22-S3-C3410X-062001

USER'S MANUAL

S3C3410X
16-Bit CMOS
Microcontrollers
Revision 2

SAMSUNG ELECTRONICS

NOTIFICATION OF REVISIONS

ORIGINATOR: Samsung Electronics, SOC Development Group, Ki-Heung, South Korea

PRODUCT NAME: S3C3410X RISC Microcontroller

DOCUMENT NAME: S3C3410X User's Manual, Revision 2

DOCUMENT NUMBER: 22-S3-C3410X-06-2001

EFFECTIVE DATE: June, 2001

SUMMARY: As a result of additional product testing and evaluation, to correct the errata and to add more detailed explanations, some specifications published in the S3C3410X User's Manual, Revision 1, have been changed. These changes for S3C3410X microcontroller, which are described in detail in the *Revision Descriptions* section below, are related to the followings:

— Chapter 1. Pin Descriptions
— Chapter 4. EXTCONx, EXTPORT, EXTDATx and Timing Diagrams
— Chapter 5. Cache Disable Operation
— Chapter 7. Port 7 and Port 9 Control Registers
— Chapter 11. Interrupt Priority Register (INTPRIx)
— Chapter 14. Multi-Master IIC-Bus Status Register (IICSTAT)

DIRECTIONS: Please note the changes in your copy (copies) of the S3C3410X User's Manual, Revision 1. Or, simply attach the *Revision Descriptions* of the next page to S3C3410X User's Manual, Revision 1.

REVISION HISTORY

Revision	Date	Remark
0	–	There is no preliminary spec.
1	August, 2000	Reviewed by Gwang-Su Han.
2	June, 2001	Reviewed by Gwang-Su Han.

REVISION DESCRIPTIONS

1. PIN DESCRIPTIONS:

1) Pin descriptions about A[23:0], D[15:0], nCS[7:0] nECS[1:0], nWAIT and nWREXP, are changed and the content of RP[7:0] are added.

   S3C3410X User's Manual reference: Table 1-3, page 1-11

2) The following errata should be corrected:
   RXD → URXD, TXD → UTXD, SIOCK[1:0] → SIOCLK[1:0], EXTAI0 → EXTAL0, SYSCFG0 → SYSCFG, MEMCONx → BANKCONx, EDVCONx → EXTCONx, EXTDATAx → EXTDATx , UTXHW → UTXH_W, URXHW → URXH_W, IICADD(0xe002) → IICADD(0xe003), IICDS(0xe003) → IICDS(0xe002)

   S3C3410X User's Manual reference: Table 1-3, page 1-11

2. EXTCONX, EXTPORT, EXTDATX AND TIMING DIAGRAMS:

1) Contents about EXTCONx, EXTPORT, and EXTDATx are changed.

   S3C3410X User's Manual reference: page 4-14 and page 4-15

2) "Multiplexed Address Mode Timing Diagrams", "nCS Timing Diagram with nWAIT", and " nECS Timing Diagram with nWAIT" are added.
3) External Device Interface Diagram is changed.

   S3C3410X User's Manual reference: Figure 4-21, page 4-30

3. CACHE DISABLE OPERATION:

1) More detailed explanations about the internal SRAM address (when the cache is disabled) is added.

   S3C3410X User's Manual reference: page 5-4

4. PORT 7 AND PORT 9 CONTROL REGISTERS:

1) The contents of P7BR(0xB00B) is added in PORT 7 and the pin descriptions of P7.x are changed to P7.x (RPx).

   S3C3410X User's Manual reference: page 7-20

2) More detailed explanations about P9.0(LP) and P9.1(DCLK) are added.

   S3C3410X User's Manual reference: page 7-25

(Continued to the next page)

5. INTERRUPT PRIORITY REGISTER:

1) The contents about the INTPRIx are changed .
   S3C3410X User's Manual reference: page 11-10

6. MULTI-MASTER IIC-BUS STATUS REGISTER:

1) The contents of INTFLAG is added to IICSTAT register.
   S3C3410X User's Manual reference: page 14-7

2) The prescaler value (4 × (prescaler value + 1)) is changed to (16 × (prescaler value + 1)) in IICPS.
   S3C3410X User′s Manual reference: page 14-9

1 PRODUCT OVERVIEW

INTRODUCTION

Samsung's S3C3410X 16/32-bit RISC microcontroller is a cost-effective and high-performance microcontroller solution for PDA and general purpose application.

An outstanding feature of the S3C3410X 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, which was developed for the use in application-specific and customer-specific integrated circuits. Its simple, elegant, and fully static design is particularly suitable for cost-sensitive and power-sensitive application.

The S3C3410X has been developed by using the ARM7TDMI core, CMOS standard cell, and a data path compiler. Most of the on-chip function blocks have been designed using an HDL synthesizer. The S3C3410X has been fully verified in SAMSUNG ASIC test environment including the internal Qualification Assurance Process.

By providing a complete set of common system peripherals, the S3C3410X can minimize the overall system cost and eliminates the need to configure additional components, externally.

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

- Integrated external memory controller (ROM/SRAM and FP/EDO DRAM/SDRAM controller)
- 2-channel general DMA controller
- Internal 4K-byte memory can be configured as (4KB Cache only), (2KB Cache and 2KB SRAM), or (4KB SRAM only).
- 1-channel UART with IrDA 1.0, 1-channel IIC, and 2-channel SIO(Synchronous serial IO)
- 3-channel 16-bit timers and 2-channel 8-bit timers
- Real time clock with calendar function.
- Crystal/Ceramic oscillator or external clock can be used as the clock source.
- Power control: Normal, Idle, and Stop mode
- 1-channel 8-bit basic timer and 3-bit watch-dog timer
- Interrupt controller: 35 interrupt sources, interrupt priority control logic and interrupt vector generation by H/W.
- 8-channel 10-bit ADC
- 10 programmable I/O port group (Total 74 I/O ports including the multiplexed I/O)

FEATURES

Architecture

- Integrated system for hand-held and general embedded application.
- Fully 16/32-bit RISC architecture(32-bit ARM instruction as well as 16-bit Thumb instruction).
- ARM7TDMI CPU core, supporting the efficient and powerful instruction set.
- On-chip ICEBreaker^{TM} debug support by JTAG-based solution.
- 4KB Unified Cache (Instruction/Data Cache Memory)

System Manager

- Address space: 16Mbytes per each bank (Total 128Mbyte)
- Support 8-bit/16-bit data bus width for external memory/device access.
- The bank can support ROM/SRAM/Flash, External I/O device or FP/EDO/SDRAM.
- Among total 8 memory banks, bank0,1,2,3,4 and 5 can be mapped to ROM/SRAM/Flash, while bank6 and 7 can be mapped to FP/EDO/SDRAM as well as ROM/SRAM/Flash.
- Fully programmable access cycle for all memory banks
- Supports self-refresh/auto-refresh mode to retain the data in the DRAM.
- Two external I/O banks can be mapped to the SFR (Special Function Register) region.

Unified(Instruction/Data) Cache Memory & Internal SRAM

- Two-way set associative 4KB cache.
- Pseudo LRU (Least Recently Used) replacement policy.
- Four depth write buffer.
- Programmable configuration of (4KB cache, only), (2KB cache and 2KB SRAM), or (4KB SRAM, only).

DMA Controller

- Two-channel general purposed DMA(Direct Memory Access) controller.
- The data transfer of Memory-to-memory, serial port-to-memory, memory-to-serial port, memory-to-SFR(Special Function Register), SFR-to-memory, internal SRAM-to-memory, and memory-to-internal SRAM without CPU intervention
- Initiated by the software or external DMA request
- Increment or decrement source or destination addresses.
- Supports 8-bit(byte), 16-bit(half-word), and 32-bit(word) of data transfer size.

I/O Ports

- 10 Programmable Input, Output, and I/O port group (74 I/O ports including the multiplexed I/O)
- One programmable Output port (2-bit multiplexed output ports)
- One programmable Input port(8-bit multiplexed input ports)
- Eight programmable I/O port group.

16-bit Timer/Counters (T0, T1, T2)

- Three-channel programmable 16-bit timer/counter
- Interval, capture, match & overflow, or DMA mode operation
- Internal or external clock source

8-bit Timer/Counters (T3, T4)

- Two-channel programmable 8-bit timer/counter
- Interval, capture, PWM, or DMA mode operation (T4 PWM with 5-byte FIFO buffer, which can provide the sound generation capability)
- Internal or external clock source

UART & SIO

- One-channel UART with DMA-based or interrupt-based operation
- Programmable baud rates
- Supports 5-bit, 6-bit, 7-bit and 8-bit serial data transmit/receive frame in UART
- Programmable accessible 8-byte transmitter FIFO and 8-byte receiver FIFO in UART
- Two-channel synchronous SIO with DMA-based or interrupt-based operation
- Support the serial data transmit/receive operation by 8-bit frame.

Interrupt Controller

- 35 interrupt sources (12 External interrupt, 2 DMA, 3 UART, 11 Timer, ADC, IIC, 2 SIO, Basic Timer, 2 RTC)
- H/W interrupt priority logic and vector generation
- Normal or fast interrupt modes (IRQ, FIQ)

A/D Converter

- Eight-channel multiplexed ADC
- Successive approximation conversion
- 10-bit ADC

WDT(Watch-Dog Timer) and Basic Timer

- 8-bit Counter (Basic Timer) and 3-bit counter (Watchdog Timer)
- The overflow signal of 8-bit counter can generate a basic timer interrupt and should be input clock for 3-bit counter(Watchdog Timer).
- The overflow signal of 3-bit counter makes a system reset

IIC Bus Interface

- One-channel multi-master IIC-bus
- Support 8-bit, bi-directional, and serial data transfer up to 100kbit/s.

RTC (Real Time Clock)

- Full clock function : second, minute, hours, day, week, month, and year
- 32.768KHz operation
- Alarm interrupt for CPU wake-up

Power Down Mode

- Power mode: Idle, Slow and Stop mode
- System clock division ratio in slow mode: 1, 1/2, 1/8, 1/16, and 1/1024

Operating Voltage Range

- 3.0 V to 3.6 V

Temperature Range

- $0^{o}C$ to $70^{o}C$

Operating Frequency

- up to 40MHz

Package Type

- 128-pin QFP

BLOCK DIAGRAM

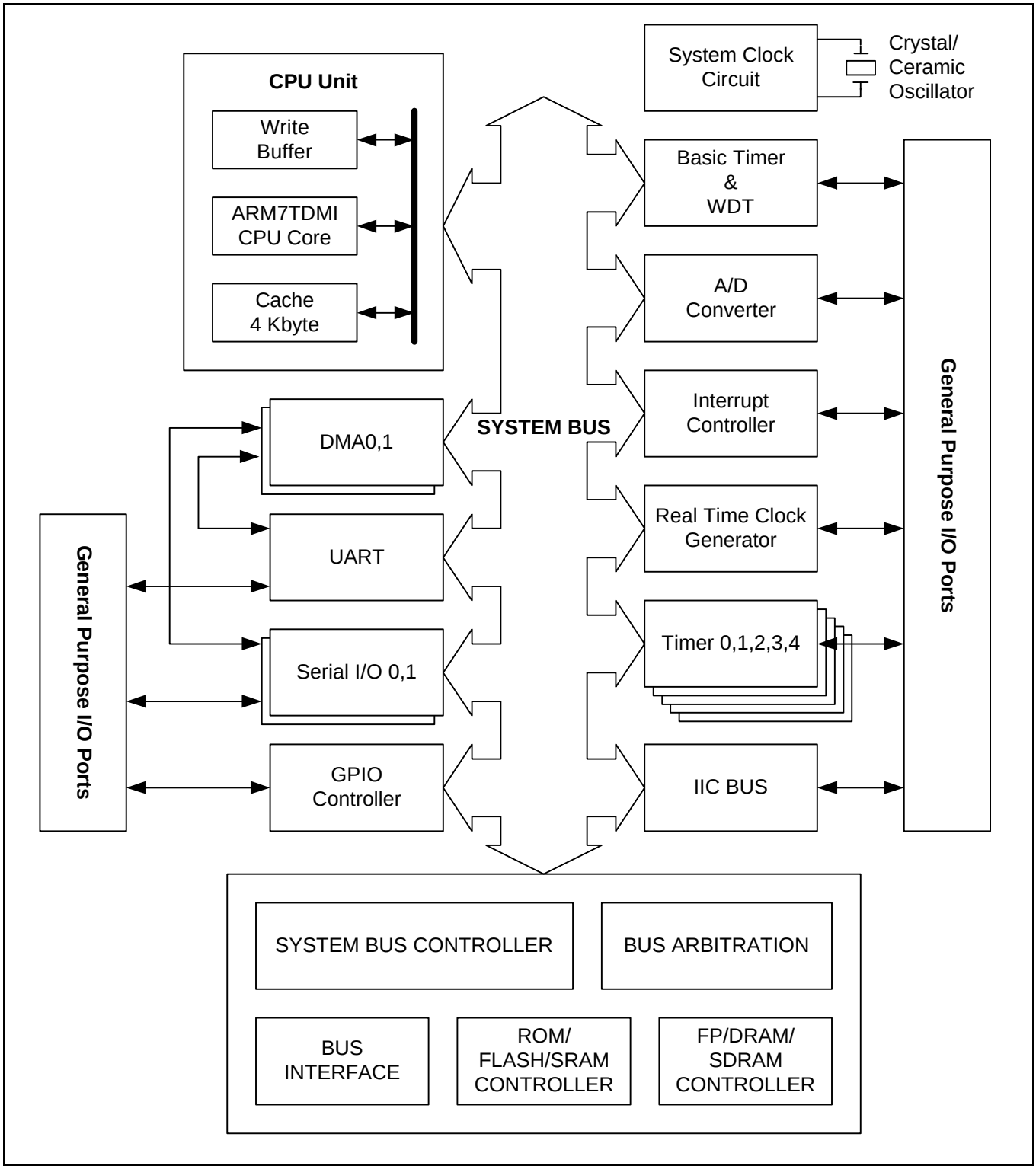


Figure 1-1. S3C3410X Block Diagram

PIN ASSIGNMENTS

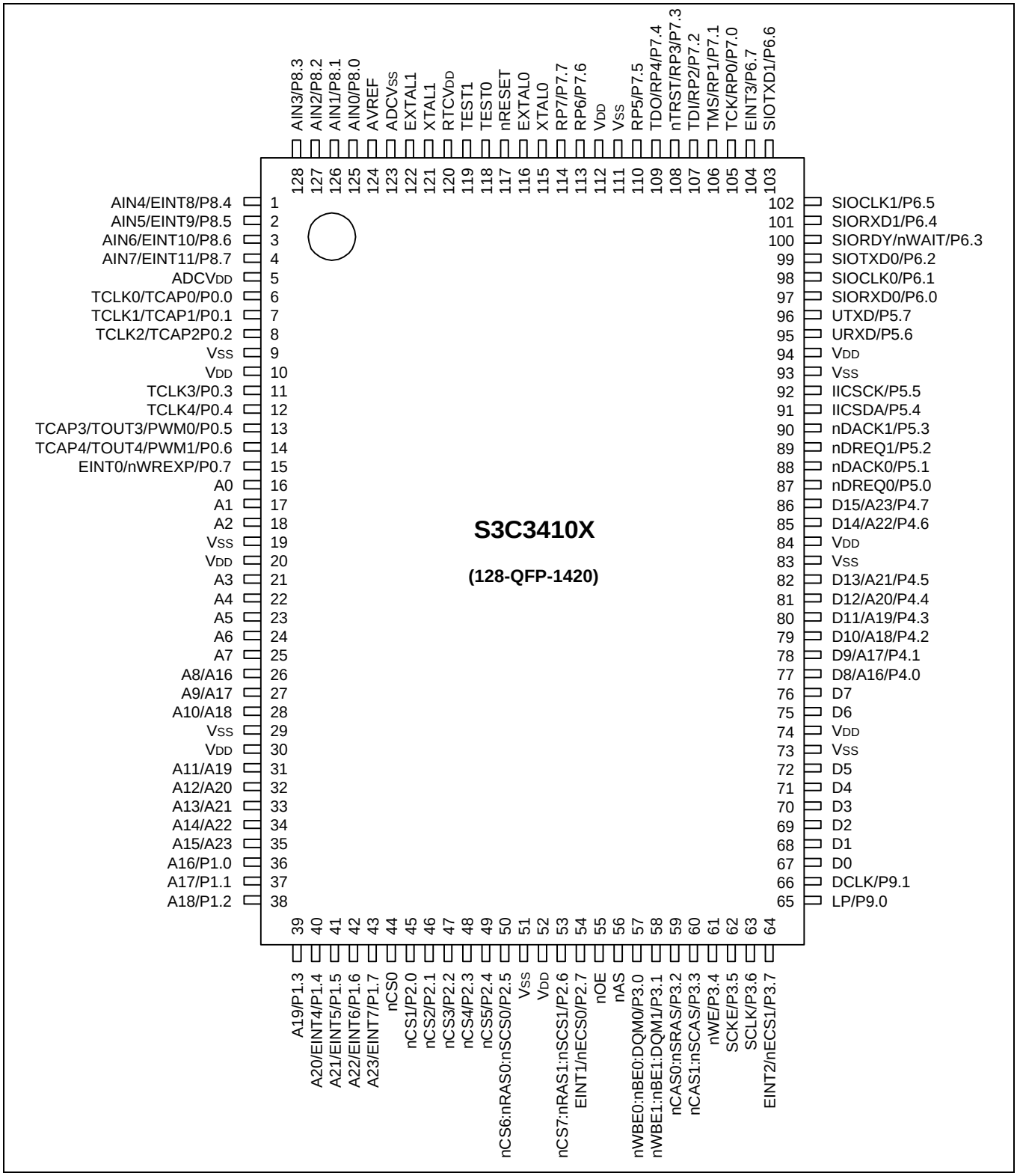


Figure 1-2. S3C3410X Pin Assignments

Table 1-1. 128-Pin QFP Pin Assignment

Pin No	Function	I/O State @Initial	I/O Type	Reset
1	AIN4/EINT8/P8.4	I	piseuc	P8.4
2	AIN5/EINT9/P8.5	I	piseuc	P8.5
3	AIN6/EINT10/P8.6	I	piseuc	P8.6
4	AIN7/EINT11/P8.7	I	piseuc	P8.7
5	ADCVDD	P	vddt	
6	TCLK0/TCAP0/P0.0	IO	pbseuct4	P0.0
7	TCLK1/TCAP1/P0.1	IO	pbseuct4	P0.1
8	TCLK2/TCAP2/P0.2	IO	pbseuct4	P0.2
9	VSS	P	vss	
10	VDD	P	vdd	
11	TCLK3/P0.3	IO	pbseuct4	P0.3
12	TCLK4/P0.4	IO	pbseuct4	P0.4
13	TCAP3/TOUT3/PWM0/P0.5	IO	pbseuct4	P0.5
14	TCAP4/TOUT4/PWM1/P0.6	IO	pbseuct4	P0.6
15	EINT0/nWREXP/P0.7	IO	pbseuct8	P0.7
16	A0	O	pob8	A0
17	A1	O	pob8	A1
18	A2	O	pob8	A2
19	VSS	P	vss	
20	VDD	P	vdd	
21	A3	O	pob8	A3
22	A4	O	pob8	A4
23	A5	O	pob8	A5
24	A6	O	pob8	A6
25	A7	O	pob8	A7
26	A8/A16	O	pob8	A8
27	A9/A17	O	pob8	A9
28	A10/A18	O	pob8	A10
29	VSS	P	vss	
30	VDD	P	vdd	
31	A11/A19	O	pob8	A11
32	A12/A20	O	pob8	A12

Table 1-1. 128-Pin QFP Pin Assignment (Continued)

Pin No	Function	I/O State @Initial	I/O Type	Reset
33	A13/A21	O	pob8	A13
34	A14/A22	O	pob8	A14
35	A15/A23	O	pob8	A15
36	A16/P1.0	IO	pbcedct8	P1.0
37	A17/P1.1	IO	pbcedct8	P1.1
38	A18/P1.2	IO	pbcedct8	P1.2
39	A19/P1.3	IO	pbcedct8	P1.3
40	A20/EINT4/P1.4	IO	pbsedct8	P1.4
41	A21/EINT5/P1.5	IO	pbsedct8	P1.5
42	A22/EINT6/P1.6	IO	pbsedct8	P1.6
43	A23/EINT7/P1.7	IO	pbsedct8	P1.7
44	nCS0	O	pob8	nCS0
45	nCS1/P2.0	IO	pbceuct8	P2.0
46	nCS2/P2.1	IO	pbceuct8	P2.1
47	nCS3/P2.2	IO	pbceuct8	P2.2
48	nCS4/P2.3	IO	pbceuct8	P2.3
49	nCS5/P2.4	IO	pbceuct8	P2.4
50	nCS6:nRAS0:nSCS0/P2.5	IO	pbceuct8	P2.5
51	VSS	P	vss	
52	VDD	P	vdd	
53	nCS7:nRAS1:nSCS1/P2.6	IO	pbceuct8	P2.6
54	EINT1/nECS0/P2.7	IO	pbseuct8	P2.7
55	nOE	O	pob8	nOE
56	nAS	O	pob8	nAS
57	nWBE0:nBE0:DQM0/P3.0	IO	pbceuct8	P3.0
58	nWBE1:nBE1:DQM1/P3.1	IO	pbceuct8	P3.1
59	nCAS0:nSRAS/P3.2	IO	pbceuct8	P3.2
60	nCAS1:nSCAS/P3.3	IO	pbceuct8	P3.3
61	nWE/P3.4	IO	pbceuct8	P3.4
62	SCKE/P3.5	IO	pbceuct8	P3.5
63	SCLK/P3.6	IO	pbceuct8	P3.6
64	EINT2/nECS1/P3.7	IO	pbseuct8	P3.7

