + OffsetLow << 1 LR: = temp 1

Examples

next

BL faraway
...

faraway

...

; Unconditionally branch to 'faraway'

; and place following instruction

; address, i.e. 'next', in R14,the Link

; register and set bit 0 of LR high.

; Note that the THUMB opcodes will

; contain the number of halfwords to offset.

; Must be Half-word aligned.

INSTRUCTION SET EXAMPLES

The following examples show ways in which the THUMB instructions may be used to generate small and efficient code. Each example also shows the ARM equivalent so these may be compared.

MULTIPLICATION BY A CONSTANT USING SHIFTS AND ADDS

The following shows code to multiply by various constants using 1, 2 or 3 Thumb instructions alongside the ARM equivalents. For other constants it is generally better to use the built-in MUL instruction rather than using a sequence of 4 or more instructions.

Thumb	ARM
1. Multiplication by 2^n (1,2,4,8,...)	
LSL Ra, Rb, LSL #n	; MOV Ra, Rb, LSL #n
2. Multiplication by 2^n+1 (3,5,9,17,...)	
LSL Rt, Rb, #n	; ADD Ra, Rb, Rb, LSL #n
ADD Ra, Rt, Rb	
3. Multiplication by 2^n-1 (3,7,15,...)	
LSL Rt, Rb, #n	; RSB Ra, Rb, Rb, LSL #n
SUB Ra, Rt, Rb	
4. Multiplication by -2^n (-2, -4, -8, ...)	
LSL Ra, Rb, #n	; MOV Ra, Rb, LSL #n
MVN Ra, Ra	; RSB Ra, Ra, #0
5. Multiplication by -2^n-1 (-3, -7, -15, ...)	
LSL Rt, Rb, #n	; SUB Ra, Rb, Rb, LSL #n
SUB Ra, Rb, Rt	

Multiplication by any $C = \{2^{n+1}, 2^n-1, -2^n \text{ or } -2^{n-1}\} * 2^n$

Effectively this is any of the multiplications in 2 to 5 followed by a final shift. This allows the following additional constants to be multiplied. 6, 10, 12, 14, 18, 20, 24, 28, 30, 34, 36, 40, 48, 56, 60, 62

(2..5)	Ra, Ra, #n	; (2..5)
LSL		; MOV Ra, Ra, LSL #n

GENERAL PURPOSE SIGNED DIVIDE

This example shows a general purpose signed divide and remainder routine in both Thumb and ARM code.

Thumb code

```

;signed_divide                                ; Signed divide of R1 by R0: returns quotient in R0,
                                                ; remainder in R1

;Get abs value of R0 into R3
        ASR      R2, R0, #31                ; Get 0 or -1 in R2 depending on sign of R0
        EOR      R0, R2                    ; EOR with -1 (0xFFFFFFFF) if negative
        SUB      R3, R0, R2                ; and ADD 1 (SUB -1) to get abs value

;SUB always sets flag so go & report division by 0 if necessary
        BEQ      divide_by_zero

;Get abs value of R1 by xoring with 0xFFFFFFFF and adding 1 if negative
        ASR      R0, R1, #31                ; Get 0 or -1 in R3 depending on sign of R1
        EOR      R1, R0                    ; EOR with -1 (0xFFFFFFFF) if negative
        SUB      R1, R0                    ; and ADD 1 (SUB -1) to get abs value

;Save signs (0 or -1 in R0 & R2) for later use in determining ; sign of quotient & remainder.
        PUSH     {R0, R2}

;Justification, shift 1 bit at a time until divisor (R0 value) ; is just <= than dividend (R1 value). To do this shift
dividend ; right by 1 and stop as soon as shifted value becomes >.
        LSR      R0, R1, #1
        MOV      R2, R3
        B        %FT0
just_l   LSL      R2, #1
0        CMP      R2, R0
        BLS      just_l
        MOV      R0, #0                    ; Set accumulator to 0
        B        %FT0                    ; Branch into division loop
div_l    LSR      R2, #1
0        CMP      R1, R2                    ; Test subtract
        BCC      %FT0
        SUB      R1, R2                    ; If successful do a real subtract
0        ADC      R0, R0                    ; Shift result and add 1 if subtract succeeded
        CMP      R2, R3                    ; Terminate when R2 == R3 (i.e. we have just
        BNE      div_l                    ; tested subtracting the 'ones' value).

;Now fixup the signs of the quotient (R0) and remainder (R1)
        POP      {R2, R3}                ; Get dividend/divisor signs back
        EOR      R3, R2                    ; Result sign
        EOR      R0, R3                    ; Negate if result sign = - 1
        SUB      R0, R3
        EOR      R1, R2                    ; Negate remainder if dividend sign = - 1
        SUB      R1, R2
        MOV      pc, lr

```


ARM Code

signed_divide ; Effectively zero a4 as top bit will be shifted out later

ANDS a4, a1, #&80000000

RSBMI a1, a1, #0

EORS ip, a4, a2, ASR #32

;ip bit 31 = sign of result

;ip bit 30 = sign of a2

RSBCS a2, a2, #0

;Central part is identical code to udiv (without MOV a4, #0 which comes for free as part of signed entry sequence)

MOVS a3, a1

BEQ divide_by_zero

just_l ; Justification stage shifts 1 bit at a time

CMP a3, a2, LSR #1

MOVLS a3, a3, LSL #1

; NB: LSL #1 is always OK if LS succeeds

BLO s_loop

div_l

CMP a2, a3

ADC a4, a4, a4

SUBCS a2, a2, a3

TEQ a3, a1

MOVNE a3, a3, LSR #1

BNE s_loop2

MOV a1, a4

MOVS ip, ip, ASL #1

RSBCS a1, a1, #0

RSBMI a2, a2, #0

MOV pc, lr

DIVISION BY A CONSTANT

Division by a constant can often be performed by a short fixed sequence of shifts, adds and subtracts.

Here is an example of a divide by 10 routine based on the algorithm in the ARM Cookbook in both Thumb and ARM code.

Thumb Code

```

udiv10                                ; Take argument in a1 returns quotient in a1,
                                      ; remainder in a2
        MOV        a2, a1
        LSR        a3, a1, #2
        SUB        a1, a3
        LSR        a3, a1, #4
        ADD        a1, a3
        LSR        a3, a1, #8
        ADD        a1, a3
        LSR        a3, a1, #16
        ADD        a1, a3
        LSR        a1, #3
        ASL        a3, a1, #2
        ADD        a3, a1
        ASL        a3, #1
        SUB        a2, a3
        CMP        a2, #10
        BLT        %FT0
        ADD        a1, #1
        SUB        a2, #10
0      MOV        pc, lr

```

ARM Code

```

udiv10                                ; Take argument in a1 returns quotient in a1,
                                      ; remainder in a2
        SUB        a2, a1, #10
        SUB        a1, a1, a1, lsr #2
        ADD        a1, a1, a1, lsr #4
        ADD        a1, a1, a1, lsr #8
        ADD        a1, a1, a1, lsr #16
        MOV        a1, a1, lsr #3
        ADD        a3, a1, a1, asl #2
        SUBS       a2, a2, a3, asl #1
        ADDPL      a1, a1, #1
        ADDMI      a2, a2, #10
        MOV        pc, lr

```

INSTRUCTION SET SUMMAY	1
FORMAT SUMMARY	1
INSTRUCTION SUMMARY	2
THE CONDITION FIELD	3
BRANCH AND EXCHANGE (BX)	4
INSTRUCTION CYCLE TIMES	4
ASSEMBLER SYNTAX	4
USING R15 AS AN OPERAND	4
BRANCH AND BRANCH WITH LINK (B, BL)	6
THE LINK BIT	6
INSTRUCTION CYCLE TIMES	6
ASSEMBLER SYNTAX	7
DATA PROCESSING	8
CPSR FLAGS	9
SHIFTS	10
IMMEDIATE OPERAND ROTATES	14
WRITING TO R15	14
USING R15 AS AN OPERAND	14
TEQ, TST, CMP AND CMN OPCODES	14
INSTRUCTION CYCLE TIMES	14
ASSEMBLER SYNTAX	15
PSR TRANSFER (MRS, MSR)	16
OPERAND RESTRICTIONS	16
RESERVED BITS	18
INSTRUCTION CYCLE TIMES	18
ASSEMBLER SYNTAX	19
MULTIPLY AND MULTIPLY-ACCUMULATE (MUL, MLA)	20
CPSR FLAGS	21

INSTRUCTION CYCLE TIMES	21
ASSEMBLER SYNTAX	21
MULTIPLY LONG AND MULTIPLY-ACCUMULATE LONG (MULL,MLAL)	22
OPERAND RESTRICTIONS	22
CPSR FLAGS	23
INSTRUCTION CYCLE TIMES	23
ASSEMBLER SYNTAX	23
SINGLE DATA TRANSFER (LDR, STR)	24
OFFSETS AND AUTO-INDEXING	25
SHIFTED REGISTER OFFSET	25
BYTES AND WORDS	25
USE OF R15	27
RESTRICTION ON THE USE OF BASE REGISTER	27
DATA ABORTS	27
INSTRUCTION CYCLE TIMES	27
ASSEMBLER SYNTAX	28
HALFWORD AND SIGNED DATA TRANSFER (LDRH/STRH/LDRSB/LDRSH)	30
OFFSETS AND AUTO-INDEXING	32
HALFWORD LOAD AND STORES	32
SIGNED BYTE AND HALFWORD LOADS	32
ENDIANNESS AND BYTE/HALFWORD SELECTION	32
USE OF R15	33
DATA ABORTS	33
INSTRUCTION CYCLE TIMES	33
ASSEMBLER SYNTAX	34
BLOCK DATA TRANSFER (LDM, STM)	36
THE REGISTER LIST	36
ADDRESSING MODES	37
ADDRESS ALIGNMENT	37

USE OF THE S BIT	39
USE OF R15 AS THE BASE	39
INCLUSION OF THE BASE IN THE REGISTER LIST	40
DATA ABORTS	40
INSTRUCTION CYCLE TIMES	40
ASSEMBLER SYNTAX	41
SINGLE DATA SWAP (SWP)	43
BYTES AND WORDS	43
USE OF R15	43
DATA ABORTS	44
INSTRUCTION CYCLE TIMES	44
ASSEMBLER SYNTAX	44
SOFTWARE INTERRUPT (SWI)	45
RETURN FROM THE SUPERVISOR	45
COMMENT FIELD	45
INSTRUCTION CYCLE TIMES	45
ASSEMBLER SYNTAX	46
COPROCESSOR DATA OPERATIONS (CDP)	47
COPROCESSOR INSTRUCTIONS	47
THE COPROCESSOR FIELDS	48
INSTRUCTION CYCLE TIMES	48
ASSEMBLER SYNTAX	48
COPROCESSOR DATA TRANSFERS (LDC, STC)	49
THE COPROCESSOR FIELDS	49
ADDRESSING MODES	50
ADDRESS ALIGNMENT	50
USE OF R15	50
DATA ABORTS	50
INSTRUCTION CYCLE TIMES	50

ASSEMBLER SYNTAX	51
COPROCESSOR REGISTER TRANSFERS (MRC, MCR)	52
THE COPROCESSOR FIELDS.....	52
TRANSFERS TO R15	53
TRANSFERS FROM R15.....	53
INSTRUCTION CYCLE TIMES	53
ASSEMBLER SYNTAX	53
UNDEFINED INSTRUCTION.....	54
INSTRUCTION CYCLE TIMES	54
ASSEMBLER SYNTAX	54
INSTRUCTION SET EXAMPLES.....	55
USING THE CONDITIONAL INSTRUCTIONS	55
PSEUDO-RANDOM BINARY SEQUENCE GENERATOR	57
MULTIPLICATION BY CONSTANT USING THE BARREL SHIFTER.....	57
LOADING A WORD FROM AN UNKNOWN ALIGNMENT	59
THUMB INSTRUCTION SET FORMAT	60
FORMAT SUMMARY	60
OPCODE SUMMARY.....	61
FORMAT 1: MOVE SHIFTED REGISTER	63
OPERATION.....	63
INSTRUCTION CYCLE TIMES	63
FORMAT 2: ADD/SUBTRACT	64
OPERATION.....	64
INSTRUCTION CYCLE TIMES	64
FORMAT 3: MOVE/COMPARE/ADD/SUBTRACT IMMEDIATE.....	65
OPERATIONS.....	65
INSTRUCTION CYCLE TIMES	65
FORMAT 4: ALU OPERATIONS.....	66
OPERATION.....	66

INSTRUCTION CYCLE TIMES	67
FORMAT 5: HI-REGISTER OPERATIONS/BRANCH EXCHANGE.....	68
OPERATION	68
INSTRUCTION CYCLE TIMES	69
THE BX INSTRUCTION	69
USING R15 AS AN OPERAND.....	70
FORMAT 6: PC-RELATIVE LOAD	71
OPERATION	71
INSTRUCTION CYCLE TIMES	71
FORMAT 7: LOAD/STORE WITH REGISTER OFFSET	72
OPERATION	72
INSTRUCTION CYCLE TIMES	73
FORMAT 8: LOAD/STORE SIGN-EXTENDED BYTE/HALFWORD	74
OPERATION	74
INSTRUCTION CYCLE TIMES	75
FORMAT 9: LOAD/STORE WITH IMMEDIATE OFFSET	76
OPERATION	76
INSTRUCTION CYCLE TIMES	77
FORMAT 10: LOAD/STORE HALFWORD	78
INSTRUCTION CYCLE TIMES	78
FORMAT 11: SP-RELATIVE LOAD/STORE	79
OPERATION	79
INSTRUCTION CYCLE TIMES	79
FORMAT 12: LOAD ADDRESS	80
OPERATION	80
INSTRUCTION CYCLE TIMES	81
FORMAT 13: ADD OFFSET TO STACK POINTER	82
OPERATION	82
INSTRUCTION CYCLE TIMES	82

FORMAT 14: PUSH/POP REGISTERS	83
OPERATION	83
INSTRUCTION CYCLE TIMES	84
FORMAT 15: MULTIPLE LOAD/STORE	85
OPERATION	85
INSTRUCTION CYCLE TIMES	85
FORMAT 16: CONDITIONAL BRANCH	86
OPERATION	86
INSTRUCTION CYCLE TIMES	87
FORMAT 17: SOFTWARE INTERRUPT	88
INSTRUCTION CYCLE TIMES	88
FORMAT 18: UNCONDITIONAL BRANCH	89
OPERATION	89
FORMAT 19: LONG BRANCH WITH LINK	90
OPERATION	90
INSTRUCTION CYCLE TIMES	91
INSTRUCTION SET EXAMPLES	92
MULTIPLICATION BY A CONSTANT USING SHIFTS AND ADDS	92
GENERAL PURPOSE SIGNED DIVIDE	93
DIVISION BY A CONSTANT	95

4

SYSTEM MANAGER

OVERVIEW

KS32C65100 System Manager provides the following features.

- It arbitrates the system bus access requests from a master block, based on a fixed priority.
- It provides the appropriate memory control signals for an external memory access.

(If a master block such as DMA or CPU generates an address that corresponds to a DRAM bank, the System Manager's DRAM control block generates appropriate DRAM control signals such as nRAS, nCAS, Address and Data.)

- It compensates for differences in bus width for data flowing between the external memory bus and the internal data bus.
- Supports big-endian mode with efficiency for most graphic device drivers (refer to Figure. 4-5).

SYSTEM MANAGER REGISTERS (SMR)

The KS32C65100 microcontroller has register files (named Special Function Register, SFR) for keeping the system control information for the system manager, cache, Internal RAM, DMA, UART blocks and so on. The SFR has System Manager Register files (SMR) for the configuration of external memory maps such as DRAM, SRAM and ROM, and extra I/O control.

Programmers can specify the memory type, external bus width, access cycles, necessary control signal's timing (eg. nRAS and nCAS, etc.), memory bank location and memory bank size of each bank which has a very configurable address spacing by utilizing the SMR. The SMR, also, provide (or accept) the features such as control signals, address, and data that are required by external I/O devices during normal system operation. The SMR is constituted of 11 registers to control one ROM bank, two SRAM banks, two DRAM banks, four extra I/O banks and a DRAM Refresh Control Register, system configuration register.

The KS32C65100 provides up to 32M bytes of address space and each bank provides up to 4M half word of memory space because the KS32C65100 has 22 address pins/16 bit data width for each bank.

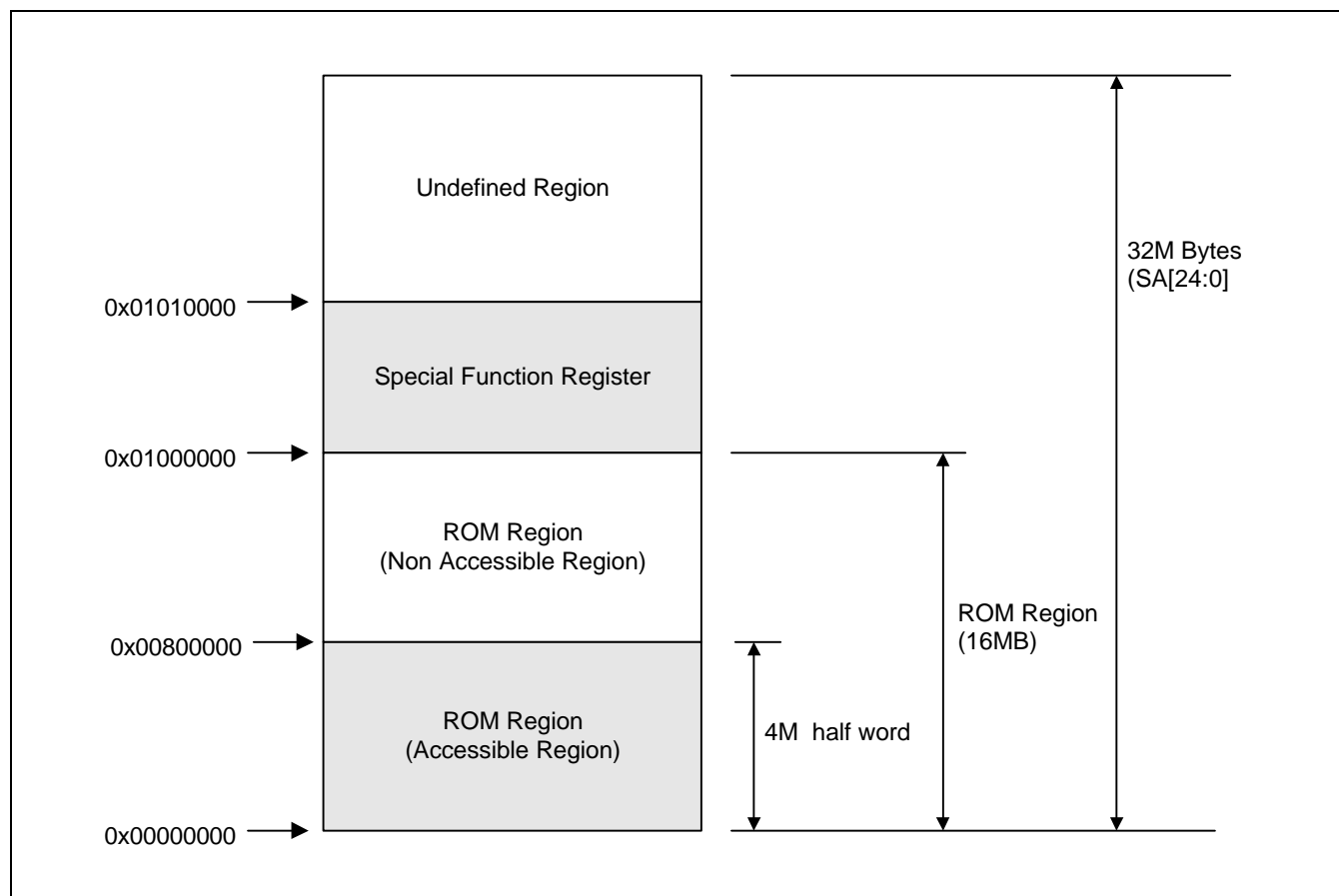


Figure 4-1. System Memory Map (Default Map After Reset)

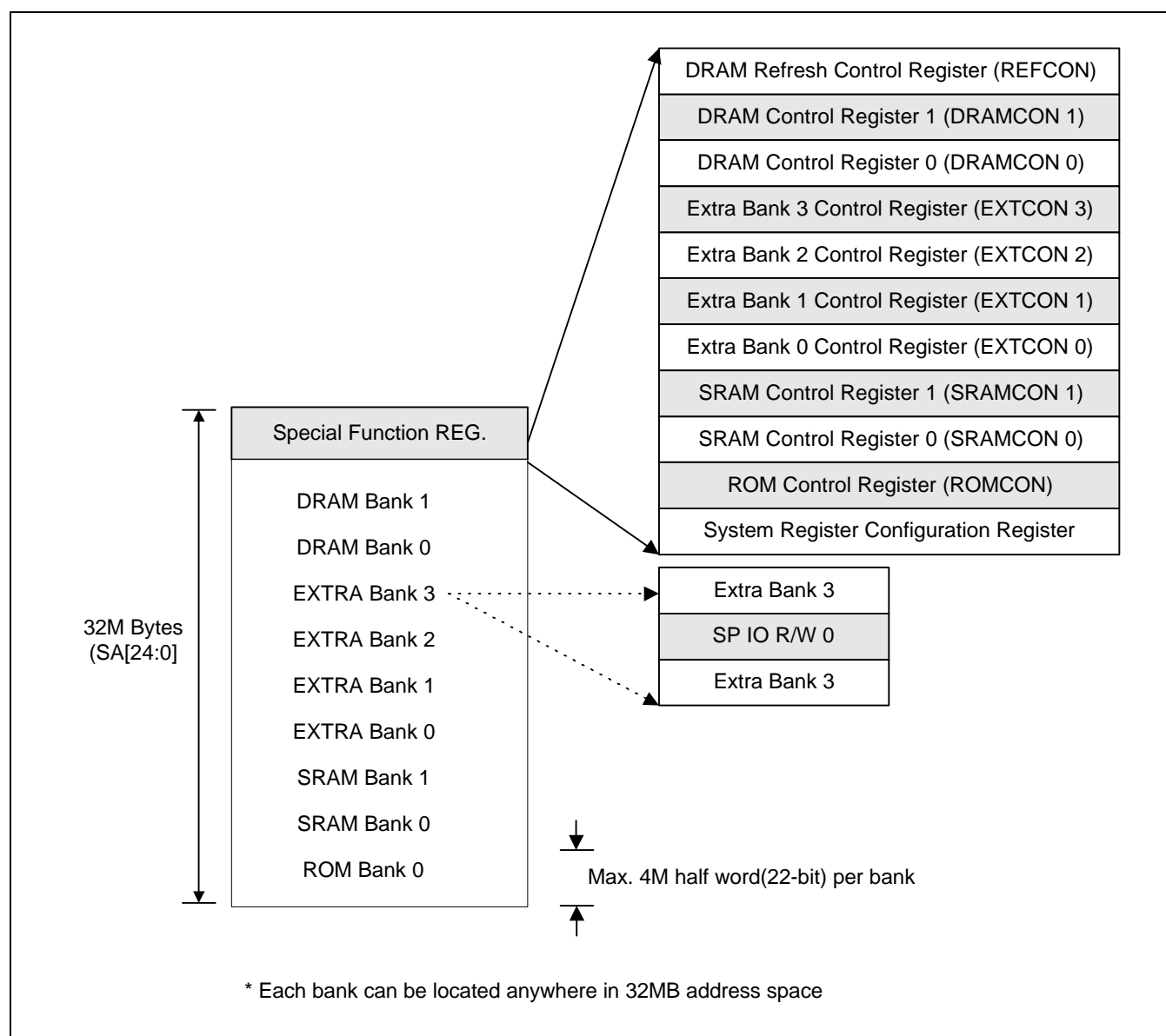


Figure 4-2. System Memory Map

The KS32C65100 uses an internal 25 bit system address bus and it can provide 32M bytes the size of memory space. The bank allocation methodology is very configurable and you can use any address area within 0000000h~1FFFFFFh by 64K byte address steps. The last 64K bytes area cannot be allocated as memory banks except SFR. Because the last 64KB bank is 1FFxxxxh, the next pointer of the last bank should have "+1", 200xxxxh, but it has 000xxxxh because the next pointer is 9-bit. If a user needs to utilize the full 32M bytes of memory space, you are recommended to allocate the SFRs to the last 64k byte area, 1FF0000h ~ 1FFFFFFh, and other banks for the rest of the area.

For programming convenience, programmers want to get rid of scattered memory area and want to have consecutively connected memory space without any blank areas. KS32C65100's configurable memory allocation methodology provides a very adaptive solution for this type of requirements. You can move the memory area easily by only changing the SMR.

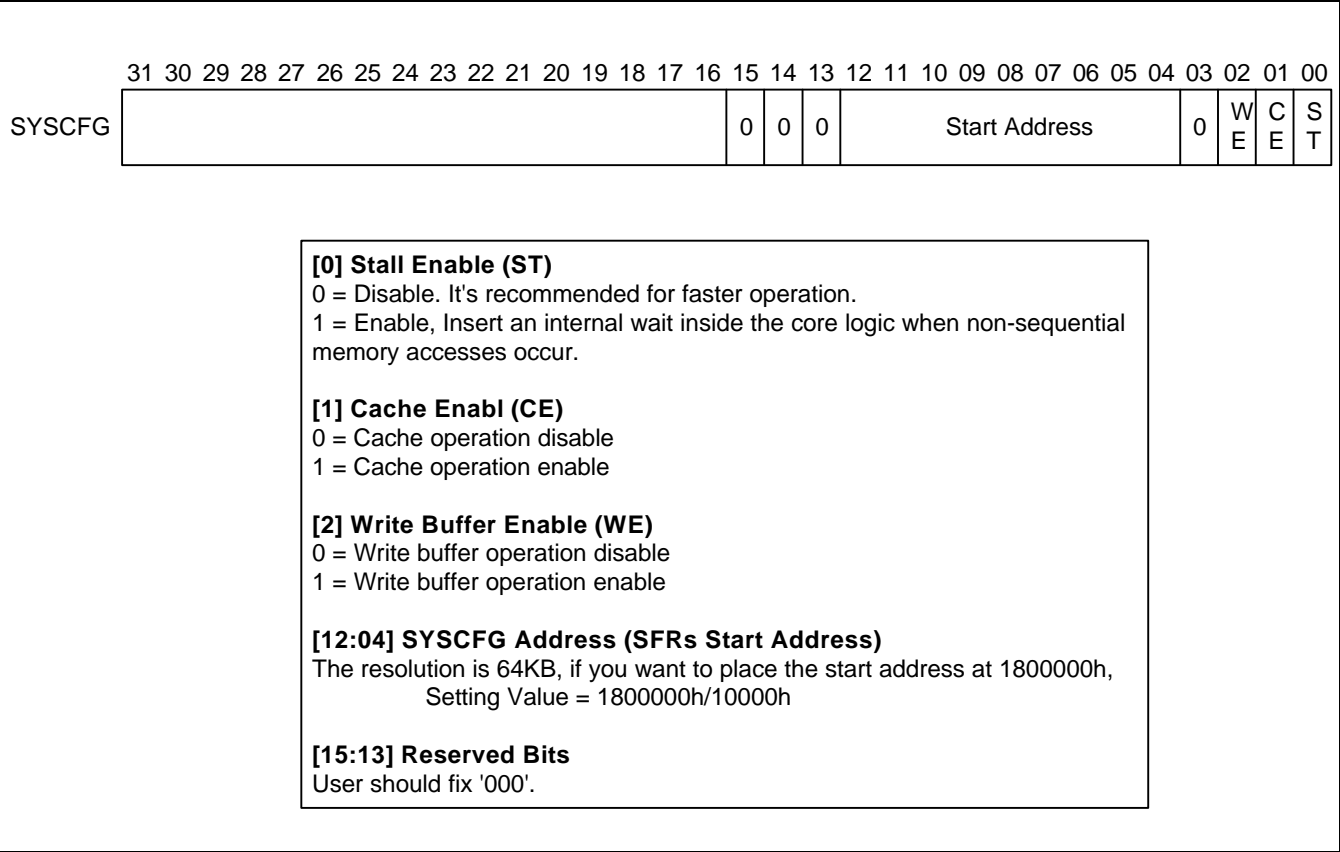
When you try to change physical DRAM memory size, for example from 1MB to 2MB, user can easily change memory configuration by modifying the system manager register(SMR) in the KS32C65100 microcontroller. KS32C65100 provides two DRAM banks and changeable memory space that has configurable DRAM size to 2M word. So then you can enlarge memory space just by changing the end point of the DRAM bank.

SYSTEM REGISTER ADDRESS CONFIGURATION REGISTER (SYSCFG)

The KS32C65100 System Manager Registers (SMR) have a register which determines the start (Base point) address of the Special Function Register (SFR) files. It is the "System Register Address Configuration Register (SYSCFG)", and its contents indicate the start (base point) address of SFR.

If the initial value is 1001h, SYSCFG is mapped to the virtual address 01000000h.

Register	Offset Address	R/W	Description	Reset Val.
SYSCFG	0x0000	R/W	Special function register start address	0x1001



Start Address

The SYSCFG[12:04] bits indicate the start address [24:16] of SFRs. As SYSCFG is located at the bottom of the Special Function Register (SFR) files, SYSCFG's location is same as the start address of SFRs.

Programmers can allocate SFRs to arbitrary locations by using the SYSCFG. You are recommended not to change the SYSCFG in mid-operation once it has been configured after system reset. The SYSCFG should not overlap with any other bank.

If a start address of SYSCFG has changed, other control registers in the SFRs will have a new start address, which is its offset address + the new address of SYSCFG. For example, after the system reset, the initial address of SYSCFG is 1000000h and ROM control register has initial address 1001000h, because the ROM control register has the offset address value, 1000h, and its initial address is the sum of 1000000h + 1000h. If the SYSCFG address is changed to 1800000h, the ROM control register address becomes 1801000h.

Cache Disable/Enable

KS32C65100 Cache memory provides the programmable Cache disable and enable feature. It also provides a non-cacheable area feature to maintain data coherency for specific memory areas. Programmers can disable or enable the cache by setting the CE bit to 0 or 1. Programmers should be cautious about data coherency when cache memory is re-enabled because cache memory doesn't have an auto-flushing mode. Programmers also have to be cautious about DMA changes the memory data. Usually, the DMA access memory area must be non-cacheable to keep data coherency.

To keep the data coherency between the cache and external memory, KS32C65100 uses a write-through policy. To compensate for the performance degradation due to the "write through policy", there is internal 4 depth write buffer. A detailed description will be given in Chapter 5.

Write Buffer Disable/Enable

KS32C65100 has four Write Buffer Registers to enhance the memory writing performance. Its operation mode is programmable. When Write buffer mode is enabled, CPU writes data into write buffer first instead of an external memory which requires longer memory write cycles. The write buffer has 4 registers and each register includes 32 bits of data field, 25 bits of address field and 2 bit of status field.

Stall Disable/Enable

When the stall option is enabled, the MCU core logic inserts a wait when non-sequential memory accesses occur. So, the MCU core has more time margin during memory access. When the stall option is disabled, the logic doesn't insert a wait, so that's faster than when the stall option is enabled.

ROM CONTROL REGISTER

The KS32C65100 ROM interface has one ROM bank for program memory and it provides configurable features such as access timing, access size and page mode support, etc. The ROM Control Register (ROMCON) in SMR supplies the control mode such as normal mode access, page mode access and wait cycles of each mode, for the external ROM bank.

The initial address of ROMCON is 01001000h and it is the sum of the initial address of SYSCFG (01000000h) and the ROM control register offset address (00001000h). The register address is re-configurable that programmers can change the ROM control register by changing the contents of SYSCFG. The real address of ROM control register is "SYSCFG address" + "Offset address" of the ROM control register.

Register	Offset Address	R/W	Description	Reset Val.
ROMCON	0x1000	R/W	ROM control register	0x02003002

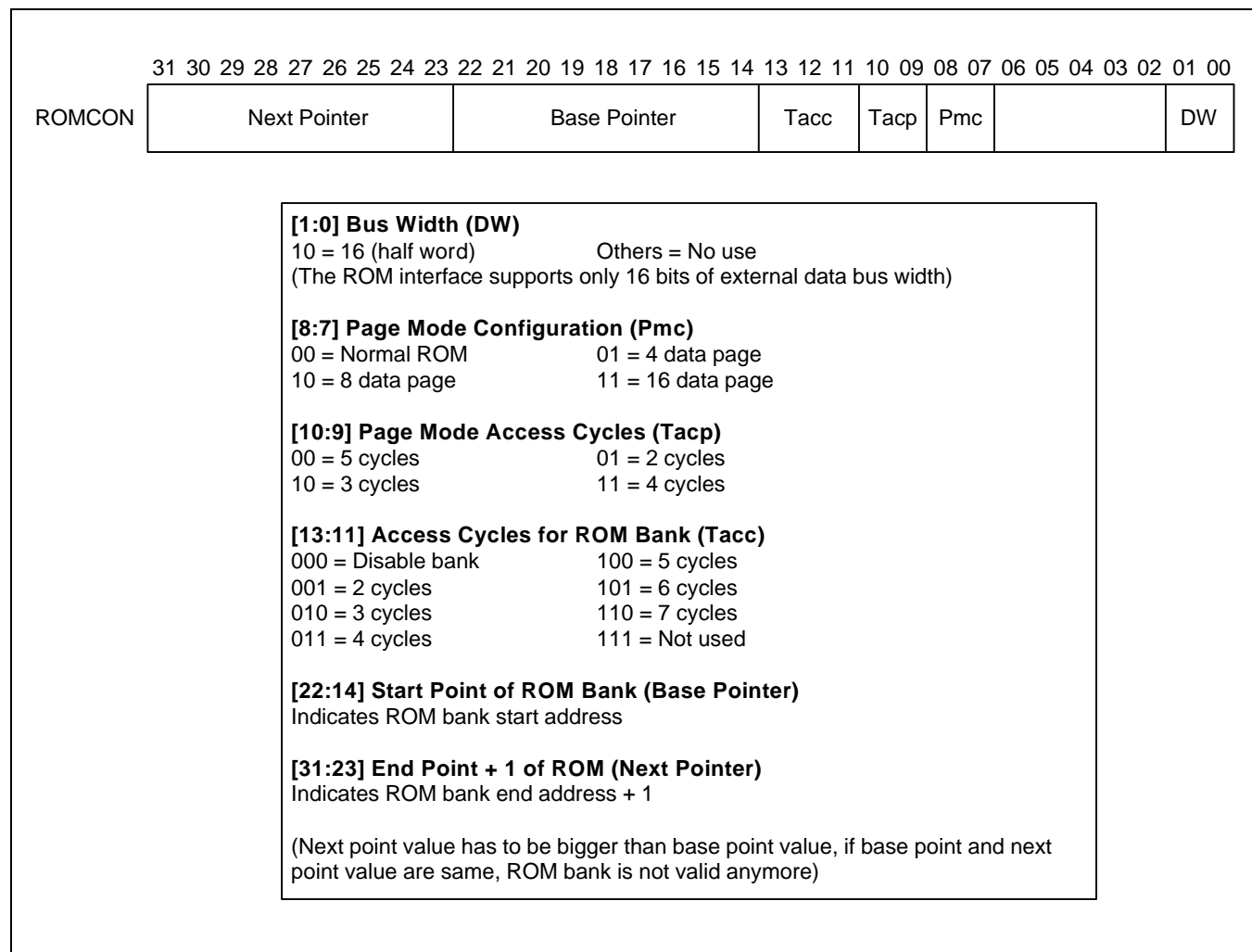


Figure 4-4. ROM Control Register (ROMCON)

Page mode ROM Access (Burst mode Access)

KS32C65100 ROM can be interfaced with simple ROM and page mode ROM. Programmers can make a burst mode enable or disable and can define the readable number of burst data by using ROMCON[8:0], ROM has two different access cycles for simple ROM and page mode ROM. When a new bank has selected, first data access time will be different from the access time of the following data of the same bank. Tacc, access cycles for ROM bank, is defined as the access cycles after the ROM bank changes. This cycle time is also used for simple ROM access mode. When CPU reads consecutive data within same bank, page mode ROM supplies data read cycles shorter than reading the simple ROM or new bank access mode. The Tacp bit in ROM control register defines consecutive data read cycles in page mode ROM.

Writes to the ROM space

KS32C65100 ROM interface provides write feature. Users can write data into ROM bank area. Physically, the internal program in ROM is not to be changed. So, if a user puts external memory instead ROM such as SRAM, flash memory, etc., it is possible to write data.

ROM Bank Space

One of good features of KS32C65100 is to have the configurable memory space. Users can program the memory bank size and bank location by modifying the contents of the ROM control register(ROMCON). ROM control register has two 9 bits address pointers, base and Next pointer. These two pointers denote the beginning and ending address of ROM bank. These 9 bits are mapped to the address [24:16], which means that bank address can be configured by 64KB range. The next pointer contents should be ROM bank end address + 1.

Initially, ROM bank start address is 00000000h and end address is 00FFFFFFh Therefore, Next pointer values must be 00FFh + 1h = 0100h. If ROM next pointer and base pointer values are same, then ROM bank will be disabled.

Initialization

When system has been initialized, the initial value of ROM control register is 80003002h and it specifies that the external bus width is 16 bit (half word), normal ROM mode is enabled and the longest page mode access cycles are selected.

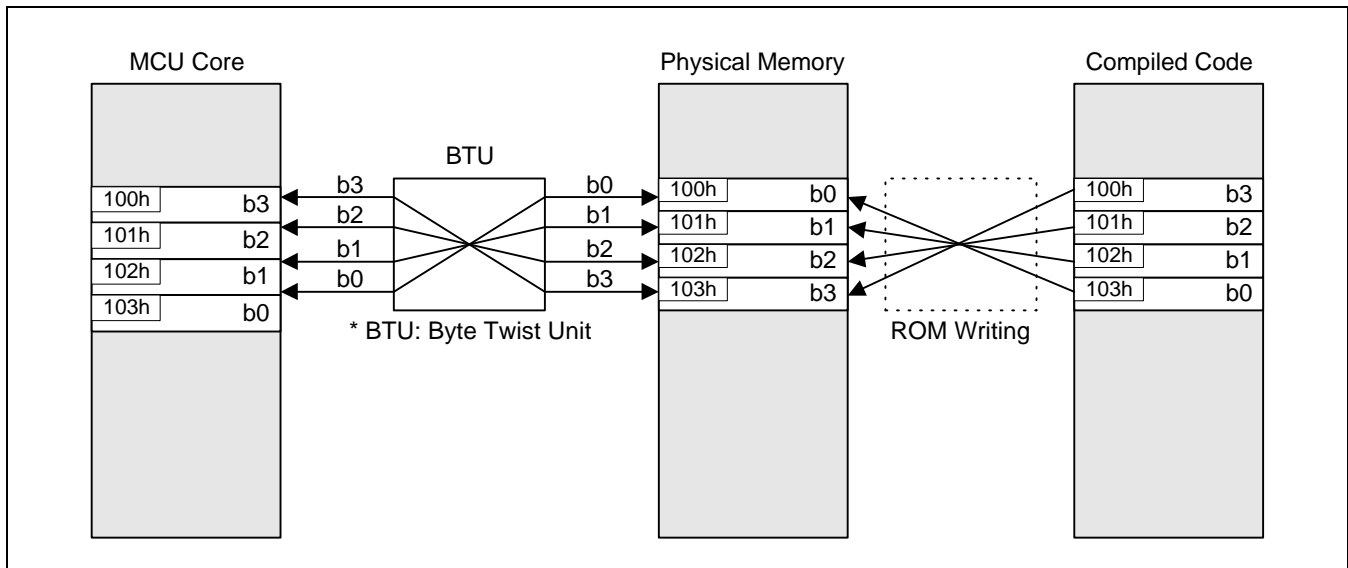


Figure 4-5. The Byte Swap Operation of BTU and the Positions of Data in Memory

ROM Programming

- Big endian supporting core and little endian supporting physical memory
KS32C65100 core and the internal peripherals support Big-endian configuration, while external memories like ROM, SRAM, and DRAM can have Little-endian configuration. Instead of having Big-endian physical memory configuration, there is BTU (Byte twist unit) in KS32C65100, internally. The main role of BTU is to swap the bytes in word as shown in Fig 4.5. In other word, when core access "11" byte, it can get the "00" byte from physical memory. To put the Big-endian data in Little-endian memory, Compiled code with the option of Big-endian has to put in memory by swapping byte in a word as shown in Fig 4.5 due to the double swapping (BTU and compiled code swapping), KS32C65100 can support Big-endian mode without any problem. The reason why we have double swapping, is due to internal H/W implementation issue.
- Big endian format/little endian format
In Big Endian format, the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte. Byte 0 of the memory system is therefore connected to data lines 31 through 24
- In little endian format, the lowest numbered byte in a word is considered the word's least significant byte, and the highest numbered byte is the most significant. Byte 0 of the memory system is therefore connected to data lines 7 through 0.

— Byte swapping in a word

The byte swap is done by using following simple C code. It changes the byte sequence in a word.

```
Unsigned int swap (unsigned int data) // Make the sequence of bytes reverse in a word
{
    return  ( (0xff000000 & data)>>24)+
            ( (0x00ff0000 & data)>>8)+
            ( (0x0000ff00 & data)<<8)+
            ( (0xxxxxxx & data)<<24) );
}
```

— ROM writing

BTU changes the sequence the byte in a word, program codes are byte-swapped. To write the program to ROM, do steps as follows;

1. Compile the program by big endian mode
2. Byte-swap the compiled code
3. Writes the code to ROM.

— Little endian format code vs. Byte swapped big endian format code.

if character strings doesn't exist in programs, little endian format codes may be same as byte-swapped big endian format codes. But, because the bytes in a string is not affected by whether little endian format or big endian format, the two codes are not same. So, the big endian format code byte-swapped has to be used in KS32C65100. if little endian format code is used, the strings are not displayed correctly. (byte swapped strings may be displayed)

— Interfacing external peripherals

Peripherals address is also byte-swapped. For example, If users want to access address 0h in memory, address 3h in MCU must be accessed. This is because of word swapping of BTU. The relation between physical address and the address used by instructions is as follows;

Table 4-1. The Relations Between Physical Address and Address in Instructions

Physical Address	Byte Wide Access (Address Used in Instructions)	Half Word Wide Access (Address Used in Instructions)
00b	11b	10b
01b	10b	N.A.
10b	01b	00b
11b	00b	N.A.

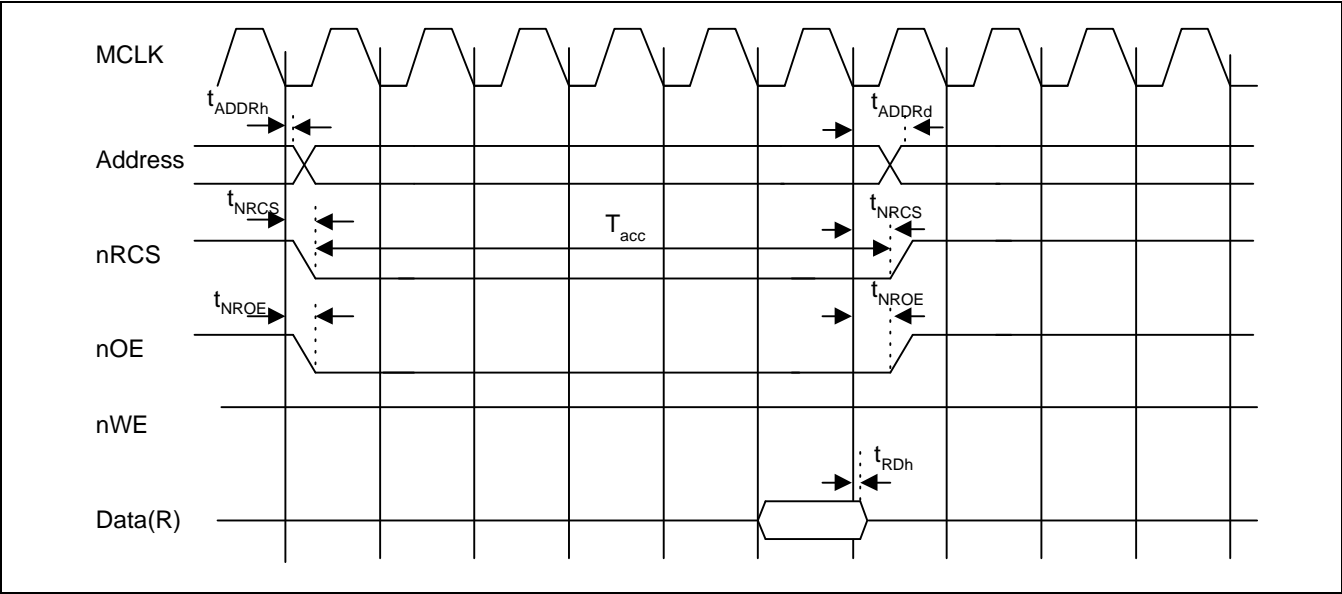


Figure 4-6. Simple ROM Access Timing

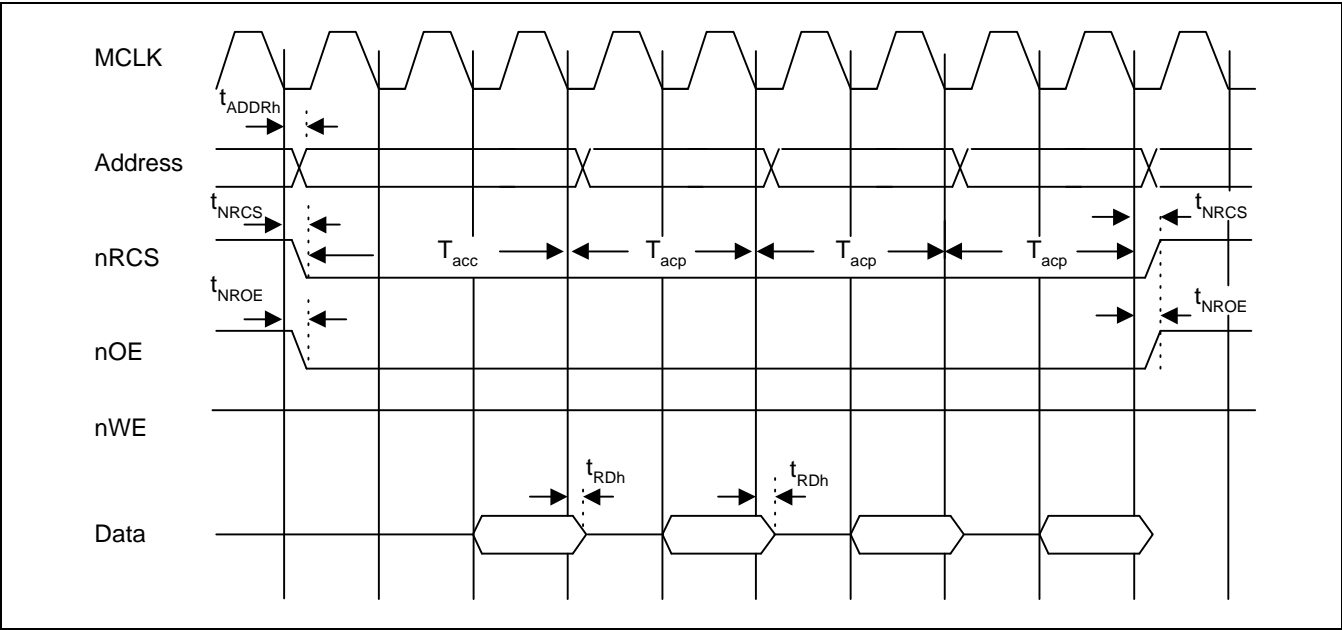


Figure 4-7. Page Mode ROM Access Timing

SRAM CONTROL REGISTERS

KS32C65100 SRAM interface has two banks of SRAM and each bank is able to set up own SRAM access configuration. The SRAM Control Registers (SRAMCON0, SRAMCON1) in SMR specifies not only the features for SRAM banks but also two Special I/Os (I/O0, I/O1) in the external bank 3.

The initial addresses of SRAM control registers are 01001004h and 01001008h, each. The real address of each SRAM control register is "SYSCFG address" + "Offset address" of each SRAM control register. The register address is re-configurable and programmers can change the SRAM control register address by changing the contents of SYSCFG.

Registers	Offset Address	R/W	Description	Reset Val.
SRAMCON0	0x1004	R/W	SRAM control register 0	0x000007fc
SRAMCON1	0x1008	R/W	SRAM control register 1	0x000007fc

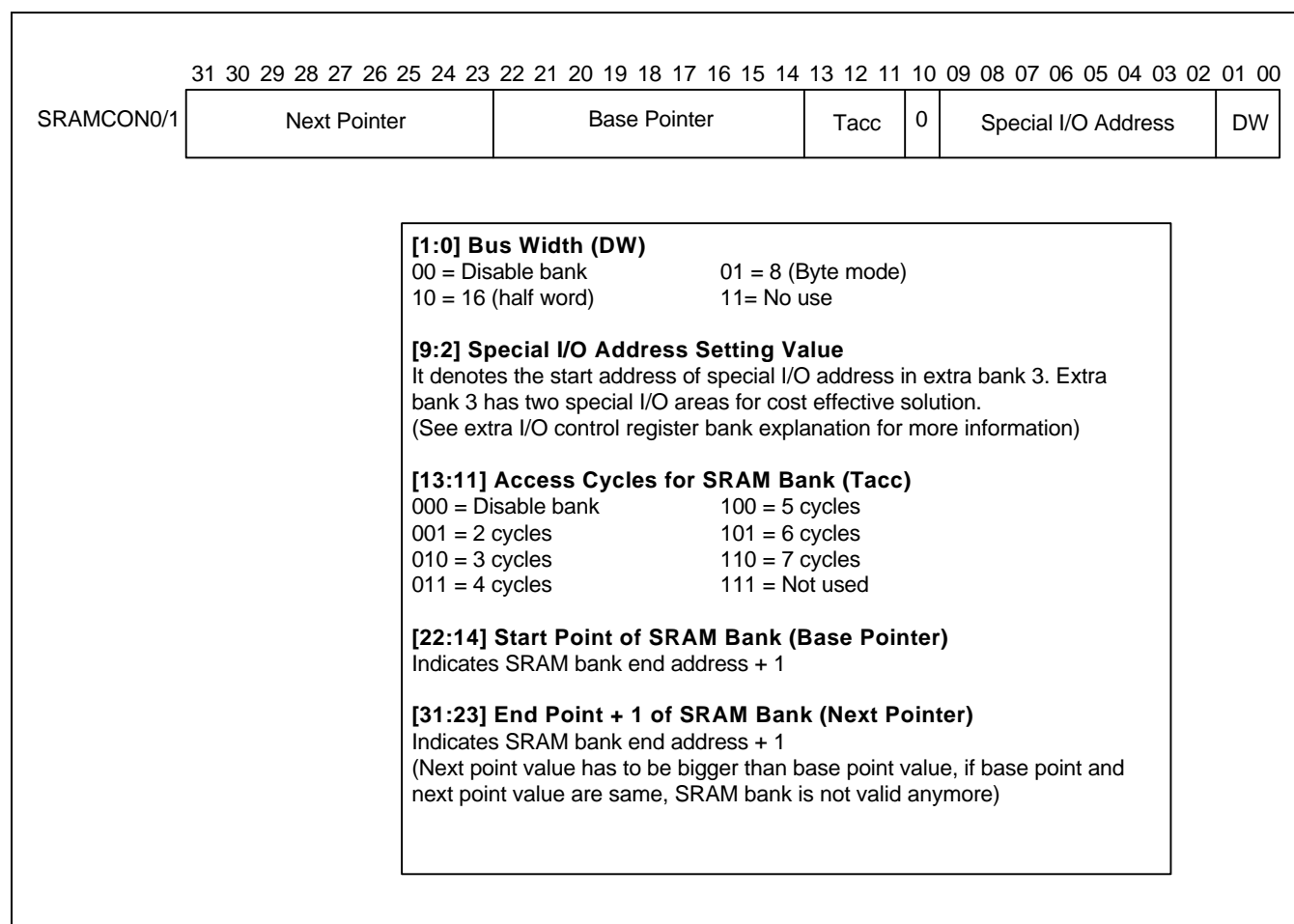


Figure 4-8. SRAM Control Registers

SRAM Bank Space

KS32C65100 SRAM interface provides two SRAM banks, each of them are able to have different configuration. Users can program the SRAM access cycles, memory bank size and bank location by using two identical SRAM control registers, SRAMCON0,1. SRAM control register has two 9 bits address pointers, base and Next pointer. These two pointers which denote the start and end address of SRAM bank. These 9 bits are mapped to the address [24:16]. Therefore, bank address offset value is 64K byte (16 bits). The next pointer contents should be SRAM bank end address + 1.

Initially, Two SRAM banks start and end addresses are 00000000h. Therefore, SRAM banks are disabled after system initialization because the next pointer and base pointer have same values

Initialization

When system has been initialized, two SRAM Control register initial values are 00000000h and it specifies the external SRAM is disabled.

Special I/O Address

The extra bank 3 of KS32C65100 has two special I/O areas for making out the simple external latch control signal. Two SRAM control registers have dedicated 9 bits for those special I/O areas in the extra bank 3. Extra bank 3 provides two special control signals, nLORD0, nLOWR0. When a user reads/writes data from/to external latch devices, these signals doesn't need additional address decoding logic. These signals are only available at the extra bank 3. When MCU accesses any of special I/O address area (64kB, 16 bit offset address) specified by SRAM control registers, extra bank interface logic generates a I/O read and write signals for the corresponding address area. Fig 4-20 shows the diagram of special I/O read/write interface logic.

Address Bus Generation

The address bus of KS32C65100 is some different from general MCUs. When 8 bit data bus is selected, the resolution of address bus is a byte. When 16bit data bus is selected, the resolution of address bus is a half word. So, although general MCUs don't use A0 pins at 16bit data bus width, KS32C65100 always uses A0 pins regardless of bus width.

Data Bus Width	External Address Pins (ADDR[21:0])	Reset Value
8 bit	A21-A0 (internal)	4M byte
16 bit	A22-A1 (internal)	4M half word

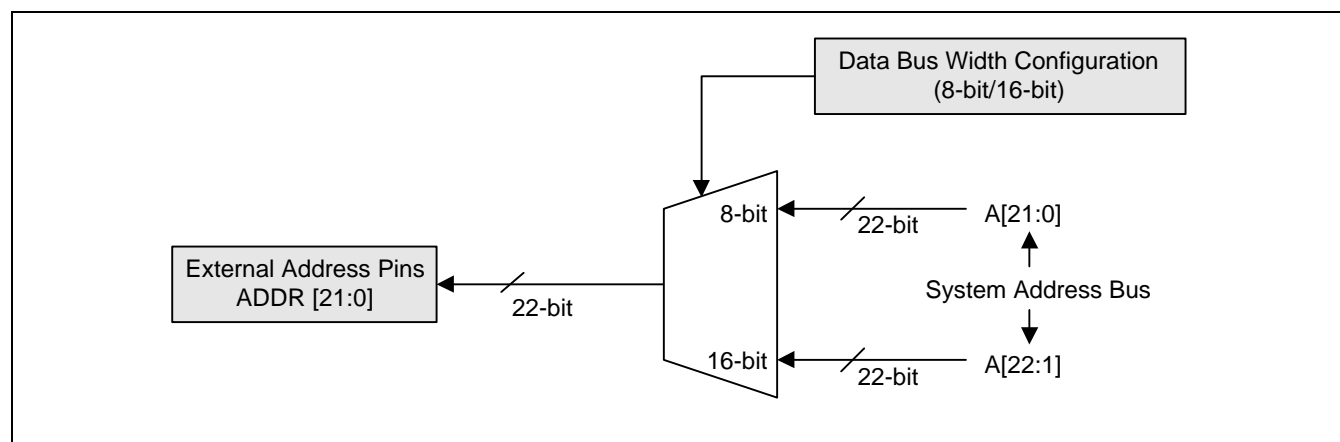


Figure 4-9. External Address Bus Generation (ADDR[21:0])

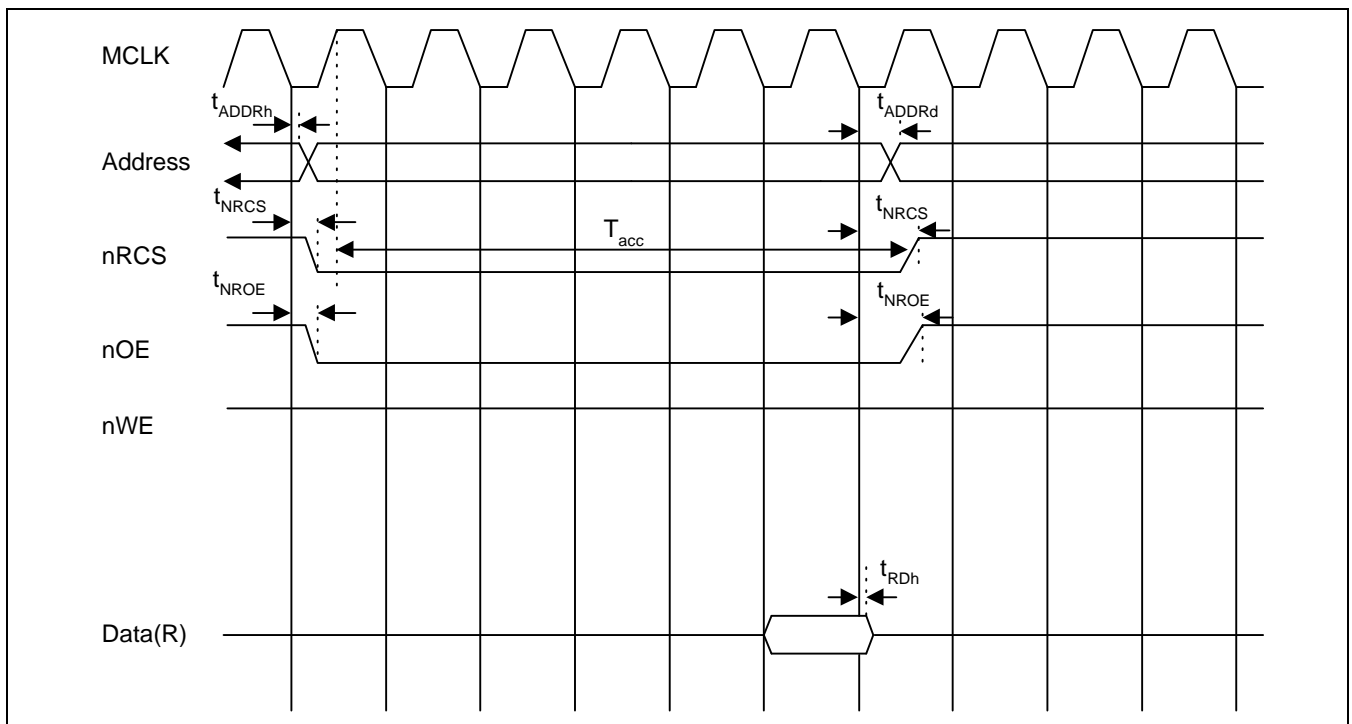


Figure 4-11. SRAM Read Timing

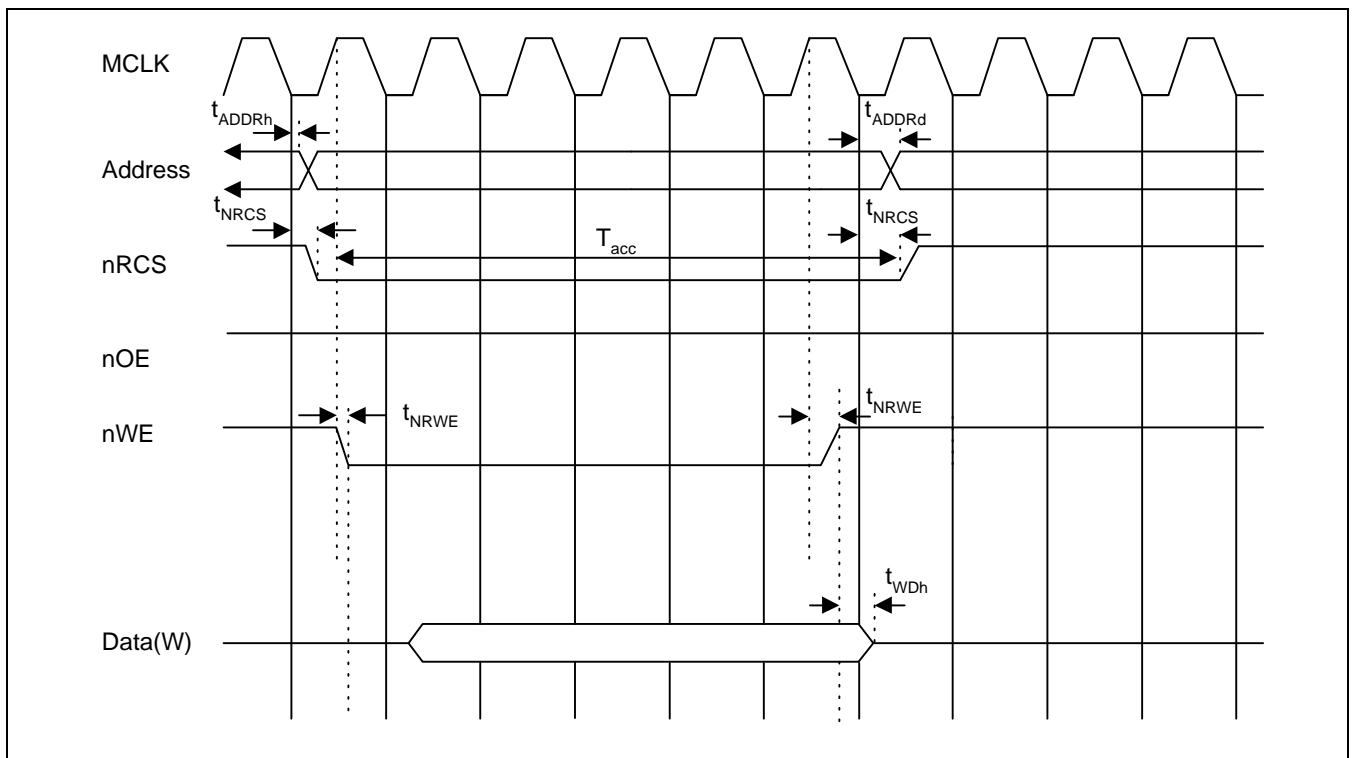


Figure 4-12. SRAM Write Timing

DRAM CONTROL REGISTERS

KS32C65100 DRAM interface has two banks of DRAM and each bank is able to control DRAM access timing as memory configurations. The DRAM interface has two DRAM control registers, DRAMCON0/1 and one DRAM refresh control register, REFCON. The initial addresses of each DRAM control registers are 0100101ch and 01001020h. The refresh control register address is 01001024h. The register address is re-configurable and programmers can change the address of DRAM control register by changing the contents of SYSCFG.

Registers	Offset Address	R/W	Description	Reset Val.
DRAMCON0	0x0000101c	R/W	DRAM 0 control register	0x00000000
DRAMCON1	0x00001020	R/W	DRAM 1 control register	0x00000000

The KS32C65100 provides fully programmable external DRAM interface features. Programmers can easily modify the interface modes such as external data bus width, number of access cycles for fast page or EDO, access cycles for each DRAM bank and row address strobe (nRAS) pre-charge timing by changing the contents of corresponding DRAM control register. The refresh control register controls DRAM refresh operation and KS32C65100 supports CAS before RAS (CBR) refresh mode & self refresh mode.

KS32C65100 can generate row & column address and supports symmetric/Asymmetric address DRAM by changing the number of address line from 8 to 11. It can support various size of DRAM by varying column address size. If the number of a column address or a row addresses is bigger than 11, the accessible DRAM memory size is smaller than the original size of the DRAM. For example, if 16M-bit DRAM with 4Mx4 (row address = 12bit & column address = 10bit) is connected to KS32C65100, the maximum accessible size of the memory is 8Mbit (11bit × 10bit) and the other 8Mbit will be obsolete.

EDO mode DRAM Accessing

Even If users specify DRAM as EDO mode, KS32C65100 gives same timing diagram compared with normal fast page mode. However, KS32C65100 CPU fetches data (when read) later by a half clock than normal fast page mode. It is possible because EDO mode can make data valid even if CAS goes to high when RAS is low. So, it can give enough time to spare for CPU to access and latch the data so that it can reduce memory access cycle time, eventually.

DRAM Bank Space

KS32C65100 DRAM interface provides two DRAM banks and each of them are able to have different configuration. Users can program the DRAM access cycles, memory bank size and bank location by using two identical DRAM control registers, DRAMCON0&1. DRAM control register has two 9 bits address pointers, base and next pointer. These two pointers which denote begin and end address of DRAM bank. These 9 bits are mapped to the address [24:16]. Therefore, bank address offset value is 64K byte (16 bits). The next pointer contents should be DRAM bank end address + 1.

Initially, Two DRAM banks start and end addresses are 00000000h. Therefore, DRAM banks are disabled after system is initialized. because next pointer and base pointer values are same .

Initialization

When system is initialized, two DRAM control register initial values are 00000000h and it specifies mode that the external DRAM is disabled.

DRAM Bank Configuration

The DRAM has different write methods from SRAM or other external memories. Normally, DRAM module has two CAS signals to separate data bus by byte order. Therefore, RAS signal is used for bank selection and CAS signal is used for byte selection mode.

Example) Settings for 60nS EDO DRAM (KM416V1204)

Condition	Setting Value for DRAMCON
Memory map: 1000000h ~ 11ffffh DRAM: 10bit(row) × 10bit(column) × 16bit(data), 60ns, EDO MCLK: 33MHz	0x9040101a

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00																																	
DRAMCON0/1		Next Pointer								Base Pointer								Trp		Trc		Tcs		Tcp		Tpgm		EDO		CAN		DW	

Figure 4-13. DRAM Control Registers (DRAMCON0 - DRAMCON1)

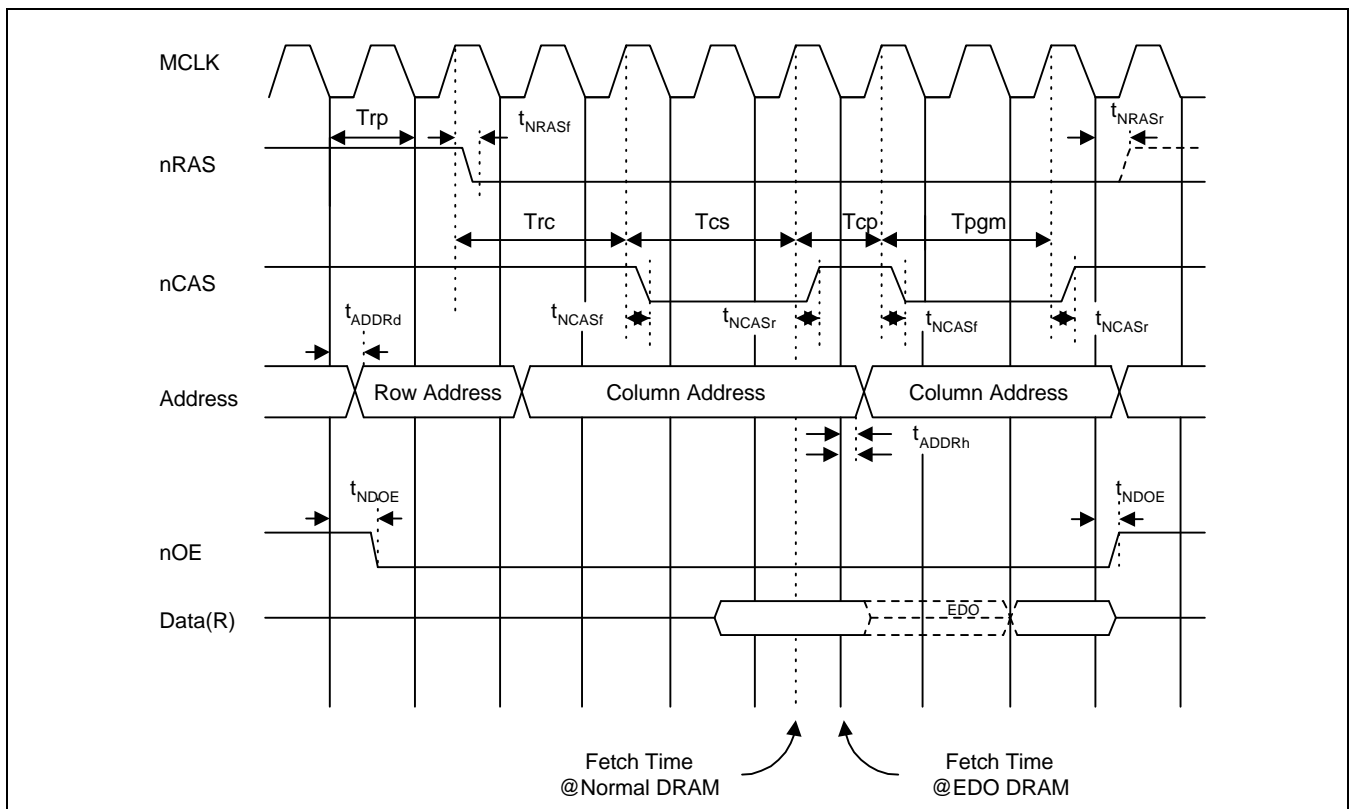


Figure 4-14. DRAM Bank Read Timing (Page Mode)

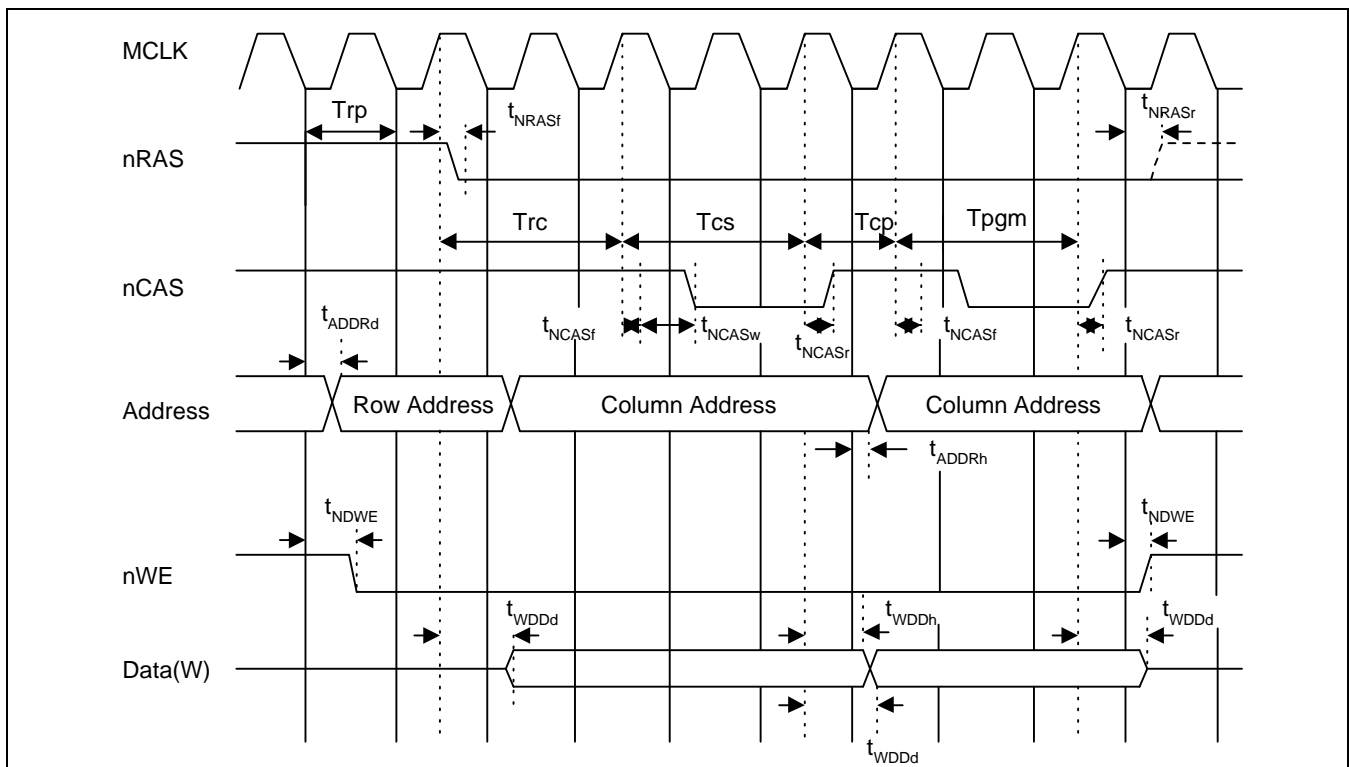


Figure 4-15. DRAM Bank Write Timing (Page Mode)

DRAM REFRESH CONTROL REGISTER

The KS32C65100 DRAM interface provides the CAS before RAS (CBR) refresh and self refresh mode. The refresh control register (REFCON) determines refresh mode, refresh timings, refresh intervals as well as external bus enable.

Register	Offset Address	R/W	Description	Reset Value
REFCON	0x00001024	R/W	DRAM refresh control	0x00000001

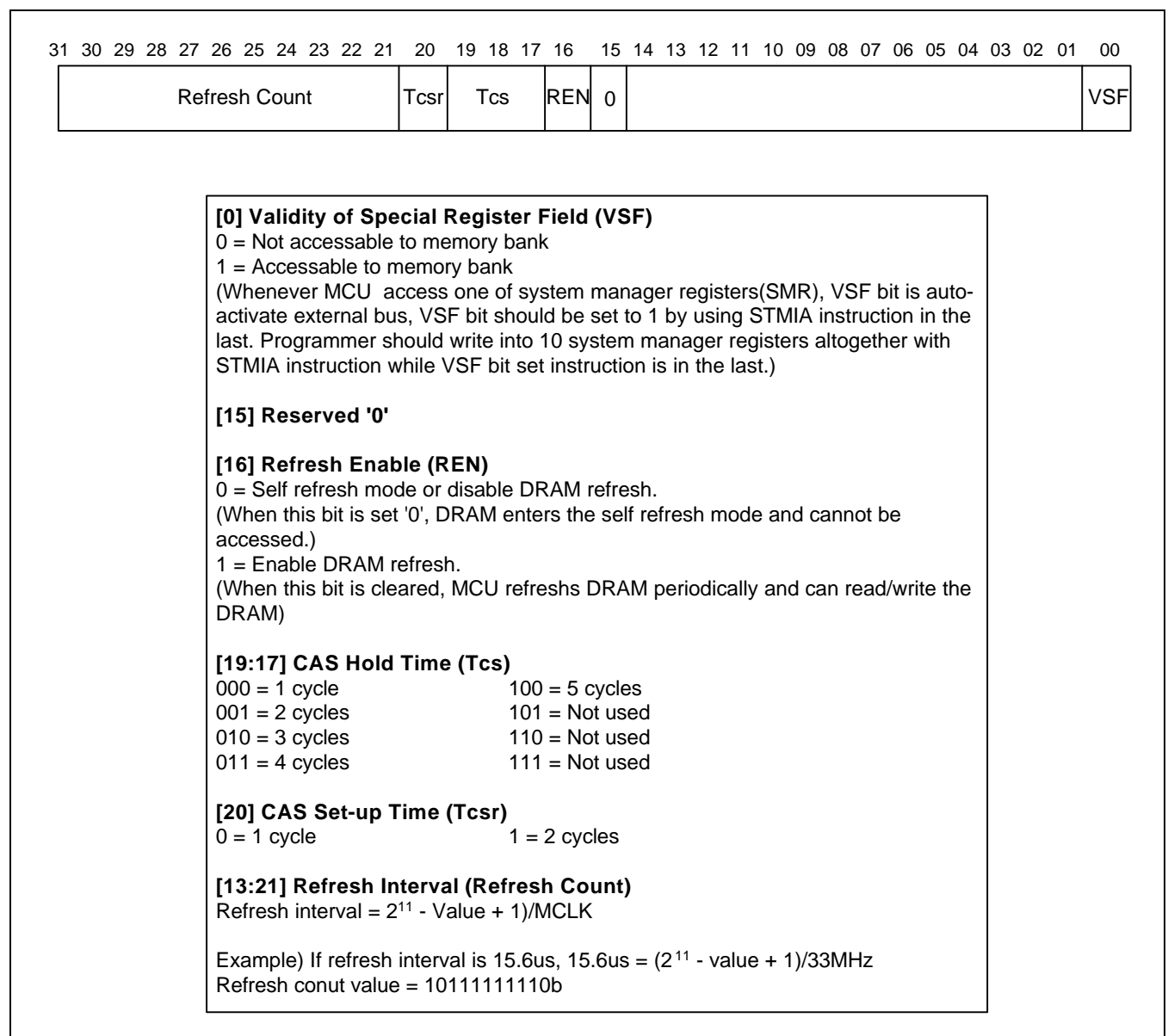


Figure 4-16. DRAM Refresh Control and Memory Configuration Register (DRAM Refresh Control)

DRAM Self refresh mode

Every DRAM requires refresh operation periodically to keep correct data and JEDEC defines couple of refresh modes. The self refresh mode is one of which has defined in JEDEC specification and it enables the DRAM to refresh memory cells internally once it has enabled without periodical external refresh control signals unless other refresh mode happens or power fails.

The self refresh operation is similar to that of CBR (CAS before RAS). Once after CPU generates CBR mode signals and it keeps CBR mode state more than 100us, DRAMs recognize refresh mode as self refresh instead CBR.

DRAM Self refresh mode Entry

1. Self Refresh mode by Hardware

When system reset pin, nRESET, is low, the system manager block generates self refresh mode signals, i.e. whenever KS32C65100 initialized, it activates self refresh mode. Hardware refresh feature enables the system to avoid DRAM data loss if system backup supplies power to DRAM while main power is disconnected.

When system main power is disconnected, KS32C65100 will be disabled. Meanwhile, if DRAM has power back-up circuitry, it still requires periodical refresh signals from KS32C65100. Therefore, it won't be able to keep valid DRAM data in a short time, if KS32C65100 does not make DRAM self refresh mode.

For this reason, when main power is disconnected and nRESET goes low, KS32C65100's system manager block makes self refresh signals. The system user can make memory back up system easily by utilizing this feature, if only DRAM is used for system memory.

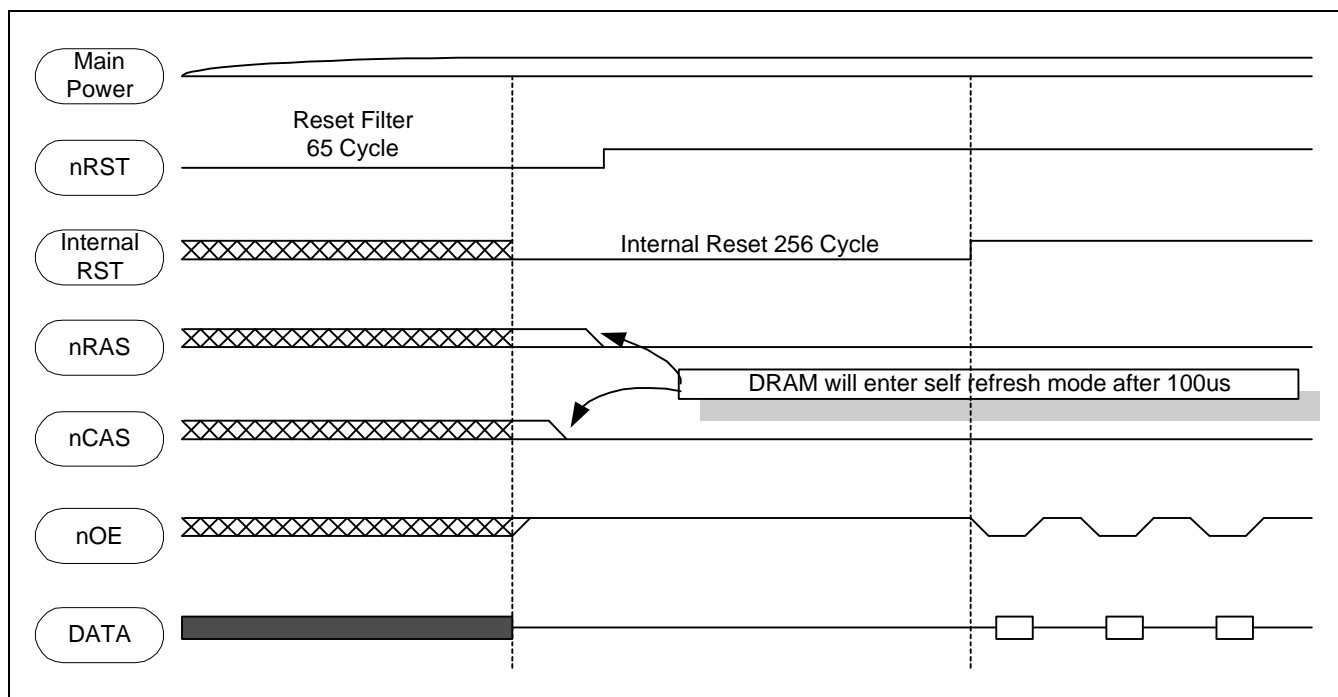


Figure 4-17. Self Refresh Mode Entry Process by nRESET

2. Self Refresh mode by Software

After system reset, KS32C65100 is in DRAM self refresh mode. By programming the REN bit of DRAM refresh control register to "1", system manager block works as normal DRAM access mode.

To enable the self refresh mode during normal system operation, programmer needs to change the REN bit to "0". system manager detects the REN bit content change from 1 to 0 and it activates the self refresh mode. If programmer wants change mode from self refresh mode to normal DRAM access mode, programmer just needs to write "1" to REN bit once again.

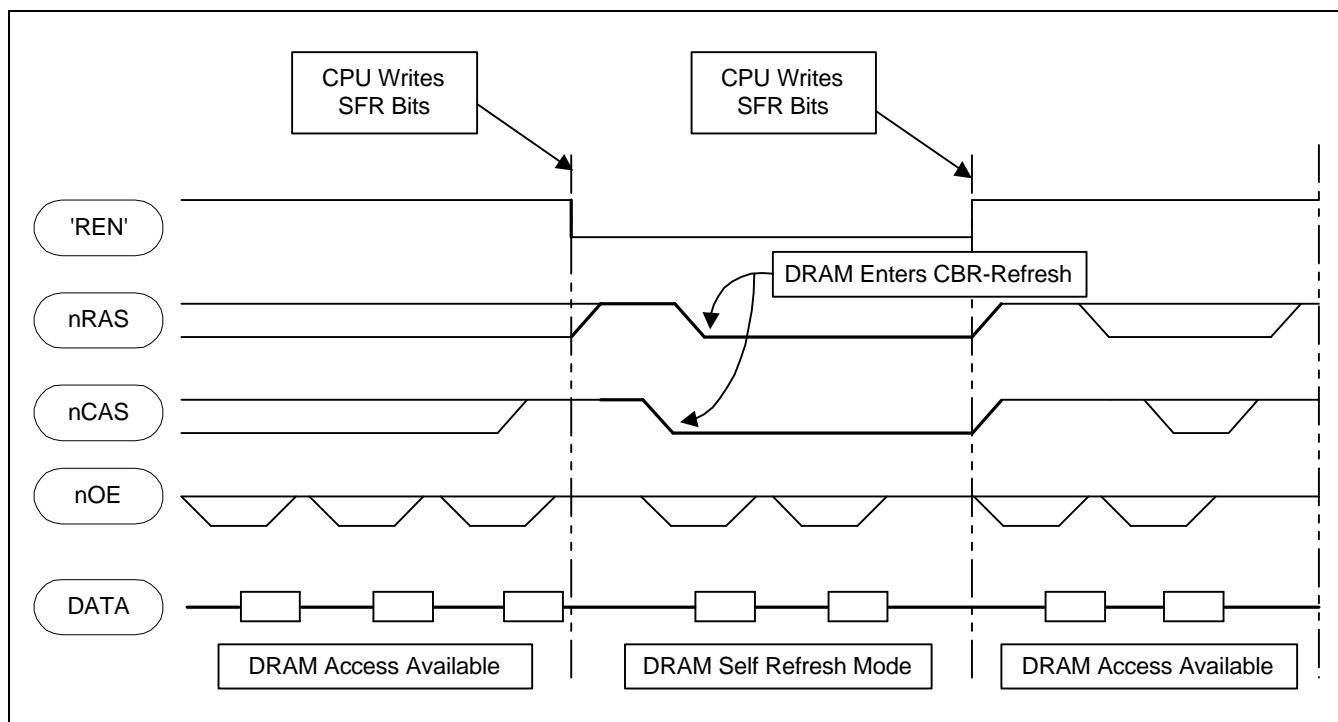


Figure 4-18. Self Refresh Mode Entry Process by Software

NOTES:

- When DRAM does not work self refresh after system initialization.
Even though KS32C65100 activates self refresh mode when system power connected, DRAM may not recognize self refresh mode correctly, because of unstable state of control signals during system initialization (Most of DRAM recognize the self refresh mode very well at power on). When it happens, KS32C65100 may fetch corrupted data from external memories because DRAM and ROM share OE (Output Enable) signal and they may generate data altogether.

KS32C65100 has a watch dog timer to cope with system malfunction problem. When KS32C65100 initialized, watchdog timer is enabled and makes external system reset signal unless MCU disables in the mid of operation. Therefore, it is recommended to put the watch dog timer disable code in the boot ROM area to disable watch dog timer. If "power on initial" is not working correctly and KS32C65100 fetches corrupted data, watch dog timer will make system reset signal and it will cause KS32C65100 reset once again. The second watch dog reset will cause DRAM self refresh mode because when it happens, system power and other states are stable.
- DRAM access during self refresh mode
If KS32C65100 accesses external DRAM for read or write data during self refresh mode, **nRAS** and **nCAS** signals are not working at all. DRAM accessing during the self refresh mode may cause corrupted data read or writing.