Table 1-1. 128-Pin QFP Pin Assignment (Continued)

Pin No	Function	I/O State @Initial	I/O Type	Reset
65	LP/P9.0	O	pob8	LP
66	DCLK/P9.1	O	pob8	DCLK
67	D0	IO	pbcedct8	D0
68	D1	IO	pbcedct8	D1
69	D2	IO	pbcedct8	D2
70	D3	IO	pbcedct8	D3
71	D4	IO	pbcedct8	D4
72	D5	IO	pbcedct8	D5
73	VSS	P	vss	
74	VDD	P	vdd	
75	D6	IO	pbcedct8	D6
76	D7	IO	pbsedct8	D7
77	D8/A16/P4.0	IO	pbcedct8	P4.0
78	D9/A17/P4.1	IO	pbcedct8	P4.1
79	D10/A18/P4.2	IO	pbcedct8	P4.2
80	D11/A19/P4.3	IO	pbcedct8	P4.3
81	D12/A20/P4.4	IO	pbcedct8	P4.4
82	D13/A21/P4.5	IO	pbcedct8	P4.5
83	VSS	P	vss	
84	VDD	P	vdd	
85	D14/A22/P4.6	IO	pbcedct8	P4.6
86	D15/A23/P4.7	IO	pbcedct8	P4.7
87	nDREQ0/P5.0	IO	pbceuct4	P5.0
88	nDACK0/P5.1	IO	pbceuct4	P5.1
89	nDREQ1/P5.2	IO	pbceuct4	P5.2
90	nDACK1/P5.3	IO	pbceuct4	P5.3
91	IICSDA/P5.4	IO	pbceuct8	P5.4
92	IICSCK/P5.5	IO	pbceuct8	P5.5
93	VSS	P	vss	
94	VDD	P	vdd	
95	URXD/P5.6	IO	pbceuct4	P5.6
96	UTXD/P5.7	IO	pbceuct4	P5.7

Table 1-1. 128-Pin QFP Pin Assignment (Continued)

Pin No	Function	I/O State @Initial	I/O Type	Reset
97	SIORXD0/P6.0	IO	pbseuct4	P6.0
98	SIOCLK0/P6.1	IO	pbseuct4	P6.1
99	SIOTXD0/P6.2	IO	pbseuct4	P6.2
100	SIORDY/nWAIT/P6.3	IO	pbseuct4	P6.3
101	SIORXD1/P6.4	IO	pbseuct4	P6.4
102	SIOCLK1/P6.5	IO	pbseuct4	P6.5
103	SIOTXD1/P6.6	IO	pbseuct4	P6.6
104	EINT3/P6.7	IO	pbseuct4	P6.7
105	TCK/RP0/P7.0	IO	pbceuct4	P7.0
106	TMS/RP1/P7.1	IO	pbceuct4	P7.1
107	TDI/RP2/P7.2	IO	pbceuct4	P7.2
108	nTRST/RP3/P7.3	IO	pbceuct4	P7.3
109	TDO/RP4/P7.4	IO	pbceuct4	P7.4
110	RP5/P7.5	IO	pbceuct4	P7.5
111	VSS	P	vss	
112	VDD	P	vdd	
113	RP6/P7.6	IO	pbceuct4	P7.6
114	RP7/P7.7	IO	pbceuct4	P7.7
115	XTAL0	I	oscm	XTAL0
116	EXTAL0	O	oscm	EXTAL0
117	RESET	I	pisu	RESET
118	TEST0	I	pis	TEST0
119	TEST1	I	pis	TEST1
120	RTCVDD	P	vddt	
121	XTAL1	I	oscm	XTAL1
122	EXTAL1	O	oscm	EXTAL1
123	ADCVSS	P	vsst	
124	AVREF	A	apad	AVREF
125	AIN0/P8.0	I	piseuc	P8.0
126	AIN1/P8.1	I	piseuc	P8.1
127	AIN2/P8.2	I	piseuc	P8.2
128	AIN3/P8.3	I	piseuc	P8.3

Table 1-2. I/O Type Description

I/O Type	Description
vdd, vss	3.3V Vdd/Vss
vddt, vsst	3.3V Vdd/Vss for analog circuitry
pbceuct4	bi-direction pad, CMOS level, pull-up resister with control, tri-state, Io = 4mA
pbseuct4	bi-direction pad, CMOS schmitt-trigger, pull-up resister with control, tri-state, Io = 4mA
pbceuct8	bi-direction pad, CMOS level, pull-up resister with control, tri-state, Io = 8mA
pbseuct8	bi-direction pad, CMOS schmitt-trigger, pull-up resister with control, tri-state, Io = 8mA
pbcedct8	bi-direction pad, CMOS level, pull-down resister with control, tri-state, Io = 8mA
pbsedct8	bi-direction pad, CMOS schmitt-trigger, pull-down resister with control, tri-state, Io = 8mA
pob8	output pad, Io = 8mA
pis	input pad, CMOS schmitt-trigger
pisu	input pad, CMOS schmitt-trigger, pull-up resister
piceuc	input pad, CMOS level, pull-up resister with control
piseuc	input pad, CMOS schmitt-trigger, pull-up resister with control
apad	pad for analog pin
oscm	pad for x-tal oscillation

PIN DESCRIPTIONS

Table 1-3. S3C3410X Pin Descriptions

Pin	I/O	Description
BUS CONTROLLER		
TEST[1:0]	I	The TEST[1:0] can configure the data bus size for bank 0 in normal or MDS mode. The normal mode is for CPU to start its operation by fetching the instruction from external memory. The MDS mode is for CPU to be debugged by the external Emulator, EmbeddedICE, etc. 00 = Normal mode with 8-bit data bus size for bank 0 access. 01 = Normal mode with 16-bit data bus size for bank 0 access. 10 = MDS mode with 8-bit data bus size for bank 0 access. 11 = MDS mode with 16-bit data bus size for bank 0 access.
A[23:0]	O	A[23:0] (address bus) generate the address when external memory access.
D[15:0]	I/O	D[15:0] (Data bus) input the data during memory read and output the data during memory write. The data bus width can be programmable for 8-bit or 16-bit by the BANKCONx register option.
nCS[7:0]	O	nCS[7:0] (Chip Select) selectively generate the chip select signal of each bank when the external memory access address is within the address range of each bank. The number of access cycle and the bank size can be programmable by the BANKCONx register option.
nECS[1:0]	O	nECS[1:0] (External Chip Select) generate the external chip select signal for the extra device (External I/O device).
nOE	O	nOE (Output Enable) indicates that the current bus cycle is a read cycle.
nWE	O	nWE (Write Enable for x16 SRAM or SDRAM) indicates that the current bus cycle is a write cycle. To support the byte write to external memory, the byte to be accessed can be determined by nBE[1:0], which is the selection on upper byte or lower byte. For example, in case of 16-bit SRAM, nBE[1:0] should play it role as UB(Upper Byte)/LB(Lower Byte) to select the upper byte or lower byte. In case of SDRAM, nWBE[1:0] should play it role as DQM[1:0] to select the upper byte or lower byte. For 16-bit access, not 8-bit access, both nWBE[1:0] should be activated at same time. In certain case, no more byte access is needed. For example, x16 Flash Memory does not need byte access through 16-bit bus when user need the programming in the flash memory. In this case, please use nWBE[0] instead of nWE to indicate that the current bus cycle is a write cycle. Summarizing, nWE should be used to indicate the write bus cycle in case of x16 SRAM and x16/x8 SDRAM. In case of x16 with two x8 SRAM, nWBE[0] and nWBE[1] should be connected to the WE of SRAM, respectively. For more detail information, please refer the chapter 4.
nWBE[1:0]	O	nWBE[1:0] (Write Byte Enable). In case of Flash or ROM access, nWBE[0] should be connected to the WE of memory. For the access to the non-volatile memory, we do not need the selection on bytes because the 8-bit write cycle via 16-bit bus is no more necessary. To program the data into the non-volatile memory, we should always use the 16-bit access. In this configuration, please use nWBE[0] instead of nWE to indicate that the current bus cycle is a write cycle. Summarizing, nWBE[0] should be used to indicate the current write bus cycle in case of x8 SRAM, x8/x16 ROM, EDODRAM or Flash memory. For more detail information, please refer the chapter 4.

Table 1-3. S3C3410X Signal Descriptions (Continued)

Pin	I/O	Description
nAS	O	nAS generates an address strobe signal for latch device in multiplexed address mode which generate A[23:16] and A[15:8] address in A[15:8] pins.
nWAIT	I	nWAIT receives request signal to prolong a current bus cycle. As long as nWAIT is "Low", the current bus cycle cannot be completed.
nWREXP	O	nWREXP outputs write strobe signal for external device, when you write any data into EXTPORT register to interface external device.
DRAM/SDRAM		
nRAS[1:0]	O	Row Address Strobe
nCAS[1:0]	O	Column Address Strobe
nSCS[1:0]	O	SDRAM Chip Select
nSRAS	O	SDRAM Row Address Strobe
nSCAS	O	SDRAM Column Address Strobe
DQM[1:0]	O	SDRAM Data Mask
SCLK	O	SDRAM Clock
SCKE	O	SDRAM Clock Enable
16-bit/8-bit Timer		
TCLK[4:0]	I	External clock input for Timer
TCAP[4:0]	I	Capture input for Timer
TOUT[4:3]	O	Timer 3, 4 output or PWM output
DMA		
nDREQ[1:0]	I	External DMA request
nDACK[1:0]	O	External DMA acknowledge
Interrupt Controller		
EINT[12:0]	I	External interrupt request
UART		
URXD	I	UART receives data input
UTXD	O	UART transmits data output
SIO		
SIOCLK[1:0]	I/O	SIO external clock
SIORXD[1:0]	I	SIO receives data input
SIOTXD[1:0]	O	SIO transmits data output
SIORDY	I/O	SIO handshakes signal when SIO operation is done by DMA
IIC-BUS		
IICSDA	I/O	IIC-bus data
IICSCK	I/O	IIC-bus Clock

Table 1-3. S3C3410X Signal Descriptions (Continued)

Pin	I/O	Description
ADC		
AIN[7:0]	A	ADC input
AVREF	A	ADC Vref
General Purpose I/O		
Pn.x	I/O	General purpose input/output ports
RP[7:0]	O	Real time buffer output ports (refer to P7)
RESET & Clock		
RESET	I	RESET is the global reset input for the S3C3410X. For a system reset, RESET must be held to "Low" level for at least 1us.
XTAL0	A	Crystal input for internal oscillation circuit for system clock
EXTAL0	A	Crystal output for internal oscillation circuit for system clock. It's the inverted output of XTAL0.
XTAL1	A	32.768KHz crystal input for RTC
EXTAL1	A	32.768KHz crystal output for RTC. It's the inverted output of XTAL1.
LCD Interface		
LP	O	LCD Line Pulse (Inversion of nECS0)
DCLK	O	LCD Clock (Inversion of nWREXP)
JTAG Test Logic		
nTRST	I	nTRST (TAP Controller Reset) can reset the TAP controller at power-up. A 100K pull-up resistor is connected to nTRST pin, internally. If the debugger(BlackICE) is not used, nTRST pin should be "Low" level or low active pulse should be applied before CPU running. For example, RESET signal can be tied with nTRST.
TMS	I	TMS (TAP Controller Mode Select) can control the sequence of the state diagram of TAP controller. A 100K pull-up resistor is connected to TMS pin, internally.
TCK	I	TCK (TAP Controller Clock) can provide the clock input for the JTAG logic. A 100K pull-up resistor is connected to TCK pin, internally.
TDI	I	TDI (TAP Controller Data Input) is the serial input for JTAG port. A 100K pull-up resistor is connected to TDI pin, internally.
TDO	O	TDO (TAP Controller Data Output) is the serial output for JTAG port.
POWER		
VDD	P	Power supply pin
VSS	P	Ground pin
RTCVDD	P	RTC power supply
ADCVDD	P	ADC power supply
ADCVSS	P	ADC ground & RTC ground

S3C3410X SPECIAL FUNCTION REGISTER

Table 1-4. S3C3410X Special Function Register

Group	Register	Offset	R/W	Description	Access	Reset Value
System Manager	SYSCFG0	0x1000	R/W	System Configuration Register	W	0xfff1
	BANKCON0	0x2000	R/W	Memory Bank 0 Control Register	W	0x00200070
	BANKCON1	0x2004	R/W	Memory Bank 1 Control Register	W	0x0
	BANKCON2	0x2008	R/W	Memory Bank 2 Control Register	W	0x0
	BANKCON3	0x200c	R/W	Memory Bank 3 Control Register	W	0x0
	BANKCON4	0x2010	R/W	Memory Bank 4 Control Register	W	0x0
	BANKCON5	0x2014	R/W	Memory Bank 5 Control Register	W	0x0
	BANKCON6	0x2018	R/W	Memory Bank 6 Control Register	W	0x0
	BANKCON7	0x201c	R/W	Memory Bank 7 Control Register	W	0x0
	REFCON	0x2020	R/W	DRAM Refresh Control Register	W	0x1
	EXTCON0	0x2030	R/W	Extra device control register 0	W	0x0
	EXTCON1	0x2034	R/W	Extra device control register 1	W	0x0
	EXTPORT	0x203e	R/W	External port data register	B/H	0x0
	EXTDAT0	0x202c	R/W	Extra chip selection data register 0	B/H	0x0
	EXTDAT1	0x202e	R/W	Extra chip selection data register 1	B/H	0x0
DMA	DMACON0	0x300c	R/W	DMA 0 control register	W	0x0
	DMASRC0	0x3000	R/W	DMA 0 source address register	W	0x0
	DMADST0	0x3004	R/W	DMA 0 destination address register	W	0x0
	DMACNT0	0x3008	R/W	DMA 0 transfer count register	W	0x0
	DMACON1	0x400c	R/W	DMA 1 Control Register	W	0x0
	DMASRC1	0x4000	R/W	DMA 1 source address register	W	0x0
	DMADST1	0x4004	R/W	DMA 1 destination address register	W	0x0
	DMACNT1	0x4008	R/W	DMA 1 transfer count register	W	0x0
I/O Port	PDAT0	0xb000	R/W	Port 0 data register	B	0x0
	PDAT1	0xb001	R/W	Port 1 data register	B	0x0
	PDAT2	0xb002	R/W	Port 2 data register	B	0x0
	PDAT3	0xb003	R/W	Port 3 data register	B	0x0
	PDAT4	0xb004	R/W	Port 4 data register	B	0x0
	PDAT5	0xb005	R/W	Port 5 data register	B	0x0
	PDAT6	0xb006	R/W	Port 6 data register	B	0x0
	PDAT7	0xb007	R/W	Port 7 data register	B	0x0
	PDAT8	0xb008	R	Port 8 data register	B	0x0

Table 1-4. S3C3410X Special Function Register (Continued)

Group	Register	Offset	R/W	Description	Access	Reset Value
I/O Port	PDAT9	0xb009	R/W	Port 9 data register	B	0x0
	P7BR	0xb00b	R/W	Port 7 buffer register	B	0x0
	PCON0	0xb010	R/W	Port 0 control register	H	0x0
	PCON1	0xb012	R/W	Port 1 control register	H	0x0
	PCON2	0xb014	R/W	Port 2 control register	H	0x0
	PCON3	0xb016	R/W	Port 3 control register	H	0x0
	PCON4	0xb018	R/W	Port 4 control register	H	0x0
	PCON5	0xb01c	R/W	Port 5 control register	W	0x0
	PCON6	0xb020	R/W	Port 6 control register	W	0x0
	PCON7	0xb024	R/W	Port 7 control register	H	0x0
	PCON8	0xb026	R/W	Port 8 control register	B	0x0
	PCON9	0xb027	R/W	Port 9 control register	B	0x0
	PUR0	0xb028	R/W	Port 0 pull-up control register	B	0x80
	PDR1	0xb029	R/W	Port 1 pull-down control register	B	0xff
	PUR2	0xb02a	R/W	Port 2 pull-up control register	B	0xff
	PUR3	0xb02b	R/W	Port 3 pull-up control register	B	0xff
	PDR4	0xb02c	R/W	Port 4 pull-down control register	B	0xff
	PUR5	0xb02d	R/W	Port 5 pull-up control register	B	0x0
	PUR6	0xb02e	R/W	Port 6 pull-up control register	B	0x0
	PUR7	0xb02f	R/W	Port 7 pull-up control register	B	0x0
	PUR8	0xb03c	R/W	Port 8 pull-up control register	B	0x0
	EINTPND	0xb031	R/W	External interrupt pending register	B	0x0
	EINTCON	0xb032	R/W	External interrupt control register	H	0x0
	EINTMOD	0xb034	R/W	External interrupt mode register	W	0x0
Timer 0	TDAT0	0x9000	R/W	Timer 0 data register	H	0xffff
	TPRE0	0x9002	R/W	Timer 0 prescaler register	B	0x0
	TCON0	0x9003	R/W	Timer 0 control register	B	0x0
	TCNT0	0x9006	R	Timer 0 counter register	H	0x0
Timer 1	TDAT1	0x9010	R/W	Timer 1 data register	H	0xffff
	TPRE1	0x9012	R/W	Timer 1 prescaler register	B	0x0
	TCON1	0x9013	R/W	Timer 1 control register	B	0x0
	TCNT1	0x9016	R	Timer 1 counter register	H	0x0

Table 1-4. S3C3410X Special Function Register (Continued)

Group	Register	Offset	R/W	Description	Access	Reset Value
Timer 2	TDAT2	0x9020	R/W	Timer 2 data register	H	0xffff
	TPRE2	0x9022	R/W	Timer 2 prescaler register	B	0x0
	TCON2	0x9023	R/W	Timer 2 control register	B	0x0
	TCNT2	0x9026	R	Timer 2 counter register	H	0x0
Timer 3	TDAT3	0x9031	R/W	Timer 3 data register	B	0xff
	TPRE3	0x9032	R/W	Timer 3 prescaler register	B	0x0
	TCON3	0x9033	R/W	Timer 3 control register	B	0x0
	TCNT3	0x9037	R	Timer 3 counter register	B	0x0
Timer 4	TDAT4	0x9041	R/W	Timer 4 data register	B	0xff
	TPRE4	0x9042	R/W	Timer 4 prescaler register	B	0x0
	TCON4	0x9043	R/W	Timer 4 control register	B	0x0
	TCNT4	0x9047	R	Timer 4 counter register	B	0x0
	TFCON	0x904f	R/W	FIFO control register of Timer 4	B	0x0
	TFSTAT	0x904e	R	FIFO status register of Timer 4	B	0x0
	TFB4	0x904b	R/W	Timer 4 FIFO register @ byte	B	0x0
	TFHW4	0x904a	R/W	Timer 4 FIFO register @ half-word	H	0x0
	TFW4	0x9048	R/W	Timer 4 FIFO register @ word	W	0x0
UART	ULCON	0x5003	R/W	UART line control register	B	0x0
	UCON	0x5007	R/W	UART control register	B	0x0
	USTAT	0x500b	R	UART status register	B	0x0
	UFCON	0x500f	R/W	UART FIFO control register	B	0x0
	UFSTAT	0x5012	R	UART FIFO status register	B	0x0
	UTXH	0x5017	R/W	UART transmit holding register	B	0x0
	UTXH_B	0x5017	R/W	UART transmit FIFO register @ byte	B	0x0
	UTXH_HW	0x5016	R/W	UART transmit FIFO register @ half-word	H	0x0
	UTXH_W	0x5014	R/W	UART transmit FIFO register @ word	W	0x0
	URXH	0x501b	R/W	UART receive buffer register	B	0x0
	URXH_B	0x501b	R/W	UART receive FIFO register @ byte	B	0x0
	URXH_HW	0x501a	R/W	UART receive FIFO register @ half-word	H	0x0
	URXH_W	0x5018	R/W	UART receive FIFO register @ word	W	0x0
	UBRDIV	0x501e	R/W	Baud rate divisor register for UART	H	0x0

Table 1-4. S3C3410X Special Function Register (Continued)

Group	Register	Offset	R/W	Description	Access	Reset Value
SIO 0	ITVCNT0	0x6000	R/W	SIO 0 interval counter register	B	0x0
	SBRDR0	0x6001	R/W	SIO 0 baud rate prescaler register	B	0x0
	SIODAT0	0x6002	R/W	SIO 0 data register	B	0x0
	SIOCON0	0x6003	R/W	SIO 0 control register	B	0x0
SIO 1	ITVCNT1	0x7000	R/W	SIO 1 interval counter register	B	0x0
	SBRDR1	0x7001	R/W	SIO 1 baud rate prescaler register	B	0x0
	SIODAT1	0x7002	R/W	SIO 1 data register	B	0x0
	SIOCON1	0x7003	R/W	SIO 1 control register	B	0x0
Interrupt	INTMOD	0xc000	R/W	Interrupt mode register	W	0x0
	INTPND	0xc004	R/W	Interrupt pending register	W	0x0
	INTMSK	0xc008	R/W	Interrupt mask register	W	0x0
	INTPRI0	0xc00c	R/W	Interrupt priority register 0	W	0x03020100
	INTPRI1	0xc010	R/W	Interrupt priority register 1	W	0x07060504
	INTPRI2	0xc014	R/W	Interrupt priority register 2	W	0x0b0a0908
	INTPRI3	0xc018	R/W	Interrupt priority register 3	W	0x0f0e0d0c
	INTPRI4	0xc01c	R/W	Interrupt priority register 4	W	0x13121110
	INTPRI5	0xc020	R/W	Interrupt priority register 5	W	0x17161514
	INTPRI6	0xc024	R/W	Interrupt priority register 6	W	0x1b1a1918
	INTPRI7	0xc028	R/W	Interrupt priority register 7	W	0x1f1e1d1c
ADC	ADCCON	0x8002	R/W	A/D Converter control register	H	0x140
	ADCDAT	0x8006	R	A/D Converter data register	H	0x0
Basic	BTCON	0xa002	R/W	Basic Timer control register	H	0x0
Timer	BTCNT	0xa007	R	Basic Timer count register	B	0x0
IIC	IICCON	0xe000	R/W	IIC-bus control register	B	0x0
	IICSTAT	0xe001	R/W	IIC-bus status register	B	0x0
	IICDS	0xe002	R/W	IIC-Bus transmit/receive data shift register	B	0x0
	IICADD	0xe003	R/W	IIC-Bus transmit/receive address register	B	0x0
	IICPS	0xe004	R/W	IIC-Bus Prescaler register	B	0x0
	IICPCNT	0xe005	R/W	IIC-Bus Prescaler Counter register	B	0x0
	SYSON	0xd003	R/W	System control register	B	0x0

Table 1-4. S3C3410X Special Function Register (Continued)

Group	Register	Offset	R/W	Description	Access	Reset Value
RTC	RTCCON	0xa013	R/W	RTC control register	B	0x0
	RTCALM	0xa012	R/W	RTC alarm control register	B	0x0
	ALMSEC	0xa033	R/W	Alarm second data register	B	0x59
	ALMMIN	0xa032	R/W	Alarm minute data register	B	0x59
	ALMHOUR	0xa031	R/W	Alarm hour data register	B	0x23
	ALMDAY	0xa037	R/W	Alarm day data register	B	0x31
	ALMMON	0xa036	R/W	Alarm month data register	B	0x12
	ALMYEAR	0xa035	R/W	Alarm year data register	B	0x99
	BCDSEC	0xa023	R/W	BCD second data register	B	–
	BCDMIN	0xa022	R/W	BCD minute data register	B	–
	BCDHOUR	0xa021	R/W	BCD hour data register	B	–
	BCDDAY	0xa027	R/W	BCD day data register	B	–
	BCDDATE	0xa020	R/W	BCD date data register	B	–
	BCDMON	0xa026	R/W	BCD month data register	B	–
	BCDYEAR	0xa025	R/W	BCD year data register	B	–
	RINTPND	0xa010	R/W	RTC time interrupt pending register	B	0x0
	RINTCON	0xa011	R/W	RTC time interrupt control register	B	0x0

2 PROGRAMMER'S MODEL

OVERVIEW

S3C3410X was developed using the advanced ARM7TDMI core designed by Advanced RISC Machines, Ltd. ARM7TDMI supports big-endian and little-endian memory formats, but the S3C3410X supports only the big-endian memory format.