3. Memory access is forbidden when the SMR is changed.

The external bus is disabled when MCU accesses any of SMR to change the system memory configurations. It is for preventing the system malfunction which will be caused by memory address space overlapping during the new configuration. To re-activate external bus operation, The VSF bit in refresh register need to be set to 1 by writing SMRs with STMIA ARM instruction. While STMIA instruction writes 10 registers of SRMs, refresh register must be written at the last step with VSF bit has "1" so that external bus can be re-activated right after system manager register has new configuration.

It is not recommended to change the SFRs after system initialization. If SFR changed, especially memory related areas, users have to flush cache memory for the data coherency.

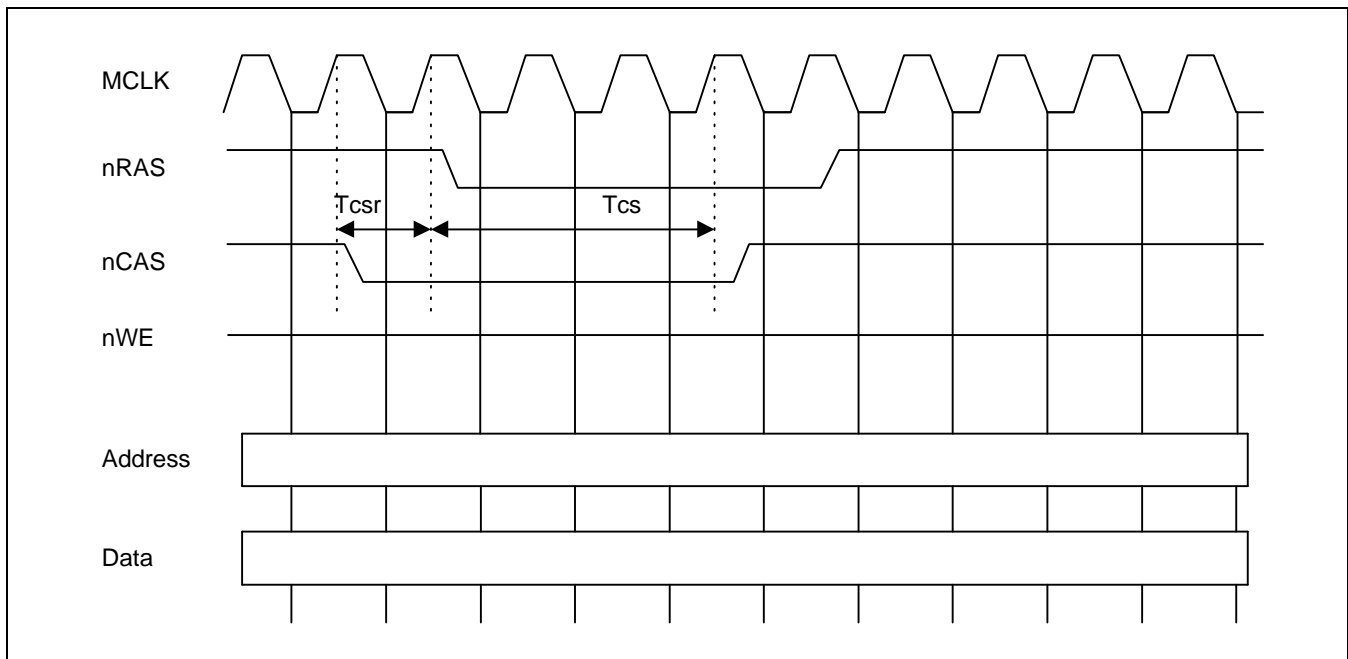


Figure 4-19. DRAM Refresh Timing

EXTRA BANK ACCESS CONTROL REGISTERS

The KS32C65100 provides four extra banks and four Extra Bank Control registers (EXTCONn) controls timing, bank size and bus width. The extra bank 3 has the special features compared with other extra banks. It has one special dedicated addresses (refer to SRAM0 control registers) for providing the low cost external I/O control solution. Extra bank 3 has special signals such as nIORD0, nIOWR0. When a user reads/writes data from/to external latch devices, these signals prevent extra address decoding logic ICs. These signals are available at only the extra bank 3. Basically, they have same timing diagram as the extra bank 3 has. The initial address of each I/O control registers are plus of its own offset address with initial SYSCFG register address, 01000000h.

Registers	Offset Address	R/W	Description	Reset Value
EXTCON0	0x100c	R/W	Extra bank 0 control register	0x00000000
EXTCON1	0x1010	R/W	Extra bank 1 control register	0x00000000
EXTCON2	0x1014	R/W	Extra bank 2 control register	0x00000000
EXTCON3	0x1018	R/W	Extra bank 3 control register	0x00000000

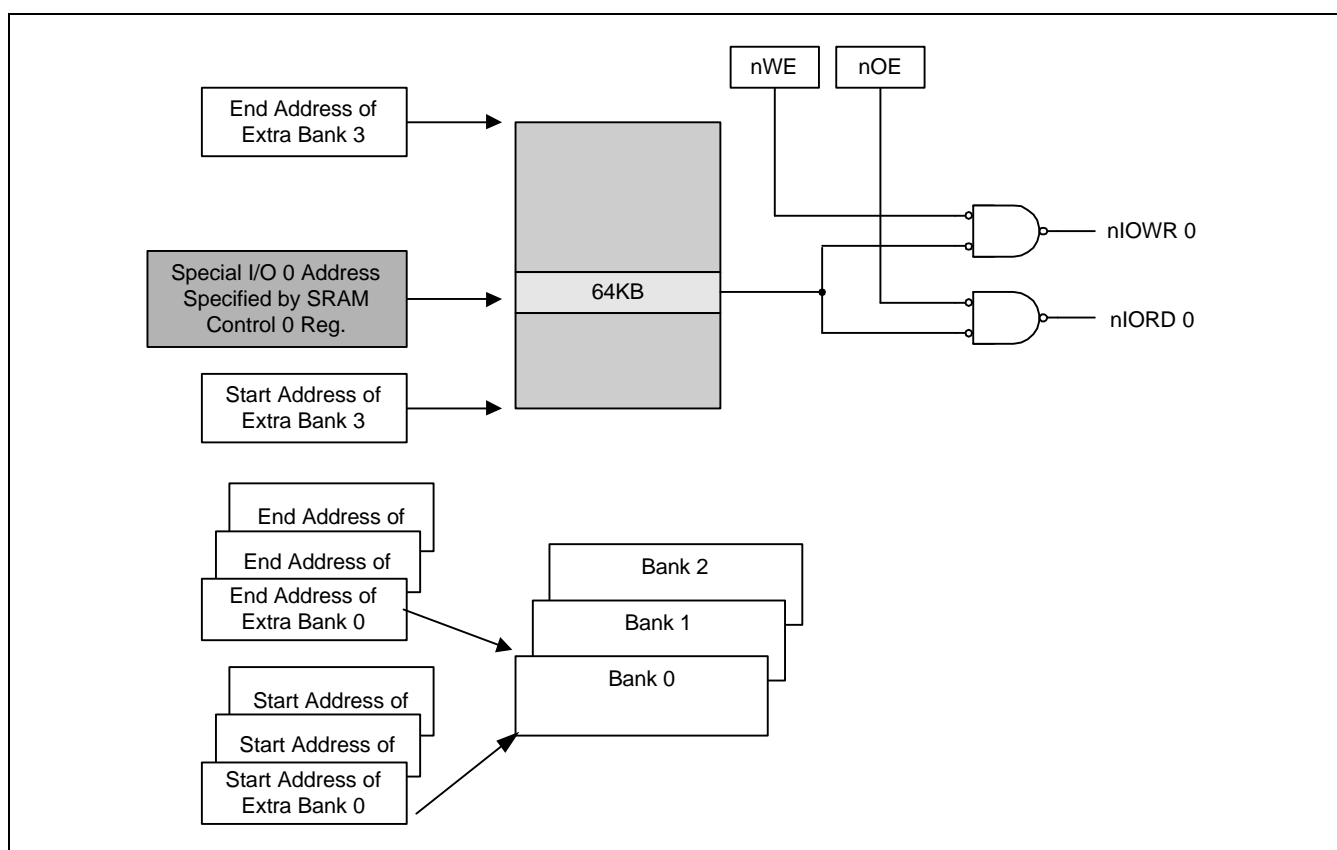


Figure 4-20. Special I/O Address Map

When fetching data, the point of data reading is the last down edge of MCLK within nECS active region. Users may be curious about the figure 4-22, nOE's de-asserting before the point of data reading. If nOE has to be de-asserted after the point of data reading, use 'tcoh' = 0 which defines the time between nOE's de-asserting and nECS's de-asserting. Setting 'tcoh' as 0, nOE is de-asserted after the point of data reading as the user wants.

In reality, extra bank 3 can't be configured. It is an imaginary bank for planning the special I/O.

Special I/O Address

One SRAM control registers have dedicated 8 bits each for the extra bank 3, for providing the low cost system solution. Bank 3 has special signals, nIORD0, nIOWR0. When a user reads/writes data from/to external latch devices, these signals prevent extra address decoding logic ICs. These signals are only available at the extra bank 3. When CPU access any of special I/O address area (64KB, 16 bit offset address) specified by SRAM control registers, extra bank interface generates a I/O read and write signals for the corresponding address area. Figure 4-22, 23 shows the timing diagram of special I/O read/write cycles.

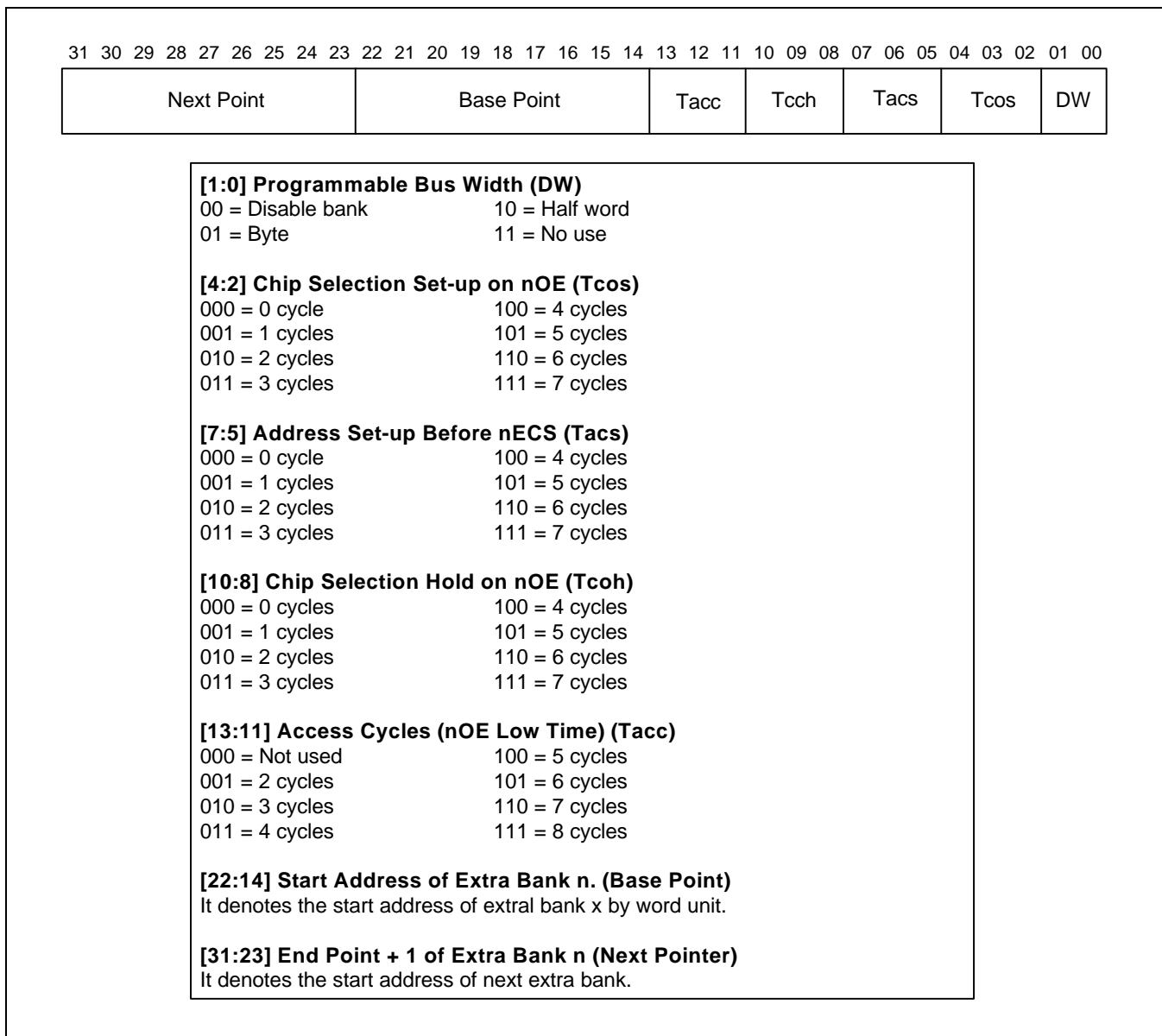


Figure 4-21. Extra Bank Control Registers (ExtCntr 0, 1, 2, 3)

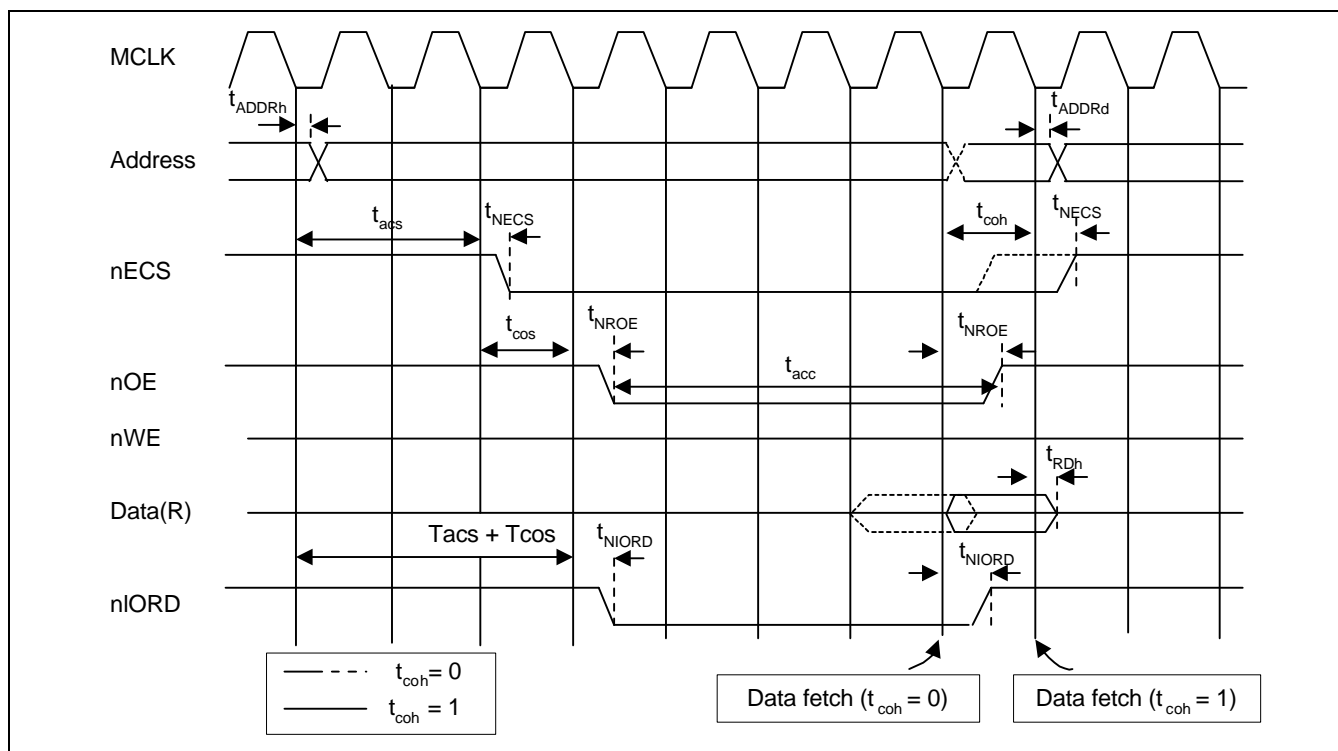
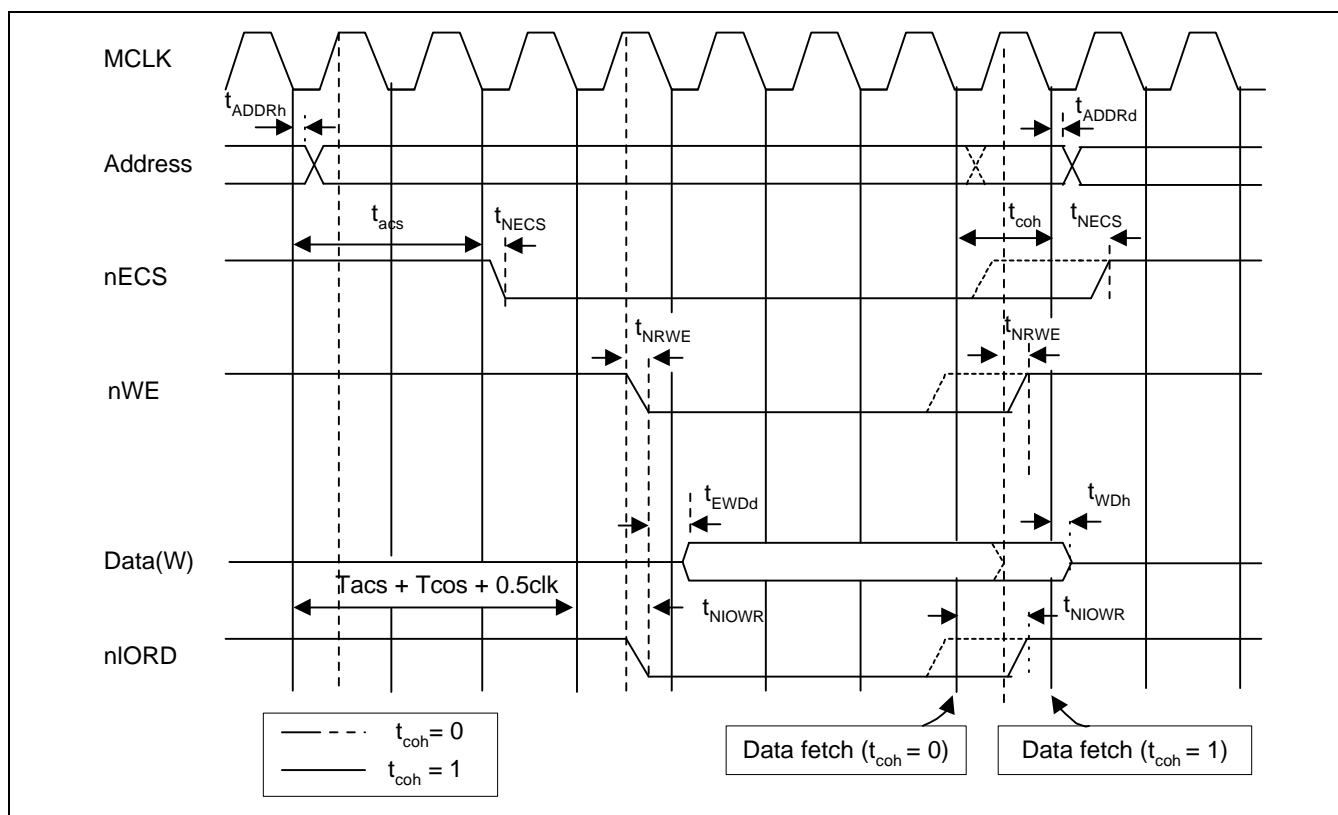
Figure 4-22. Extra Bank Read Timing ($t_{coh} = 1$, $t_{acc} = 4$, $t_{cos} = 1$, $t_{acs} = 2$)

Figure 4-23. Extra Bank Write Timing

A.C ELECTRICAL CHARACTERISTICS(Ta = 0 °C to +70 °C, V_{DD} + 3.00V to 3.60V)

Name	Description	Min	Max	Unit
tADDRh	Address hold time	7.0		ns
tADDRd	Address delay time		25.1	ns
tNRCS	ROM bank chip select delay time		20.6	ns
tNROE	ROM/SRAM/ExtIO bank out enable delay		23.5	ns
tNRWE	SRAM or ExtIO bank write enable delay		18.2	ns
tRDh	Read data hold time	3.0		ns
tWDd	Write data delay time (SRAM/EXIO)		9.8	ns
tWDh	Write data hold time (SRAM/EXIO)	26.3		ns
tNRASf	DRAM raw address strobe active delay		15.2	ns
tNRASr	DRAM raw address strobe release delay		27.0	ns
tNCASf	DRAM column address strobe active delay		16.1	ns
tNCASr	DRAM CAS signal release delay time		17.1	ns
tNCASw	DRAM CAS write active delay		19.8	ns
tNDWE	DRAM bank write enable delay time		24.4	ns
tNDOE	DRAM bank out enable delay time		23.5	ns
tNECS	External IO bank chip select delay time		20.6	ns
tNIORD	Special IO bank read signal delay time		23.5	ns
tNIOWR	Special IO bank write signal delay time		18.2	ns
tWDDd	DRAM write data delay time (DRAM)		14.2	ns
tWDDh	DRAM write data hold time (DRAM)	7.4		ns

Memory mapping for external memory and I/O is shown in Figure 4-24

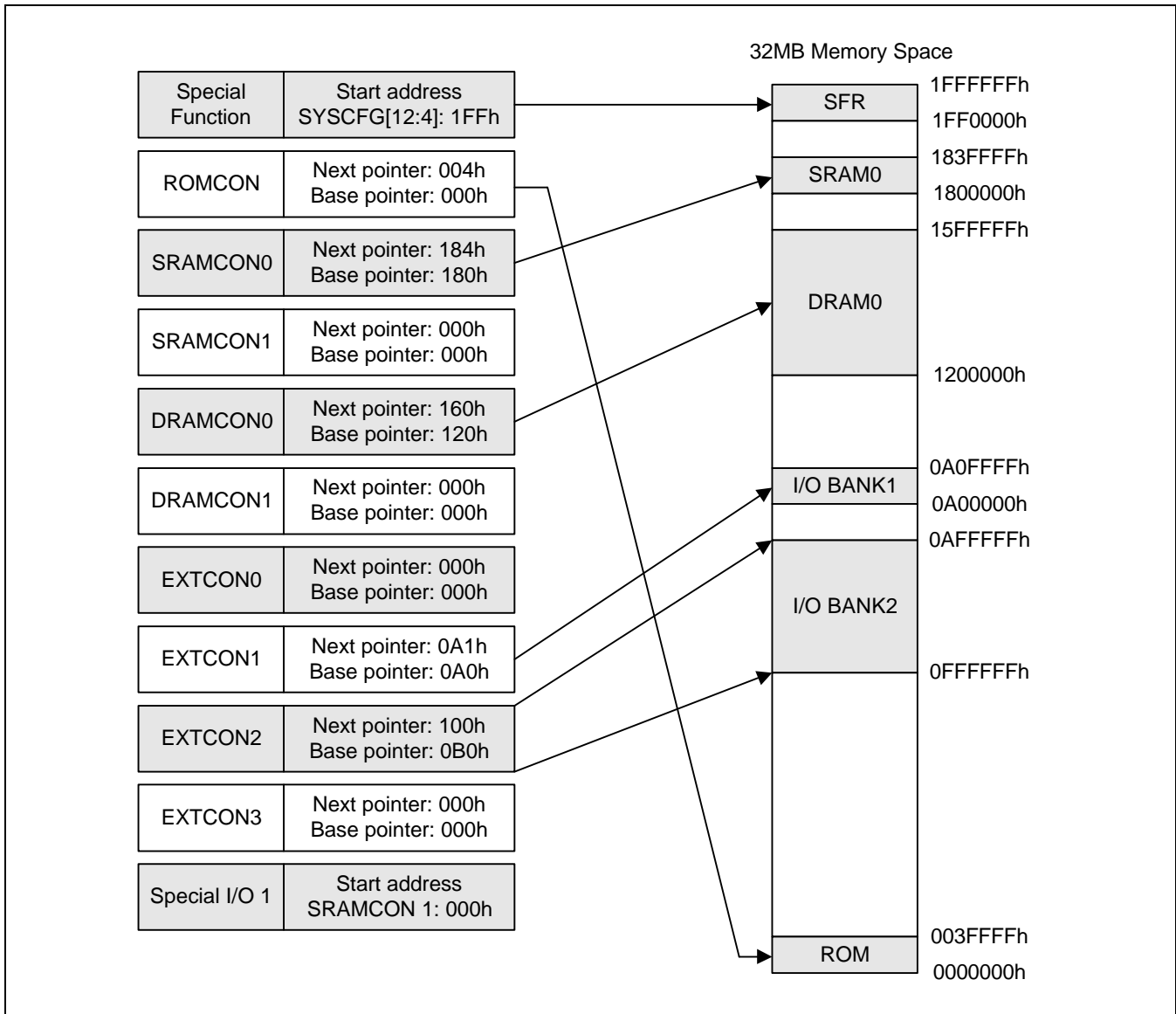


Figure 4-24. An Example of System Manager Register Settings

5

CACHE CONTROLLER

OVERVIEW

The KS32C65100 CPU has an internal 2K Bytes of unified (instruction/data) cache. The cache is two-way set associative and the line size is four words (16 Bytes). It has a write-through policy. When a miss occurs, words of memory are sequentially fetched from external memory. It has a LRU (Least Recently Used) replacing algorithm.

Typically, the RISC CPU uses instruction and data caches to improve performance. Without caches, the bottleneck that occurs during the instruction and data fetches from external memory may seriously degrade performance. The unified cache deals with instruction and data in the same way.

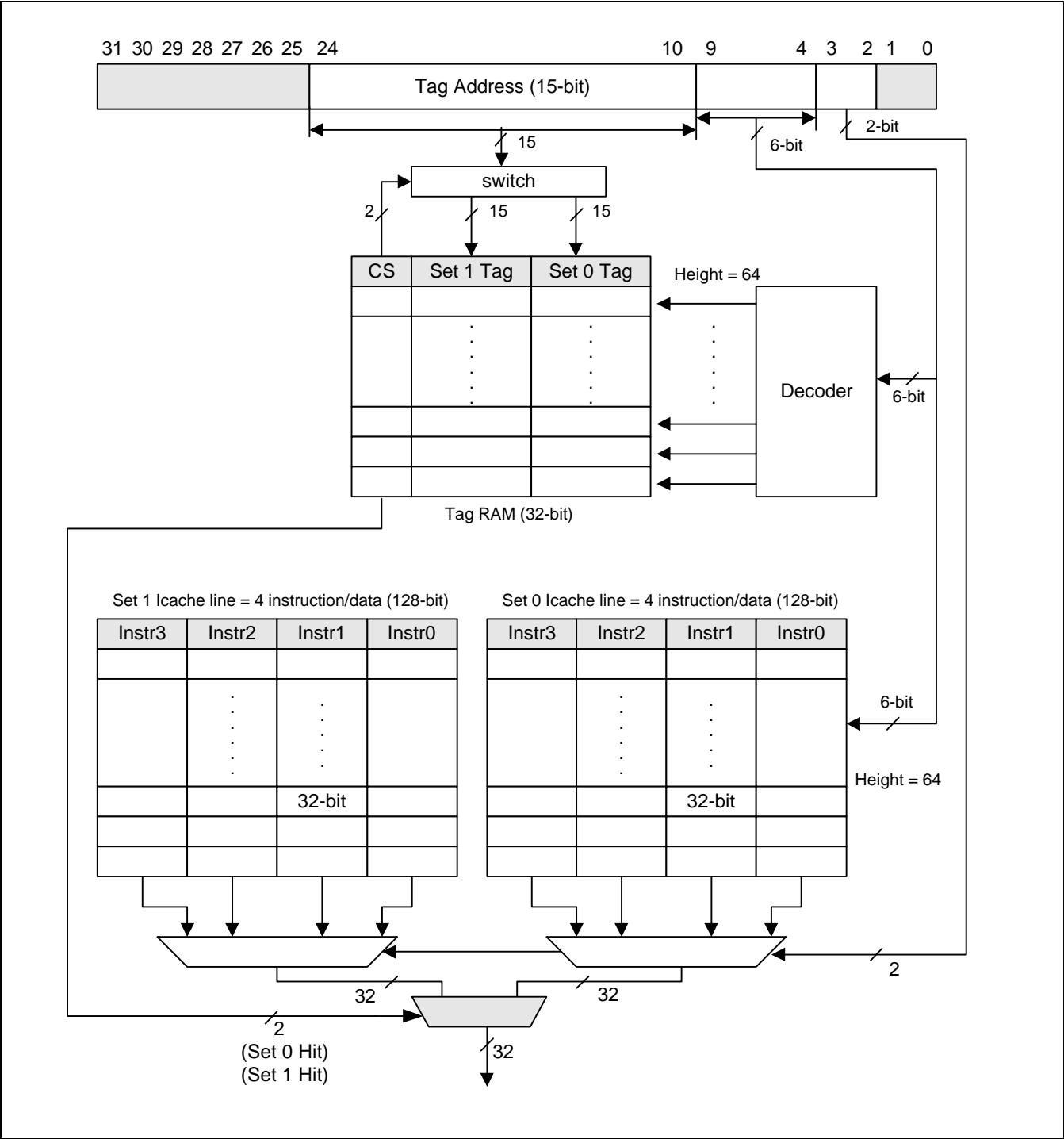


Figure 5-1. Cache Memory Configuration

CACHE OPERATION

Cache Organization

The KS32C65100 Cache has two sets of 2KB cache memory and one small Tag RAM. The Tag RAM has two bits of CS (Cache Status) and two sets of Tag memory, set 0 and 1. Each Tag set has 15 bits of address field [24:10] which is being stored in the cache memory. The CS has two bits and it indicates the validity of cached data for corresponding cache memory line. It is also used for the cache replacement algorithm and for selecting a data coming from Set 0 and 1. Cache memory has two sets, Set 0 and Set 1. Each set has 64 lines and each line has four words of memory space (128 bits).

Cache Replace Operation

After system is initialized, CS is "00" which represents that the contents of set 0 and set 1 cache memory are invalid. When first cache fill occurs, CS is changed to "01" at the specified line which represents that only set 0 is valid. When subsequent cache fills occurs, CS will be "11" at the specified line which represents that contents of both set 0 and set 1 are valid.

When the contents of the two sets are valid and when it needs content replacement due to cache miss, CS is changed to "10" at the specified line which represents that the content of set 0 was replaced. When CS is "10" and when it needs another replacement due to cache miss, the content of set 1 will be replaced by changing CS as "11". Summarizing, at normal steady state, CS will be changed from "11"/"10" to "10"/"11" which gives the information for the implementation of the 2-bit pseudo LRU(Least Recently Used) replacement policy.

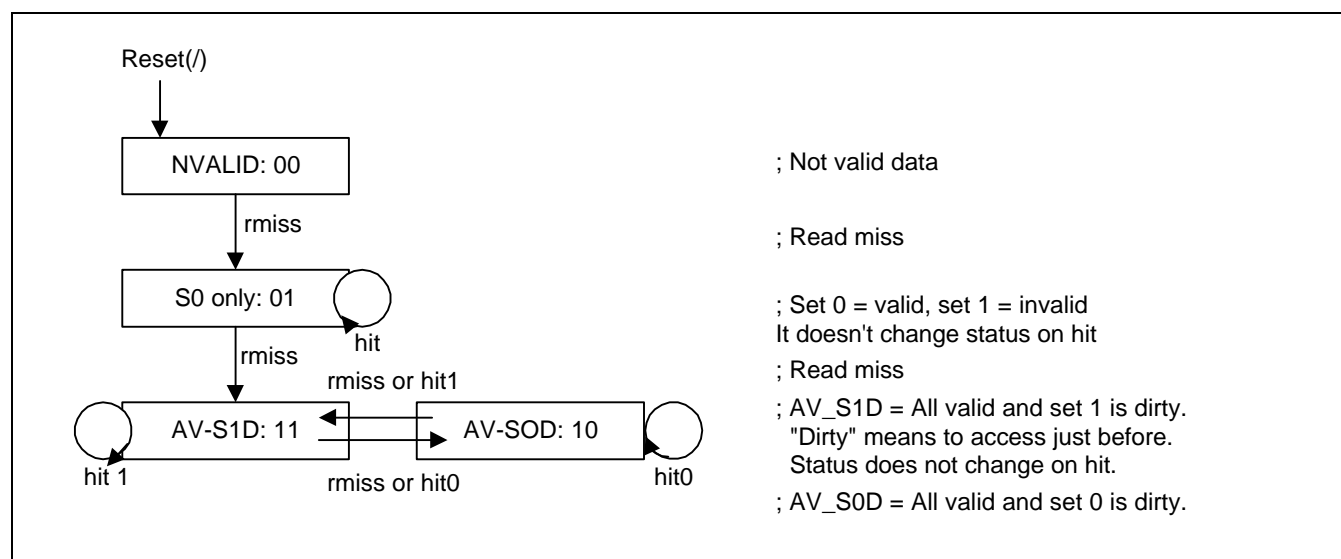


Figure 5-2. CS-bit Status Diagram

Cache Disable Operation

The KS32C65100 Cache provides a programmable entire cache enable/disable mode. The cache can be enabled by setting the CE bit in SYSCFG to 1, and disabled by clearing SYSCFG[1]. When the disable mode is specified, instructions and data are always fetched from external memory. KS32C65100 can also provide non-cacheable areas in cache enable mode for some particular memory access operations, such as the DMA operation. The two non-cacheable areas are specified by four special registers to be introduced later.

Programmers have to be cautious about data coherency when cache memory is enabled again because cache memory does not have auto flush mode. Programmers also have to be cautious if DMA changes memory data. The DMA access memory area must be non-cacheable for keeping the data coherency. To keep the data coherency between the cache and external memory, KS32C65100 uses the write-through method.

Write Buffer Operation

KS32C65100 has four write buffer registers to enhance the memory writing performance. When write buffer mode is enabled, CPU writes data into the write buffer instead an external memory when the external bus was already occupied by other bus masters like DMA. The write buffer has 4 registers and each register includes 32 bits of data field, 25 bits of address field and 2 bits of status field.

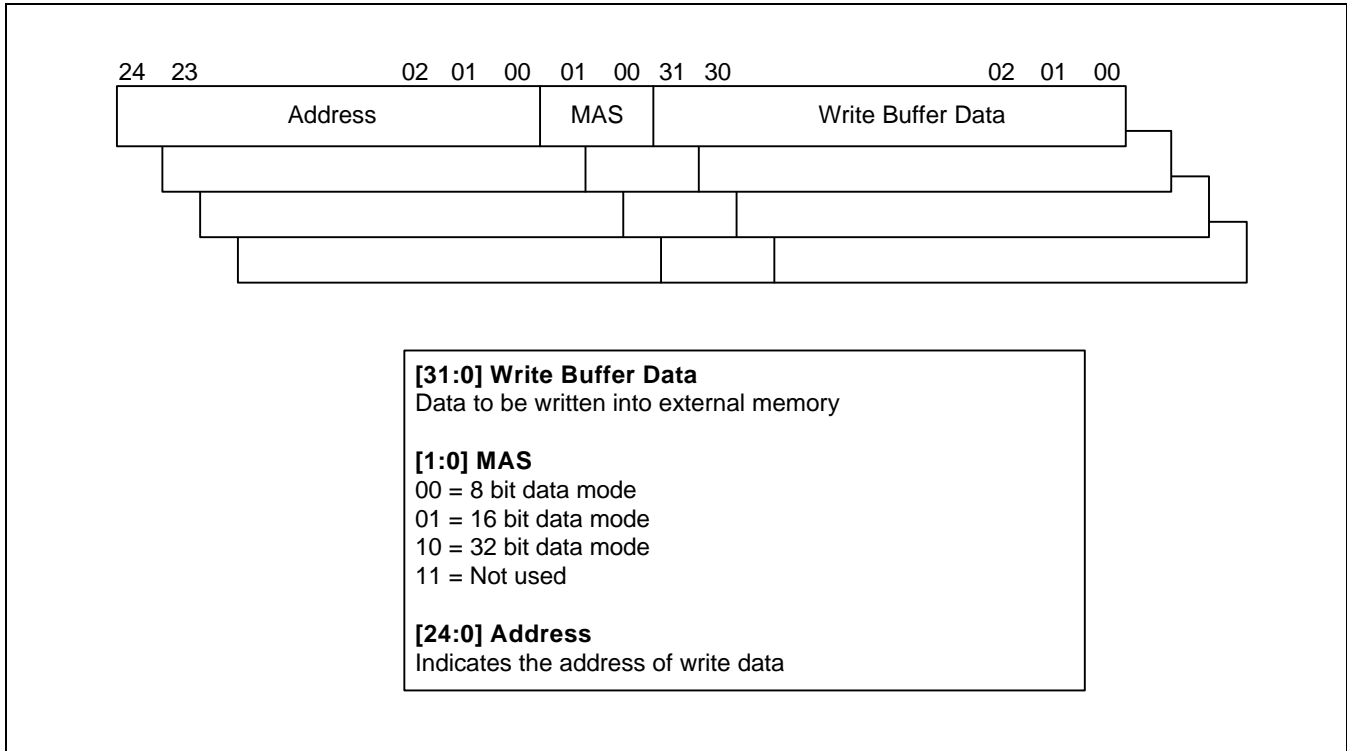


Figure 5-3. Write Buffer Configuration

CACHE CONTROL REGISTERS

The KS32C65100 Cache provides two non-cacheable areas. It has four Cache Control registers to specify two non-cacheable areas. Basically, cache stores any data within the whole system memory area, but sometimes it needs non-cacheable operation to keep the data consistency between the external memory and cache memory.

KS32C65100 provides two non-cacheable areas and each of them requires two cache control registers to indicate the start and stop address of the non-cacheable area. If a non-cacheable area is specified, that area won't be cached while read miss occurs.

Registers	Offset Address	R/W	Description	Reset Value
CACHNAB0	0x0004	R/W	Start address of non-cacheable area 0	0x0000000
CACHNAE0	0x0008	R/W	End address+1 of non-cacheable area 0	0x0000000
CACHNAB1	0x000c	R/W	Start address of non-cacheable area 1	0x0000000
CACHNAE1	0x0010	R/W	End address+1 of non-cacheable area 1	0x0000000

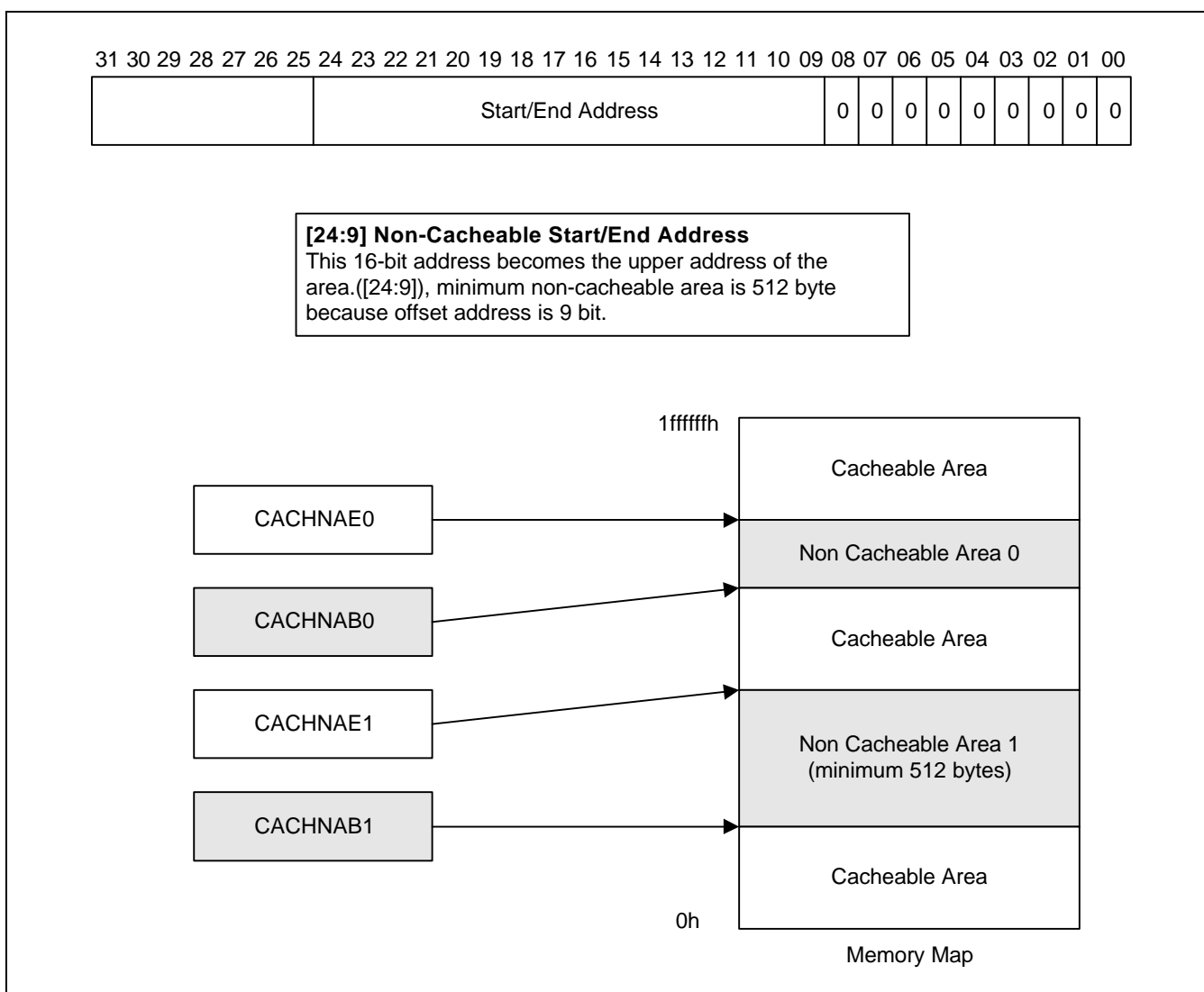


Figure 5-4. Non-Cacheable Area Register

6

DERASTERIZER

OVERVIEW

The KS65100 derasterizer provides the 16 x 16 bit image data rotation feature. The derasterizer consists of 16 registers which has a 16 bit data width. This 16 x 16 bits of register array is used to rotate raster image data 90 or 270 degrees.

Registers	Offset Address	R/W	Description	Reset Value
DRAST0	0x4800	R/W	16 bits of derasterizer data register 0	0xFFFF
DRAST1	0x4804	R/W	16 bits of derasterizer data register 1	0xFFFF
...
DRAST14	0x4838	R/W	16 bits of derasterizer data register 14	0xFFFF
DRAST15	0x483c	R/W	16 bits of derasterizer data register 15	0xFFFF

NOTE: When h[15:0] is written and v[15:0] is read, the address of DRAST0~DRAST15 is used.

ROTATION

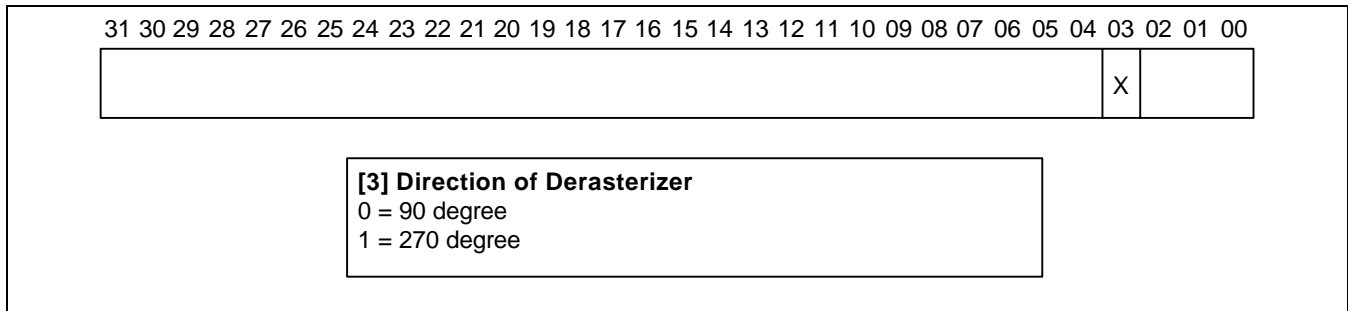
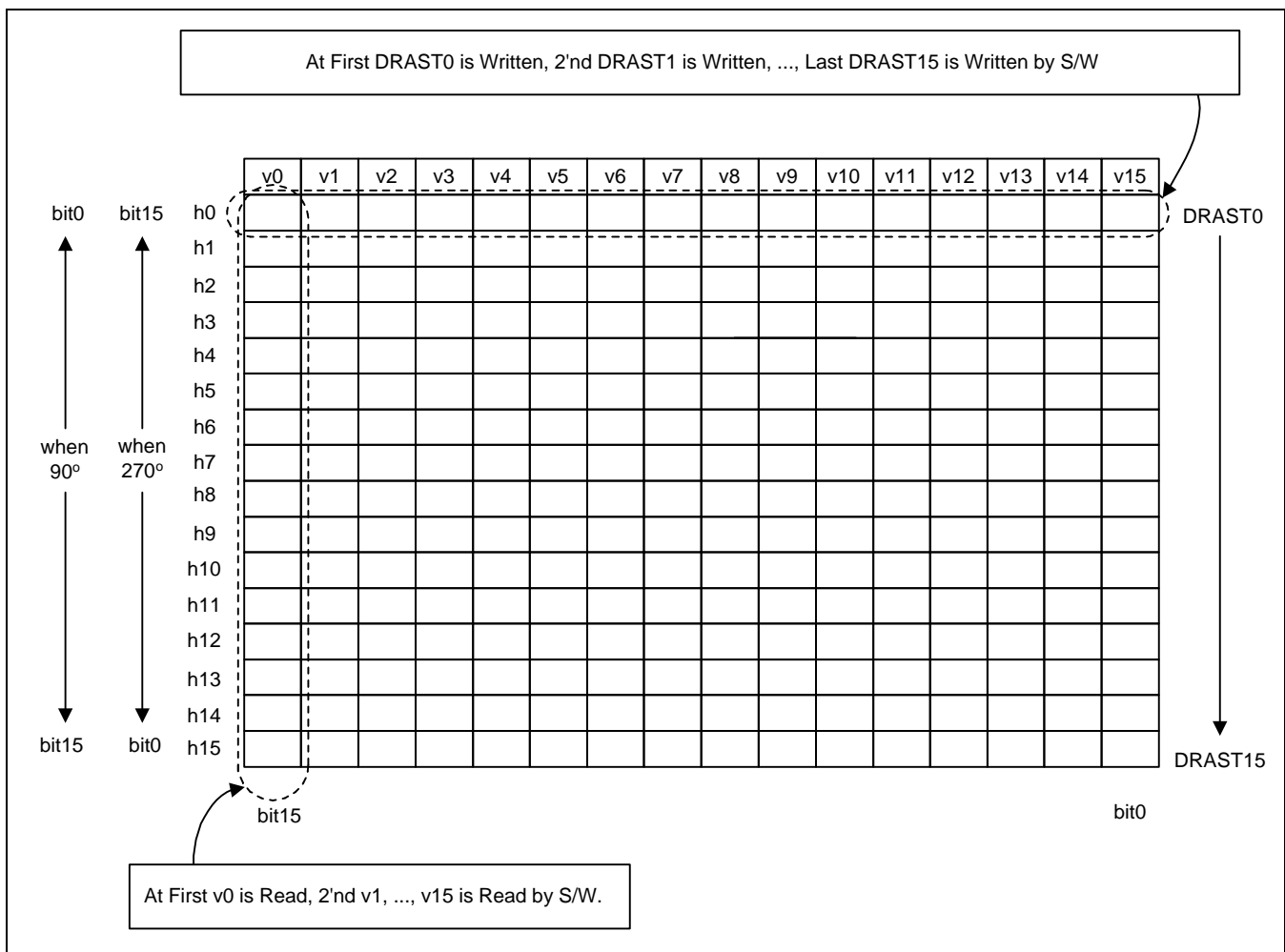
To rotate image data, programmers should fill image data into the 16 x 16 bit register array from DRAST0 to DRAST15, horizontally. The image data, made by reading the 16x16, has rotated the image. The rotation direction depends on the shift control register, SFTCON[3]. When SFTCON[3] is 0, read image data is 90 degrees rotated and when SFTCON[3] is set to 1, it is rotated 270° degrees.

- Write: h0 → h15 (DRAST0 → DRAST15)
- Read: 90 degrees: (horizontal direction) v0 → v15 ⇒ (vertical direction) MSB → h15, LSB → h0
270 degrees: (horizontal direction) v15 → v0 ⇒ (vertical direction) MSB → h0, LSB → h15

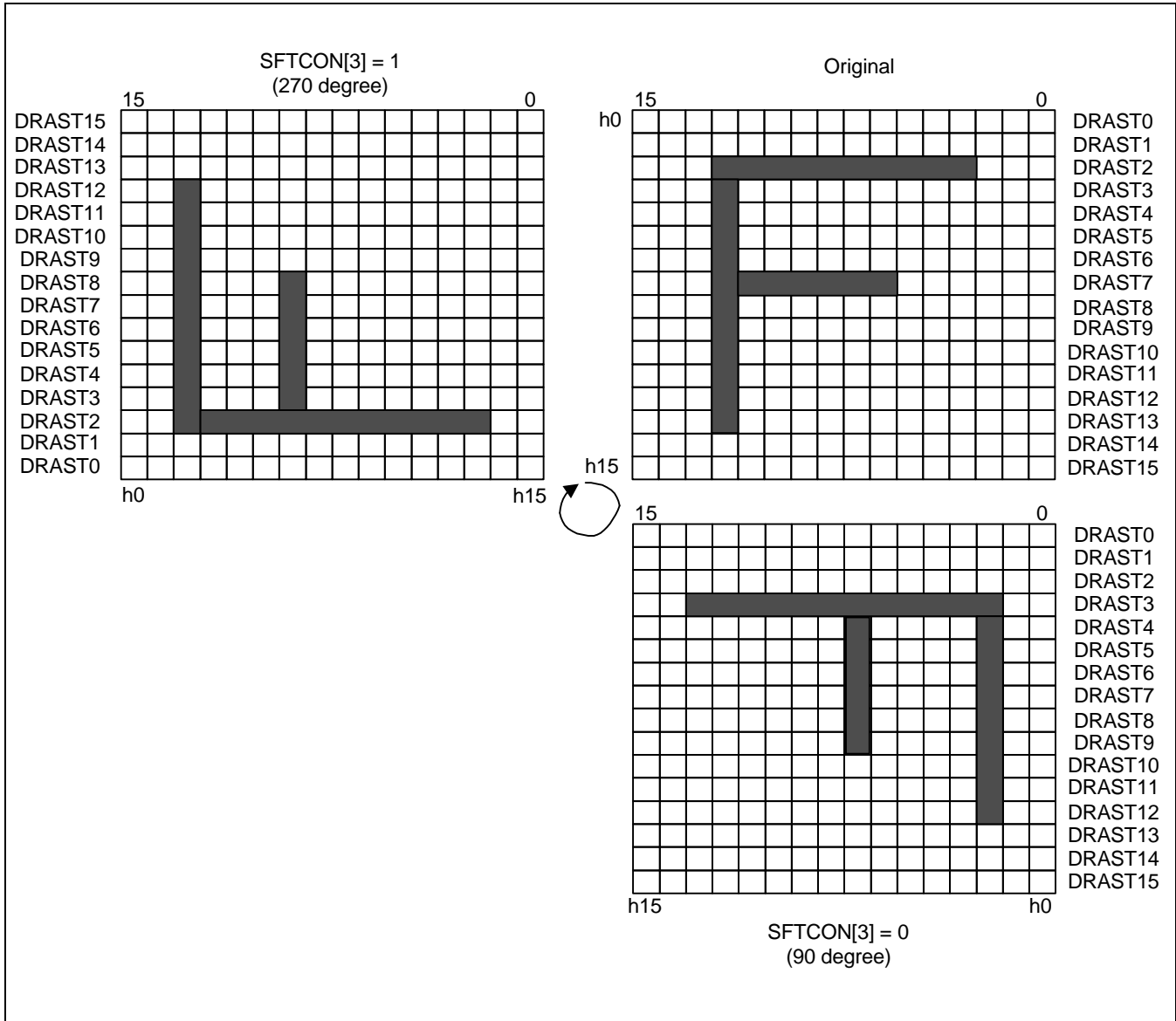
SHIFT CONTROL REGISTER (SFTCON)

Shift Control Register (SFTCON) specifies the rotation degree of the derasterizer data

Register	Offset Address	R/W	Description	Reset Value
SFTCON	0x5004	R/W	Shift control register	0x0

**Figure 6-1. Shift Control Register****Figure 6-2. Rotation Configuration**

Example



7

GENERAL ADC

OVERVIEW

The 10-bit CMOS A/D converter consists of the 3-channel analog input multiplexer, auto offset calibration comparator, high resolution R-string DAC, clock generator, 8-bit successive approximation register (SAR), ADC control register (ADCCON), and the tri-state output register (ADCDATA). The CMOS comparator includes sample and hold functions without such a circuit and has high comparator gain in two-stages. This ADC provides software-selection power-down mode. The device operates with a single +3.3V supply and its A/D conversion rate is 500 KSPS. The external clock XP1 is 25MHz. The operating temperature range is 0 ~ 70°C according to commercial specifications.

FUNCTIONS

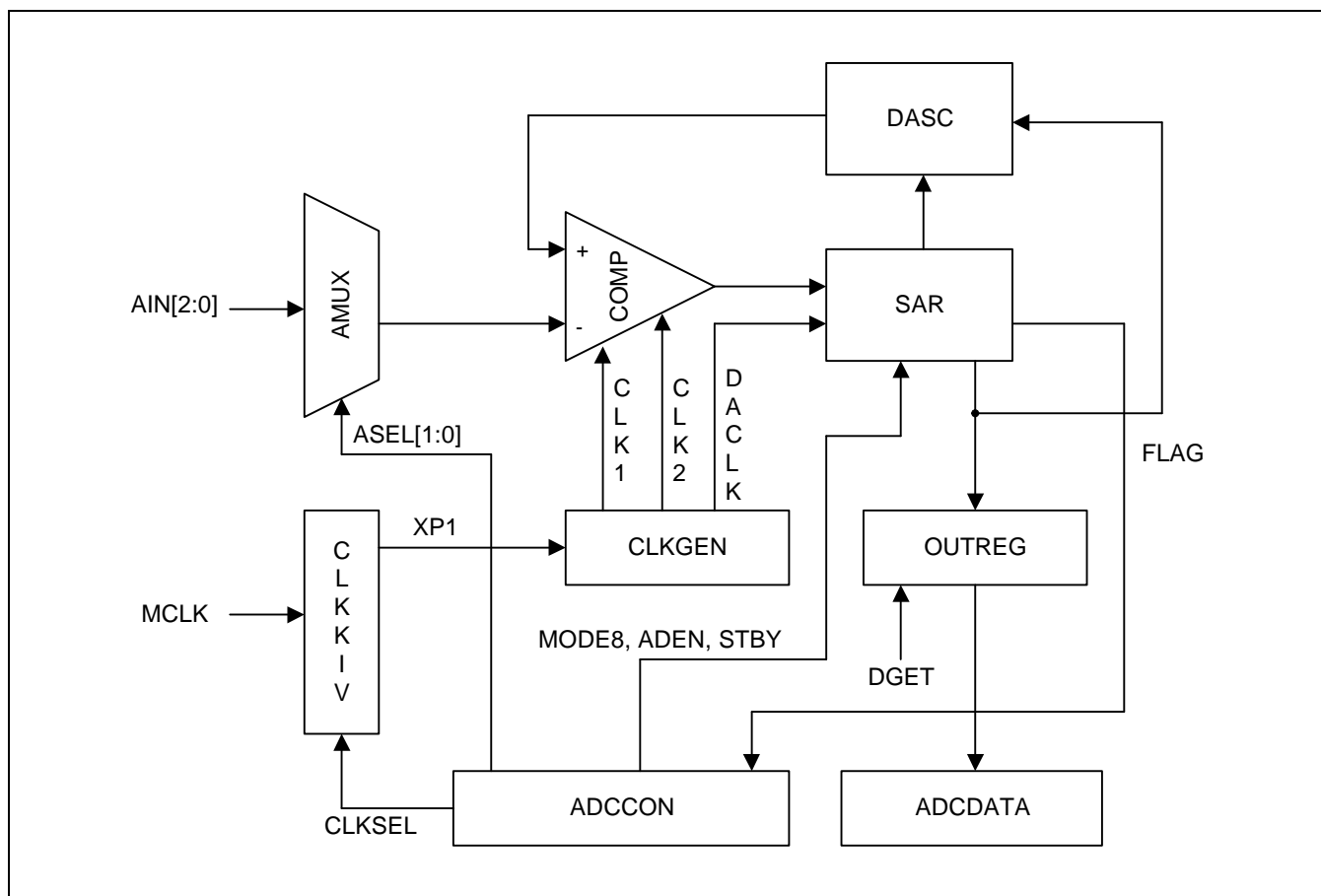


Figure 7-1. Functional Block Diagram of General ADC

SAR (SUCCESSIVE APPROXIMATION REGISTER) A/D CONVERTER OPERATION

A SAR type A/D converter basically consists of the comparator, D/A converter, and SAR logic. At the beginning of the conversion, the MSB is switched on and the analog input signal is compared to the output signal of the D/A converter. When the input signal is larger than the output signal of the D/A converter, then the MSB remains on and the next bit is switched on, and a comparison will be performed. A bit by bit operation is performed in this system to bring the D/A output signal within 1 LSB to the time discrete input signal.

COMPARATOR (COMP) AND DAC (DIGITAL TO ANALOG CONVERTER)

The CMOS comparator produces digital output as the result of comparing selected analog input with reference voltage. This comparator operates every CLK1 and CLK2, where the two clocks are non-overlapping and have anti-phase. Note that the comparator has no sample-and-hold circuit for the reduction of the current consumption.

Especially, the D/A converter consists of 128 resistor strings and switches with 7-bit resolution. So, the comparator performs the comparison with 3-bit resolution. The D/A converter generates the digitized analog output (DAOUT) from data of SAR logic block as follows.

$$DAOUT = (AVREF - AVSS)/128 \times D[9:0]$$

where AVREF and AVSS are analog reference voltage and ground that are applied to the comparator and the D/A converter block. This 128 resolution DAOUT is supplied to the CMOS comparator.

XP1 Generator and Clock Generator (CLKDIV and CKGEN)

The CLKDIV block of the A/D converter in KS32C65100 can choose two clock sources - x2 and x4 from system clock MCLK - by setting the CLKSEL bit of the ADCCON register. For the selected clock (XP1), CKGEN block generates CLK1, CLK2, and DACLK. CLK1 and CLK2 are used in the comparator, while DACLK is used to operate the SAR logic block. Note that the maximum frequency of XP1 is 25MHz.

A/D Conversion Time

When we use the main oscillation frequency of 33MHz and select the A/D converter clock to XPx4, then the total 10-bit conversion time is as follows.

$$33\text{MHz}/4 \text{ (divide 4 frequency)}/45 \text{ (at least 45 cycles by 10 bit operation)} = 183.3\text{kHz} = 5.45\mu\text{s}$$

This A/D converter was designed to operate at 25MHz XP1 clock source and the maximum conversion rate goes up to 500 KSPS.

Power-Down Mode

When the power-down mode is activated by setting the STBY bit of the ADCCON register to '1', the A/D converter is kept in standby mode without A/D conversion operation. If STBY bit is set to '1' even at A/D conversion mode, flag bit goes high immediately. When the DGET is activated by read operation during power down mode, the previous A/D conversion data are produced.

SPECIAL REGISTER**ADC Control Register**

The A/D converter control register ADCCON is used to control the operation of the 10-bit A/D converter as follows.

Register	Offset Address	R/W	Description	Reset Value
ADCCON	0xd800	R/W	ADC control register	0xa0

[0]	A/D conversion enable		This bit is A/D conversion start bit. This bit is auto-cleared after A/D conversion start-up.
[2:1]	Analog input		Select the analog input to be converted from AIN[2:0]. Select AIN[0] for "00", AIN[1] for "01", AIN[2] for '10', and none if '11'
[3]	Reserved		This bit must be '0'
[4]	Clock select		Select between MCLK divided by 2 and MCLK divided by 4 as the XP1 clock. Select 2 for "0" and 4 for "1".
[5]	Stand-by mode		This bit is used for keeping standby without A/D conversion operation. For A/D conversion, its state must be changed from '1' to '0' for at least one XP1 period.
[6]	8-bit mode		It switches the ADC function between 8 bit and 10 bit. The Low state is maintained in 10 bit operation, and high state in 8 bit operation
[7]	Flag		Its state goes '0' during A/D conversion, and goes '1' after the A/D conversion. If STBY signal is applied even at A/D conversion mode, its state goes '1' immediately.

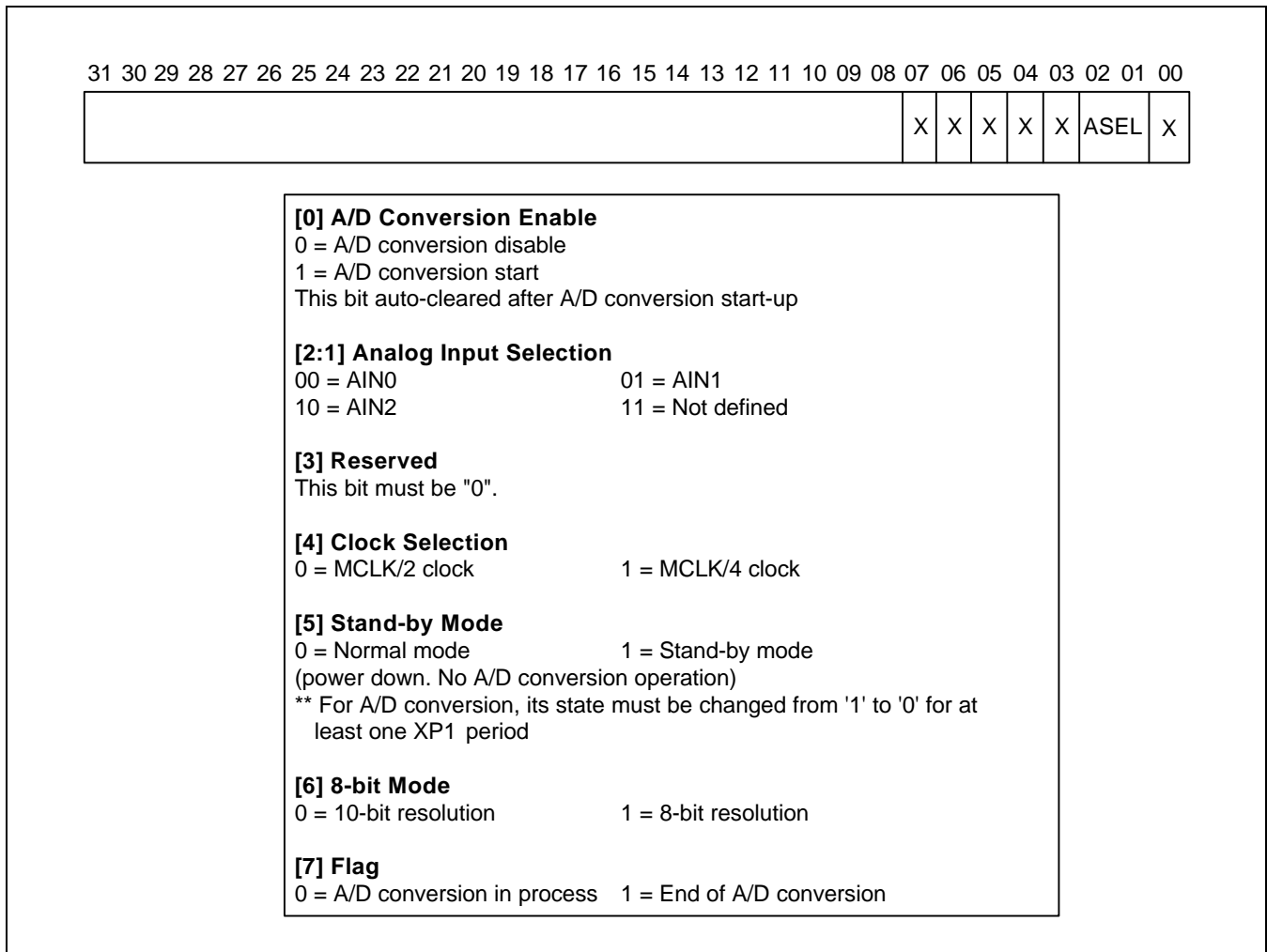


Figure 7-2. ADC Control Register (ADCCON)

ADC Data Register

ADCDATA loads the A/D conversion data during read operation after the conversion process is completed and flag goes '1'. Internally, DGET signal is activated by read operation of ADCDATA and A/D converted data are produced by applying DGET.

Register	Offset Address	R/W	Description	Reset Value
ADCDATA	0xd804	R	ADC data register	0xFFFF

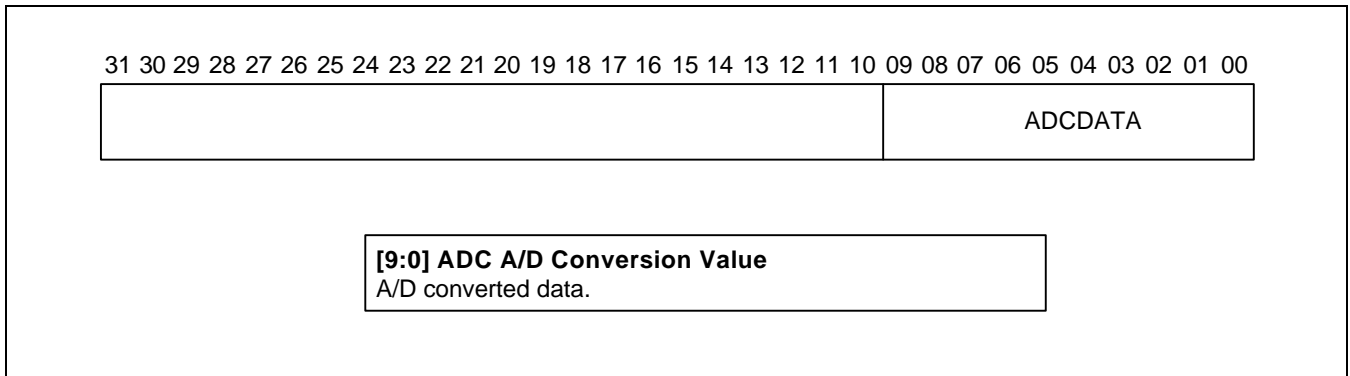


Figure 7-3. ADC Data Register (ADCDATA)

8

TIMER

OVERVIEW

Timer Block has three 16-bit timers. Three timer blocks share an 8-bit prescaler and a clock divider which has 4 different divided signals. Each timer block receives its own clock signals (signal name is "Timer Clock") from the clock divider which receives the clock from the 8-bit prescaler. The 8-bit prescaler is programmable and it divides the MCLK signal depending on the loading value which is stored in TSTCON[14:7] bits.

The timer count value register (TBCNTn) stores initial count value and its data is loaded into the down counter when the timer is enabled. Each timer has its own 16-bit down counter that is driven by the timer clock. When one of the down counters reaches zero, the timer counter interrupt request is generated to inform the CPU that one of the timer operations is completed. When it reaches zero, the corresponding TBCNTn content is automatically loaded into the down counter to continue the next operation. However, if a timer is stopped, for example if you clear the timer enable bit in TCON during the timer running mode, the count value in TBCNTn will not be reloaded into counter.

The timer count value register is used to define the duration for timer operation, and contains the number of timer clock periods needed for one operation duration.

The timer duration can be calculated as follows:

$$\text{Timer_clock} = \text{MCLK} / (\text{prescale_value} + 1) / \text{Division_factor (Hz)}$$

$$\text{Timer_duration} = \text{count_value} / \text{Timer_clock}$$

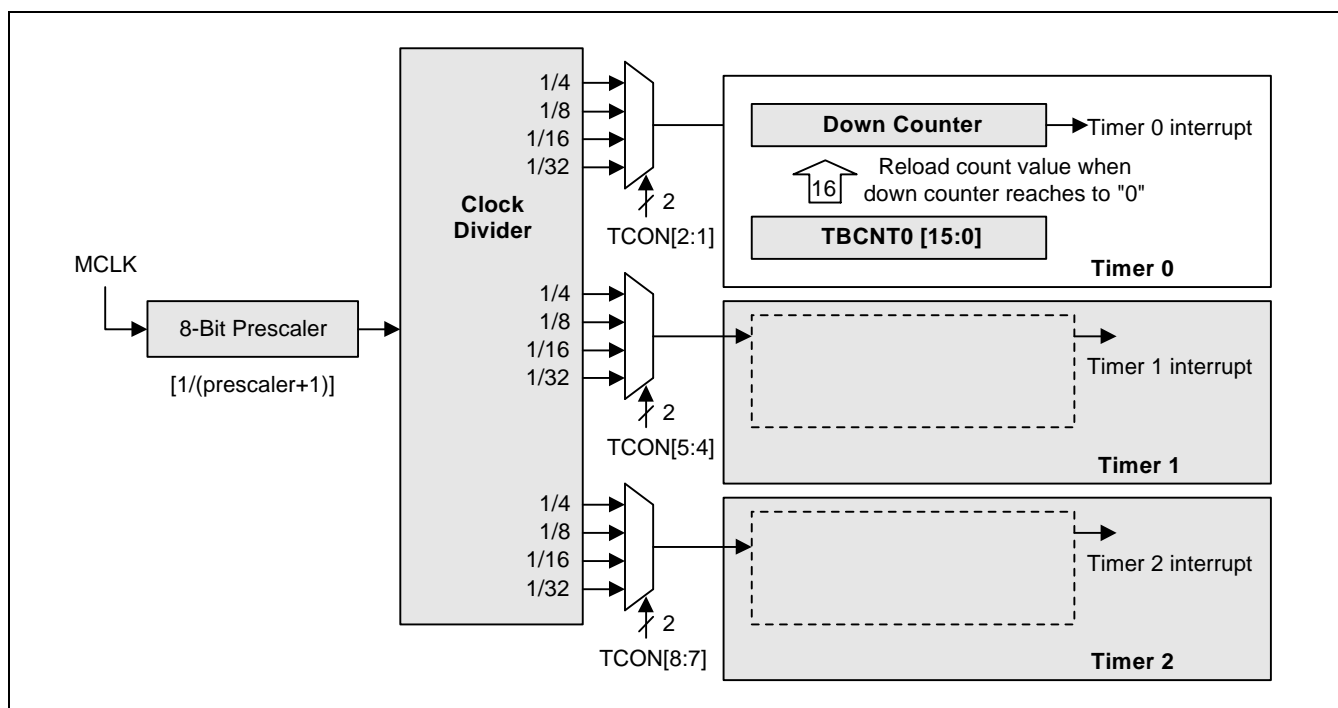


Figure 8-1. 16-Bit Timer Block Diagram

TIMER CONTROL REGISTER

Programmers can disable or enable the timer operation and can select a clock divider output from 4 divided signals by using the Timer Control Register (TCON).

Register	Offset Address	R/W	Description	Reset Value
TCON	0x3000	R/W	System timer control register	0x000h

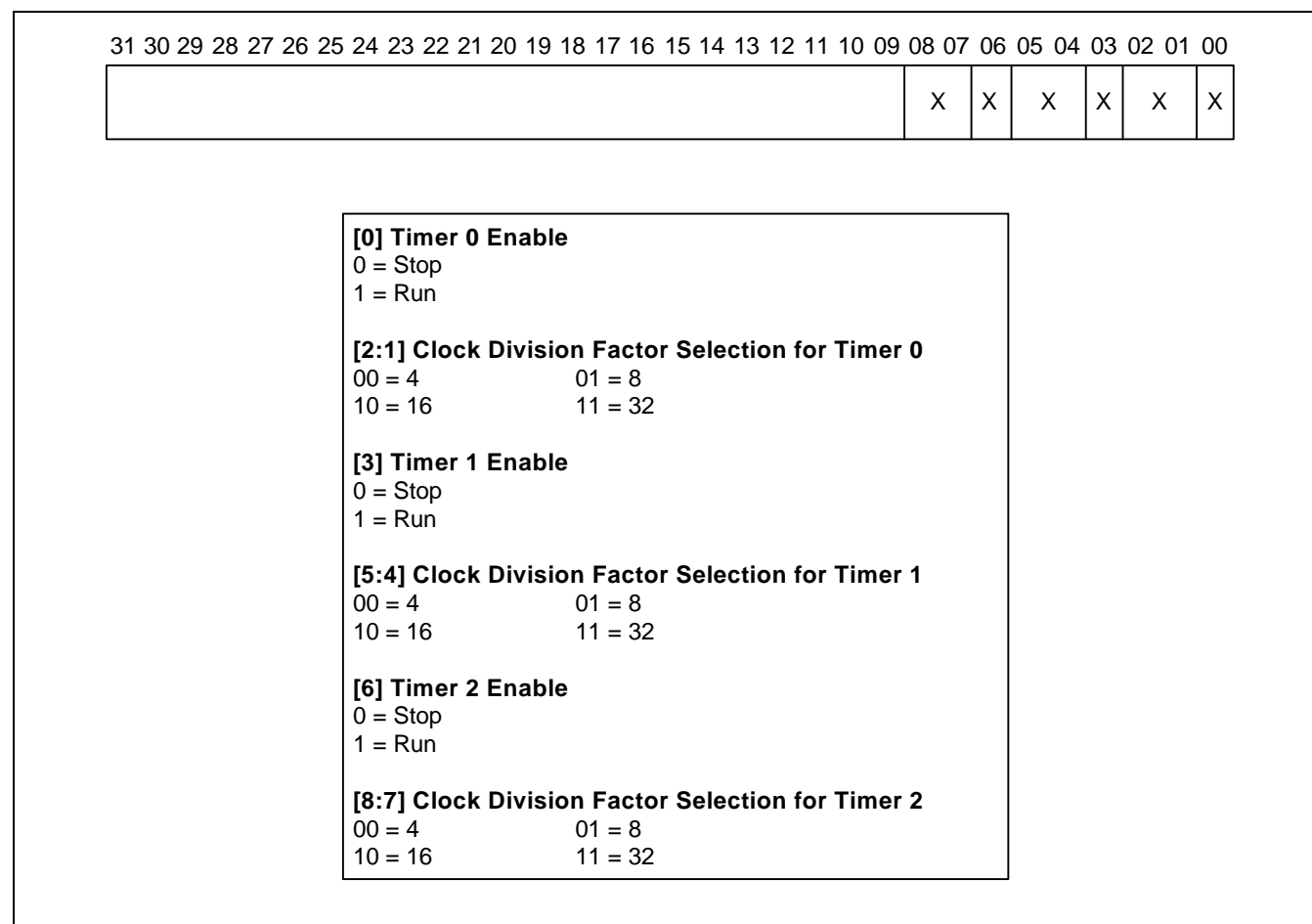


Figure 8-2. Timer Control Register

TIMER COUNT VALUE REGISTER

The timer count value registers, TBCNTn, are used to specify the time-out duration for each timers. Counting value will be loaded or reloaded into down counter automatically when timer operation is enabled or timer-out occurs (i.e. the down counter is decreased to "0").

Registers	Offset Address	R/W	Description	Reset Value
TBCNT0	0x3004	R/W	Timer 0 count value register	0xFFFF
TBCNT1	0x3008	R/W	Timer 1 count value register	0xFFFF
TBCNT2	0x301c	R/W	Timer 2 count value register	0xFFFF

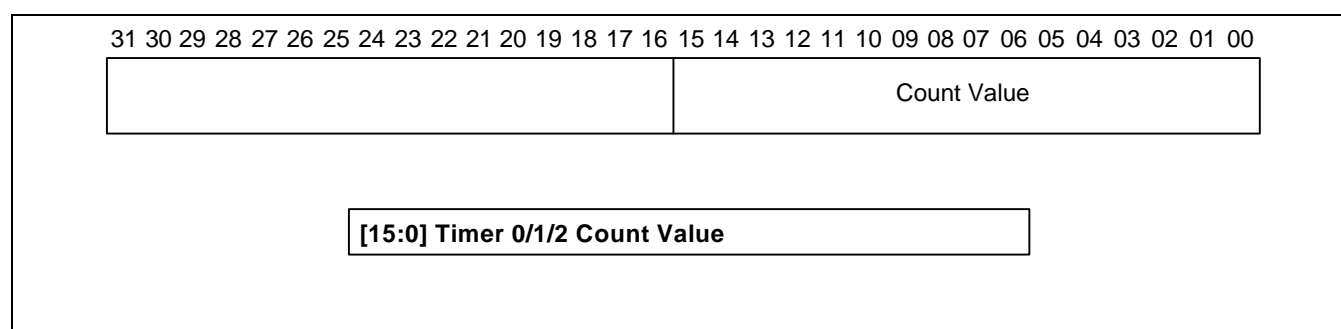


Figure 8-3. Timer Count Value Register

If a programmer changes the contents of TBCNTn while the timer is enabled, the new value will be written in this register and the counter continues to count with new value.

Example: The timer programming sequence is shown below. The count value and timer clock definition including the prescaling value and clock division factor, should be specified before the timer-enable bit setting.

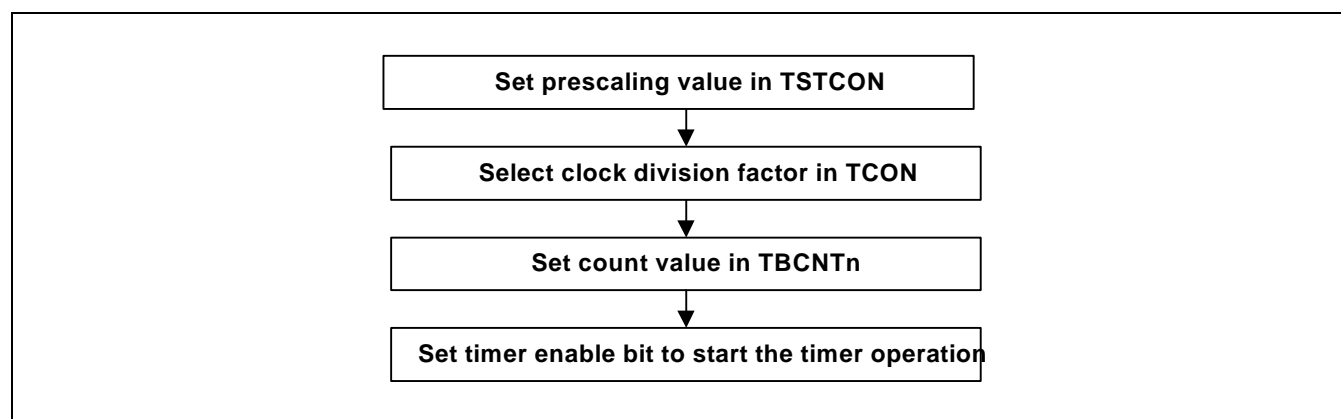


Figure 8-4. Timer Programming Sequence

9

DMA

OVERVIEW

The KS32C65100 has two general direct memory access channels (GDMA, CDMA). These DMA channels perform the data transfers between the following sources without CPU intervention:

- Memory and memory
- IP and memory (GDMA)
- Parallel port and memory (CDMA)
- Serial port and memory

The on-chip DMA controller can be started by software and/or by an external DMA request. DMA operation can also be stopped and restarted by software. The CPU can recognize when a DMA operation has been completed by software polling and/or by DMA interrupt request. The KS32C65100 DMA controller can increase or decrease source or destination address, and conduct 8-bit (byte), 16-bit (half-word), or 32-bit (word) data transfers. Detailed information about the DMA block's operation is provided in the descriptions of each DMA register.

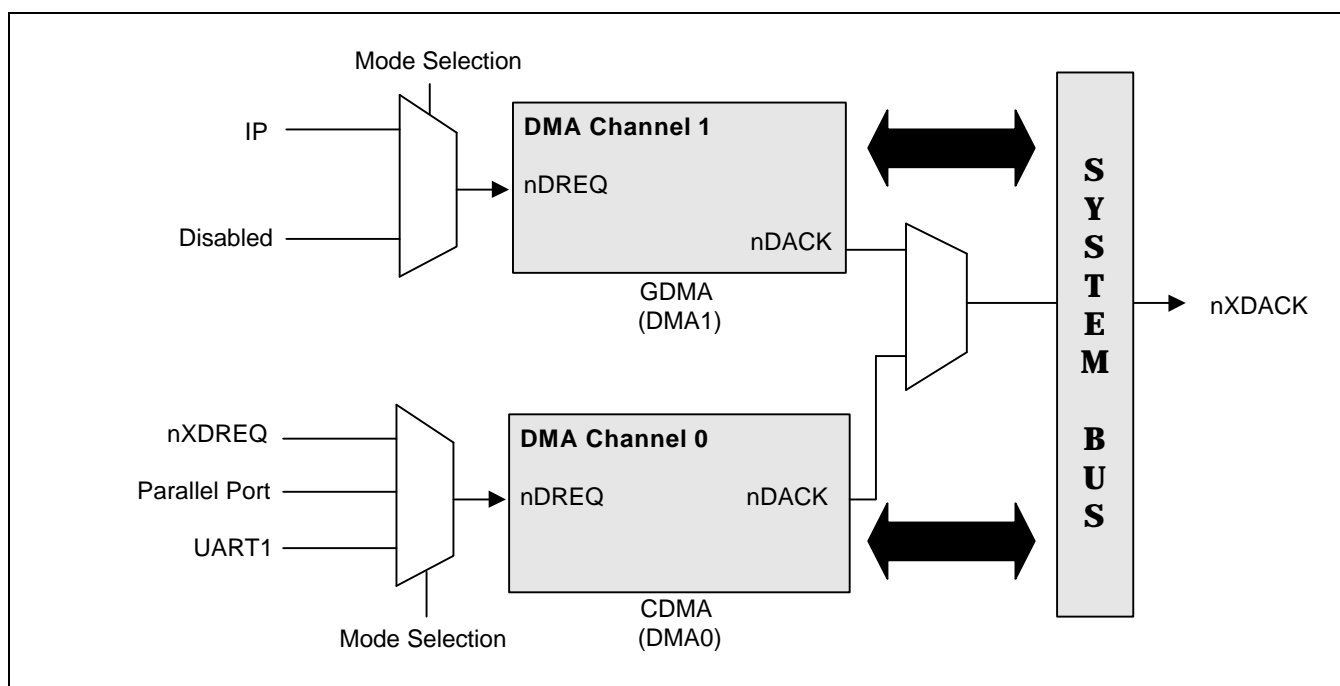


Figure 9-1. GDMA/CDMA Unit Block Diagram

DMA OPERATION

The following sections describe the operation of the DMA.

DMA Transfers

The DMA transfers data directly between a requester and a target. The requester and target are memory, UART, IP(GDMA), parallel port (CDMA), or external devices (CDMA). An external device requests DMA service by activating an nXDREQ signal.

A channel is programmed by writing to registers which contain the requester address, target address, the amount of data, and other control contents.

UART, IP, parallel port, external I/O, or Software (memory) can request DMA service. UART, IP and parallel port are internally connected to the DMA. In particular, UART1 requests the DMA service to CDMA.

Bus Control Arbitration

Because GDMA, CDMA and DRAM controller (DRAM refresh) can all request bus control, bus control priority must be arbitrated. The priority of these bus masters is fixed as follows:

- 1: GDMA
- 2: DRAM controller (DRAM refresh)
- 3: CDMA

For very fast response of GDMA request, GDMA has the highest priority. As GDMA has higher priority than the DRAM controller, GDMA is used very carefully not to disturb the DRAM controller refreshing the DRAM. You may think that GDMA can't move the large amount of data for DRAM because of DRAM refresh, but GDMA can transfer the large amount of data DRAM if user don't use a continuous mode of GDMA. The GDMA which doesn't use the continuous mode, releases the internal bus request in a short time after one unit of data (1 word, 1 half-word (16-bit) or 1 byte). When the bus is just released, the DRAM controller may have the bus and can refresh DRAMs.

If CDMA, which has the lower priority than the DRAM controller, holds bus by continuous mode, the DRAM refresh controller can not have the bus control until CDMA frees the bus control.

Starting/Ending DMA transfers

DMA starts to transfer data after the DMA receives service request from then XDREQ signal, UART, parallel port, or Software. When the entire buffer of data has been transferred, the DMA becomes idle. If you want to perform another buffer transfer, the DMA must be reprogrammed. Although the same buffer transfer will be performed again, the DMA must be reprogrammed.

The Major Difference Between GDMA and CDMA

GDMA and CDMA has differences as shown in table 9-1.

Table 9-1. Difference Between GDMA and CDMA

Functions	GDMA	CDMA
Single mode	O	O
Block mode	O	O
Demand mode	O	X
Byte swap mode	X	O

DATA TRANSFERS MODE

Single Mode

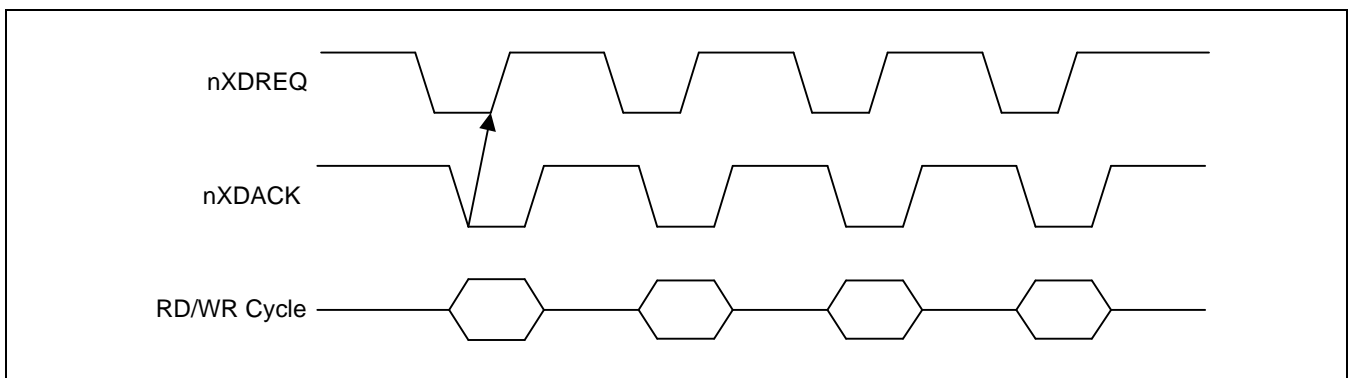
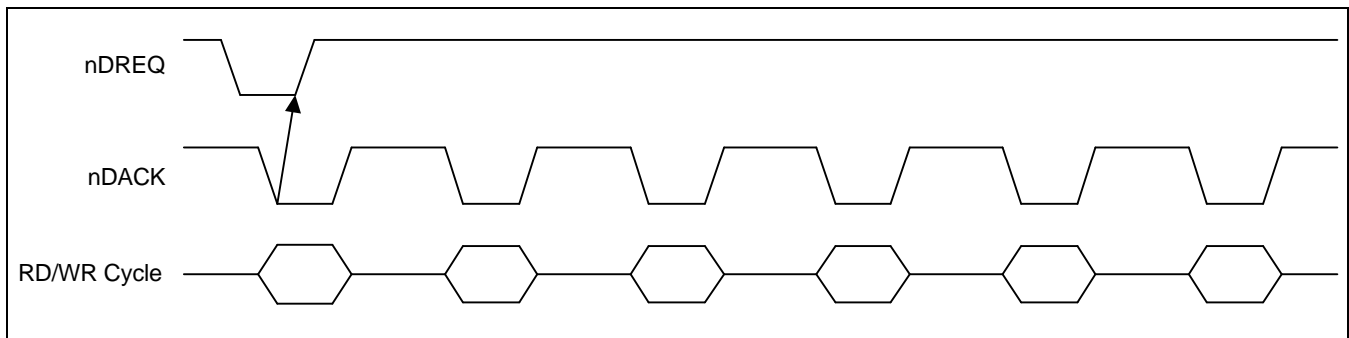
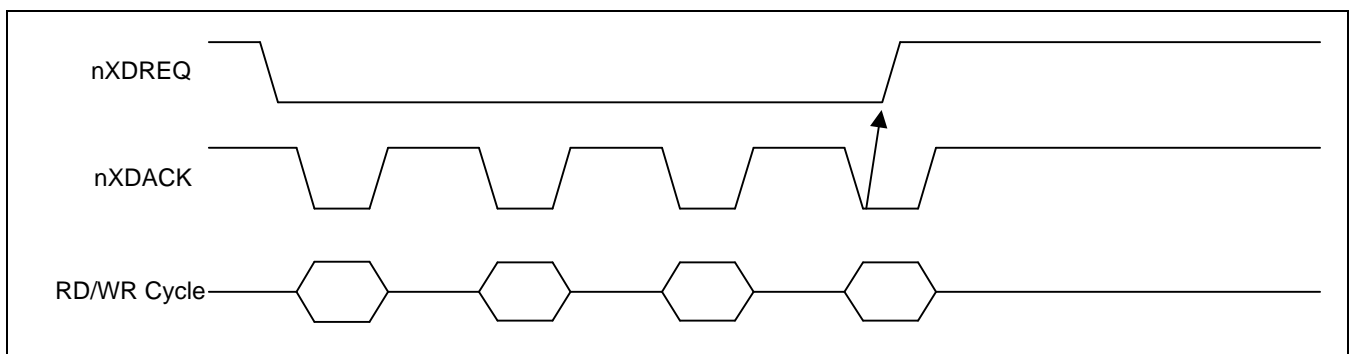


Figure 9-2. External DMA Requests @ Single Mode

The DMA request (nXDREQ or internal request) causes one byte, one half word, or one word to be transmitted. The single mode requires the DMA request for every data transfer. The nXDREQ signal may be de-asserted after checking nXDACK to be asserted.

Block Mode**Figure 9-3. External DAM Requests @ Block Mode**

The assertion of only one DMA request (nXDREQ or internal request) causes the entire data, which is set in control registers, to be transmitted. DMA transfer will be completed when the counter reaches zero. The nXDREQ signal may be de-asserted after checking nXDACK to be asserted.

Demand Mode**Figure 9-4. External DMA Requests @ Demand Mode**

The amount of data that DMA transfers depends on how long the DMA request input (nXDREQ) is held active. In the demand mode, the DMA (GDMA only) continues to transfer data while the DMA request input (nXDREQ) is held active.

GENERAL DMA CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
DMACON1	0x9000	R/W	GDMA control register	0x0000

GDMA Control Register Description**[0] Run enable/disable**

The DMA operation starts. When you set this bit to '1' To stop DMA, you must clear this bit to '0'. To control only this bit, use the address 0x9020. By using 0x9020, the other values in the control register will not be affected.

[1] BUSY status

When DMA starts, this read-only status bit is automatically set to '1'. When DMA is in an idle state, this bit is '0'.

[3:2] GDMA mode selection

Four sources can initiate a DMA operation: software (memory to memory), the IP block. The mode selection bits determine which source can initiate a DMA operation at any given time (see Figure 9-5).

[4] Destination adr direction

This bit determines whether the destination address will be decreased or increased during a DMA operation.

[5] Source address direction

This bit determines whether the source address will be decreased or increased during a DMA operation.

[6] Destination address fix

This bit determines whether the destination address will be changed or not during a DMA operation. This feature is used when transferring data from multiple sources to a single destination.

[7] Source adr fix

This bit determines whether the source address will be changed or not during a DMA operation. This feature is used when transferring data from a single source to multiple destinations.

[8] Stop interrupt enable

A DMA operation is started/stopped by setting/clearing the run enable/disable bit. If this bit is set to '1' and DMA is running, a 'stop interrupt' is generated when DMA operation forced to stop on purpose. If this bit is '0', the 'stop interrupt' is not generated. The interrupt which is generated when the DMA counter is expired cannot be masked by this bit.

[9] Reset

If this bit is set to '1', then the DMA control register value will refer to default values. When this bit is cleared to '0', you can specify other control values.

[10] Peripheral direction

This mode bit specifies the direction of the DMA operation. If this bit is set to '1', DMA operates from memory to peripheral (IP). If this bit is cleared to '0', DMA operates from peripheral to memory.

[13:12] Transfer width

This determines the transfer data width to be byte (8-bit), halfword (16-bit), or word (32-bit). If transfer length is a byte, source/destination address will be increased/ decreased by 1. If it is a halfword, then the address will change by 2. If it is a word, the address will increase/decrease by 4. It's important that the "transfer width" is not the size of a physical data bus. The size of a physical data bus is determined by SMR configurations.

[14] Continuous mode

This bit specifies whether the DMA operation will hold the system bus or not until the count value is 0. Therefore, this bit must be carefully used for the whole operation time not to exceed the appropriate interval (ex: DRAM Refresh).

[15] Demand mode

If this bit is set during the DMA operation, DMA never goes to the idle state. Altogether, the external device transfers/receives the amount of data which it wants to transfer/receive. The amount of data depends on how long the REQ signal is active.

NOTE: All control bits have to be configured independently and carefully
External I/O related bits have no effect because nXDREQ is not connected to GDMA.

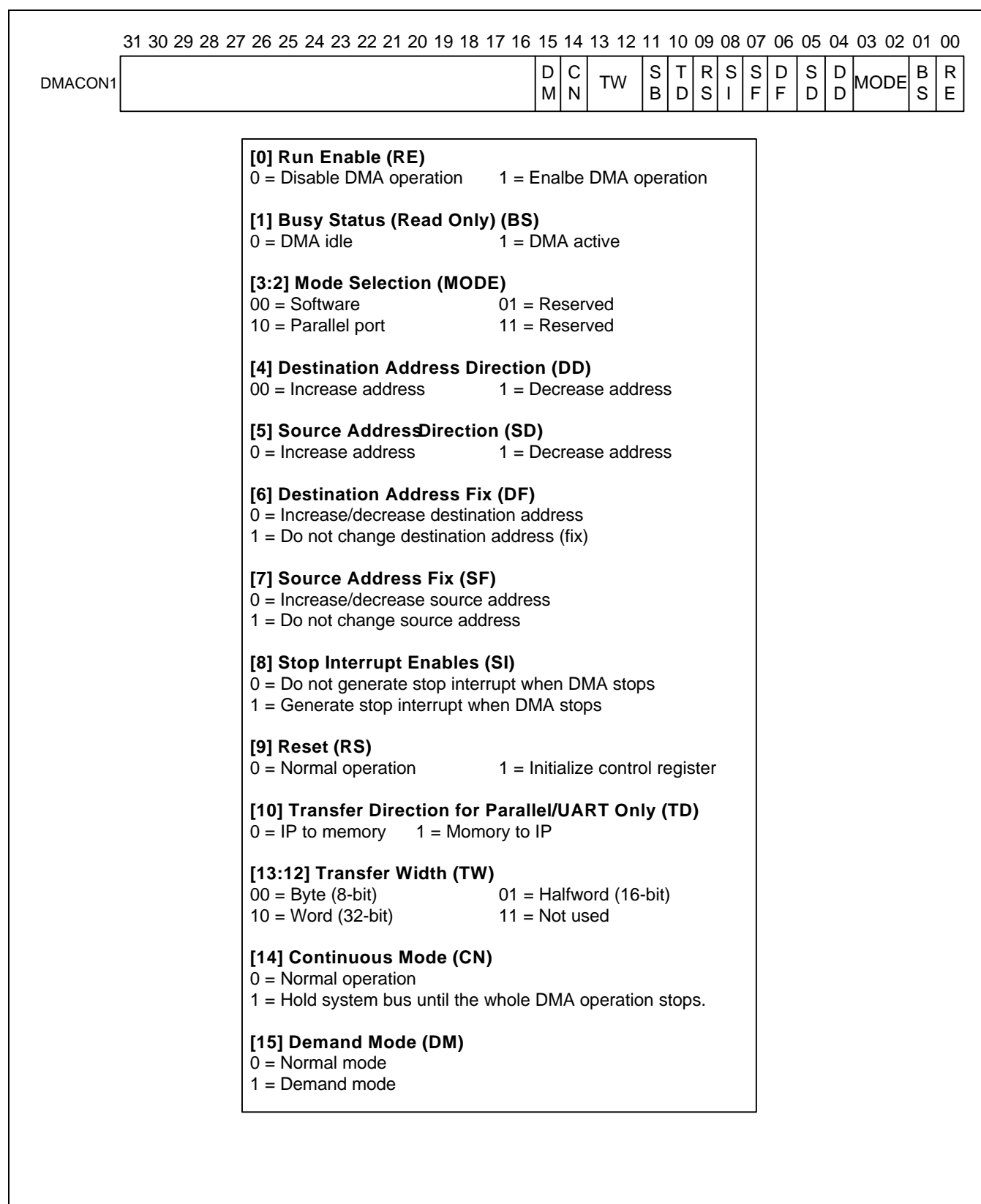


Figure 9-5. GDMA Control Register

GDMA SOURCE/DESTINATION ADDRESS REGISTER

These registers contain the 25-bit source/destination address for a DMA channel. Depending on the setting of the DMA control register (DMACON1), these addresses will increase, decrease, or remain the same.

Registers	Offset Address	R/W	Description	Reset Value
DMA_SRC1	0x9004	R/W	GDMA source address register	0xFFFFFFFF
DMA_DST1	0x9008	R/W	GDMA destination address register	0xFFFFFFFF

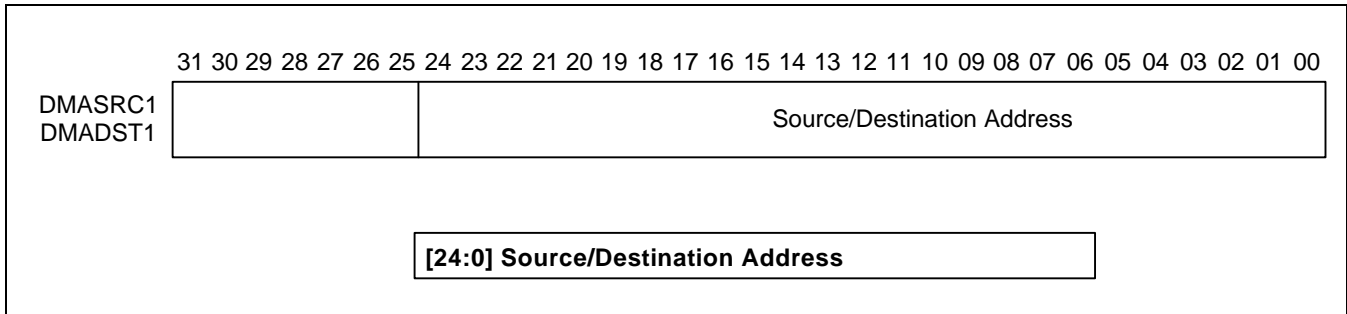


Figure 9-6. GDMA Source/Destination Address Register

GDMA TRANSFER COUNT REGISTER

This register contains a 24-bit value which is the number of completed DMA transfers. This value is decreased by 1 when one DMA operation is completed regardless of the width of the data that was transferred.

Register	Offset Address	R/W	Description	Reset Value
DMA_CNT1	0x900c	R/W	GDMA transfer count register	0xFFFFFFFF

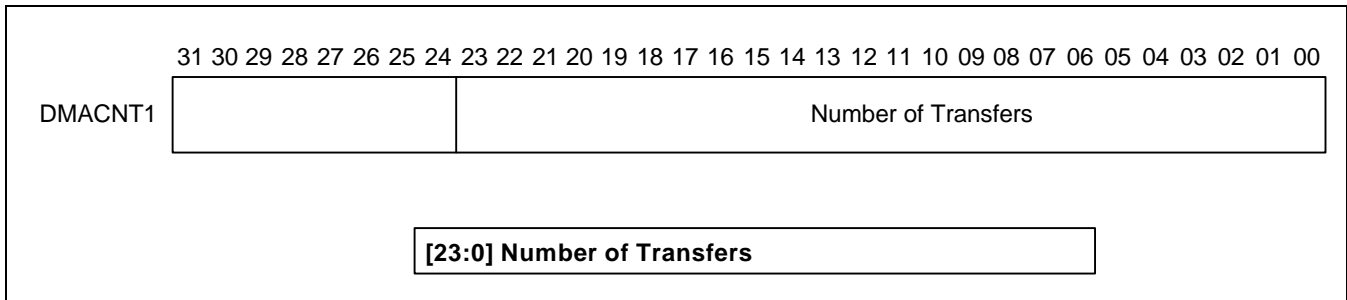


Figure 9-7. GDMA Transfer Count Register

CDMA CONTROL REGISTER

CDMA(C-DMA) is 2nd DMA. CDMA can transfer the data by byte swap mode. UART1 can transfer data only through CDMA.

CDMA (Compress/Decompress DMA) Control Register

Register	Offset Address	R/W	Description	Reset Value
DMACON0	0x8800	R/W	CDMA control register	0x00000

CDMA Control Register Descriptions (Note: " * " denotes a read only bit)

[0] Run enable/disable

CDMA operation starts When you set this bit to '1'. To stop CDMA, you must clear this bit to '0'. To control this bit only, use the address 0x8810. By using 0xc810, the other values in the control register will not be affected.

[1] BUSY status*

When CDMA starts, this read-only status bit is automatically set to '1'. When it is '0', CDMA is in an idle status.

[3:2] CDMA mode selection

Four sources can initiate a CDMA operation: software, an external CDMA request (nXDREQ) , the parallel port, and the UART block. The CDMA mode selection bits determine which source can initiate a CDMA operation at any given time (see Figure 9-8).

[4] Destination adr direction

This bit determines whether the destination address will be decreased or increased during a CDMA operation.

[5] Source adr direction

This bit determines whether the source address will be decreased or increased during a CDMA operation.

[6] Destination adr fix

This bit determines whether the destination address will change or not during a CDMA operation. This feature is used when transferring data from multiple sources to a single destination.

[7] Source adr fix

This bit determines whether the source address will change or not during a CDMA operation. This feature is used when transferring data from a single source to multiple destinations.

[8] Stop interrupt enable

A CDMA operation is started/stopped by setting/clearing the run enable/disable bit. This bit is set to '1' when DMA operation starts. a 'stop interrupt' is generated when CDMA operation stops. If this bit is '0', the 'stop interrupt' is not generated. the interrupt which is generated when the DMA counter is expired cannot be masked by this bit.

[9] Reset

If this bit is set to '1', the CDMA control register value will be initialized. When this bit is cleared to '0', you can specify other control values.

[10] Peripheral direction

When the mode bit is set to '10'(parallel port from/to memory) or '11'(UART from/to memory), this direction bit specifies the direction of the CDMA operation. If this bit is set to '1', then CDMA operates from memory to peripheral(parallel port/UART). If this bit is cleared to '0', CDMA operates from peripheral to memory.

[11] Single/Block mode

This bit determines the number of external CDMA requests(nXDREQ) that are required for CDMA operation. At single mode (this bit is set to '0'), the KS32C65100 requires an external DMA request for every CDMA operation. At block mode (this bit is set to '1'), the KS32C65100 requires only one DMA request during the entire CDMA operation. An entire CDMA operation is defined as the operation of CDMA until the counter is '0'.

[13:12] Transfer width

This determines the width of the data being transferred to be a byte, a halfword, or a word. If byte operation is set, then source/destination address will be increased/decreased by 1. If it is a halfword, then the address is changed by 2. If it is a word, the address is changed by 4. It's important that the "transfer width" is not the size of a physical data bus. The size of physical data bus is determined by SMR configurations.

[14] Continuous mode

This bit specifies that CDMA operations hold the system bus until the count value is 0 Therefore, this bit must be carefully used unless the whole operation time can not over appropriate interval.

[16] Byte swap mode

When the transfer size is a halfword or a word, this bit specifies whether a byte swap operation has occurred or not.

For example, if this bit is set to '1', 11223344h (read) → 44332211h (write)

NOTE: All control bits have to be configured independently and carefully.

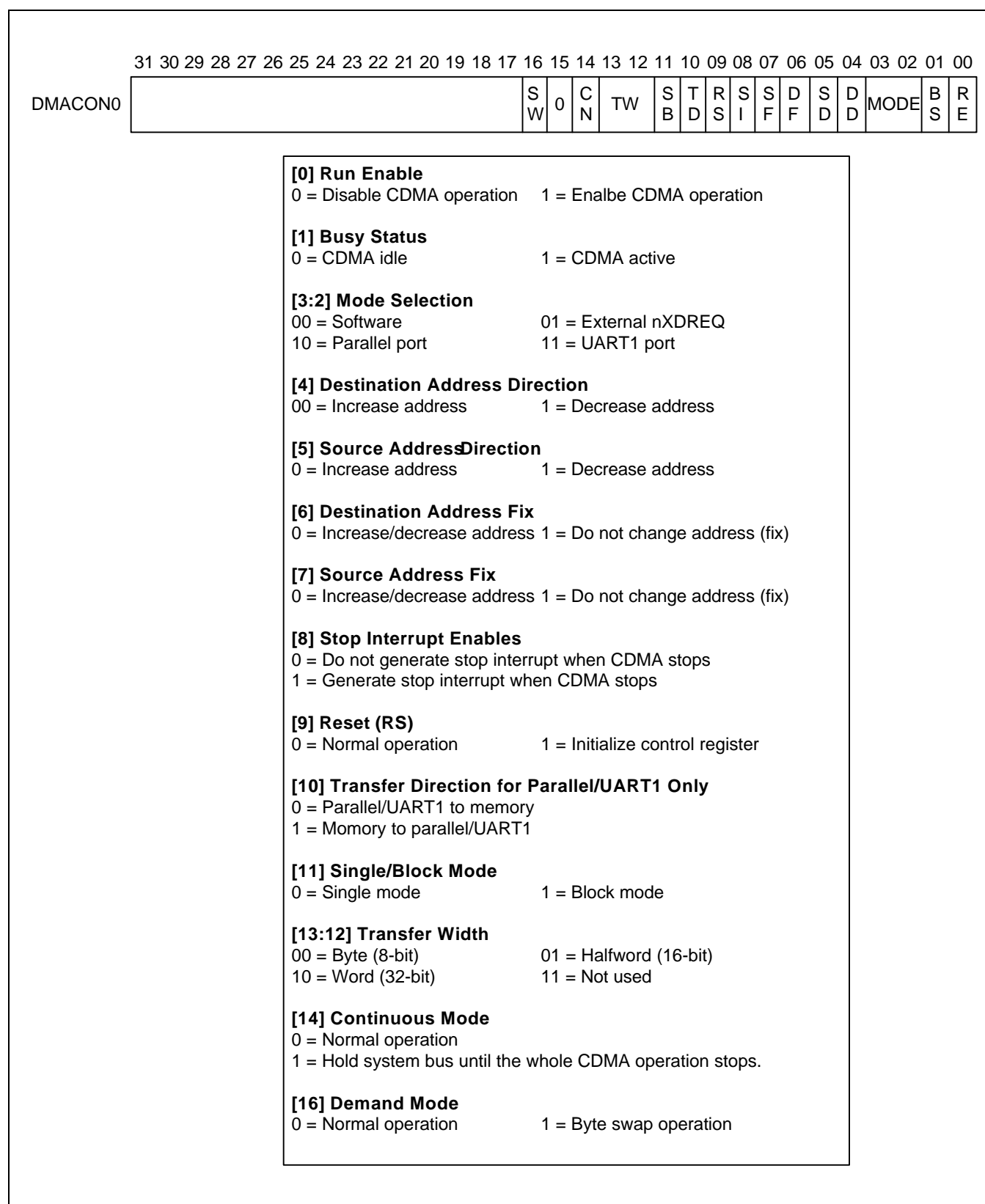


Figure 9-8. CDMA Control Register

CDMA Source/Destination Address Register

These registers contain the 25-bit source/destination address for a CDMA channel.

Depending on the setting of the CDMA control register (DMACON0), these addresses will increase, decrease, or remain the same.

Registers	Offset Address	R/W	Description	Reset Value
DMA_SRC0	0x8804	R/W	CDMA source address register	0xFFFFFFFF
DMA_DST0	0x8808	R/W	CDMA destination address register	0xFFFFFFFF

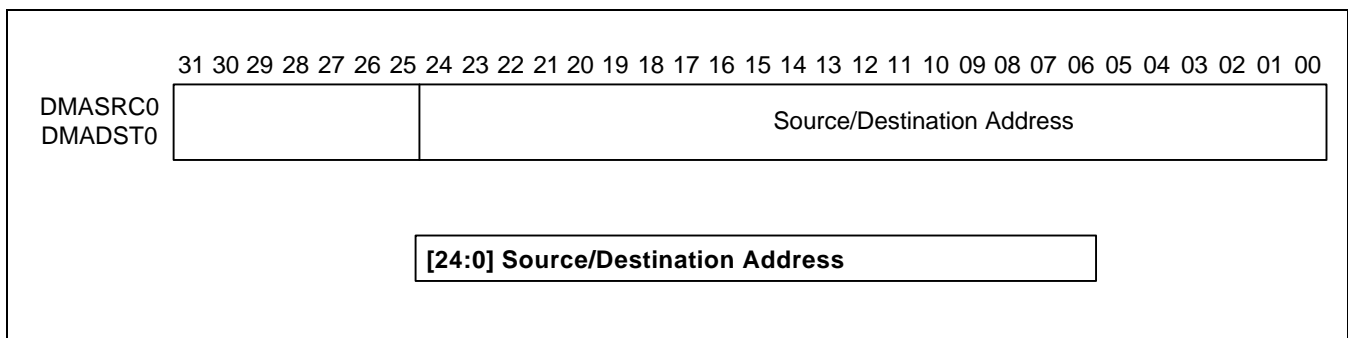


Figure 9-9. CDMA Source/Destination Address Register

CDMA Transfer Count Register

This register contains a 24-bit value which is the number of CDMA transfers completed for CDMA. This value is decreased by 1 when one DMA operation is completed regardless of the width of the data that was transferred.

Register	Offset Address	R/W	Description	Reset Value
DMA_CNT0	0x880c	R/W	CDMA transfer count register	0xFFFFFFFF

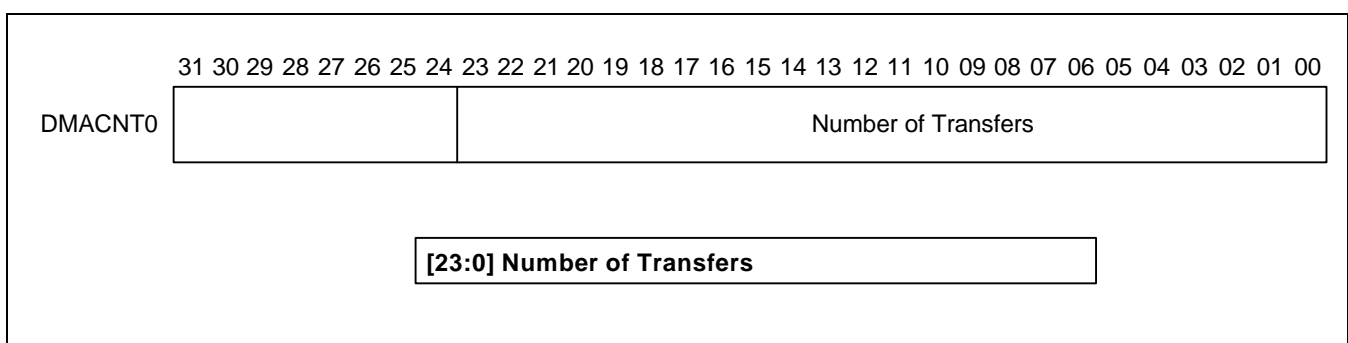


Figure 9-10. CDMA Transfer Count Register

10

PARALLEL PORT INTERFACE

OVERVIEW

The KS32C65100's parallel port interface controller (PPIC) supports four IEEE 1284 standard communication modes:

- Compatibility mode (Centronics TM)
- Nibble mode
- Byte mode
- Enhanced Capabilities Port (ECP) mode

The PPIC also supports all variants of these communication modes, including device ID requests and Run-Length Encoded (RLE) data compression.

The PPIC contains specific hardware to support the following operations:

- Automatic hardware handshaking between host and peripheral in compatible and ECP modes,
- Run-length detection and compression/decompression of host-to-peripheral or peripheral-to-host data during ECP mode transfers.

These features can substantially improve data rates when operating the parallel port in compatibility or ECP mode.

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

NOTE

Here we assume that you are familiar with the parallel port communication protocols specified in IEEE 1284 parallel port standard. If not, we strongly recommend you to read this standard beforehand. It will help you to understand the contents described in this section.

A detailed technical introduction to IEEE 1284 parallel port standard can be found in the web site:

<http://www.fapo.com/ieee1284.htm>

KS32C65100 PPIC OPERATING MODES

The KS32C65100 PPIC supports four kinds of handshaking modes for data transfers:

- Software handshaking mode for forward and reverse data transfers
- Compatibility hardware handshaking mode for forward data transfers
- ECP hardware handshaking without RLE support (ECP-without-RLE) mode for forward and reverse data transfers
- ECP hardware handshaking with RLE support (ECP-with-RLE) mode for forward and reverse data transfers

Mode selection is specified in PPIC control register (PPCON). By setting the PPCON[3:2], one of these four modes can be enabled.

Software Handshaking Mode

This mode is enabled by setting the PPCON's mode-selection bits as "00", i.e. PPCON[3:2] = 00.

In this mode, by using PPIC interrupt event registers (PPINTEN & PPINTPND) and reading/writing PPIC status register (PPSTAT) to detect and control the logic levels on all parallel port signal pins, software can control all parallel port operations, including all four kinds of parallel port communications protocols supported by KS32C65100 (refer to IEEE 1284 standard for operation control). In addition, it also gives software the flexibility to adapt to new and revised protocols.

Compatibility Hardware Handshaking Mode

Compatibility hardware handshaking mode is enabled by setting the PPCON's mode-selection bits as "01", i.e. PPCON[3:2] = 01. In this mode, hardware generates all handshaking signals needed to implement compatibility mode parallel port communication protocol.

When this mode is enabled, the PPIC automatically generates a BUSY signal on receiving the leading edge of nSTROBE from the host, and latches the logic levels on PPD7-PPD0 pins into PPDATA register. The PPIC then waits for nSTROBE to negate and the PPDATA's data field to be read. After the PPDATA is read, the PPIC asserts nACK for the duration specified in the Ack Width Register (PPACKWTH) and then negates the nACK and BUSY signal to conclude the data transfer, as shown in Figure 10-1.

NOTE

Since the initial value of the BUSY-control bit in the PPSTAT register, PPSTAT[3], is "1" after system reset, the BUSY output has a high logic level and handshaking is disabled. To enable hardware handshaking in this mode, the BUSY-control bit PPSTAT[3] must be cleared by software beforehand.

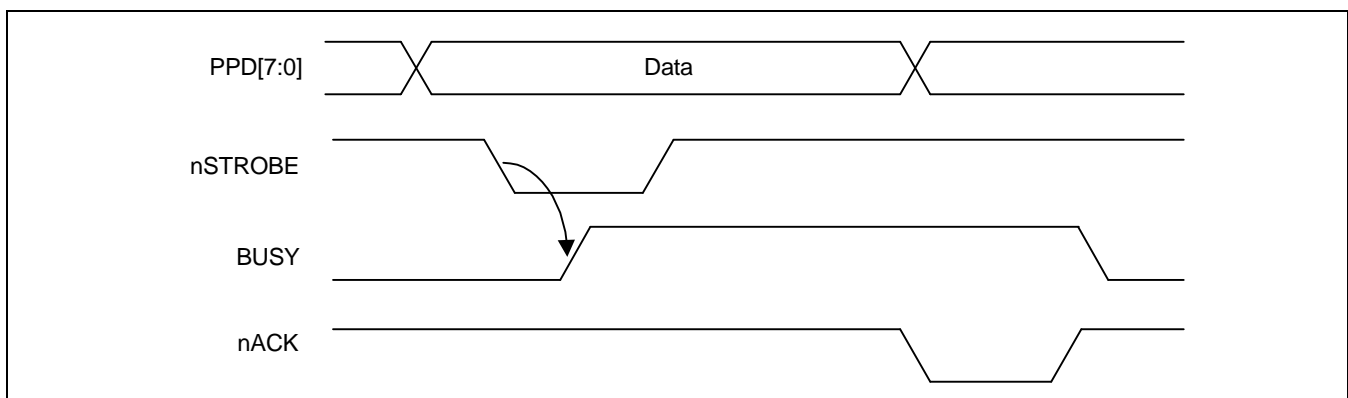


Figure 10-1. Compatibility Hardware Handshaking Timing

ECP-Without-RLE Mode

ECP-without-RLE hardware handshaking mode is enabled by setting the PPCON's mode-selection bits as "10", i.e. PPCON[3:2] = 10. In this mode, hardware generates handshaking signals needed to implement the ECP mode parallel port communication protocol.

When receiving data from host, PPIC automatically responds to the high-to-low transition on nSTROBE by latching the logic levels on PPD7-PPD0 and nAUTOFD in the PPDATA register, in which the nAUTOFD logic level indicates the current data in PPD[7:0] is a data byte or a command byte and is latched to PPDATA[8]. When the PPDATA is read, the PPIC drives BUSY high, waits for nSTROBE to go high, and then drives BUSY low to conclude one forward data transfer operation, as shown in Figure 10-2.

Reception of a command byte, indicated by PPDATA[8] = 0, causes the command received bit in PPIC interrupt pending register, PPINTPND[9], to be set to "1". By examining the PPDATA[7], software will interpret the command byte as a channel address if it is "1" and carry out corresponding operation, or interpret the command byte as a run-length count if it is "0" and then perform data decompression.

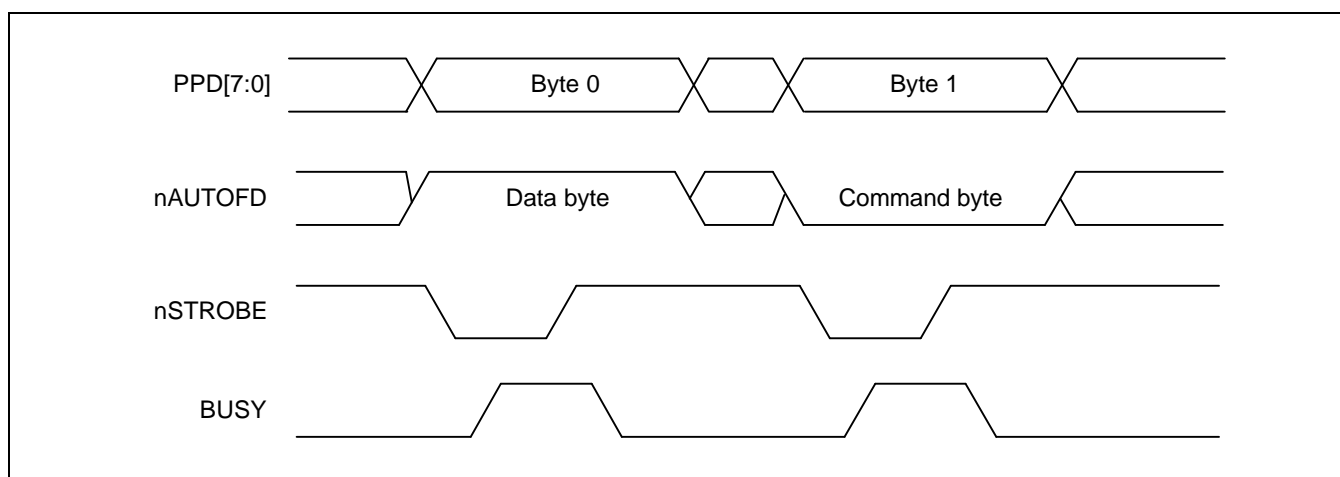


Figure 10-2. ECP Hardware Handshaking Timing (Forward)

During reverse data transfers, software is responsible for data compression and writing data or command bytes in PPDATA to define the logic levels on PPD7-PPD0 and BUSY pins, in which the PPDATA[8] indicates whether the current data in PPDATA[7:0] is a data byte or a command byte and outputs to the BUSY pin. Responding to writing PPDATA, PPIC automatically drives the nACK low, waits for the nAUTOFD to go to high, and then drives nACK high to conclude one reverse data transfer operation, as shown in Figure 10-3.

ECP-with-RLE mode

ECP-with-RLE hardware handshaking mode is enabled by setting the PPCON's mode-selection bits as "11", i.e. $PPCON[3:2] = 11$. In this mode, PPIC performs the same ECP mode handshaking as in ECP-without-RLE mode, except that run-length compression/decompression is also carried out by hardware.

During forward data transfers, PPIC automatically detects and intercepts run-length counts, and carries out data decompression. Only the channel addresses will cause the command received bit in the PPINTPND register, $PPINTPND[9]$, to be set, and software responds by only performing operations associated with it.

Similarly, PPIC automatically carries out the data compression in PPDATA during reverse data transfers.

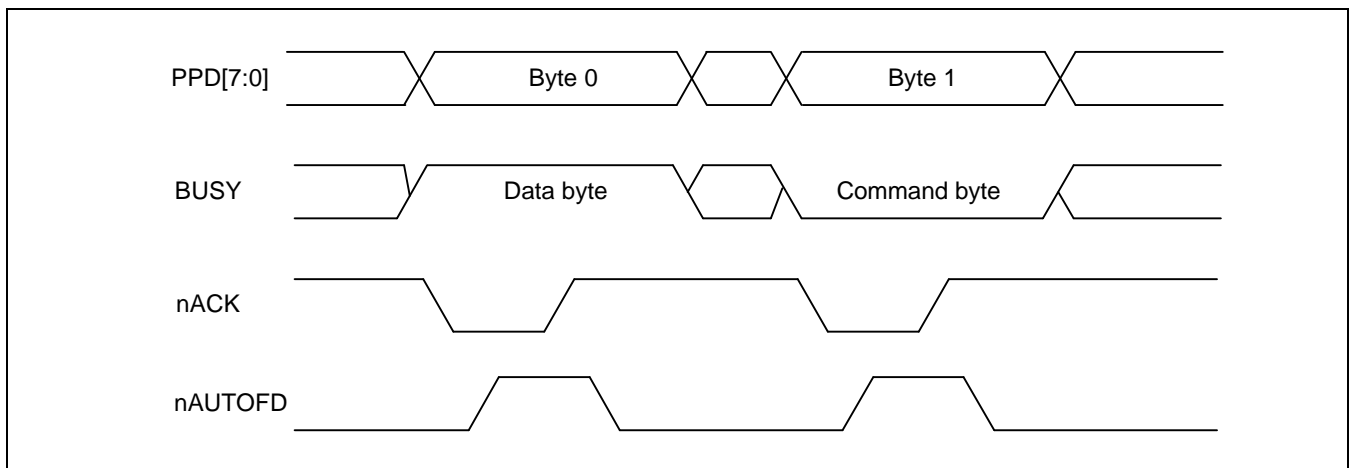


Figure 10-3. ECP Hardware Handshaking Timing (Reverse)

Digital Filtering

KS32C65100 provides the digital filtering function on host control signal inputs, $nSELECTIN$, $nSTROBE$, $nAUTOFD$ and $nINIT$, to improve noise immunity and make the PPIC more impervious to inductive switching noise. The digital filtering function can be enabled regardless of whether hardware handshaking or software handshaking is enabled.

If this function is enabled, the host control signal can be detected only when its input level keeps stable during two sampling periods.

Digital filtering can be disabled to avoid signal missing in some specialized applications with the high bandwidth requirement. Otherwise, it is recommended that digital filtering be enabled.

PPIC SPECIAL REGISTERS

PARALLEL PORT DATA REGISTER

The parallel port data register, PPDATA, contains an 8-bit data field, PPDATA[7:0], that defines the logic level on the parallel port data pins, PPD[7:0]. It also contains a status bit, PPDATA[8], which is used to indicate when a command byte (RLE count or channel address) is received during forward data transfers in ECP mode.

Register	Offset Address	R/W	Description	Reset value
PPDATA	0x8000	R/W	Parallel port data register	0x100

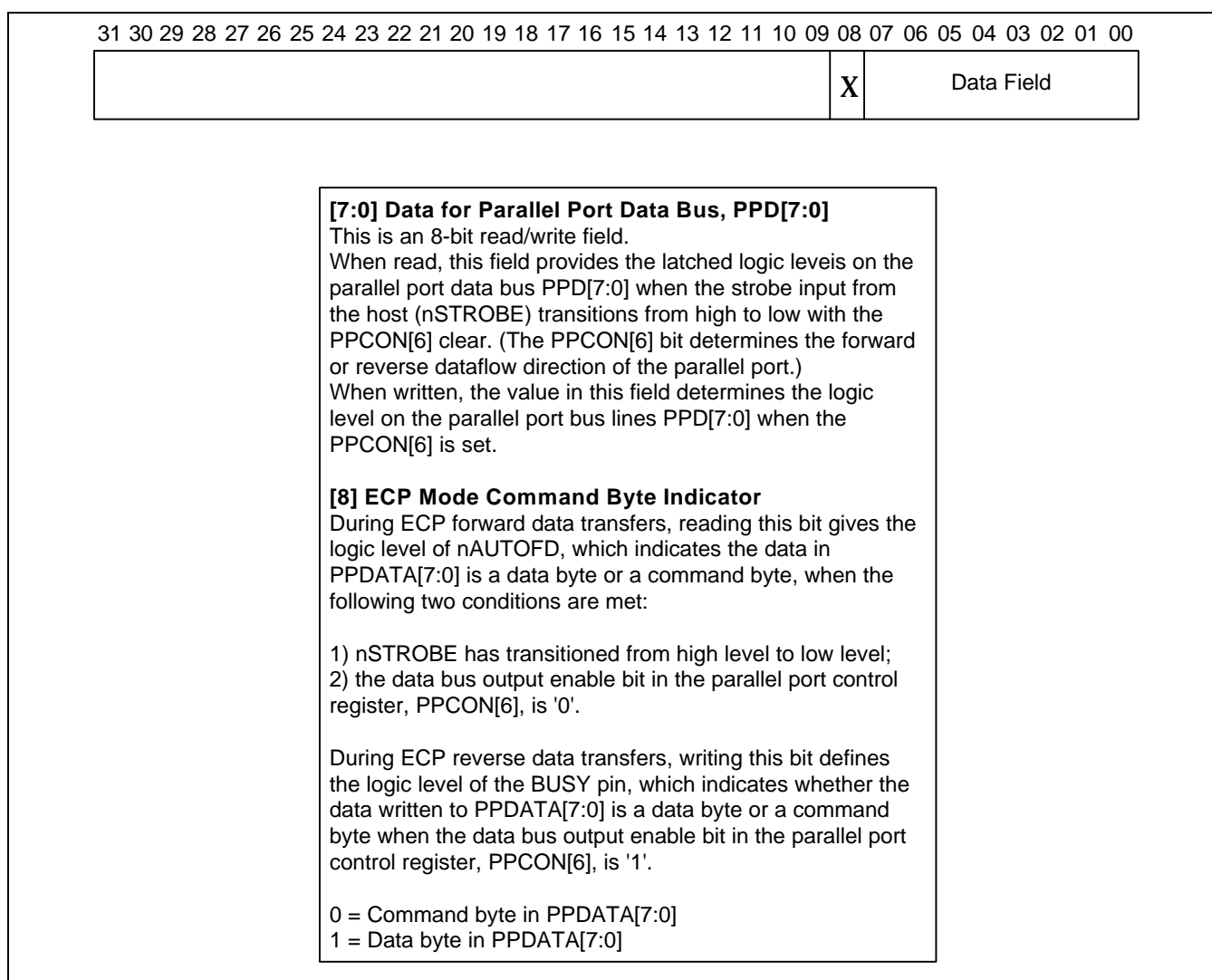


Figure 10-4. Parallel Port Data Register

PARALLEL PORT STATUS REGISTER

The parallel port status register, PPSTAT, contains eleven bits to control the parallel port interface signals. These eleven bits consist of four read-only bits that are used to read the logic level of the host input pins, two read-only bits to read the logic level on the BUSY and nACK output pins, and five read/write bits control the logic levels on the printer output pins, which can be used by software for handshaking control.

Register	Offset Address	R/W	Description	Reset value
PPSTAT	0x8004	R/W	Parallel port status register	0x7e8

[0]	nFAULT control	Setting this bit drives the nFAULT output to low level; clearing it drives the signal High on the external nFAULT pin. nFAULT is used to indicate to the host that there is a fault condition in the printer engine.
[1]	SELECT control	Setting this bit drives SELECT output to High level; clearing it drives the signal low on the external SELECT pin. SELECT indicates to the host that there has been a response from the printer engine.
[2]	PERROR control	Setting this bit drives PERROR output to high level; clearing it drives the signal low on the external PERROR pin. PERROR indicates to the host that a paper error has occurred in the engine.
[3]	BUSY control	Setting this bit drives the external BUSY output to high level. This is generally done to disable hardware handshaking. The PPSTAT[3] bit value is logically ORed with the internal busy signal that is provided by the PPIC to control hardware handshaking operations.
[4]	nACK control	Setting this bit to "1" forces the external nACK output to be driven low. This is generally done when hardware handshaking is disabled. The inverted logic of the PPSTAT[4] bit value is logically ANDed with the internal ACK signal that is provided by the PPIC to control hardware handshaking.
[5]	BUSY status	This read-only bit reflects the logic level on the external BUSY output pin. After a system reset, PPSTAT[3] is "1", which results in PPSTAT[5] being "1". So, for compatibility mode operation, you must clear the PPSTAT[3] by software beforehand so as to enable the hardware handshaking.
[6]	nACK status	This read-only bit reflects the inverted logic level on the external nACK output pin. After a system reset, PPSTAT[6] is "1".
[7]	nSLCTIN status	This read-only bit reflects the level read on the nSLCTIN input pin after synchronization (and optional digital filtering when the digital filtering enable bit, PPCON[1], is set).

[8]	nSTROBE status	This read-only bit reflects the level read on the nSTROBE input pin after synchronization (and optional digital filtering when the digital filtering enable bit, PPCON[1], is set).
[9]	nAUTOFD status	This read-only bit reflects the level read on the nAUTOFD input pin after synchronization (and optional digital filtering when the digital filtering enable bit, PPCON[1], is set).
[10]	nINIT status	This read-only bit reflects the level read on the nINIT input pin after synchronization (and optional digital filtering when the digital filtering enable bit, PPCON[1], is set).

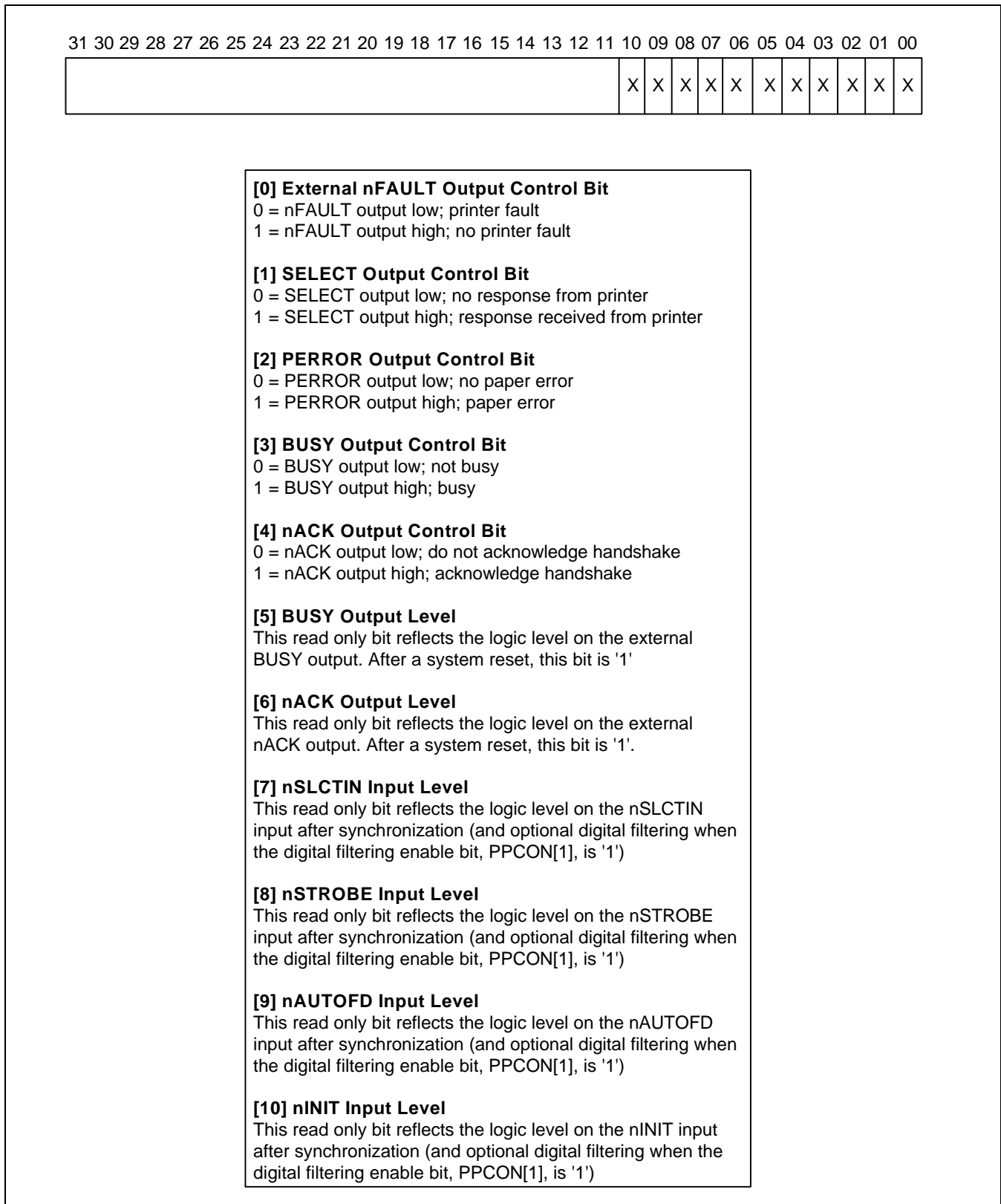


Figure 10-5. Parallel Port Status Register

PARALLEL PORT ACK WIDTH REGISTER

This register contains the 9-bit nACK pulse width field. This value defines the nACK pulse width whenever the parallel port interface controller enters compatibility mode (that is when the parallel port control register mode bits, PPCON[3:2] are "01"). The nACK pulse width can be selected from 0 to 511 MCLK periods.

The nACK pulse width can be modified at any time and with any PPIC operation mode selected, but it can only be used during a compatibility handshaking cycle. If you change the nACK width near the end of a data transfer (when nACK is already low), the new pulse width value does not affect the current cycle. The new pulse width value would be used at the start of the next cycle.

Register	Offset Address	R/W	Description	Reset value
PPACKWTH	0x8008	R/W	Parallel port acknowledge width register	0xXXX

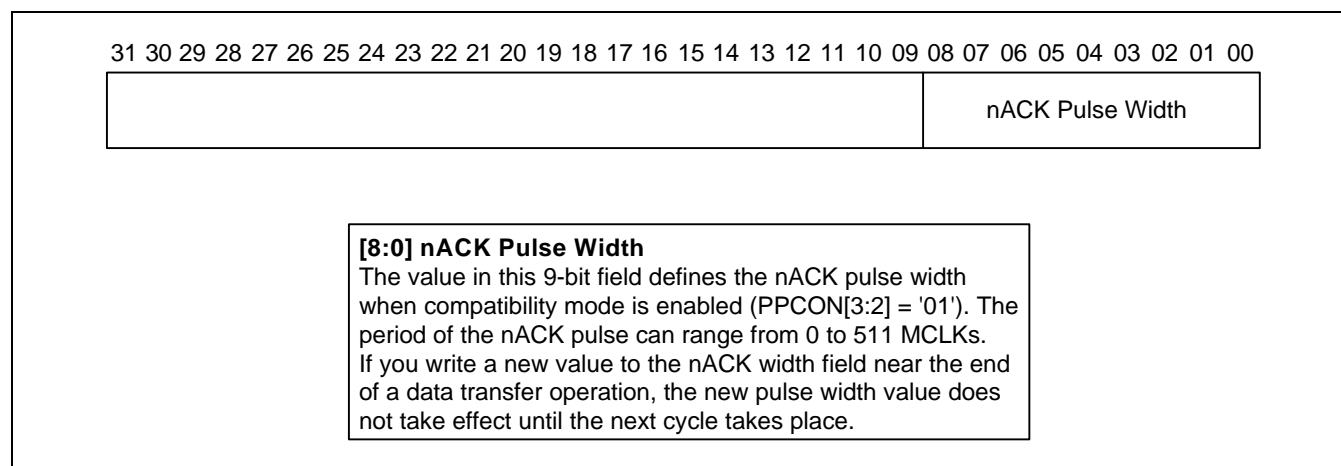


Figure 10-6. Parallel Port ACK Width Register

PARALLEL PORT CONTROL REGISTER

The parallel port control register, PPCON, is used to configure the PPI operations, such as handshaking, digital filtering, operating mode, data bus output, abort operations, and DMA. PPCON[15:13] are read-only.

Register	Offset Address	R/W	Description	Reset value
PPCON	0x800c	R/W	Parallel port control register	0x0000

[0]	Software reset		Setting the software reset bit causes the PPIC's handshaking control and compression/decompression logic to immediately terminate the current operation and return to software Idle state. When PPCON[0] is set to "1", the run-length decompression status bit, PPCON[13], and the full status bit, PPCON[14], are automatically cleared to "0".
[1]	Digital filter enable		Setting this bit enables digital filtering on all four host control signal inputs: nSELECTIN, nSTROBE, nAUTOFD, and nINIT.
[3:2]	Mode selection		<p>This two-bit value selects the current operating mode of the parallel port interface (see Figure 14-4). Software mode: disables all hardware handshaking so that handshaking can be performed by software.</p> <p>Compatibility mode: Compatibility mode hardware handshaking can be enabled during a forward data transfer. You can change the mode selection at any time, but if a compatibility mode operation is currently in-progress, it will be completed as normal.</p> <p>Mode should be changed from compatibility mode only when BUSY is high level. This ensures that there is no parallel port activity during the time when the parallel port is being re-configured. ECP-without-RLE mode: ECP mode hardware handshaking without RLE support can be enabled during forward or reverse data transfers. You can change the mode selection at any time, but if an ECP cycle is currently in progress, it will be completed as normal.</p> <p>ECP-with-RLE mode: ECP mode hardware handshaking with RLE support can be enabled during forward or reverse data transfers.</p> <p>Change on the mode selection doesn't affect current data transfer operation, including compression/ decompression, until it completes.</p> <p>To immediately abort an operation, you can set the software-reset bit, PPCON[0], to "1".</p>
[4]	ECP direction		This bit determines the direction of ECP is forward or reverse. If this bit is set to '1', then the ECP is operated in reverse direction.

- | | | |
|-----|------------------------|--|
| [5] | Error cycle | <p>The error cycle bit is used to execute an error cycle when in compatibility mode. When PPCON[5] is set to "1", the BUSY status bit in the parallel port interface register, PPSTAT[5], is set to "1". This immediately causes the KS32C65100 to drive the BUSY level high.</p> <p>If you set the error cycle bit when a compatibility mode handshaking sequence is in progress, PPSTAT[5] will remain set to "1" beyond the end of the current cycle. The error cycle bit does not affect the nACK pulse if it is already active, but it will prevent an nACK pulse if it is about to be generated.</p> <p>When PPCON[5] is "1", software can set or clear the parallel port status register control bits: PPSTAT[0] (nFAULT control), PPSTAT[1] (SELECT control), and PPSTAT[2] (PERROR control). When PPCON[5] is cleared to "0", the parallel port interface controller generates an nACK pulse and negates BUSY to conclude the error cycle.</p> |
| [6] | Data bus output enable | <p>The parallel port data bus output enable bit performs two functions: 1) It controls the state of the tri-state output drivers, and 2) It qualifies the latching of data from the output drivers into the parallel port interface register 0's data field, PPDATA[7:0].</p> <p>When PPCON[6] is "0", parallel port data bus output are disabled. This allows data to be latched into the PPDATA data field. When PPCON[6] is "1", PPD output are enabled and data is prevented from being latched into the PPDATA data field. In this frozen state, the data field is unaffected by transitions of nSTROBE.</p> <p>The setting of the abort bit, PPCON[7], affects the operation of the data bus output enable bit, PPCON[6]. If PPCON[7] is "1", nSELECTIN must remain high to allow PPCON[6] to be set or to remain set. If PPCON[6] is "1" and nSELECTIN goes low, PPCON[6] is cleared and setting this bit will have no effect. The external PPD[7:0] outputs reflect the current state of PPCON[6].</p> |
| [7] | Abort | <p>The abort bit causes the parallel port interface controller to use nSELECTIN to detect when the host suddenly aborts a reverse transfer and returns to compatibility mode. If PPCON[7] is "1", a low level on nSELECTIN causes the parallel port data bus output enable bit, PPCON[6], to be cleared and the output drivers for the data bus lines PPD[7:0] to be tri-stated.</p> |
| [8] | DMA selection | <p>The PPIC can issue a DMA request during a forward data transfer in compatibility mode, ECP-without-RLE mode, or in ECP-with-RLE mode, if the DMA request enable bit, PPCON[7], is set. The DMA selection bit determines which DMA channel is used for forward data transfer. When PPCON[8] is "0", DMA channel 0 is used; when it is "1", DMA channel 1 is used.</p> |

[9]	DMA request enable	When this bit is set to "1", the PPIC issues a DMA request to DMA channel 0 or 1 during a forward data transfer. otherwise, an interrupt is requested for the data transfer.
[10]	Flush request	When this bit is set to "1", the PPIC issues a DMA request to send the remaining data to parallel port. The remaining data means run-length code and data in the PPIC's buffer while reverse ECP mode is operating.
[12]	Zero insert	When the run-length count is '0', this bit specifies whether to send the RLE count to PPIC during ECP-with-RLE reverse data transfers. If this bit is set to '1', then the count "0" will be sent, but if otherwise, it is not sent.
[13]	RLE status	This bit indicates the run-length decompression is taking place during forward data transfers in ECP-with-RLE mode. It is set when a run-length count is received and loaded into the internal counter, and cleared when the last read of the PPD's data field takes place.
[14]	Data latch status	If a data is latched to PPDATA, then this bit is set to '1'. It is automatically cleared while PPDATA is read.
[15]	Write status	When reverse ECP mode, this bit specifies the PPDATA is empty. It is automatically cleared while PPDATA is written with a new data.

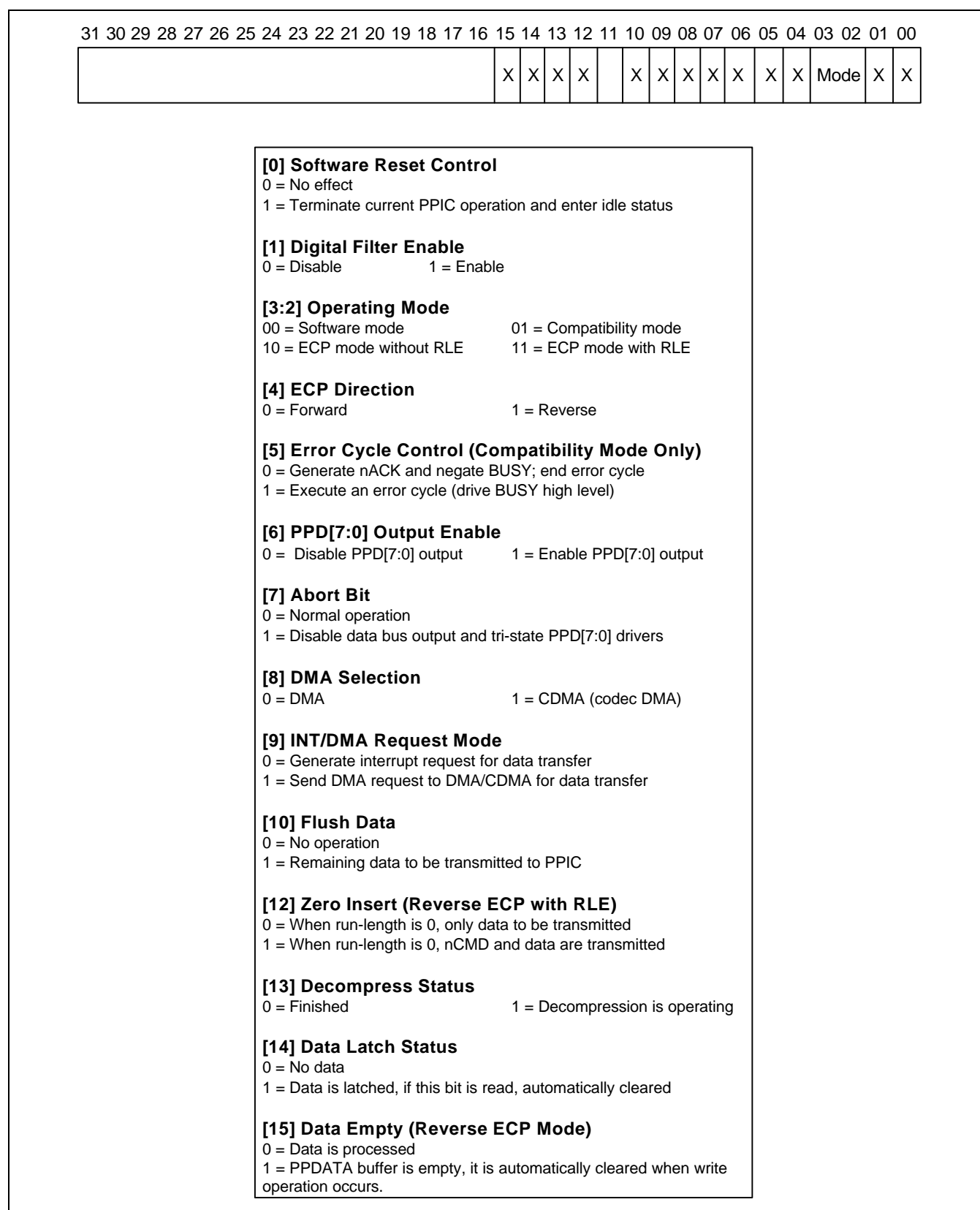


Figure10-7. Parallel Port Control Register

PARALLEL PORT INTERRUPT EVENT REGISTERS (PPINTEN, PPINTPND)

The two parallel port interrupt event registers, PPINTEN and PPINTPND, control interrupt-related events for the input signal originating from the host, as well as data reception, command reception, and invalid events. Enable register, PPINTEN, contains the interrupt enable bits for each interrupt event that is indicated by the PPINTPND status bits. If its PPINTEN enable bit is "1", the corresponding event causes the KS32C65100 CPU to generate an interrupt request. Otherwise, no interrupt request is issued.

Registers	Offset Address	R/W	Description	Reset value
PPINTEN	0x8010	R/W	Parallel port enable interrupt event register	0x000
PPINTPND	0x8014	R/W	Parallel port interrupt pending register	0x000

[0]	nSLCTIN Low-to-High	The bit of PPINTPND is set when a Low-to-High transition on nSLCTIN is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.		
[1]	nSLCTIN High-to-Low	The bit of PPINTPND is set when a High-to-Low transition on nSLCTIN is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.		
[2]	nSTROBE Low-to-High	The bit of PPINTPND is set when a Low-to-High transition on nSTROBE is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.		
[3]	nSTROBE High-to-Low	The bit of PPINTPND is set when a High-to-Low transition on nSTROBE is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.		
[4]	nAUTOFD Low-to-High	The bit of PPINTPND is set when a Low-to-High transition on nAUTOFD is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.		
[5]	nAUTOFD High-to-Low	The bit of PPINTPND is set when a High-to-Low transition on nAUTOFD is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.		
[6]	nINIT Low-to-High	The bit of PPINTPND is set when a Low-to-High transition on nINIT is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.		
[7]	nINIT High-to-Low	The bit of PPINTPND is set when a High-to-Low transition on nINIT is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.		

[8]	Data received	The bit of PPINTPND is set when data is latched into the PPDATA register's data field. This occurs on every High-to-Low transition of nSTROBE when the parallel port data bus enable bit, PPCON[6], is "0". An interrupt is also generated if ECP-with-RLE mode is enabled, and if a data decompression is in progress.
[9]	Command received	The bit of PPINTPND is set when a command byte is latched into the PPDATA register data field. If ECP-without-RLE mode is enabled, a command received interrupt is issued whenever a run-length or channel address is received. If ECP-with-RLE mode is enabled, a command received interrupt is issued only when a channel address is received. This event can be posted only when ECP mode is enabled. The corresponding enable bit in the PPINTEN register determines whether an interrupt request will be generated when a command byte is received.
[10]	Invalid transition	The bit of PPINTPND is set when nSLCTIN transitions from High-to-Low in the middle of an ECP forward data transfer handshaking sequence. This interrupt is issued if nSLCTIN is Low when nSTROBE is Low or when BUSY is High. This event can only be detected when ECP mode is enabled.
[11]	Transmit data empty	The PPINTPND bit is set when the transmit data register (= PPDATA) can be written in the middle of an ECP reverse data transfer handshake sequence.

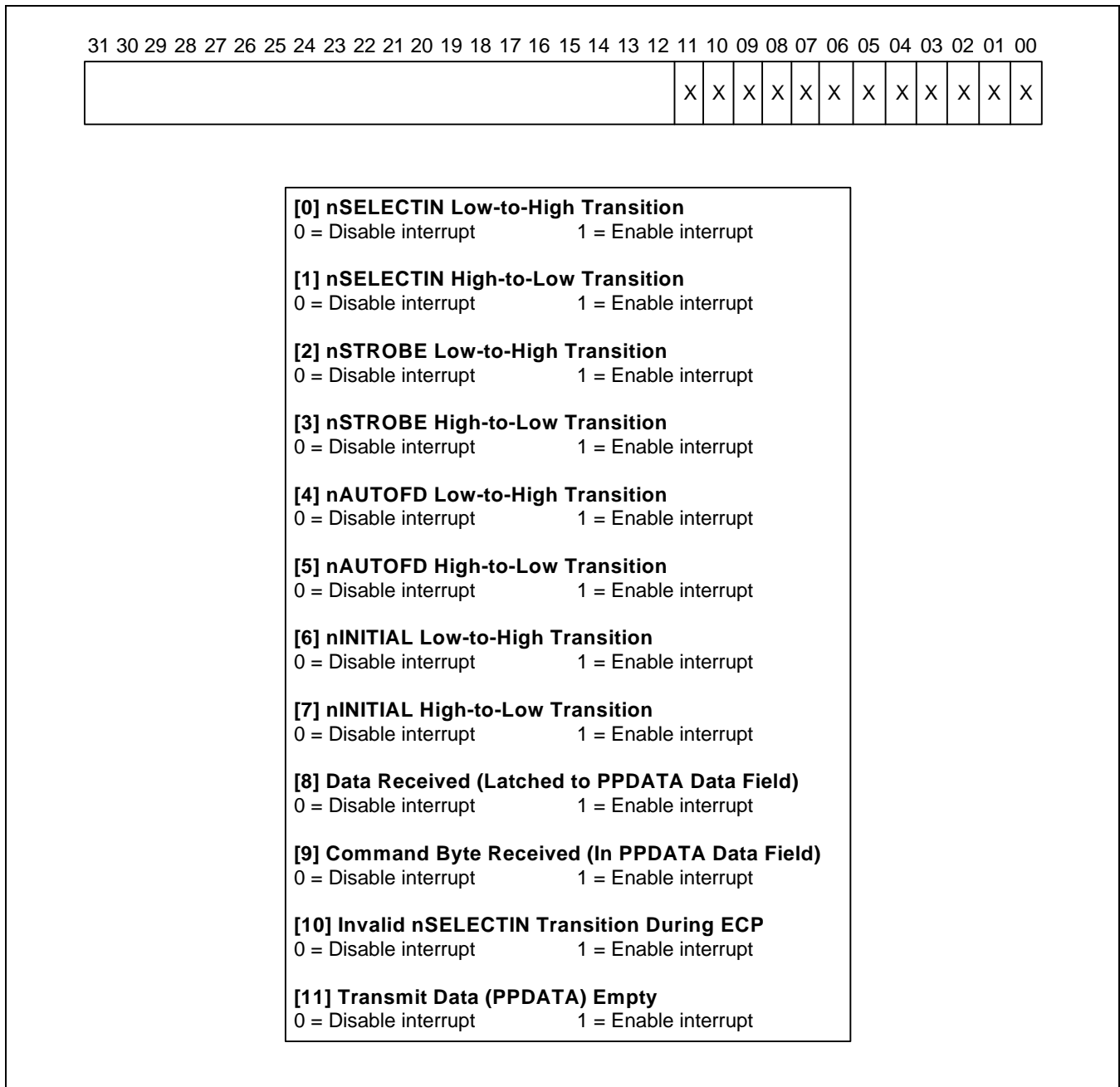


Figure 10-8. Parallel Port Event Interrupt Enable Register (PPINTEN)

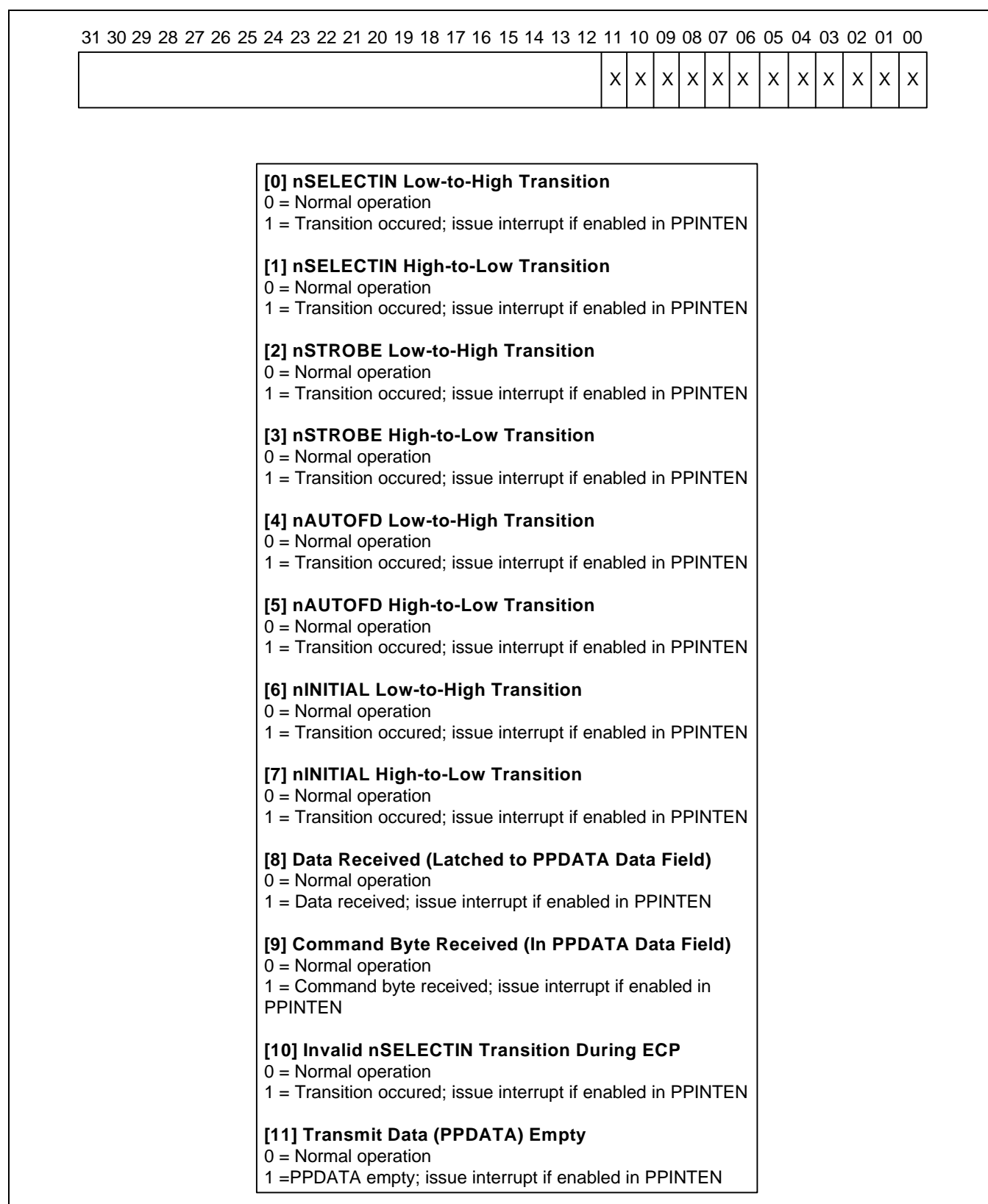


Figure 10-9. Parallel Port Event Interrupt Pending Register (PPINTPND)

11

UART

OVERVIEW

The KS32C65100 UART(Universal Asynchronous Receiver and Transmitter) unit provides two independent asynchronous serial I/O (SIO) ports, each of which can operate in interrupt-based or DMA-based mode. For example, SIO can generate an interrupt or a DMA request for data transfers between CPU and SIO.

Main features of the KS32C65100 UART include programmable baudrates, infra-red (IR) transmit/receive, one or two stop bit insertion, 5-bit, 6-bit, 7-bit or 8-bit data transfers, and parity checking.

Each SIO contains a baud-rate generator, transmitter, receiver and a control unit, as shown in Figure11-1. The baud-rate generator can be clocked by the internal system clock (MCLK). The transmitter and receiver contain data buffer registers and data shifters. Data to be transmitted is written to the transmit buffer register and then copied to the transmit shifter and shifted out by the transmit data pin (TXDn). Data received is shifted in by the receive data pin (RXDn), and then copied from shifter to receive buffer register once one data byte has been received. The control unit will provide controls for mode selection, and status/interrupt generation..

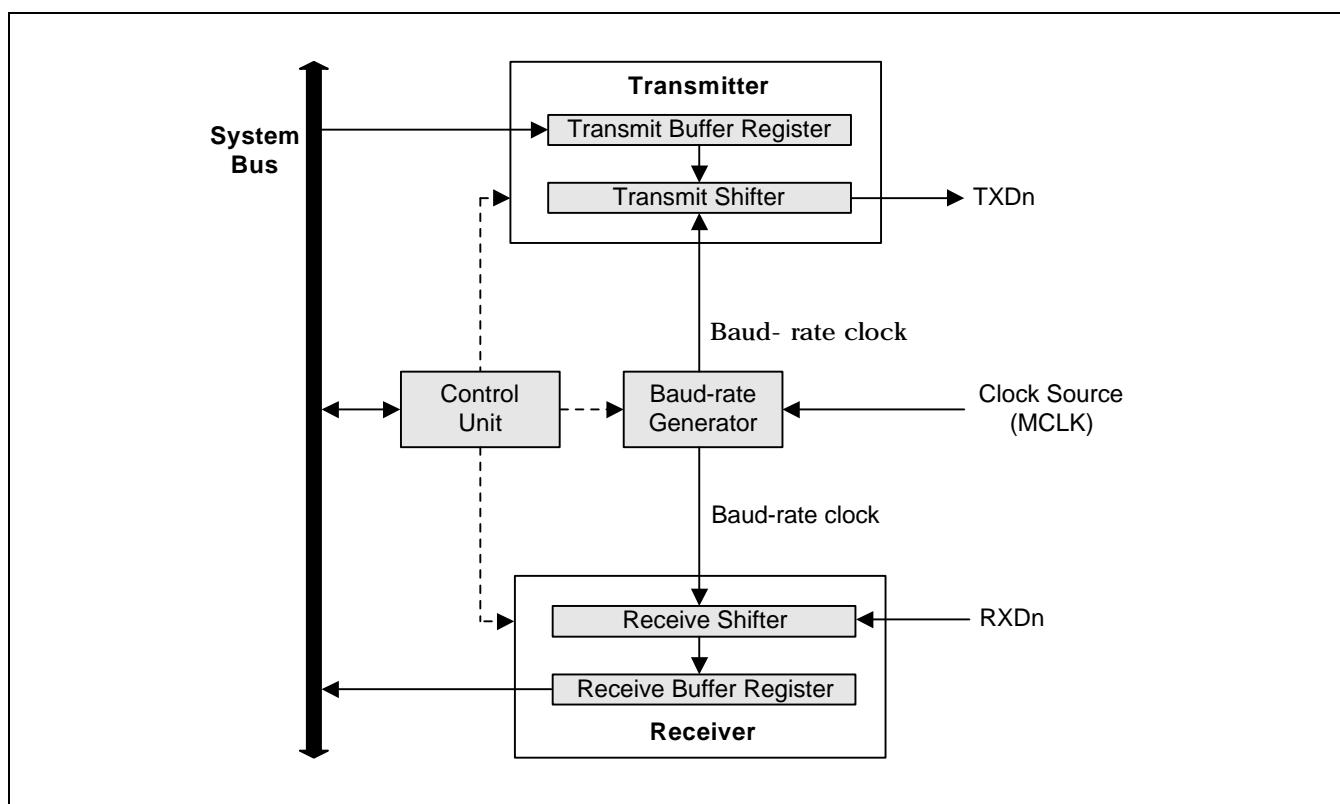


Figure 11-1. UART Block Diagram

UART OPERATION

The following sections describe the UART operations that include infra-red mode, loop-back mode, interrupt generation, baud-rate generation, data transmission, data reception and so on.

Infra-Red Mode

The KS32C65100 UART block supports infra-red (IR) transmit and receive which can be selected by setting the infra-red-mode bit in the line control register (ULCONn).

The implementation on the mode is shown in Figure 11-2.

In IR mode, the transmit period is pulsed at a rate of 3/16 the of the normal serial transmit rate (when the transmit data value in the TBR register is zero); and in IR receive mode, the receiver must detect the 3/16 pulsed period to recognize a zero value in the receive buffer register, RBR, as the IR receive data. (refer to the frame timing diagrams shown in Figure 11-15 and 11-16)

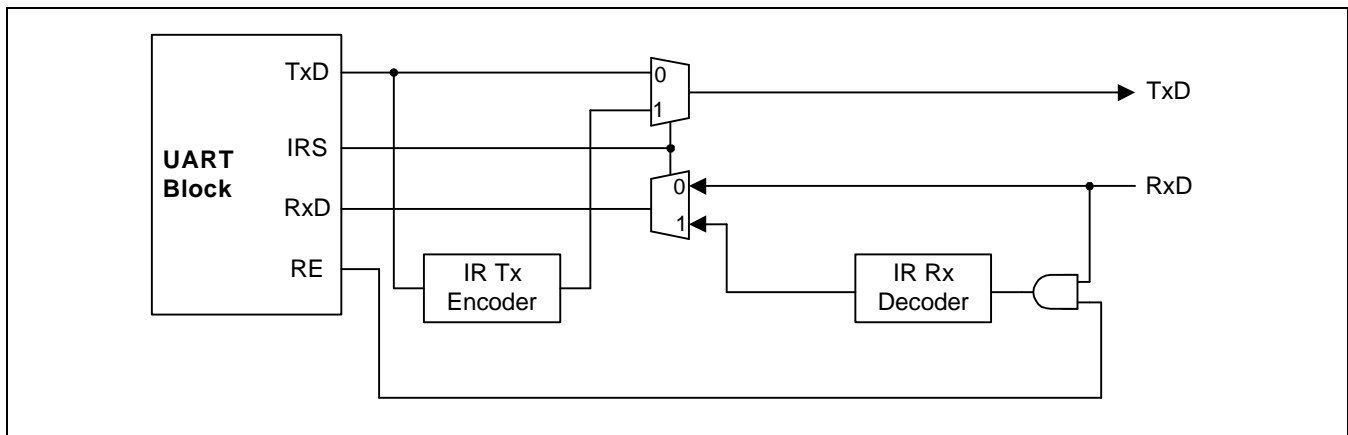


Figure 11-2. UART Block Diagram

Loop-back Mode

The KS32C65100 UART provides a test mode, referred to as the loop-back mode, to aid in isolating faults in the communications link. In this mode, data that is transmitted is immediately received. This feature allows the processor to verify the internal transmit and receive data paths of each SIO channel.

This mode can be selected by setting the loop-back-bit in UART control register (UCONn).

Interrupt/DMA Request Generation

Each SIO of KS32C65100 UART has seven status signals: overrun error, parity error, frame error, break, receive buffer full, transmit buffer register empty and transmitter empty, all of which are indicated by the corresponding UART status register (USTATn).

The overrun error, parity error, frame error and break condition are referred as the receive status, each of which can cause the receive status interrupt request, (i.e. the error interrupt to be mentioned in Section 19, if the receive-status-interrupt-enable bit is set in control register UCONn). When a receive status interrupt request is detected, you can determine which signal caused the request by reading the status register (USTATn).

When the receiver transfers data from its shifter to its buffer, it activates the receive buffer full status signal which will cause the receive interrupt, If the receive mode in control register is selected as interrupt mode; and when the transmitter transfers data from its transmit buffer register to its shifter, it activates

the transmit buffer register empty status signal which will cause the transmit interrupt if the transmit mode in control register is selected as interrupt mode.

The receive buffer full and transmit buffer register empty status signals can also be connected to generate the DMA request signals if the receive/transmit mode in control register is selected as DMA mode. As mentioned before, two DMA channels, GDMA and CDMA, are provided in KS32C65100. However, each SIO can only be connected with a fixed DMA channel. In other words, the UART0 can only generate the GDMA request, the UART1 can only generate the CDMA request, and UART2 can not generate any DMA request.

Baud-Rate Generation

Each UART's baud-rate generator provides the serial clock for transmitter and receiver. The source clock for the baud-rate generator is KS32C65100's internal system clock (MCLK). The baud-rate clock is generated by dividing the source clock by 16 and a 16-bit divisor specified by UART baud rate divisor register (UBRDIVn). UBRDIVn can be determined as follows:

$$\text{UBRDIVn} = (\text{int})(\text{source_clock} / (\text{bps} \times 16)) - 1$$

where the divisor should be a value from 1 to $(2^{16}-1)$.

For example, if the baud rate is 56000bps and MCLK is 33Mhz (use internal system clock), UBRDIVn is calculated as follows:

$$\begin{aligned}\text{UBRDIVn} &= (\text{int})(33000000 / (56000 \times 16)) - 1 \\ &= (\text{int})(36.83) - 1 \\ &= 36 - 1 = 35\end{aligned}$$

;

Data Transmission

The data frame for transmissions is programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits, which can be specified in the line control register (ULCONn). The transmitter can also produce break conditions. The break condition forces the serial output to logic 0 state for a duration longer than one frame transmission time. On the receiving end, a break condition sets an error flag as mentioned above.

The data transmission process is shown in Figure 11-3, in which the transmitter transfers data through such a path: data source → transmit buffer register → transmit shift register → TXDn pin, and completes parallel-to-serial data conversions. Two flags (status signals), transmit buffer register empty and transmitter empty, are used to indicate the status of the transmit buffer register and transmitter which includes both the buffer register and transmit shifter.

Data Reception

Like the transmissions, the data frame for receptions is also programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits, as the settings in the line control register (ULCONn). The receiver can detect overrun error, parity error, frame error and break condition, each of which can set an error flag. The overrun error indicates that new data has overwritten old data before the old data has been read. The parity error indicates the receiver has detected a parity condition other than what it was programmed for. The frame error indicates that the received data did not have a valid stop bit. The break condition indicates that the received data input is held in the logic 0 state for a duration longer than one frame transmission time.

The data reception process is shown in Figure 11-4, in which the receiver transfers data through such a path: RXDn pin → receive shift register → receive buffer register → destination, and completes serial-to-parallel data conversions. In addition to receive error status flags, a receive buffer full flag is used to indicate the status of the receive buffer register.