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 halfwords.

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.

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.

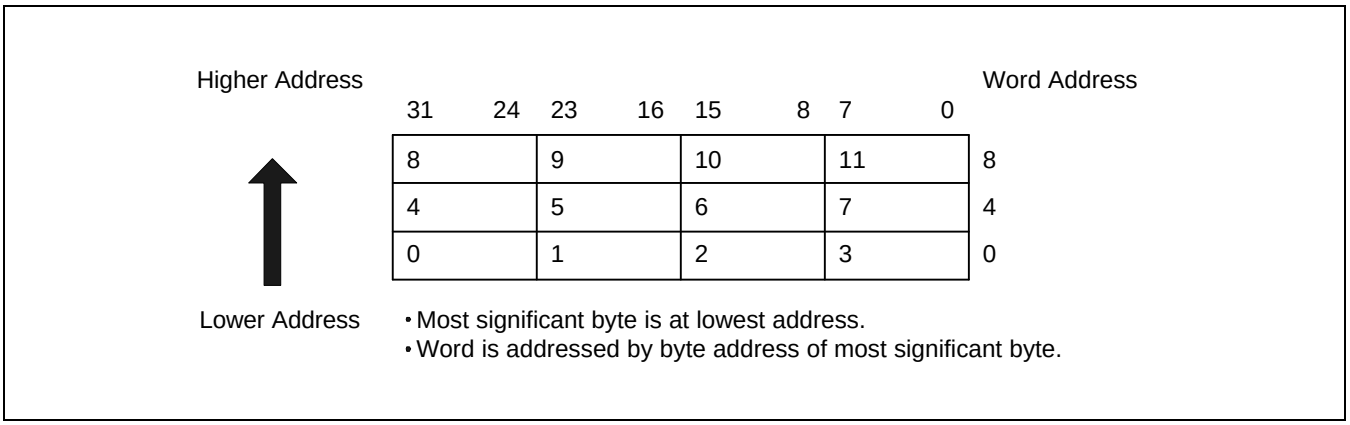


Figure 2-1. Big-Endian Addresses of Bytes within Words

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.

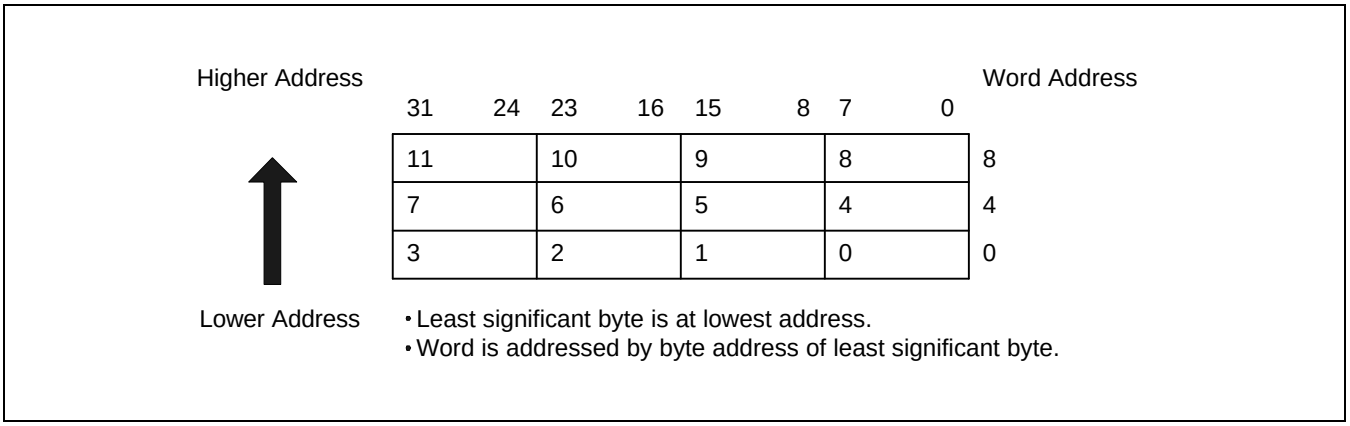


Figure 2-2. Little-Endian Addresses of Bytes whthin Words

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), halfword (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	holds the Program Counter (PC). In ARM state, bits [1:0] of R15 are zero and bits [31:2] contain the PC. In THUMB state, bit [0] is zero and bits [31:1] contain the PC.
Register 16	is the CPSR (Current Program Status Register). This contains condition code flags and 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.

ARM State General Registers and Program Counter

User/System	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

ARM State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

= banked register

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.

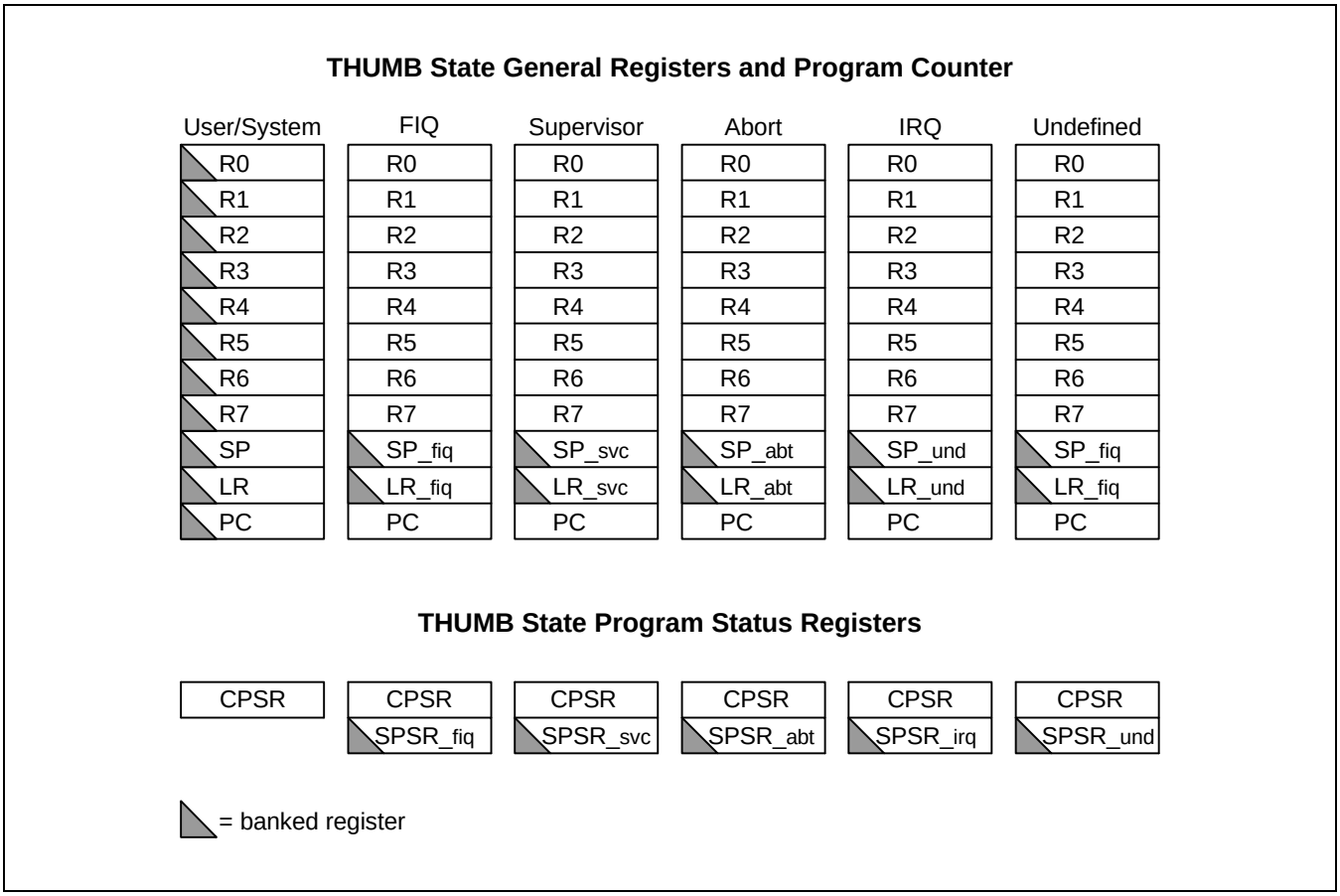


Figure 2-4. Register Organization in THUMB state

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.

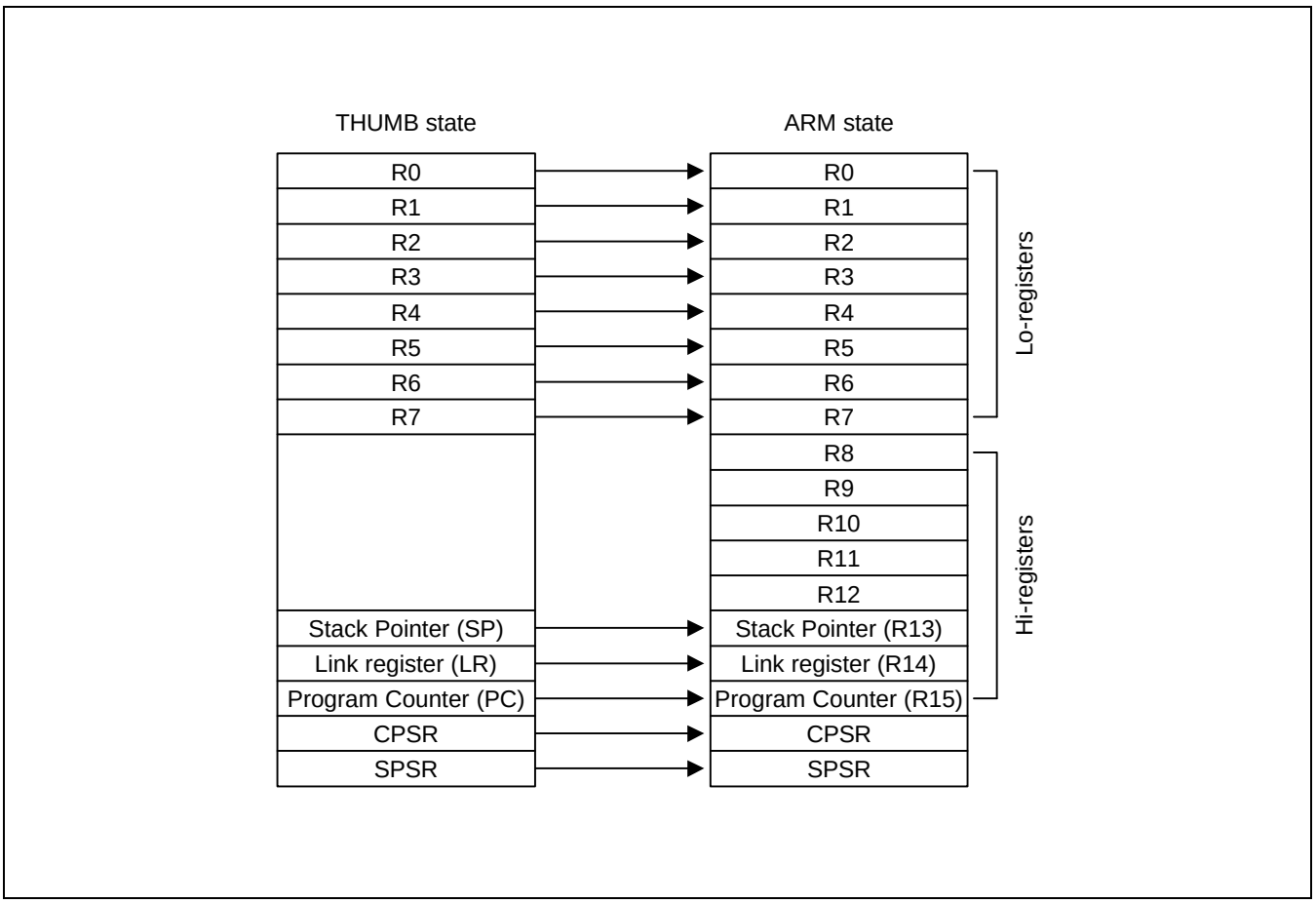


Figure 2-5. Mapping of THUMB State Registers onto ARM State Registers

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.

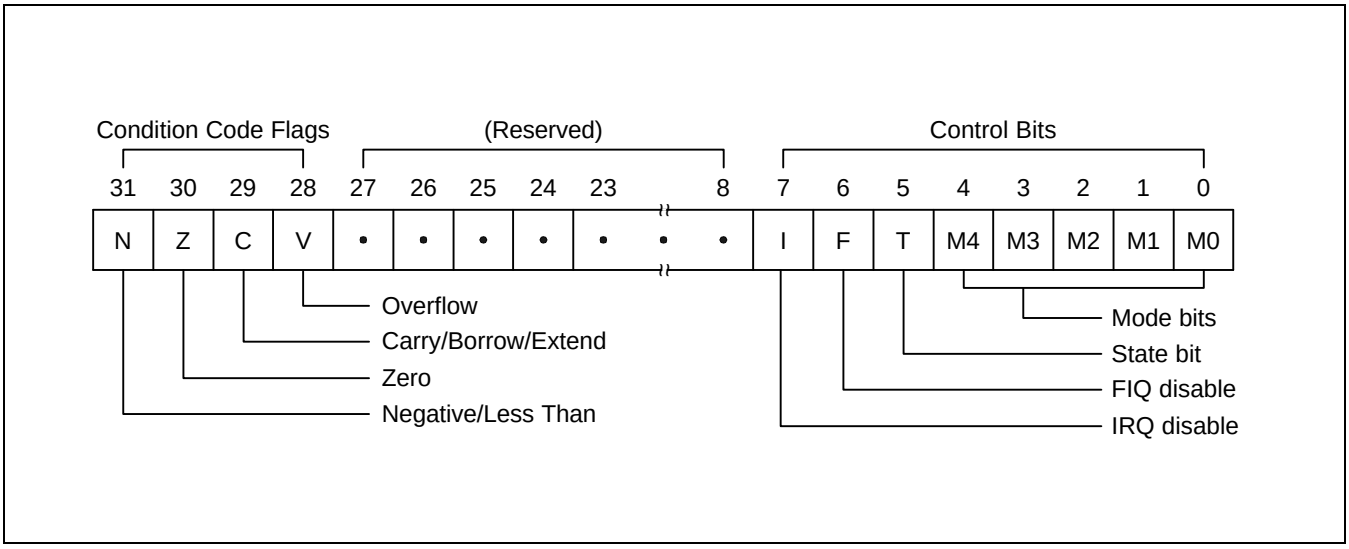


Figure 2-6. Program Status Register Format

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 be changed when an exception arises. If the processor is operating in a privileged mode, they can also be manipulated by software.

The T bit	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. 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.
Interrupt disable bits	The I and F bits are the interrupt disable bits. When set, these disable the IRQ and FIQ interrupts respectively.
The mode bits	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, M[4:0], then the processor will enter an unrecoverable state. If this occurs, reset should be applied.
Reserved bits	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.

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

Reserved bits — The remaining bits in the PSR's 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.

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 + 4	2
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 4	2
PABT	SUBS PC, R14_abt, #4	PC + 4	PC + 4	1
DABT	SUBS PC, R14_abt, #8	PC + 8	PC + 8	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 (ie 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 in 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 Priorites

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 (ie 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 (*Tsyncmax* if asynchronous), plus the time for the longest instruction to complete (*Tldm*, the longest instruction is an LDM which loads all the registers including the PC), plus the time for the data abort entry (*Texc*), plus the time for FIQ entry (*Tfiq*). At the end of this time ARM7TDMI will be executing the instruction at 0x1C.

Tsyncmax is 3 processor cycles, *Tldm* is 20 cycles, *Texc* is 3 cycles, and *Tfiq* 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 20 MHz 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 (*Tsyncmin*) plus *Tfiq*. This is 4 processor cycles.

RESET

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

When **RESET** 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.

NOTES

3 INSTRUCTION SET

INSTRUCTION SET SUMMAY

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.

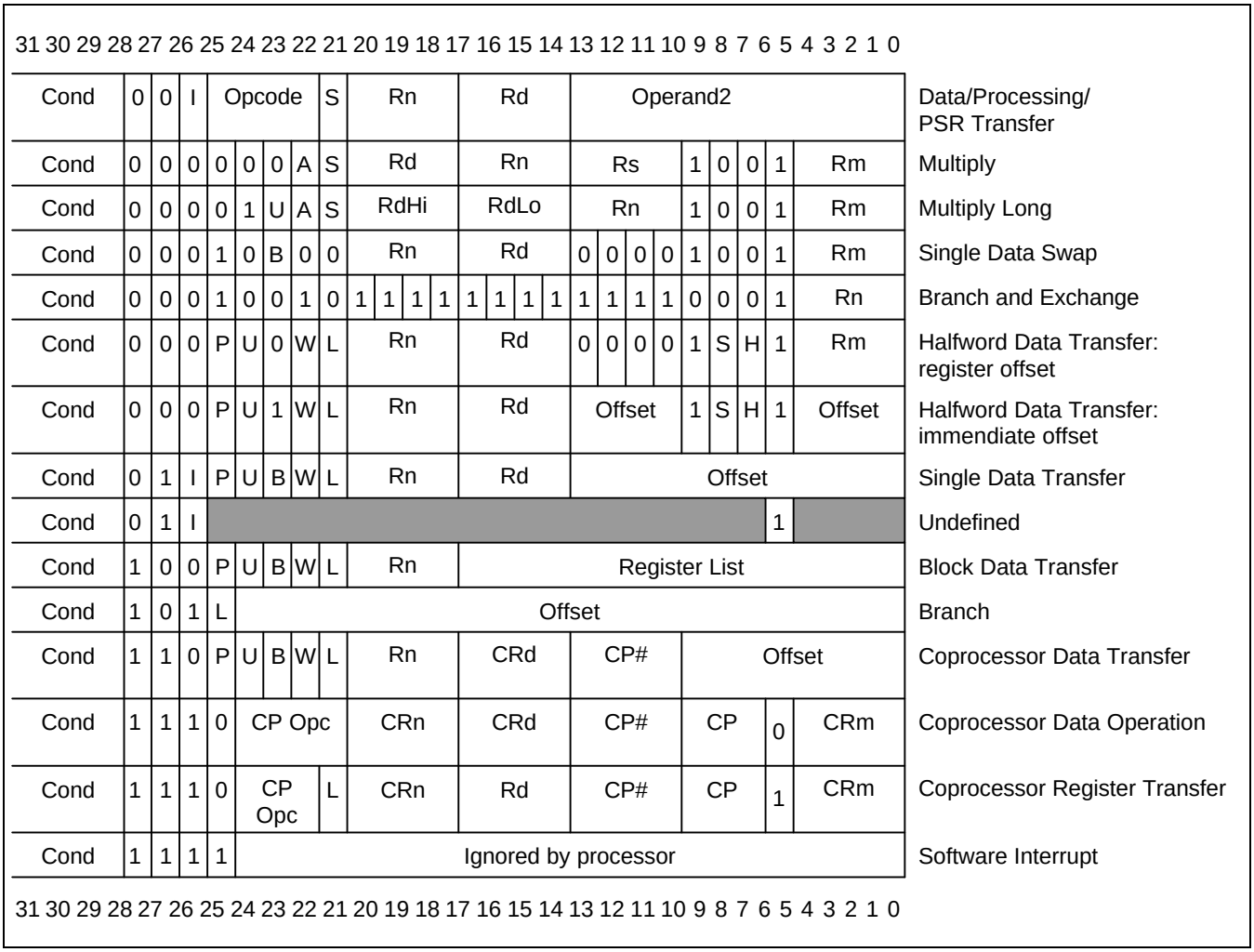


Figure 3-1. ARM Instruction Set Format

NOTE

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 + Carry
ADD	Add	Rd: = Rn + Op2
AND	AND	Rd: = Rn AND Op2
B	Branch	R15: = address
BIC	Bit Clear	Rd: = Rn AND NOT Op2
BL	Branch with Link	R14: = R15, R15: = address
BX	Branch and Exchange	R15: = Rn, T bit: = Rn[0]
CDP	Coprocessor Data Processing	(Coprocessor-specific)
CMN	Compare Negative	CPSR flags: = Rn + Op2
CMP	Compare	CPSR flags: = Rn – Op2
EOR	Exclusive OR	Rd: = (Rn AND NOT Op2) OR (Op2 AND NOT Rn)
LDC	Load coprocessor from memory	Coprocessor load
LDM	Load multiple registers	Stack manipulation (Pop)
LDR	Load register from memory	Rd: = (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

Table 3-1. The ARM Instruction Set (Continued)

Mnemonic	Instruction	Action
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: = 0 × FFFFFFFF EOR Op2
ORR	OR	Rd: = Rn OR Op2
RSB	Reverse Subtract	Rd: = Op2 – Rn
RSC	Reverse Subtract with Carry	Rd: = Op2 – Rn – 1 + Carry
SBC	Subtract with Carry	Rd: = Rn – Op2 – 1 + Carry
STC	Store coprocessor register to memory	address: = CRn
STM	Store Multiple	Stack manipulation (Push)
STR	Store register to memory	<address>: = Rd
SUB	Subtract	Rd: = Rn – Op2
SWI	Software Interrupt	OS call
SWP	Swap register with memory	Rd: = [Rn], [Rn] := Rm
TEQ	Test bit wise equality	CPSR flags: = Rn EOR Op2
TST	Test bits	CPSR flags: = Rn 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.

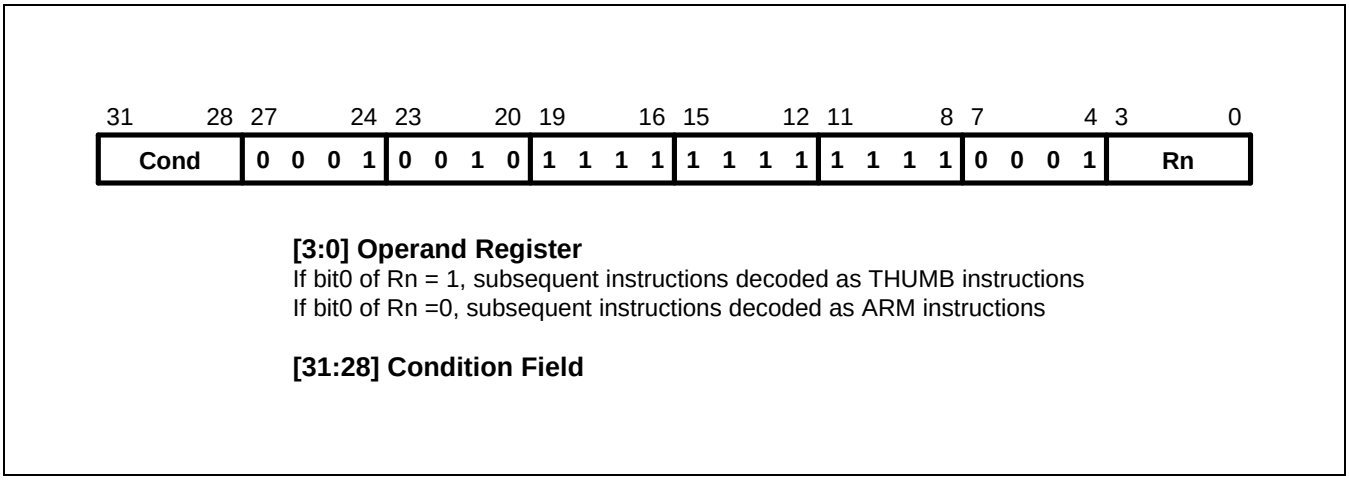


Figure 3-2. Branch and Exchange Instructions

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.

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 behavior 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.

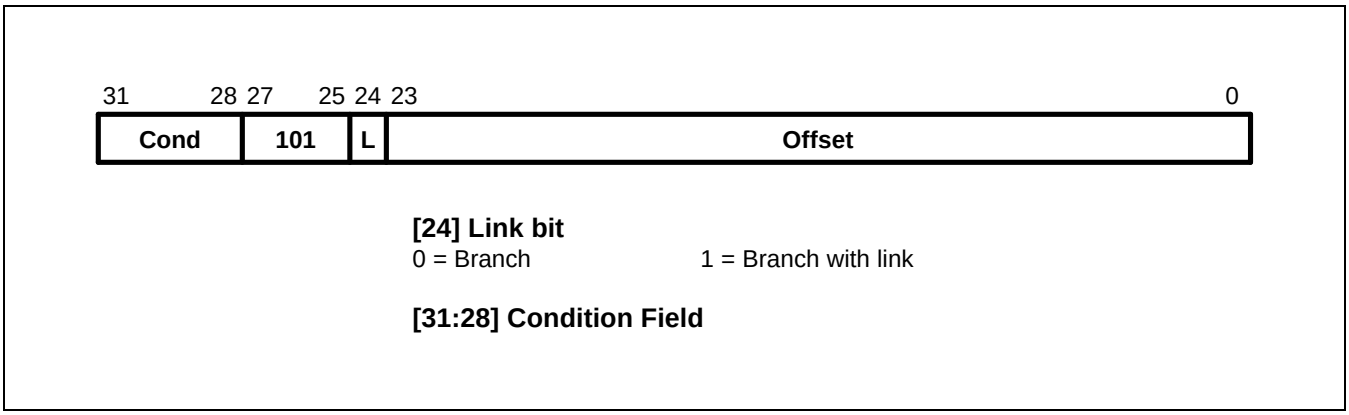


Figure 3-3. Branch Instructions

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>

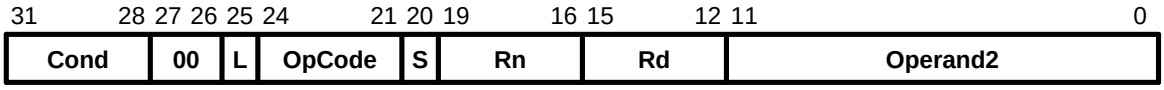
{L}	Used to request the Branch with Link form of the instruction. If absent, R14 will not be affected by the instruction.
{cond}	A two-character mnemonic as shown in Table 3-2. If absent then AL (ALways) will be used.
<expression>	The destination. The assembler calculates the offset.

EXAMPLES

```
here    BAL     here        ;  Assembles to 0xEAFFFFFE (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.



[15:12] Destination register

0 = Branch 1 = Branch with link

[19:16] 1st operand register

0 = Branch 1 = Branch with link

[20] Set condition codes

0 = Do not after condition codes 1 = Set condition codes

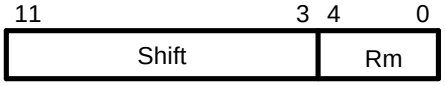
[24:21] Operation codes

0000 = AND-Rd: = Op1 AND Op2
0001 = EOR-Rd: = Op1 EOR Op2
0010 = SUB-Rd: = Op1-Op2
0011 = RSB-Rd: = Op2-Op1
0100 = ADD-Rd: = Op1+Op2
0101 = ADC-Rd: = Op1+Op2+C
0110 = SBC-Rd: = OP1-Op2+C-1
0111 = RSC-Rd: = Op2-Op1+C-1
1000 = TST-set condition codes on Op1 AND Op2
1001 = TEO-set condition codes on OP1 EOR Op2
1010 = CMP-set condition codes on Op1-Op2
1011 = SMN-set condition codes on Op1+Op2
1100 = ORR-Rd: = Op1 OR Op2
1101 = MOV-Rd: =Op2
1110 = BIC-Rd: = Op1 AND NOT Op2
1111 = MVN-Rd: = NOT Op2

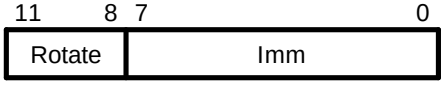
[25] Immediate operand

0 = Operand 2 is a register 1 = Operand 2 is an immediate value

[11:0] Operand 2 type selection



[3:0] 2nd operand register [11:4] Shift applied to Rm



[7:0] Unsigned 8 bit immediate value [11:8] Shift applied to Imm

[31:28] Condition field

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 operand2
EOR	0001	Operand1 EOR operand2
WUB	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.

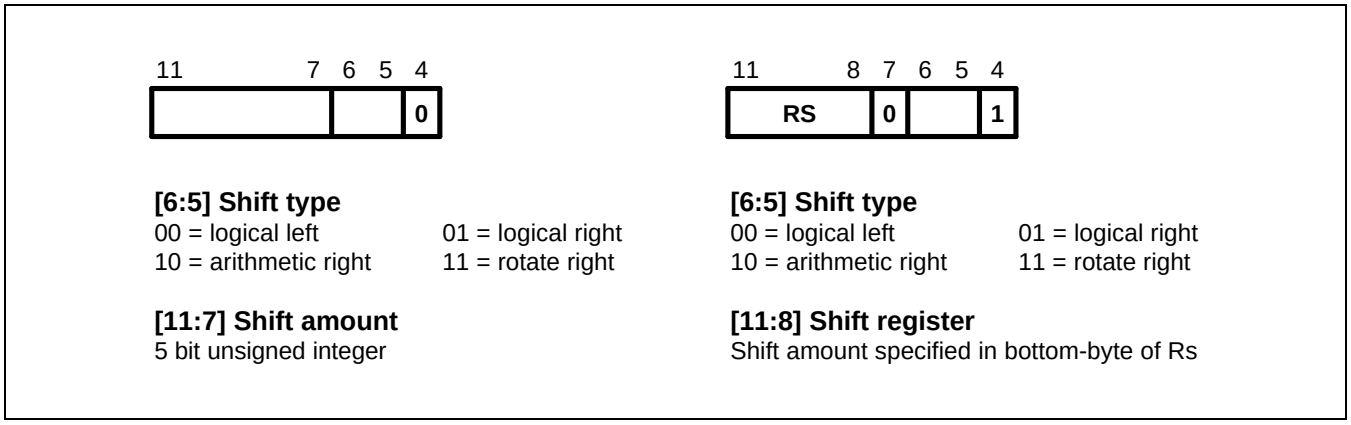


Figure 3-5. ARM Shift Operations

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.

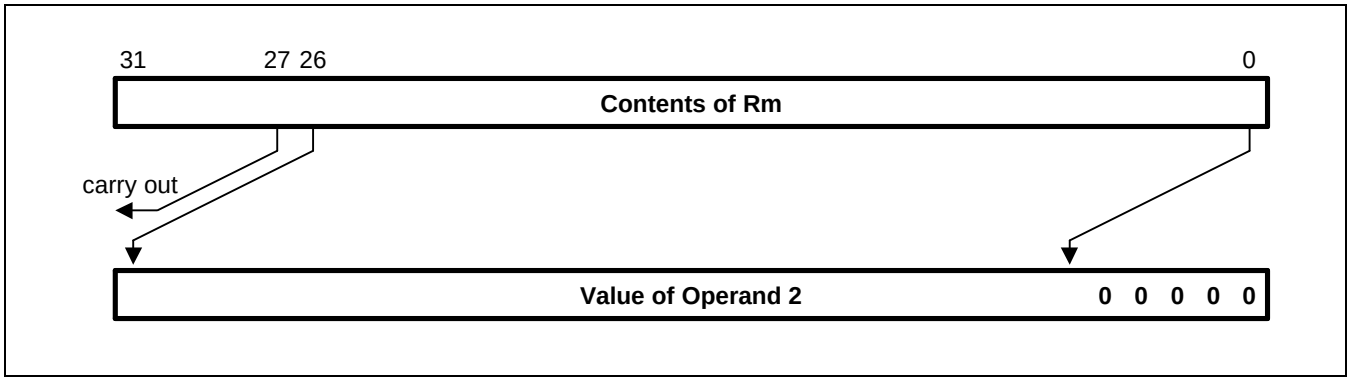


Figure 3-6. Logical Shift Left

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.

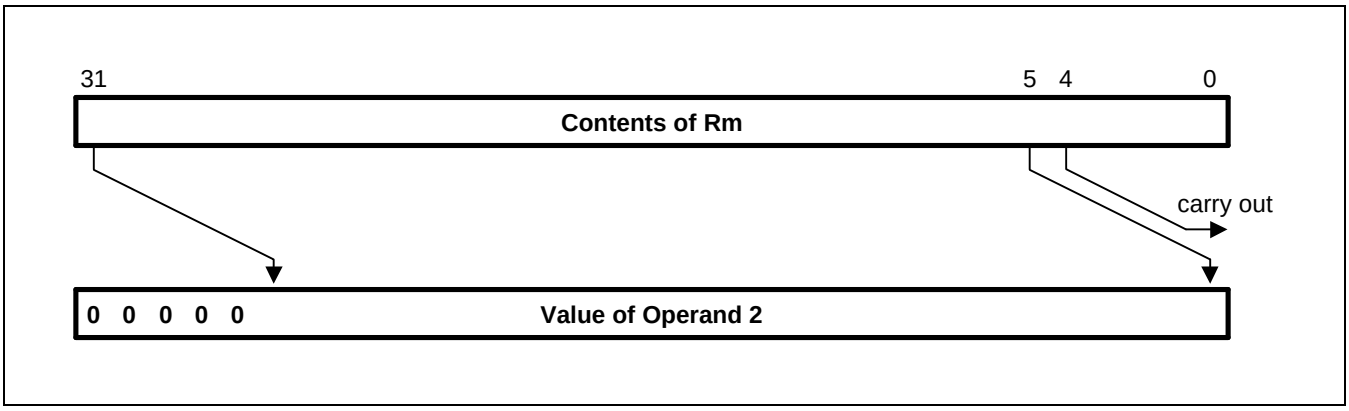


Figure 3-7. Logical Shift Right

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.

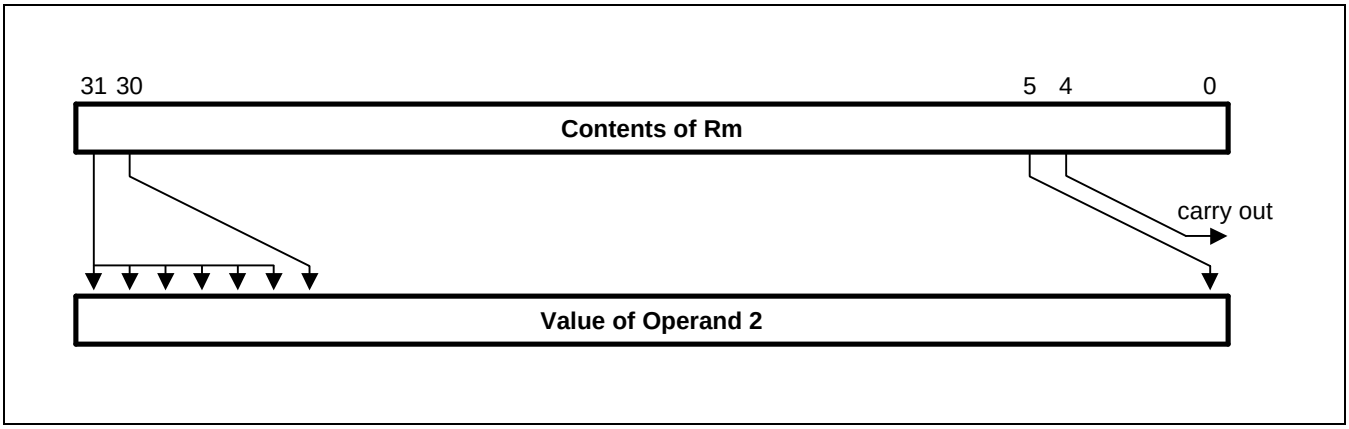


Figure 3-8. Arithmetic Shift Right

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.

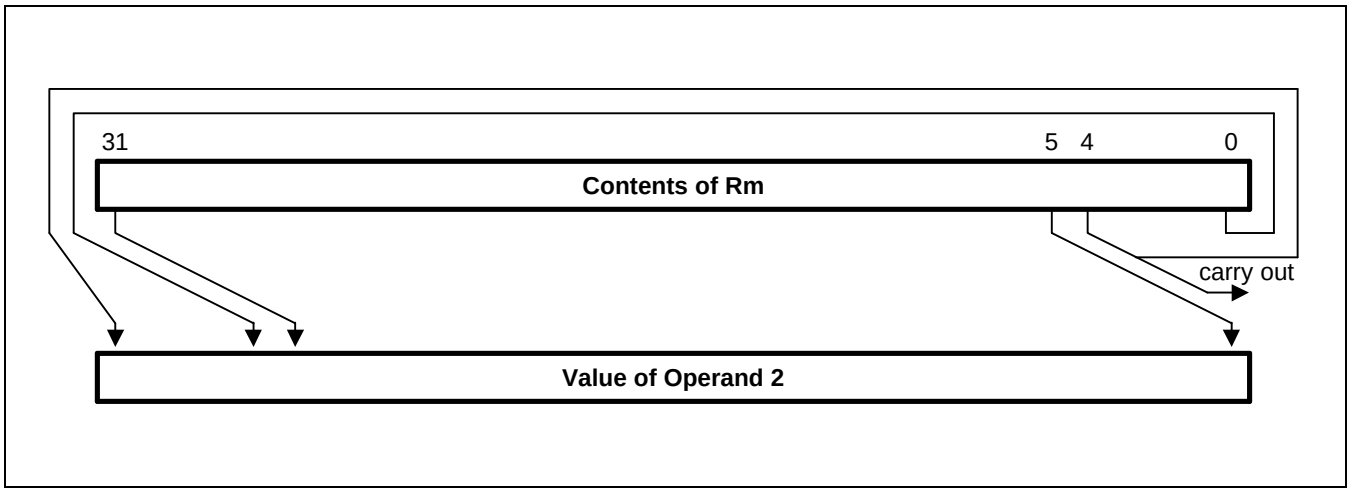


Figure 3-9. Rotate Right

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.

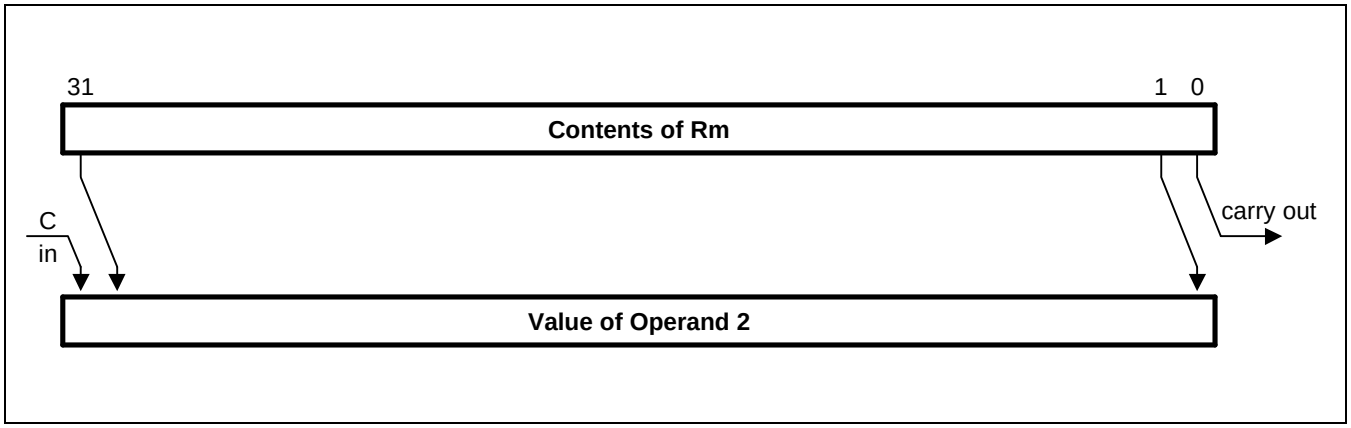


Figure 3-10. Rotate Right Extended

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 changes which atomically restore both PC and CPSR. This form of instruction should not be used in User mode.

USING R15 AS AN OPERANDY

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 modify

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>

where:

<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.

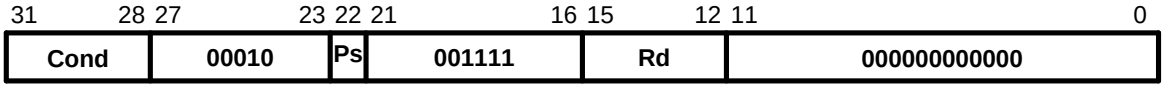
These instructions allow access to the CPSR and SPSR registers. The MRS instruction allows the contents of the CPSR or SPSR_<mode> to be moved to a general register. The MSR instruction allows the contents of a general register to be moved to the CPSR or SPSR_<mode> register.

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)

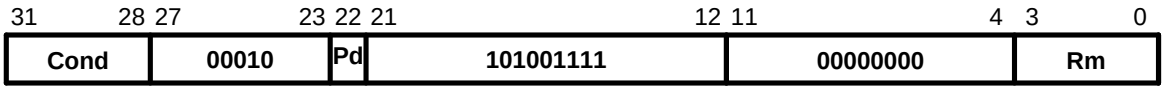


[15:21] Destination Register

[19:16] Source PSR
0 = CPSR 1 = SPSR_<current mode>

[31:28] Condition Field

MRS (transfer register contents to PSR)

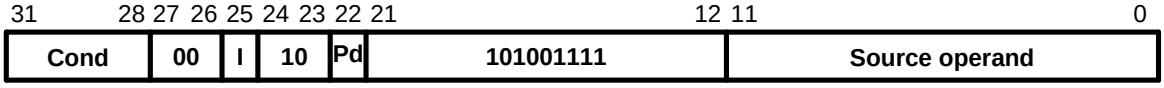


[3:0] Source Register

[22] Destination PSR
0 = CPSR 1 = SPSR_<current mode>

[31:28] Condition Field

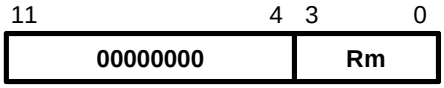
MRS (transfer register contents or immediate value to PSR flag bits only)



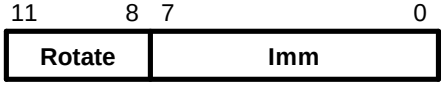
[22] Destination PSR
0 = CPSR 1 = SPSR_<current mode>

[25] Immediate Operand
0 = Source operand is a register
1 = SPSR_<current mode>

[11:0] Source Operand



[3:0] 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] Condition 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).

ASSEMBLY 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 symbolize 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.

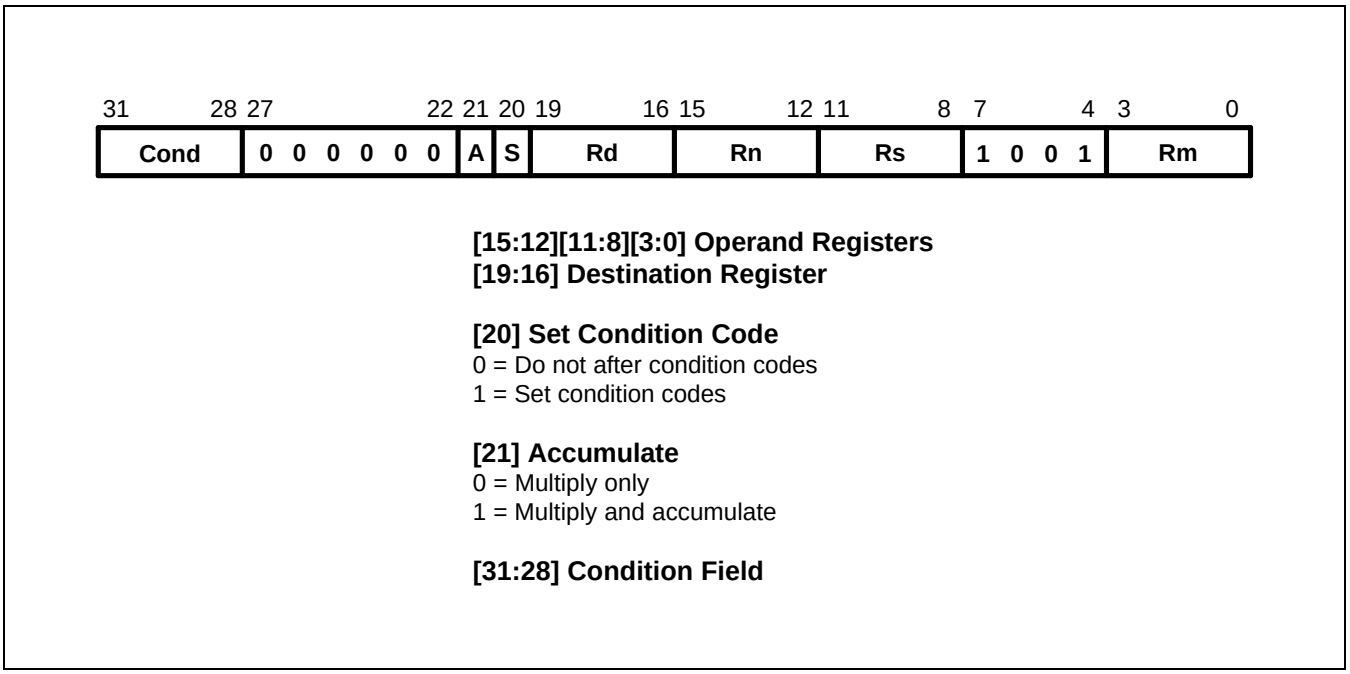


Figure 3-12. Multiply Instructions

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
0xFFFFFFF6	0x0000001	0xFFFFFF38

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 0xFFFFFF38.

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 0xFFFFFF38.

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.

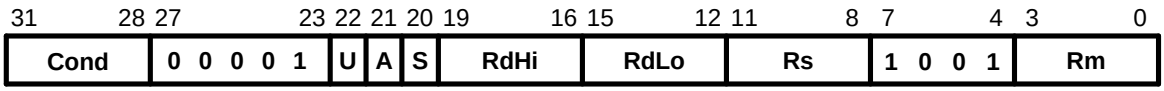
EXAMPLES