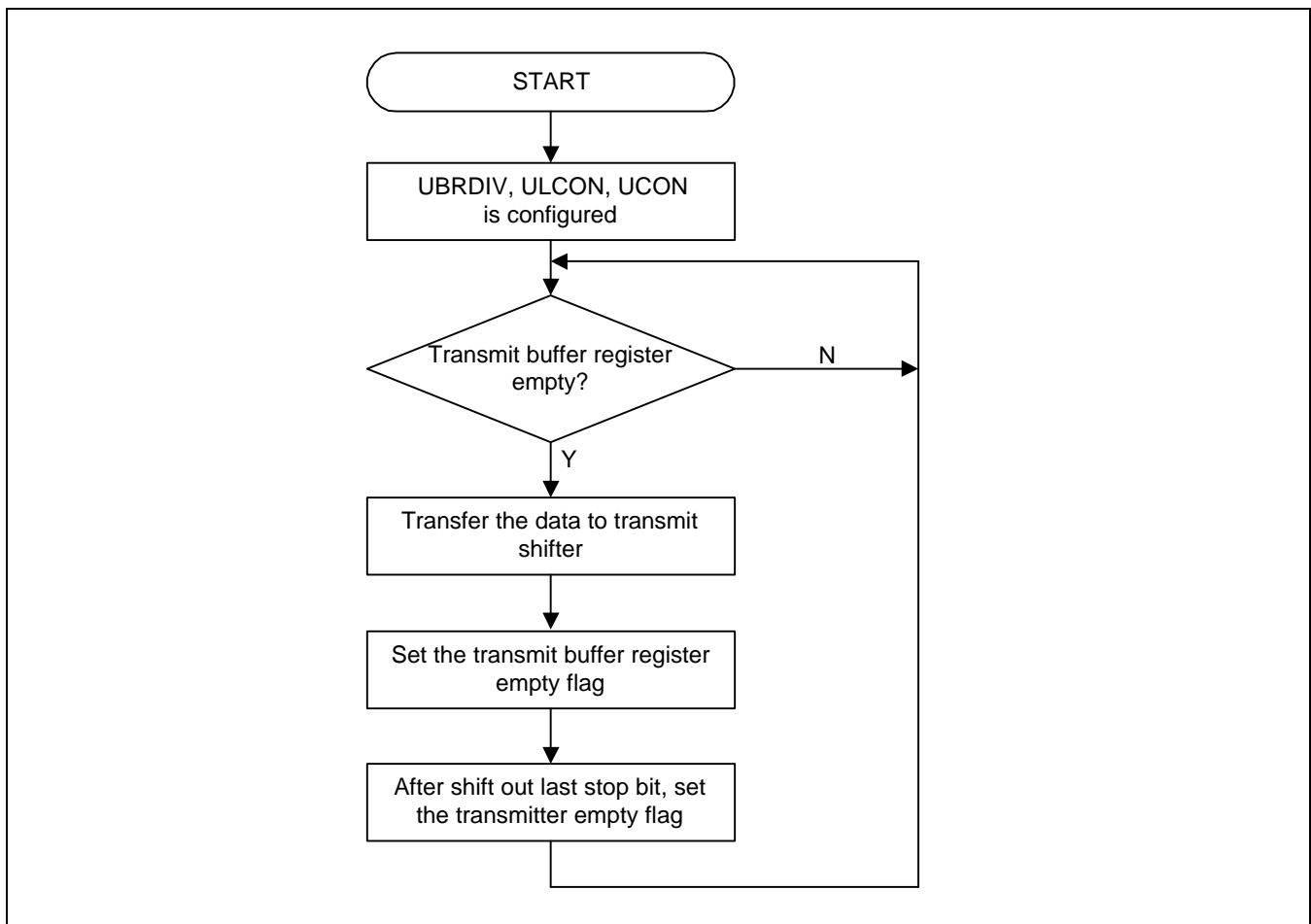
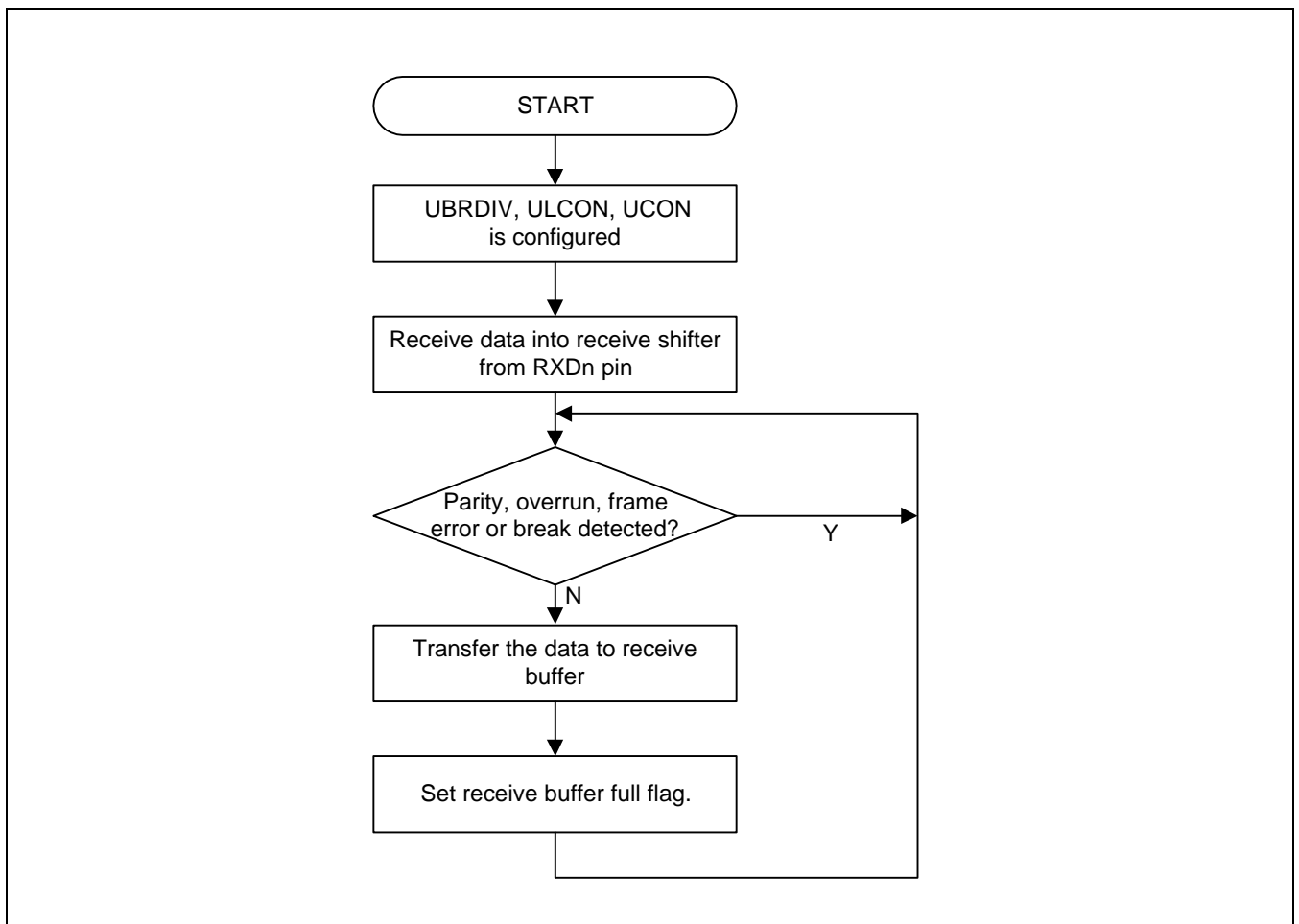


Figure 11-3. UART Data Transmission Process

**Figure 11-4. UART Data Reception Process**

UART SPECIAL REGISTERS**UART Line Control Register**

There are three identical UART line control registers (ULCON0, 1, 2) in the UART block, one for each UART channel.

Registers	Offset Address	R/W	Description	Reset Value
ULCON0	0xb000	R/W	UART ch-0 line control register	0x00
ULCON1	0xb800	R/W	UART ch-1 line control register	0x00
ULCON2	0xc000	R/W	UART ch-2 line control register	0x00

[1:0]	Word length(WL)	The two-bit word length value indicates the number of data bits to be transmitted or received per frame. 00 = 5bits 01 = 6bits 10 = 7bits 11 = 8bits
[2]	No. of stop bit	ULCON[2] specifies how many stop bits are used to signal end-of-frame (EOF). 0 = One stop bit per frame 1 = Two stop bit per frame
[5:3]	Parity mode(PM)	The 3-bit parity mode value specifies how parity generation and checking are to be performed during UART transmit and receive operations. 0xx = No parity 100 = Odd parity 101 = Even parity 110 = Parity forced/checked as "1". 111 = Parity forced/checked as "0".
[6]	Reserved	This bit must be '0'.
[7]	Infra-red mode	This bit determines whether to use the infra-red mode 0 = Normal mode operation 1 = Infra-red Tx/Rx mode

NOTE: ULCONn has to be configured before UCONn is configured.

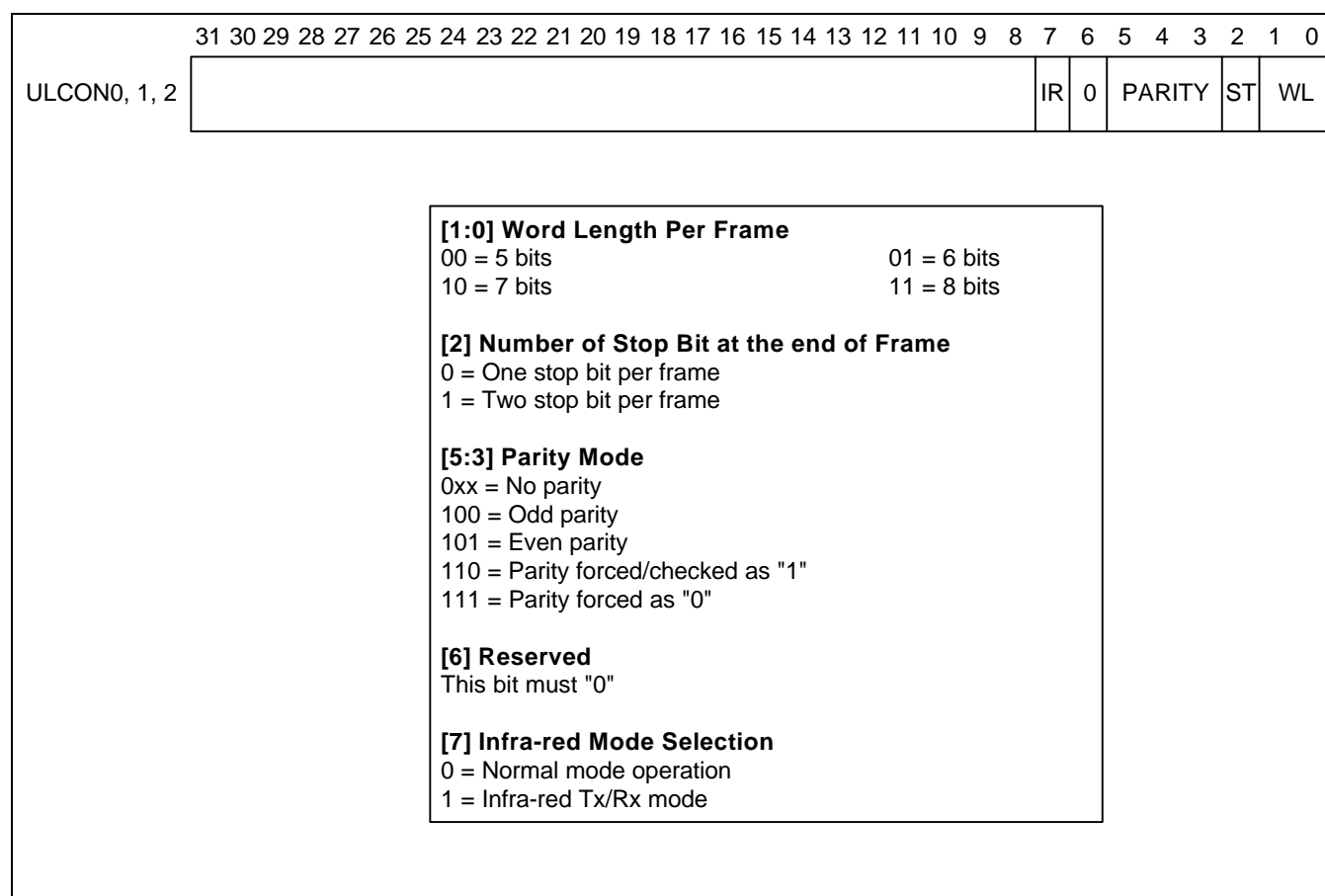


Figure 11-5. UART Line Control Register (ULCON0, 1, 2)

UART Control Register

There are two identical UART control registers (UCON0,1,2) in the UART block, each for a UART channel. UCONn has to be configured after ULCONn is configured.

Registers	Offset Address	R/W	Description	Reset Value
UCON0	0xb004	R/W	UART ch-0 control register	0x00
UCON1	0xb804	R/W	UART ch-1 control register	0x00
UCON2	0xc004	R/W	UART ch-2 control register	0x00

[1:0] Receive mode (RxM)

This two-bit value determines which function is currently able to read data from the UART receive buffer register, RBR. The difference between UCON0 and UCON1 should be noted. UART0 can only generate GDMA requests, UART1 can only generate CDMA requests, and UART2 cannot generate any DMA requests.

[2]	Rx status interrupt enable	<p>This bit enables the UART to generate an interrupt if an exception (break, frame error, parity error, or overrun error) occurs during a receive operation.</p> <p>0 = Do not generate receive status interrupt 1 = Generate receive status interrupt</p>
[4:3]	Transmit mode (TxM)	<p>This two-bit value determines which function is currently able to write Tx data to the UART transmit buffer registers, TBR. The difference between UCON0 and UCON1 should be noted. UART0 can only generate GDMA requests, UART1 can only generate CDMA requests, and UART2 cannot generate any DMA requests.</p>
[6]	Send break	<p>Setting UCON[6] causes the UART to send a break. Break is defined as a continuous Low level signal on the transmit data output with a duration of more than one frame transmission time. By setting this bit when the transmitter is empty(transmitter empty bit, USTAT[7] = "1"), you can use the transmitter to time the frame. When USTAT[7] is "1", write the transmit buffer register, TBR, with the data to be transmitted, then poll the USTAT[7] value. When it returns to "1", clear (reset) the send break bit, UCON[6].</p> <p>0 = Do not send break 1 = Send break</p>
[7]	Loop-back bit	<p>Setting this bit causes the UART to enter loop-back mode. In loop-back mode, the transmit data output is sent to high level and the transmit buffer register (TBR) is internally connected to the receive buffer register (RBR). This mode is provided for test purposes only.</p> <p>0 = Normal SIO operation mode 1 = Enable SIO loop-back mode (for testing only)</p>

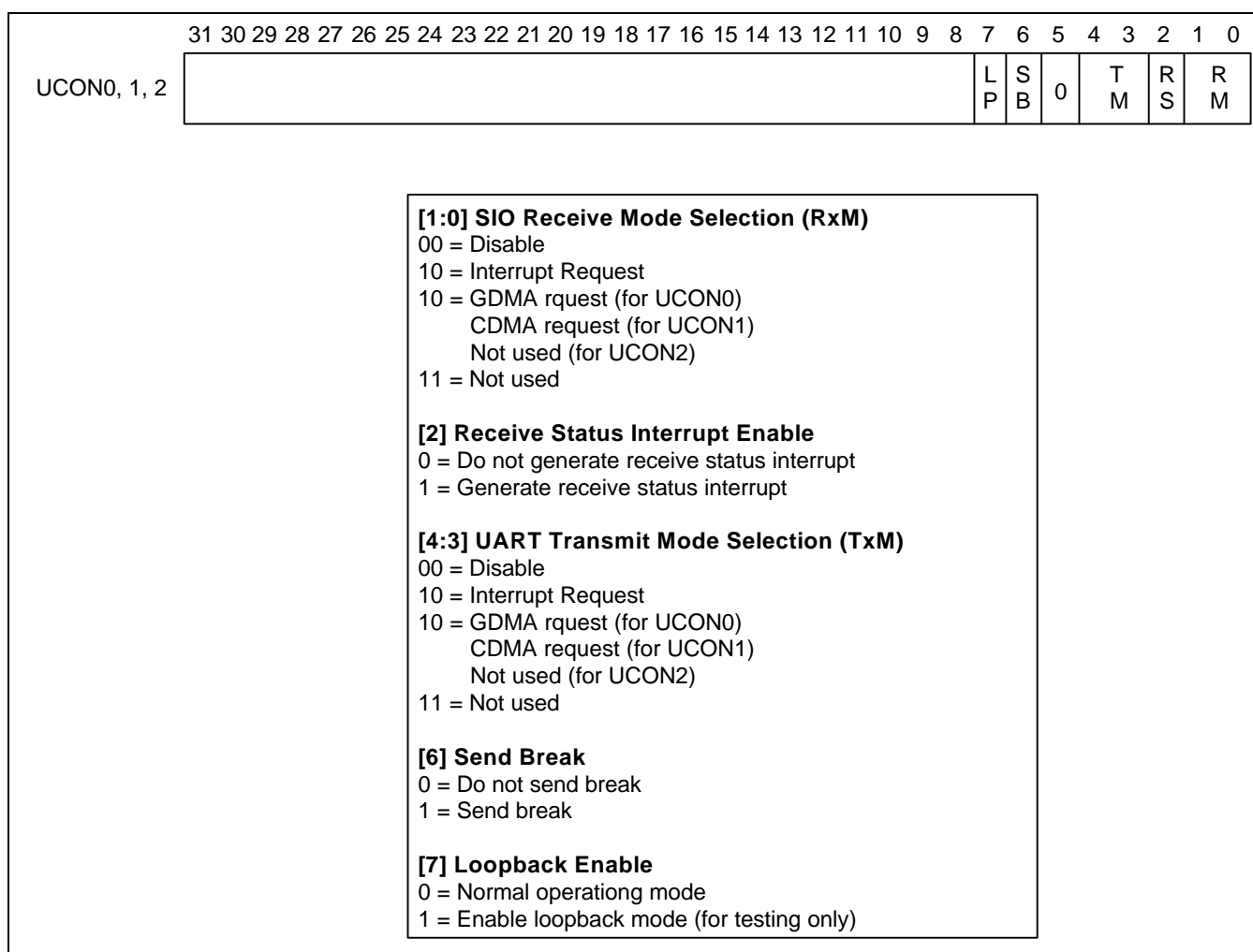


Figure 11-6. UART Control Register (UCON0,1,2)

UART Status Register

There are two identical UART status registers (USTAT0,1) in the UART block, for each SIO channel. The USTAT is a read-only register that is used to monitor the status of SIO.

Registers	Offset Address	R/W	Description	Reset Value
USTAT0	0xb008	R/W	UART ch-0 status register	0xc0
USTAT 1	0xb808	R/W	UART ch-1 status register	0xc0
USTAT 2	0xc008	R/W	UART ch-2 status register	0xc0

[0]	Overrun error	<p>This bit is automatically set to "1" whenever an overrun error occurs during a serial data receive operation. If the receive status interrupt enable bit UCONn[2] is "1", a receive status interrupt will be generated if an overrun error occurs.</p> <p>This bit is automatically cleared to "0" whenever the UART status register (USTATn) is read.</p>
-----	---------------	---

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

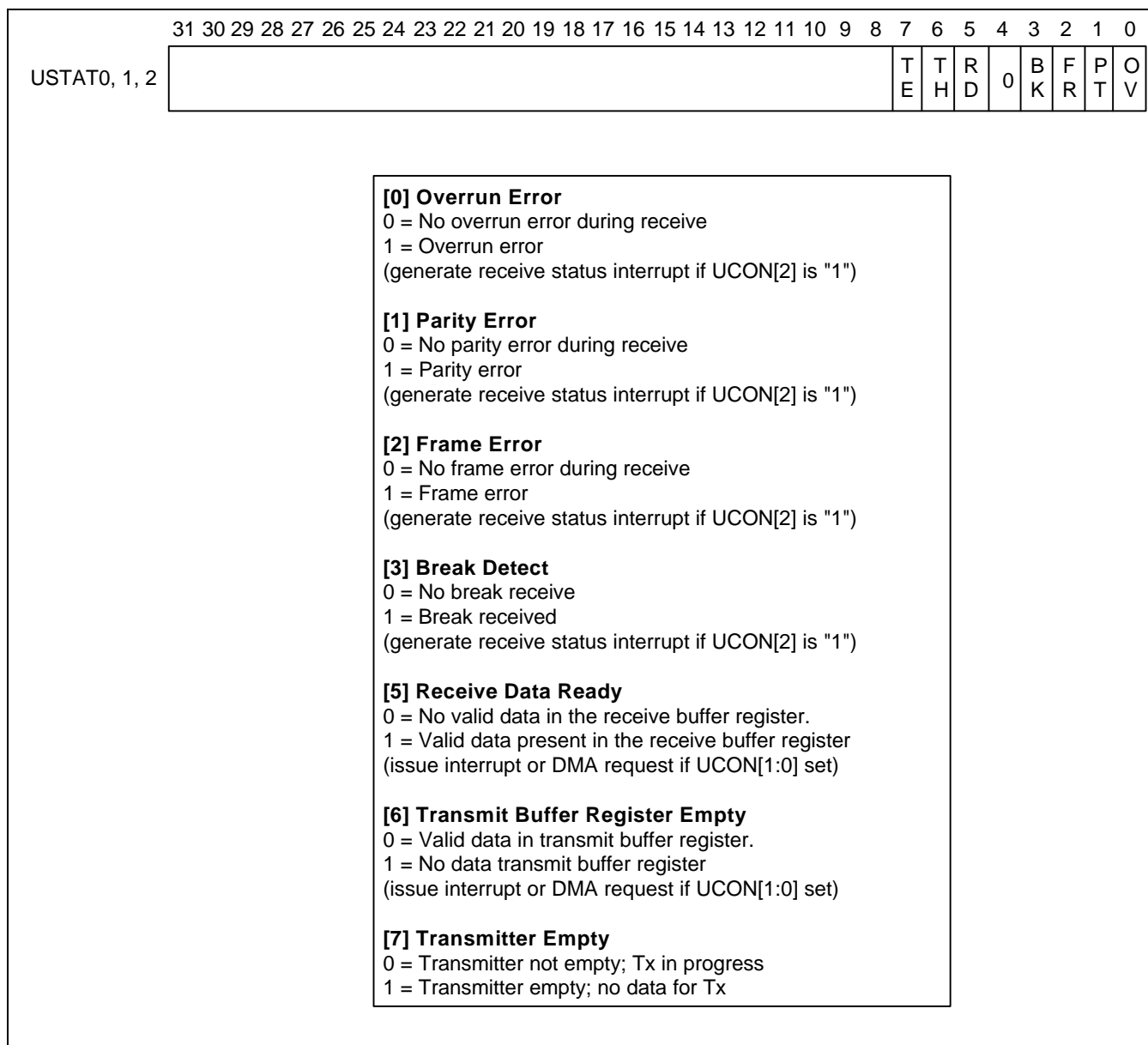


Figure 11-7. UART Status Register (USTAT0,1,2)

UART Transmit Buffer Register

There are two identical UART transmit buffer registers (TBR) in the UART block for the two SIO channels, each of which contains an 8-bit data value to be transmitted over the SIO channel.

In DMA-based transmit mode, the address of the transmit buffer register should be set into the DMA destination address register as the destination of the DMA channel.

Registers	Offset Address	R/W	Description	Reset Value
UTXBUF0	0xb00c	W	UART ch-0 transmit buffer register	0x00
UTXBUF1	0xb80c	W	UART ch-1 transmit buffer register	0x00
UTXBUF2	0xc00c	W	UART ch-2 transmit buffer register	0x00

[7:0] Transmit data

This field contains the data to be transmitted by the corresponding SIO channel. When this register is written, the transmit buffer register empty bit in the status register USTAT[6] should be "0" This prevents overwriting transmit data that may already be present in the TBR. Whenever the TBR is written with new value, the transmit register emptybit USTAT[6] is automatically cleared to "0".

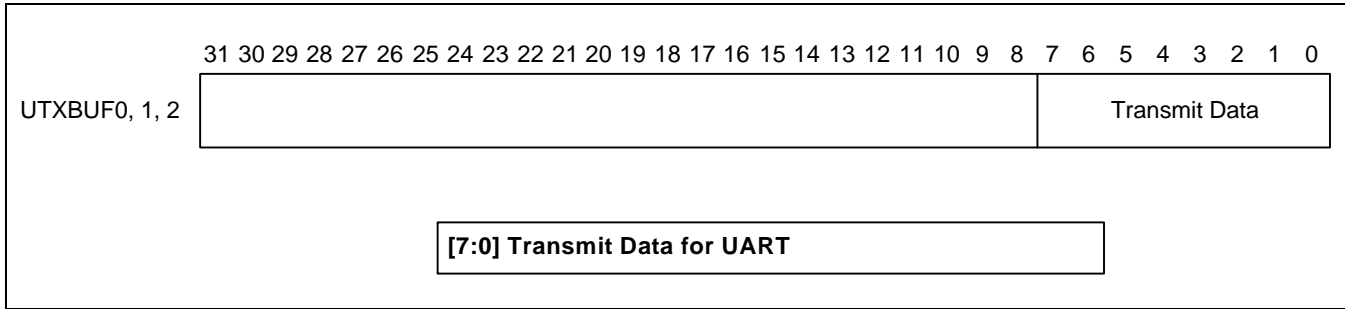


Figure 11-8. UART Transmit Buffer Register (UTXBUF0,1,2)

UART Receive Buffer Register

There are two identical UART receive buffer registers (RBR) in the UART block for the two SIO channels, each of which contains an 8-bit data value for received serial data.

In DMA-based receive mode, the address of the receive buffer register should be set into the DMA source address register as the source of the DMA channel.

Registers	Offset Address	R/W	Description	Reset Value
URXBUF0	0xb010	R	UART ch-0 receive buffer register	0x00
URXBUF1	0xb810	R	UART ch-1 receive buffer register	0x00
URXBUF2	0xc010	R	UART ch-2 receive buffer register	0x00

[7:0] Receive data

This field contains the data received from the corresponding SIO channel. When UART finishes receiving a data frame, the receive data ready bit in the UART status register USTAT[5] should be "1". This prevents reading invalid receive data that may already be present in the RBR. Whenever the RBR is read, the receive data ready bit USTAT[5] is automatically cleared to "0".

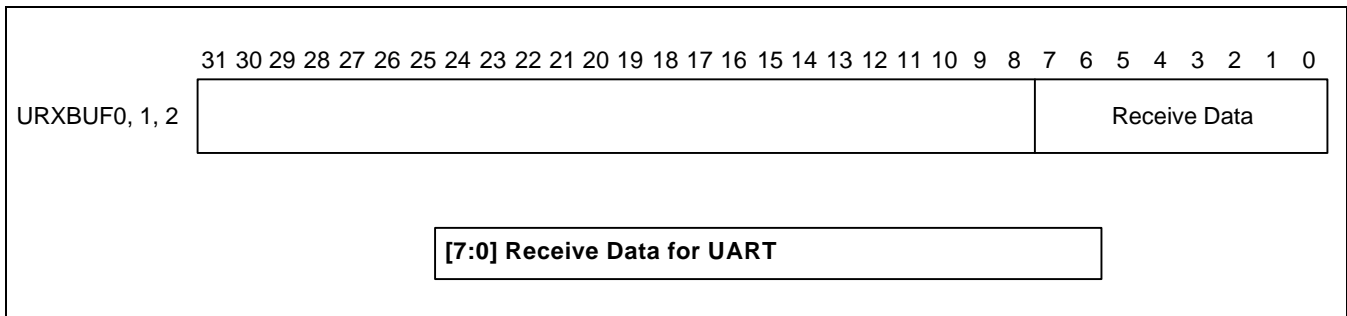


Figure 11-9. UART Receive Buffer Register (URXBUF0, 1, 2)

UART Baud Rate Divisor Registers

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

$$UBRDIVn = (\text{int})(\text{source_clock} / (\text{bps} \times 16)) - 1$$

where the source_clock is either MCLK (the internal master clock) or UCLK (the external UART clock input), as determined by the setting of the serial clock selection bit in the line control register, ULCON[6].

Registers	Offset Address	R/W	Description	Reset Value
UBRDIV0	0xb014	R/W	Baud rate divisor register 0	0x0000
UBRDIV1	0xb814	R/W	Baud rate divisor register 1	0x0000
UBRDIV2	0xc014	R/W	Baud rate divisor register 2	0x0000

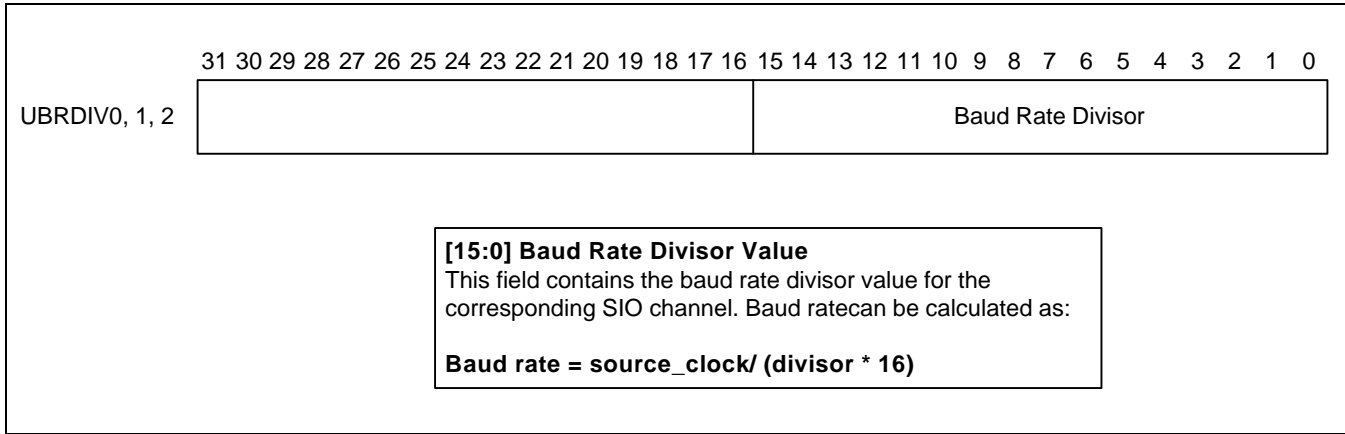


Figure 11-10. UART Baud Rate Divisor Register (UBRDIV0,1, 2)

NOTE: THE BAUD RATE DIVISOR SHOULD BE A VALUE FROM 1 TO (2^16-1).

TIMING DIAGRAMS

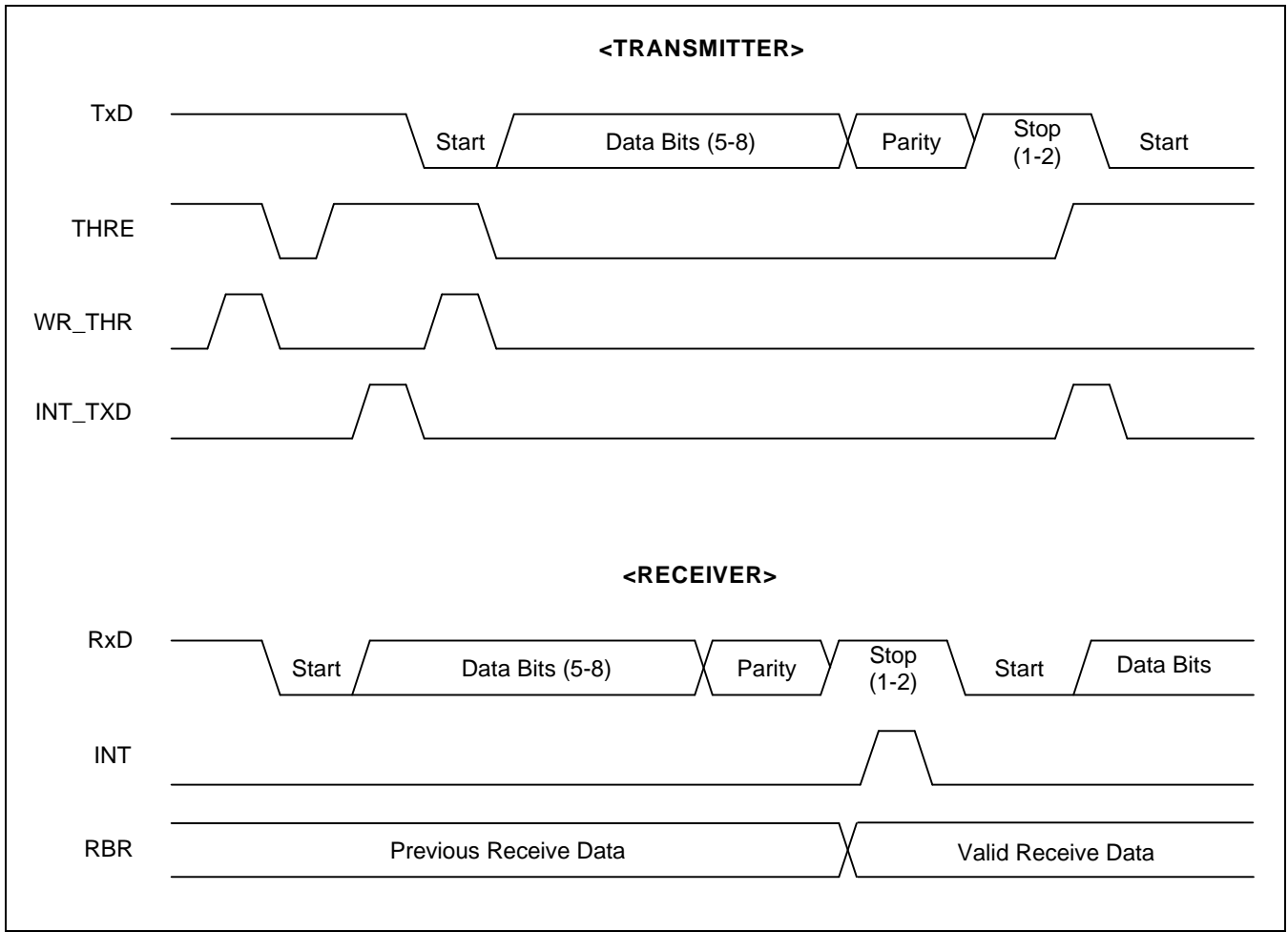


Figure 11-11. Interrupt-Based Serial I/O Timing Diagram (Tx and Rx)

12

TONE GENERATOR

OVERVIEW

The KS32C65100 Tone Generator provides a programmable tone signal which has 50% duty cycle and can be used to make a 'key-click' sound. The tone Generator block has a tone counter which includes 8_bit programmable divider and a 1/2 divider for making the 50% duty cycle, and a Tone Data register (TONDATA) which has the tone enable or disable bit and tone count data bits. The 8 bit programmable divider receives $MCLK/(prescaler+1)/128$ clock signals and divides it depending on the count value in TONDATA[7:0] bits. Also, you can set the prescaler value (Initial value: 0xC) in TSTCON (Figure 14-9).

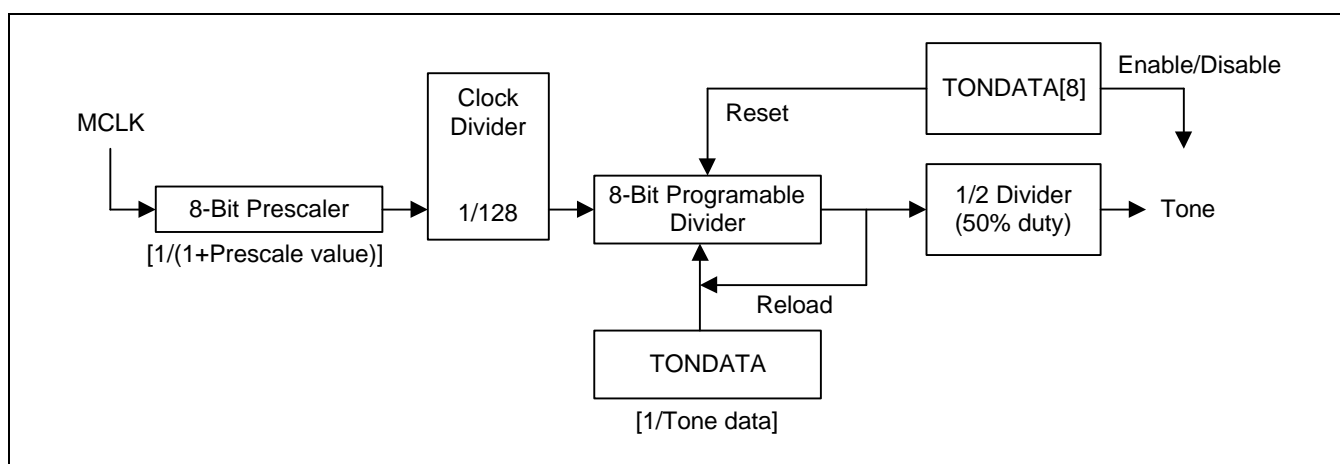


Figure 12-1. Tone Generator Block Diagram

The TONDATA[8] bit enables or disables the Tone generator operation. When it is cleared to '0', the tone output is disabled (stopped) and the programmable divider is automatically cleared while the tone data register (TONDATA) retains its contents. The initial value of the tone enable bit is '0'.

The input clock to the tone generator is $MCLK/(prescaler+1)/128$. The division ratio of the tone counter is determined by the tone data register value, and ranges from 0 to 255.

A user has to load data into the tone data register (TONDATA) before enabling the tone generator to get the correct tone signal. To make out the 50% duty cycle tone signal, KS32C65100 tone generator has a 1/2 divider with a programmable divider. The output of the programmable divider is divided by the 1/2 divider.

The frequency of the tone is calculated as follows:

$$\frac{MCLK}{[(Prescaler+1) * 128 * ToneData * 2]}$$

Table 12-1. Tone Generator Data Value Setting (MCLK = 33 MHz)

[@ Prescale = 0xC]

TONDATA	Tone Freq.	TONDATA	Tone Freq.
0	No tone (all high)	4	2.479 kHz
1	9.915 kHz
2	4.958 kHz	100	99.159 Hz
3	3.305 kHz	255	38.886 Hz

TONE GENERATOR DATA REGISTER (TONDATA)

The tone generator data register (TONDATA) stores an 8-bit value which determines the frequency of the tone generator output. The value in the TONDATA register determines the division ratio of the programmable divider. The divided-by value, therefore, ranges from 0 to 255. Then the output value of the tone counter is divided by two, producing a 50% duty tone output signal. A reset clears the TONDATA value to '00h'. The tone frequency is therefore calculated, based on the tone data value, as follows.

$$\frac{\text{MCLK}}{[(\text{Prescaler} + 1) * 128 * \text{ToneData} * 2]}$$

Register	Offset Address	R/W	Description	Reset Value
TONDATA	0x3804	R/W	Tone generator data register	0x0ff

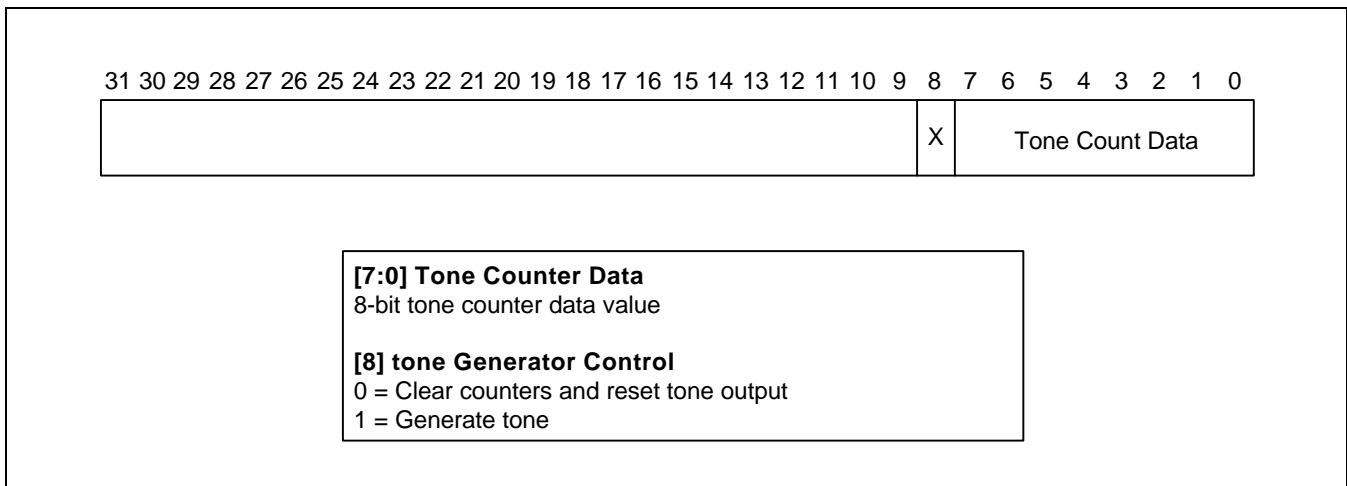


Figure 12-2. Tone Data Register (TONDATA)

13

WATCHDOG TIMER

OVERVIEW

The KS32C65100 Watchdog Timer is used to resume controller operation when it is disturbed due to noise or other kinds of system errors or malfunctions. It can be used as a normal interval timer to request interrupt services. also, you can set the prescaler value (initial value: 0xC) in TCR (Figure 13-3).

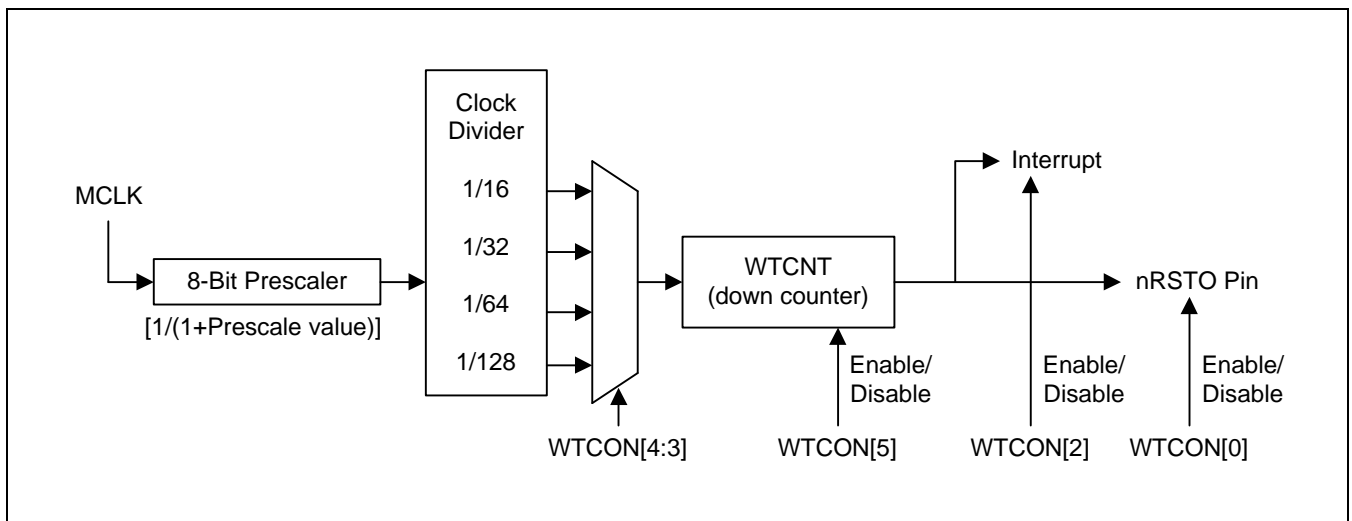


Figure 13-1. Watchdog Timer

WATCHDOG TIMER COUNTER REGISTERS

The watchdog timer counter register WTCNT is used to specify the time out duration.

The watchdog timer enable bit (bit5, WTCON) must be '0' before loading a value to this register.

Watchdog Timer_clock = $MCLK / (prescale_value + 1) / division_factor$

Watchdog Timer_duration = $count_val. / Watchdog\ Timer_clock$

Table 13-1. Watchdog Timer Counter Setting (MCLK = 33MHz)

[@ Prescale = 0xC, WTCNT = 16_bit count]

Clock Source	Resolution	Maximum Interval	Remark
MCLK/(prescale+1)/16	6.24us	408.9ms	Default setting
MCLK/(prescale+1)/32	12.28us	817.9ms	-
MCLK/(prescale+1)/64	24.96us	1.636s	-
MCLK/(prescale+1)/128	49.92us	3.272s	-

Register	Offset Address	R/W	Description	Reset Value
WTCNT	0x4004	R/W	Watchdog timer count register	0x0003

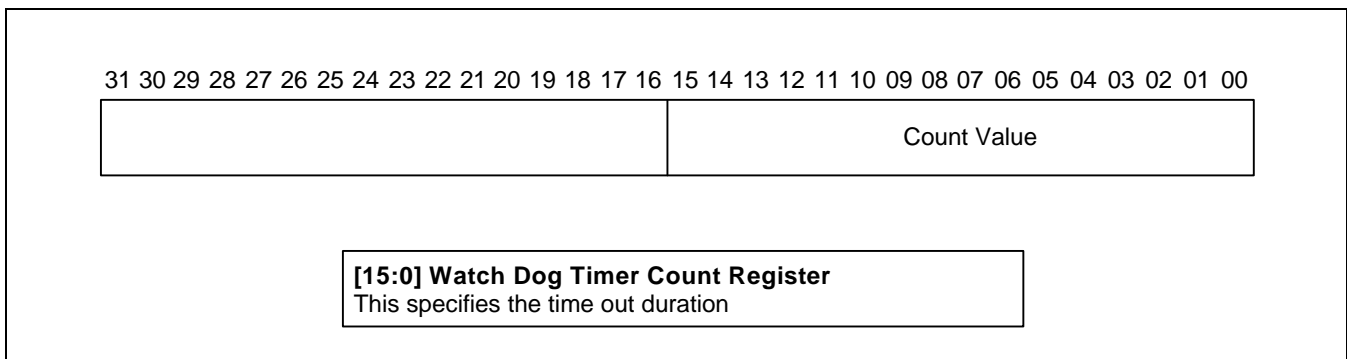


Figure 13-2. Watchdog Timer Count Register (WTCNT)

WATCHDOG TIMER CONTROL REGISTER

The Watchdog Timer Control register WTCN provides the control bits for the enable/disable of the watchdog timer, selects the clock signal from 4 different sources, enables/disables interrupts, and enables/disables the watchdog timer reset output signal nWDTO pin. If the watchdog timer is set to 0, WTCN is cleared to 0x0.

Register	Offset Address	R/W	Description	Reset Value
WTCON	0x4000	R/W	Watchdog timer control register	0x21

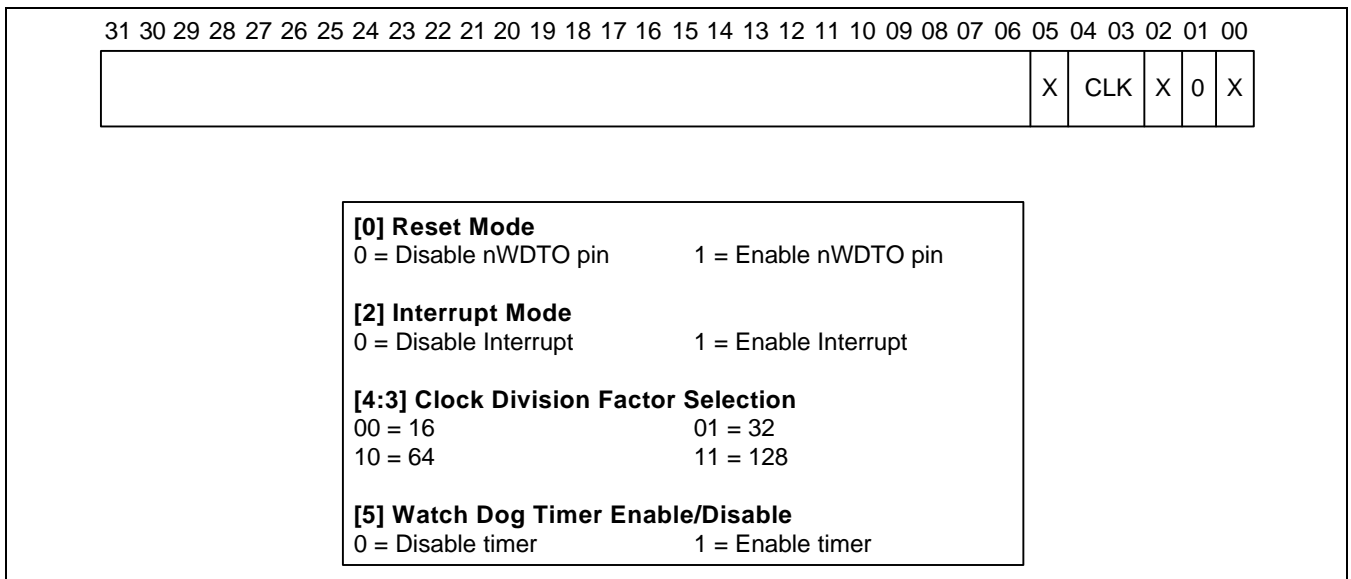


Figure 13-3. Watchdog Timer Control Register (WTCON)

WATCHDOG TIMER OPERATION

Before loading or reading a count value of the Watchdog Timer Count Register (WTCNT), users have to disable the watchdog timer by setting the WTCON[5] bit to zero. When WTCON[5] bit is set to "1", the watchdog timer is enabled and the counter starts down-count. The watchdog counter register value is accessible at any time while the watchdog timer is enabled, because it provides read and write features.

The watchdog timer provides general timer interrupt as well as system reset features. To enable a timer interrupt, WTCN[2] bit has to be set to 1. When a timer interrupt is enabled, the interrupt signal generates one pulse of a request signal to CPU to compare it to the long watchdog reset signal. The interrupt pending bit (bit2, INTPNDR) is automatically set to '1' when an underflow occurs.

When WTCN[0] bit is '1', the nWDTO pin is enabled and the watchdog reset signal comes through the nWDTO pin. If the watchdog counter reaches to zero, the nWDTO signal is activated during for 128 MCLK cycles for some reason, and WTCN will be 0x0. To avoid the watchdog timer activating the nWDTO signal, MPU has to periodically reload the counter value into the watchdog counter register (WTCNT).

The nWDTO signal is not connected to nRESET internally. If nWDTO is connected to nRESET by an external logic, the KS32C65100 initialization routine will be executed by the nWDTO signal.

14 I/O PORTS

OVERVIEW

The KS32C65100 has 18 input, 43 output, and 27 input/output ports.

I/O PORT SPECIAL REGISTERS

Two registers control the I/O port configuration: IOPMOD and IOP.

Table 14-1 shows the possible values for the port mode registers. The IOP register contains one bit for each port which reflects the signal level at the respective port pin.

NOTE: I/O port muxed pin configuration

Table 14-1. I/O Port Mode Configuration and Settings

I/O Port Pin	I/O Port Mode Configuration Settings	
	Function for 1	Function for 0
GIP[0]: RXD0	GIP[0]	RXD0
GIP[1]: RXD1	GIP[1]	RXD1
GIP[2]: RXD2	GIP[2]	RXD2
GIP[3]: nEXT_INT0	GIP[3]	nEXT_INT0
GIP[4]: nEXT_INT1	GIP[4]	nEXT_INT1
GIP[5]: nEXT_INT2	GIP[5]	nEXT_INT2
GIP[6]: nEXT_DREQ	GIP[6]	nEXT_DREQ
GIP[7]: nWAIT	GIP[7]	nWAIT
GIP[8]: ECD_IN1	GIP[8]	ECD_IN1
GIP[9]: ECD_IN2	GIP[9]	ECD_IN2
GIP[10]: nHSYNC1	GIP[10]	nHSYNC1
GIP[11]: nLREADY	GIP[11]	nLREADY
GIP[12]: nHSYNC2	GIP[12]	nHSYNC2
GIP[13]: nVDI	GIP[13]	nVDI
GIP[14]: nVCLK	GIP[14]	nVCLK
GIP[15]: nINIT	GIP[15]	nINIT
GIP[16]: nSLCTIN	GIP[16]	nSLCTIN
GIP[17]: nAUTOFD	GIP[17]	nAUTOFD

Table14-1. I/O Port Mode Configuration and Settings (Continued)

I/O Port Pin	I/O Port Mode Configuration Settings	
	Function for 1	Function for 0
GOPA[0]: TXD0	GOPA[0]	TXD0
GOPA[1]: TXD1	GOPA[1]	TXD1
GOPA[2]: TXD2	GOPA[2]	TXD2
GOPA[3]: TONE	GOPA[3]	TONE
GOPA[4]: nWDTO	GOPA[4]	nWDTO
GOPA[5]: nEXT_DACK	GOPA[5]	nEXT_DACK
GOPA[6]: CLKOUT	GOPA[6]	CLKOUT
GOPA[7]: nRCS1	GOPA[7]	nRCS1
GOPA[8]: nECS2	GOPA[8]	nECS2
GOPA[9]: nIORD	GOPA[9]	nIORD
GOPA[10]: nIOWR	GOPA[10]	nIOWR
GOPA[13:11]: PWMO[2:0]	GOPA[13:11]	PWMO[2:0]
GOPA[14]: nVDO1	GOPA[14]	nVDO1
GOPA[15]: LSU_CLK	GOPA[15]	LSU_CLK
GOPA[18:16]: SLED[2:0]	GOPA[18:16]	SLED[18:16]
GOPA[20:19]: SNM_CON[1:0]	GOPA[20:19]	SNM_CON[1:0] (SM_PHB, SM_PHA)
GOPA[22:21]: LFM_CON[1:0]	GOPA[22:21]	LFM_CON[1:0] (LF_PHB, LF_PHA)
GOPA[24:23]: CR_PH[1:0]	GOPA[24:23]	CR_PH[1:0] (CR_PHB, CR_PHA)
GOPA[28:25]: CR_CUR[3:0]	GOPA[28:25]	CR_CUR[3:0] (IB1, IB0, IA1, IA0)
GOPA[29]: nVDO2	GOPA[29]	nVDO2
GOPB[12:0]: PHGA[12:0]	GOPB[12:0]	PHGA[13:01]
GIOP[26:11]: PHOE[16:1]	GIOP[26:11]	PHOE[16:01]

I/O PORT MODE REGISTER

The I/O port mode register GIOPMOD is used to configure the GIOP (general in/out port).

Register	Offset Address	R/W	Description	Reset Val.
GIOPMOD	0x2800	R/W	Bi-directional port mode register	0xffff800

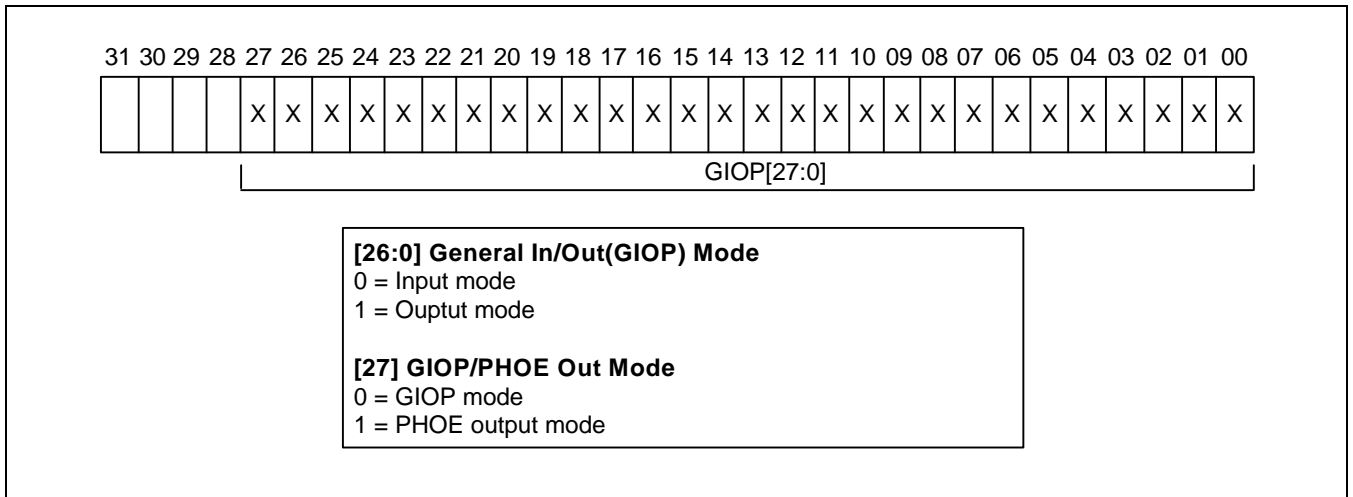


Figure 14-1. Bi-directional Port Mode Register (GIOPMOD)

INPUT PORT MODE REGISTER

Register	Offset Address	R/W	Description	Reset Val.
GIPMOD	0x2804	R/W	Input port mode register	0x00000

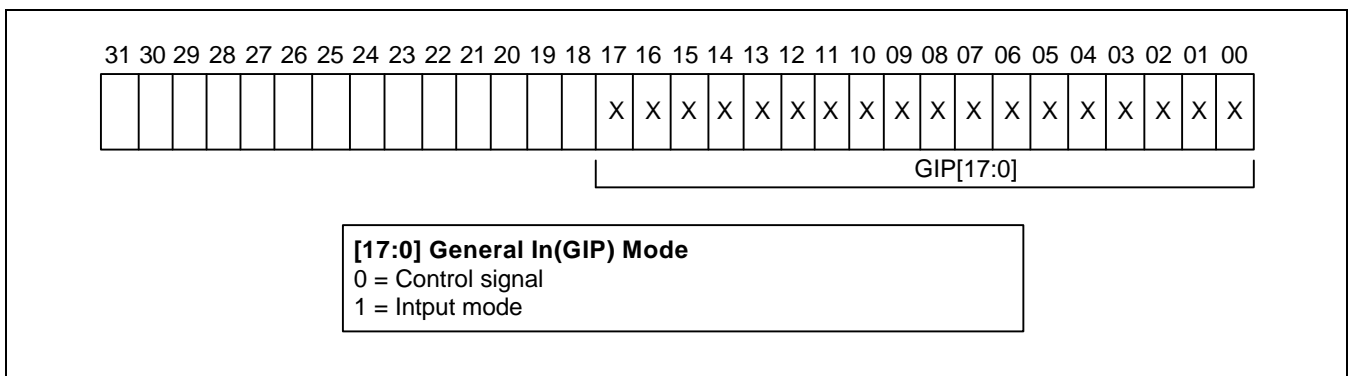


Figure 14-2. Input Port Mode Register (GIPMOD)

OUTPUT A PORT MODE REGISTER

Register	Offset Address	R/W	Description	Reset Value
GOPAMOD	0x2808	R/W	Output port mode register	0x00000000

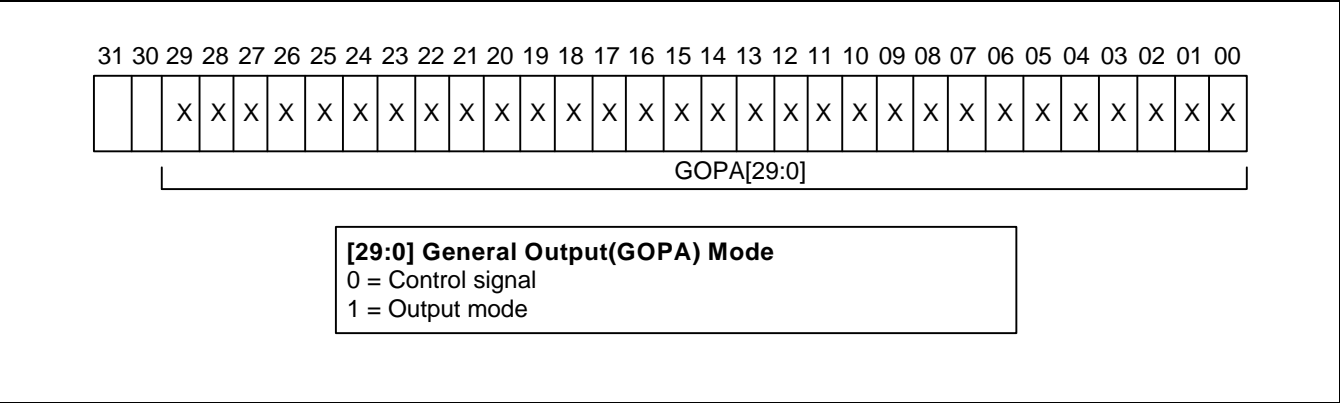


Figure 14-3. Output Port Mode Register (GOPAMOD)

OUTPUT B PORT MODE REGISTER

Register	Offset Address	R/W	Description	Reset Value
GOPBMOD	0x280C	R/W	Output port mode register	0x0000

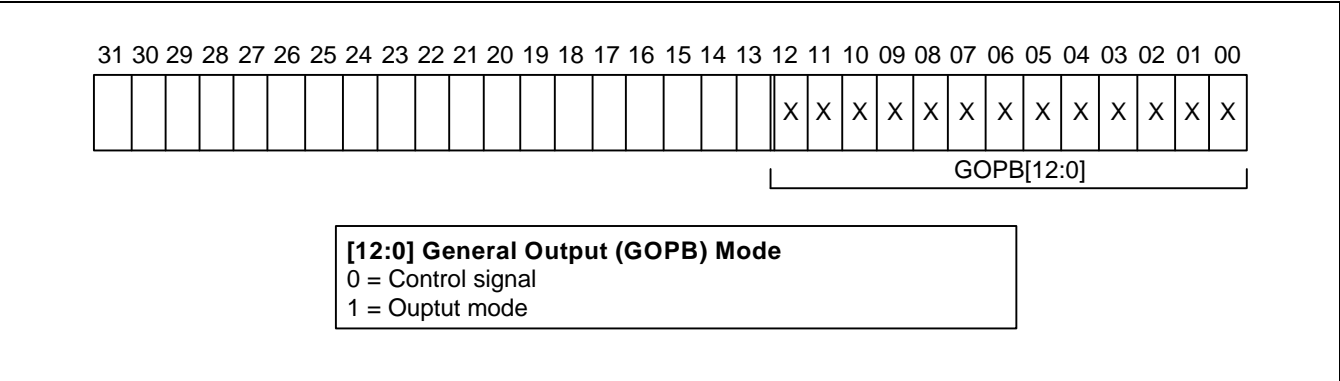


Figure 14-4. Output Port Mode Register (GOPBMOD)

I/O PORT DATA REGISTER

The I/O port data register, GIOPD, contains one-bit values for I/O ports that are configured to input mode and one-bit write value for ports that are in output mode.

Register	Offset Address	R/W	Description	Reset Value
GIOPD	0x2810	R/W	Bi-directional port data register	0x0000000

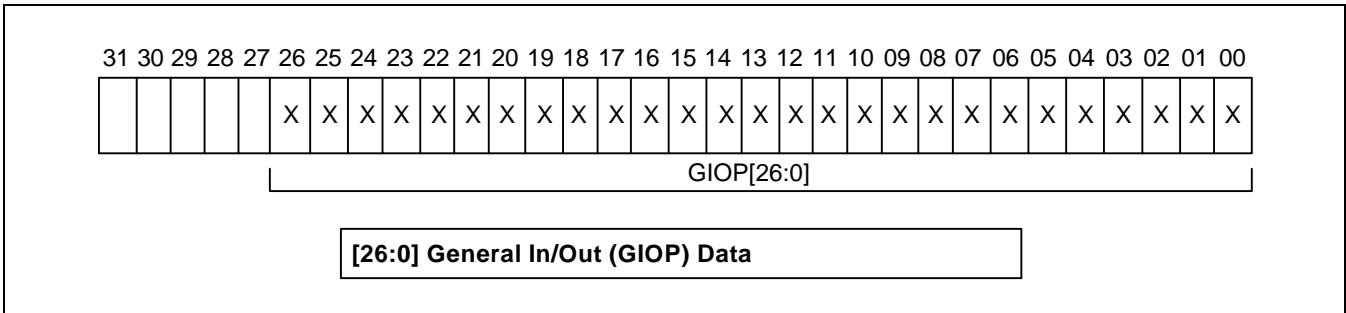


Figure 14-5. Bi-directional Port Data Register (GIOPD)

INPUT PORT DATA REGISTER

Register	Offset Address	R/W	Description	Reset Value
GIPD	0x2814	R/W	Input port data register	0xXXXXXX

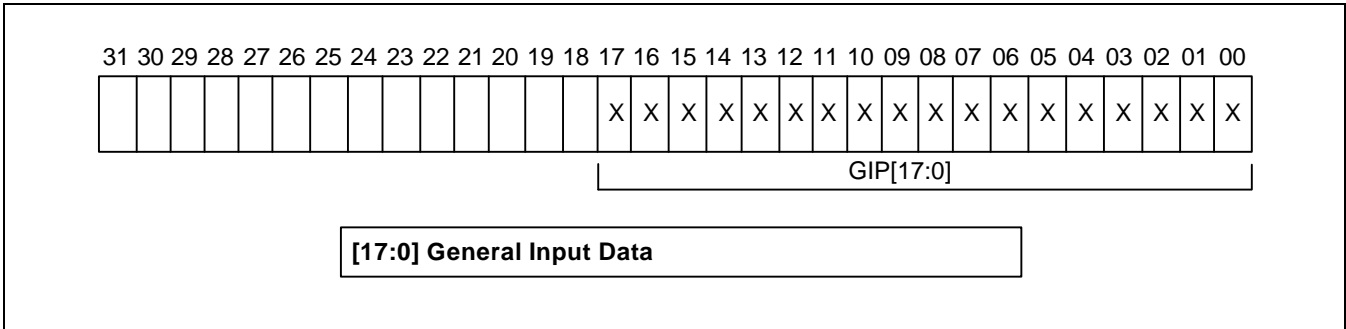


Figure 14-6. Input Port Data Register (GIPD)

OUTPUT A PORT DATA REGISTER

Register	Offset Address	R/W	Description	Reset Value
GOPAD	0x2818	R/W	Output port data register	0x00000000

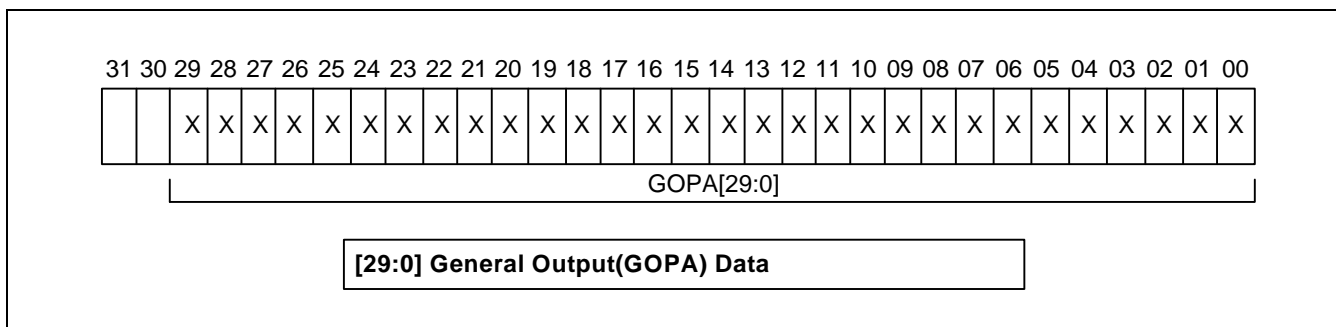


Figure 14-7. Output Port A Data Register (GOPAD)

OUTPUT B PORT DATA REGISTER

Register	Offset Address	R/W	Description	Reset Value
GOPBD	0x281C	R/W	Output port data register	0x0000

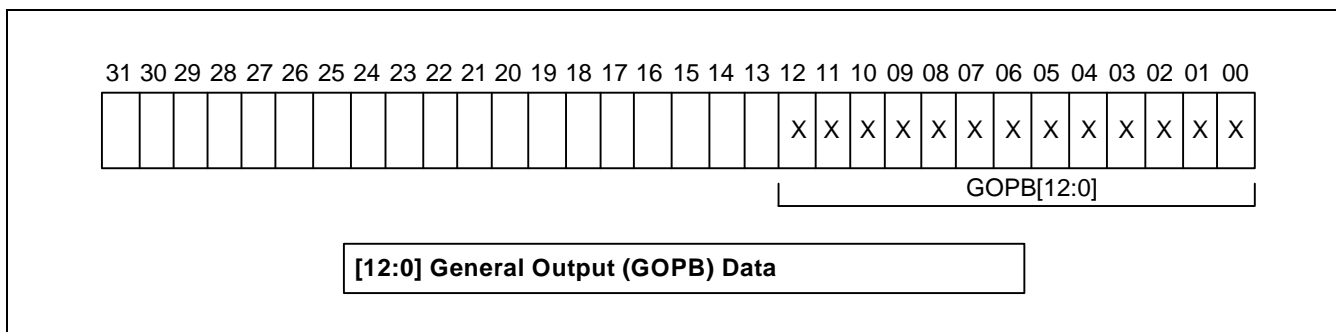


Figure 14-8. Output Port B Data Register (GOPBD)

TEST CONTROL REGISTER

The test control register (TSTCON) contains the 16 bits for testing the functions of CHORUS. These bits for testing are only used during fabrication. These bits are not specified in this manual. The other bits which you can use are as follows:

- CKOUT mode
 - The CKOUT mode bit determines whether CKOUT output is divided by 2 or not.
 - 0 = MCLK
 - 1 = MCLK / 2
- Prescaler value
 - Timer0, Timer1, Timer2, watchdog timer, tone generator, and line feed motor timer use this prescaler value to divide MCLK.
- Bidirectional control pin

Register	Offset Address	R/W	Description	Reset Value
TSTCON	0x2820	R/W	Test control register	0x00600

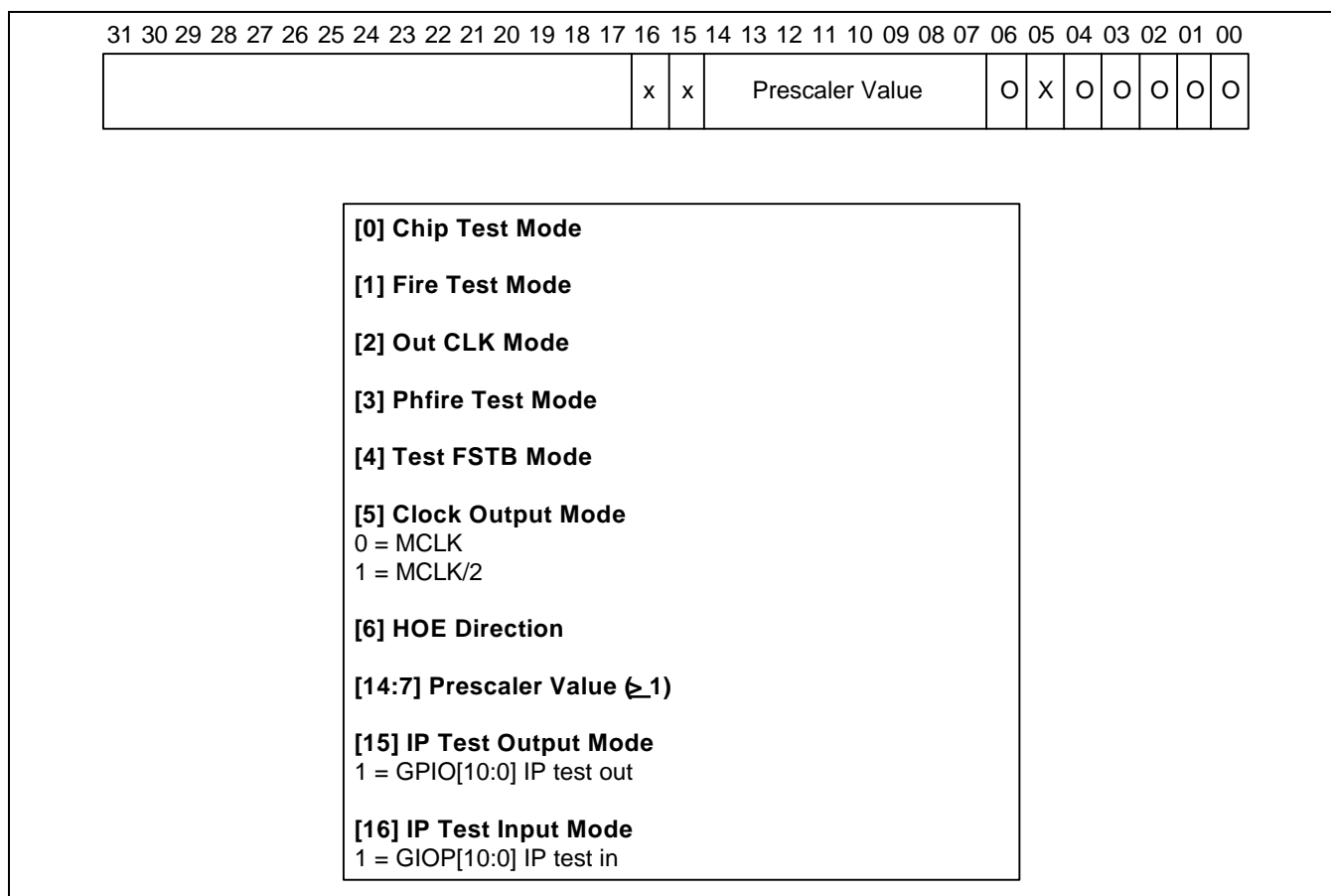


Figure 14-9. Test Control Register (TSTCON)

EXTERNAL INTERRUPT CONTROL REGISTER

The external interrupt control register (INTCON) is used for external interrupt signal filter mode control.

Register	Offset Address	R/W	Description	Reset Value
INTCON	0x2824	R/W	External interrupt control register	0x000

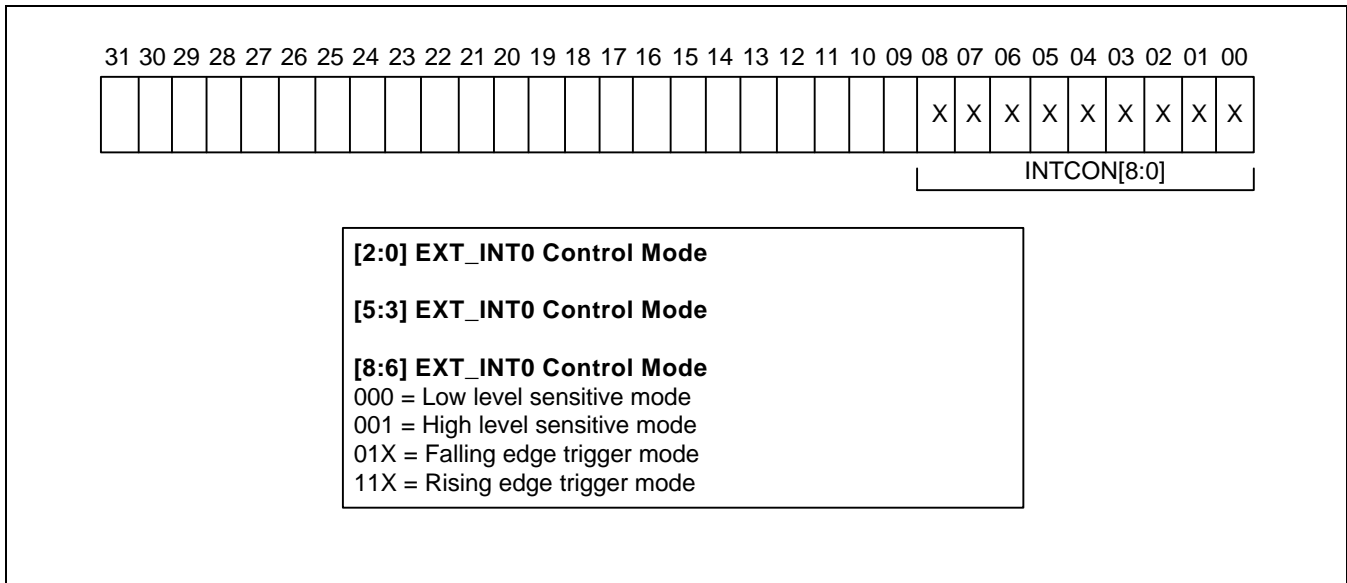


Figure 14-10. External Interrupt Control Register (INTCON)

TEST PIN SETTING

Test2	Test1	Test0	Test Mode	Internal Clock
0	0	0	Normal mode0 (LF, Scan Mtr output PHASE[3:0]: Initial value ⇒ "H")	OSC_CLK
0	0	1	Normal mode1 (LF, Scan Mtr output PHASE[3:0]: Initial value ⇒ "L")	
0	1	0	IP scan test mode	
0	1	1	IP ADC test mode	
1	0	0	Normal mode0 (LF, Scan Mtr Output PHASE[3:0]: Initial value ⇒ "H")	OSC_CLK/2
1	0	1	Normal mode1 (LF, Scan Mtr output PHASE[3:0]: Initial value ⇒ "L")	
1	1	0	Core test mode	OSC_CLK
1	1	1	NAND tree test mode	

15

INTERRUPT CONTROLLER

OVERVIEW

The KS32C65100 interrupt structure has a total of 30 interrupt sources, which can be individually or globally enabled or disabled. Interrupt requests can be generated by internal function blocks and at external pins. The ARM7TDMI core recognizes two kinds of interrupts: a normal interrupt request (IRQ) and a fast interrupt request (FIQ). Therefore, all KS32C65100 interrupts can be categorized as either IRQ or FIQ. The KS32C65100 interrupt controller extends the number of multiple interrupt sources that can be serviced by using three special registers, INTMOD, INTPND, and INTMSK:

- **Interrupt mode register.**
Defines the interrupt mode, IRQ or FIQ, for each interrupt source.
- **Interrupt pending register.**
Indicates that an interrupt requests is pending (that is, when the I-flag or F-flag is set in the program status register, PSR). This status prevents any additional interrupts from being acknowledged. When the pending bit is set, the interrupt service routine starts whenever the I-flag or F-flag is cleared to '0'. The service routine must clear the pending condition by writing '1' to the corresponding pending bit.
- **Interrupt mask register.**
Indicates that the current interrupt has been disabled if the corresponding mask bit is '0'. If an interrupt mask bit is '1', the interrupt will be serviced normally. And if a global mask bit (bit 31) is cleared, all interrupts are not serviced. However, the source's pending bit is set when the interrupt is generated, even if the corresponding mask bit is '0'. After the global mask bit is set, the interrupt will be serviced.

INTERRUPT SOURCES

The 30 interrupt sources in the KS32C65100 interrupt structure are described, in brief, in Table 15.1.

Table 15-1. Interrupt Sources

No.	Source Name	Description
0	INT_EXT2	External interrupt 2 (comes from general input port 5)
1	INT_EXT1	External interrupt 1 (comes from general input port 4)
2	INT_EXT0	External interrupt 0 (comes from general input port 3)
3	INT_WATCHDOG	Watch dog timer interrupt
4	INT_TXD2	UART2 transmit interrupt
5	INT_TXD1	UART1 transmit interrupt
6	INT_TXD0	UART0 transmit interrupt
7	INT_RXD2	UART2 receive interrupt
8	INT_RXD1	UART1 receive interrupt
9	INT_RXD0	UART0 receive interrupt
10	INT_ERR2	UART2 error interrupt
11	INT_ERR1	UART1 error interrupt
12	INT_ERR0	UART0 error interrupt
13	INT_DMA1	GDMA transfer finish interrupt
14	INT_DMA0	CDMA transfer finish interrupt
15	INT_TIMER2	Timer2 interrupt
16	INT_TIMER1	Timer1 interrupt
17	INT_TIMER0	Timer0 interrupt
18	INT_PPIC	Parallel port interface controller interrupt
19	INT_IP1	Image processor interrupt 1 (Motor interrupt)
20	INT_IP0	Image processor interrupt 0 (SI interrupt)
21	INT_POS	Carrier position interrupt
22	INT_LFMTR	Line feed step interrupt
23	INT_CRST	Carrier step interrupt
24	INT_PRINT	Print interrupt
25	INT_HDMA	Head DMA interrupt
26	Reserved	Not Used
27	INT_EOP	PIFC end of page interrupt
28	INT_SOD	PIFC Start of DMA interrupt
29	INT_PUR	PIFC page under-run interrupt
30	INT_SYNC1	PIFC Psync request interrupt

SPECIAL REGISTER

Interrupt Mode Register

Bits in the interrupt mode register INTMOD specify if an interrupt is to be serviced as a fast or normal interrupt.

Register	Offset Address	R/W	Description	Reset Value
INTMOD	0x2000	R/W	Interrupt mode register	0x00000000

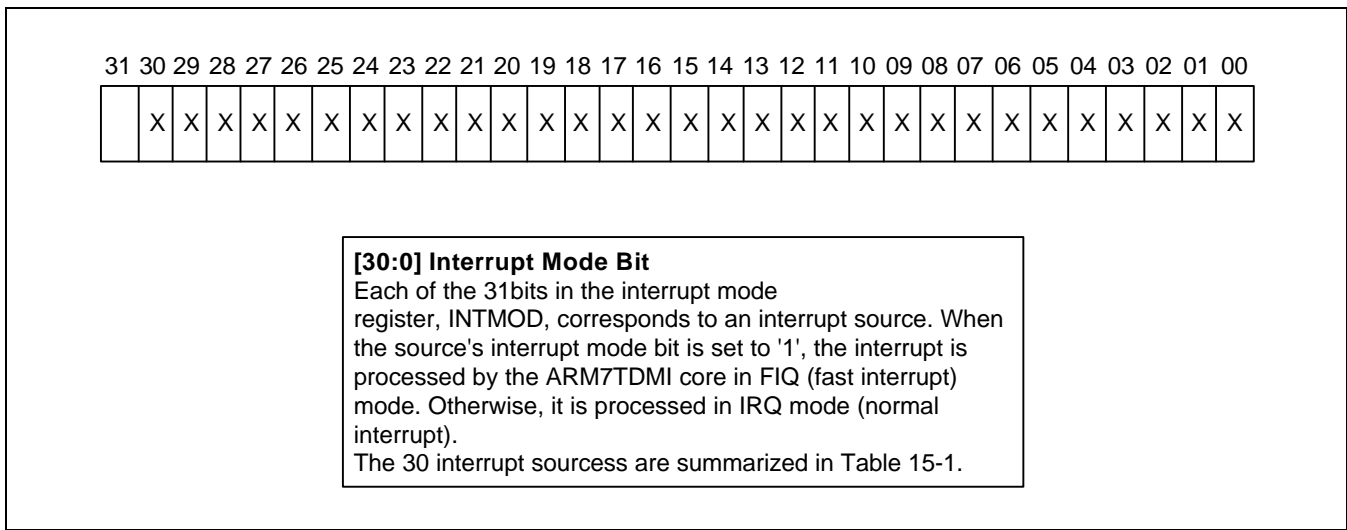


Figure 15-1. Interrupt Mode Register

Interrupt Pending Register

The interrupt pending register INTPND contains interrupt pending bits for each interrupt source. The INTPND has nothing to do with INTMSK. Although INTMSK forbids an Interrupt request generated, INTPND operates properly, independent of INTMSK.

Register	Offset Address	R/W	Description	Reset Value
INTPND	0x2004	R/W	Interrupt pending register	0x00000000

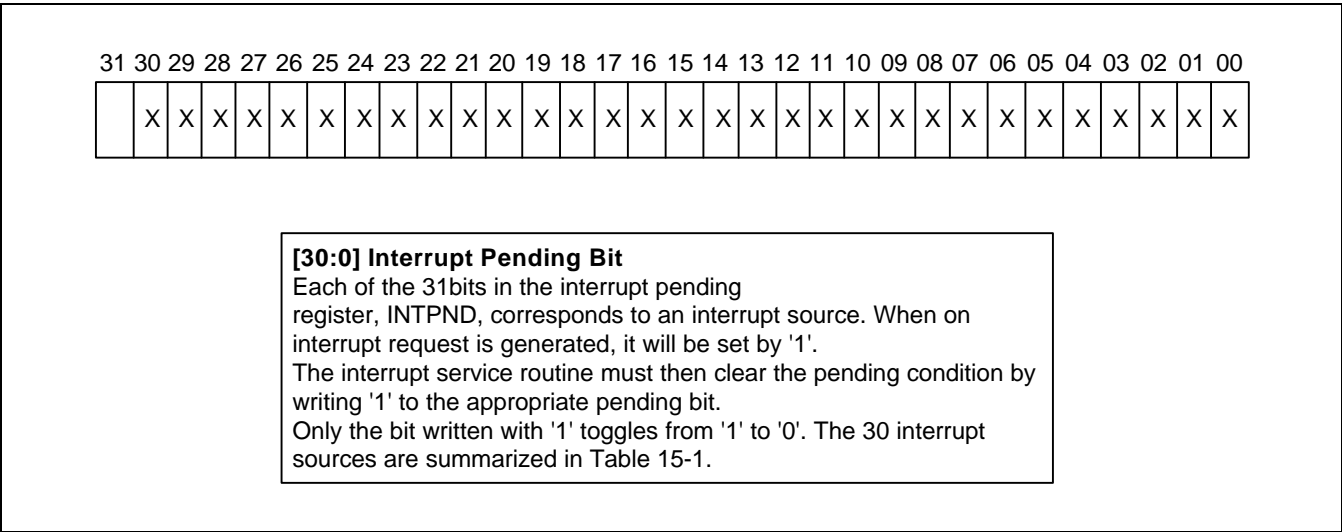


Figure 15-2. Interrupt Pending Register

Interrupt Mask Register

Register	Offset Address	R/W	Description	Reset Value
INTMSK	0x2008	R/W	Interrupt mask register	0x00000000

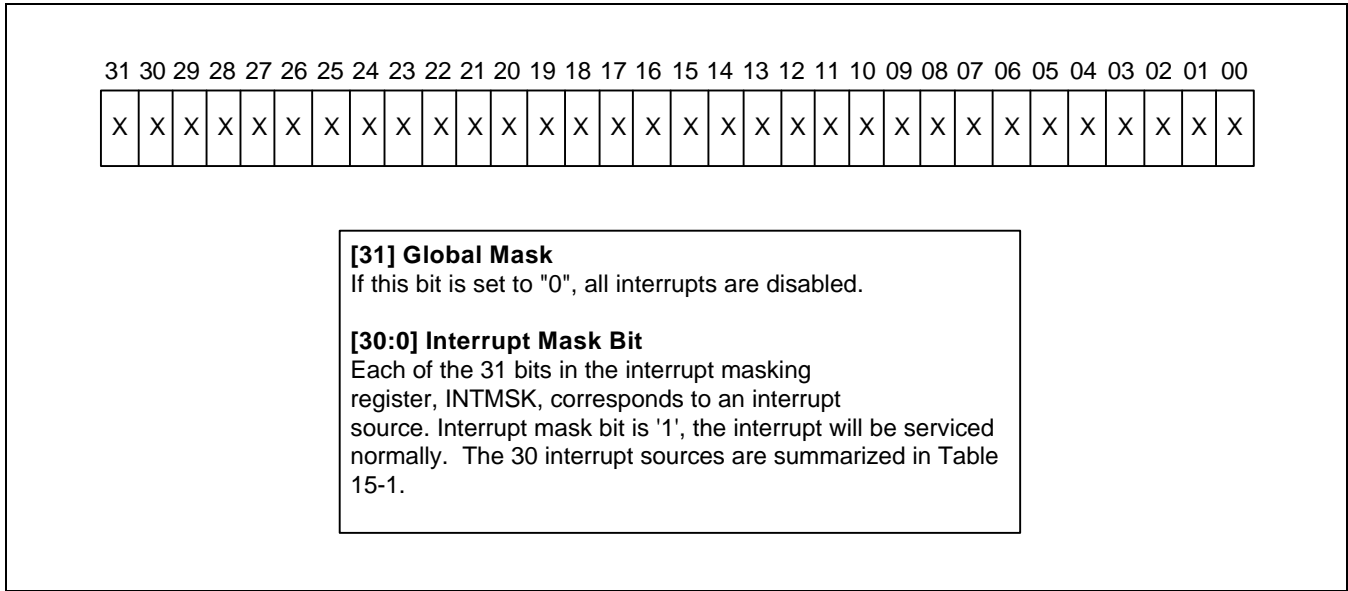


Figure 15-3. Interrupt Mask Register

16

LF MOTOR

OVERVIEW

This module performs the following functions:

- Step interrupt generation for driving line feed motor.
- 14-bit timer for step interrupt which controls the change of drive signal for line feed motor using selectable clock.
- Phase can be written by software or hardware.