```
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.



[11:8][3:0] Operand Registers
[19:16][15:12] Source Destination Registers

[20] Set Condition Code
0 = Do not alter condition codes
1 = Set condition codes

[21] Accumulate
0 = Multiply only
1 = Multiply and accumulate

[22] Unsigned
0 = Unsigned
1 = Signed

[31:28] Condition Field

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 x 32 = 64
UMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply & Accumulate Long	32 x 32 + 64 = 64
SMULL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply Long	32 x 32 = 64
SMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply & Accumulate Long	32 x 32 + 64 = 64

where:

{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

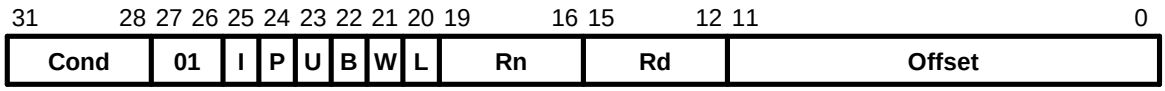
```
UMULL    R1,R4,R2,R3      ; R4,R1:=R2×R3
UMLALS   R1,R5,R2,R3      ; R5,R1:=R2×R3+R5,R1 also setting condition codes
```

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.



[15:12] Source/Destination Registers

[19:16] Base Register

[20] Load/Store Bit
0 = Store to memory
1 = Load from memory

[21] Write-back Bit
0 = No write-back
1 = Write address into base

[22] Byte/Word Bit
0 = Transfer word quantity
1 = Transfer byte quantity

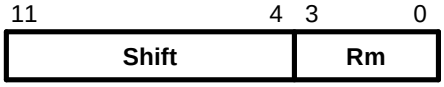
[23] Up/Down Bit
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing Bit
0 = Post: add offset after transfer
1 = Pre: add offset before transfer

[25] Immediate Offset
0 = Offset is an immediate value

[11:0] Offset

[11:0] Unsigned 12-bit immediate offset



[3:0] Offset register [11:4] Shift applied to Rm

[31:28] Condition Field

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 Rn. 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.

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.

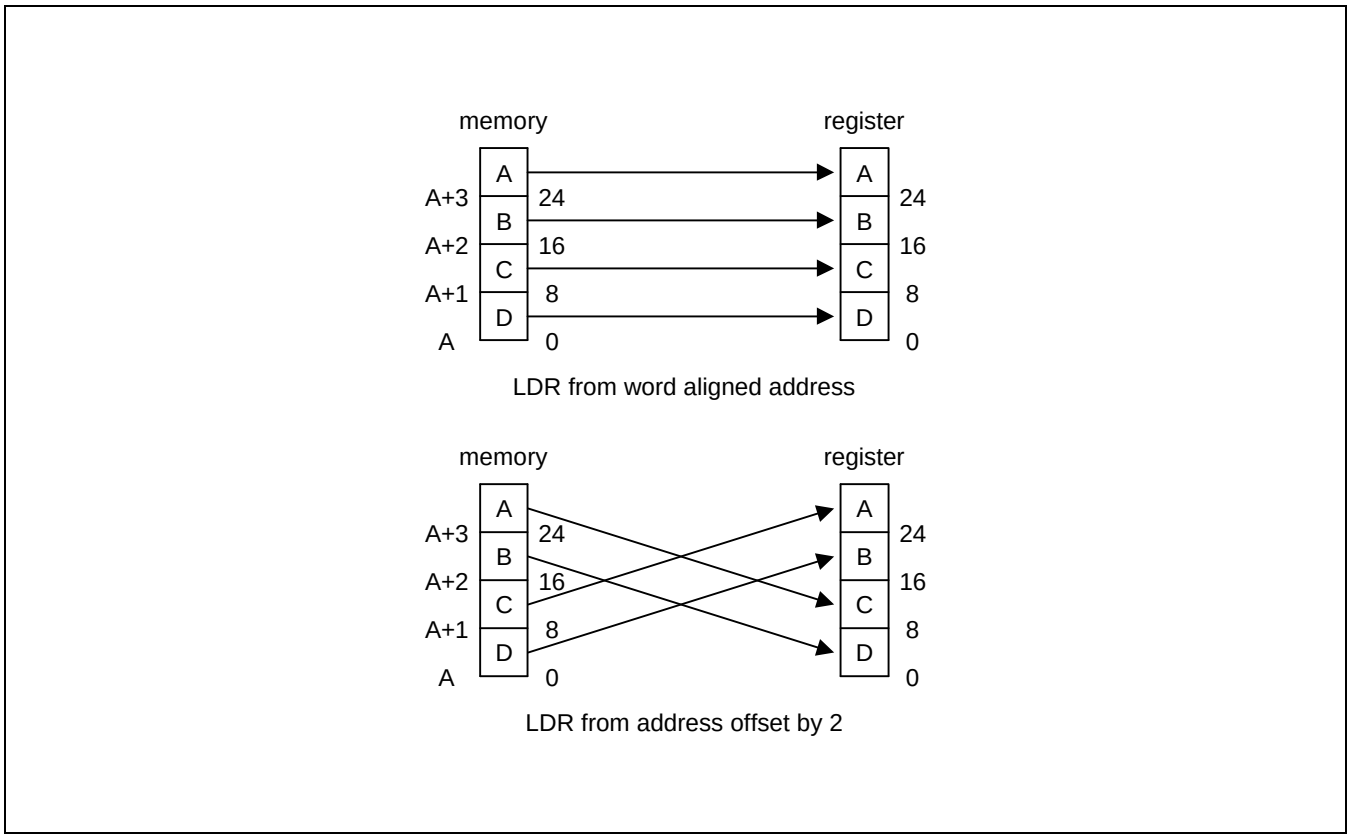


Figure 3-15. Little-Endian Offset Addressing

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	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{,<shift>}]{!}	offset of +/– contents of index register, shifted by <shift>
3	A post-indexed addressing specification:	
	[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.	

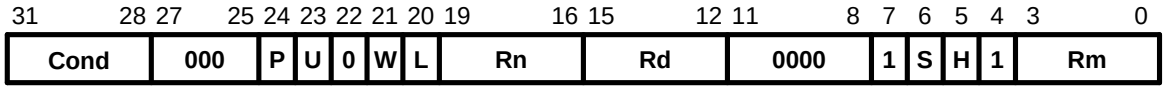
EXAMPLES

```
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.



[3:0] Offset Register

[6][5] S H
0 0 = SWP instruction
0 1 = Unsigned halfword
1 1 = Signed byte
1 1 = Signed halfword

[15:12] Source/Destination Register

[19:16] Base Register

[20] Load/Store
0 = Store to memory
1 = Load from memory

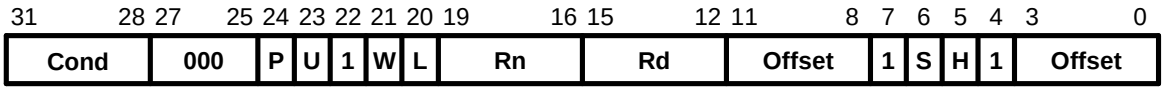
[21] Write-back
0 = No write-back
1 = Write address into base

[23] Up/Down
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing
0 = Post: add/subtract offset after transfer
1 = Pre: add/subtract offset bofore transfer

[31:28] Condition Field

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



[3:0] Immediate Offset (Low Nibble)

[6][5] S H
0 0 = SWP instruction
0 1 = Unsigned halfword
1 1 = Signed byte
1 1 = Signed halfword

[11:8] Immediate Offset (High Nibble)

[15:12] Source/Destination Register

[19:16] Base Register

[20] Load/Store
0 = Store to memory
1 = Load from memory

[21] Write-back
0 = No write-back
1 = Write address into base

[23] Up/Down
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing
0 = Post: add/subtract offset after transfer
1 = Pre: add/subtract offset bofore transfer

[31:28] Condition Field

Figure 3-17. Halfword and Signed 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 Rn. 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 behavior.

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 behavior.

USE OF R15

Write-back should 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 should not be specified as the register offset (Rm).

When R15 is the source register (Rd) 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–HERE8)]; 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.

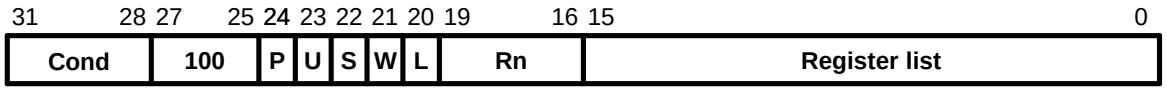
Block data transfer instructions are used to load (LDM) or store (STM) any subset of the currently visible registers. They support all possible stacking modes, maintaining full or empty stacks which can grow up or down memory, and are very efficient instructions for saving or restoring context, or for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



[19:16] Base Register

[20] Load/Store Bit
0 = Store to memory
1 = Load from memory

[21] Write-back Bit
0 = No write-back
1 = Write address into base

[22] PSR & Force User Bit
0 = Do not load PSR or user mode
1 = Load PSR or force user mode

[23] Up/Down Bit
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing Bit
0 = Post: add offset after transfer
1 = Pre: add offset bofore transfer

[31:28] Condition Field

Figure 3-18. Block Data Transfer Instructions

ADDRESSING MODES

The transfer addresses are determined by the contents of the base register (Rn), the pre/post bit (P) and the up/down bit (U). The registers are transferred in the order lowest to highest, so R15 (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 R1, R5 and R7 in the case where Rn=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 Rn after the instruction has completed.

In all cases, had write back of the modified base not been required (W=0), Rn 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.

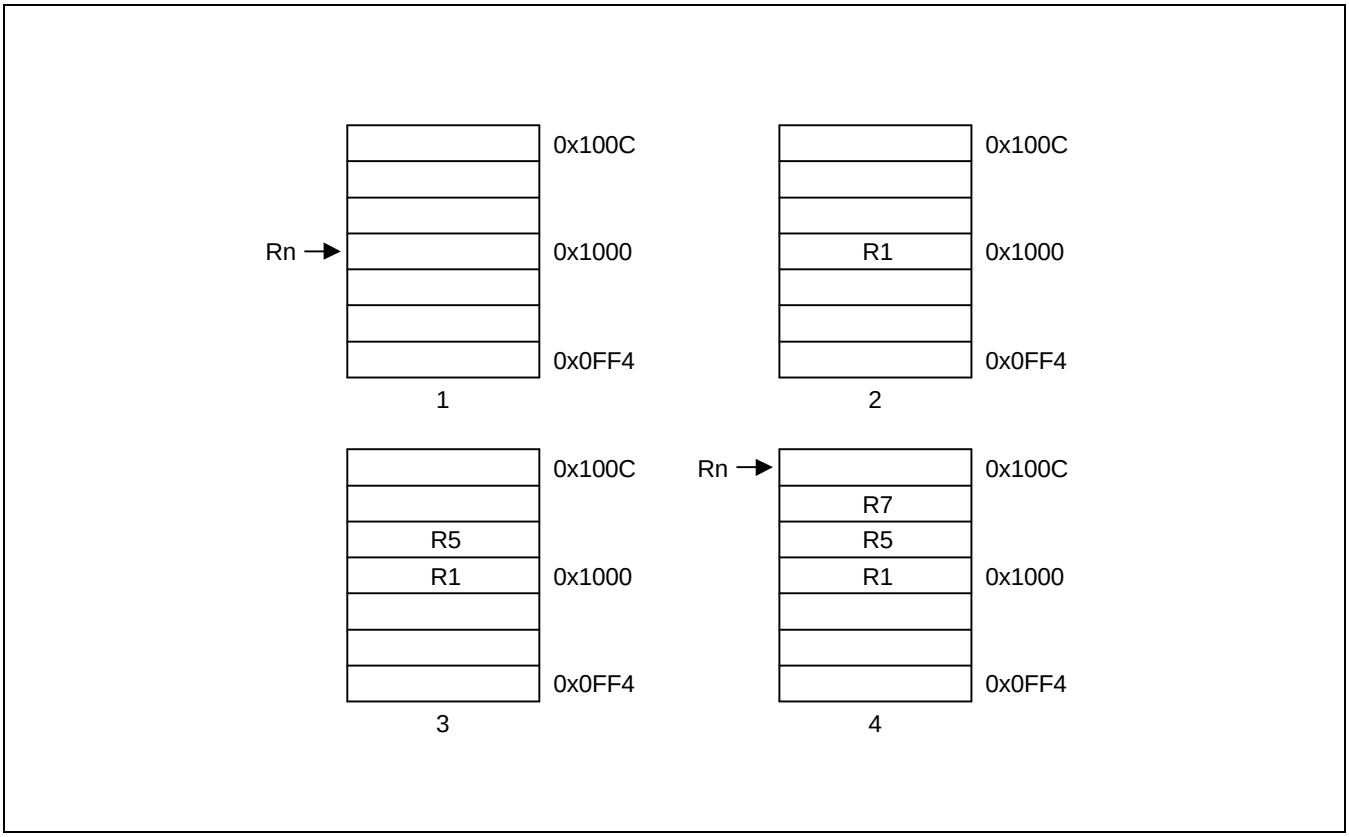


Figure 3-19. Post-Increment Addressing

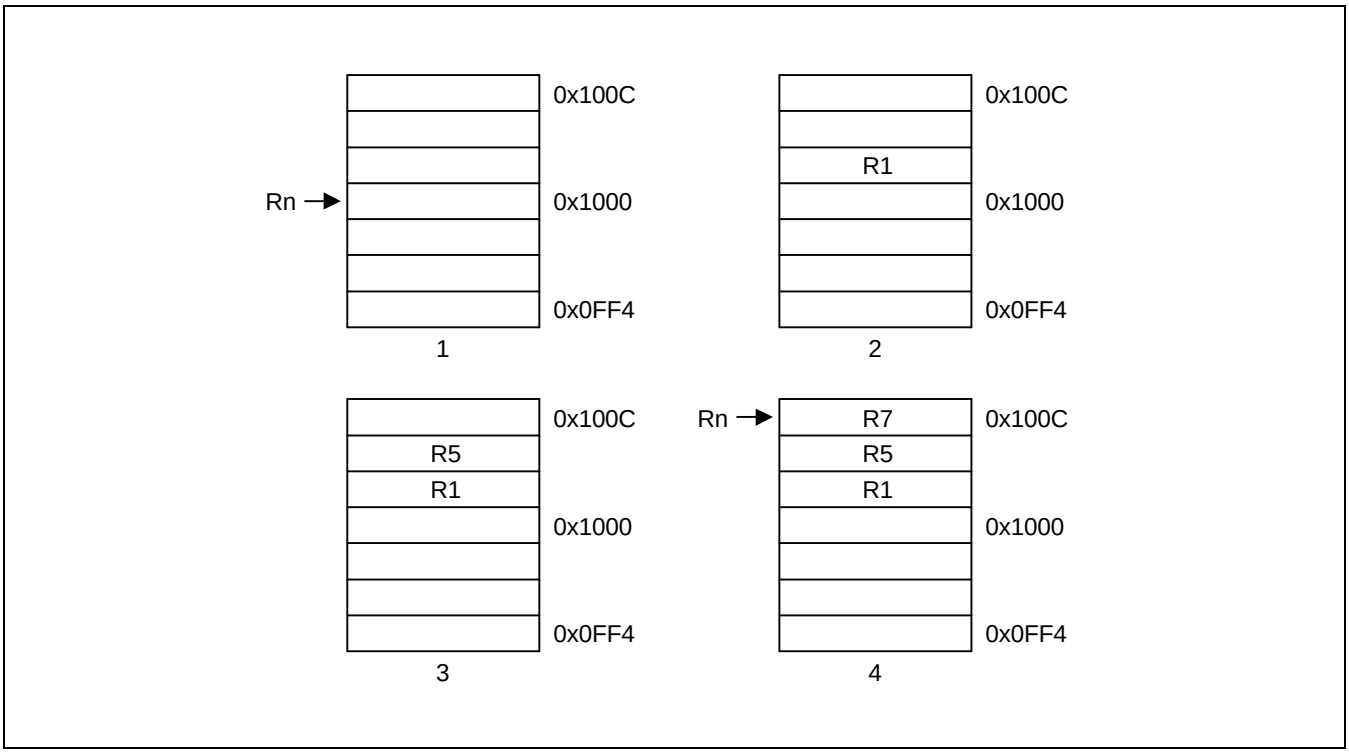


Figure 3-20. Pre-Increment Addressing

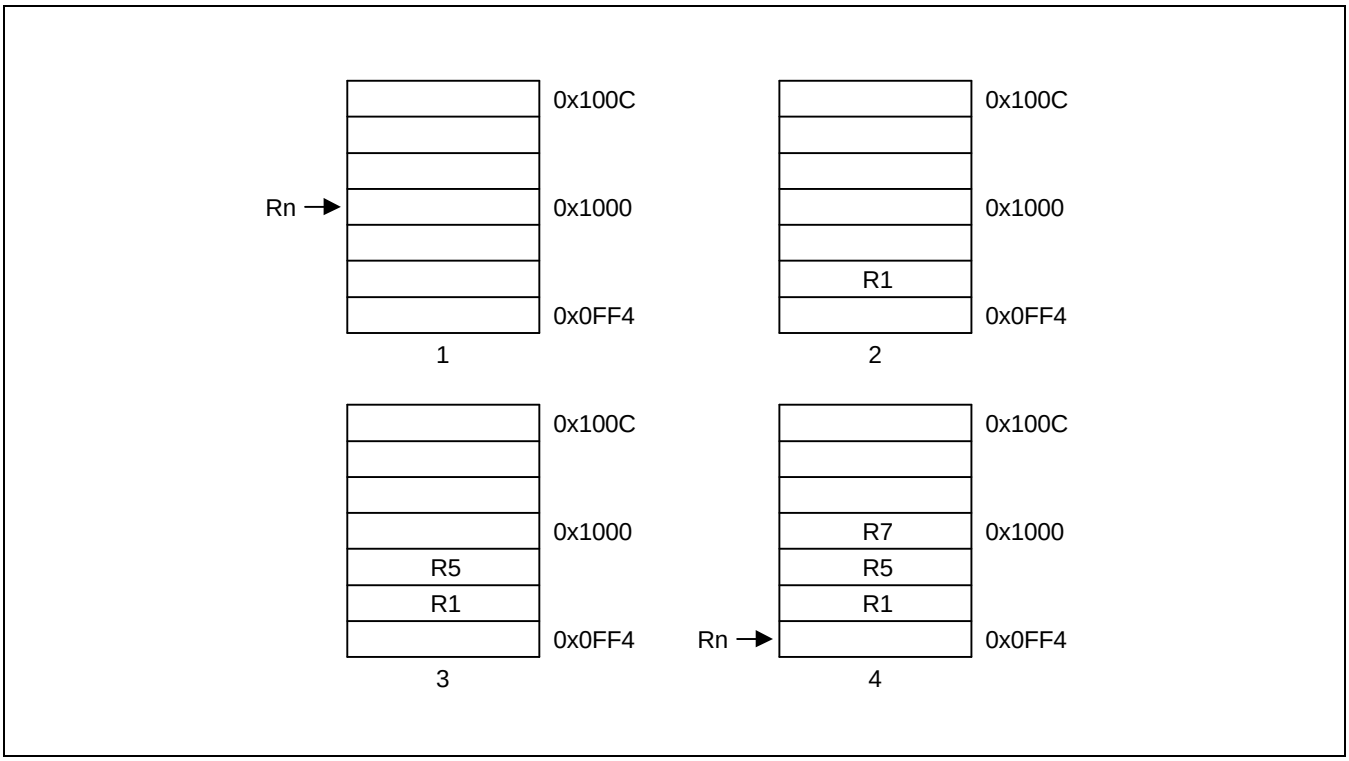


Figure 3-21. Post-Decrement Addressing

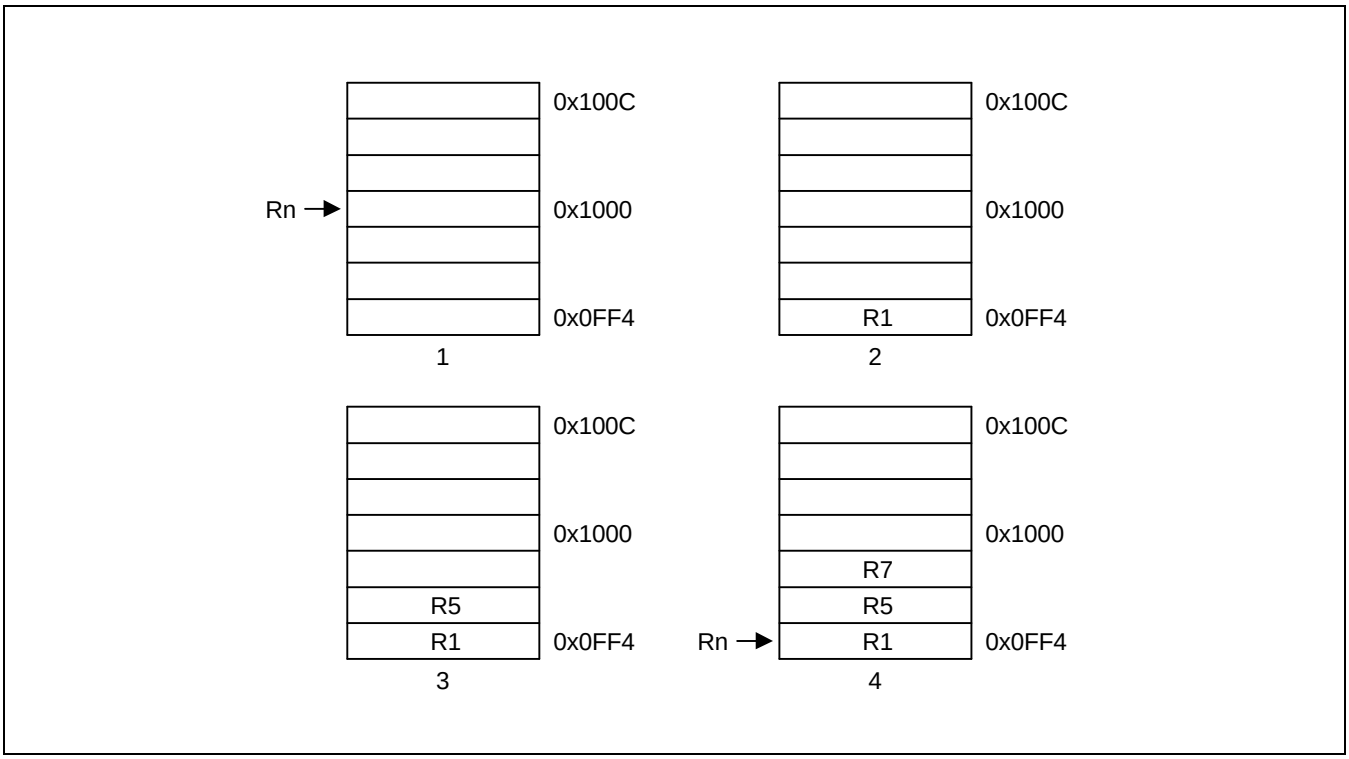


Figure 3-22. Pre-Decrement Addressing

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.

Abort 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.

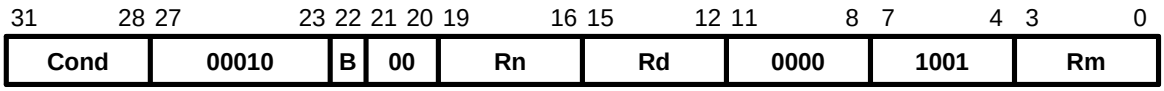
EXAMPLES

```
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)



[3:0] Source Register

[15:12] Destination Register

[19:16] Base Register

[22] Byte/Word Bit
0 = Swap word quantity
1 = Swap word quantity

[31:28] Condition Field

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.

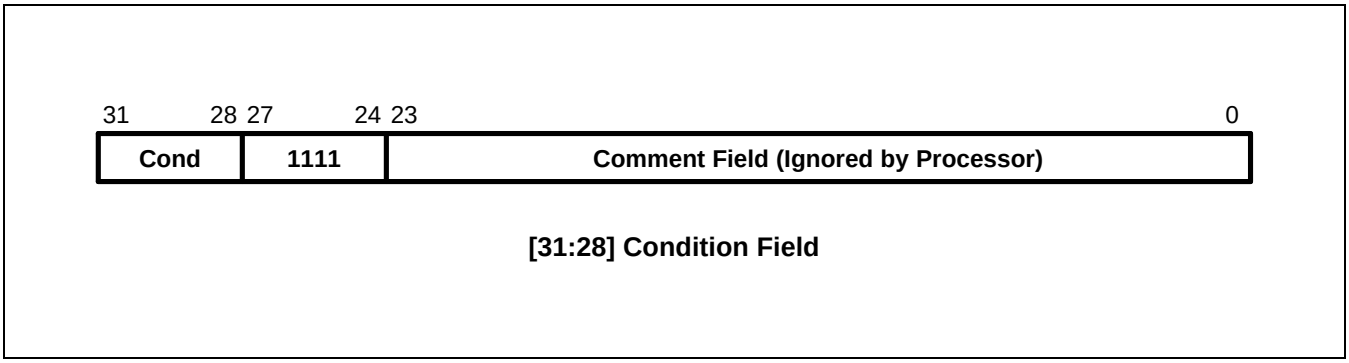


Figure 3-24. Software Interrupt Instruction

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     WriteI+"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 WriteIRtn
        • • •
        Zero        EQU  0
ReadC   EQU  256
WriteI  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.
        WriteIRtn                       ;  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 KS32C41000, 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 KS32C41000. These coprocessor instructions can be emulated by the undefined trap handler. Even though external coprocessor can not be connected to the KS32C41000, the coprocessor instructions are still described here in full for completeness. (Remember that any external coprocessor described in this section is a software emulation.)

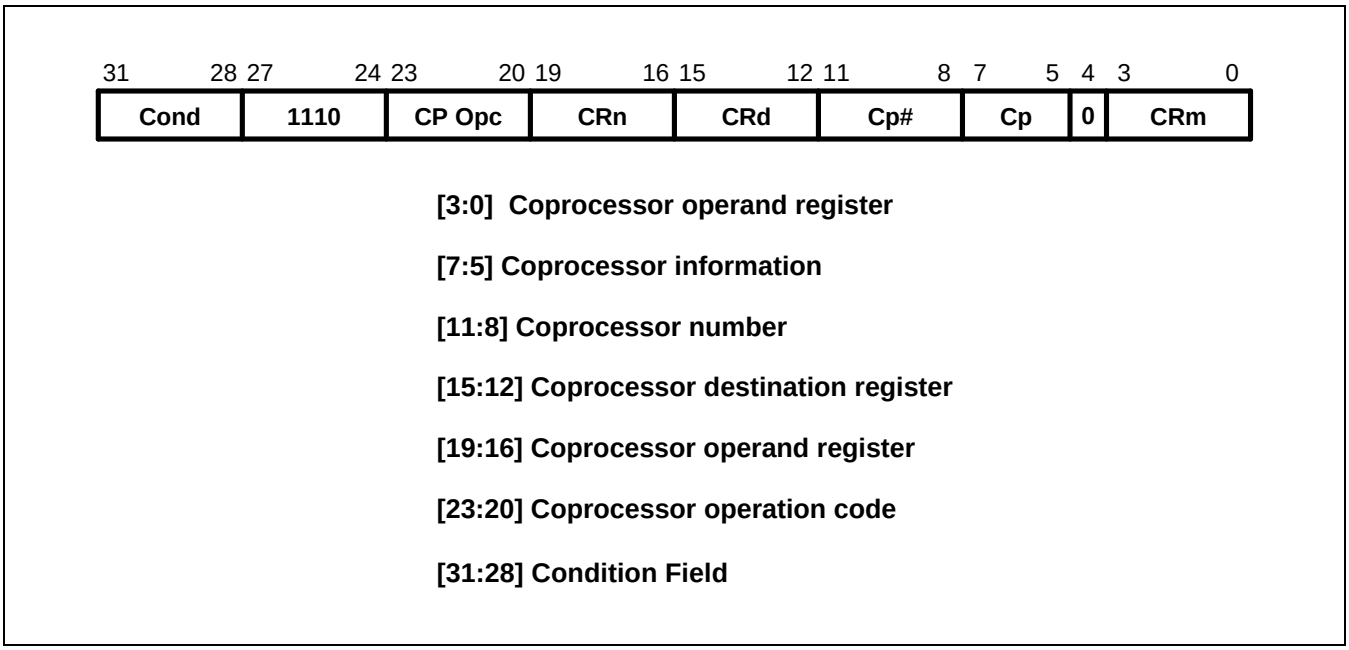


Figure 3-25. Coprocessor Data Operation Instruction

Only bit 4 and bits 24 to 31 The coprocessor fields 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

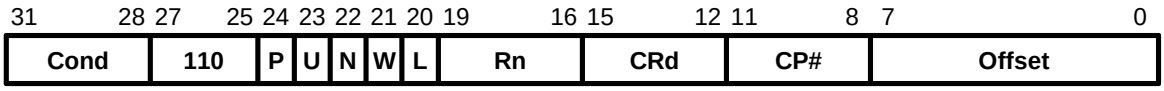
EXAMPLES

```
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.



[7:0] Unsigned 8 Bit Immediate Offset

[11:8] Coprocessor Number

[15:12] Coprocessor Source/Destination Register

[19:16] Base Register

[20] Load/Store Bit
0 = Store to memory
1 = Load from memory

[21] Write-back Bit
0 = No write-back
1 = Write address into base

[22] Transfer Length

[23] Up/Down Bit
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing Bit
0 = Post: add offset after transfer
1 = Pre: add offset before transfer

[31:28] Condition Field

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 (Rn); 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 Rn 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 + bI 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.

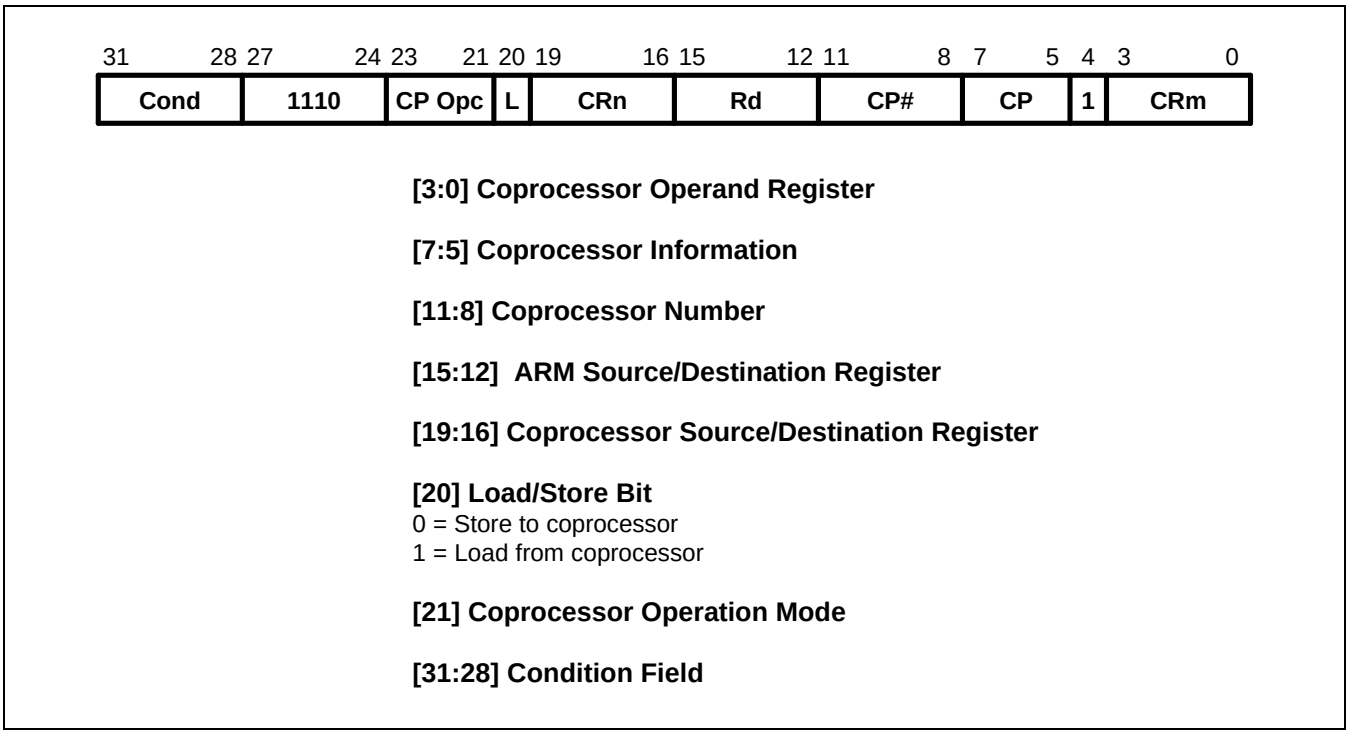


Figure 3-27. Coprocessor Register Transfer Instructions

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.

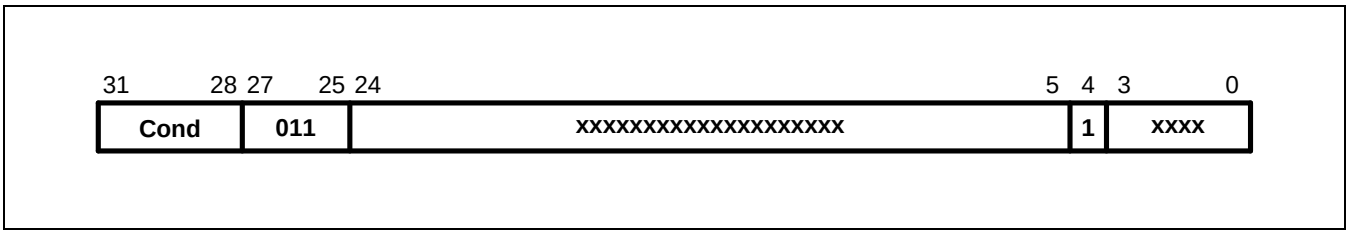


Figure 3-28. Undefined Instruction

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           ;  If Rn=p OR Rm=q THEN GOTO Label.
BEQ     Label
CMP     Rm,#q
BEQ     Label
```

This can be replaced by

```
CMP     Rn,#p
CMPNE   Rm,#q           ;  If condition not satisfied try other test.
BEQ     Label
```

Absolute Value

```
TEQ     Rn,#0           ;  Test sign
RSBMI   Rn,Rn,#0        ;  and 2's complement if necessary.
```

Multiplication by 4, 5 or 6 (Run Time)

```
MOV     Rc,Ra,LSL#2     ;  Multiply by 4,
CMP     Rb,#5           ;  Test value,
ADDCS   Rc,Rc,Ra        ;  Complete multiply by 5,
ADDHI   Rc,Rc,Ra        ;  Complete multiply by 6.
```

Combining Discrete and Range Tests

```
TEQ     Rc,#127         ;  Discrete test,
CMPNE   Rc,# " "–1      ;  Range test
MOVLS   Rc,# ""         ;  IF Rc<= "" 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.

```
                                        ;  Enter with numbers in Ra and Rb.
        MOV     Rcnt,#1                 ;  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   Rl,Rh,Rm,Rn     ;  3 to 6 cycles
ADDS    Rl,Rl,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   Rl,Rh,Rm,Rn     ;  3 to 6 cycles
ADDS    Rl,Rl,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 eor 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 MOD 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 MOD 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

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

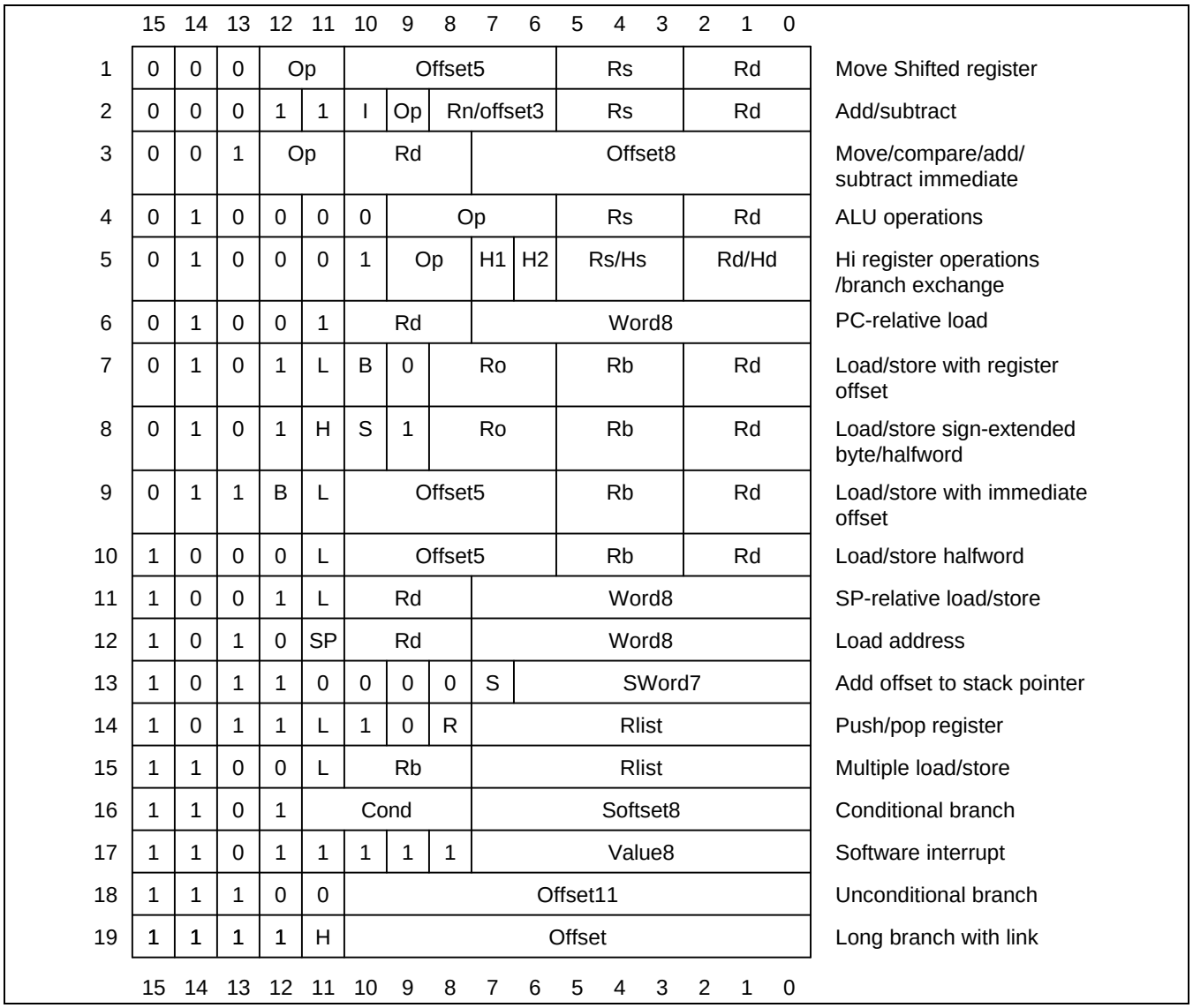


Figure 3-29. THUMB Instruction Set Formats

OPCODE SUMMARY

The following table summarizes 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	Y	–	Y
ADD	Add	Y	Y	Y (1)
AND	AND	Y	–	Y
ASR	Arithmetic Shift Right	Y	–	Y
B	Unconditional branch	Y	–	–
Bxx	Conditional branch	Y	–	–
BIC	Bit Clear	Y	–	Y
BL	Branch and Link	–	–	–
BX	Branch and Exchange	Y	Y	–
CMN	Compare Negative	Y	–	Y
CMP	Compare	Y	Y	Y
EOR	EOR	Y	–	Y
LDMIA	Load multiple	Y	–	–
LDR	Load word	Y	–	–
LDRB	Load byte	Y	–	–
LDRH	Load halfword	Y	–	–
LSL	Logical Shift Left	Y	–	Y
LDSB	Load sign-extended byte	Y	–	–
LDSH	Load sign-extended halfword	Y	–	–
LSR	Logical Shift Right	Y	–	Y
MOV	Move register	Y	Y	Y (2)
MUL	Multiply	Y	–	Y
MVN	Move Negative register	Y	–	Y