NOTE

1. When the timer is enabled, it begins to decrease from the base value.
2. When the timer expires, the associated interrupt is generated, the base value is reloaded and the timer continues to decrease.
3. If a new value is loaded in this register before the timer is expired, the timer will keep counting with the new value.

SPECIAL FUNCTION REGISTER

LINE FEED MOTOR CONTROL REGISTER

This register selects the phase written by software or hardware, direction, interrupt request generation, and clock select.

Register	Offset Address	R/W	Description	Reset Value
LFCR	0x5800	R/W	Line feed motor control register	0x0800

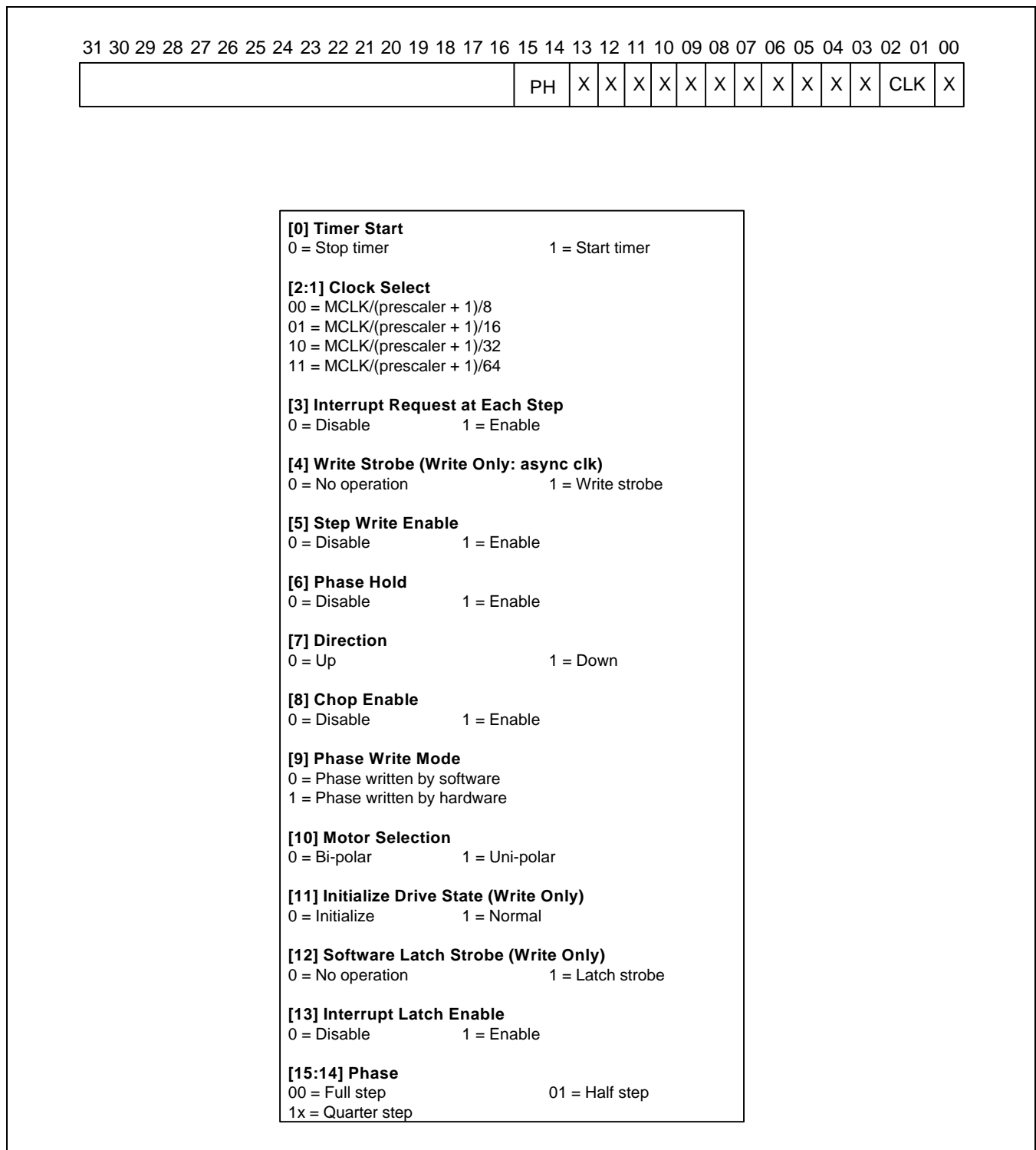


Figure 16-1. LF Motor Control Register

LINE FEED MOTOR PHASE CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
LFPCR	0x5804	R/W	Line feed motor phase control register	0x3c0

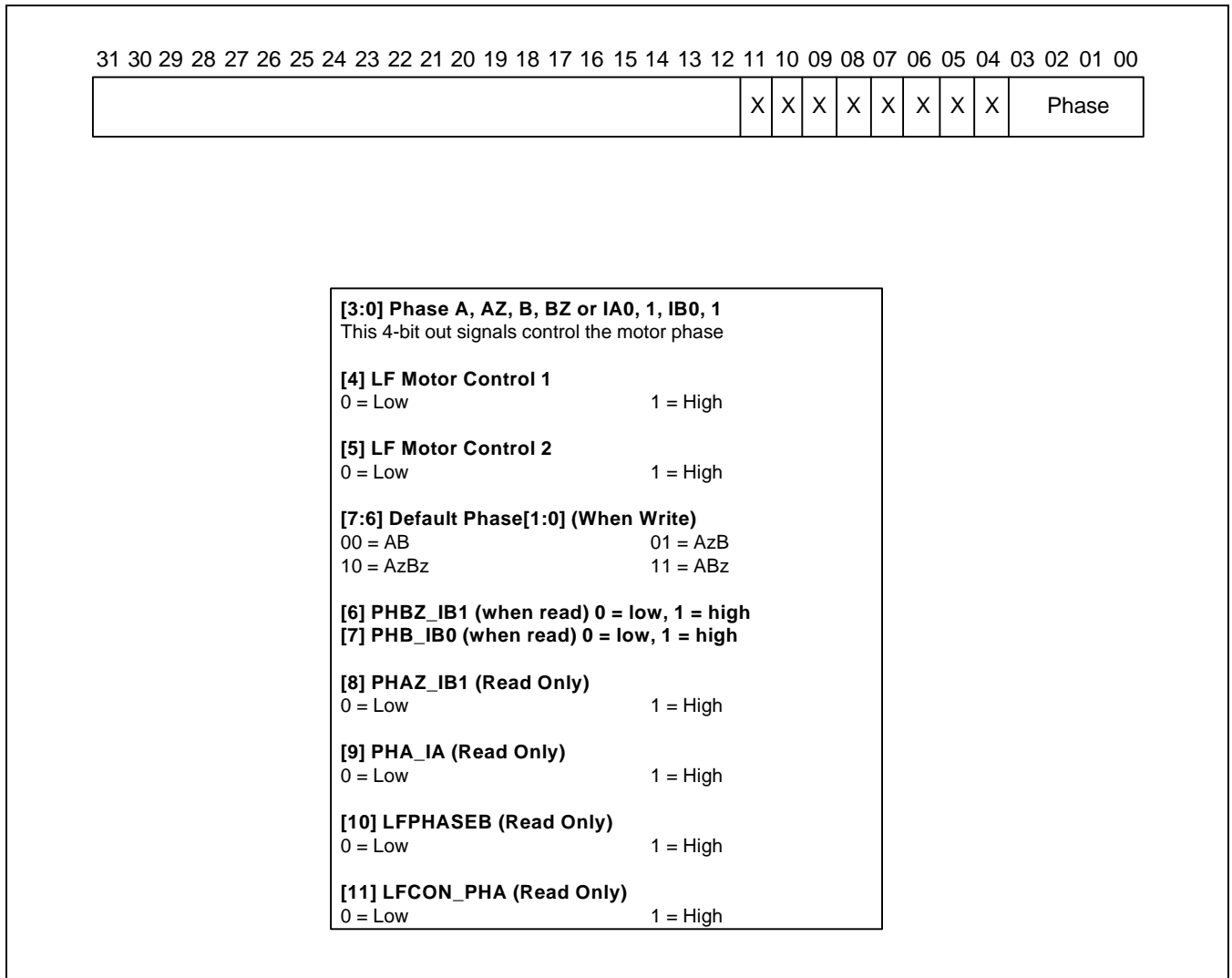


Figure 16-2. LF Motor Phase Control Register

LINE FEED TIMER REGISTER

This 14-bit timer is used to generate the Line feed motor's phase which is driven by software or hardware according to line feed motor control register.

Registers	Offset Address	R/W	Description	Reset Value
LFTBR	0x5808	R/W	Line feed motor timer base register	0x00000000
LFTOR	0x580c	R	Line feed motor timer observation register	0x1e0d
LFTCBR	0x5810	R/W	Line feed motor timer compare base register	0x0000
LFTCOR	0x5814	R	LF motor timer compare observation register	0x0000

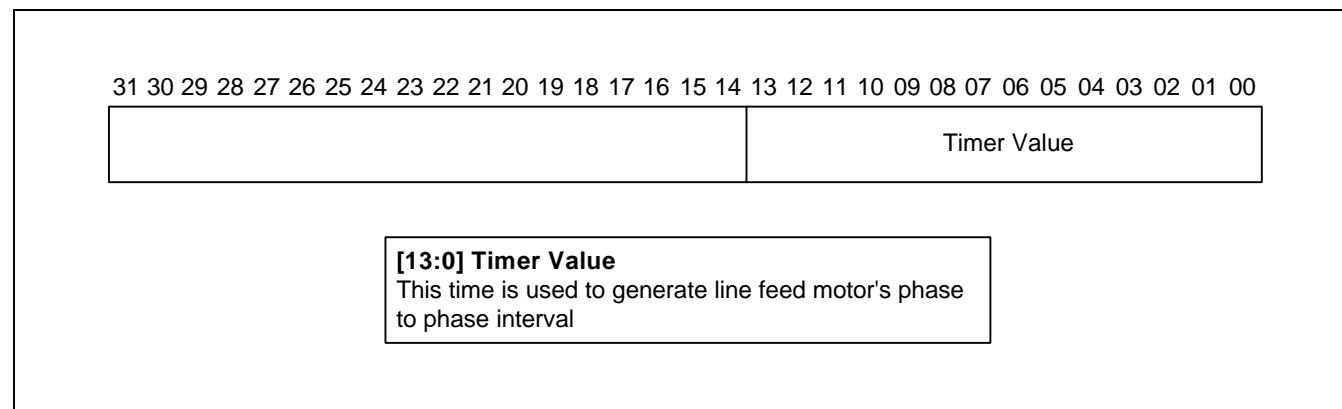


Figure 16-3. LF Motor Timer Register

LFCON EACH CONTROL REGISTER

This register is used to set current level of each steps in bi-polar mode.

Register	Offset Address	R/W	Description	Reset Value
LFCON	0x5818	R/W	LF step each control register	0x0000

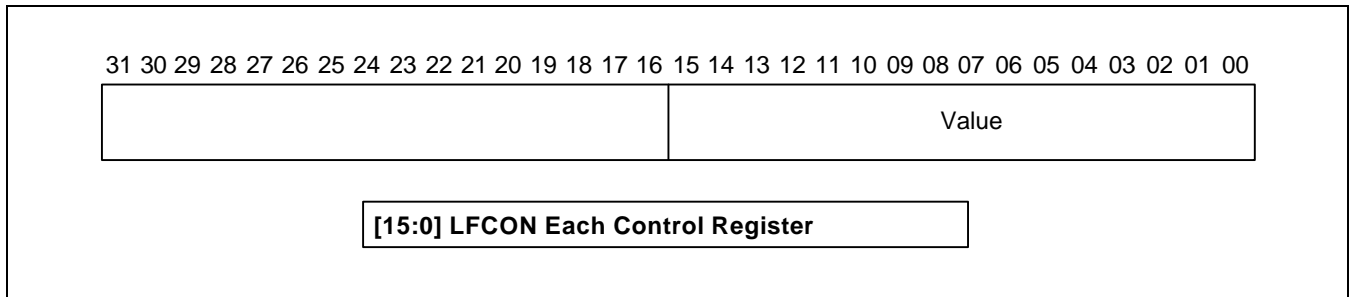


Figure 16-4. LFCON Register

CAUTION

1. When the timer is enabled, it begins to decrease from the base value.
2. When the timer expires, the associated interrupt is generated, the base value is reloaded and the timer continues to decrease.
3. If a new value is loaded in this register before the timer is expired, the timer will keep counting with the new value.
4. In bi-polar mode, lfcr[12] and lfpcr[5:0] should all be set to 0.

PHASE STATE AND CURRENT TABLE FOR FULL/HALF/QUARTER STEP MODE

Direction	Phase State	Current State & Level					Each Control Register	Holding Register = 1	Remark
		IA1	IA0	IB1	IB0	IA/IB%			
	AB	0	0	0	0	100/100%	0	33/33%	
		0	1	0	1	66/66%	1		
	AB	1	0	0	0	33/100%	0	33/33%	
				0	1	33/66%	1		
CW CCW	(A)B	1	1	0	0	0/100%	0	0/33%	
	(Az)B			0	1	0/66%	1		
	AzB	1	0	0	0	33/100%	0	33/33%	
				0	1	33/66%	1		
	AzB	0	0	0	0	100/100%	0	33/33%	
		0	1	0	1	66/66%	1		
	AzB	0	0	1	0	100/33%	0	33/33%	
		0	1			66/33%	1		
CW CCW	Az(B)	0	0	1	1	100/0%	0	33/0%	
	Az(Bz)	0	1			66/0%	1		
	AzBz	0	0	1	0	100/33%	0	33/33%	
		0	1			66/33%	1		
	AzBz	0	0	0	0	100/100%	0	33/33%	
		0	1	0	1	66/66%	1		
	AzBz	1	0	0	0	33/100%	0	33/33%	
			1	0	1	33/66%	1		
	(Az)Bz	1	0	0	0	0/100%	0	0/33%	
	(A)Bz			0	1	0/66%	1		
	ABz	1	0	0	0	33/100%	0	33/33%	
				0	1	33/66%	1		
	ABz	0	0	0	0	100/100%	0	33/33%	
		0	1	0	1	66/66%	1		
	ABz	0	0	1	0	100/33%	0	33/33%	
		0	1			66/33%	1		
CW CCW	A(Bz)	0	0	1	1	100/0%	0	33/0%	
	A(B)	0	1			66/0%	1		
	AB	0	0	1	0	100/33%	0	33/33%	
		0	1			66/33%	1		

17

CR CONTROL

OVERVIEW

This module is configured as follows:

- Basic Timer using MCLK clock is 16-bit-down-counter.
- Prestep timer using 19200/9600 PPI (Pulse Per Inch) clock is 10-bit down-counter.
- Phase and current control signal generation for step motor of bi-polar type.
- Filter for photo sensor input, position counting strobe and direction generation for DC motor.
- Encoder cycle counter of 20-bit up-counter and latch with MCLK clock to calculate the cycle of photo sensor input in DC motor mode.
- Interrupt interval counter of 16-bit up-counter and latch with MCLK/32 clock to calculate the interval of carrier interrupt in DC motor mode.

This module performs the following functions in step motor mode:

- Basic timer generates the basic pulse of 19200/9600 PPI to control the state and position of carrier step motor and to generate fire strobe to control the printhead.
- Prestep timer is used to generate carrier step pulse to control the change of output state signals which are phase and current control signals for carrier motor driver and carrier step interrupt according to carrier motor step rate.
- State control block is used to generate two phases and four current control signals for every step interrupt according to setting of motor direction and state mode.

This module performs the following functions in DC motor mode:

- Filter block is used to protect from false information by noise onto input signals from photo sensor.
- Encoder counter is used to calculate and store the cycle time of preceding input from photo sensor. This cycle time is used for base value of basic timer which generates the basic pulse of 2400 PPI for fire strobe control in DC motor mode, and this time value can be read by CPU.
- Interrupt counter is used to calculate and store the interval time of DC motor interrupt. If this counter overflows before the next interrupt has been observed, an interrupt will be issued, and the counter will go back to zero.
- In DC motor mode, prestep timer is used to set the number of rising edge on preceding input from photo sensor to issue an interrupt for DC motor position control.

NOTES:

1. Writing the value of basic timer base register 1 must be preceded by that of basic timer base register 2 after the reset is done.
2. If the next base value is written to the other base register when the basic timer based on one of the base registers is running, the counter will keep counting with next value. If the next base value is not written to the other register, the counter will repeat counting with current base value until the timer is disabled.
3. In order to set DC motor mode, bit 9 of CMCR has to be set "1".
4. For the step motor mode, CMCR[11:4] has to be set zero .

SPECIAL FUNCTION REGISTER

Carrier Motor Control Register

This register determines whether the interrupt request is generated or not, and enables or disables the prestep timer and basic timer.

Register	Offset Address	R/W	Description	Reset Value
CMCR	0x6000	R/W	Carrier motor control register	0x204

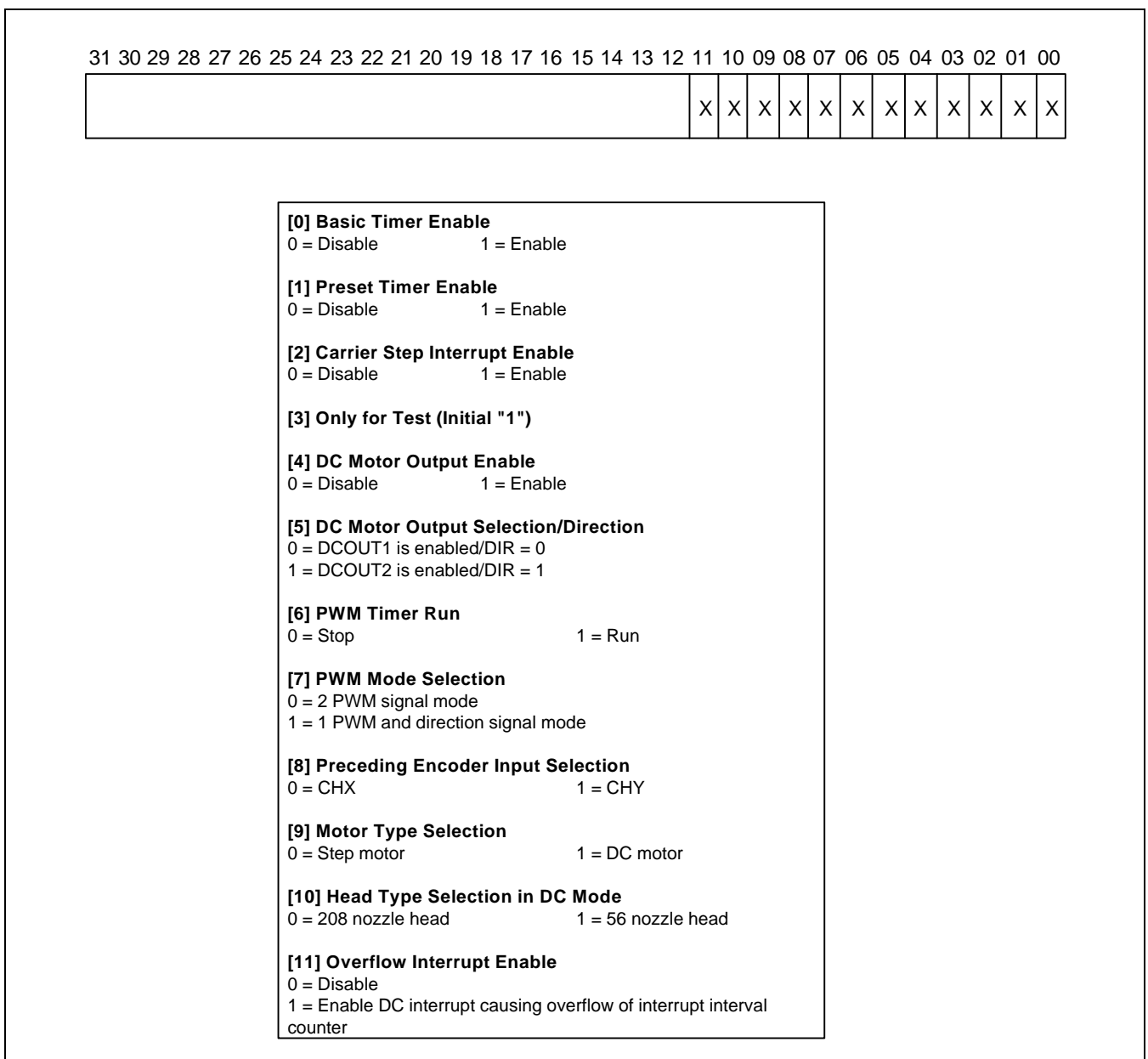


Figure 17-1. Carrier Motor Control Register

Basic Timer Base Register

There are two basic timers, BTB1R and BTB2R. At first, the timer starts with the value of BTB1R, but after down-count stops the timer starts with the value of BTB2R, and only BTB2R is written to a new value. Otherwise, the timer starts with the current base timer value (repeat mode).

Registers	Offset Address	R/W	Description	Reset Value
BTB1R	0x6004	R/W	Basic timer base register 1	0xFFFF
BTB2R	0x6008	R/W	Basic timer base register 2	0xFFFF

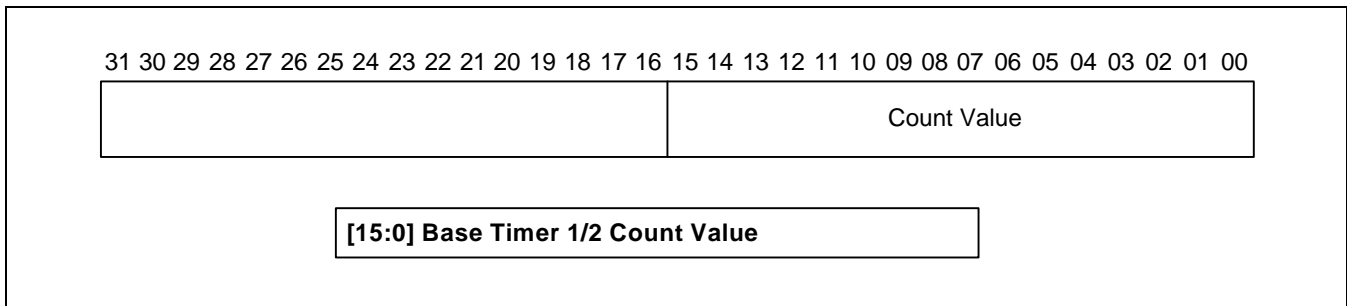


Figure 17-2. Basic Timer Base Register

Prestep Timer Base Register

Register	Offset Address	R/W	Description	Reset Value
PSTBR	0x600c	R/W	CR_Step_INT counter & prestep counter base register	0x000

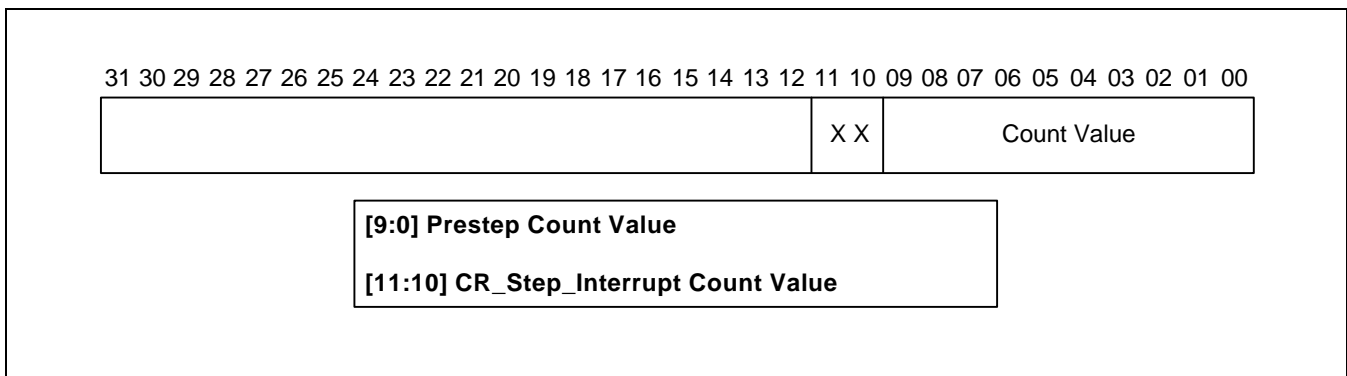


Figure 17-3. Pre-step Timer Base Register

CR State Control Register

This register can generate two phase lines and four current control lines to drive a bipolar stepping motor. Eight output combinations are sequentially presented on these six lines.

Register	Offset Address	R/W	Description	Reset Value
CRSCR	0x6010	R/W	CR state control register	0x603f

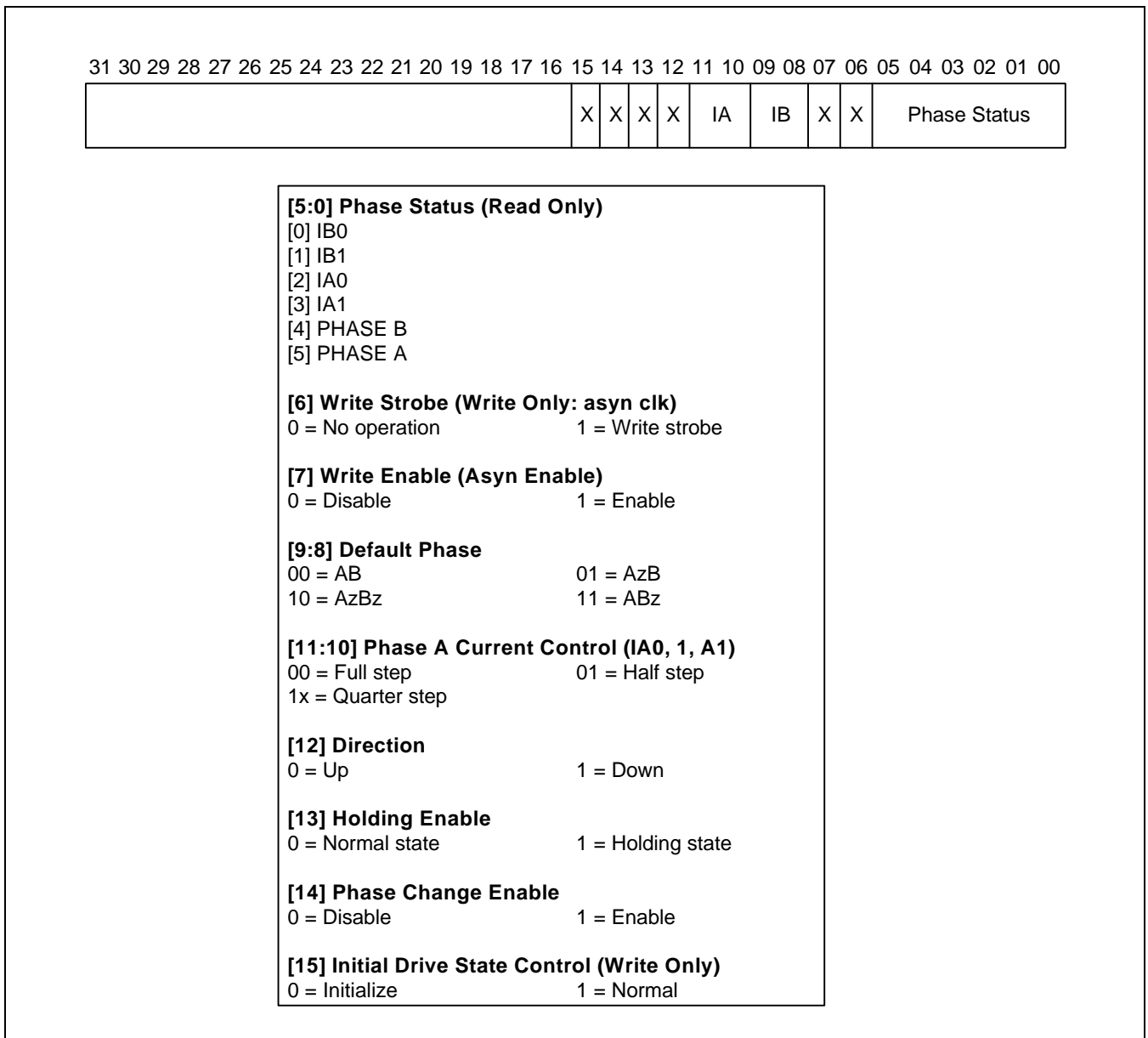


Figure 17-4. CR State Control Register

CRSREG Each Control Register

This register is used to set the active current level for each step in bi-polar mode.
You can refer to the phase state and current table the next page for detailed current level of each step.

Register	Offset Address	R/W	Description	Reset Value
CRSREG	0x6030	R/W	CR step each control register	0x0000

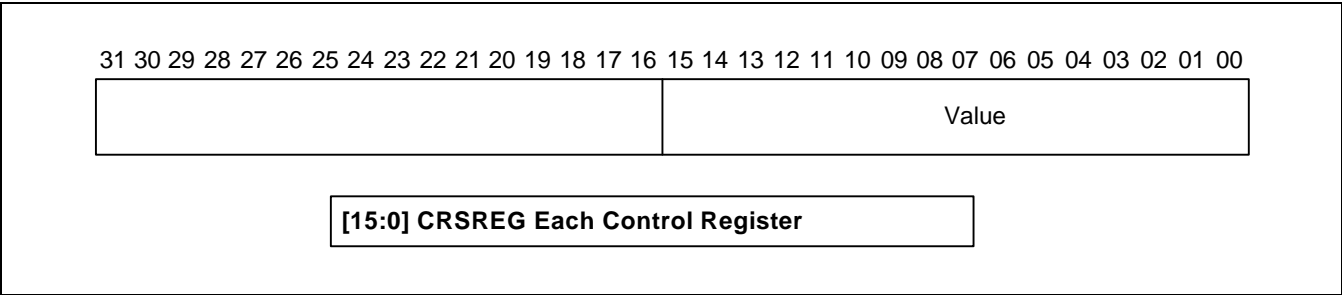


Figure 17-5. CRSREG Register

Phase State and Current Table for Full/Half/Quarter Step Mode

Direction	Phase State	Current State & Level					Each Control Register	Holding Register = 1	Remark
		IA1	IA0	IB1	IB0	IA/IB%			
	AB	0	0	0	0	100/100%	0	33/33%	
		0	1	0	1	66/66%	1		
	AB	1	0	0	0	33/100%	0	33/33%	
				0	1	33/66%	1		
CW	(A)B	1	1	0	0	0/100%	0	0/33%	
CCW	(Az)B			0	1	0/66%	1		
	AzB	1	0	0	0	33/100%	0	33/33%	
				0	1	33/66%	1		
	AzB	0	0	0	0	100/100%	0	33/33%	
		0	1	0	1	66/66%	1		
	AzB	0	0	1	0	100/33%	0	33/33%	
		0	1			66/33%	1		
CW	Az(B)	0	0	1	1	100/0%	0	33/0%	
CCW	Az(Bz)	0	1			66/0%	1		
	AzBz	0	0	1	0	100/33%	0	33/33%	
		0	1			66/33%	1		
	AzBz	0	0	0	0	100/100%	0	33/33%	
		0	1	0	1	66/66%	1		
	AzBz	1	0	0	0	33/100%	0	33/33%	
			1	0	1	33/66%	1		
	(Az)Bz	1	0	0	0	0/100%	0	0/33%	
	(A)Bz			0	1	0/66%	1		
	ABz	1	0	0	0	33/100%	0	33/33%	
				0	1	33/66%	1		
	ABz	0	0	0	0	100/100%	0	33/33%	
		0	1	0	1	66/66%	1		
	ABz	0	0	1	0	100/33%	0	33/33%	
		0	1			66/33%	1		
CW	A(Bz)	0	0	1	1	100/0%	0	33/0%	
CCW	A(B)	0	1			66/0%	1		
	AB	0	0	1	0	100/33%	0	33/33%	
		0	1			66/33%	1		

CR_PWM TIMER

Logic Configuration

The PWM block is configured of the Cycle_Time base register, On_Time base register, counter observation register and 16-bit down-counter.

Function

- The PWM output signal's period and the On/Off time within it is decided by the Cycle_Time base value and the On_Time base value. If the On_Time base value is the same or larger than the Cycle_Time base value, the PWM output signal maintains On status.
- The 16-bit down-counter's RUN (enable) or STOP (disable) status is decided by the CMCR[6].
- The PWM block operation and the output according to CMCR's bits 4, 5 and 7 are shown in the following table.

CMCR	Description	DC motor control signal output status			
CMCR[4]	DC motor output enable	Enables the PWM outputs			
CMCR[7]	PWM mode selection	0		1	
CMCR[5]	DC motor direction/output selection	0	1	x	x
DC_CRIA0 Pin		PWM signal	0	PWM signal	
DC_CRIA1 Pin		0	PWM signal	Direction	

Counter Base Register and Observation Register

Registers	Offset Address	R/W	Description	Reset Value
PWMOBS	0x6014	R	PWM counter observation register	0x0000
PWMCYL	0x6018	R/W	PWM cycle time base register	0x0000
PWMONT	0x601C	R/W	PWM on time base register	0x0000

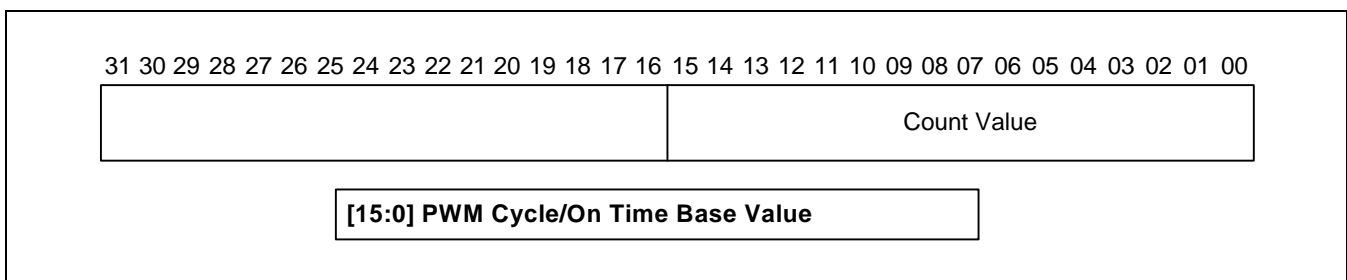


Figure 17-5.PWM Counter Base Register

Caution

Real cycle time = (base value + 1) × 1/MCLK

Real on time = (base value + 1) × 1/MCLK

— This block only operates in DC mode.

ENCODER COUNTER

Logic Configuration

This block is configured of a 20-bit up-counter and a 20-bit register for storing the counting results.

Function

- It counts the period of the photo sensor (encoder sensor) input and stores the value in the register.
- The period is used as a base value for calculating the fire strobe and fire window time according to the setting of the fire DPI.

Countering Result Register and Observation Register

Registers	Offset Address	R/W	Description	Reset Value
ECDTIM	0x6020	R	Encoder counter observation register	0x20292
ECDVAL	0x6024	R	Encoder cycle value register	0x00000

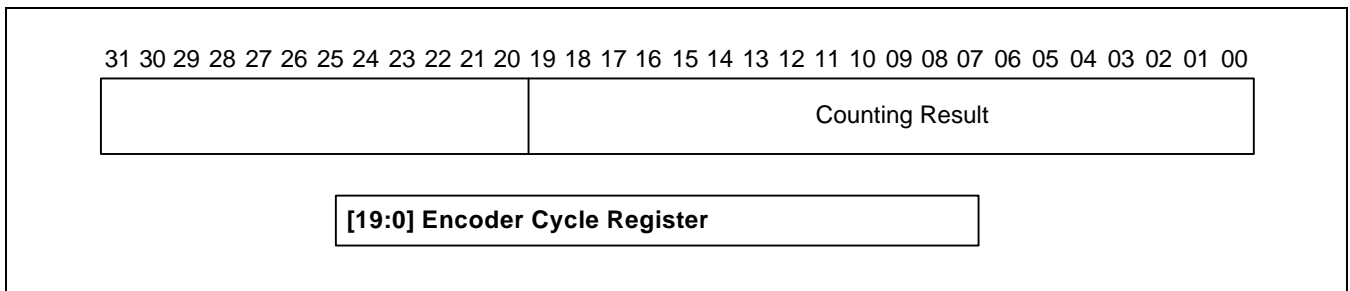


Figure 17-6. Encoder Cycle Register

Caution

— This block only operates in DC mode.

INTERRUPT INTERVAL COUNTER

Logic Configuration

This block is composed of a 16-bit up-counter and a 16-bit register for storing the counting results.

Function

- This logic starts operating after the first DC motor interrupt is generated.
- It counts the interval between each DC motor interrupt and the next, and stores the value in the register.
- When CMCR[11] is set to 1 and the counter overflows before the generation of the next interrupt, the DC motor interrupt is generated, and the counter and pre-step timer are cleared and restarted.
- The DC motor interrupt is generated in DC mode when the photo (encoder) sensor input's rising edge occurs for the number of times specified in the pre-step timer.

Counting Result Register and Observation Register

Registers	Offset Address	R/W	Description	Reset Value
INTTIM	0x6028	R	Interval counter observation register	0x0000
INTVAL	0x602C	R	Interrupt interval value register	0x0000

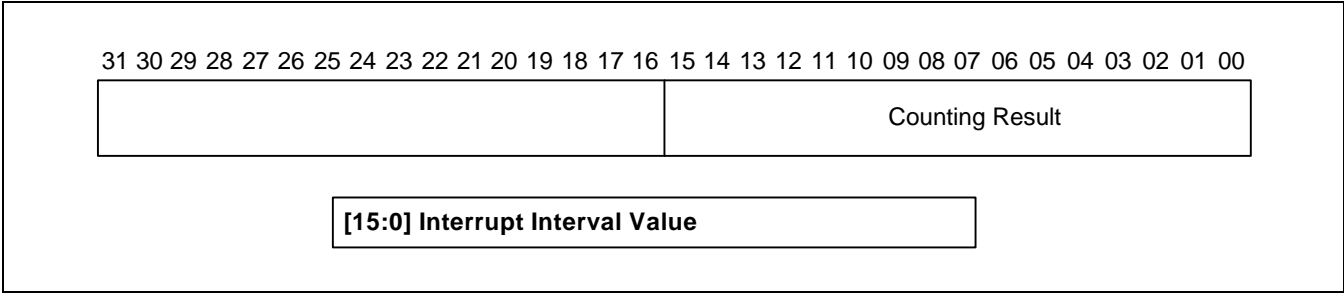


Figure 17-7. Interrupt Interval Value Register

SUGGESTIONS FOR CARRIER MOTOR DRIVE F/W DESIGN**When CR Motor is Stopped**

- The BASIC timer and PRESTEP timer's RUN bit must be reset to "0".
{Bits 0, 1 of CMCR(0x6000)}
- The position block enable bit (bit 0 of PFCR(0x6820)) must be reset to "0".

When CR Motor is Restarted

- Must write new (or previous) values in the BASIC timer base registers.
- Must write new (or previous) values in the PRESTEP timer base Register.
- Must set the position block enable bit (bit 0 of PFCR(0x6820)) to "1".
- Must set BASIC timer and PRESTEP timer's RUN bit to "1".

In other words, before starting Re-RUN after stopping the BASIC timer, you must:

- Rewrite the BASIC timer base register and PRESTEP timer base register values, and
- Reset the position block enable bit before setting, to reduce the error in carrier motor position.

To reduce location errors, you should fix the position & fire control register's bit 6 (position counter clock) to 1, adjust the position and fire Pre-Scaler values, and set the fire DPI.

- The value of position & fire control register's bit 6 should not be changed during system operation.

18

CR FIRE

OVERVIEW

This module performs the following functions:

- Count and control the position of carrier motor
- Fire strobe and start signal generation
- 16-bit counter for the position of carrier motor
- 16-bit print slice counter for counting fire strobe
- 6-bit prescaler for the clock of carrier position
- 8-bit prescaler for the clock of fire strobe

NOTES

1. This block is responsible for positioning the printhead and regulating the printhead fire strobe timing.
2. Two conditions must be met before the fire strobe logic can be activated.
First, the print slice count must be greater than zero. Second, the position counter must be equal to the print start position.
3. When the start position is reached, the fire logic is enabled, and the first fire strobe is generated. Each fire strobe decrease the slice count by one.
When the slice count reaches to zero, the fire logic is disabled.
4. For step motor mode, the cycle of fire strobe is decided by setting only the base value of the fire prescaler. (Fire DPI = 19200/Fire prescaler)
For DC motor mode, the cycle of fire strobe is decided by the setting base value of the fire prescaler and DPI mode setting bit of PFCR.

SPECIAL FUNCTION REGISTER

POSITION & FIRE CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
PFCR	0x6820	R/W	Position & fire control register	0x0080d0

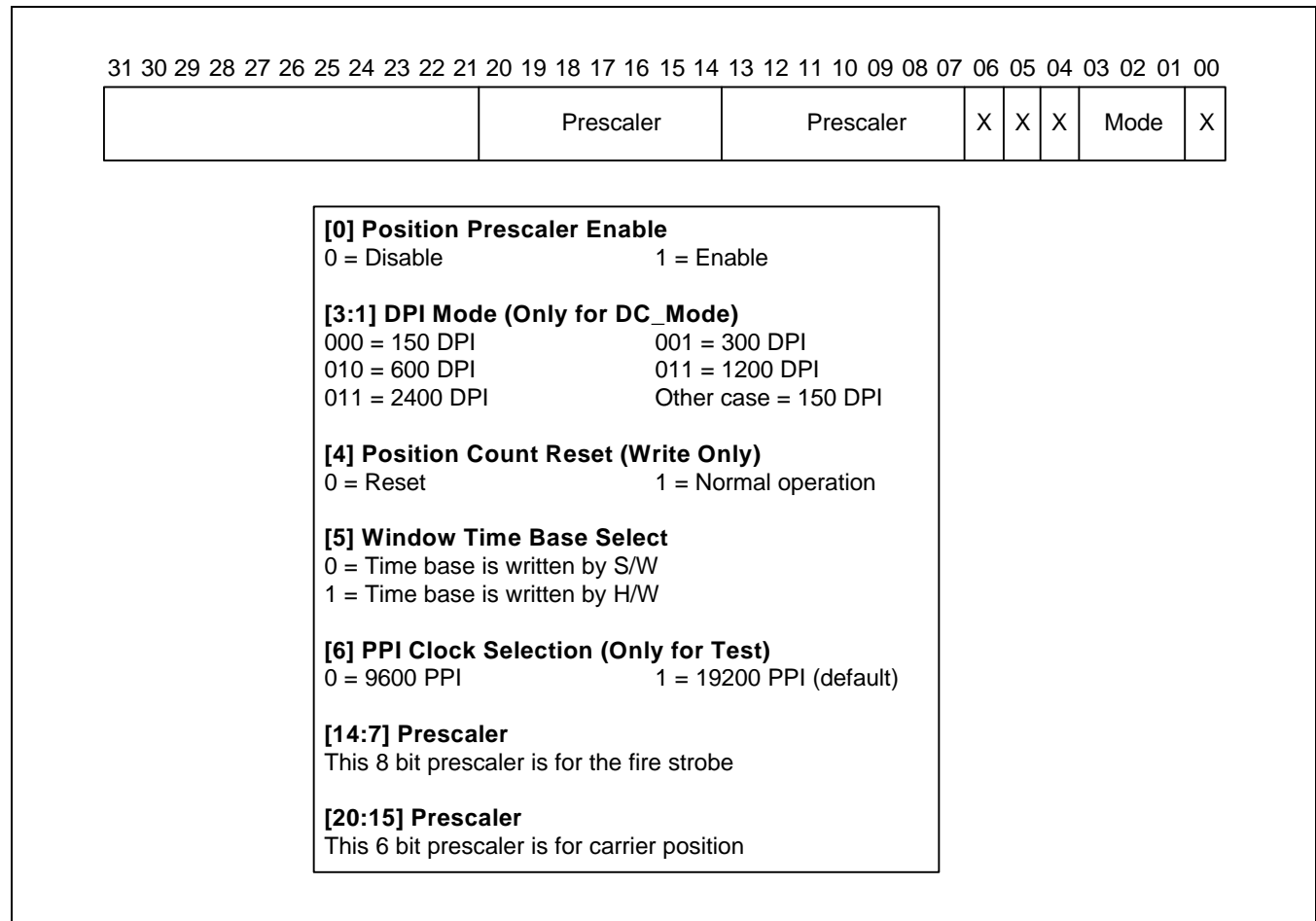


Figure 18-1. Position & Fire Control Register

CR POSITION AND FIRE CONTROL REGISTER

There are four registers in this block: Carrier Position Count Register(CPCR), Print Start Position Register (PSPR), Print Slice Counter Register(PSCR), and Position Interrupt Register(PIR).

CPCR: The carrier position is updated based on the carrier movement of 1/600 inch.

PSPR: The fire strobe control logic requires two conditions to be met before it will generate fire strobe to print logic.

- A non-zero value must be loaded into the print slice counter register.
- The carrier position must match the value in the print start position register.
- Once the two requirements have been met, the logic will begin producing fire strobes after 1/75 inch.

PSCR: This value is decreased once for each fire strobe that is generated.

PIR: When outputs of carrier position counter become same as the value of this register, position interrupt request occurs.

Registers	Offset Address	R/W	Description	Reset Value
CPCR	0x6824	R/W	Carrier position counter register	0x0000
PSPR	0x6828	R/W	Print start position register	0xffff
PSCR	0x682c	R/W	Print slice counter register	0x0000
PIR	0x6830	R/W	Position interrupt register	0xffff

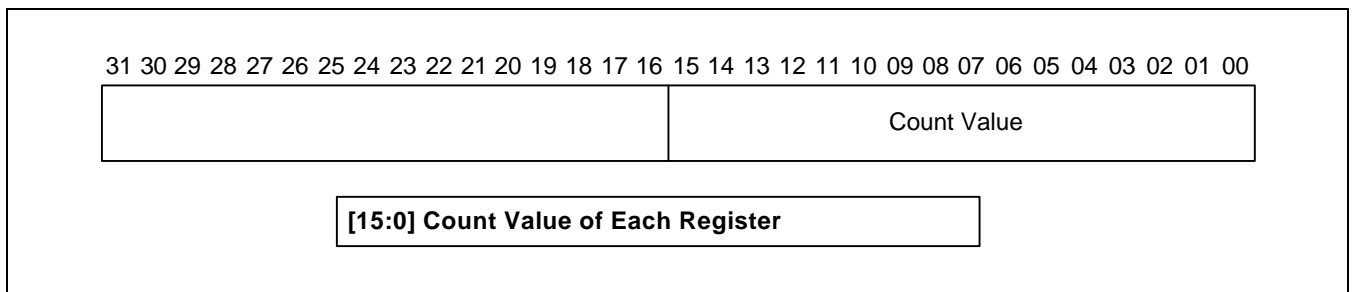


Figure 18-2. CR Count Register

SUGGESTIONS FOR FW DESIGN

When CR Motor is Stopped

- Must reset BASIC timer and PRESTEP timer's RUN bit to "0".
- Must reset the position block enable bit (bit 0 of PFCR(0x6820)) to "0".

When Restarting the CR Motor

- Must write new (or previous) values in the BASIC timer base registers.
- Must write new (or previous) values in the PRESTEP timer base register.
- Must set the position block enable bit (bit 0 of PFCR(0x6820)) to "1".
- Must set the BASIC timer and PRESTEP timer's RUN bit to "1".

In other words, before starting Re-RUN after stopping the BASIC timer, you must:

- Rewrite the BASIC timer base register and PRESTEP timer base register values, and
- Reset the position block enable bit before setting to reduce the error in carrier motor position.

To reduce location errors, you should fix the position & fire control register's bit 6 (position counter clock) to 1, adjust the position and fire pre-scaler values, and set the fire DPI.

The value of position & fire control register's bit 6 should not be changed during system operation.

Example of Position DPI Setting for Step Motor Mode

Position DPI	Pre-scaler Value(PFCR[20:15])
300 DPI	64 (19200/300)
600 DPI	32 (19200/600)
1200 DPI	16 (19200/1200)

Example of Fire DPI Setting

Fire DPI	DC Motor Mode		Step Motor Mode
	Prescaler value (PFCR[14:7])	DPI mode setting value	Prescaler value (PFCR[14:7])
150 DPI	16 (2400/150)	000, other case	128 (19200/150)
300 DPI	8 (2400/300)	001	64 (19200/300)
600 DPI	4 (2400/600)	010	32 (19200/600)
1200 DPI	2 (2400/1200)	011	16 (19200/1200)
2400 DPI	1 (2400/2400)	100	8 (19200/2400)

19

PRINT HEAD

OVERVIEW

This module performs the following functions:

- Fire pulse generation.
- DMA request for reading data.
- Three 32-bit dot counters for color.
- One 32-bit dot counter for mono.
- Fire strobe delay for horizontal alignment of dot.
- 8-bit decrement timer for the width of the fire enable pulse of print head logic using MCLK.
- 10-bit decrement timer for the width of the fire group window of print head logic using MCLK.
- Four 12-bit timers for the fire strobe delay using selectable clock. (clock = main clock/1, 2, 4, or 8)
- 4-bit decrement counter for Td delay.
- 6-bit pre-heat pulse timer.
- 6-bit pre-heat delay timer.

Head Control Register			Head Type	Number of Data
PHCR[8]	PHCR[7]	PHCR[11]		
0	0	0	DH, mono head (208 nozzle)	13 half-word
0	1	0	DH, colour head (192 nozzle)	12 half-word
1	0	0	SH, mono head (56 nozzle)	4 half-word
1	1	0	SH, colour head (48 nozzle)	3 half-word
1	0	1	SH, mono head (56 nozzle)	7 bytes

SPECIAL FUNCTION REGISTER

PRINT HEAD CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
PHCR	0xa000	R/W	Print head control register	0x000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
						X	X	Address		CLK	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Dither			

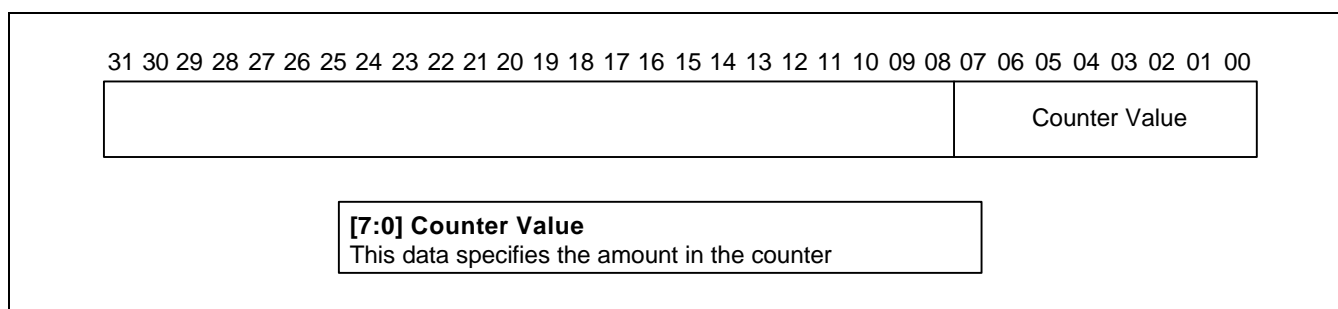
[3:0] Dither Count Bit	
[4] Decrement Through Enable Group	
0 = Up	1 = Down
[5] Black Dot Counter Select	
0 = Disable	1 = Enable
[6] Color Dot Counter Select	
0 = Disable	1 = Enable
[7] Color Head Select	
0 = Black	1 = Color
[8] Head Type Select	
0 = 208/192 nozzle head	1 = 56/48 nozzle head
[9] Consecutive Dot Eliminator	
0 = Disable	1 = Enable
[10] Top Nozzle Group Select	
0 = Right	1 = Left
[11] Vertical 300DPI Mode	
0 = Half-word	1 = Byte
[12] Horizontal 300DPI Mode	
0 = Disable	1 = Enable
[13] Decompression Mode	
0 = Disable	1 = Enable
[14] Data Select	
0 = By H/W	1 = By S/W
[15] Simulation Test Control (HDMA Request Generation)	
0 = Disable	1 = Enable
[16] Perform a Fire Cycle (Write Only)	
0 = Disable	1 = Enable
[17] Perform a Data Cycle (Write Only)	
0 = Disable	1 = Enable
[19:18] Clock Select	
00 = MCLK/1	10 = MCLK/4
01 = MCLK/2	11 = MCLK/8
[23:20] Address Line (for the Nozzle)	
[24] Address Line (by S/W)	
0 = Disable	1 = Enable
[25] Current Mode	
0 = Printing	1 = Scanning

Figure 19-1. Print Head Control Register

FIRE ENABLE TIMER/OBSERVATION REGISTER

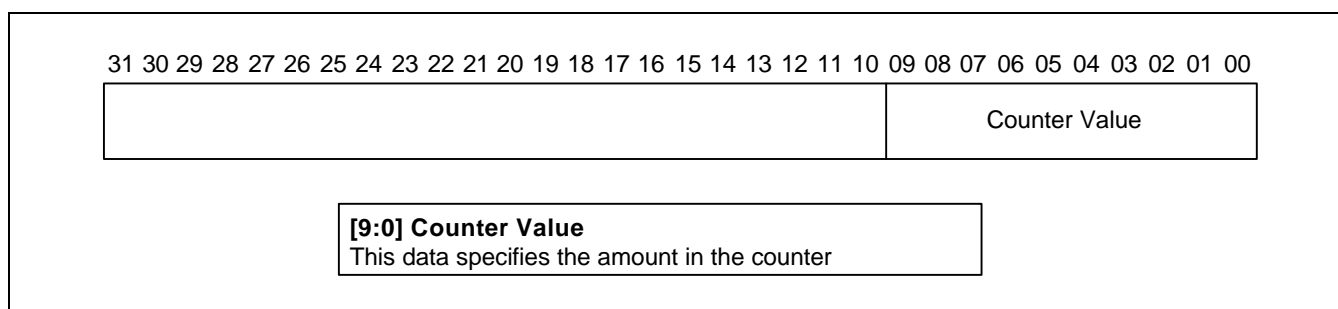
This 8-bit timer is used for fire enable duration counter value.
The observation register is read-only which is of the current value.

Registers	Offset Address	R/W	Description	Reset Value
FETR	0x7004	R/W	Fire enable timer register	0x00
FETOR	0x7008	R	Fire enable timer observation register	0x00

**Figure 19-2. Fire Enable Timer/Observation Register****FIRE WINDOW TIMER/OBSERVATION REGISTER**

This 10-bit timer is used for the fire window enable duration counter value.
The observation register is read-only which is of the current value.

Registers	Offset Address	R/W	Description	Reset Value
FWTR	0x700c	R/W	Fire window timer register	0x000
FWTOR	0x7010	R	Fire window timer observation register	0x000

**Figure 19-3. Fire Window Timer/Observation Register**

FIRE STROBE DELAY TIMER/OBSERVATION REGISTER

This 12-bit timer is used for the fire strobe delay duration counter value.

The print head logic contains four 12-bit timers of the carrier used to delay the fire strobes from the carrier motor logic before sending them to the print head drivers. The timers alternate for each fire strobe.

When the timer delay cycle expires, the timer will be reset to the first fire stroke value. When the timer delay cycle expires, the timer will be reset to the first fire stroke value. When the timer delay cycle expires, the timer will be reset to the first fire stroke value.

Registers	Offset Address	R/W	Description	Reset Value
FSDTR	0x7014	R/W	Fire strobe delay timer register	0x000
FSDT0OR	0x7018	R	Fire strobe delay timer 0 observation register	0x000
FSDT1OR	0x701c	R	Fire strobe delay timer 1 observation register	0x000
FSDT2OR	0x7020	R	Fire strobe delay timer 2 observation register	0x000
FSDT3OR	0x7024	R	Fire strobe delay timer 3 observation register	0x000

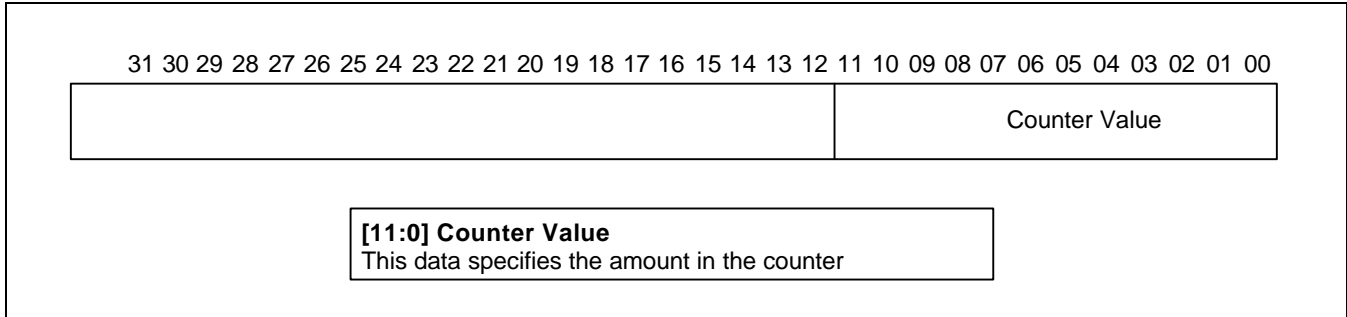


Figure 19-4. Fire Strobe Delay Timer/Observation Register

PRE-HEAT PULSE TIMER/OBSERVATION REGISTER

This 6-bit timer is used for the pre-heat pulse enable duration counter value.

The observation register is read-only which is of the current value.

Registers	Offset Address	R/W	Description	Reset Value
PHPTR	0x7028	R/W	Pre-heat pulse timer register	0x00
PHPTOR	0x702c	R	Pre-heat pulse timer observation register	0x00

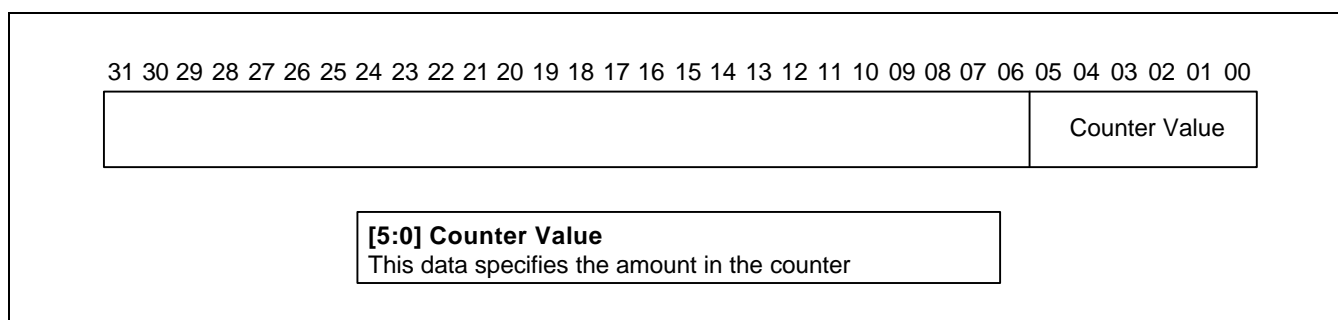


Figure 19-5. Pre-Heat Pulse Timer/Observation Register

PRE-HEAT DELAY TIMER/OBSERVATION REGISTER

This 6-bit timer is used for the pre-heat delay enable duration counter value.

The observation register is read-only which is of the current value.

Registers	Offset Address	R/W	Description	Reset Value
PHDTR	0x7030	R/W	Pre-heat delay timer register	0x00
PHDTOR	0x7034	R	Pre-heat delay timer observation register	0x00

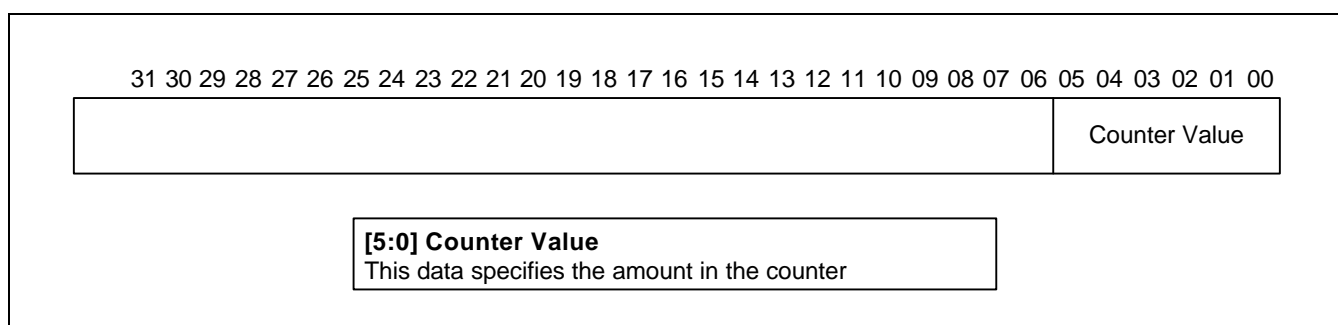
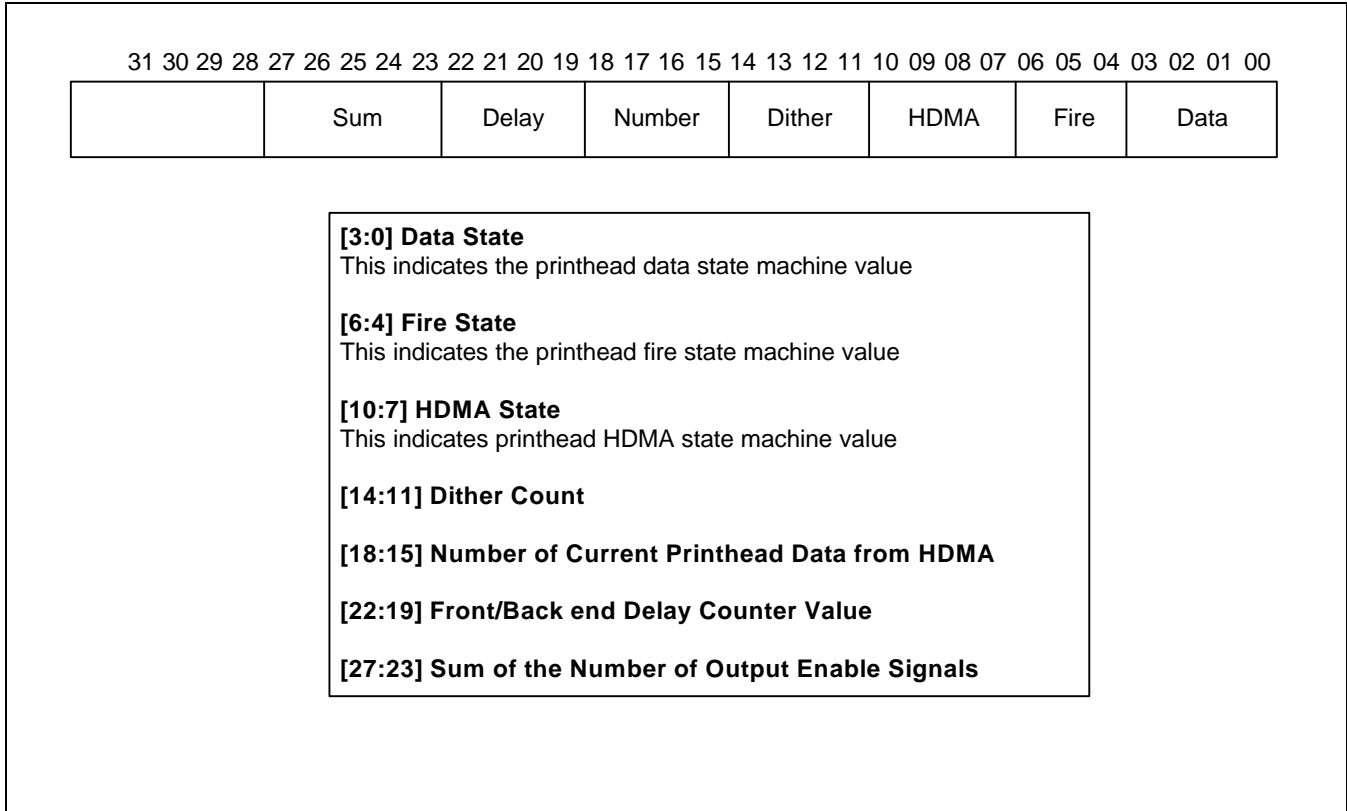


Figure 19-6. Pre-Heat Delay Timer/Observation Register

PRINthead OBSERVATION REGISTER

Register	Offset Address	R/W	Description	Reset Value
PHOR	0x7038	R	Print Head observation register	0x0000000



FRONT AND BACK END DELAY COUNTER REGISTER

This 4-bit timer is used for the front/back end delay duration counter value.

Register	Offset Address	R/W	Description	Reset Value
TDCR	0x703c	R/W	Td delay counter register	0x00

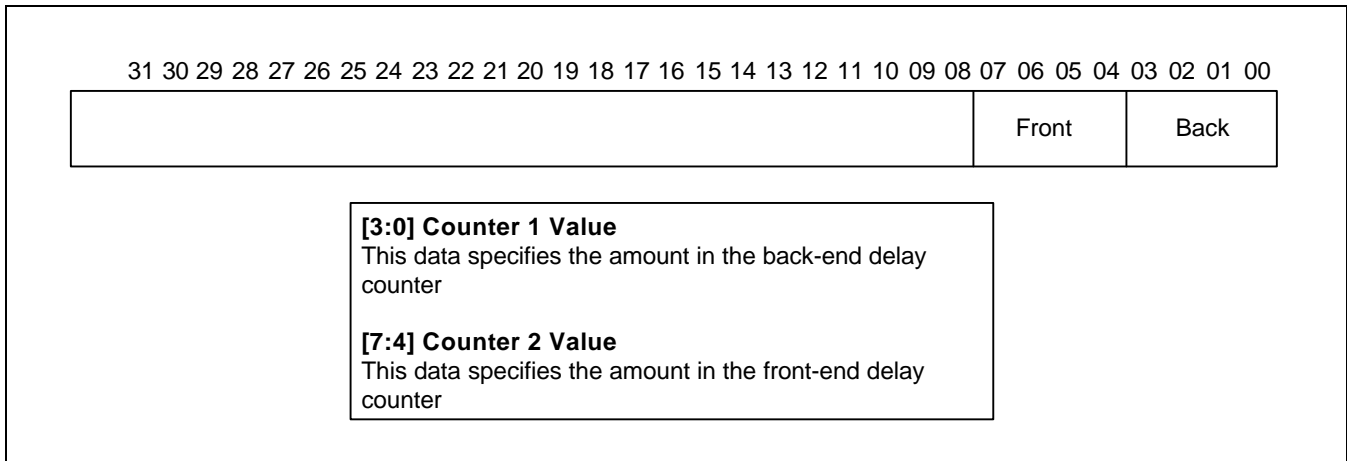


Figure 19-8. Td Delay Counter Register

PRINT HEAD DATA WORD REGISTER

Registers	Offset Address	R/W	Description	Reset Value
PHDW0R	0x7040	R/W	Print head data word 0 register	0x0000
PHDW1R	0x7044	R/W	Print head data word 1 register	0x0000
PHDW2R	0x7048	R/W	Print head data word 2 register	0x0000
PHDW3R	0x704c	R/W	Print head data word 3 register	0x0000
PHDW4R	0x7050	R/W	Print head data word 4 register	0x0000
PHDW5R	0x7054	R/W	Print head data word 5 register	0x0000
PHDW6R	0x7058	R/W	Print head data word 6 register	0x0000
PHDW7R	0x705c	R/W	Print head data word 7 register	0x0000
PHDW8R	0x7060	R/W	Print head data word 8 register	0x0000
PHDW9R	0x7064	R/W	Print head data word 9 register	0x0000
PHDW10R	0x7068	R/W	Print head data word 10 register	0x0000
PHDW11R	0x706c	R/W	Print head data word 11 register	0x0000
PHDW12R	0x7070	R/W	Print head data word 12 register	0x0000

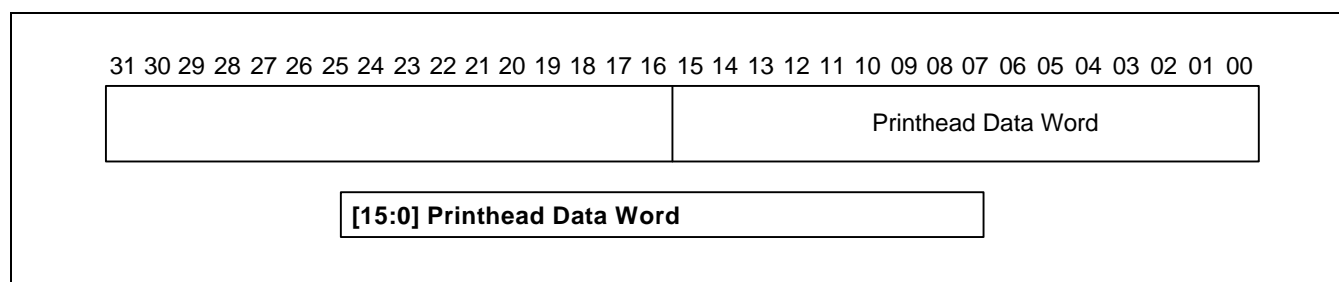


Figure 19-9. Print Head Data Word Register

DOT COUNTER REGISTER

Registers	Offset Address	R/W	Description	Reset Value
DCBR	0x7074	R/W	Dot counter black register	0x00000000
DCYR	0x7078	R/W	Dot counter yellow register	0x00000000
DCCR	0x707c	R/W	Dot counter cyan register	0x00000000
DCMR	0x7080	R/W	Dot counter magenta register	0x00000000

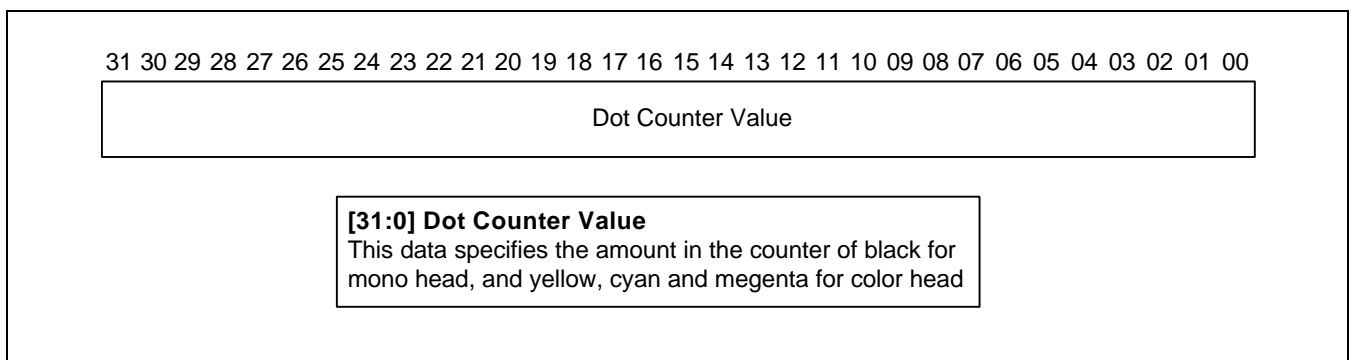


Figure 19-10. Dot Counter Register

DOT COUNTER CONTROL OBSERVATION REGISTER

Register	Offset Address	R/W	Description	Reset Value
DCCOR	0x7084	R	Dot counter control observation register	0x000

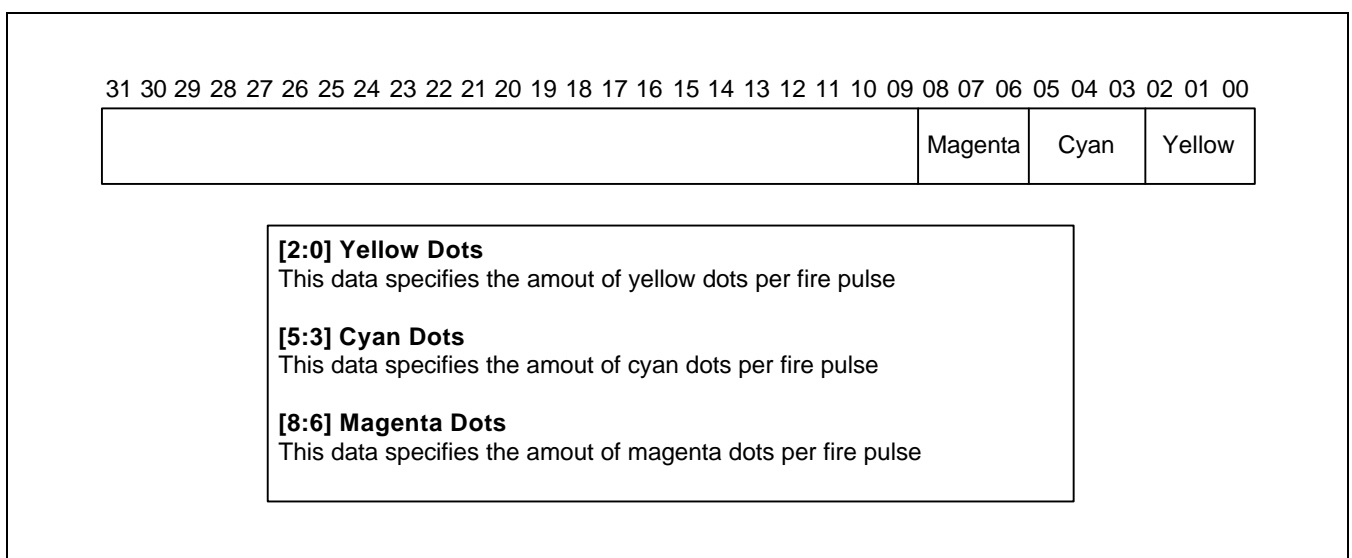


Figure 19-11. Dot Counter Control Observation Register

Caution

- The print head logic is responsible for receiving image data, conditioning the data for print, and routing the data to the print head in the proper sequence. The nature of the print head design is the primary motivation behind the structure of the print head firing logic.
- The print head contains 208 nozzles arranged in two columns that are divided into sixteen groups containing 13 nozzles each. The print head is configured so that only one nozzle from each group may be fired at any time. This necessitates a sequential firing scheme passing through each of the 13-nozzle group, one at a time, firing a maximum of sixteen nozzles.
- Many of the critical timing relationships for the print head firing are controlled by carrier motor logic. It is imperative that print head motion and nozzle firing be directly linked to produce the desired print output characteristics. Additional software control is provided to aid print alignment.
- Data for the print head logic is received from memory via the HDMA pair or directly from the KS32C65100. During a print fire cycle, the logic will issue a data request for 7 byte, 3, 4, 12, or 13 half words of data transferred per print slice.

20

HDMA

OVERVIEW

This module is used to transfer head data from memory to the head data register by DMA with match function.

HDMA SPECIAL REGISTERS

HEAD DMA CONTROL REGISTER

This DMA has a kind of DMA operation under the control of the print module. HDMA reads from memory, and writes to head. HDMA can transfer data by bytes/half-words. The transfer size is decided by setting the head control register.

Register	Offset Address	R/W	Description	Reset Value
HDCON	0x7800	R/W	Head DMA control register	0x0000000

- [0] Run enable/disable
When you set this bit to '1', HDMA operation starts. To stop HDMA, you must clear this bit to '0'. If you control this bit only, 0x7810 address will be used (if 0x7810 address is used, other value will not be changed).
- [1] BUSY status
When HDMA starts, this read-only status bit is automatically set to '1'.
When it is '0', HDMA is in idle status.
- [2] Source address direction
Only one source can initiate an HDMA operation.
If this bit is set, the source address will be decreased.
If this bit is cleared, the source address will be increased.
- [3] Source address fix
This bit determines whether the source address will be changed or not during an HDMA operation. This feature is used when transferring data from a single source to multiple destinations.
- [4] Reset
If this bit is set to '1', then the HDMA control register value will be initialized, after this bit is cleared to '0', you can specify other control values.
- [5] Not Used
- [6] Match pend status
If the value of the source address register (HDSAR) and the value of the match address register (HDMAR) are matched, the match pend status bit is set. If you would like

[7] Match interrupt	<p>to clear the status bit, write zero.</p> <p>This bit determines whether the interrupt pending by match of source/match pending enable address register occurs or not. In the case of a match, HDMA operates until the source address is the match address.</p>
[8] HDMA Interrupt enable	<p>An HDMA operation is started/stopped by setting/clearing the run enable/disable bit. If this bit is set to '1' when DMA starts, a 'stop interrupt' is generated when HDMA operation stops. If this bit is '0', an interrupt and match interrupt are not generated.</p>
[9] Auto Load	<p>This bit should be enabled for source address register's parallel load.</p>
[10] Alternate Enable	<p>This bit determines to alternate register banks.</p>
[11] Current queuing	<p>This bit indicates whether the current queuing bank is '0' or '1'. You can bank selection set/clear the queuing bit selection.</p>
[16] Queuing 0 enable	<p>If this bit is set and HDCON[10] is set, HDMA alternates bank 0.</p>
[24] Queuing 1 enable	<p>If this bit is set and HDCON[10] is set, HDMA alternates bank 1.</p>

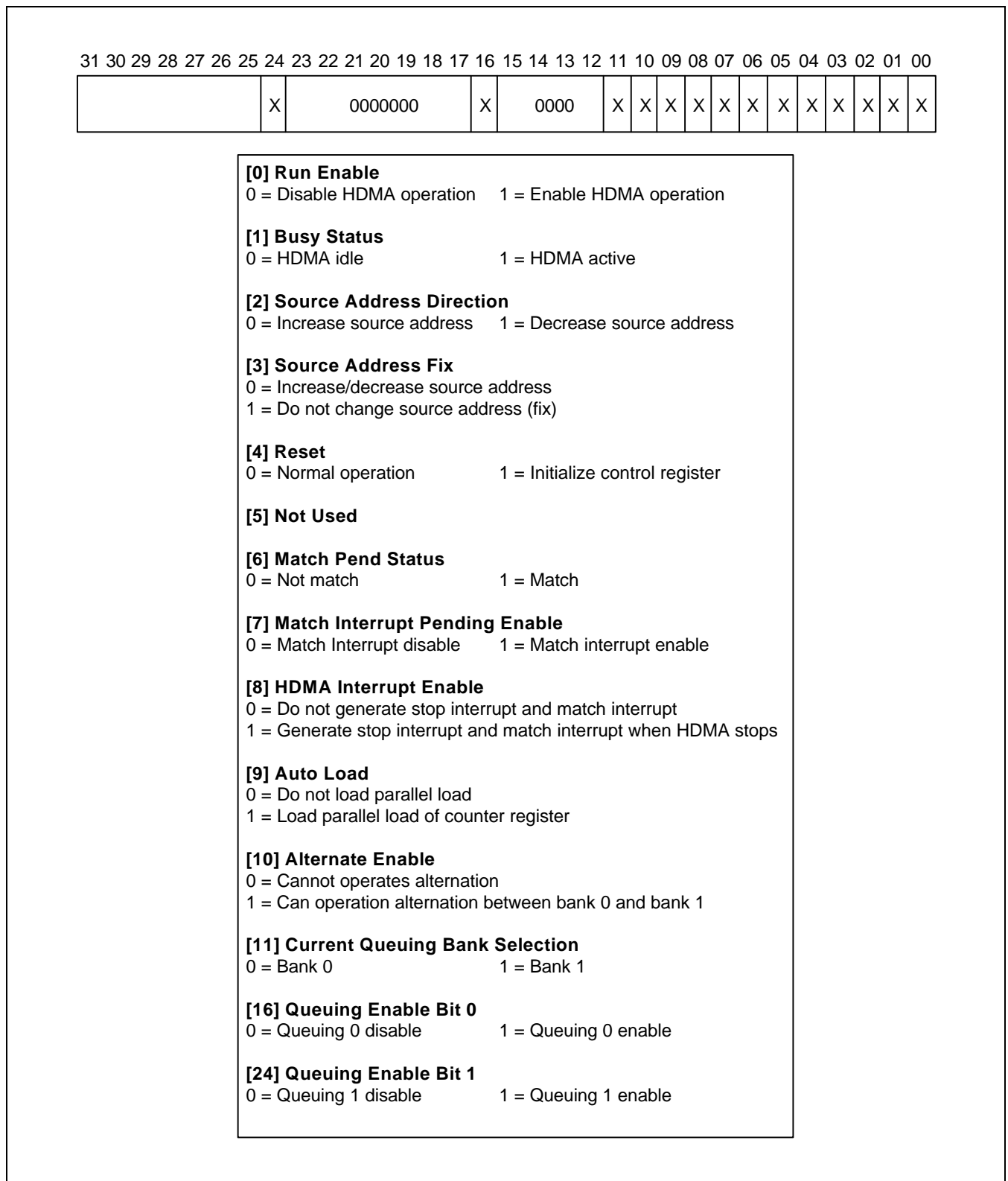


Figure 20-1. HDMA Control Register

HDMA SOURCE ADDRESS REGISTER

These registers contain the 28-bit source/destination address for the HDMA channel.
Depending on the settings you make to the HDMA control register(HDCON), theses adr will be fixed, increased or decreased.

Register	Offset Address	R/W	Description	Reset Value
HDSAR	0x7804	R/W	HDMA source address register	0x0000000

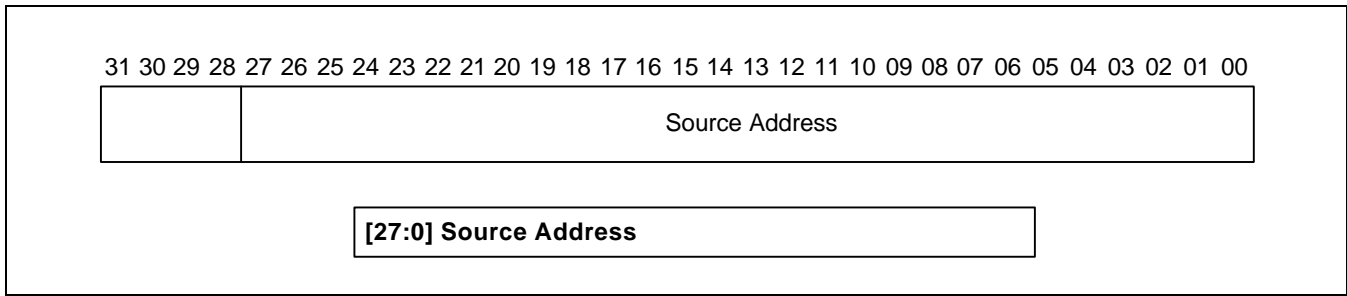


Figure 20-2. HDMA Source Address

HDMA TRANSFER COUNT REGISTER

This register contains the 24-bit current count value of the number of HDMA transfers completed for HDMA. This count value is decreased by 1 while one DMA operation is completed regardless of transfer width.

Register	Offset Address	R/W	Description	Reset Value
HDTCR	0x780c	R/W	HDMA transfer count register	0x0000000

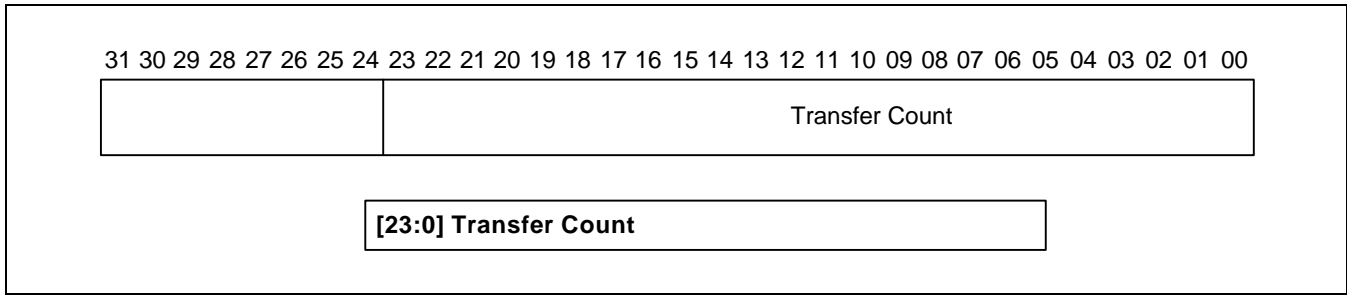


Figure 20-3. HDMA Transfer Count Register

HDMA SOURCE/MATCH ADR REGISTER

These registers contain the 28-bit source/destination address for the HDMA channel.

Depending on the settings you make to the HDMA control register (HDCON), these adr will be fixed, increased or decreased.