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

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
NEG	Negate	Y	–	Y
ORR	OR	Y	–	Y
POP	Pop register	Y	–	–
PUSH	Push register	Y	–	–
ROR	Rotate Right	Y	–	Y
SBC	Subtract with Carry	Y	–	Y
STMIA	Store Multiple	Y	–	–
STR	Store word	Y	–	–
STRB	Store byte	Y	–	–
STRH	Store halfword	Y	–	–
SWI	Software Interrupt	–	–	–
SUB	Subtract	Y	–	Y
TST	Test bits	Y	–	Y

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

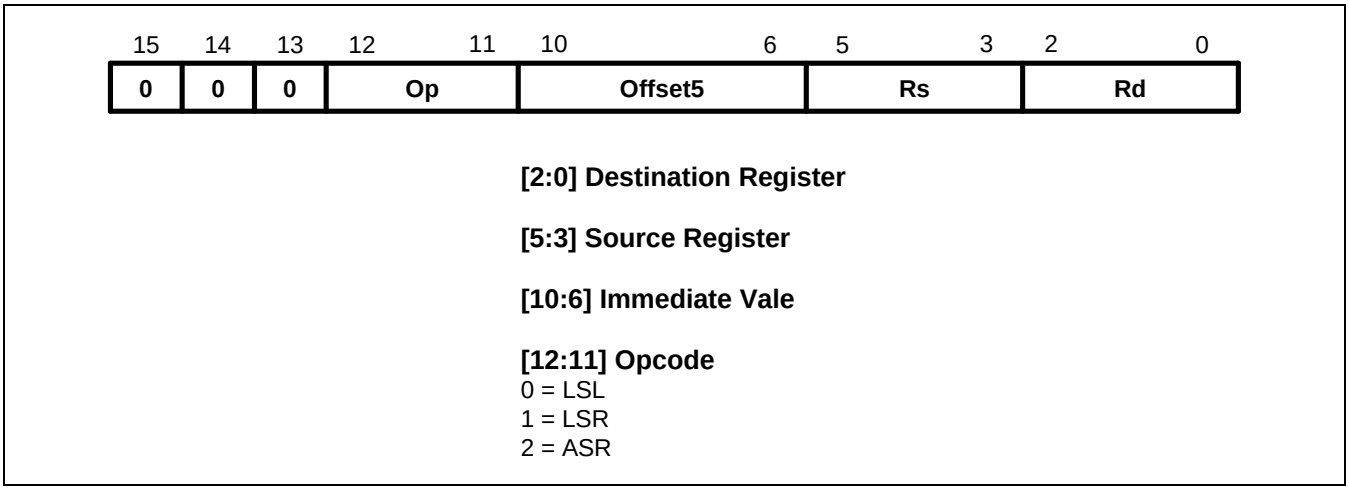


Figure 3-30. Format 1

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 Equipment	Action
00	LSL Rd, Rs, #Offset5	MOVS Rd, Rs, LSL #Offset5	Shift Rs left by a 5-bit immediate value and store the result in Rd.
01	LSR Rd, Rs, #Offset5	MOVS 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	MOVS Rd, Rs, ASR #Offset5	Perform arithmetic shift right on Rs by a 5-bit immediate value and store the result in Rd.

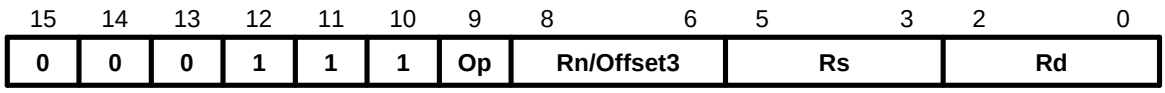
INSTRUCTION CYCLE TIMES

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



[2:0] Destination Register

[5:3] Source Register

[8:6] Register/Immediate Vale

[9] Opcode
0 = ADD
1 = SUB

[10] Immediate Flag
0 = Register operand
1 = Immediate oerand

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 Equipment	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

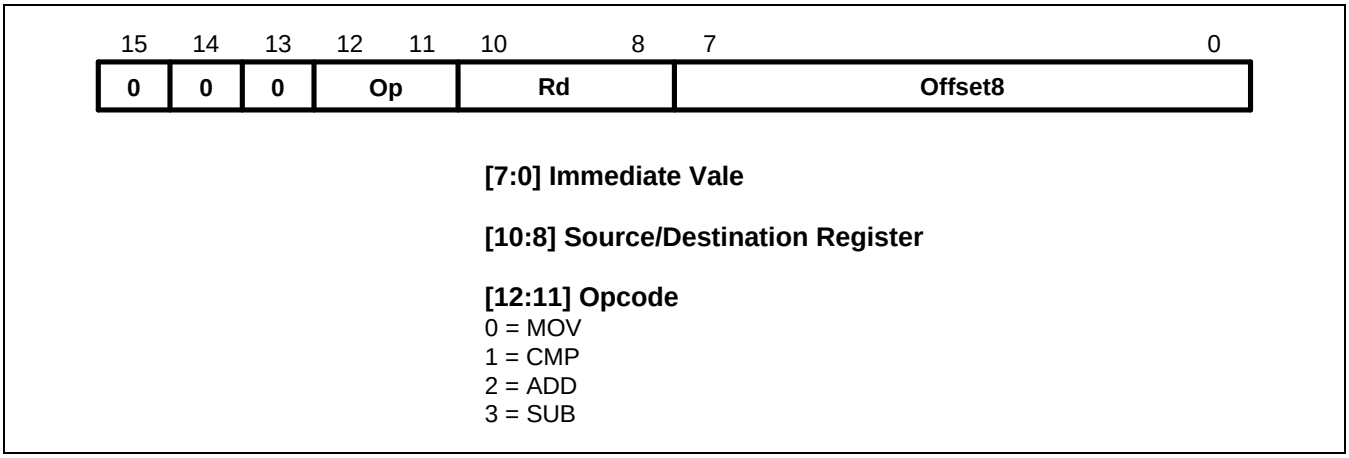


Figure 3-32. Format 3

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 Equipment	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.

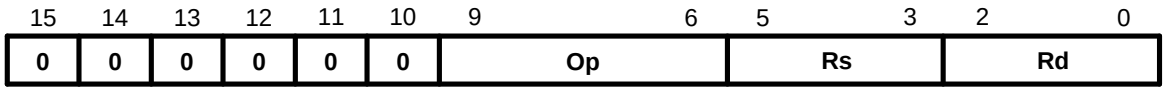
INSTRUCTION CYCLE TIMES

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



[2:0] Source/Destination Register

[5:3] Source Register 2

[9:6] Opcode

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 Equipment	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

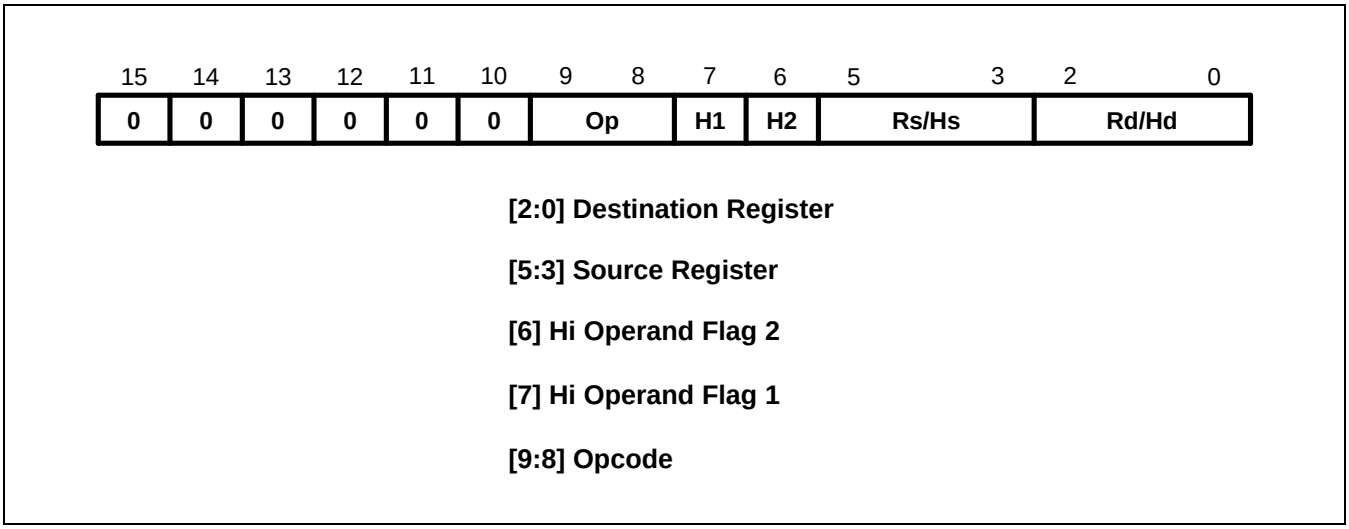


Figure 3-34. Format 5

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.

Table 3-12. Summary of Format 5 Instructions (Continued)

Op	H1	H2	THUMB assembler	ARM equivalent	Action
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

```
                        ;  Switch from THUMB to ARM state.
ADR     R1,outofTHUMB   ;  Load address of outofTHUMB into R1.
MOV     R11,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

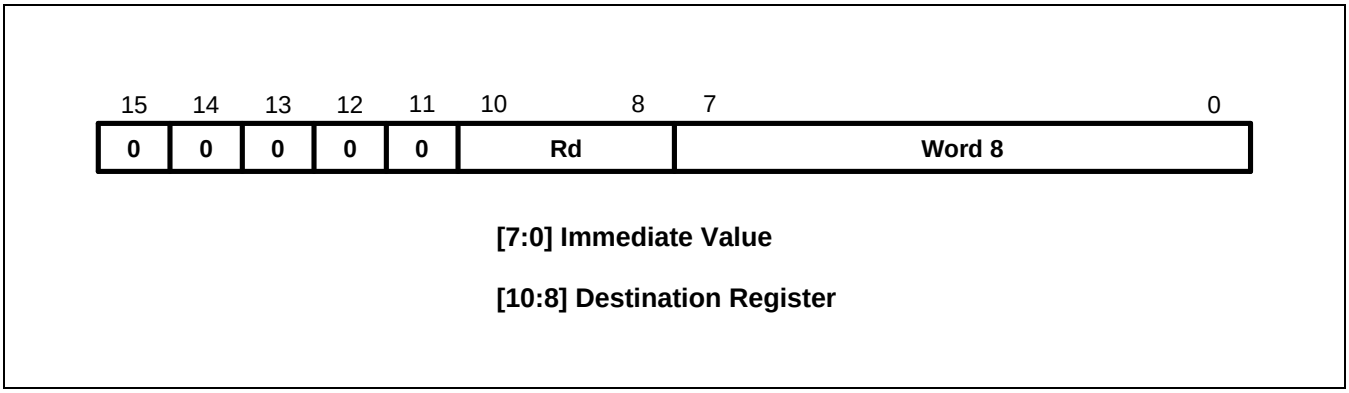


Figure 3-35. Format 6

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 (ie 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.

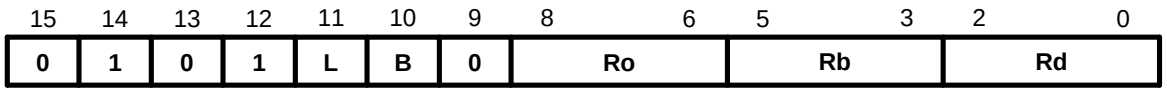
INSTRUCTION CYCLE TIMES

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



[2:0] Source/Destination Register

[5:3] Base Register

[8:6] Offset Register

[10] Byte/Word Flag
0 = Transfer word quantity
1 = Transfer byte quantity

[11] Load/Store Flag
0 = Store to memory
1 = Load from memory

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 3-14. 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 to R2.
LDRB    R2, [R0,R7]     ; Load into R2 the byte found at
                        ; the address formed by adding R7 to R0.
```

FORMAT 8: LOAD/STORE SIGN-EXTENDED BYTE/HALFWORD

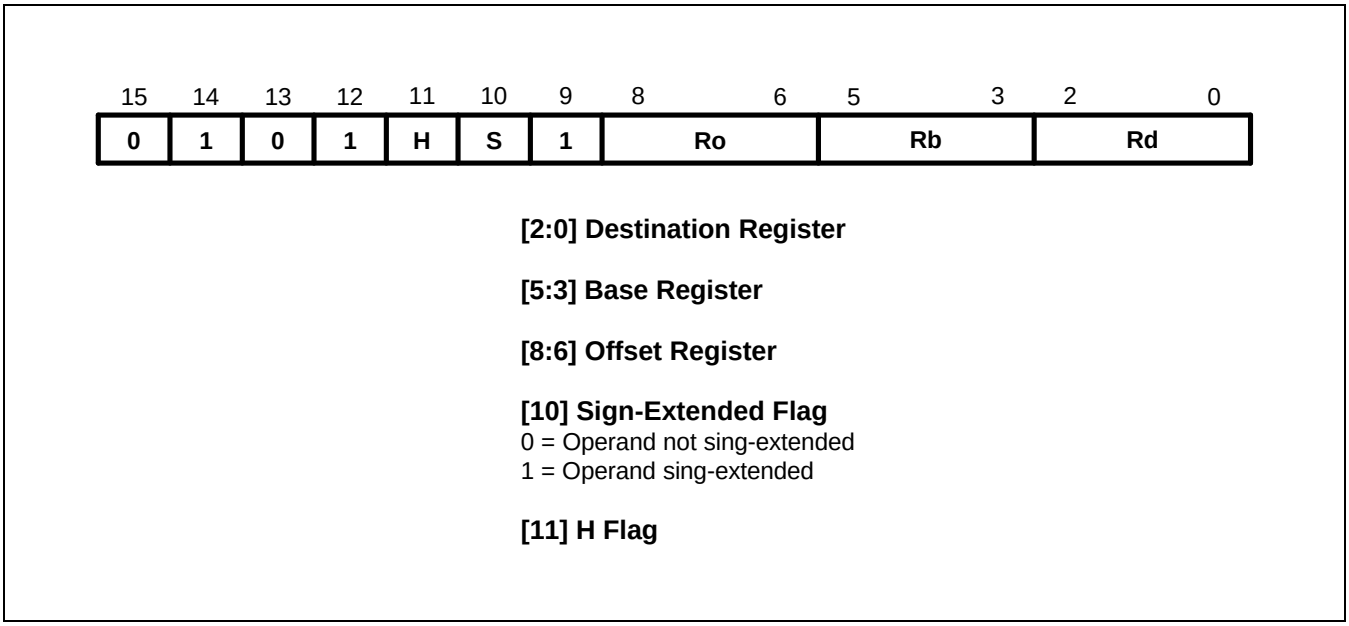


Figure 3-37. Format 8

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

L	B	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.

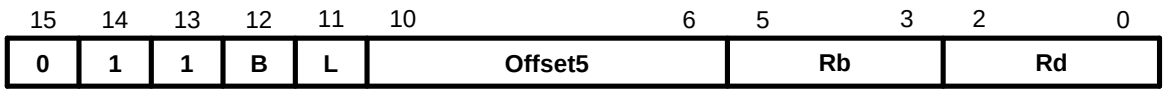
INSTRUCTION CYCLE TIMES

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 to R3.
LDSB    R2, [R7, R1]    ;  Load into R2 the sign extended byte
                        ;  found at the address formed by adding R1 to R7.
LDSH    R3, [R4, R2]    ;  Load into R3 the sign extended halfword
                        ;  found at the address formed by adding R2 to R4.
```

FORMAT 9: LOAD/STORE WITH IMMEDIATE OFFSET



[2:0] Source/Destination Register

[5:3] Base Register

[10:6] Offset Register

[11] Load/Store Flag
0 = Store to memory
1 = Load from memory

[12] Byte/Word Flad
0 = Transfer word quantity
1 = Transfer byte quantity

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

L	B	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.
1	0	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.
0	1	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 to 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 to R0.
                        ; Note that the THUMB opcode will
                        ; contain 13 as the Offset5 value.
```

FORMAT 10: LOAD/STORE HALFWORD

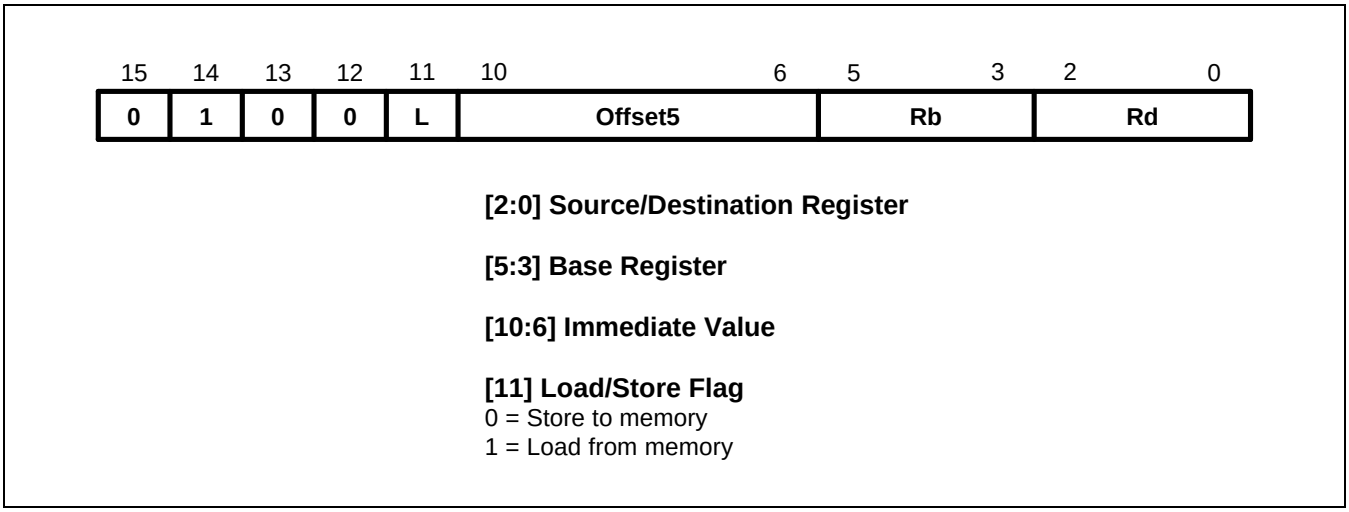


Figure 3-39. Format 10

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.