Registers	Offset Address	R/W	Description	Reset Value
HDSAR0	0x7814	R/W	HDMA source address register 0	0x0000000
HDMA0	0x7818	R/W	HDMA match address register 0	0x0000000
HDSAR1	0x781c	R/W	HDMA source address register 1	0x0000001
HDMA1	0x7820	R/W	HDMA match address register 1	0x0000000

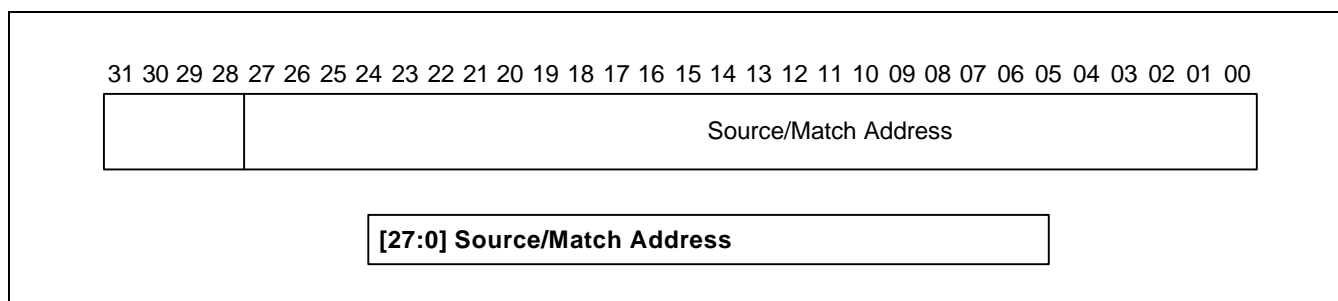


Figure 20-4. HDMA Source/Match Address

Examples of setting registers to use the match and queuing function of HDMA

1. Case 1: Bank0 → Bank1 → Bank0 → Bank1 → Bank0

- 1) Set HDSAR0, HDMA0, HDSAR1, HDMA1, HDTCR
- 2) Set HDCON with 0x1010781

(match interrupt pending enable, HDMA interrupt enable, auto load enable, alternate enable, select bank0 for current queuing bank, enable queuing enable bit 0/1, run HDMA)

2. Case 2: Bank1 → Bank0 → Bank1 → Bank0 → Bank1

- 1) Set HDSAR0, HDMA0, HDSAR1, HDMA1, HDTCR
- 2) Set HDCON with 0x1010f81

(match interrupt pending enable, HDMA interrupt enable, auto load enable, alternate enable, select bank1 for current queuing bank, enable queuing enable bit 0/1, run HDMA)

3. Case 2: Bank0 → Bank 1

- 1) Set HDSAR0, HDMA0, HDSAR1, HDMA1, HDTCR
- 2) Set HDCON with 0x1000781

(match interrupt pending enable, HDMA interrupt enable, auto load enable, alternate enable, select bank0 for current queuing bank, enable queuing enable bit 1, run HDMA)

21

IMAGE PROCESSOR

OVERVIEW

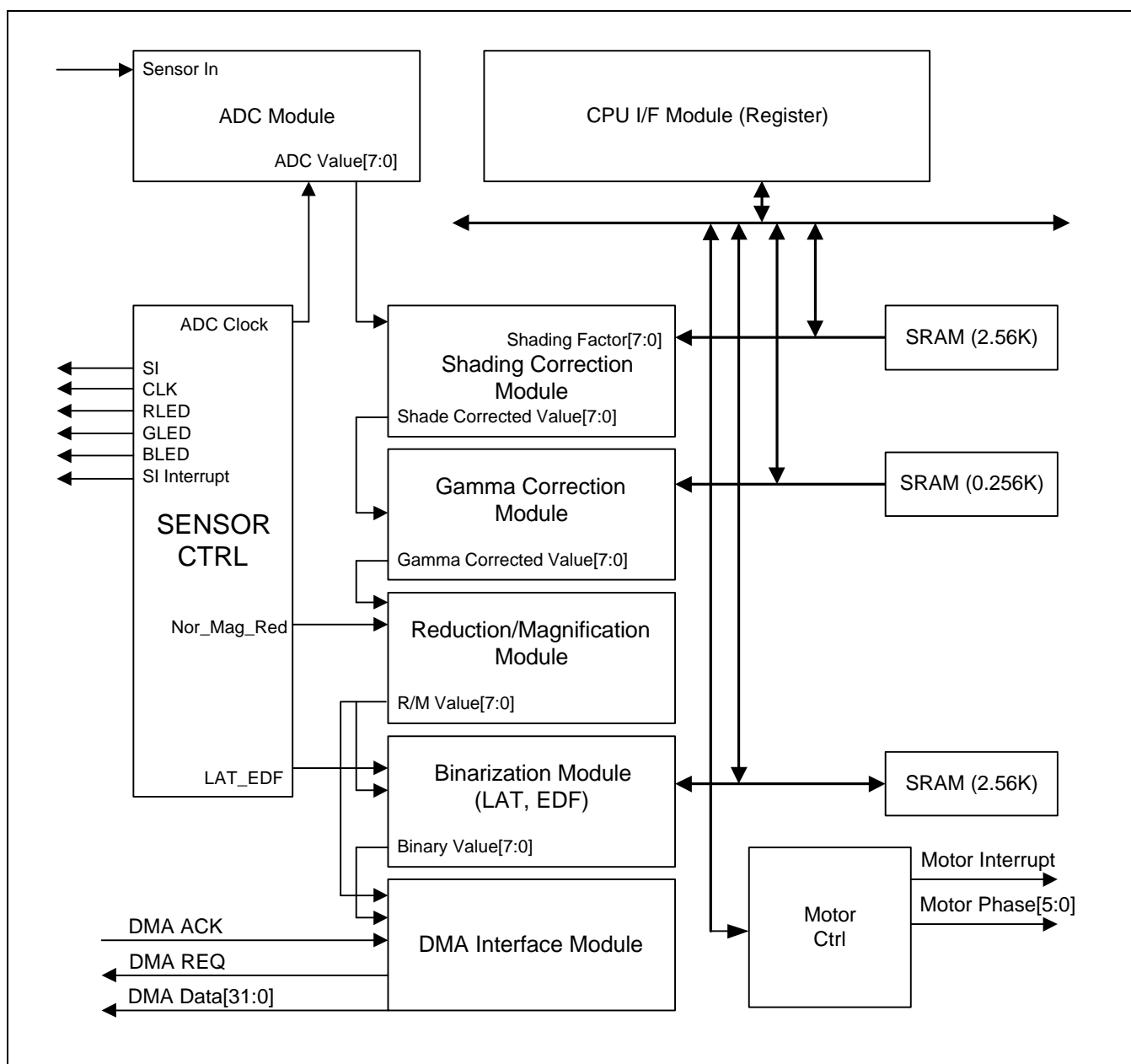


Figure 21-1. Image Processor Block Diagram

These modules adjust and convert the input data into data that can be output to the printer.

- CIS sensor control
- Digital shading correction
- GAMMA correction
- Magnification/reduction
- Photo/text mode binarization

IMAGE PROCESSOR SPECIAL REGISTERS

SENSOR SHIFT CLOCK CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
SEN_CLK	0x9800	R/W	Sensor shift clock control register	0x00818

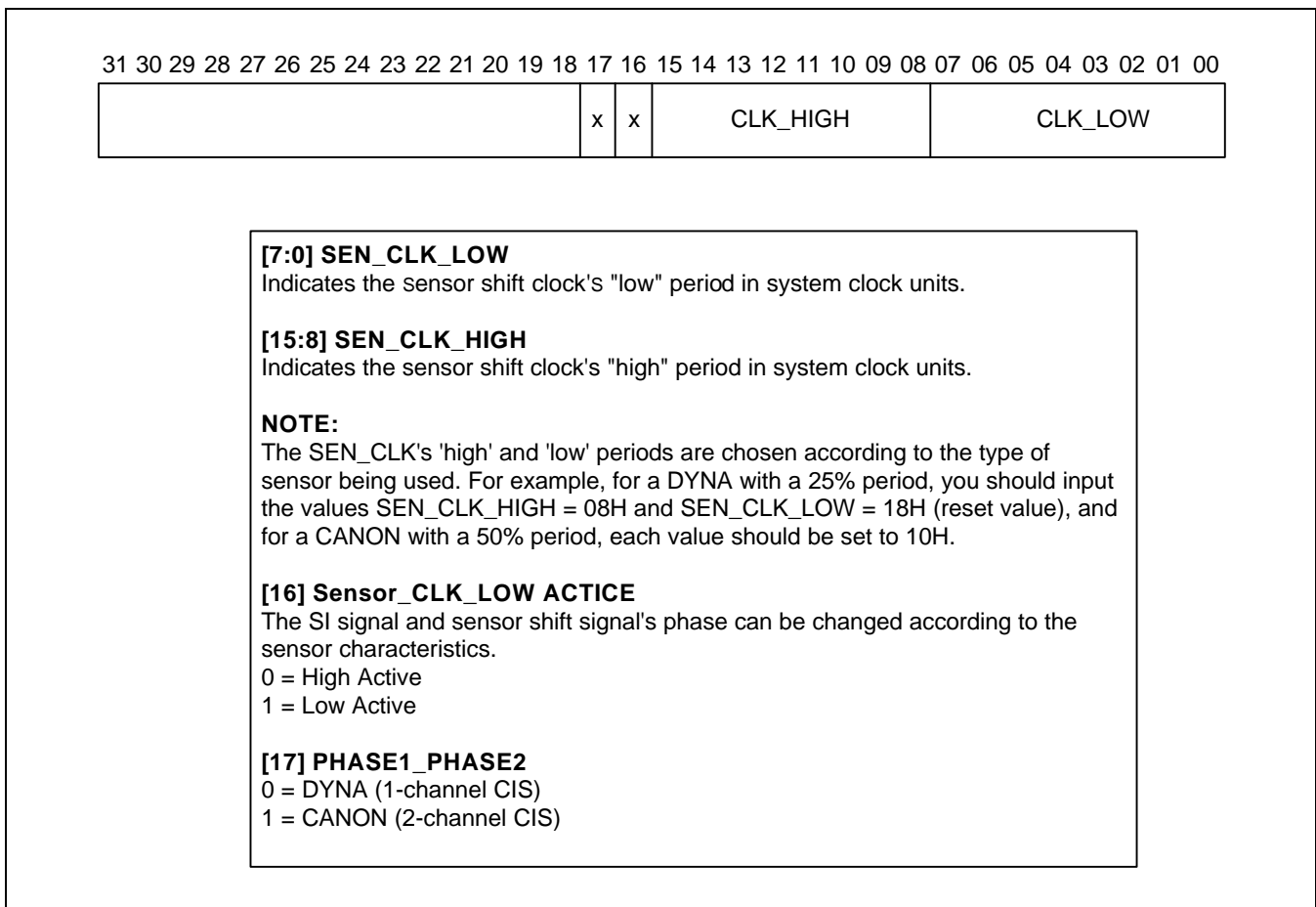


Figure 21-2. Sensor Shift Clock Control Register

SENSOR SI CLOCK CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
SI_TERM	0x9804	R/W	Sensor SI clock control register	0x09c4

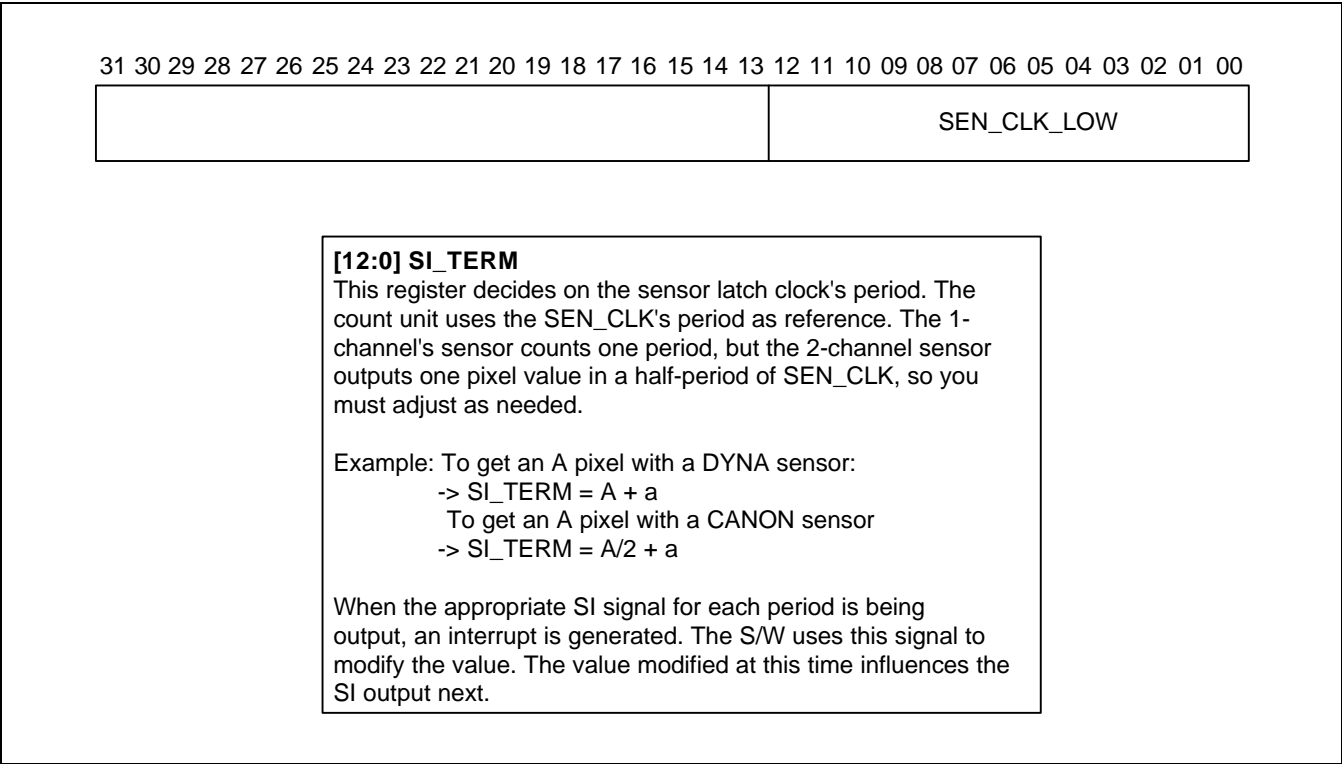
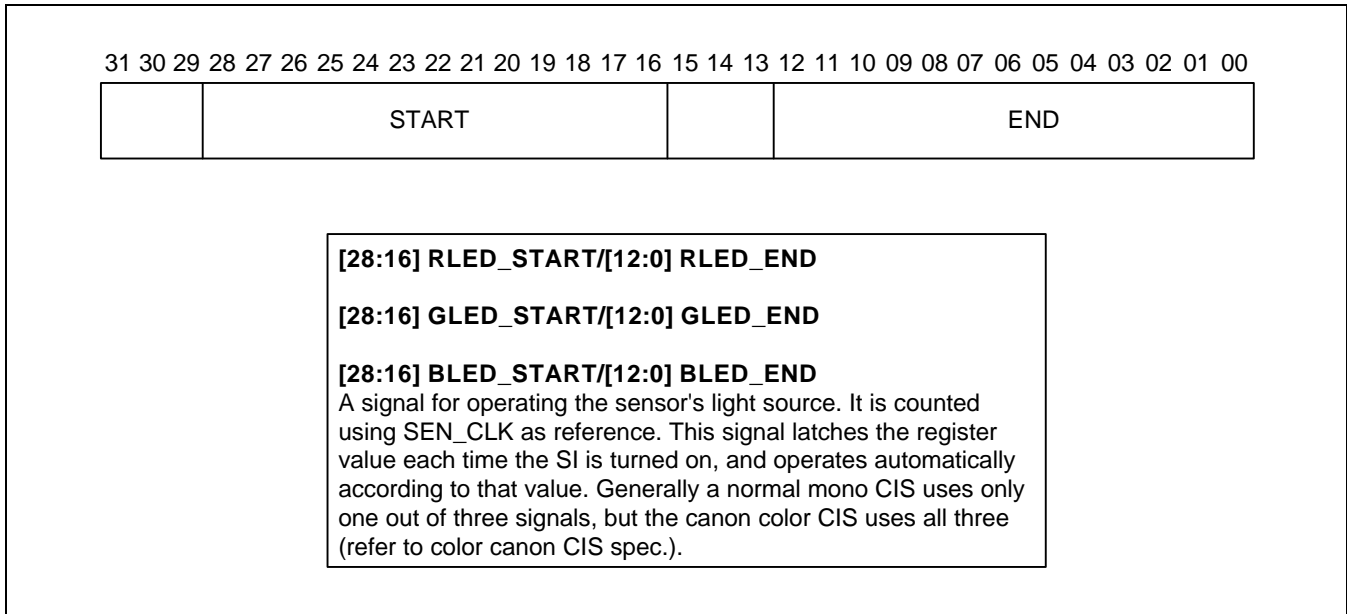


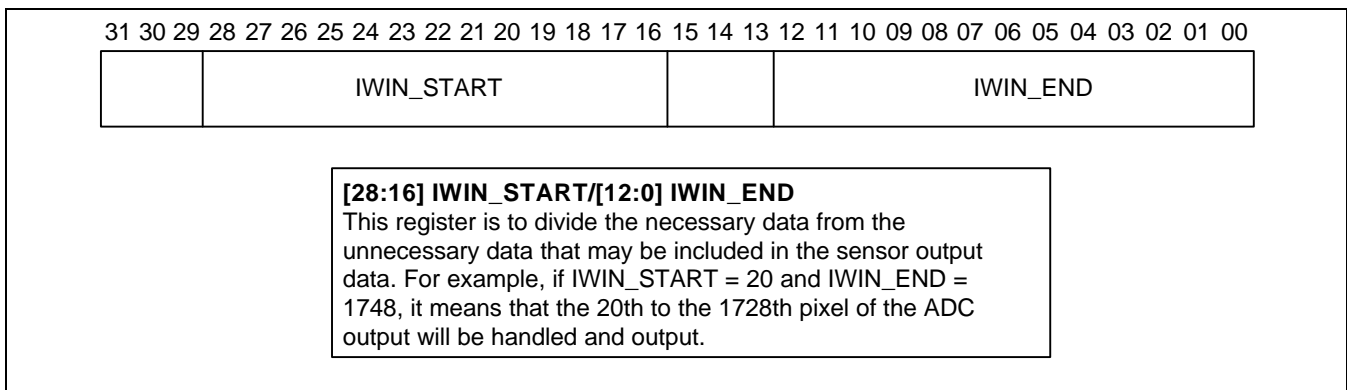
Figure 21-3. Sensor SI Clock Control Register

SENSOR R (GB) LED CONTROL REGISTER

Registers	Offset Address	R/W	Description	Reset Value
RLED	0x9808	R/W	Sensor R LED control register	0x00000960
GLED	0x980c	R/W	Sensor G LED control register	0x00000960
BLED	0x9810	R/W	Sensor B LED control register	0x00000960

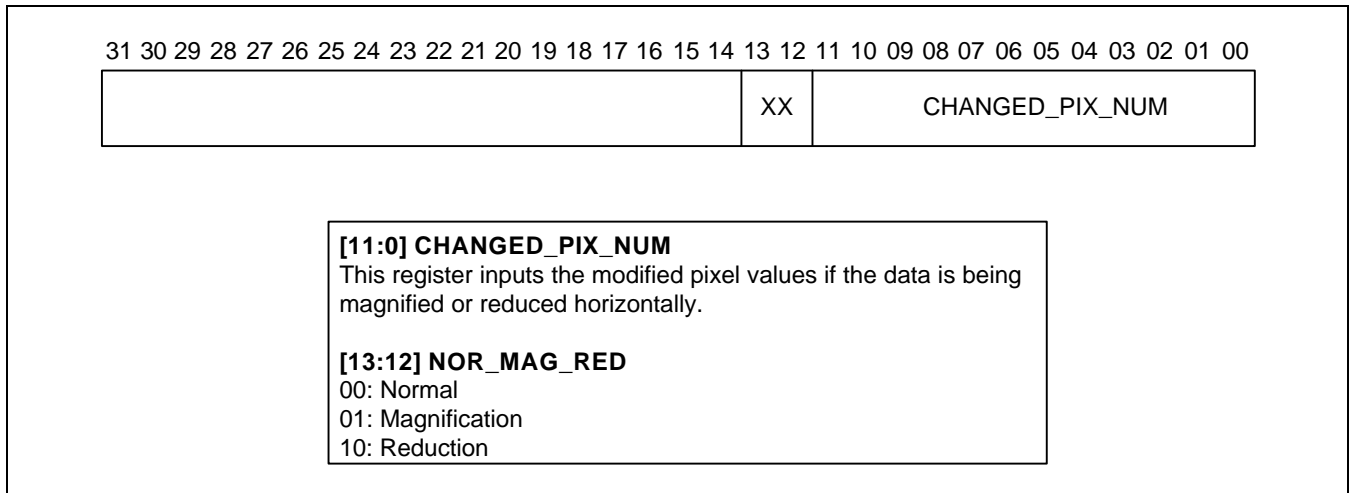
**Figure 21-4. Sensor R(GB) LED Control Register****IWIN CONTROL REGISTER**

Register	Offset Address	R/W	Description	Reset Value
IWIN	0x9814	R/W	Effective pixels num. control register	0x000006b8

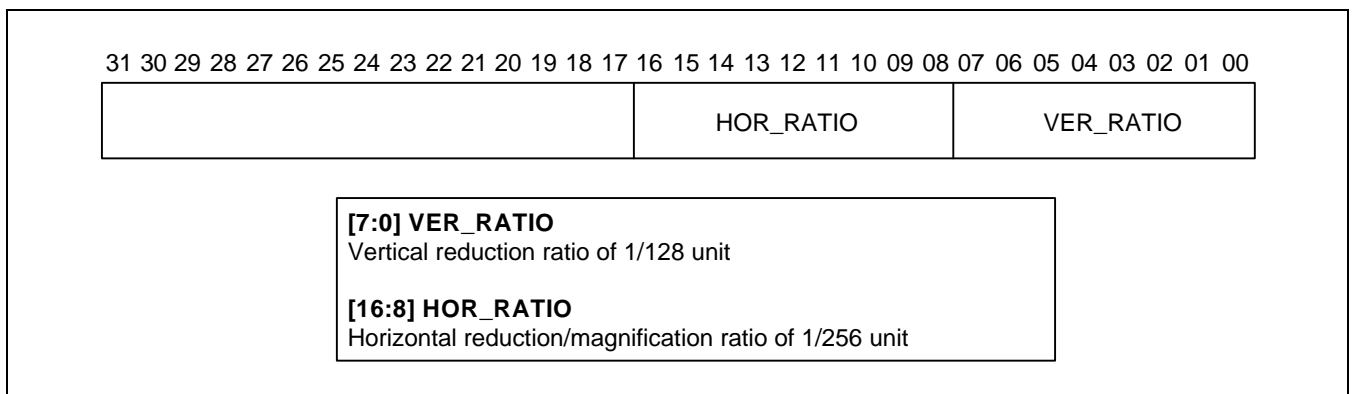
**Figure 21-5. IWIN Control Register**

CHANGED IWIN CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
CHANGED_IWIN	0x9818	R/W	Mag/Red pixels num. control register	0x06b8

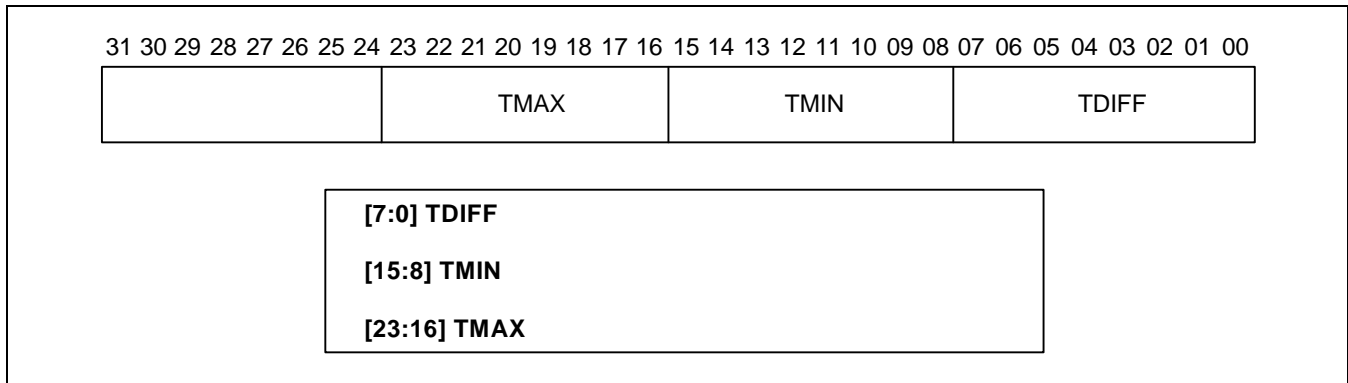
**Figure 21-6. CHANGED_IWIN Control Register****MAG/RED RATIO CONTROL REGISTER**

Register	Offset Address	R/W	Description	Reset Value
RATIO	0x981c	R/W	Mag/Red ratio control register	0x10080

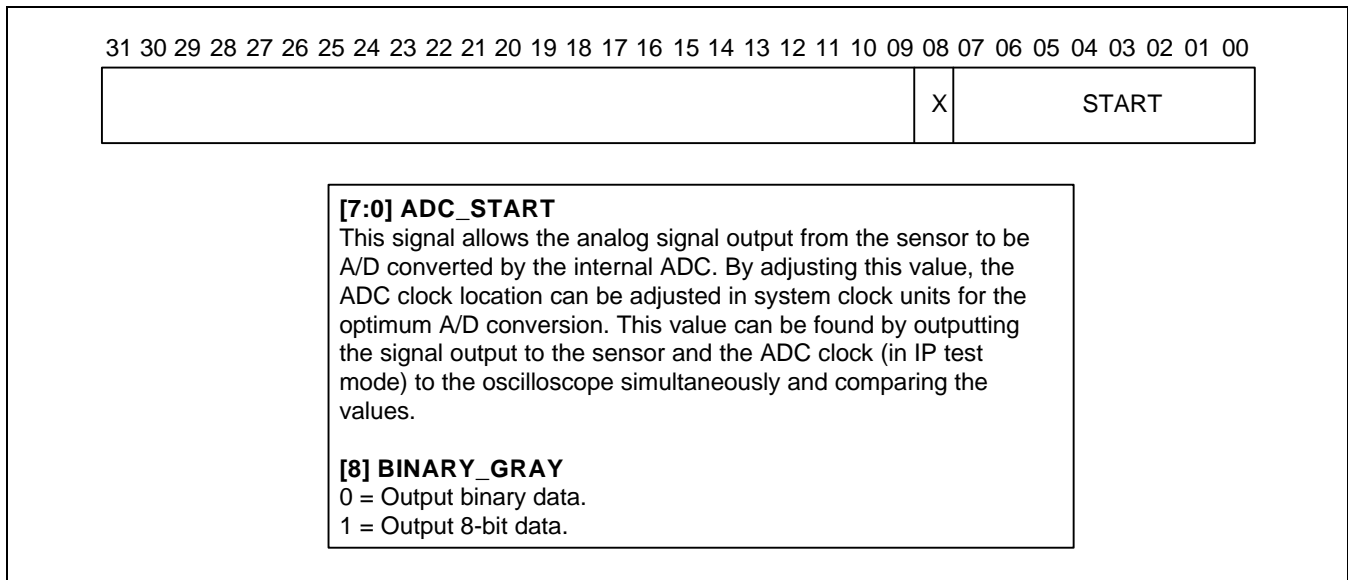
**Figure 21-7. Mag/Red Ratio Control Register**

LAT (LOCAL ADAPTIVE THRESHOLD) CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
LAT	0x9820	R/W	Local adaptive threshold control register	0xdc7f40

**Figure 21-8. LAT Control Register****ADC CONTROL REGISTER**

Register	Offset Address	R/W	Description	Reset Value
ADC	0x9824	R/W	ADC clock control register	0x005

**Figure 21-9. ADC Control Register****OPERATION CONTROL REGISTER**

Register	Offset Address	R/W	Description	Reset Value
OPERATION	0x9828	W	Operation control register	0x000

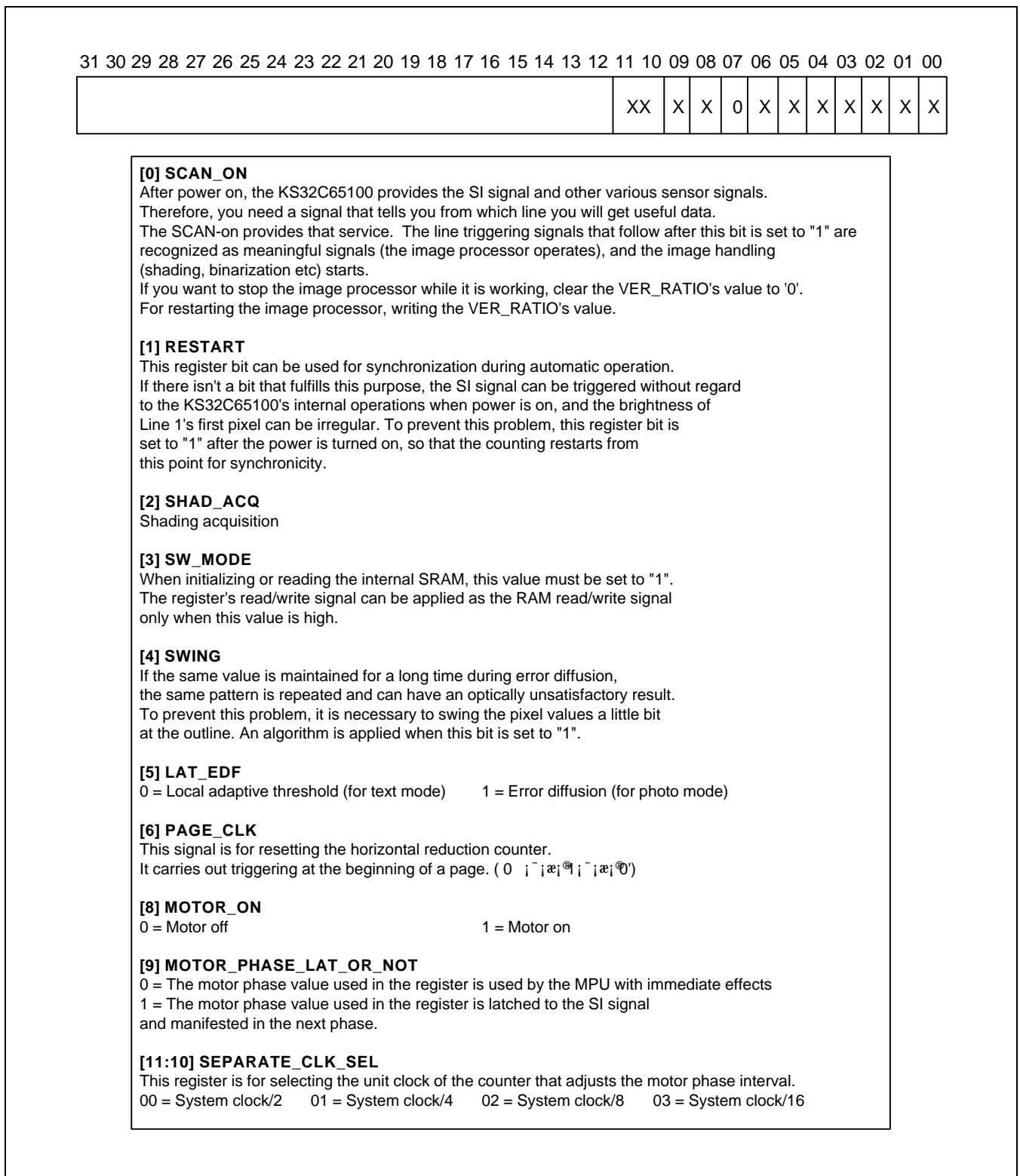
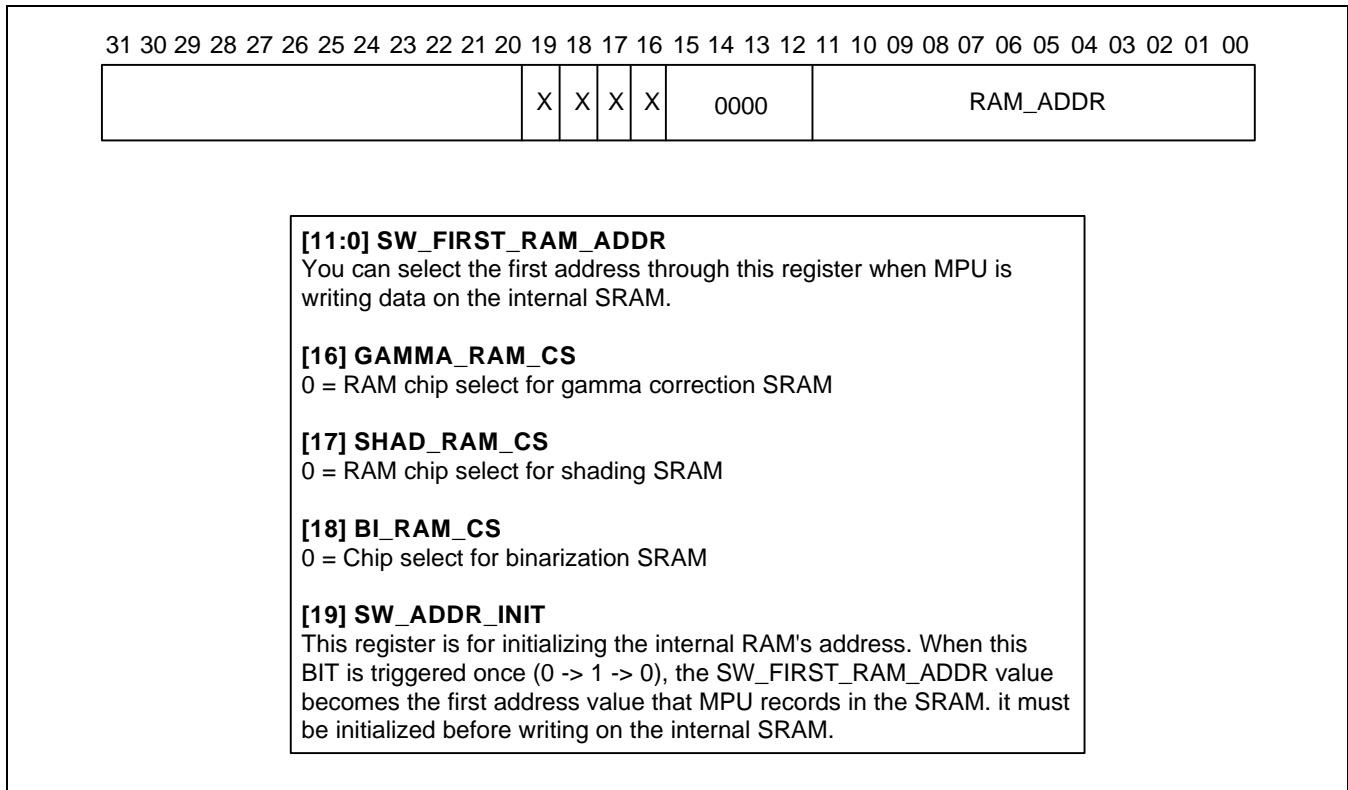


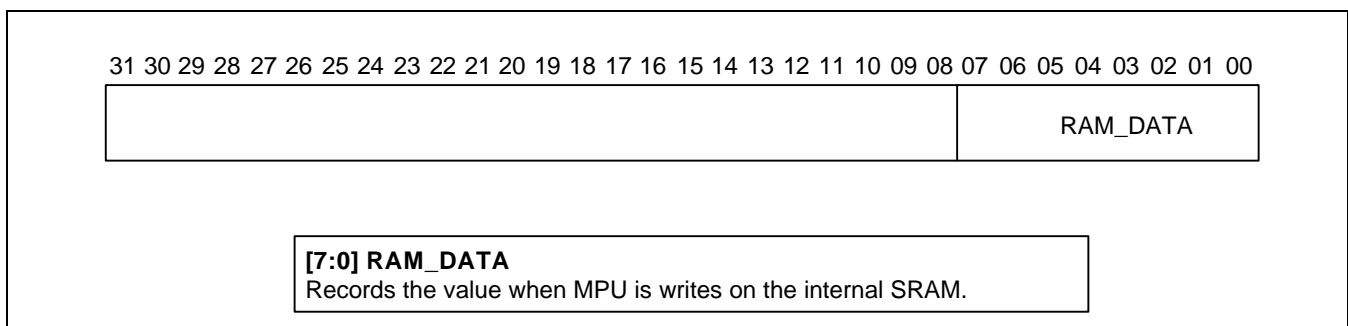
Figure 21-10. Operation Control Register

SRAM CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
RAM_CTRL	0x982c	R/W	IP Inner SRAM control register	0x70000

**Figure 21-11. SRAM Control Register****SRAM DATA REGISTER**

Register	Offset Address	R/W	Description	Reset Value
RAM_DATA	0x9830	R/W	SRAM data register	0x00

**Figure 21-12. SRAM Data Register**

MOTOR TERM CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
MOTOR_TERM	0x9834	R/W	Motor term control register	0x0000

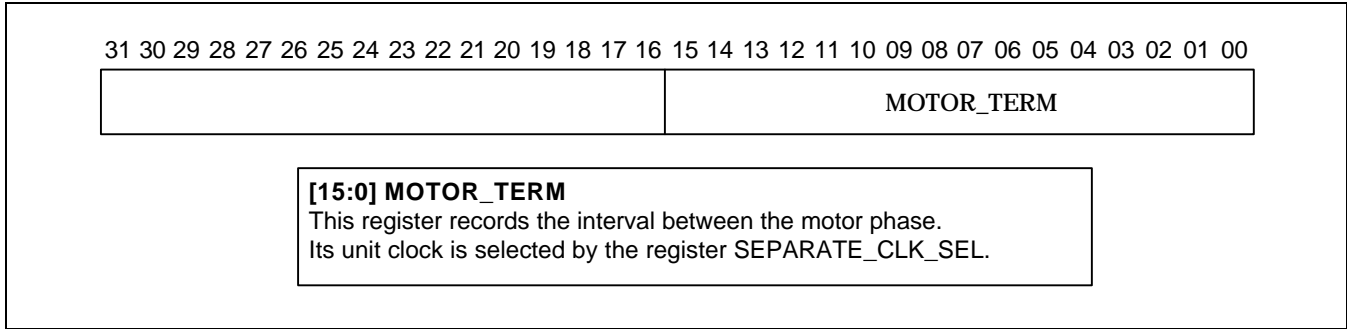


Figure 21-13. Motor Term Control Register

MOTOR PHASE CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
MOTOR_PHASE	0x9838	R/W	Motor phase control register	0x00

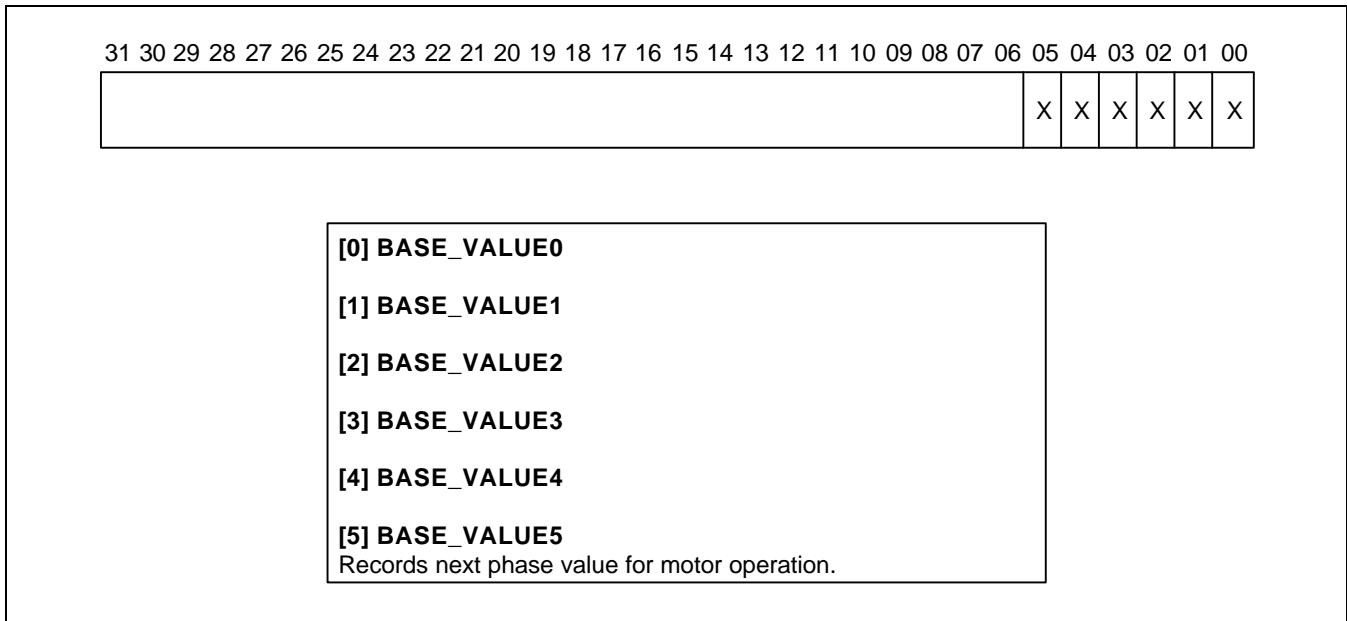
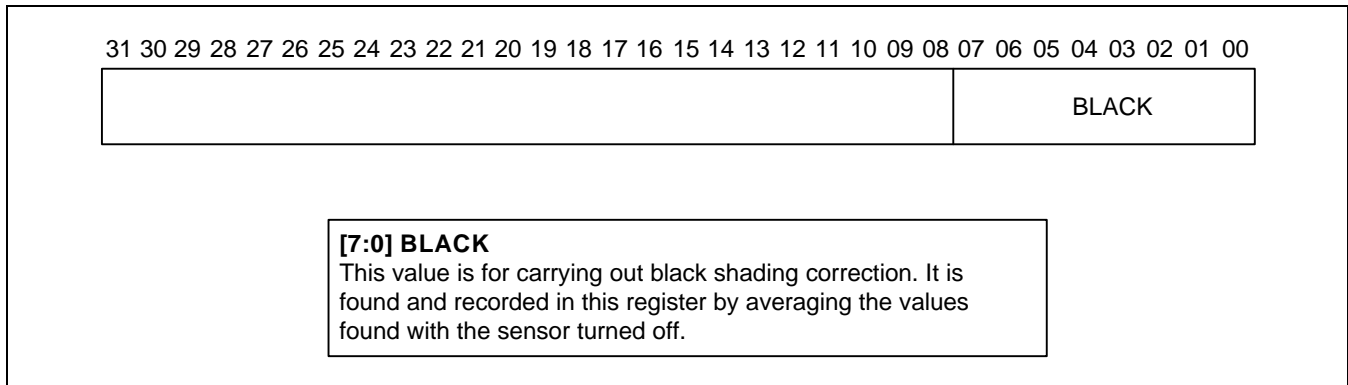


Figure 21-14. Motor Phase Control Register

BLACK SHADING CORRECTION FACTOR REGISTER

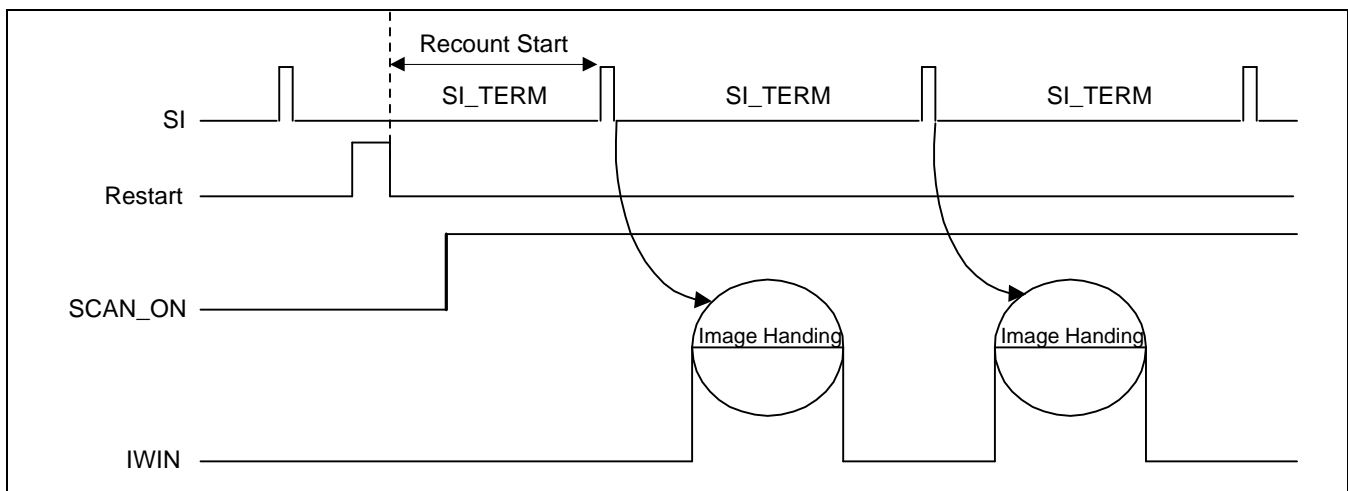
Register	Offset Address	R/W	Description	Reset Value
BLACK	0x983c	R/W	Black shading correction value register	0x00

**Figure 21-15. Block Shading Correction Factor Register****SENSOR(CIS)**

The sensor is influenced by the 300DPI Canon/DYNA'S CIS. The signal output from the controller to the sensor include the SI, SEN_CLK, RLED, GLED, and bled. The reset values for all registers related to the sensor are set with the DYNA 200DPI as reference.

SI

The SI signal is for latching the electrical charge in the sensor. The period is in SEN_CLK units according to the value of register SI[12:0]. This signal is adjusted during operation by S/W, so that it is synchronous to the motor. The SI signal is automatically operated once the power is on, but an interrupt occurs every time an SI signal is generated, so you can change the SI period if you change the value according to this signal. One thing to remember is that if you change the register SI value according to the interrupt, it influences the period of the next SI. If you give the restart signal by S/W for synchronicity, a new SI signal is generated from that signal. Shading and binarization are carried out only when the SI signal and the SCAN_ON are both "high".

**Figure 21-16. Restart & SCAN_ON Timing Diagram**

SEN_CLK

The SEN_CLK is a clock for shifting the values latched by the SI signal to the outside of the sensor. The period of the SEN_CLK varies for different brands such as DYNA and CANON. You need a 50% period for a CANON product, and a 25% period for a DYNA product. The SEN_CLK[15:8] adjusts the SEN_CLK signal's high period and the SEN_CLK[7:0] adjust the SEN_CLK's low period in system clock units. It takes at least 10 system clocks to handle an actual pixel.

NOTE

The phase of the SI and SEN-CLK signal can change according to the different brands. For example, for a 200DPI DYNA or a 300DPI CANON, the two signals are high active, but for a 300DPI DYNA, it is low active. Also, a CANON product uses the 2-channel method which reads values from SEN_CLK's both high and low level. For that, the register RATIO[17] (SEN_CLK_LOW_ACTIVE) and RATIO[18] (PHASE1_PHASE2) are prepared.

RLED, GLED, BLED

This signal is for controlling the sensor's brightness. The lamp's operational range is decided using the values of registers RLED, GLED, And BLED. The SEN_CLK period as the unit reference. Generally, only one of three signals is needed for normal mono CIS, but the canon color CIS uses all three signals.

REDUCTION & MAGNIFICATION

This feature uses the register value to magnify (up-to 200%) or reduce horizontally to a unit less than 1%, and to reduce vertically.

Reduction: Horizontal

A 9-bit register can be used for reduction in 256 steps. Reduction can't take place if the register value is 9 'H100', and for any value below it, reduction is carried out according to the formula given below. This feature is used when sending an image to a 203DPI-level fax when using a 300DPI-level sensor. This feature uses sampling by an adder. You only need to set the reduction bit, and you can adjust in units less than 1%.

$$\text{Register Value} = \text{Reduction Ratio} \times 256$$

To find the number of reduced pixels compared to the input pixels, use the following formula:

$$\frac{\text{Register Value}}{256} \times \text{Number of input pixels} = \text{number of reduced pixels}$$

(Except, if the value behind the decimal point of the formula's result is larger than

$$\frac{\text{Register Value}}{256} \text{ add 1 to the result.}$$

If not, the value found is the actual number of reduced pixels.)

For example, when reducing a 300DPI image to 203DPI, the register setting value is

$$\text{Register Value} = \frac{203}{300} \times 256 = 137.23$$

so you should set 137 to the register. The number of reduced pixels can be found in the following manner.

$$\frac{173}{256} \times 2557 = 1727.9 \text{ in } 0.97 \text{ is larger than } \frac{173}{256} = 0.67 \text{ so the number of pixels actually reduced is } 1728.$$

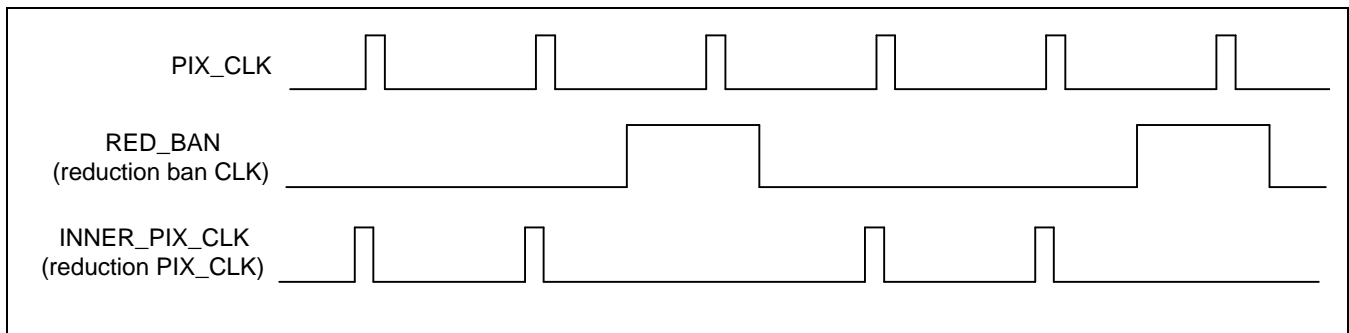


Figure 21-17. Reduction Pixel Clock Timing Diagram

Reduction: Vertical

The feature described above allows vertical reduction in units of 1/128 using a 3-bit register. The basic algorithm is the same as that in the horizontal direction, but the unit of reduction is 8 bits, and the reduction ratio is not as dense. During reduction, all signals transmitted to the sensor and ADC are generated, but the data isn't accepted. You need to use the reduction ratio register for vertical reduction. You can find the register value using the following formula.

$$\text{Register Value} = \text{Reduction Ratio} \times 128$$

The register's initialization value is 8 'H80', and the value signifies a mode that does not carry out reduction.

Magnification

The design of this product is such that the reduction feature described above is also used for image magnification. This feature is to magnify and print a 300DPI image using a 200DPI-level sensor. To do so, first increase the pixel clock (PIX_CLK) speed to twice the original. Since the internal PIX_CLK (INNER_PIX_CLK) is made using the reduction feature, you can have a maximum velocity (MSLT) of approximately 2ms for 33MHz in magnification. In other words, you are guaranteeing the time for one PIX_CLK to be generated between the PIX_CLKs, and allowing a new PIX_CLK to be generated there according to the register value. The timing diagram is given below.

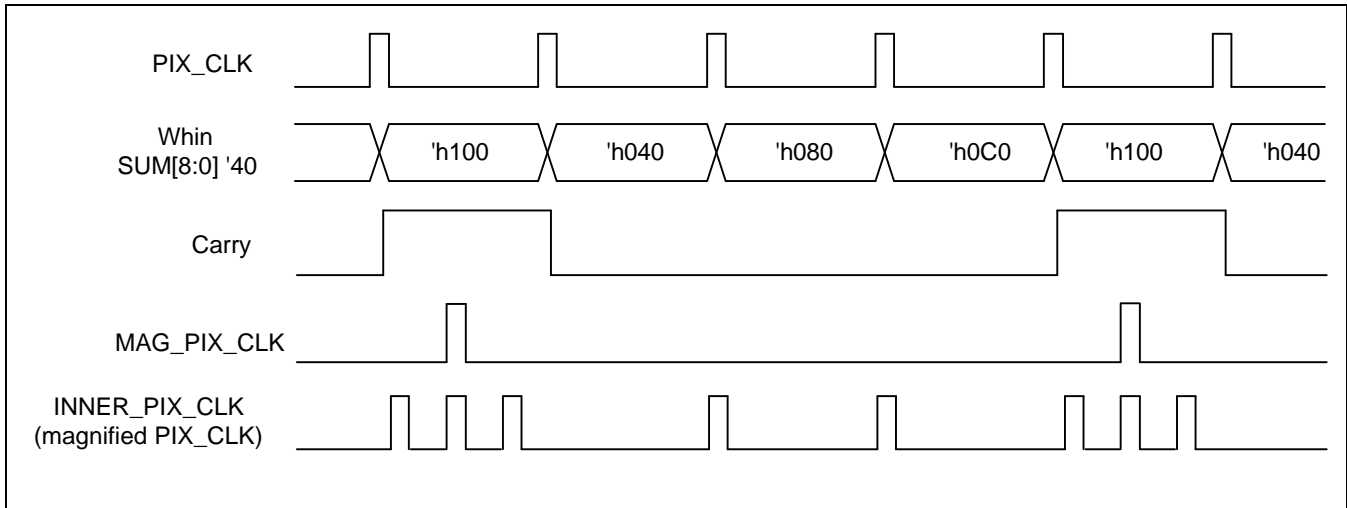


Figure 21-18. Magnification Pixel Clock Timing Diagram

To carry out magnification, you must select the magnification bit (NOR_MAG_RED==1), and record the actual number of pixels calculated using the magnification ratio. For a 125% magnification, the magnification ratio is 0.25, and the register value is found in the following manner.

Register Value = Magnification Ratio × 256

To find the number of magnified pixels, use the following formula.

$$\frac{\text{Register Value}}{256} \times \text{Total number of line input pixels} = \text{number of added pixels}$$

(Except, if the value behind the decimal point of the formula's result is truncated)

The sum of this value and the number of originally input pixels give the number of actually magnified pixels. For example, for a 137% magnification, the magnification ratio register setting value (for 1720 pixels/line) is Magnification Register Value = 0.37 × 256 = 94.72,

so you can set 94 to the register. The number of magnified pixels is the sum of the original value 1720 and the number of added pixels, 631.

Since in $\frac{94}{256} \times 1720 = 631.56$ (0.56 is truncated), the number of added pixels from the magnification is 631.

DIGITAL SHADING CORRECTION

This feature is for adjusting the sensor's non-uniform illumination characteristics using the 16×8 divider. In this controller, the ratio between the value with 8-bit steps and the actual value of white is found, and the ratio is used for conversion during scanning. For example, if we call the value from white pad W, the value from black pad B, and the actual value from scanning X, the digital shading corrected value (Y) can be found by the following formula.

$$Y = \frac{X - B}{W - B} \times 256$$

Here, the process of storing the value that will read the white pad (W) in memory is called white shading acquisition. If you set the shading acquisition bit to "1" and scanning process is performed, the value is immediately stored in the shading memory. If you go into the scanning process after this step, the shading corrected value for the actual value is generated using W and B as reference.

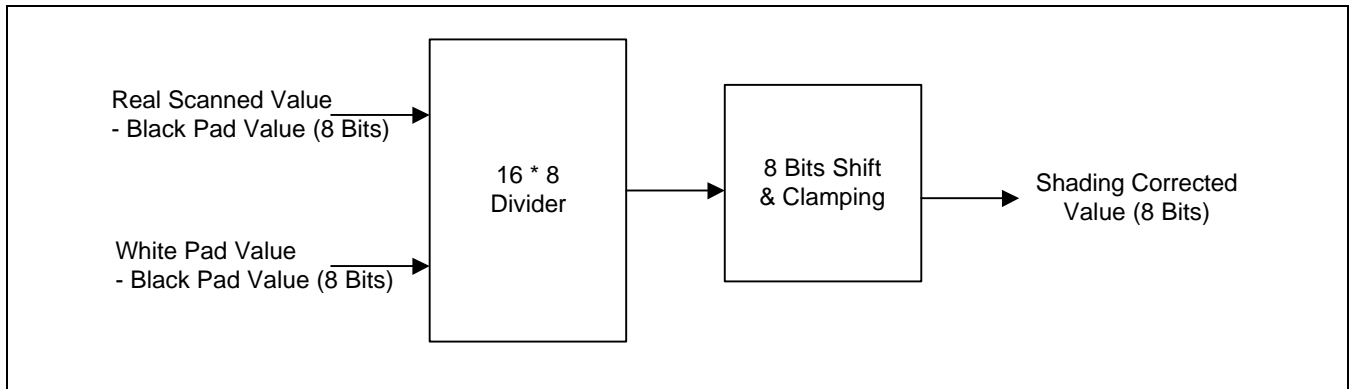


Figure 21-19. Shading Correction Block Diagram

GAMMA CORRECTION

This process uses a 256×8-bit SRAM to carry out gamma correction of the RGB value that was shading corrected in the previous step.

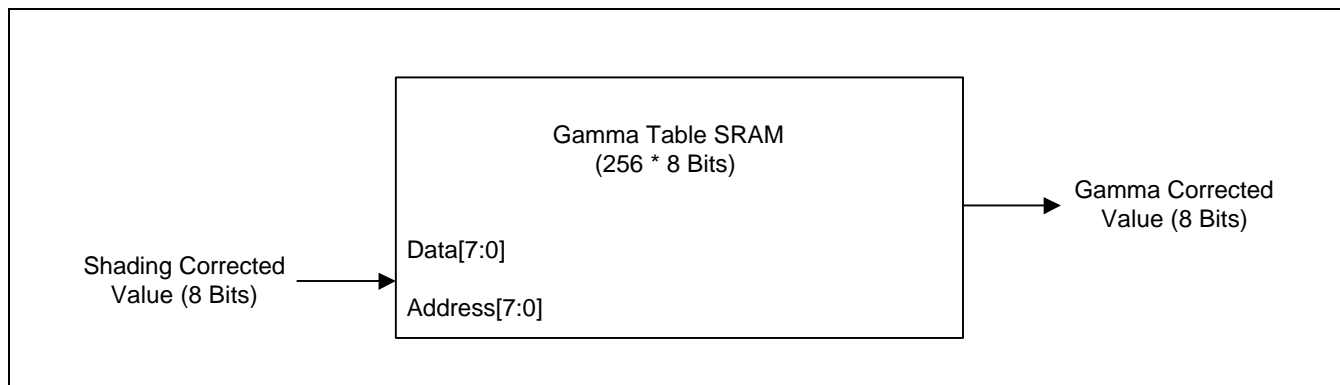


Figure 21- 20. Gamma Correction Block Diagram

BINARIZATION

Error Diffusion

This process is for binarization in the image mode. The algorithm used is in the FLOYD method, and uses the mask given below.

1(UC)	5(U)	3(UR)
7(L)	Input Pixel(C)	-

- $C' = C + [(1/16) \times \text{Error}(UL)] + [(5/16) \times \text{Error}(U)]$
 $+ [(3/16) \times \text{Error}(UR)] + [(7/16) \times \text{Error}(L)]$
- Binary output decision and error calculation.
 IF $C' \geq 128$, OUT White(0), AND $\text{Error}(C') = C' - 255$
 ELSE, OUT Black(1), AND $\text{Error}(C') = C'$

This method is advantageous from the aspect of H/W, but if a certain brightness is maintained on the algorithm, an optically displeasing stripe may be generated on the screen. To compensate for this problem, you can swing the outline values a little.

Local Adaptive Threshold

This method is for binarization in text mode. You don't need to use an edge-emphasis algorithm when using this method, and you can expect the ABC effect. You need to select the values for the entire area as shown below.

- Tmax: decides if the pixels will have absolute white value
- Tmin: decides if the pixels will have absolute black value
- Tdiff: decides if the pixels have edge components

The algorithm using the above values is as follows.

- ϕ MIN/MAX Decision
 - Decide MIN/MAX gray value of 2x3 matrix
 - Calculate the average of min/max value($AVE = [MIN + MAX]/2$)
- δ Edge Pixel Decision
 - Pixel which is larger than Tdiff is edge pixel \Rightarrow EXIT
- δ n Case of Edge Pixel
 - Edge pixel which is larger than AVE is white \Rightarrow EXIT
 - Else is black \Rightarrow EXIT
- δ n Case of Non Edge Point
 - Pixel which is larger than Tmin is white \Rightarrow EXIT
 - Else is black \Rightarrow EXIT

ADC CONTROL

The ADC signal is for operating the internal ADC, and the signal must always maintain a 50% period. For A/D conversion of the analog signal, you need a register that adjusts the ADC signal's starting point in units of system clock for the register value. The figure below shows the ADC controlling diagram for a product using the 2-channel method, such as CANON.

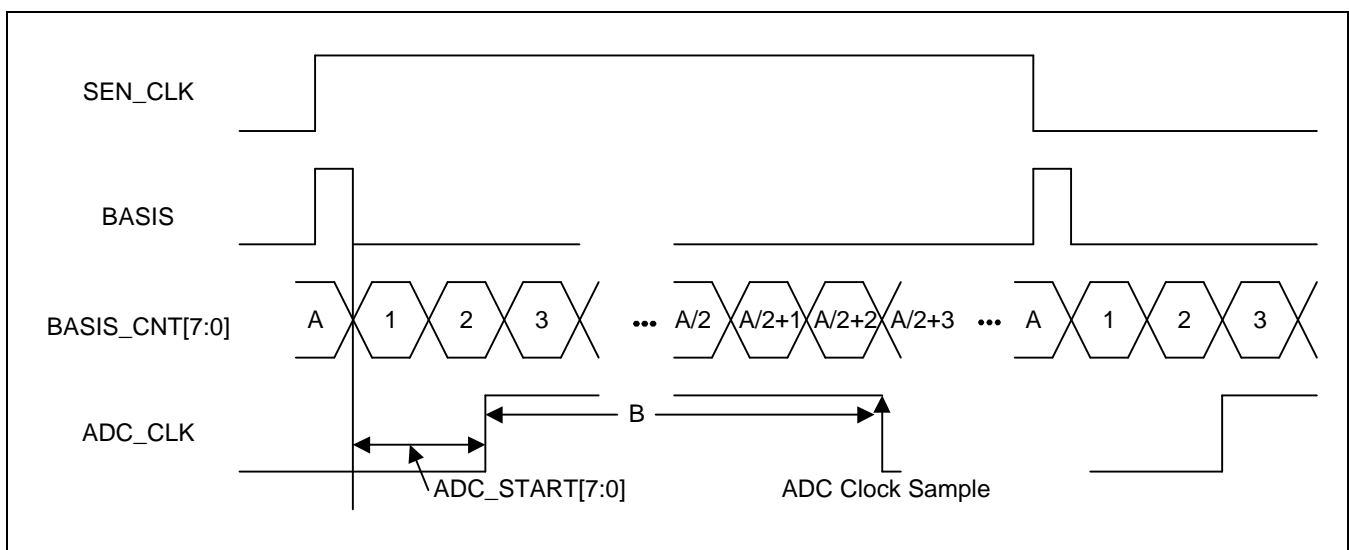


Figure 21-21. ADC Control Timing Diagram (CANON)

As shown in the figure above, the range of the ADC-STAR[7:0] value that adjusts the ADC signal location becomes 1/2 of the SEN_CLK's LOW period from 1, and the ADC signal's high period (B) becomes $(SEN_CLK_LOW)/2$.

For a product using the 1-channel method such as DYNA CIS, refer to the diagram below.

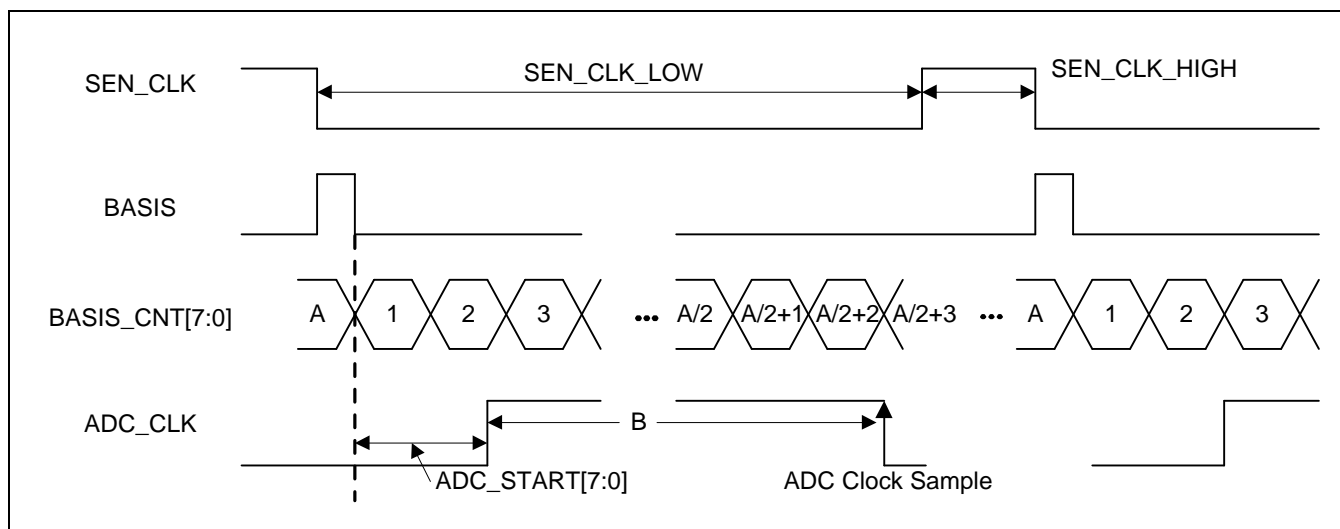


Figure 21-22. ADC Control Timing Diagram (DYNA)

In this case, you need a 25% DUTY, so the SEN_CLK's HIGH period and low period are added to make the ADC signal's high period. Therefore, the width of B in the figure above is $(\text{SEN_CLK_LOW} + \text{SEN_CLK_HIGH})/2$.

MOTOR CONTROL

There are 6 port phase outputs for motor control. Each output signal can be adjusted by S/W. In other words, the register MOTOR_TERM[15:0]'s value can change the interval value for the changing motor phase. An interrupt occurs every time a signal signifying the interval is generated, so the S/W changes the register value using that signal, which later influences the next motor phase interval. This feature operates as described above if the bit of register OPERATION[9] (MOTOR_PHASE_LAT_OR_NOT) is set.

If the bit is reset, the value immediately influences the operation. You can synchronize the motor and SI using this feature and the SI period-adjusting feature. When the value of register OPERAION[8] (MOTOR_HIGH) becomes high, down counting is carried out from the MOTOR_TERM value. When the value reaches 1, the MOTOR_PHASE[5:0] value used by the previous interrupt is output to each phase. The reference counting clock is the one selected by the OPERATION[11:10](SEPARATE_CLK_SEL) value.

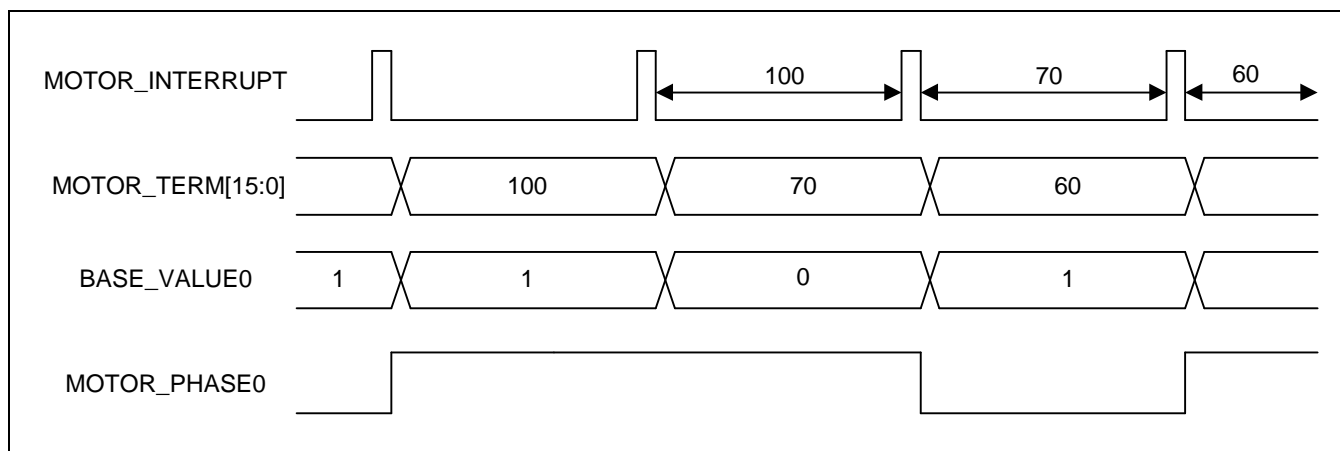


Figure 21- 23. Motor Interrupt/Phase Timing Diagram

REGISTER READ/WRITE

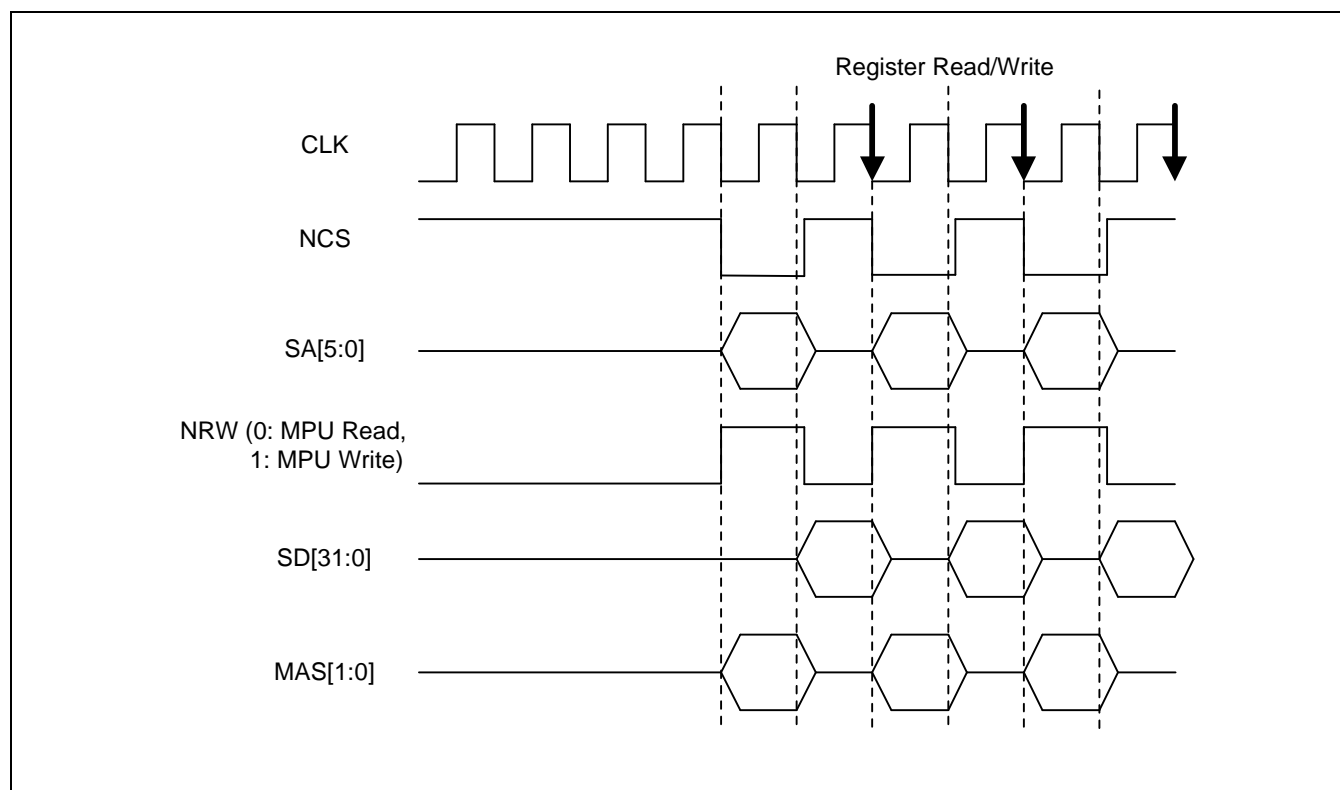


Figure 21-24. Register Read/Write Timing Diagram

Register Read/Write

When you want to write the register value from the MPU to the IP, you receive the input as shown in Figure 21-24. When writing or reading the value to the register in the sections shown in the diagram, the register read/write is carried out within 2 cycles.

RAM Initialization by Register Read/Write

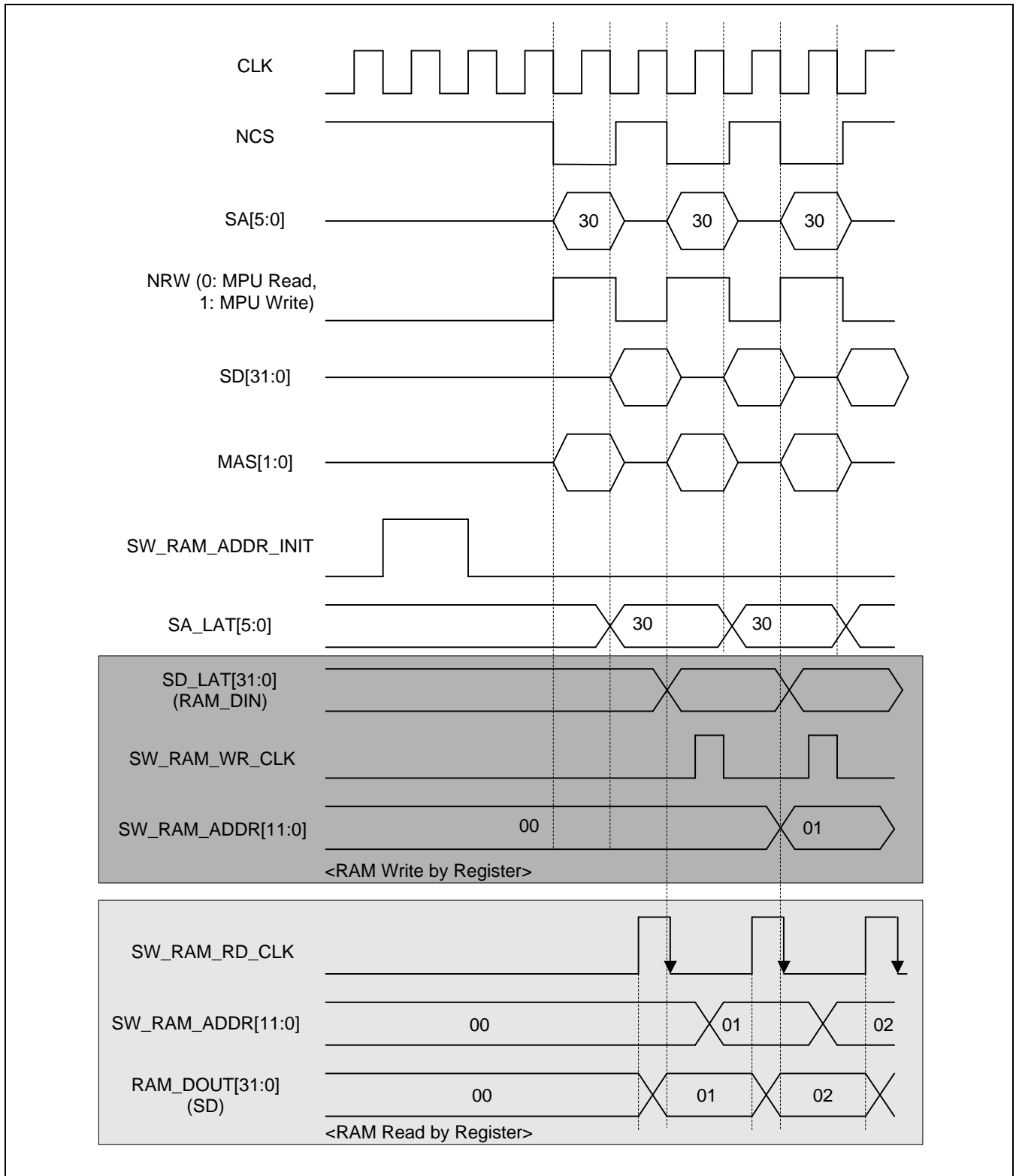


Figure 21-25. Timing Diagram SRAM Read/Write by Register

The CONTROLLER has the following features so that it can read or write on the RAM using the register. There is the RAM_CTRL[14:12]'s CHIP SELECT for selecting the internal SRAM. The SRAM's first ADDRESS is received through the register. This value is loaded using the value of the register called SW_ADDR_INIT. After setting the environment, if you record the value to the register RAM_DATA, it is read or written to the SRAM. The RAM ADDRESS increases by 1 for each READ/WRITE from the original value from the register. This process can be summarized as follows.

- Record first address of the SRAM area for read/write, and choose the SRAM (RAM_CS).
- Initialize the SRAM address. (Register SW_RAM_ADDR_INIT)
- Record value on the register RAM_DATA during write
- Read using the register RAM_DATA's address value during read.

NOTE

When you read the register value, the internal ram must not be selected. Of course, when you read the internal ram values, the internal ram has to be selected.

DMA OUTPUT

Binary Data Output

The binary data output through the IP goes through the 32-bit cycle stealing DMA process. In other words, if the binary data for 32 pixels is output, one request is output. And if each line's last DMA is not 32 bits, the remaining parts are filled with "0" for output. To signify the location of the last pixel, you must set the value of the register CHANGED_PIX_NUM. If you do not magnify or reduce the image, you can use the number of value pixels/line for the last value, but if you do magnify or reduce, you must use a modified value.

NOTE

If the number of magnified pixels exceeds 2560 during magnification, the DMA operates fixed to 2560, regardless of the CHANGED_PIX_NUM value.

GRAY Data Output

If you set the BINARY_GRAY register bit to "1", the gamma corrected value (8 bits) is immediately output through DMA. Since the DMA must maintain 32 bits, DMA request is output once for each time 4 pixels are handled, so you must select the CHANGED_PIX_NUM accordingly.

22

REAL TIME CLOCK

OVERVIEW

The Real Time Clock (RTC) unit is operated by the system power (+5V) or the backup battery if the system power is turned off. The RTC transmits 8-bit data to the CPU as BCD (Binary-Coded Decimal) values using STRB/LDRB ARM operation. The data include second, minute, hour, date, day, month, and year. The RTC unit works with an external 32.768kHz crystal.

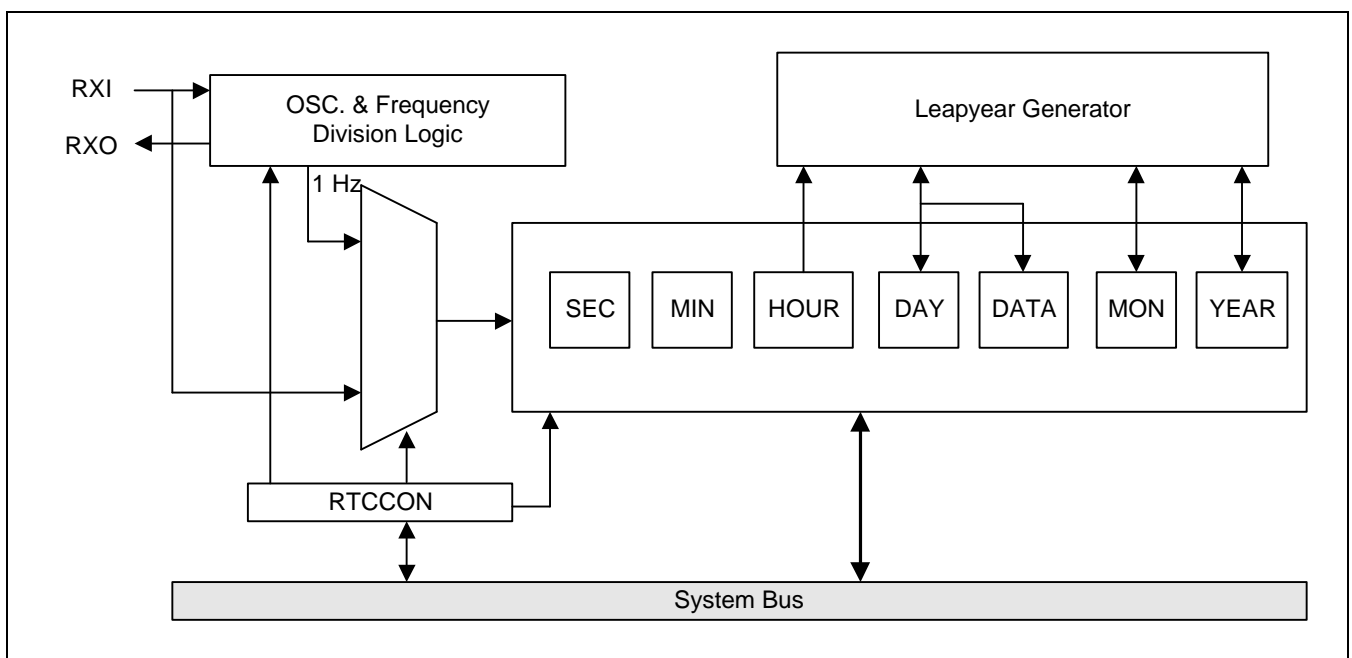


Figure 22-1. Real Time Clock Block Diagram

LEAP YEAR GENERATOR

This generator calculates if the last date of each month is 28, 29, 30 or 31 based on data from BCDDAY, BCDMON and BCDYEAR. It also considers leap years in deciding the last date.

An 8-bit counter can just represent 2 BCD digits, so it cannot decide whether 00 year is a leap year or not. We know year 2000 is a leap year, therefore the leap year generator is hard-wired to work up to 2-29-00.

SYSTEM POWER OPERATION (+5V)

It is required to set bit 0 of the RTCCON register for interfacing between CPU and RTC logic. A 1 second error can occur when the CPU reads or writes data into BCD counters, and this can cause the change of the higher time units. When the CPU reads/writes data to/from the BCD counters, another time unit may be changed if BCDSEC register is overflowed. To avoid this problem, the CPU should reset the BCDSEC register to 00h. The reading sequence of the BCD counters is BCDYEAR, BCDMON, BCDDATE, BCDDAY, BCDHOUR, BCDMIN and BCDSEC. It is required to read it again from BCDYEAR to BCDSEC if BCDSEC is zero.

BACKUP BATTERY OPERATION

The RTC logic is driven by a backup battery if the system power is off. The interfaces of the CPU and RTC logic are blocked and the backup battery only drives the oscillation circuit and the BCD counters, to minimize power dissipation.

REAL TIME CLOCK REGISTERS

RTCCON REGISTER

The RTCCON register is comprised of RTCE (RTC Enable: bit 0) which controls the write-disable of the BCD registers, RCLK (RTC Clock: bit1), CNTSEL (Counter Select: bit 2), and CLKRST (Clock Reset: bit 3) for testing.

Bit RTCE controls all interfaces between the CPU and the RTC, so it should be set to 1 in an initialization routine to enable data transfer after a system reset. Instead of working BCD with 1Hz, bit RCLK enables the operation of BCD counters with an external clock which is entered through the pin RXI to the test BCD counters. Bit CNTSEL converts the dependent operation of BCD counters into independent counters for testing. CLKRST resets the frequency divided-logic in the RTC unit.

Register	Offset Address	R/W	Description	Reset Value
RTCCON	0xc840	R/W	RTC control register	0x0

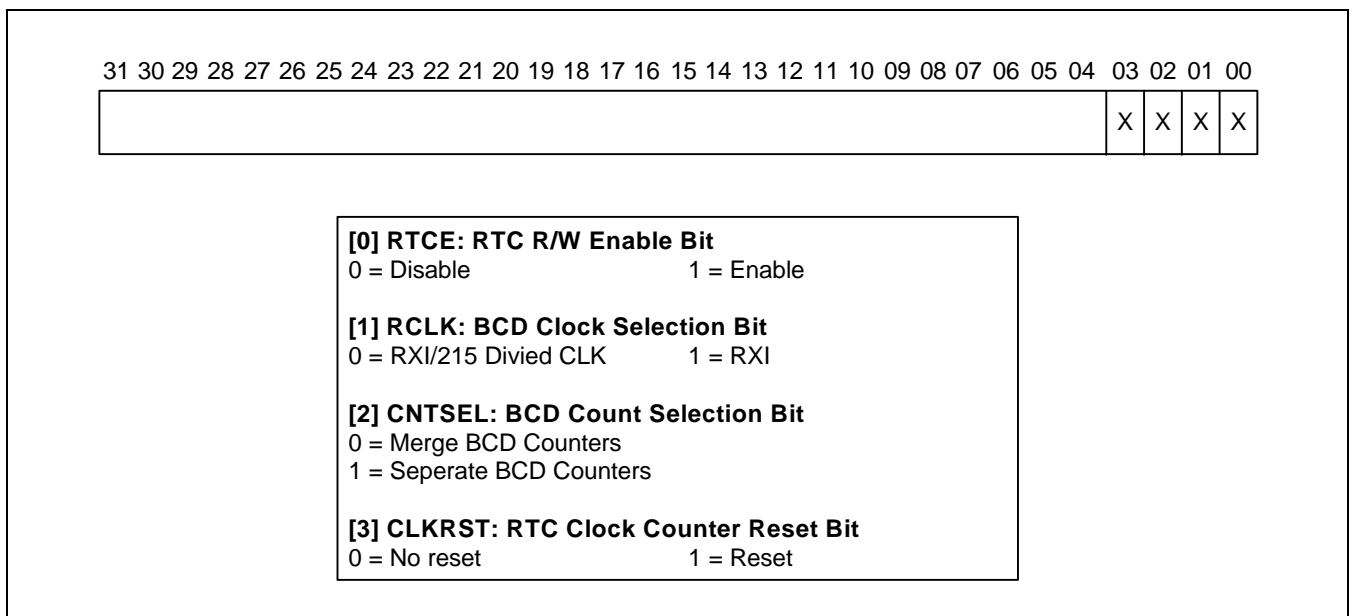


Figure 22-2. RTCCON Register

BCDSEC COUNTER REGISTER

BCD count register for seconds.

Register	Offset Address	R/W	Description	Reset Value
BCDSEC	0xc870	R/W	RTC second register	0xXX

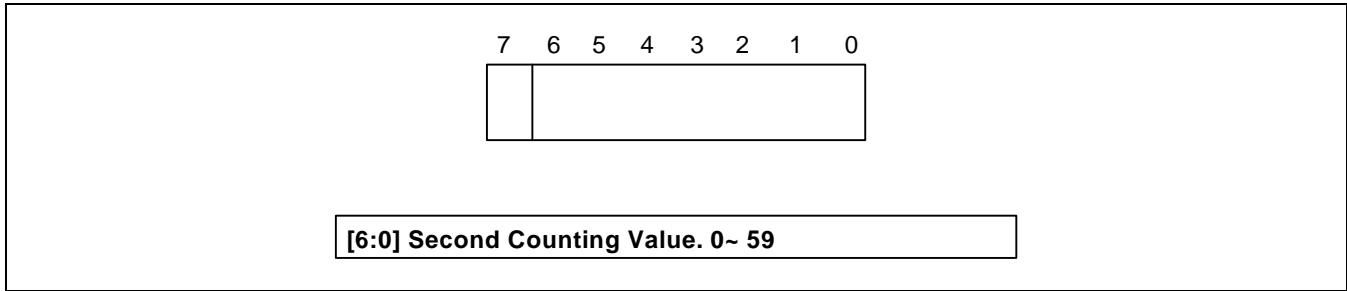


Figure 22-3. BCDSEC Counter Register

BCDMIN COUNTER REGISTER

BCD count register for minutes.

Register	Offset Address	R/W	Description	Reset Value
BCDMIN	0xc874	R/W	RTC minute register	0xXX

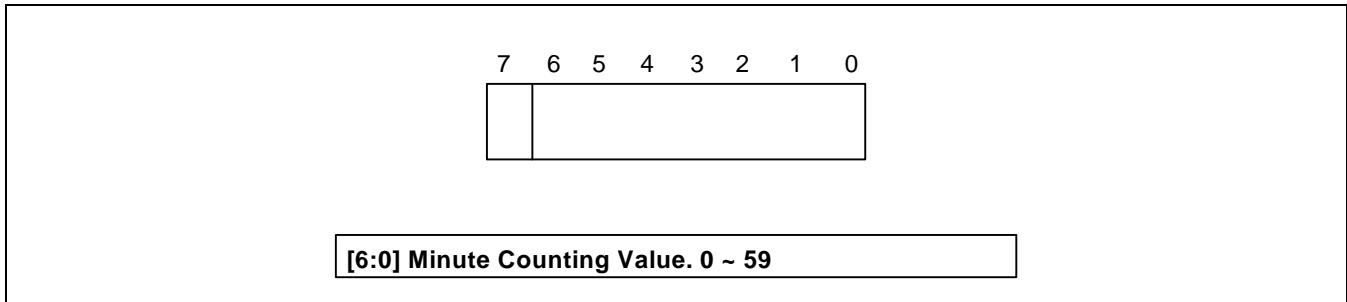


Figure 22-4. BCDMIN Counter Register

BCD HOUR COUNTER REGISTER

BCD count register for hours.

Register	Offset Address	R/W	Description	Reset Value
BCD HOUR	0xc878	R/W	RTC hour register	0xXX

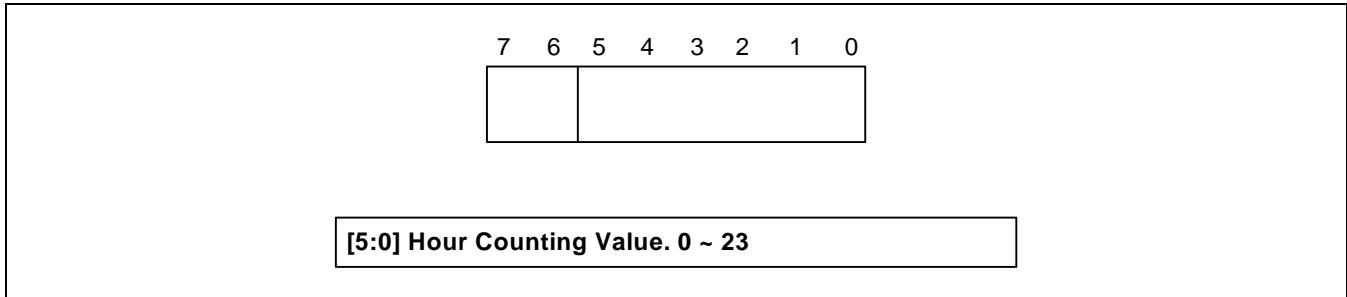


Figure 22-5. BCD HOUR Counter Register

BCDDAY COUNTER REGISTER

BCD count register for days.

Register	Offset Address	R/W	Description	Reset Value
BCDDAY	0xc87c	R/W	RTC day register	0xXX

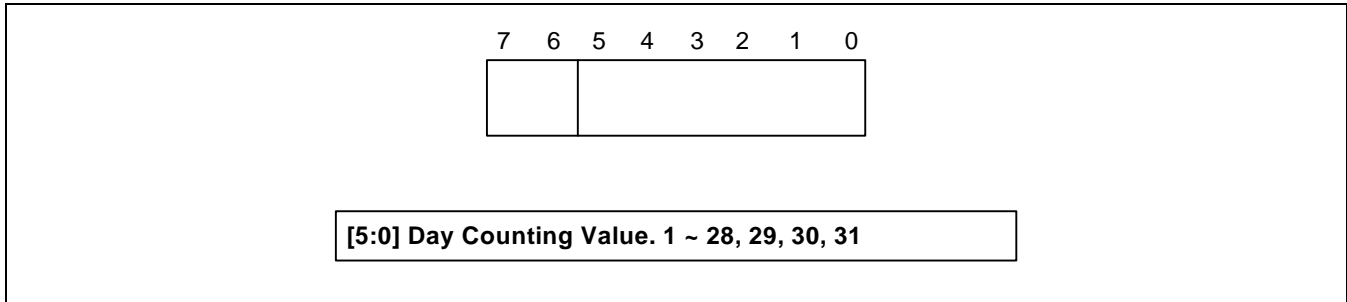


Figure 22-6. BCDDAY Counter Register

BCDDATE COUNTER REGISTER

BCD count register for the date.

Register	Offset Address	R/W	Description	Reset Value
BCDDATE	0xc880	R/W	RTC date register	0xX

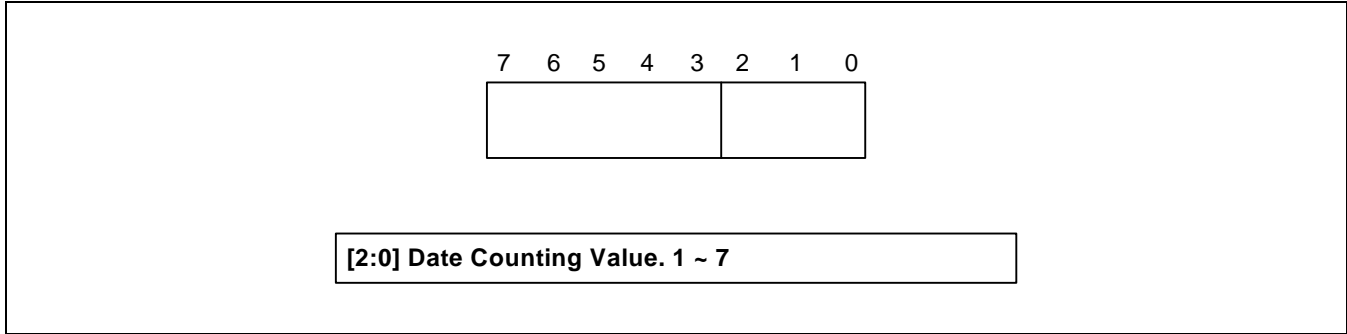


Figure 22-7. BCDDATE Counter Register

BCDMON COUNTER REGISTER

BCD count register for months.

Register	Offset Address	R/W	Description	Reset Value
BCDMON	0xc884	R/W	RTC month register	0xXX

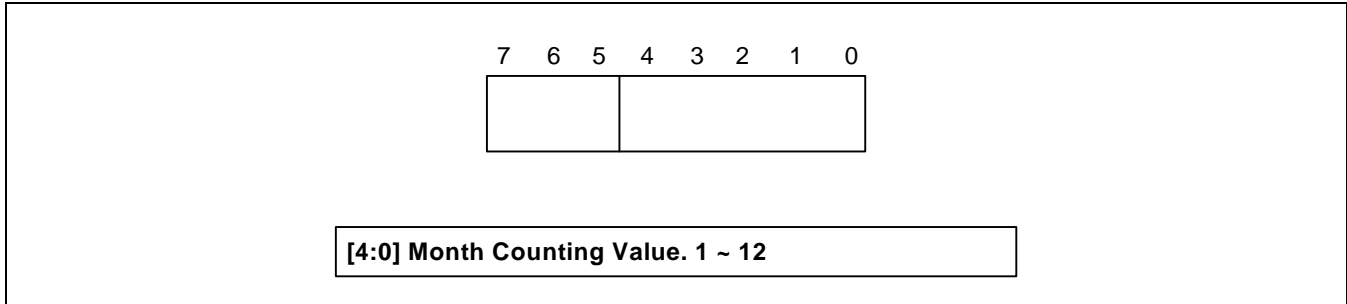


Figure 22-8. BCDMON Counter Register

BCDYEAR COUNTER REGISTER

BCD count register for years.

Register	Offset Address	R/W	Description	Reset Value
BCDYEAR	0xc888	R/W	RTC year register	0xXX

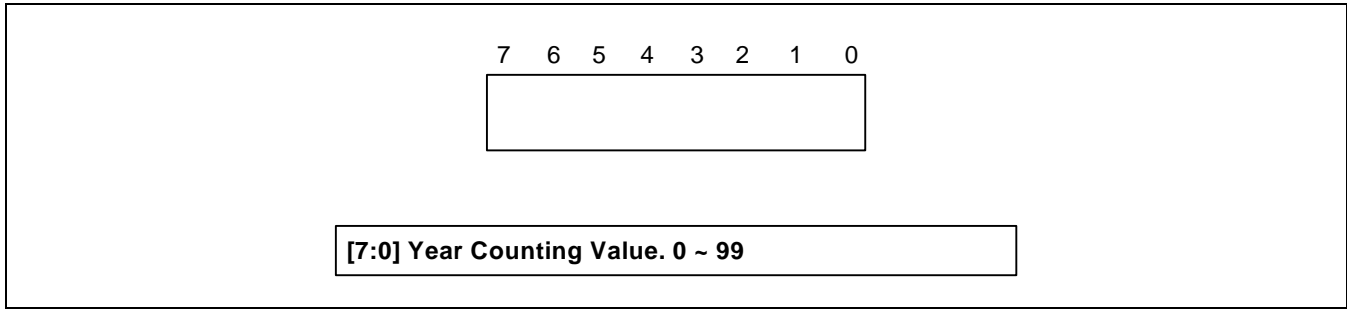


Figure 22-9. BCDYEAR Counter Register

23

CLOCK SAVE & PLL CONTROL

OVERVIEW

PLL is used to generate a higher internal clock from a low external clock source.
Clock saving provides that power dissipation of the periphery decreases in sleeping mode.
SYSTEM CLOCK divided by 40 is cnt_40, and RSTCLK, the frequency divided-logic in the CLKSAV unit, is the reset filtering logic.
SMCLK is MCLK in normal mode or cnt_40 in sleeping mode

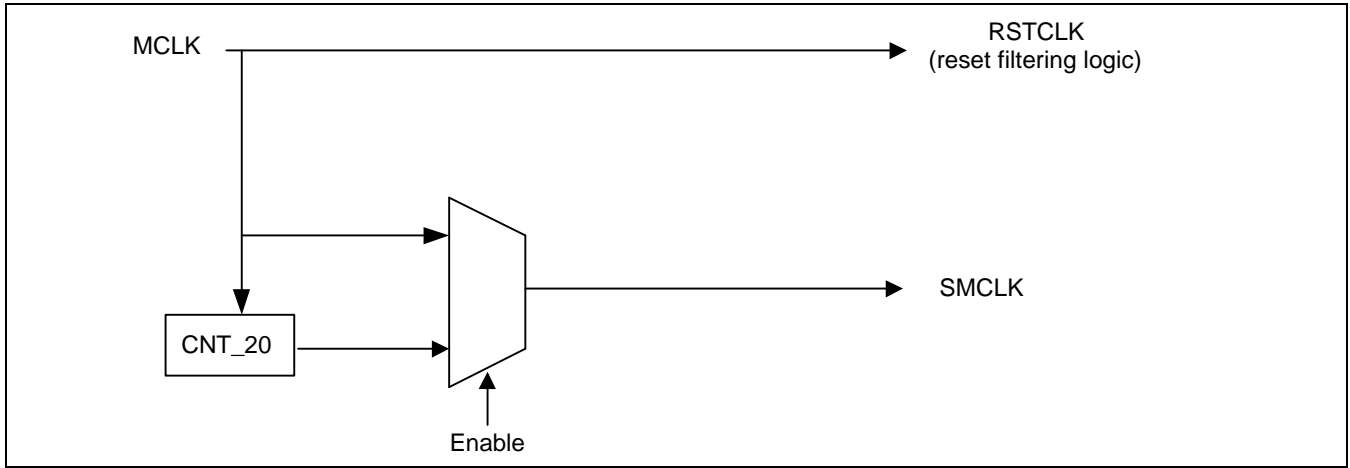


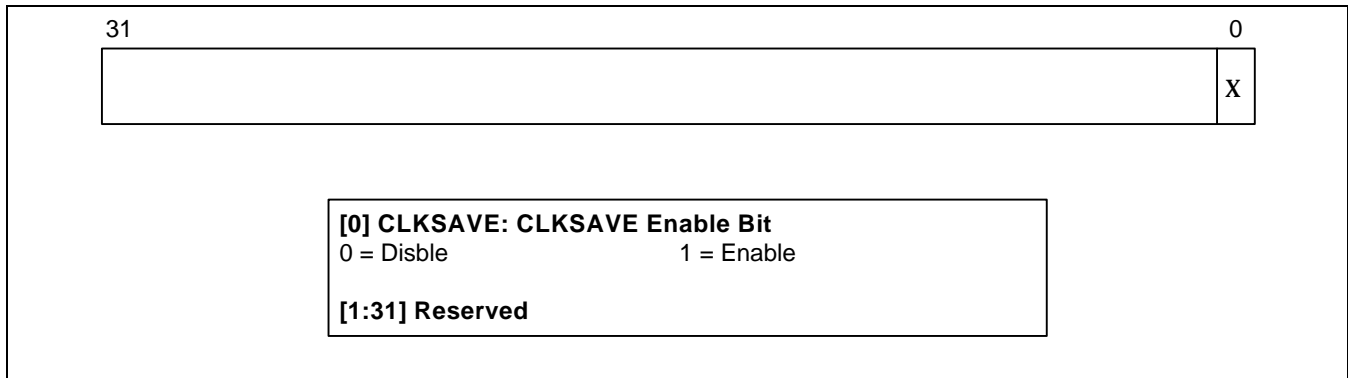
Figure 23-1. Clock Save Block Diagram

REGISTERS

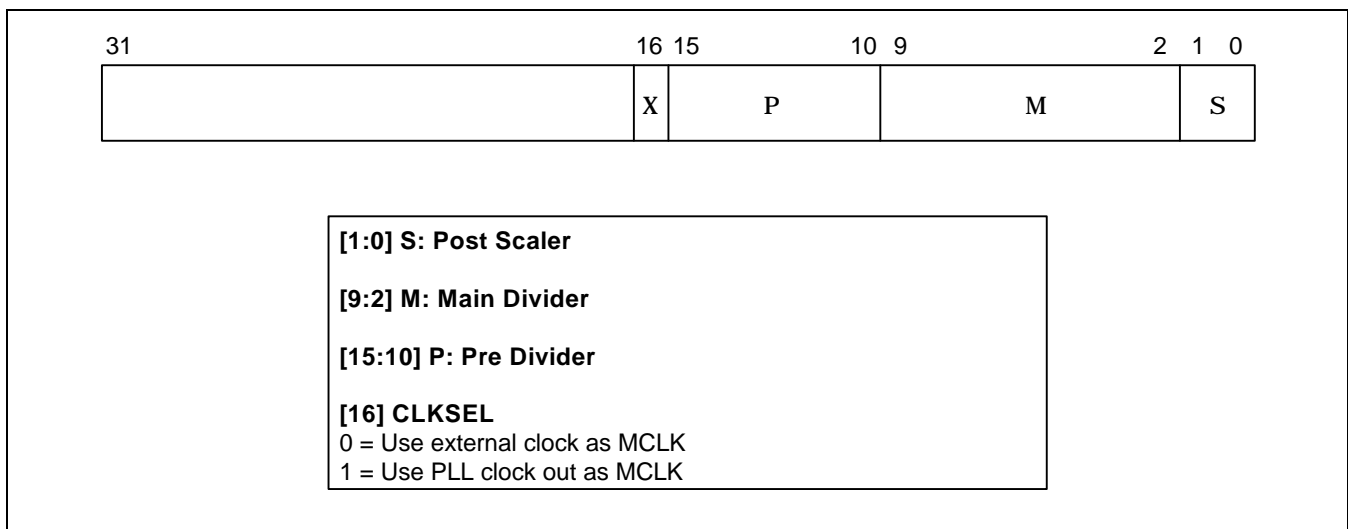
Registers	Offset Address	R/W	Description	Value
CLKSAVCON	0x1800	R/W	CLKSAV control register	0xX
PLLCON	0x1804	W	PLL control register	0x00000

CLKSAVCON REGISTER

The CLKSAVCON register is comprised of the CLKSAVE (CLKSAV Enable: bit 0) which decides whether or not to enable clock saving for the peripherals.

**Figure 23-2. CLKSAVCON****PLLCON REGISTER**

PLLCON controls PLL, and decides whether to use the PLL-generated clock or the external clock as the system clock.

**Figure 23-3. PLLCON**

The frequency of the PLL-generated clock is found by the following formula.

$$\text{PLL clock out(MHz)} = (M+8) * \text{external clock} / ((P+2) * (2^{**}S))$$

If the external clock is 20MHz and M = 0, P = 0, and S = 0, the PLL clock out is $8*20/(2*1) = 80\text{MHz}$.

System Clock Calculation Method when using the Frequency Synthesizer PLL

— Output Frequency Equation: $F_{out} = \frac{(m+8)}{(p+2) * 2^s} \times F_{IN}$

m: value of 8-bit Main-divider, $0 \leq m \leq 255$
p: value of 6-bit Pre-divider, $0 \leq p \leq 63$
s: value of 2-bit Post-scaler, $0 \leq s \leq 3$

- 1) Clock_Input: 10MHz ~ 20MHz (recommend condition)
- 2) Fliter_Input: 820 pF
- 3) Clock_output: Main clock

NOTE

The setting of the PLLCON register can change only one time. For example, after power on, the value of the PLLCON register is 0x0, this is PLL clock is not used. After this you can set the PLLCON register only one time.

Also, We recommend the s's value is greater than or equal to 1, and the $F_{IN}/(p+2)$ is greater than 1MHz.

24

LSU CONTROL

INTRODUCTION

This module performs the following functions:

- V_Window and LD_PreHeat pulse generation
- LSU ready state check
- VDO masking and software on/off control
- LSU motor clock generation
- nHSYNC Filtering

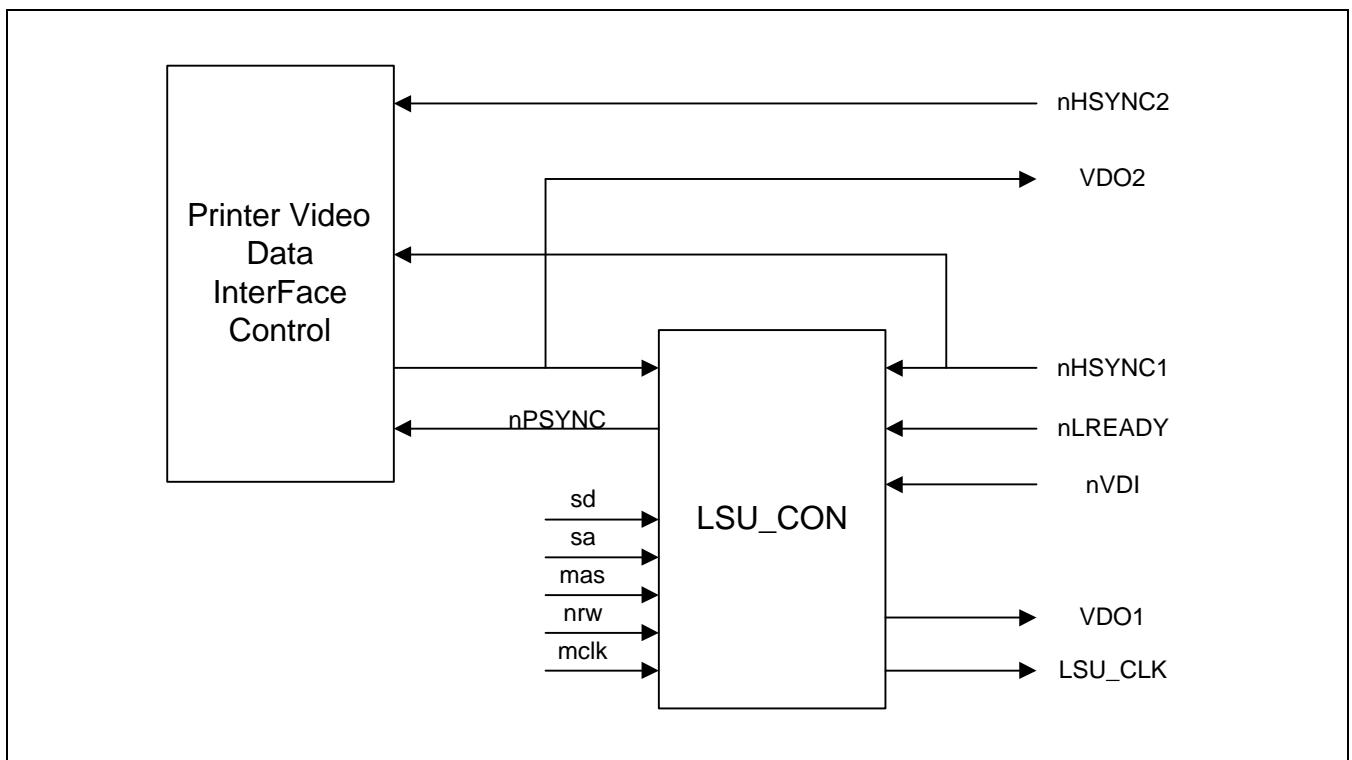


Figure 24-1. LSU Control

MAIN INPUT/OUTPUT SIGNALS

Input

- nLREADY: Signal activated when polygon motor is within the accurate speed
- nHSYNC1: Horizontal beam detect signal from LSU
- nVDI: Video data from PIFC block

Output

- VDO1: Laser diode on/off output (external output, initial "H", active "L")
- nPSYNC: Page sync. signal set by S/W for PIFC block
- LSU_CLK: Clock signal for LSU Motor [$MCLK / \{ (LSUCK_CNT \text{ value} + 1) \times 2 \}$]

SPECIAL REGISTER

LSU_CON CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
LSUCON	0xd000	R/W	LSU_CON control register	0x0000

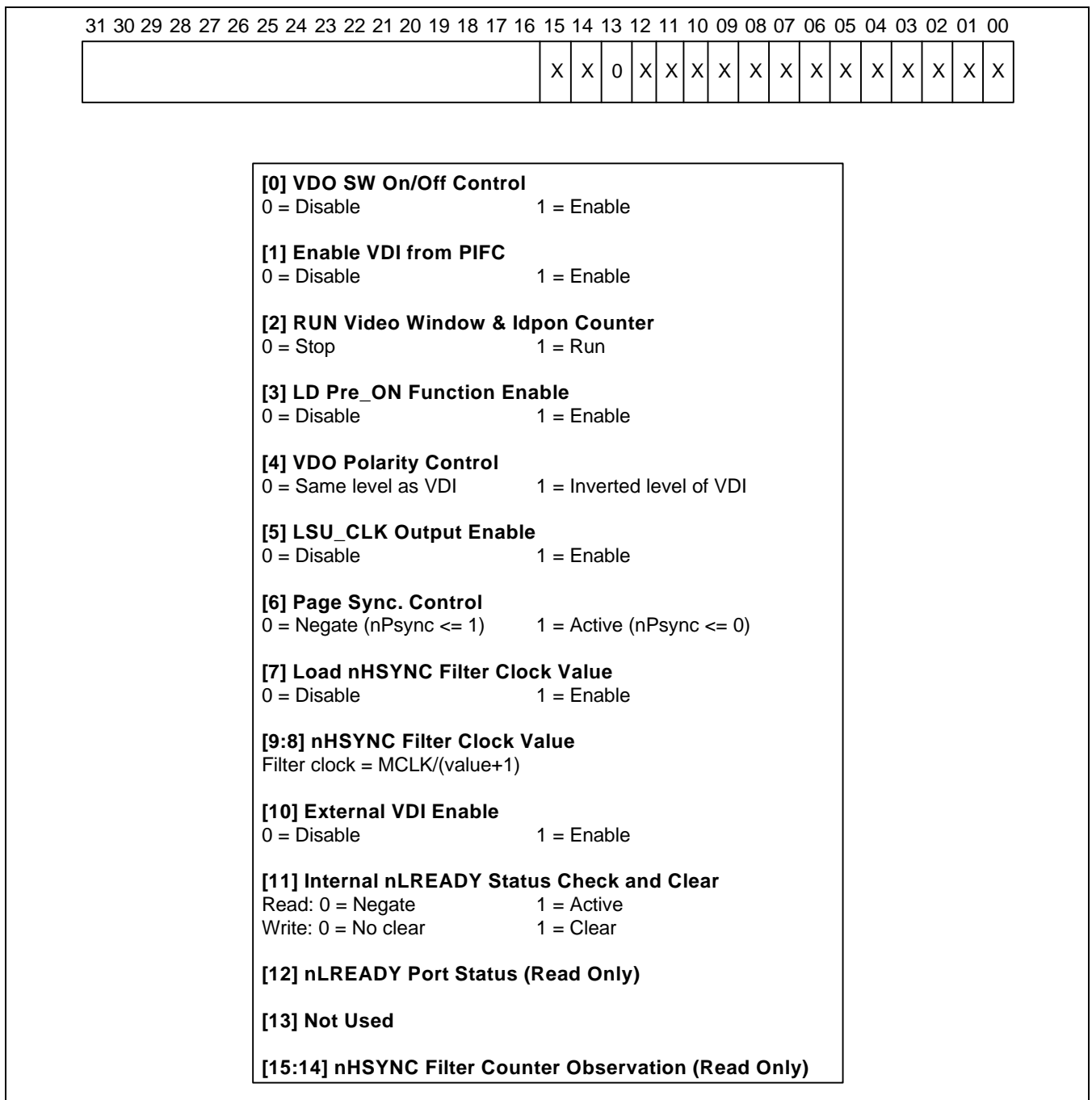
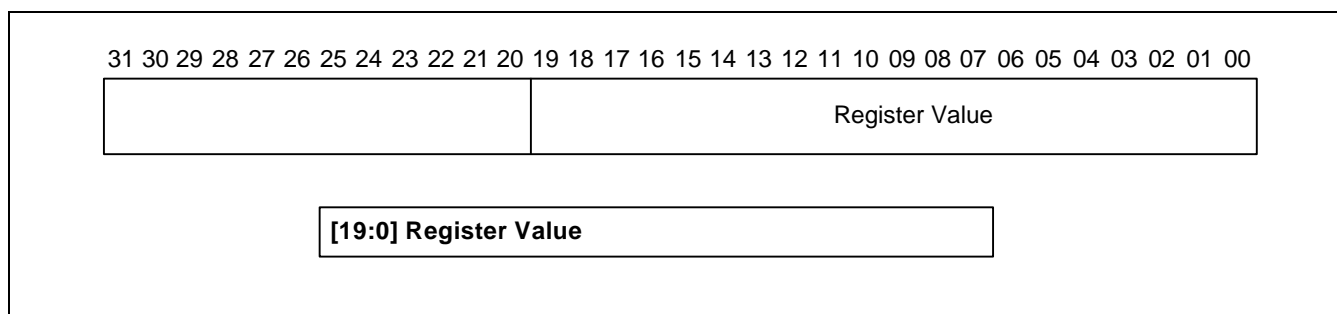


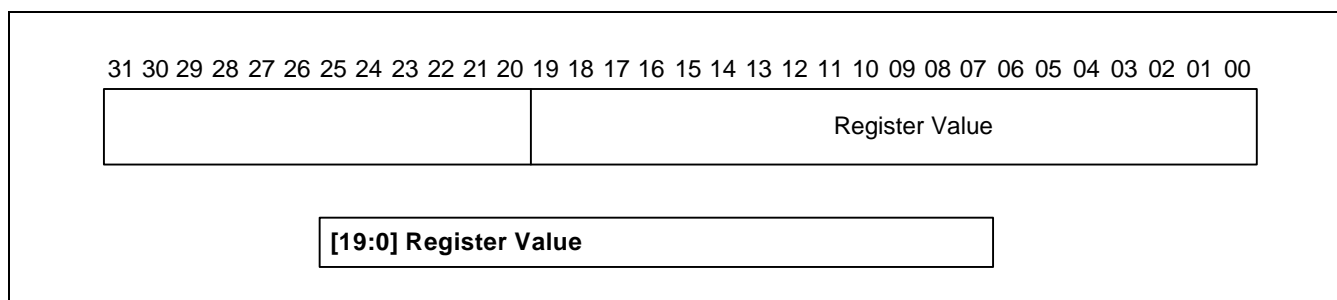
Figure 24-2. LSU_CON Control Register

V_WINDOW START/END TIME REGISTER

Registers	Offset Address	R/W	Description	Reset Value
VWIN_STR	0xd004	R/W	V_Window time start register	0x00000
VWIN_END	0xd008	R/W	V_Window time end register	0x00000

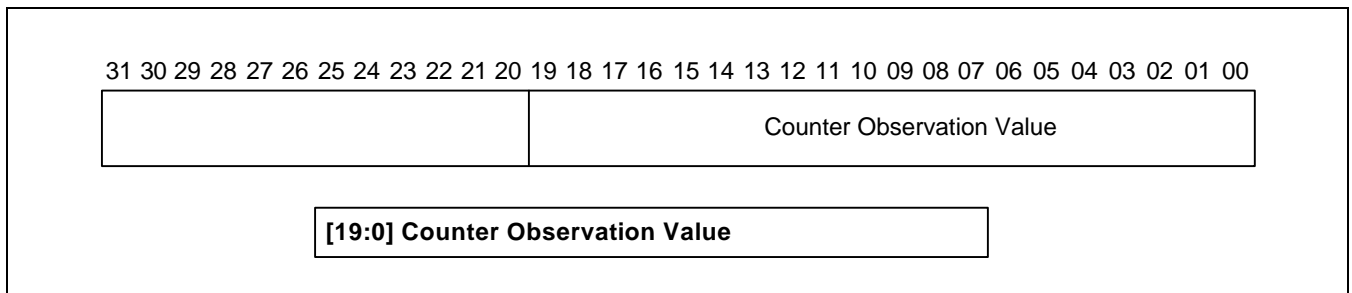
**Figure 24-3. V-Window Time Start/End Register****LD_ON PRE/POST TIME REGISTER**

Registers	Offset Address	R/W	Description	Reset Value
LDON_Pre	0xd00c	R/W	LD ON Pre time register	0x00000
LDON_Post	0xd010	R/W	LD ON Post time register	0x00000

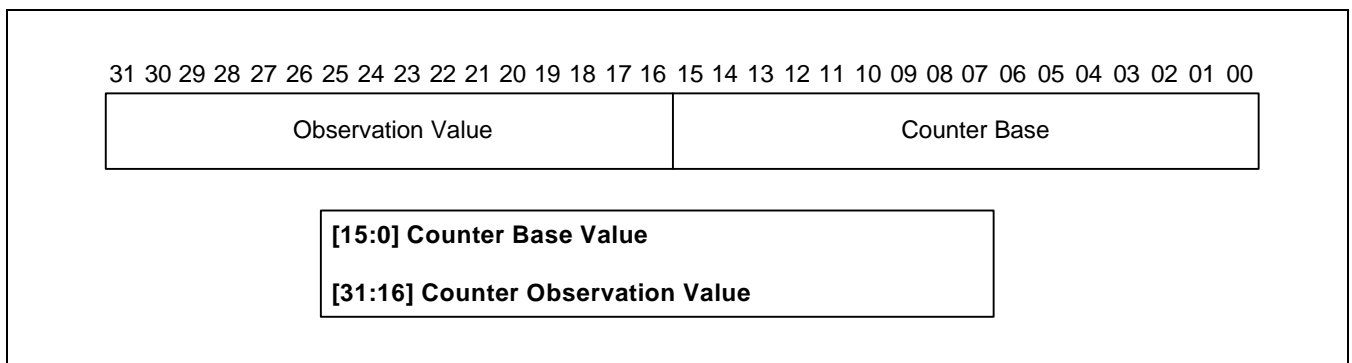
**Figure 24-4. LDON_Pre/Post Time Register**

V_WINDOW COUNTER OBSERVATION REGISTER

Register	Offset Address	R/W	Description	Reset Value
VCNT_OBS	0xd014	R/W	V_Window counter observation register	0x00000

**Figure 24-5. V-Window Counter Observation Register****LSU MOTOR CLOCK GENERATION COUNTER REGISTER**

Register	Offset Address	R/W	Description	Reset Value
LSUCK_CNT	0xd018	R/W	LSU Motor Clock counter base & observation register	0x00000000

**Figure 24-6. LSU CLK Counter Base/Observation Register****Caution**

The counter of the V-Window time start/end register and the LDON_ pre/post time register should be run when the control register's run bit is stop ("L") and the needed initial value is written.

The value written when the counter is being run is applied after the counting for the previous value is finished.

25

PRINTER INTERFACE CONTROLLER

OVERVIEW

The PIFC performs direct memory accesses to fetch video data, and then, serializes the data and handshakes with the printer to transmit the video data after pattern procession. It has the following important features:

- It uses dedicated DMA to accelerate data transfers between page memory and the laser printer engine. The dedicated DMA supports queued operations to facilitate the smooth switching between blocks of banded page memory.
- The PIFC's DMA controller can transfer strings of consecutive zeros (the 0's in a given banded bit map, or blank data) without accessing external memory. The length of a zeros string is determined by the value in the transfer count register of the PIFC's queue 0 or queue 1.
- The KS32C65100 PIFC employs pixel chopping to save printer toner.
- It provides a fine edge to print images by shrinking the first pixel dot whenever there is a string of consecutive 1's (that is, at the position where the left edge of the image starts).
- It supports 2 to 4-times image expanding print.
- It is able to control top-margin, left-margin and image width for page layout.

PAGE IMAGE DATA FETCH OPERATION

Page images are stored in an area of memory known as the band buffer. After a page image is rendered, the PIFC can be programmed to fetch the contents of the band buffer to fill its FIFO.

The data fetch operation is performed by PDMA, and queued PDMA operations are supported by PIFC specially for the print task with a large amount of video data. The principle of queued operation is that to divide the whole video data into several data blocks. The first block of data is transferred by DMA queue 0 and the second block is transferred by DMA queue 1, and during one queue operation the other DMA queue can be set to prepare for next block transfer so that the next block transfer operation can start as soon as the previous block transfer operation is completed. The switching between two DMA queues is implemented automatically by a data fetch controller, so as to guarantee the continuity of data transfer.

Normally, an EOP (End-of-Page) interrupt is posted when a whole page video data transmission is completed, and then the PIFC returns to idle. However, an abnormal interrupt, PUR (Page Under-run), may be generated if one DMA queue is not been ready when another DMA queue operation is completed.

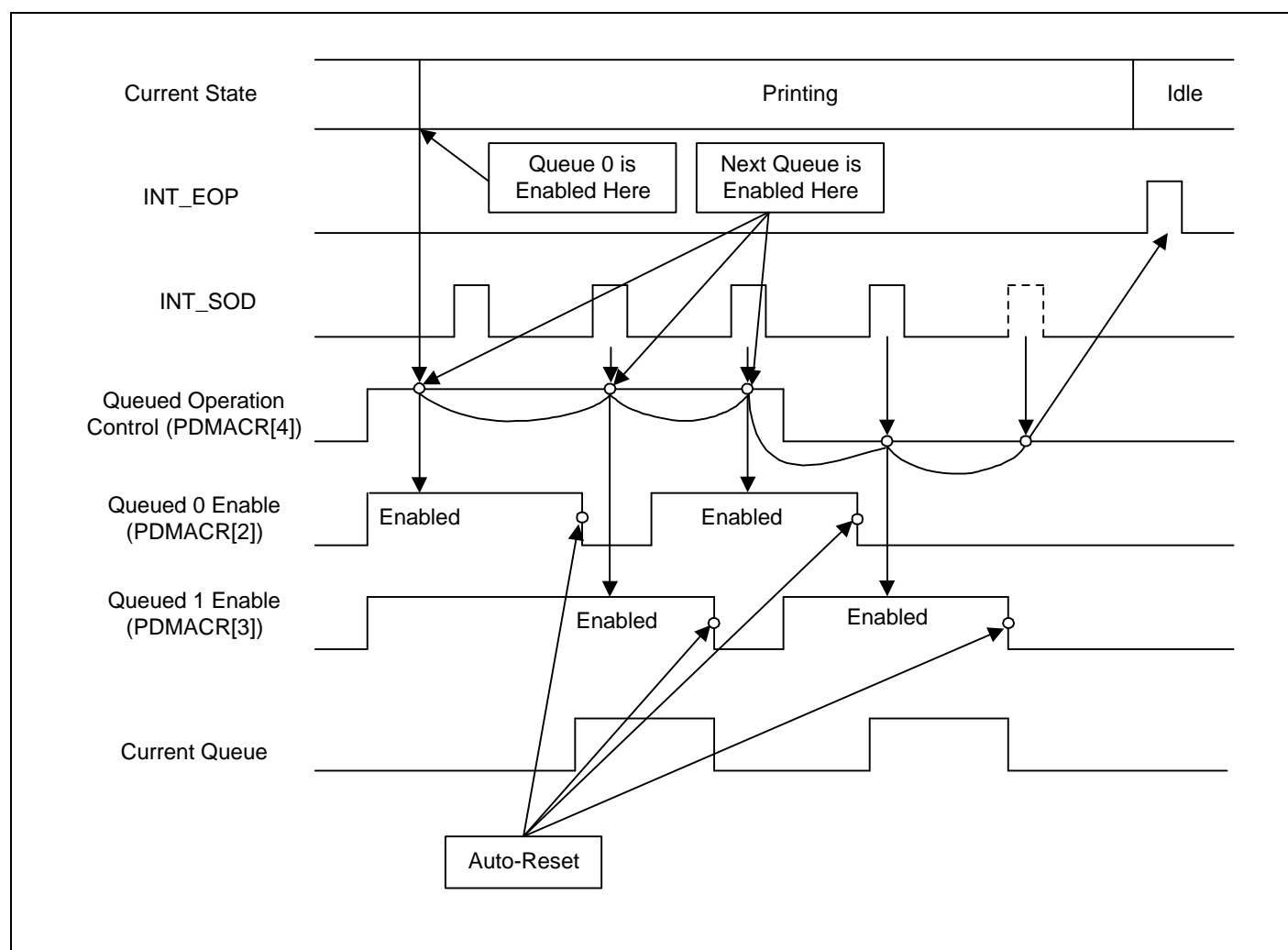


Figure 25-1. Queued Operation for End-of-Page (EOP)

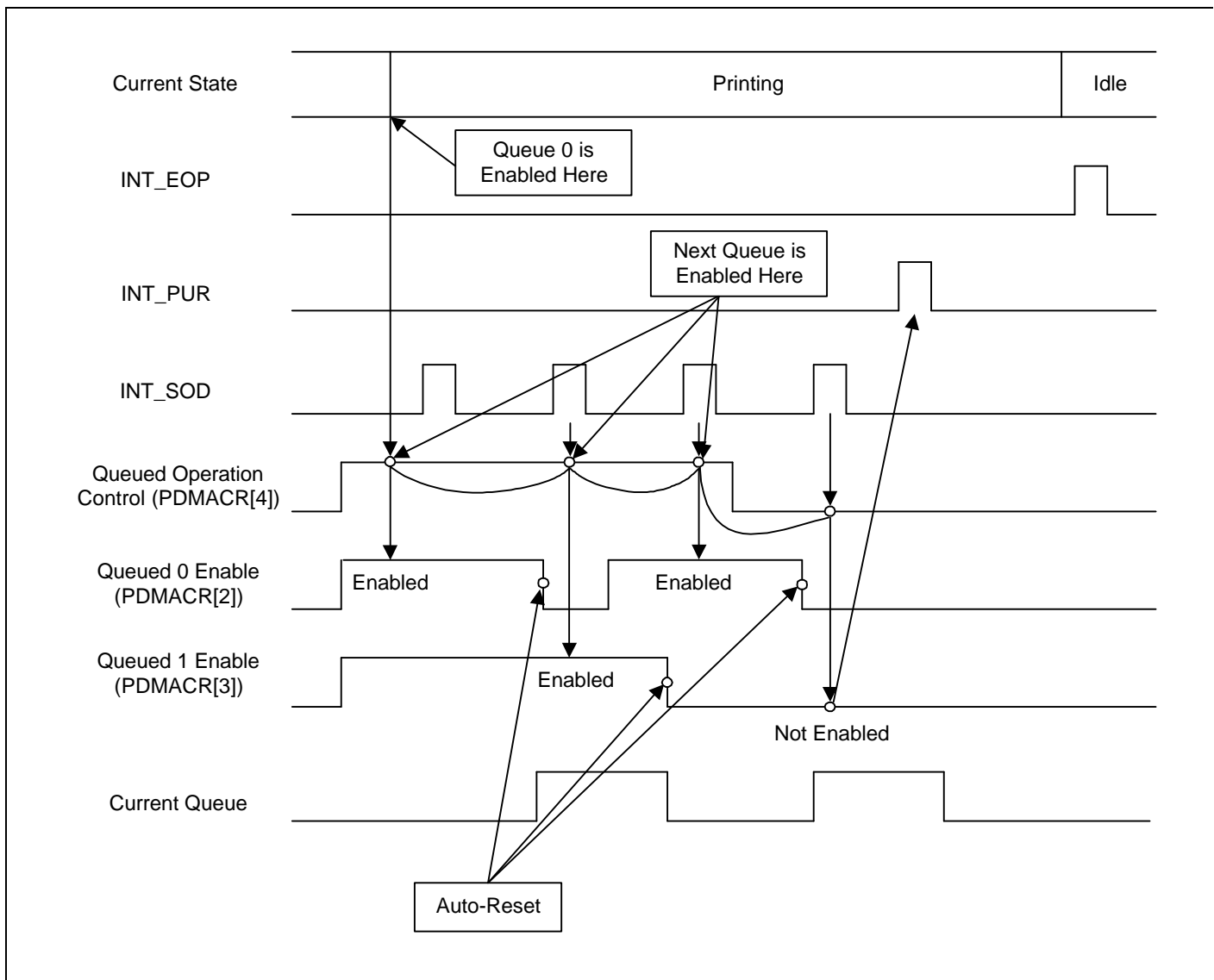


Figure 25-2. Queued Operation for Page Under-run (PUR)

PRINT OPERATION

The print job is started by PIFC issuing an active print command signal nPRINT (setting VCON[1] to "1"), which means that the KS32C65100 PIFC is ready to start a print job. The PIFC then begins waiting for the nPSYNCRQ from LSU_CON. After nPSYNCRQ arrives, the PIFC activates nPSYNC signal by setting VCON[2] to "1", and in the meantime, the top margin counting operation begins. The top margin counter is decreased until the count reaches "0", and then the PIFC begins to transmit video data.

The nPRINT signal must be held active until nPSYNC becomes inactive. By using interrupts, the nPSYNC time interval can be controlled. As shown in Figure 25-3, the transitions on nPSYNCRQ signal level cause the occurrences of SYNC1 interrupts. So the nPSYNC can be activated and inactivated in the ISR (interrupt service routine) of INT_SYNC1.

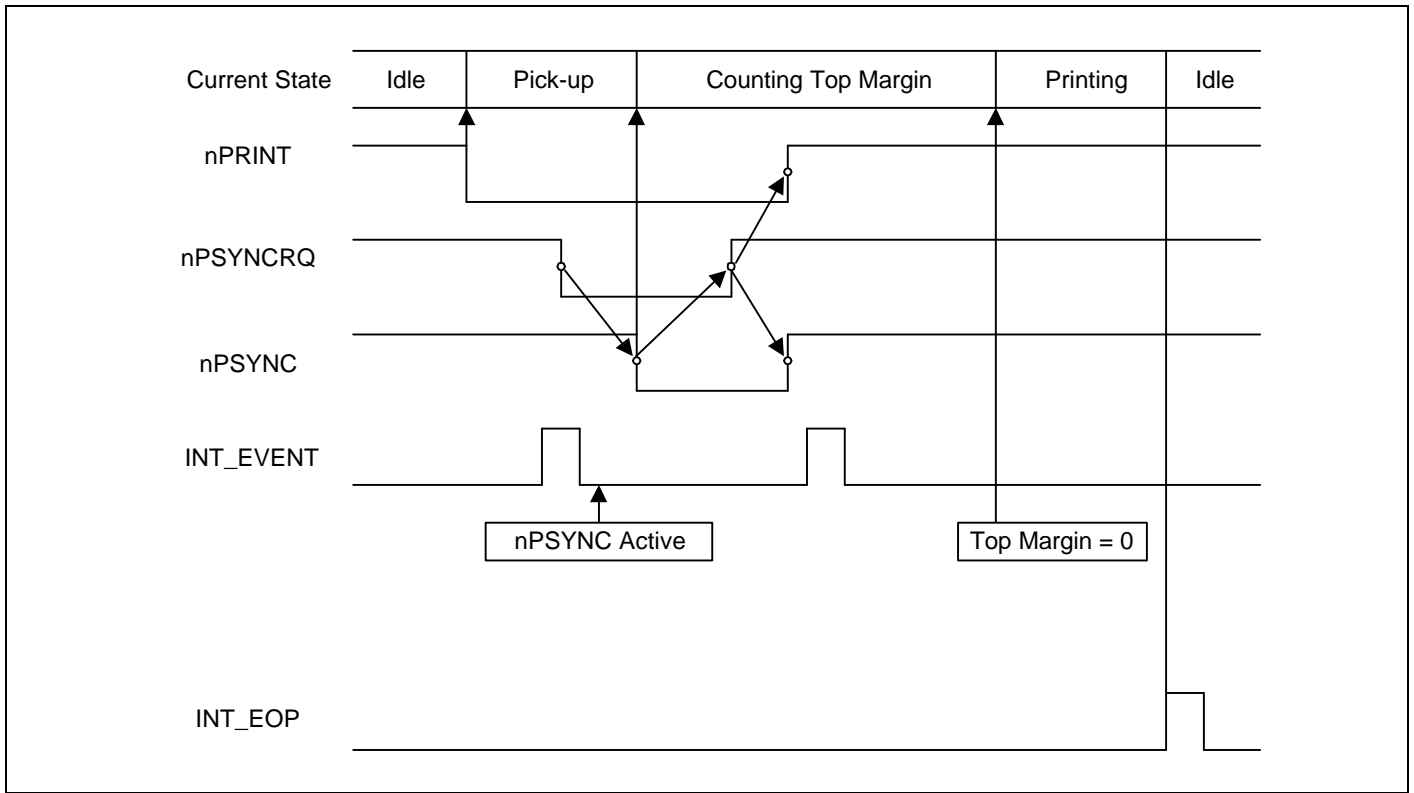


Figure 25-3. Protocol Diagram (PIFC and Printer Engine)

PIFC SPECIAL REGISTERS

PDMA AND ENGINE INTERFACE STATUS REGISTER

The printer interface controller's PDMA and engine interface status register, status, contains read-only status bits used to monitor the progress of print operations, including power ready, ready to print, print synchronization, PIFC status, and currently active DMA queue.

Register	Offset Address	R/W	Description	Reset Value
STATUS	0xa000	R	PDMA and engine interface status register	0x00

[1:0]	Reserved	
[2]	Print Synchronization request	When STATUS[2] is "1", a print synchronization request (nPSYNCRQ) is being received from the laser printer engine. When the engine issues this request, it is ready to receive the synchronization pulse, nPSYNC, from the KS32C65100.
[4:3]	Current PIFC status	The value of this bit-pair indicates the current operating status of the printer interface controller. There are four states: idle, pick-up, counting top margin, and active printing.
[5]	Current DMA queue	The KS32C65100 uses two DMA queues for dedicated printer DMA, DMA 0 and DMA 1. The STATUS[5] status bit indicates which queue is currently active during a PDMA operation. When STATUS[5] is "0", DMA queue 0 is active; when it is "1", DMA queue 1 is active.

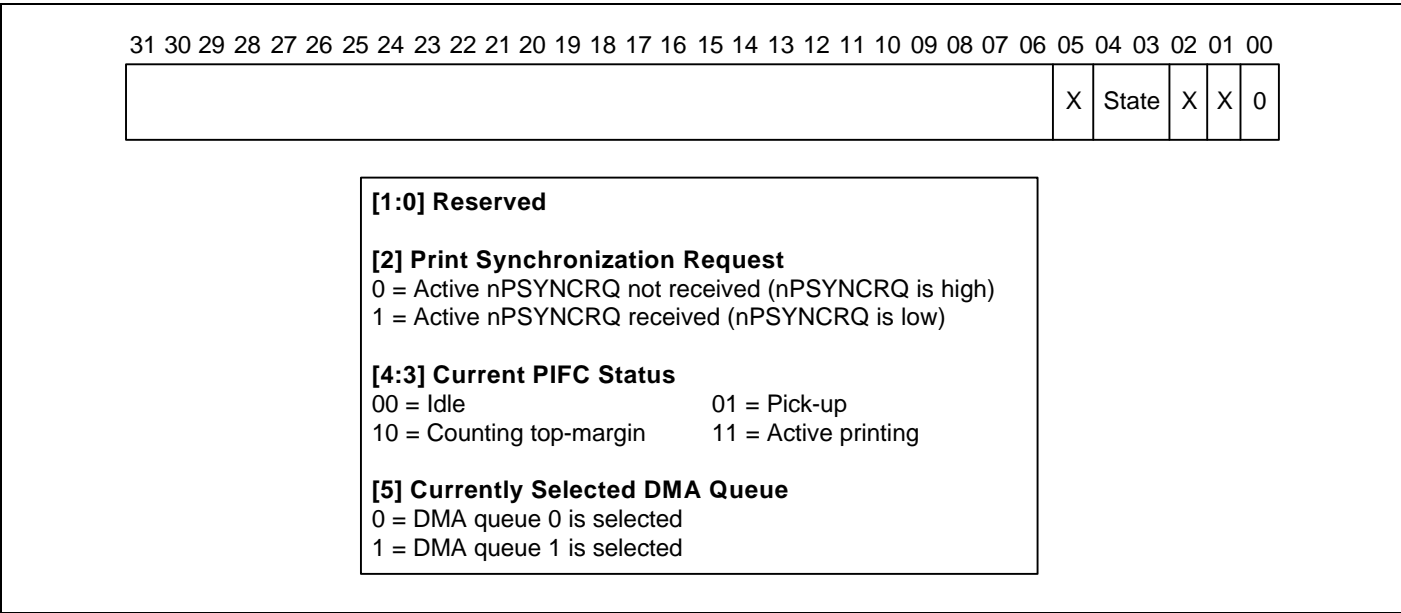


Figure 25-4.PDMA and Engine Interface Status Register (STATUS)

VIDEO CONTROL REGISTER

Settings in the PIFC video control register, VCON, control activities of the KS32C65100 printer interface controller during a printing operation, including video clock selection and the shift direction of video data.

Register	Offset Address	R/W	Description	Reset Value
VCON	0xa004	R/W	Video control register	0x00

[0]	Reserved		
[1]	nPRINT output		When VCON[1] is "1", it signals the printer engine that the KS32C65100 PIFC is ready to start a print job.
[2]	Print synchronization		This bit activates or inactivates nPSYNC signal.
[3]	Video clock inversion		When using external a video clock (VCLK), if VCON[3] is "1", the PIFC uses a non-inverted external video clock (VCLK) as its clock. Otherwise, it uses the inverted external VCLK. The VCLK selection (VCLK) depends on the setting in PCON[3:2].
[4]	Video data shift direction		In video data transmission, if VCON[4] is "1", the shift direction of video data in the shift register is LSB-first. Otherwise, the shift direction is MSB-first.
[5]	Stop printing		When VCON[5] is set to "1", PIFC stops printing and generates the End-of-Page interrupt (INT_EOP), and then VCON[5] is auto-cleared to "0" and all of PIFC state is reset.

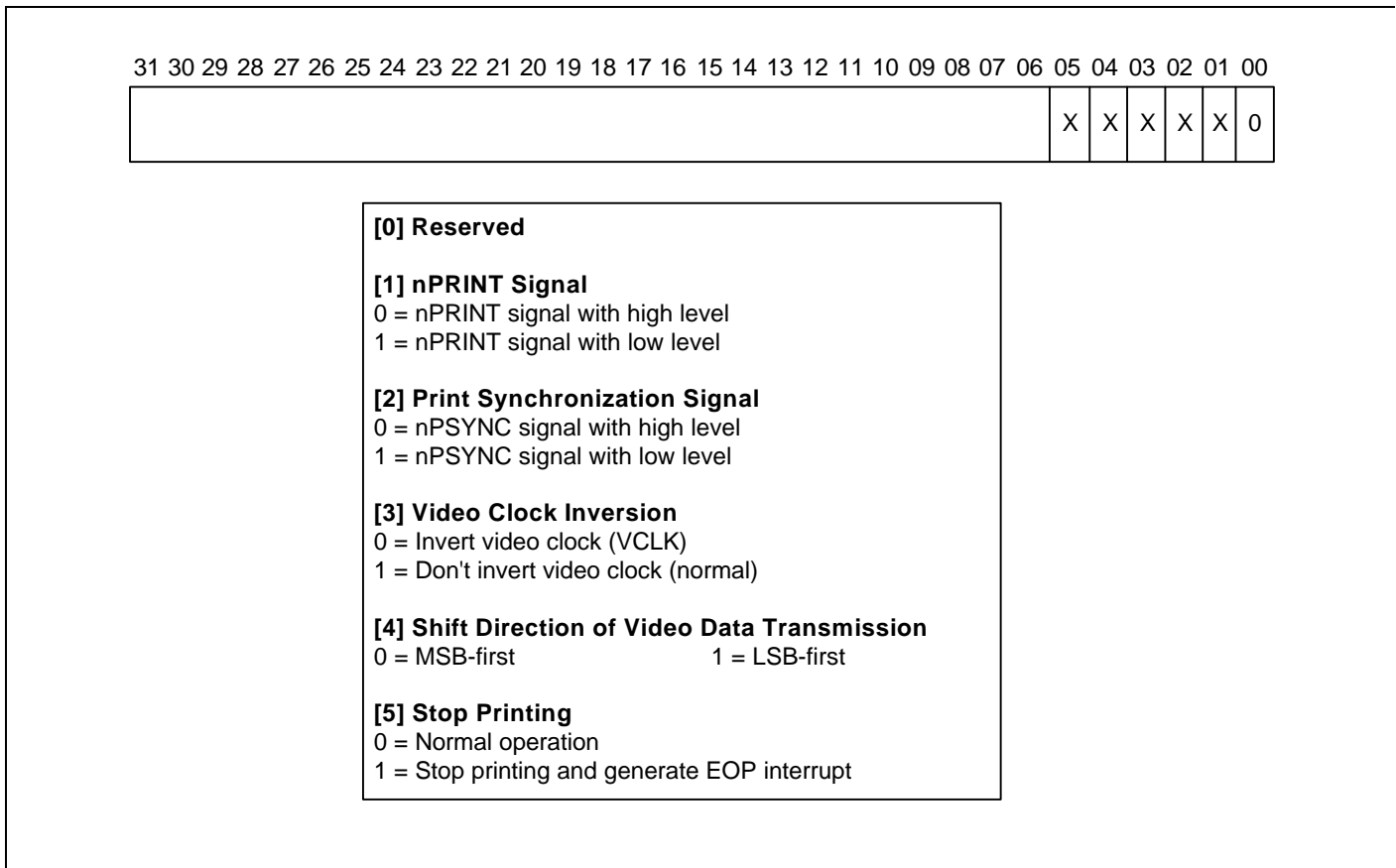


Figure 25-5. Video Control Register (VCON)

PATTERN CONTROL REGISTER

Settings in the printer interface controller's pattern control register, PCON, control various video data functions including video data polarity, border data polarity, video clock selection, clock divisor, shrink pattern, data chopping selection for toner savings and image expanding.

Register	Offset Address	R/W	Description	Reset Value
PCON	0xa008	R/W	Pattern control register	0x000000

[0]	Video data polarity	When PCON[0] is "0", the video data that the KS32C65100 sends to the printer engine is inverted. Otherwise, the video data sent in a non-inverted stream.
[1]	Border data polarity	When PCON[1] is "0", the border data, which corresponds to the blank area on paper around the image to be printed, including the top, left, right and bottom margins, is inverted. otherwise, the border data is not inverted.
[3:2]	Video clock selection	When PCON[3:2] is "01", the PIFC selects the external video clock 0 (VCLK0) as its video clock; and when PCON[3:2] is "10", the PIFC doesn't selects any clock. Otherwise, it selects the internal system clock, MCLK.
[6:4]	Video clock divisor selection	This 3-bit value determines the divisor for the selected video clock.
[9:7]	Video data shrink pattern	Using this 3-bit value, you can create special effects in the printed image. Depending on the video clock divisor n, to achieve a fine print edge, the size of the first pixel dot that is detected at the left edge of the image is shrunk by 1/n of the normal pixel size, or by 2/n, 3/n, and so on. The left edge of an image is defined as the pixel from which a string of consecutive 1's is detected on a scan line. In other words, the size of the first pixel in the string of consecutive 1's is reduced in order to achieve a sharper "left edge" of the printing area.
[17:10]	Video data chopping	Each bit of video data corresponds to a pixel dot in printing, and the pixel dot consists of n sub-pixels (n is the video clock divisor defined by PCON[6:4]). To save printer toner, one or more sub-pixels for each bit pixel can be chopped in printing and the position of the sub-pixel to be chopped is specified by PCON[17:10]. Among the eight bits of PCON[17:10], the positions of zeros determines the positions of the sub-pixels to be chopped. For example, PCON[6:4] is specified as "111" (i.e. the n is equal to 8), then each bit of video data (one pixel dot) corresponds to 8 sub-pixels in printing. If PCON[17:10] is specified as "10110010", the 1st, 3rd, 4th and 7th sub-pixel for each pixel will be chopped in printing.

[19:18]	Image expanding ratio	This 2-bit value determines the image expanding ratio. When PCON[19:18] is not "00", the image to be sent to the printer engine is expanded first according to the defined ratio and then sent to the engine.
[20]	HSYNC selection	Selects the HSYNC signal to be used between HSYNC1 and HSYNC2.
[21]	Test mode	If '0', normal mode. If '1', outputs the test pattern mode by the TPVAL register and TPON register.

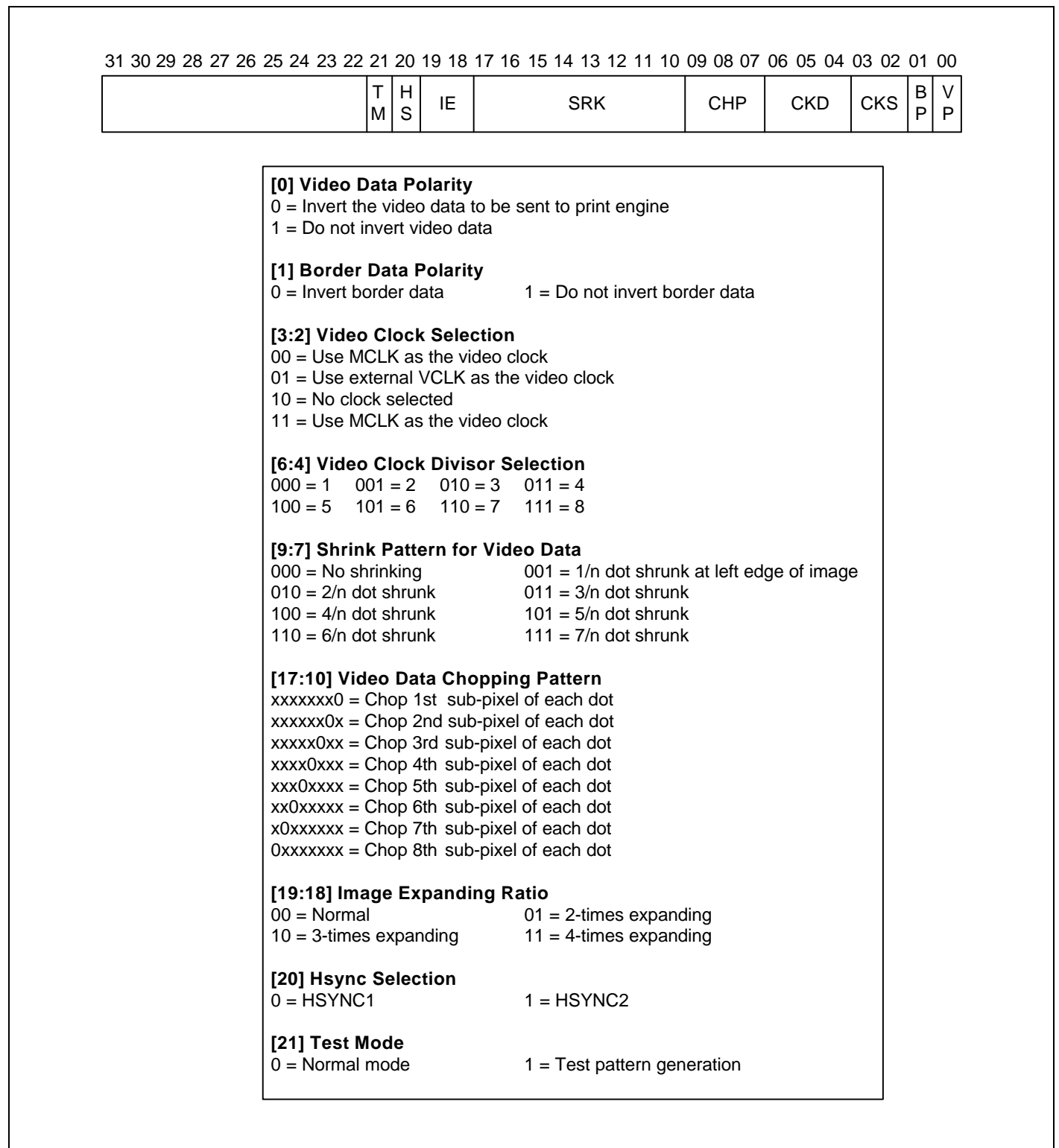


Figure 25-6. Pattern Control Register (PCON)

PRINTER DMA CONTROL REGISTER

The printer DMA control register, PDMACON, is used to control the operation of the printer DMA queues.

Register	Offset Address	R/W	Description	Reset Value
PDMACON	0xa00c	R/W	PDMA control register	0x00

[0]	Blank mode: DMA queue 0	When PDMACON[0] is "1", the shift register of printer DMA queue 0 sends a stream of zeros to the laser printer engine as video data. No external memory access is required during this PDMA operation. Blank mode is useful for sending a "blank image" if the bit map of a certain banded image consists of all zeros (blank). When this bit is "0", all PDMA accesses are in normal mode. That is, external page memory must be accessed to fetch the page bit map.		
[1]	Blank mode: DMA queue 1	When PDMACON[1] is "1", the shift register of DMA queue 1 sends a stream of zeros to the laser printer engine as video data. (This control bit has the same effect for PDMA queue1 as PDMACON[0] does for PDMA queue 0.)		
[2]	DMA queue 0 enable	When PDMACON[2] is set to "1", queue 0 is enabled and a printer DMA 0 operation can start. When the queue 0 operation is completed, this bit is automatically cleared to "0".		
[3]	DMA queue 1 enable	When PDMACON[3] is set to "1", queue 1 is enabled and a printer DMA 1 operation can start. When the queue 1 operation is completed, this bit is automatically cleared to "0".		
[4]	Queued operation enable	The value of this bit determines whether PDMA uses queued operation to transfer banded bit-mapped data to the laser engine. If PDMACON[4] is "0", PDMA queue 0 or queue 1 transfers data over one queue or the other, without alternating between the two. If PDMACON[4] is "1", banded bit-mapped data is transferred in an alternating queue operation using both queues.		
[5]	PDMA direction	The PDMACON[5] control bit determines whether the bit map in a PDMA operation is printed from top-to-bottom (down-printing) or from bottom-to-top (up-printing).		

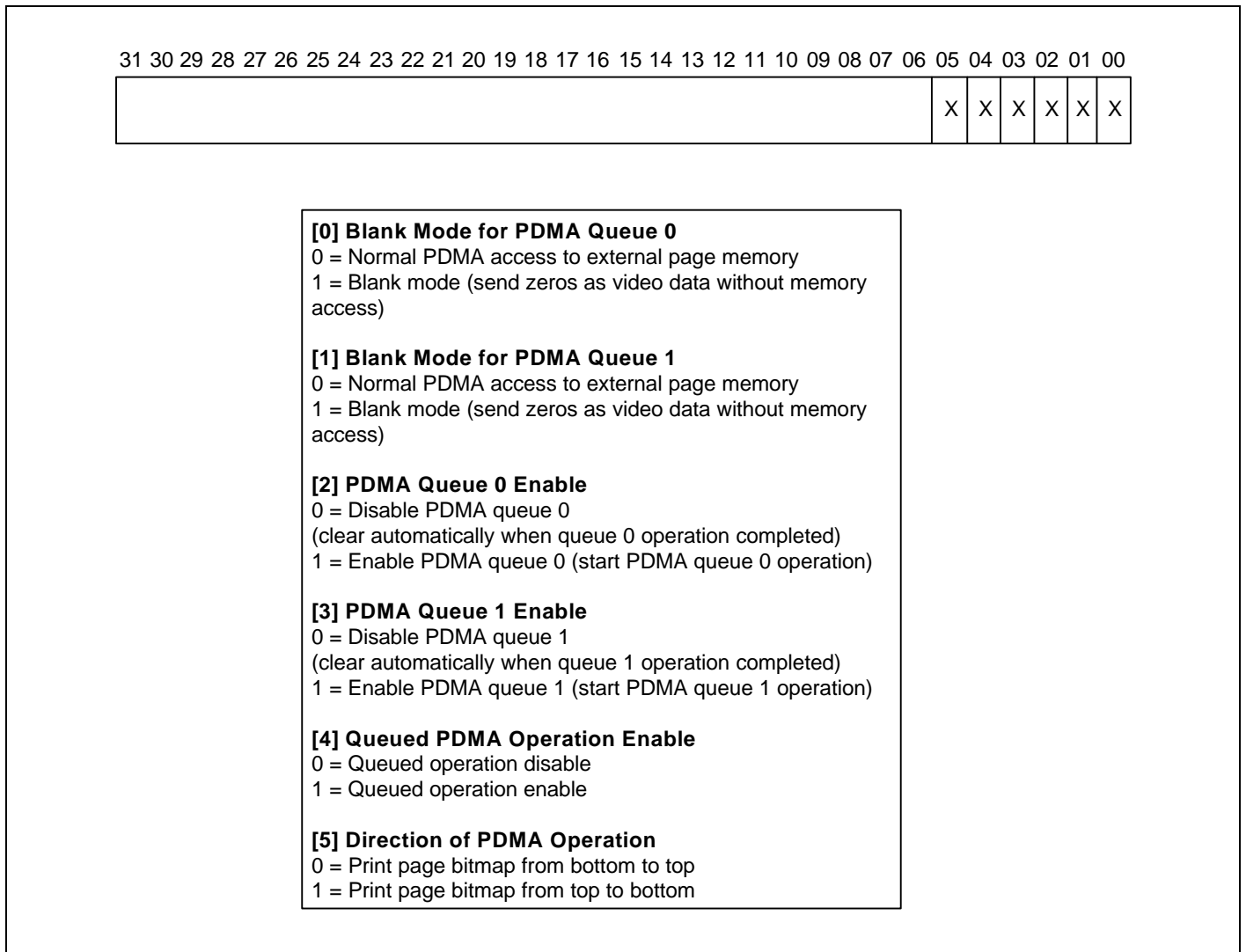


Figure 25-7. Printer DMA Control Register (PDMACON)

TOP MARGIN REGISTER

The value written to the top margin register, TOP, controls the number of scan lines to be skipped when printing starts. An internal counter records the number of nENGHSYNC pulses to determine the beginning of the effective printing area.

Register	Offset Address	R/W	Description	Reset Value
TOP	0xa010	R/W	Top margin register	0x0000

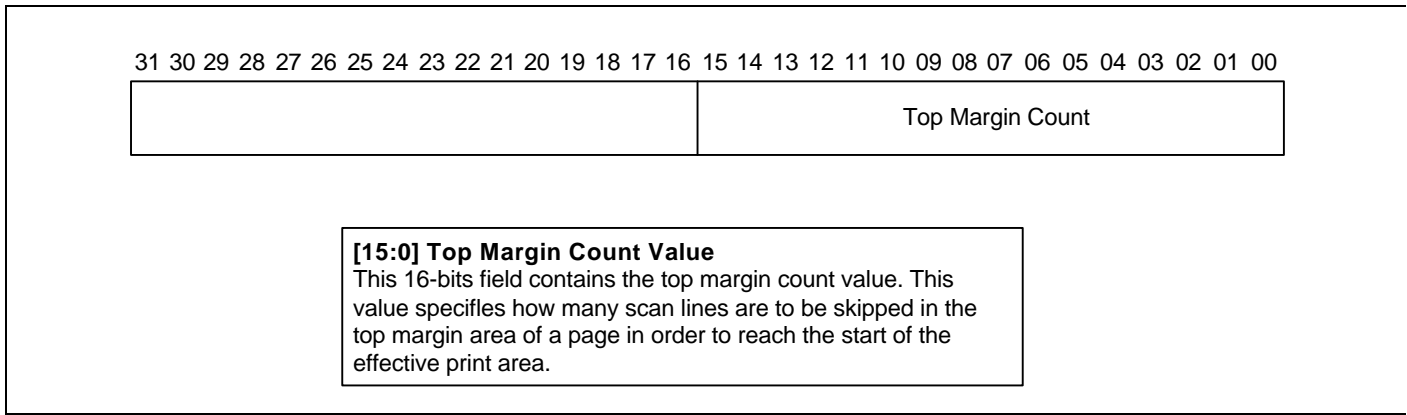


Figure 25-8. Top Margin Register (TOP)

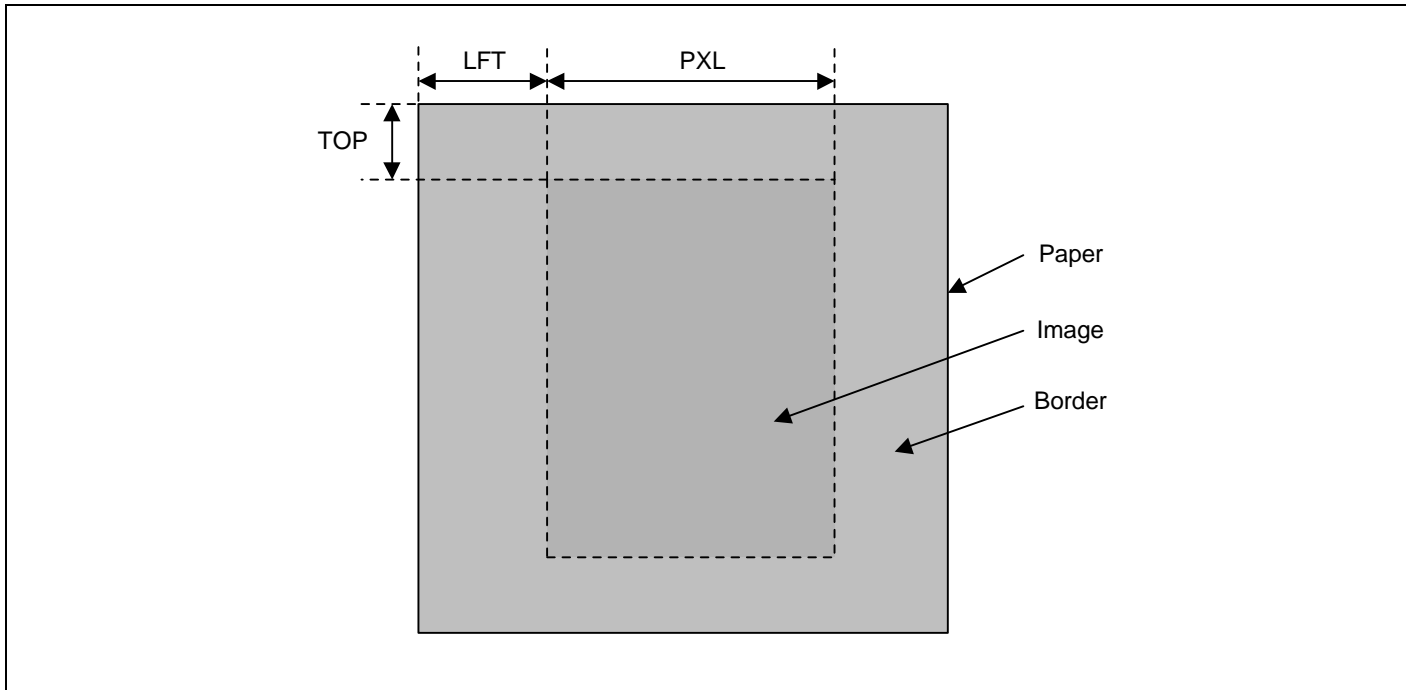


Figure 25-9. Page Layout

LEFT MARGIN REGISTER

The PIFC left margin register, LFT, controls the number of pixels that are skipped when a scan-line operation starts in synchronization with nENGHSYNC. An internal counter records the number of pixels skipped in order to determine the starting pixel of the scan-line operation.

Register	Offset Address	R/W	Description	Reset Value
LFT	0xa014	R/W	Left margin register	0x0000

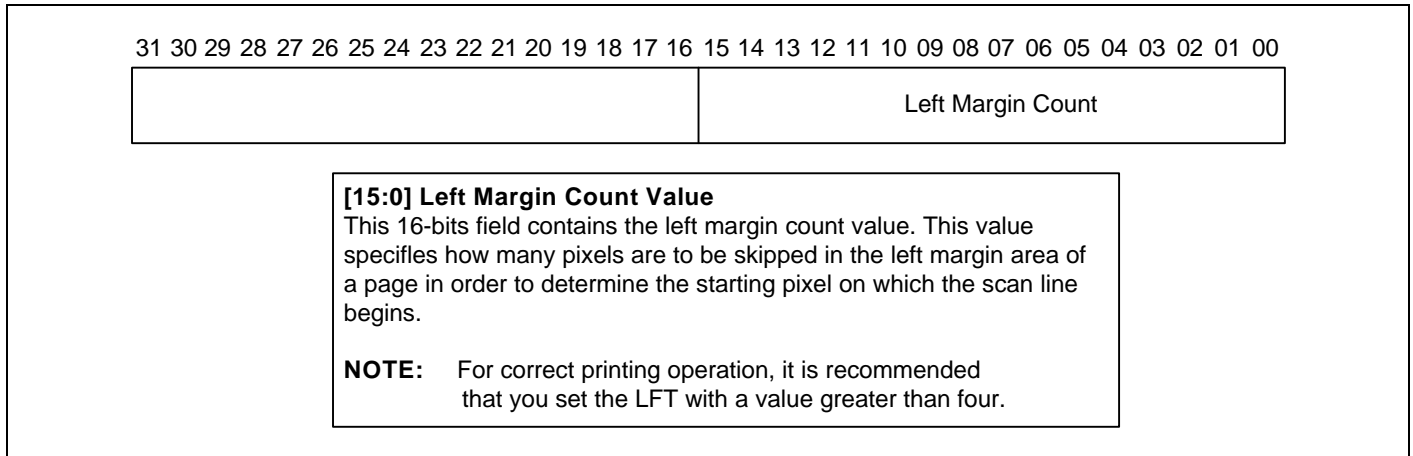


Figure 25-10. Left Margin Register (LFT)

PIXEL COUNT REGISTER

The value stored in the pixel count register, PXL, determines the total number of pixels per scan line.

Register	Offset Address	R/W	Description	Reset Value
PXL	0xa018	R/W	Pixel count register	0x0000

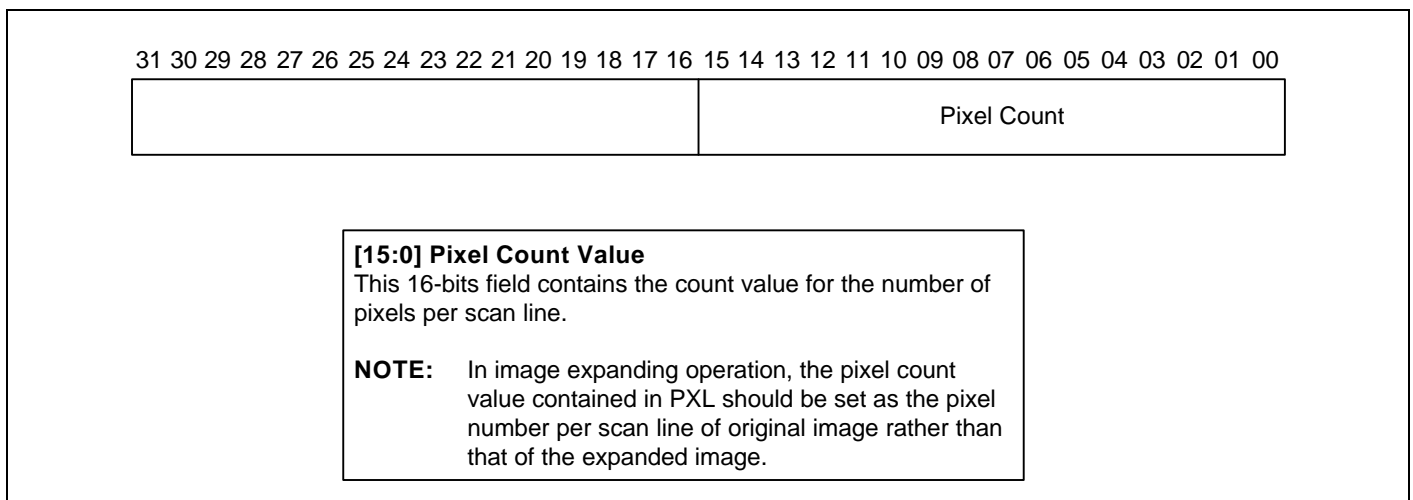


Figure 25-11. Pixel Count Register (PXL)

QUEUE 0/1 START ADDRESS REGISTERS

The values written to the two queue start address registers, QSAR0 and QSAR1, respectively define the starting byte address for PDMA queues 0 and 1.

Registers	Offset Address	R/W	Description	Reset Value
QSAR0	0xa01c	R/W	PDMA queue 0 start address register	0x0000000
QSAR1	0xa024	R/W	PDMA queue 1 start address register	0x0000000

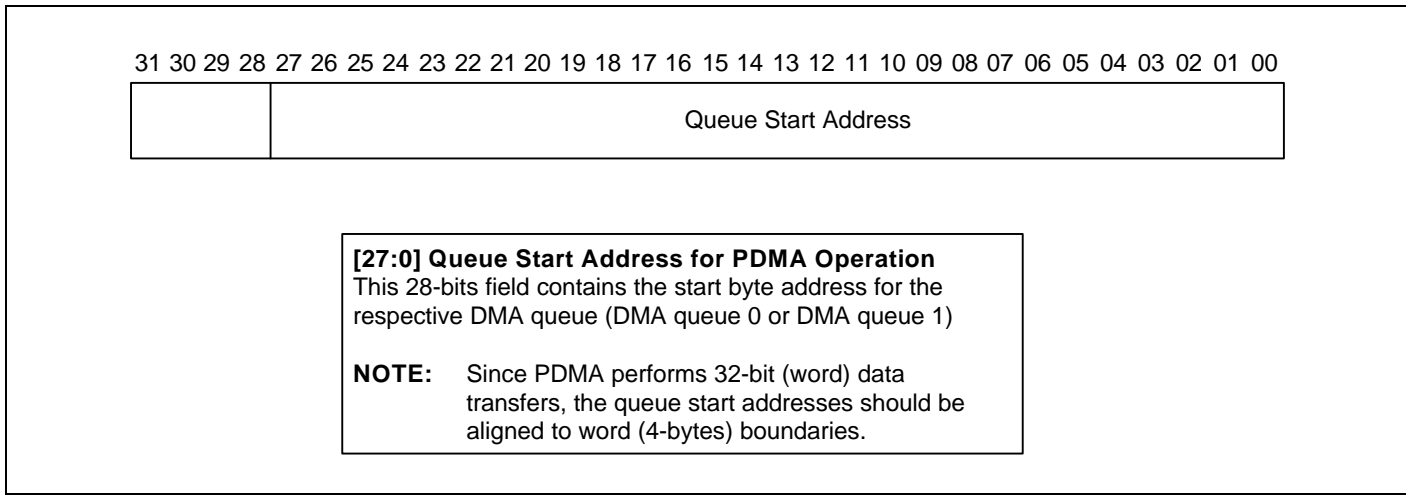


Figure 25-12. Queue 0/1 Start Address Registers (QSAR0, QSAR1)

QUEUE 0/1 TRANSFER COUNT REGISTERS

The values written to the two queue transfer count registers, QTCR0 and QTCR1, define the transfer count in word (32-bit) units when the DMA operation starts for the corresponding queue.

Registers	Offset Address	R/W	Description	Reset Value
QTCR0	0xa020	R/W	PDMA queue 0 transfer count register	0x000000
QTCR1	0xa028	R/W	PDMA queue 1 transfer count register	0x000000

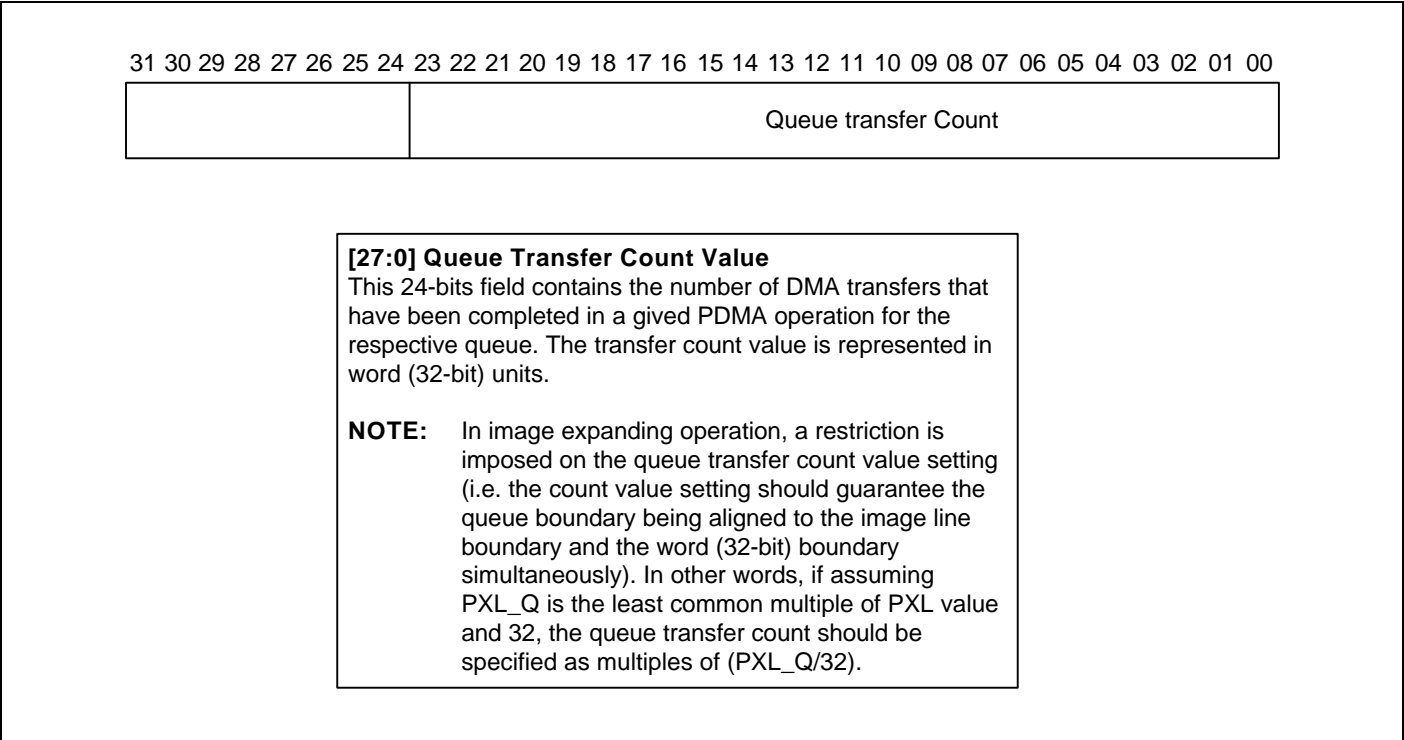


Figure 25-13. Queue 0/1 Transfer Count Registers (QTCR0, QTCR1)

F-q LENS COMPENSATION CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
FTCON	0xa02c	R/W	F-θ control register	0x0

- [0]

F-θ Enable

Disable if 0. Enable if 1.
- [1]

CPU Access Enable

0: F-θ compensation block accesses the F-θ table memory (read/write)
1: CPU access.
To configure a table, this bit must be set to 1 before CPU write.
- [2]

Clock Selection

Select divided clock.
0: MCLK*2
1: External video clock

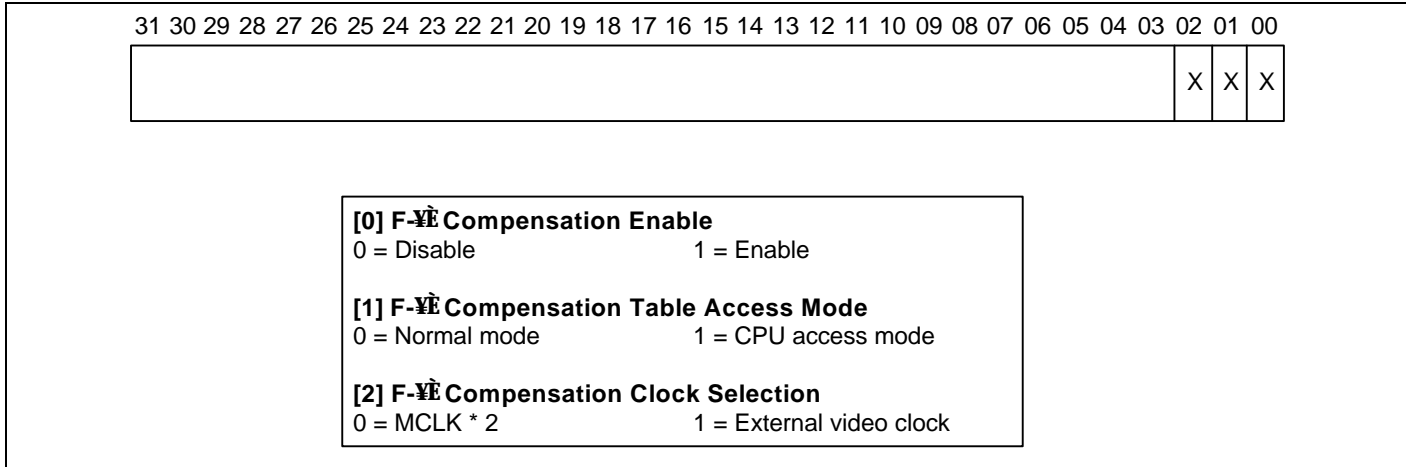


Figure 25-14. F-q Compensation Control Register (FTCON)

F-q COMPENSATION TABLE START ADDRESS

This is the start address for accessing the table (read/write, normal access mode, CPU access mode, etc.).

If you wish to access a different access during operation, you must change this register value.

If you access after writing 00, the F-θ compensation block accesses from 00 in order. If you write 20h to this register, it accesses from 20h.

Register	Offset Address	R/W	Description	Reset Value
FSADDR	0xa030	R/W	F-θ Table start address	0x00

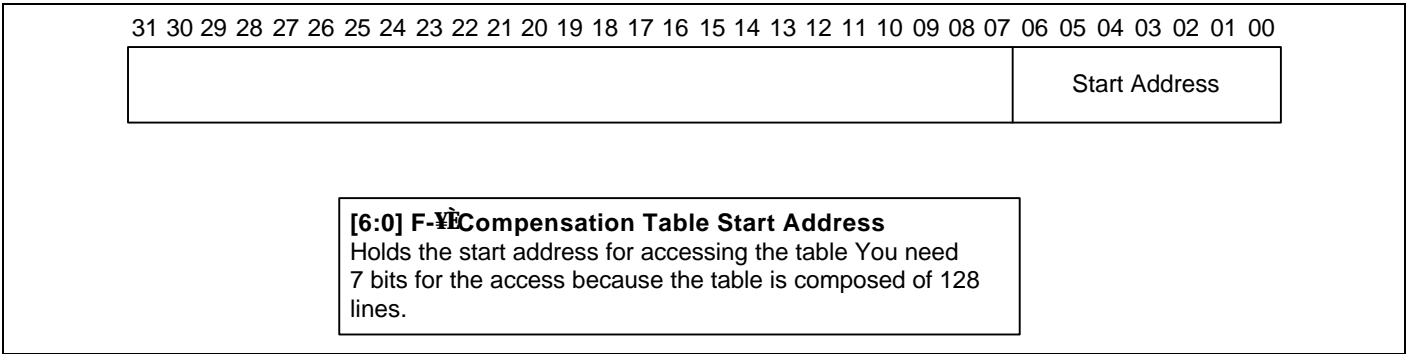


Figure 25-15. F-q Compensation Table Start Address (FSADDR)

F-q COMPENSATION TABLE DATA REGISTER

The CPU reads/writes this register when accessing the F-θ compensation Table.

Register	Offset Address	R/W	Description	Reset Value
FDATA	0xa034	R/W	F-θ compensation data register	0xeffb

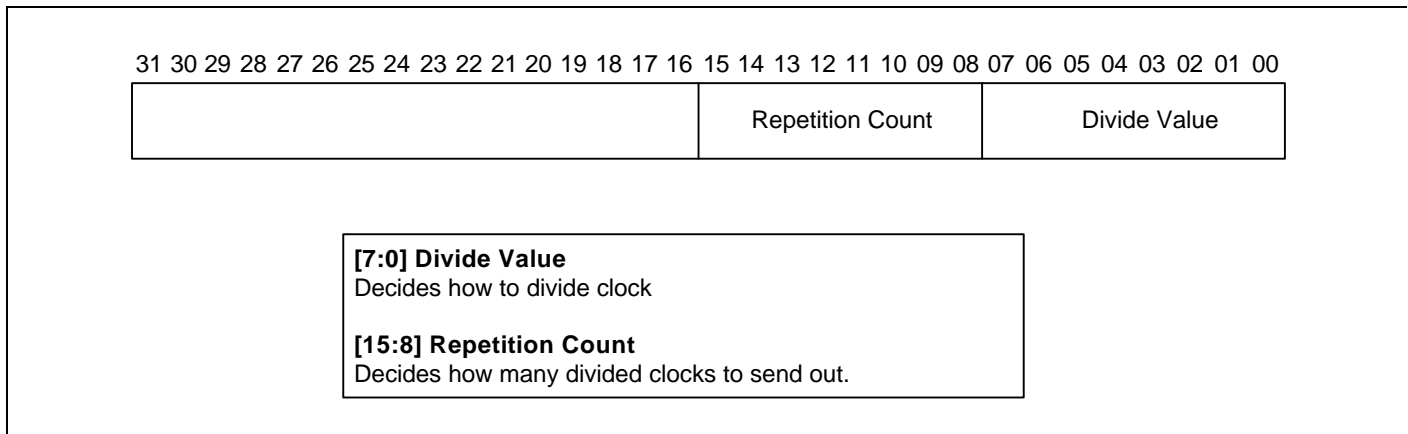


Figure 25-16. F-q Compensation Table Data Register (FDATA)

TONER COUNTER SETTING REGISTER

Register	Offset Address	R/W	Description	Reset Value
TCVAL	0xa038	R/W	Toner counter setting value	0x00000000

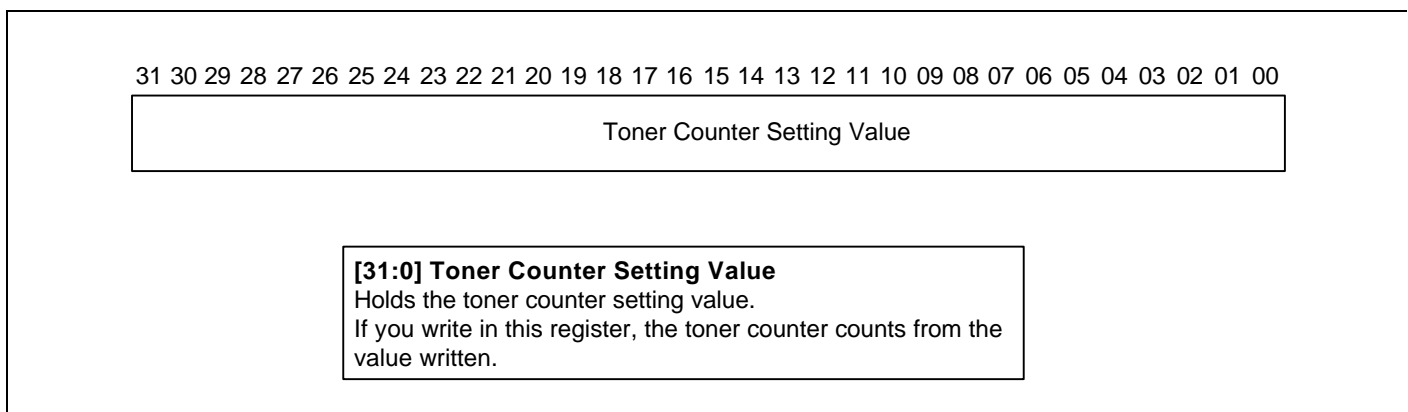


Figure 25-17. Toner Counter Setting Register (TCVAL)

TONER COUNT REGISTER

Register	Offset Address	R/W	Description	Reset Value
TNCNT	0xa03c	R	Toner count value register	0x00000000

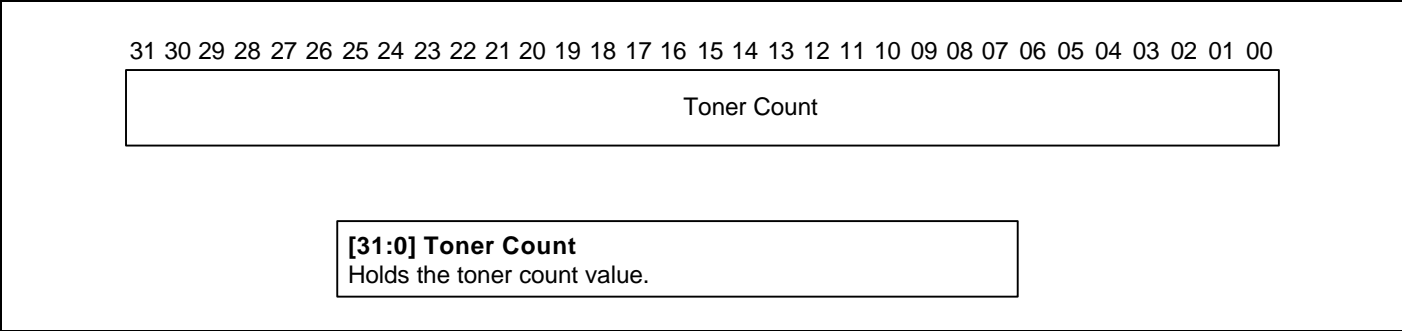


Figure 25-18. Toner Count Register (TNCNT)

Test Pattern Duration

Register	Offset Address	R/W	Description	Reset Value
TPVAL	0xa040	R/W	Test pattern duration value register	0x00

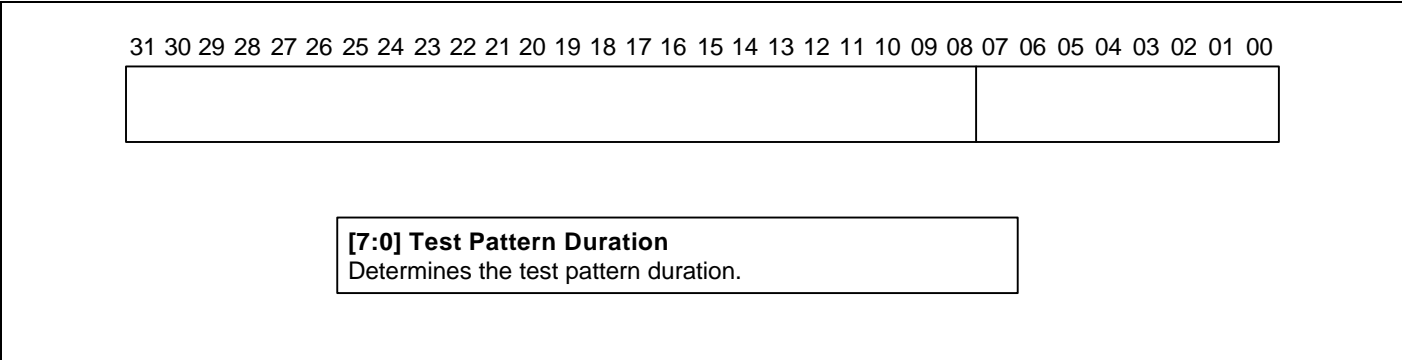


Figure 25-19. Test Pattern Duration (TPVAL)

Test Pattern Width

Register	Offset Address	R/W	Description	Reset Value
TPON	0xa044	R/W	Test pattern width register	0x00

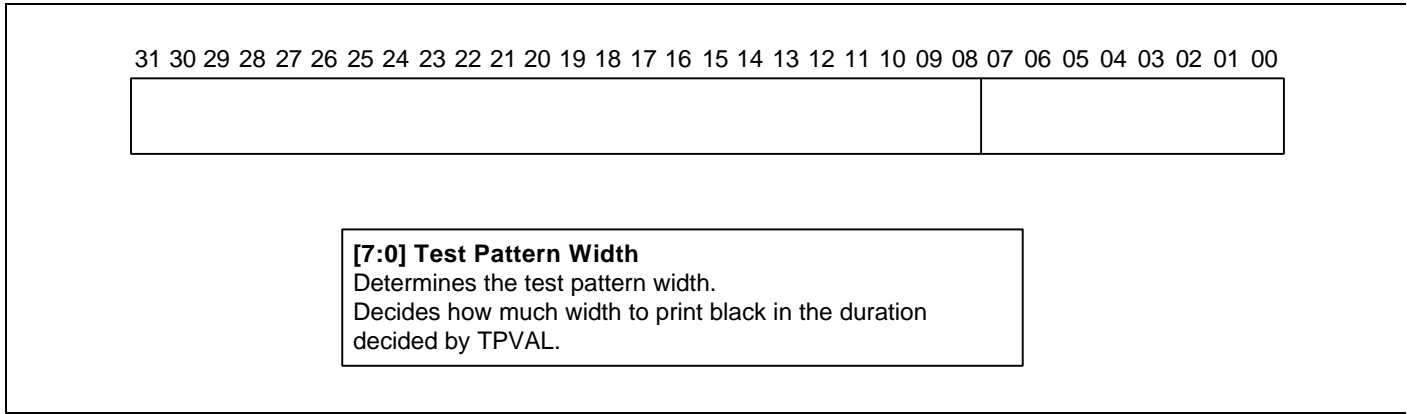


Figure 25-20. Test Pattern Width (TPON)

26

VARIABLE IMAGE SCALING

OVERVIEW

The VIS unit can support the variable-ratio image scaling operation. In other words, the factor of image expansion can be in either integer or fraction. For example, it supports image scaling with ratios $3/2$, $5/4$, 2 , $13/5$ and so on. To implement this operation, five registers are involved in this unit, out of which two size registers, SrcSize and DstSize, specify the scanline sizes of the source image and destination image; two data registers, SrcReg and DstReg, are used to contain the scanline data of the input source image and the scaled scanline data of the output destination image; and one status register, VISSR, indicates the operation status during VIS running. The image scaling ratio can be determined by comparing the values in two data size registers. For example, if the source size register is set to 4 and the destination size register is set to 5, then the image scaling ratio is $5/4$.

For integral-ratio image scaling, each pixel of input source image scanline is replicated by hardware according to the specified scaling ratio to generate the destination image scanline output. This operation is the same as the image expander's operation mentioned before, except that in this unit the expanding factor can be an arbitrary integer. However, for fractional-ratio image scaling, hardware performs the pixel replication following a particular algorithm. The hardware has to decide which pixel in input source image scanline should be replicated or how many times it should be replicated, with specified fractional scaling ratio.

ALGORITHM

The VIS algorithm is given out in the form of a C-program, as shown in Figure 26-1.

```

/*****
/*  Variable Descriptions:
/*      Dst_Pixel_IDx * Pixel position in destination data register
/*      Src_Pixel_IDx * Pixel position in source data register
/*      Dst_Size * Destination size register's setting value
/*      Src_Size * Source size register's setting value
/*      DstReg * Destination data register
/*      SrcReg * Source data register
*****/

VIS_Operation()
{
    Frac = 0;
    Dst_Pixel_IDx = 0;
    Src_Pixel_IDx = 0;

    for (i = 0; i < Src_Size; i++)
    {
        Frac = Frac + Dst_Size;
        while (Frac >= Src_Size)
        {
            Frac = Frac - Src_Size;
            DstReg[Dst_Pixel_IDx] = SrcReg[Src_Pixel_IDx];
            Dst_Pixel_IDx ++;
        }
        Src_Pixel_IDx ++;
    }
}

```

Figure 26-1. VIS Algorithm Description

EXAMPLE OF VIS OPERATION

To carry out the VIS operation, S/W should run the following steps:

- Set the control register, FUNCON1, as zero to select the VIS operation;
- Set the size registers, SrcSize and DstSize, to specify the scaling ratio;
- Write source image data to source data register (SrcReg);
- Check the read-request bit in status register (VISSR[0]), and read the scaled image data from destination data register (DstReg) once the read-request bit is one; repeat this step until all scaled data are read out.
- If more data is to be processed, check the write-request bit in status register (VISSR[1]) and repeat steps 2-4 once the write-request bit is one.

Inside the VIS unit, hardware performs the image data replication automatically after obtaining the source image data from SrcReg according to the algorithm described above, and outputs the scaled image data to DstReg. Figure 26-2 shows some examples for VIS's internal image data replication process.

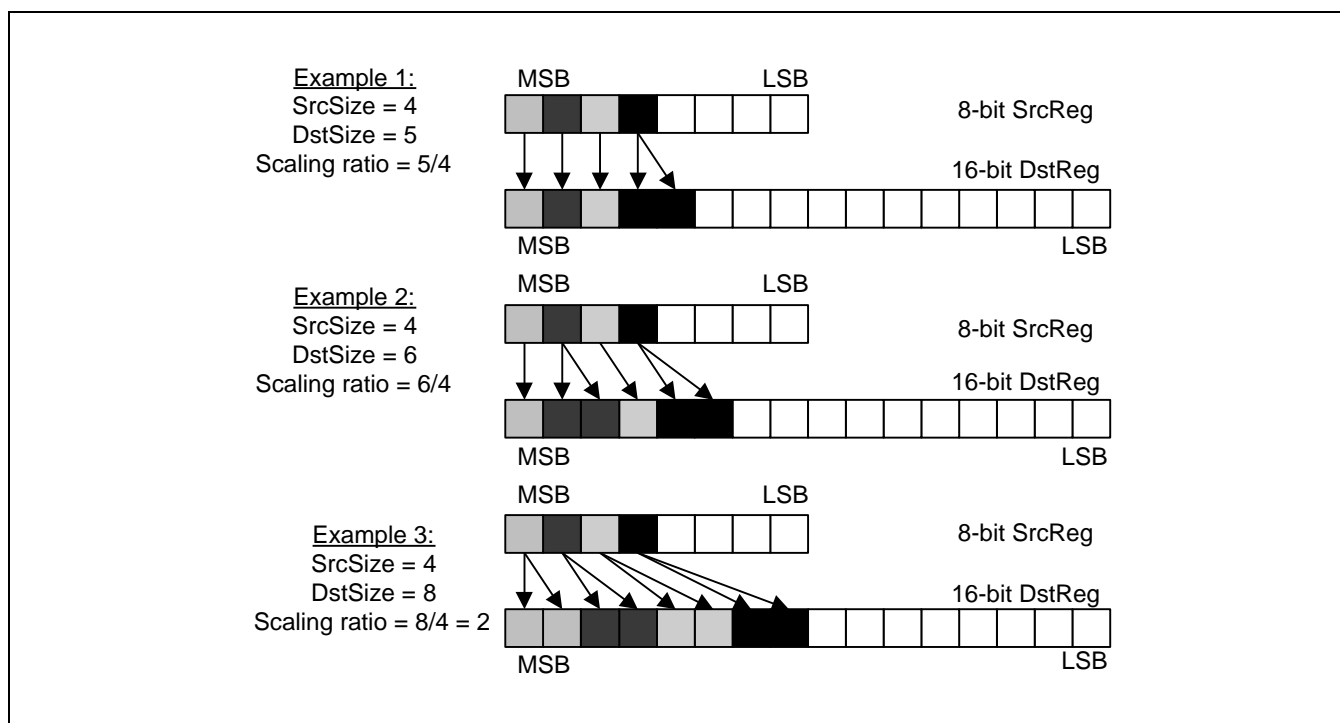


Figure 26-2. Examples of VIS's Internal Operation

HALFTONING

The halftoning unit performs the operation to convert the gray-level image into a bi-value halftone image. To support the PCL6.0 protocol, the input gray-level image, in which the pixel's gray level is 8-bit scaled and each pixel corresponds 8-bit scale data, should be converted to halftone image suitable to be printed. For halftone image, the image gray level is represented by the density of the black pixels (i.e. each pixel in this kind of image only corresponds one bit and may represent as white (zero) or black (one) only).

In this unit, the conversion from gray-level image to halftone image is implemented by hardware based on a comparison algorithm. To generate the halftone image, each pixel data (8-bit) of the gray-level image is compared with an 8-bit reference data (i.e. the threshold value), and a one-bit halftone image pixel value is output according to the comparison result.

To support this operation, four registers are provided in this unit, in which three 16-bit data register (PixIn, RefIn and HftReg) are used to contain the source image (gray-level image) pixels' data, reference data (threshold values) and the halftone data, and a control register (VISCON) is used to initialize/enable the half toning operation and select the 'dot mode' to be introduced below.

Since the data register is 16-bit while the input image pixel data is 8-bit, two pixels are input and processed at the same time. To carry out the half toning operation, S/W runs in the following steps:

- Set the control register's 0 bit, VISCON[0], as one to enable the VIS operation, and set the VISCON[1] to select dot mode.
- Write two pixel thresholds to RefIn register's lower 8-bit and upper 8-bit to provide two pixel reference data;
- Write two pixel data of source image to PixIn register's lower 8-bit and upper 8-bit to compare with reference.
- Repeat steps 2-3 seven times, and then read the HftReg to obtain the 16-bit output, i.e. the 16 pixels's data of halftone image.
- Repeat steps 2-4 until all pixels of source image are processed.

The half toning algorithm is described in Figure 26-3, in which two kinds of dot modes are included. The dot mode selection in half toning operation depends on the setting of VISCON[1].

<u>Dot mode 0:</u>	
if (source_pixel_data > reference_data)	halftone_pixel_data = 0;
else	halftone_pixel_data = 1;
<u>Dot mode 1:</u>	
if (source_pixel_data > reference_data)	halftone_pixel_data = 1;
else	halftone_pixel_data = 0;

Figure 26-3. Half toning Algorithm Description

SPECIAL REGISTER

VIS Status Register

The VIS status register, VISSR, is a read-only register which is used to monitor the status of VIS operation.

Register	Offset Address	R/W	Description	Reset Value
VISSR	0xa800	R	VIS status register	0x0

- [0]

Read request

VISSR[0] is automatically set to "1" whenever the scaled image data has been prepared in DstReg. When it is "1", it indicates that you can read the scaled results from DstReg.
- [1]

Write request

VISSR[1] is automatically set to "1" whenever the VIS operation for all the data in SrcReg has been completed. When it is "1", it indicates that you can write the next source data to SrcReg.
- [2]

Busy flag

VISSR[2] is automatically set to "1" whenever VIS operation starts; and when it is "0" VIS is in an idle state.

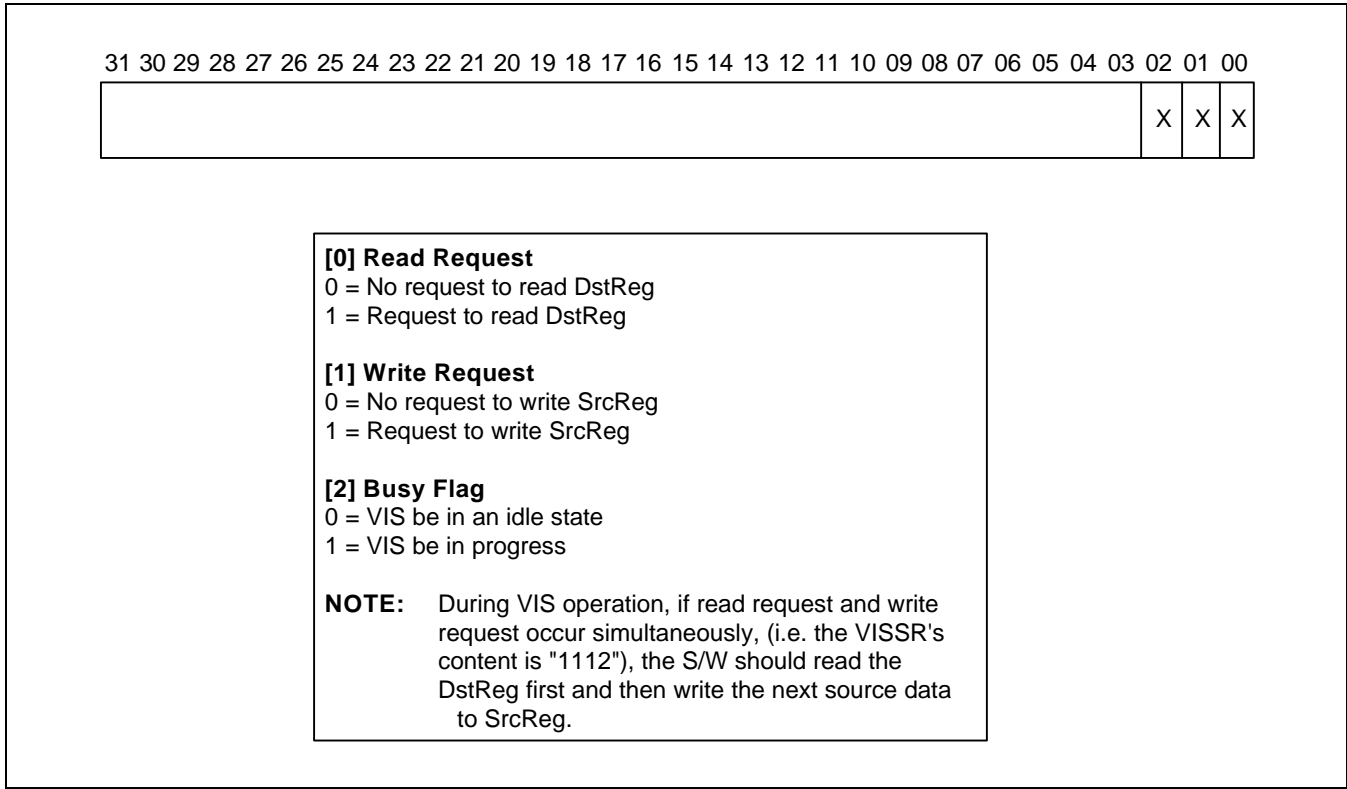


Figure 26-4. VIS Status Register (VISSR)

VIS Control Register

The VIS control register, VISCON, controls the VIS/half toning operation. Two bits in this register are used to respectively enable the VIS/half toning operation and select the algorithm for half toning operation.

Register	Offset Address	R/W	Description	Reset Value
VISCON	0xa804	R/W	VIS control register	0x0

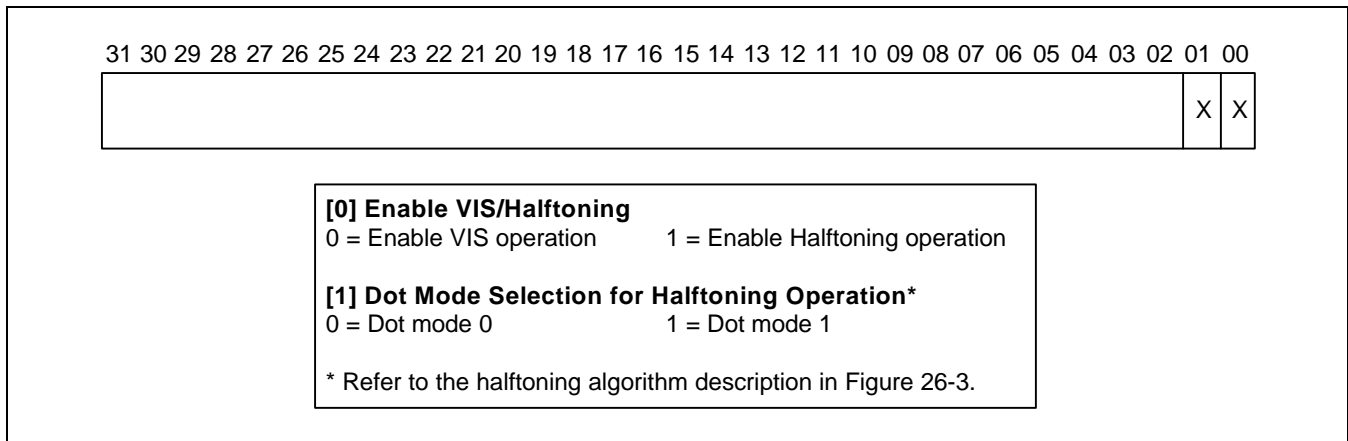


Figure 26-5. VIS Control Register (VISCON)

VIS Data Size Registers

Two VIS data size registers, SrcSize and DstSize, are used to define the image data length before and after the VIS process (i.e. the input source image data length and the output destination image data length). The image scaling ratio can be determined by these two registers* contents, i.e.

$$\text{image_scaling_ratio} = \text{DstSize_value} / \text{SrcSize_value}$$

Registers	Offset Address	R/W	Description	Reset Value
DstSize	0xa808	R/W	Destination image data size register	0xFFFF
SrcSize	0xa80c	R/W	Source image data size register	0xFFFF

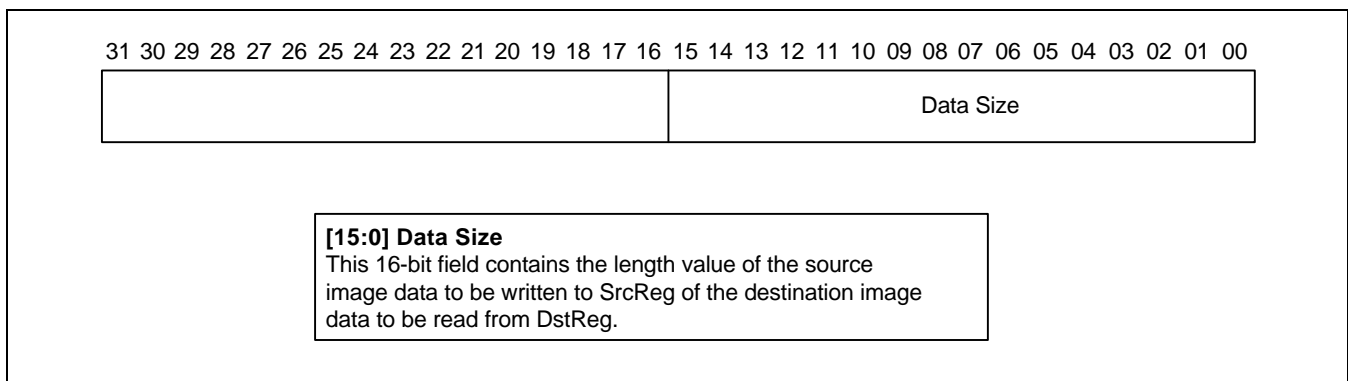


Figure 26-6. VIS Data Size Registers (DstSize, SrcSize)

VIS Data Registers

Two VIS data registers, SrcReg and DstReg, respectively contain the input source image data before the VIS process and the output destination image data after the VIS process. The SrcReg is an 8-bit register and the DstReg is a 16-bit register.

Registers	Offset Address	R/W	Description	Reset Value
SrcReg	0xa810	R/W	Source image data register	0xXX
DstReg	0xa814	R	Destination image data register	0xFFFF

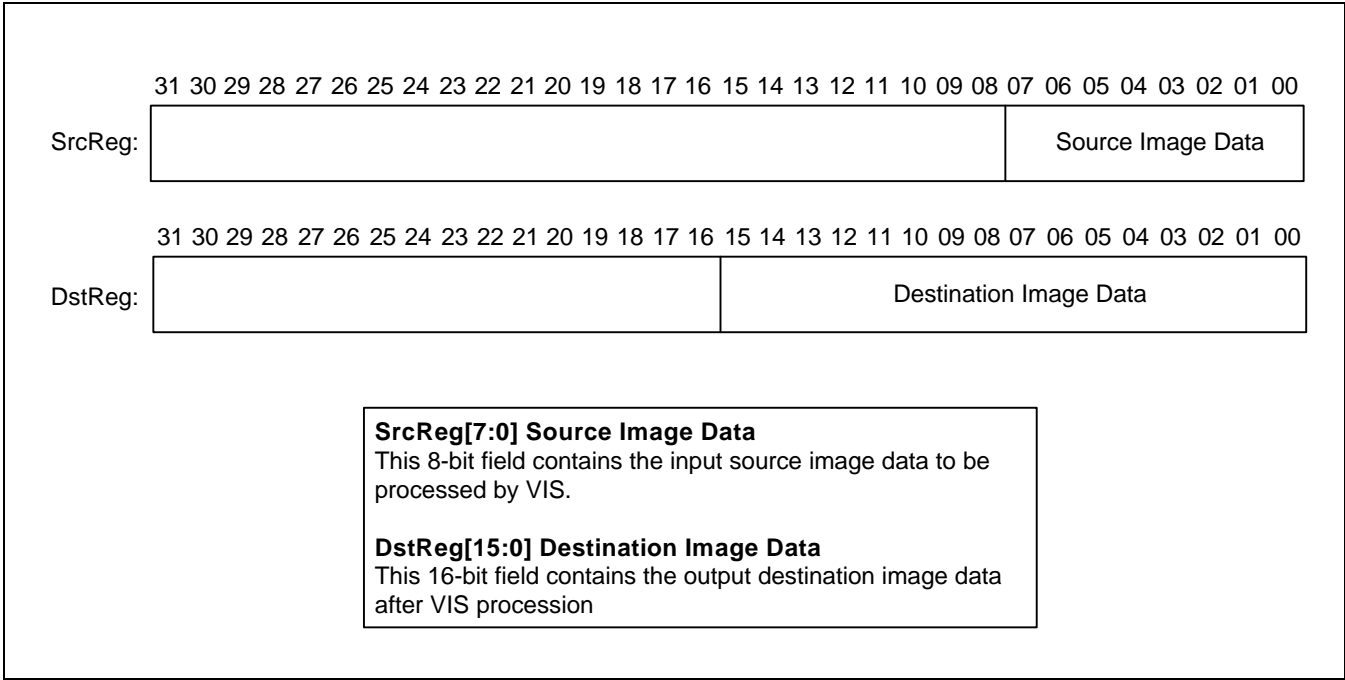


Figure 26-7. VIS Data Registers (SrcReg, DstReg)

Half toner Data Registers

Three half toner data registers, RefIn/PixIn and HftReg, respectively contain the input reference/source pixel data before the half toning process and the output the halftone data after the half toning process.

Registers	Offset Address	R/W	Description	Reset Value
RefIn	0xa818	R/W	Reference data register	0xFFFF
PixIn	0xa81c	R/W	Source image pixel data register	0xFFFF
HftReg	0xa820	R	Halftone image data register	0xFFFF

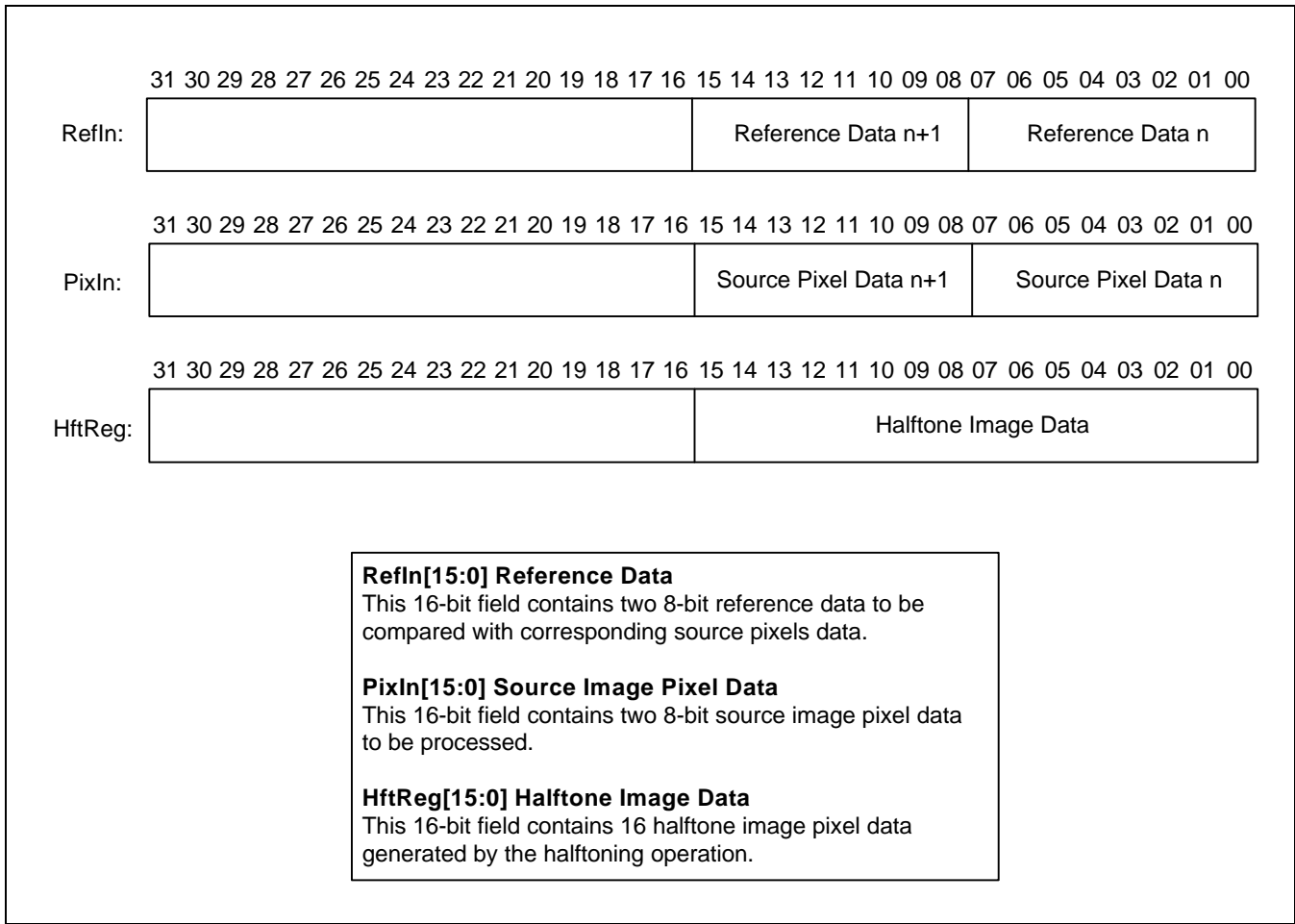


Figure 26-7. VIS Data Registers (SrcReg, DstReg)

27

PWM TIMER CONTROL

INTRODUCTION

The PWM control is composed of the following: one 8-bit pre-scaler, one 16-bit down_counter, two 4-bit pre-scalers, and 16-bit down_counter.

- The 16-bit counter is either enabled (RUN) or disabled (STOP) according to the each control register's bit selection.
- The PWM output signal and the on/off time within its period are decided according to the Cycle_Time base value and On-Time base value. If the On-Time base value is the same or larger than the Cycle_Time base value when the counter is enabled, the PWM output signal maintains On status.
- If a new cycle time or on time value is written, the PWM output is generated according to the modified value starting from the next cycle.
 - $\text{PWM_Counter_Clock} = \text{MCLK} / (\text{pre-scaler value} + 1)$
 - $\text{Cycle time pulse width} = (\text{Cycle time value} + 1) / \text{PWM_Counter_Clock}$
 - $\text{On time pulse width} = (\text{On time value} + 1) / \text{PWM_Counter_Clock}$

MAIN INPUT/OUTPUT SIGNALS

Output

- PWM_OUT0: PWM0 timer output signal
- PWM_OUT1: PWM1 timer output signal
- PWM_OUT2: PWM2 timer output signal

SPECIAL FUNCTION REGISTER

PWM Control Register

Register	Offset Address	R/W	Description	Reset Value
PWMCONR	0xe000	R/W	PWM_CON control register	0x0

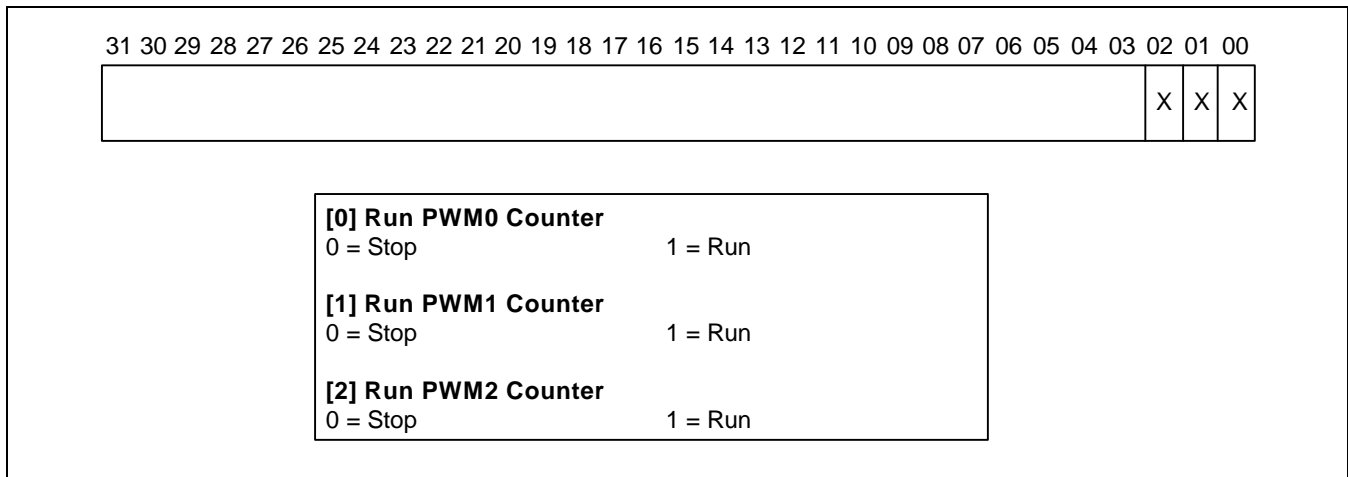


Figure 27-1. PWM_CON Control Register

PWM Counter Pre-Scaler

Register	Offset Address	R/W	Description	Reset Value
PWM_PRSC	0xe004	R/W	PWM Pre-Scaler counter base value register	0x00000000

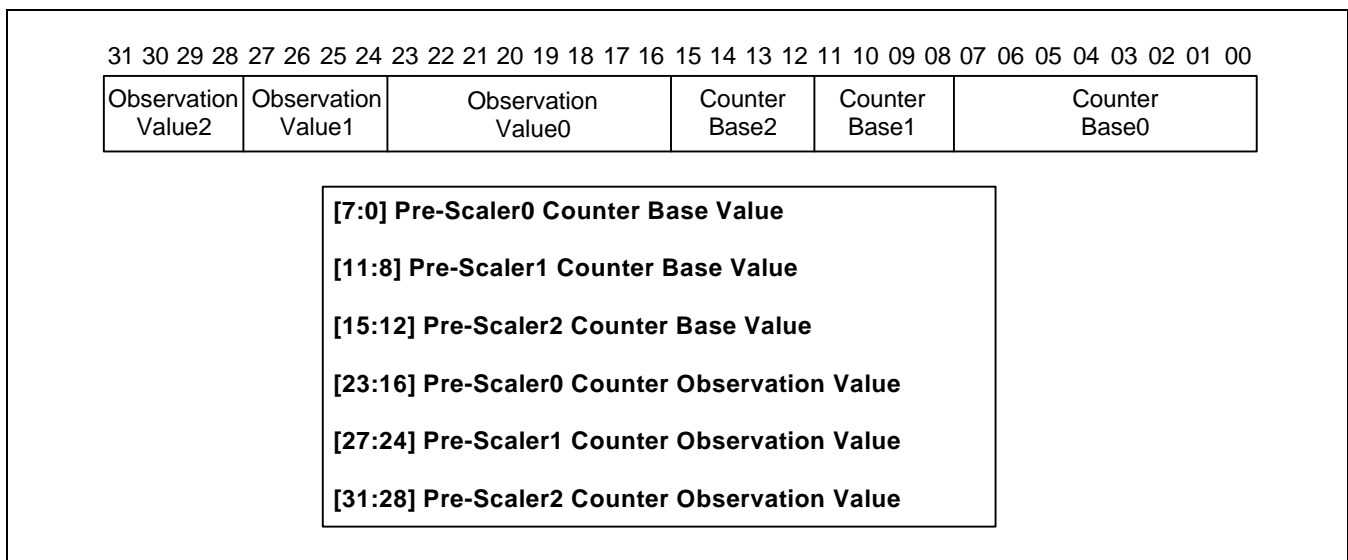
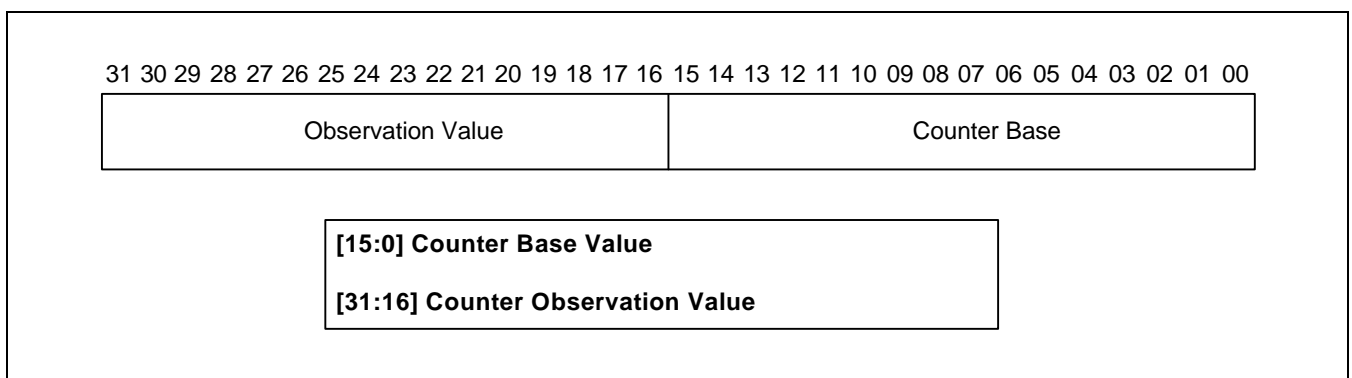
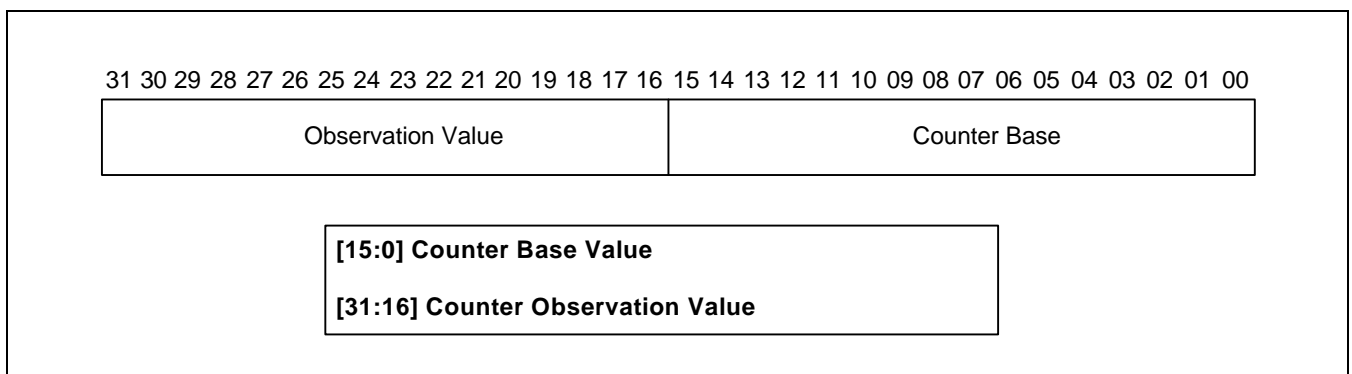


Figure 27-2. PWM Pre-Scaler Counter Base/Observation Register

PWM Cycle/On Time Base & Observation Register

Registers	Offset Address	R/W	Description	Reset Value
PWM_CYT0	0xe008	R/W	PWM0 cycle time & observation register	0x00000000
PWM_ONTO	0xe00c	R/W	PWM0 on time & observation register	0x00000000
PWM_CYT1	0xe010	R/W	PWM1 cycle time & observation register	0x00000000
PWM_ONTO1	0xe014	R/W	PWM1 on time & observation register	0x00000000
PWM_CYT2	0xe018	R/W	PWM2 cycle time & observation register	0x00000000
PWM_ONTO2	0xe01c	R/W	PWM2 on time & observation register	0x00000000

**Figure 27-3. PWM Cycle Time Base/Observation Register****Figure 27-4. PWM On Time Base/Observation Register**

Caution

- $\text{PWM_Counter_Clock} = \text{MCLK} / (\text{pre-scaler value} + 1)$
- $\text{Cycle time pulse width} = (\text{cycle time value} + 1) / \text{PWM_Counter_Clock}$
- $\text{On time pulse width} = (\text{on time value} + 1) / \text{PWM_Counter_Clock}$

*** PWM Timer Setting Process**

- Write PWM_Counter Pre-Scaler value
- Write PWM_Cycle time value
- Write PWM_On time value
- RUN PWM_Timer
- Write Next PWM_Cycle time value or PWM_On time value
(When you change the PWM_Cycle time or the PWM_On time value)

28

MECHANICAL DATA

PACKAGE DIMENSIONS

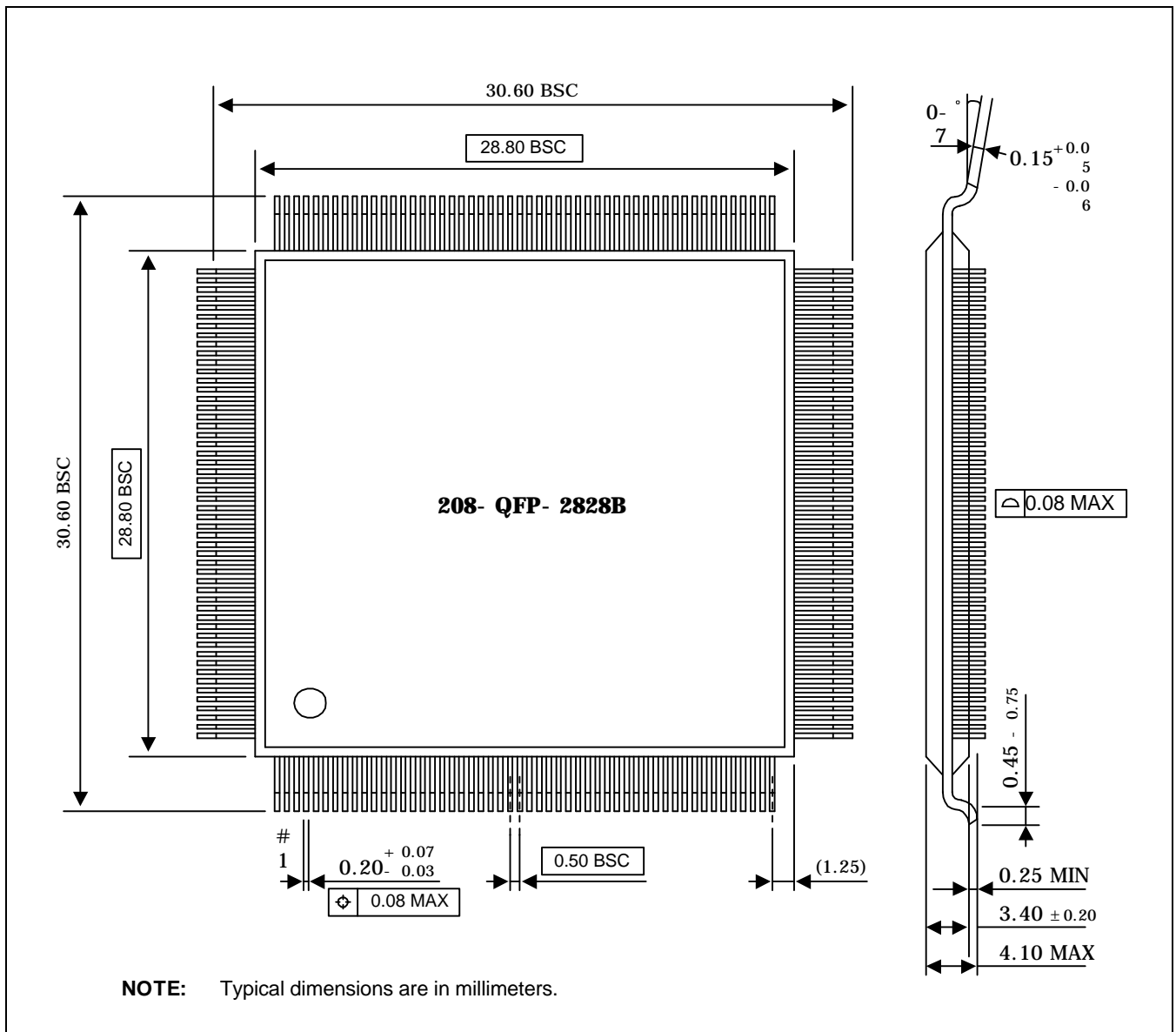


Figure 28-1. 208-QFP-2828 Package Dimensions

29

EVALUATION BOARD

INTRODUCTION

KS32C65100 evaluation board is a platform that is suitable for code development and exploration of KS32C65100. It supports various memory devices such as DRAM , SRAM, EPROM, and Flash. Using the embedded ICE interface, you can debug the KS32C65100 directly.

SYSTEM REQUIREMENTS

- Host computer: IBM compatible PC
- Evaluation board of KS32C65100
- DC power supply with the following outputs: +5V at 0.5 A
- Parallel Cable (25-pin)
- Serial cable (9-pin)

BOARD COMPONENTS

The arrangement of major components on the board is shown in Figure 29-1. The major components include:

EPROM/Flash Memory	There are two sockets, U13 and U14, which will accept 8-bit FLASH or EPROM with 64 K size for lower byte (U13) and upper byte (U14) data access, respectively. The two sockets finally form 64 K x 16-bit ROM bank. You can control the memory type by setting the jumper JP15 and JP16.
SRAM	Two sockets, U18 and U19, are supplied for SRAM memory bank with 128 K x 16-bit size. The U18 and the U19 will accept the 128 K 8-bit SRAM for lower byte data and upper byte data, respectively.
DRAM	Two sockets, U15 and U16, are supplied for DRAM memory bank with 1M x 16-bit size.
Parallel Port	One parallel port (PRINT) is supplied to support parallel data communication between the host PC and the evaluation board.
Serial Ports	Two 9-pin serial ports (SERIAL-1 and SERIAL-2) are supplied for serial data communication between the host PC and the evaluation board. You can control the SIO1, SIO2 by setting the jumper JP3 and JP5.
JTAG Port	One 14-pin JTAG port (CON4) is supplied to connect with the Embedded ICE Unit.
Expansion Connectors	Three 50-pin connectors (U11, U12, U17) are supplied for system expansion. They contain board data bus, address bus, external memory bank/device control, and external master control signals.
Buttons	Five buttons are supplied on the board. One button (S5) is for system reset and the others (S1-S4, S7) are reserved for external intervention during system running. Depending on the setting of jumpers (JP10-JP14), the S1-S4, S7 are optionally connected to four KS32C65100's general purpose Input pins (GIP3-GIP7) and the external intervention can be detected and handled by S/W.
LED Indicators	Five LEDs are supplied on the KS32C65100 board. One LED (LD9, adjacent to the power connector) is for board power indication and the rest (LD2, LD5-LD8) is reserved for other status indication. LD2, LD5-LD8 are optionally connected with four KS32C65100's general purpose Output pins (GOPA5, GOPA7-GOPA10). Depending on the setting of jumpers (JP4, JP6-JP9), their on/off status can be controlled by S/W.

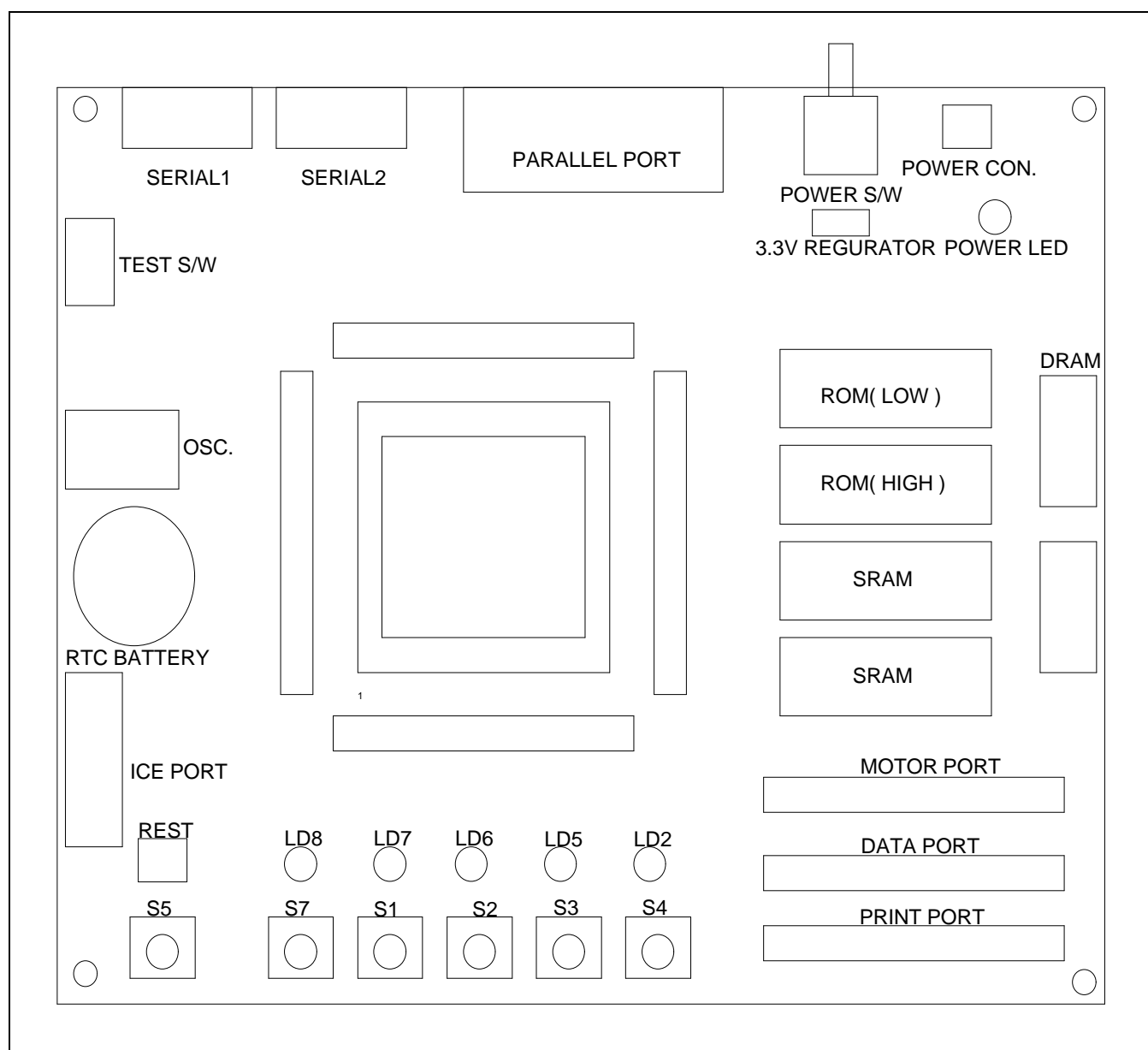


Figure 29-1. Evaluation Board

BOOTING SYSTEM

After power is turned on, the Boot Code is activated automatically. The Boot Code then performs system initialization and configuration. Once this procedure is completed, the four LEDs (LD1-LD4) on the bottom of the board should light on together. At the same time, a message appears on the PC, which shows that the system is waiting for program downloading.

If four LEDs fail to light on, the board is either faulty or incorrectly powered. If the LEDs light on but no message or some strange symbols appear on the communication window activated on the host PC, you should check if the parameter setting for the communication window (such as, the Hyper Terminal) is matched to the relative setting for board, such as baud rate, parity, stop bit setting and so on

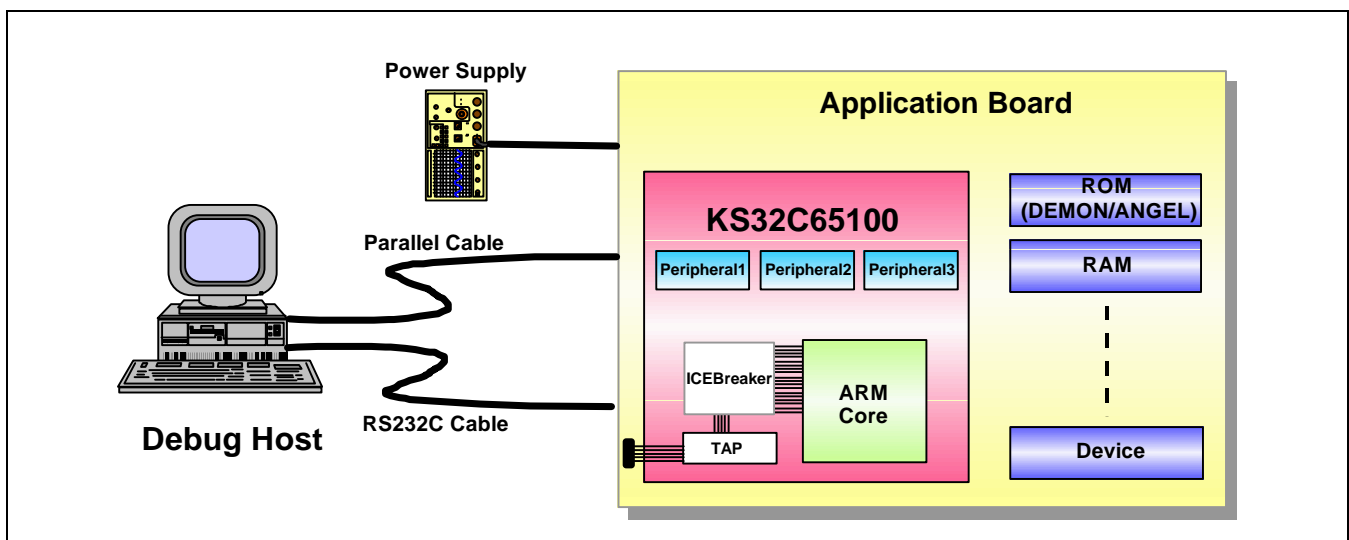


Figure 29-2. Connection to Host PC

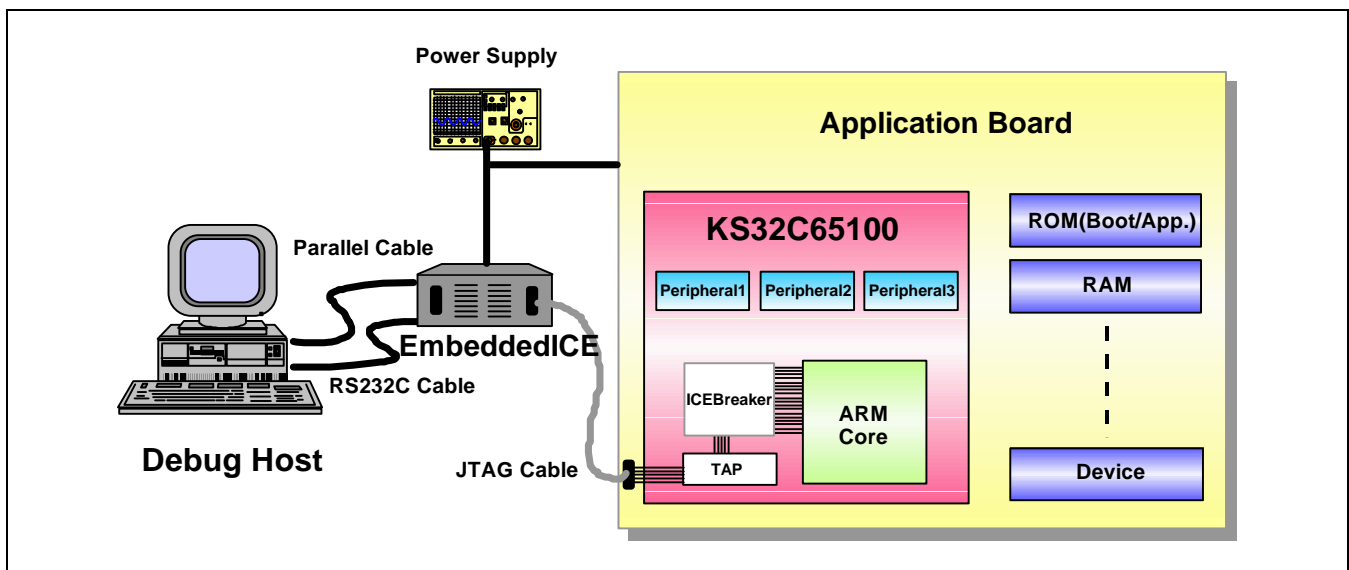


Figure 29-3. Connection to Embedded ICE

EMBEDDED ICE UNIT INSTALLATION

EMBEDDED ICE UNIT

The Embedded ICE Unit can also be connected with the KS32C65100 evaluation board as a debugging system for software applications development. Embedded ICE is a JTAG-based, non-intrusive, debugging system for ARM-based controllers or processors. Embedded ICE provides the interface between a debugger and the ARM-based controller development board.

To use the Embedded ICE, the following additional equipment are required:

- Embedded ICE Interface Unit
- 14-way ribbon cable
- 9-pin RS232 cable
- 25-pin parallel cable (optional)
- 7–9 V at 500mA DC power supply

CONNECTING KS32C65100 EVALUATION BOARD AND PC

The Embedded ICE Unit should be connected to the KS32C65100 evaluation board's JTAG Port (CN1) via a 14-way cable, and to the host PC via a 9-pin RS232 serial cable. A parallel cable can optionally be connected between the 25-pin parallel port connector on the Embedded ICE interface and the printer port on the host PC. Using the parallel cable can speed up the code download.

To power on the Embedded ICE interface, 7–9 V DC power supply is required. The system connection with Embedded ICE is shown in Figure 29-3.

POWERING UP THE BOARD AND EMBEDDED ICE

We recommend that you power on the evaluation board before the Embedded ICE is powered on. In this way, the system initialization and memory configuration for KS32C65100 evaluation board performed by the Boot Code can be completed first. Otherwise, it may cause the failure of code download via Embedded ICE.

DEBUG APPLICATION WITH EMBEDDED ICE

1. Install ARM Tool kit for Windows.
2. Run the "Hyper Terminal" in host PC.
3. Configure the serial port settings of the "Hyper Terminal" as 38400bps, 8-bit data, no parity and 1 stop bit.
4. Install the evaluation board and Embedded ICE interface as Figure 29-3.
5. Power on the board and Embedded ICE.

Configuring the ARM Windows Debugger

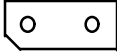





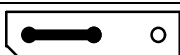
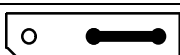
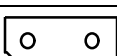
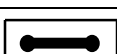
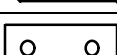
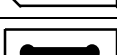
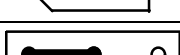

1. Run the ARM Windows Debugger.
2. Select "Options/configure Debugger/Debugger" menu to set "Big" for "Endian" item.
3. Select "Options/configure Debugger/Target" menu to set "Remote_A" for "Target Environment" item.
4. Click "Configure" button in "Options/configure Debugger/Target" menu to open the "Angel Remote Configuration" window. In this configuration window, you select "serial" or "serial/parallel" for "Remote Connection" item, select an appropriate COM port, select an appropriate baud rate for serial line speed, and then click the "OK" button to end the configuration.
5. Click the "OK" button in "Options/configure Debugger" to conclude the debugger configuration.
6. Select "File/Exit" menu to quit the ARM Windows Debugger.

Debugging the application with Embedded ICE

1. Run the ARM Project Manager.
2. Open "Hello.apj" in directory "example/ICEdbg/."
3. Click the "force build" button to build the application.
4. Click the "Debug" button to start the ARM Windows Debugger.
5. Click the "YES" button when you see the message box "Are you sure that you want to start in remote debugging?"
6. After code downloading is completed, type in the following command in the command window :
"ob a:\example\ICEdbg\armsd.in"
7. Then, you can run and debug the application using any functions provided by the Debugger.

SWITCH AND JUMPERS DESCRIPTION

Table 29-1. Jumper Description

Jumper	Status	Description
JP1 (5V _{DD}) JP2 (3V _{DD})		Main power and chip power (5V, 3.3V) are separated.
		Main power and chip power (5V, 3.3V) are connected.
JP3 (SIO TXD)		SERIAL2 and TXD1 are connected
		SERIAL2 and TXD2 are connected
JP5 (SIO RXD)		SERIAL2 and RXD1 are connected
		SERIAL2 and RXD2 are connected
JP17 (CLOCK)		MCLK and Crystal are connected.
		MCLK and Oscillator are connected.
JP4, JP6, JP7, JP8, JP9		GOPA5,GOPA7~10 and LED(LD2,LD5~8) are separated.
		GOPA5,GOPA7~10 and LED(LD2,LD5~8) are connected.
JP10-JP14		GIP3~7 and SWITCH(S1~4,S7) are separated.
		GIP3~7 and SWITCH(S1~4,S7) are connected.
JP19, JP20		Select EPROM (27512) on ROM sockets (U13 & U14).
		Select Flash memory(29EE512) on ROM sockets (U13 & U14).

NOTE: The grayed rows are the default settings of the evaluation board.

Table 29-2. Switch Description

Switch Key	Status	Description
SW1		Main power 5V switch
S8	On	Pin1 ~3 is TEST0~1 ground, Pin 4 ICE TDI ground (When ICE non-connection)
	Off	Pin1 ~3 is TEST0~1 VDD, Pin 4 ICE TDI open (When ICE connection)
S5		Reset switch

NOTE: The grayed rows are the default settings of the evaluation board



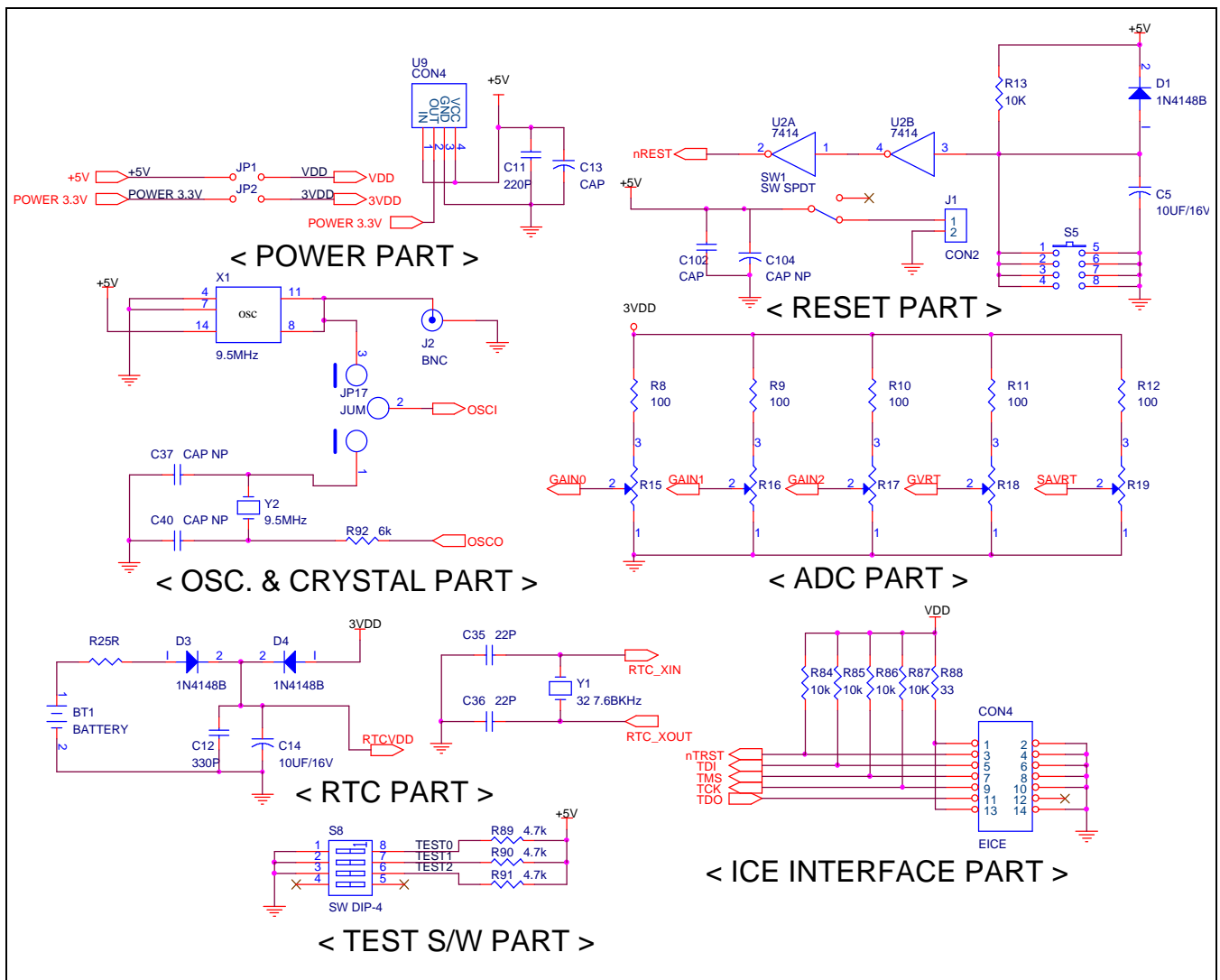


Figure 29-5. Evaluation Board Schematic 2

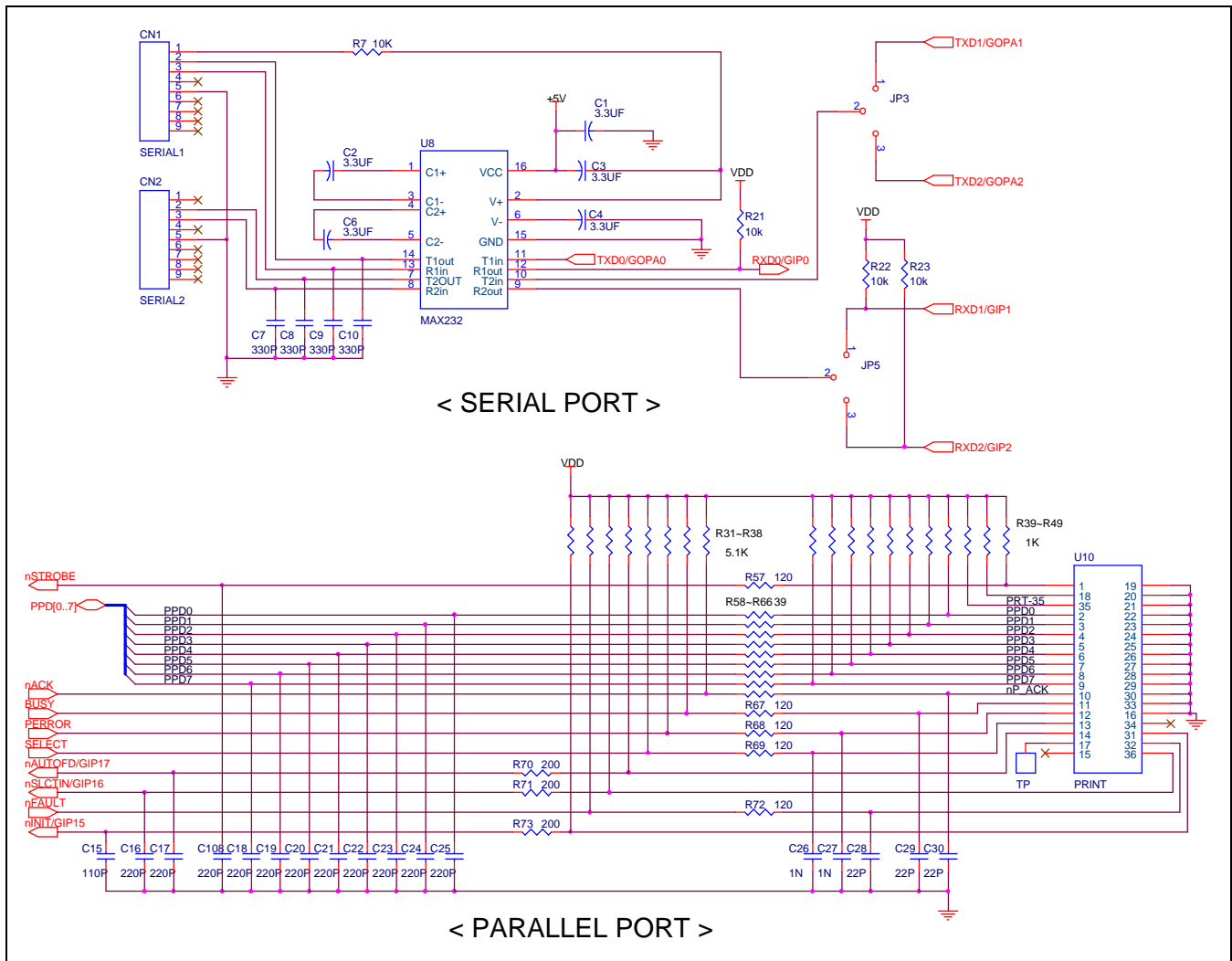
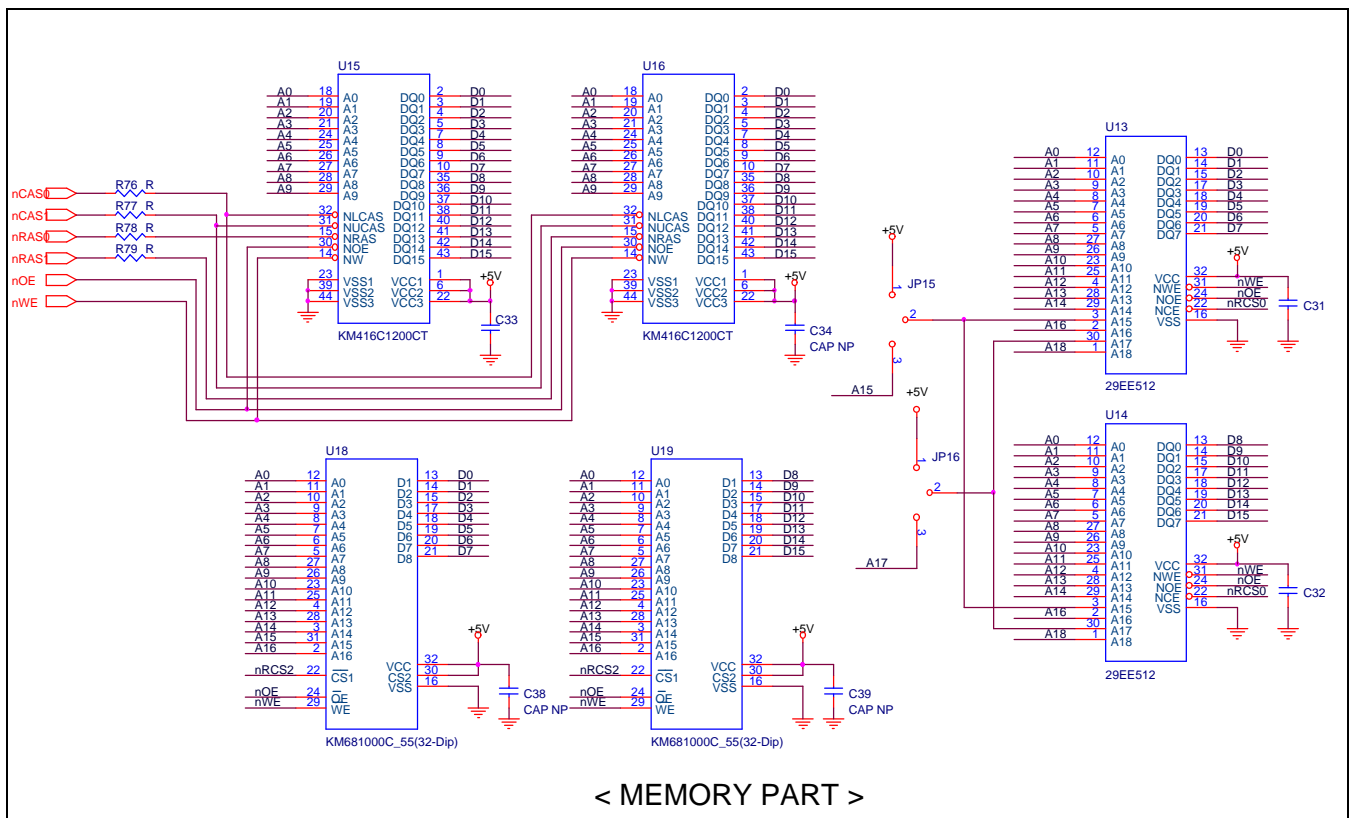


Figure 29-6. Evaluation Board Schematic 3



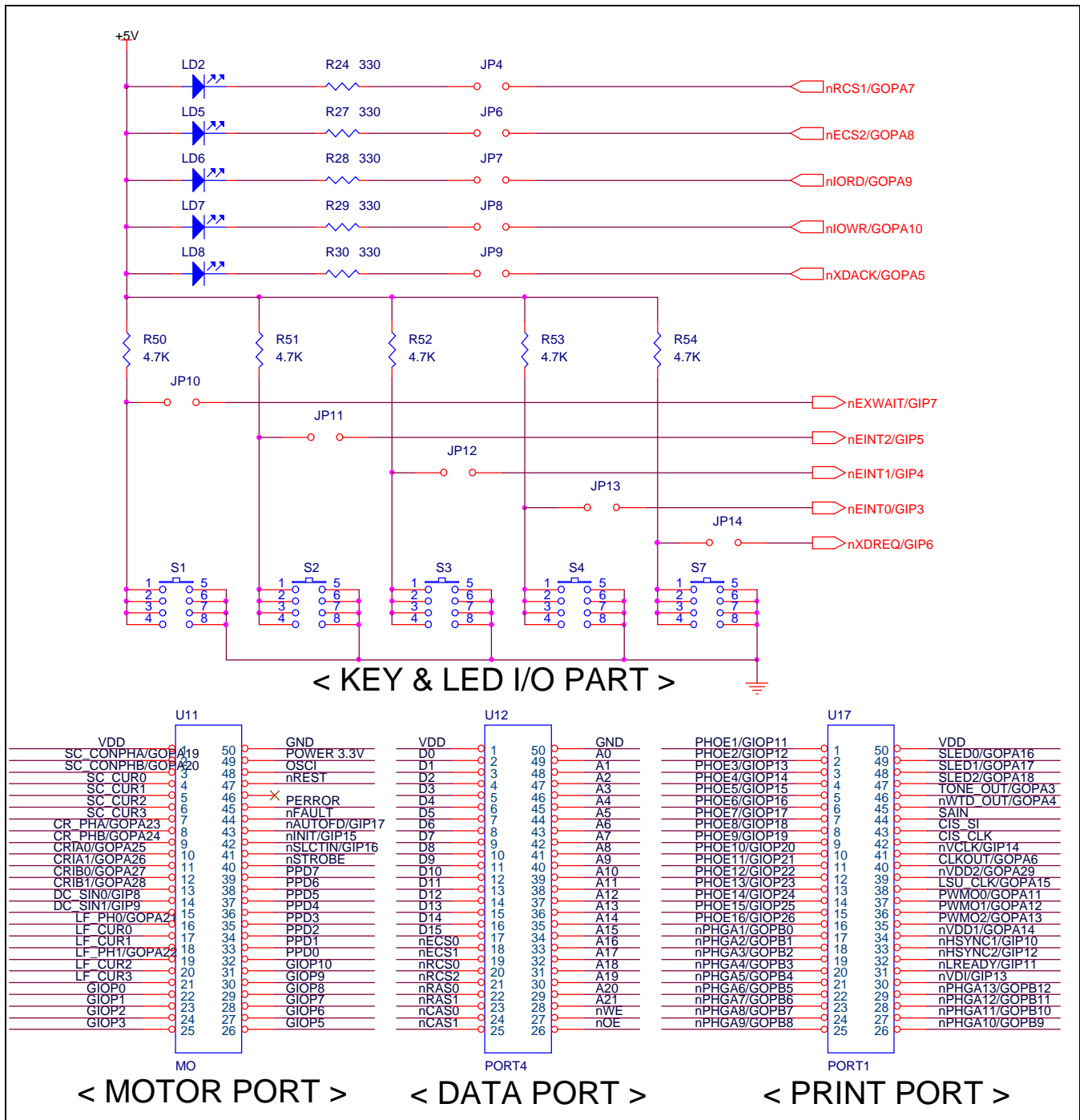


Figure 29-8. Evaluation Board Schematic 5