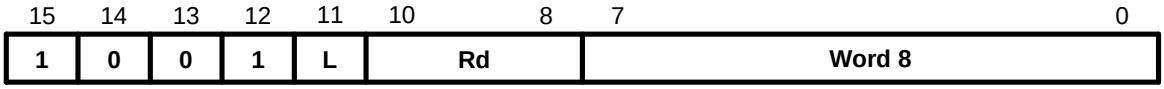
INSTRUCTION CYCLE TIMES

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 contain
                        ; 2 as the Offset5 value.
```

FORMAT 11: SP-RELATIVE LOAD/STORE



[7:0] Immediate Value

[10:8] Destination Register

[11] Load/Store Bit
0 = Store to memory
1 = Load from memory

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.

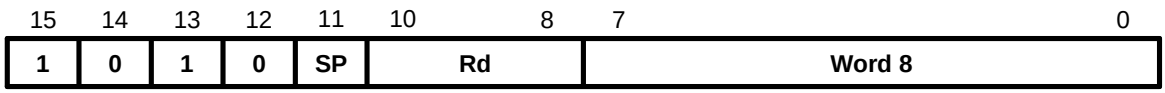
INSTRUCTION CYCLE TIMES

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



[7:0] 8-bit Unsigned Constant

[10:8] Destination Register

[11] Source
0 = PC
1 = SP

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

L	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

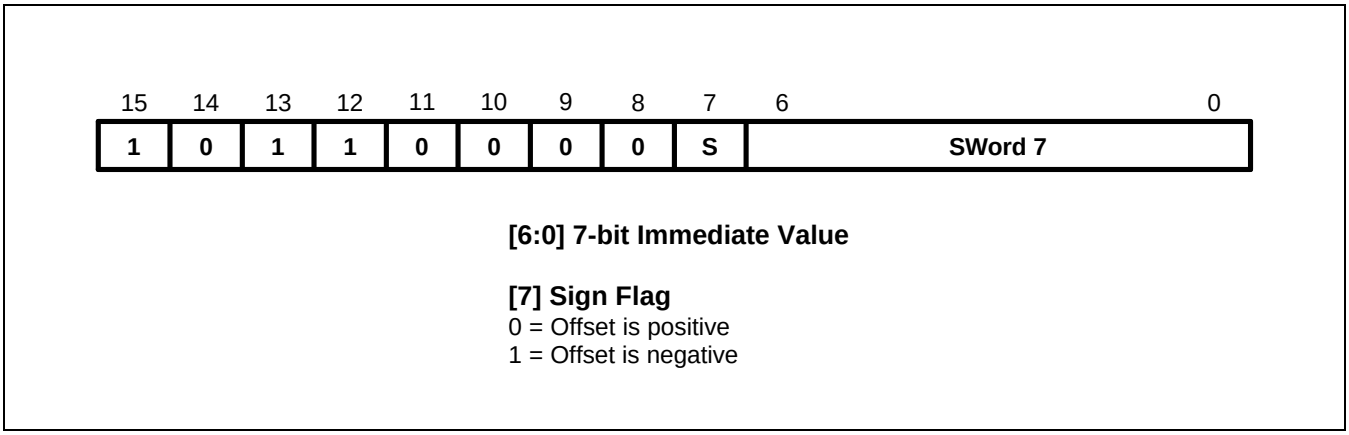


Figure 3-42. Format 13

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 Instruction

L	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 (ie 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.

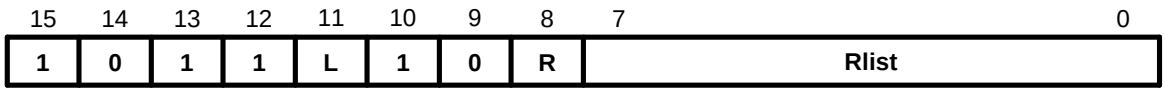
INSTRUCTION CYCLE TIMES

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

```
ADD     SP, #268        ;  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.
ADD     SP, #–104       ;  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



[7:0] Register List

[8] PC/LR Bit
0 = Do not store LR/Load PC
1 = Store LR/Load PC

[11] Load/Store Bit
0 = Store to memory
1 = Load from memory

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

L	B	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

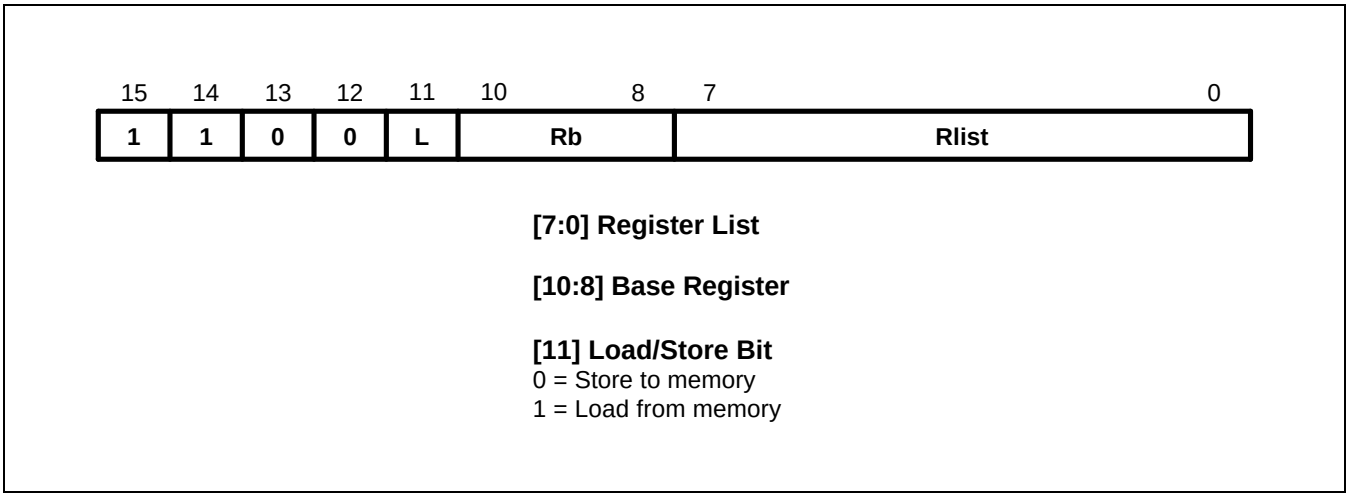


Figure 3-44. Format 15

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

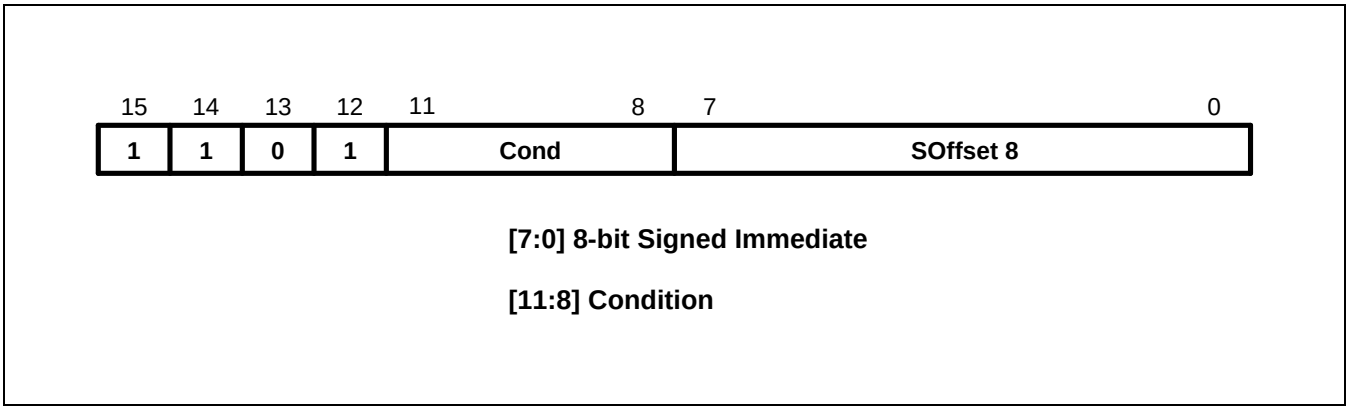


Figure 3-45. Format 16

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 2-23. The Conditional Branch Instructions

L	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)

Table 2-23. The Conditional Branch Instructions (Continued)

L	THUMB assembler	ARM equivalent	Action
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

1. While label specifies a full 9-bit two's complement address, this must always be halfword-aligned (ie 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

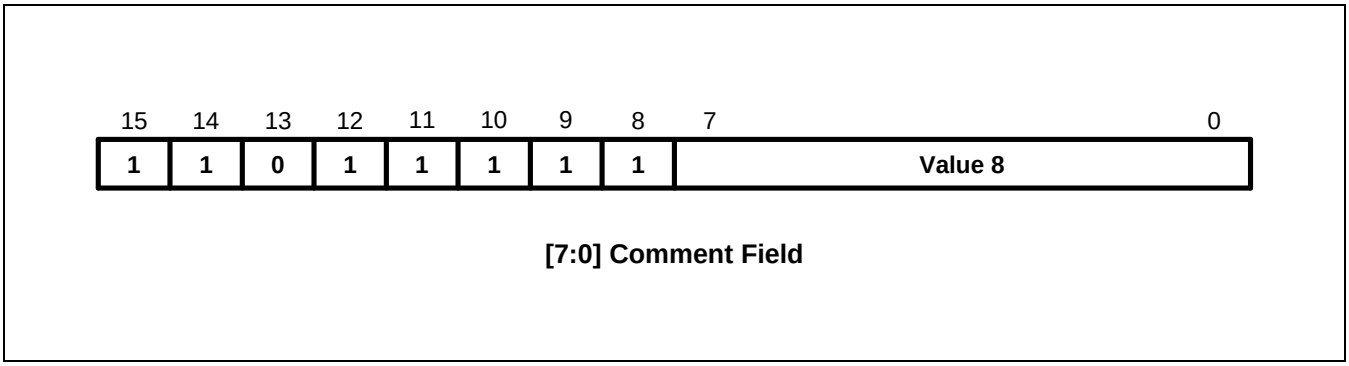


Figure 3-46. Format 17

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

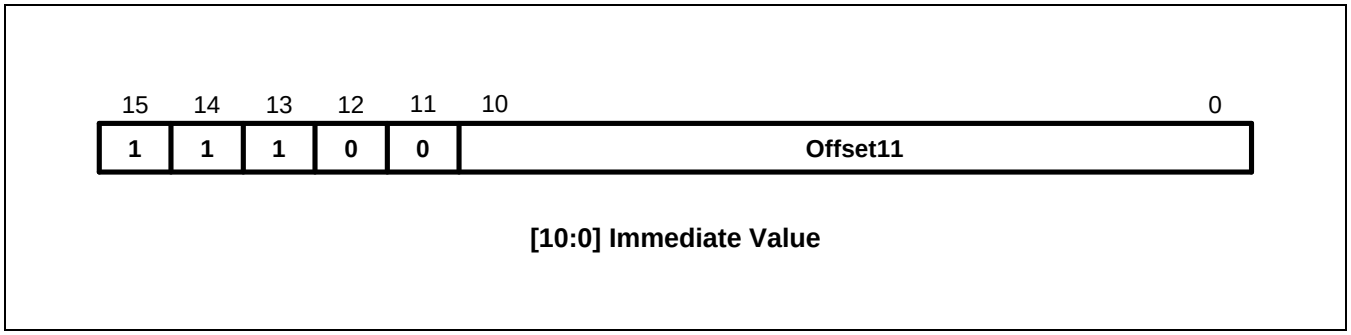


Figure 3-47. Format 18

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 (ie 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
        •
        •               ;  halfwords to offset.
jimmy   •               ;  Must be halfword aligned.
```

FORMAT 19: LONG BRANCH WITH LINK

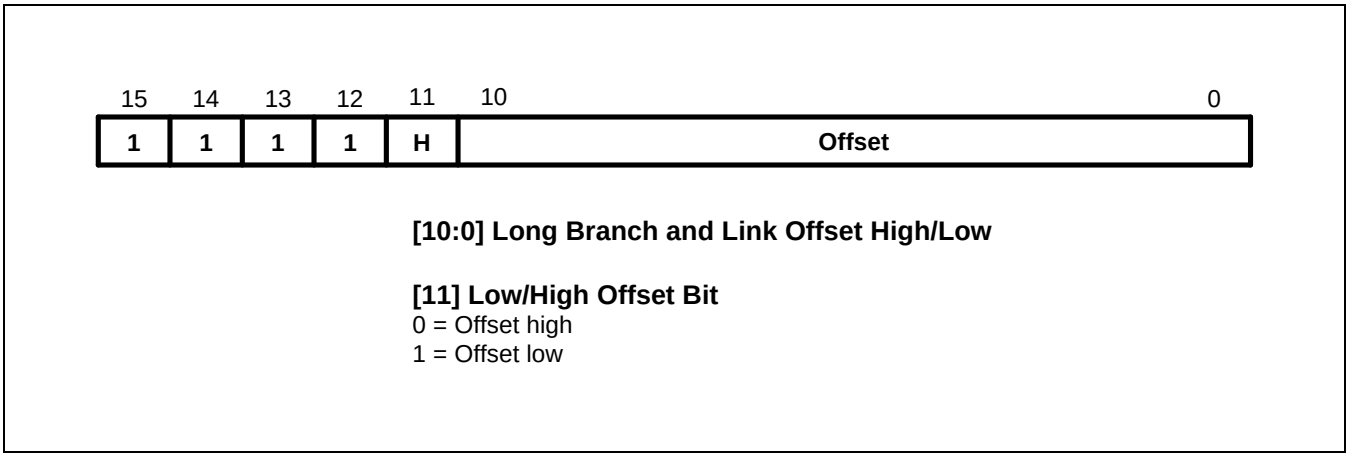


Figure 3-48. Format 19

OPERATION

This format specifies a long branch with link.

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.

Instruction 1 (H = 0)

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.

Instruction 2 (H =1)

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.

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

INSTRUCTION CYCLE TIMES

This instruction format does not have an equivalent ARM instruction.

Table 3-26. The BL Instruction

L	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

```
        BL faraway      ; Unconditionally Branch to 'faraway'
next    •               ; and place following instruction
        •               ; address, ie "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.
faraway •               ; 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 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)                      ;   (2..5)
        LSL     Ra, Ra, #n          ;   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 (0×FFFFFFFF) 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 (0×FFFFFFFF) 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 (ie 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
```

4 SYSTEM MANAGER

OVERVIEW

The S3C3410X System Manager has the following functionality:

- Arbitrate the bus usage requests from several master blocks, based on a fixed priority.
- Generate the necessary memory control signals for external memory access. For example, if a master block such as DMA or the CPU generates an address which corresponds to a DRAM bank, the DRAM controller inside System Manager should generate the necessary DRAM control signals (nRAS, nCAS, and so on).
- Support only the big-endian mode. The access to the internal register or the external memory should be done based on the big-endian mode.

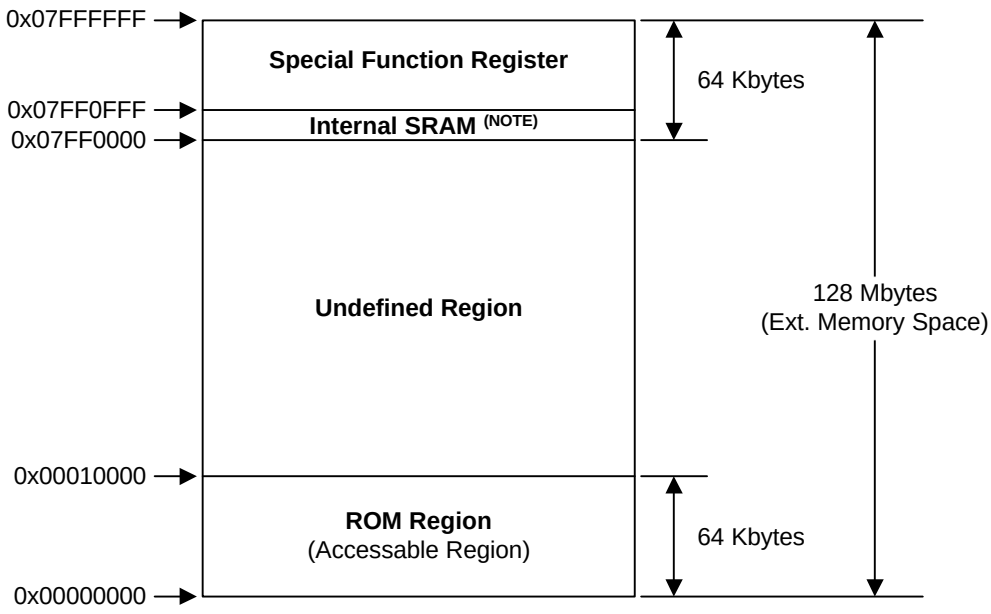
SYSTEM MANAGER REGISTER

The S3C3410X microcontroller has the SFRs, Special Function Register Set, to keep the system control information of system manager, cache, DMA, UART, and so on. The SFRs have the SMRs, System Manager Register Set, to configure the external memory map as well as the access-related option for SDRAM, DRAM, SRAM, ROM and extra-I/O control.

By utilizing the SMR, user can specify the memory type, external bus width, access cycles, necessary control signal timings(nRAS, nCAS, and so on), location of memory bank, and each memory bank size. The SMR can provide(or accept) the information of control signals, address, and data which are required by external devices during normal system operation. There are eleven registers to control memory bank (ROM, SRAM, DRAM/SDRAM), extra-device control and DRAM refresh.

The S3C3410X can provide up to 128M bytes of address space and each bank can provide up to 16M bytes memory space because each bank can have 24 address pins and 8-bit/16-bit data width.

The S3C3410X can also support two external I/O banks. These I/O banks are mapped into the SFR region. The two external I/O bank can give the smart interface between S3C3410X and external I/O device, which will improve the cost, PCB size, and reliability of system.



NOTE: If you use not cache but an internal SRAM as an internal memory, then the SRAM area are from 0x07FF0000 to 0x07FF0FFF. Refer to 5-4 page.

Figure 4-1. S3C3410X Memory Map (Default Map after Reset)

The S3C3410X can support 128M-byte memory space, which means that the S3C3410X should have an internal 27-bit system address bus. User can allocate the start address of bank by 64K-byte step from 0000000h to 7FFFFFFh. In other word, each bank can be located anywhere in the 128M-byte address space. The SFRs(Special Function Register) Set should occupy the 64K-byte region and the start address of normal memory bank should not be allocated in the region of SFR area. The region of SFR is a kind of memory mapped one and it can not allow the sharing the region with other banks.

ADDRESS BUS GENERATION

The address bus of the S3C3410X is quite different from the general MCU's. Although the general MCU does not use the A0 pin for 16-bit data bus width, the S3C3410X always uses the A0 pin regardless of data bus width. In other word, A0 should be connected to the lowest address bit of memory regardless of 16-bit bus width or 8-bit bus width. The bus width of bank 0(Boot ROM bank) can be configured by external pin(TEST[1:0]) and the bus width of other bank should be configured by writing the option information in SMRs. The memory controller in System Manager can generate A0 suitable for 8-bit bus width or 16-bit bus width, automatically. When an 8-bit data bus is selected, the resolution of address bus will be a byte and when a 16-bit is selected, the resolution of address bus will be a half-word.

Data Bus Width	External Address Pins : A[23:0]	Accessible Memory Size
8-bit	SA[23:0] (Internal)	16 M bytes
16-bit	SA[24:1] (Internal)	16 M half-word (32 M bytes)

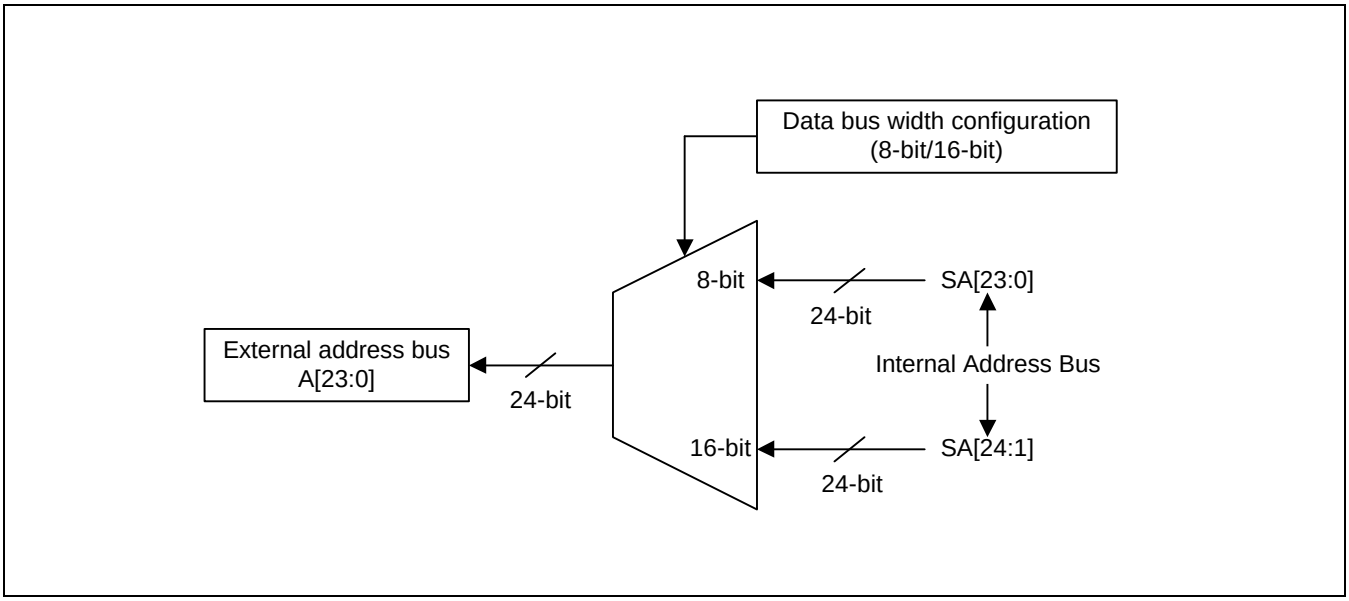


Figure 4-2. External Address Bus Generation (A[23:0])

nWE(NOT WRITE ENABLE)/nWBE[1:0](NOT WRITE BYTE ENABLE)

The nWE is the signal to indicate that the current bus cycle is for writing the data into the memory. But, if user want to write the byte data through 16-bit bus into the memory, there should be byte selection option. For example, user should have CAS0 and CAS1 signal in case of EDO DRAM. Similarly with EDO DRAM case, there should be nWBE[1:0] to select the byte. The x16 SRAM has nWE for indication of write cycle and LB(Lower Byte Selection)/UB(Upper Byte Selection) for selecting the byte. In this case, nWE from S3C3410X should be connected to WE of x16 SRAM, and nBE[0] and nBE[1] should be connected to LB and UB for the byte selection. Differently from x16 SRAM, in case of x16 SRAM with two x8 SRAM, nWBE[0] and nWBE[1] should be connected to the WE of SRAM, respectively.
In case of SDRAM attachment, nWE should be connected to WE of SDRAM and nWBE[1]/nWBE[0] should be connected to DQM[1]/DQM[0].

If user want x8 bus width for external memory access, please have following connection. In case of x8 SRAM, nWBE[0], not nWE, should be connected to the WE of SRAM. In case of x8 SDRAM, nWE and nWBE[0] should be connected to the WE and DQM of SDRAM.

There is certain case that no more byte access is needed. For example, x16 Flash Memory does not need byte access through 16-bit bus when user need the programming the data in the flash memory. In this case, please use nWBE[0] instead of nWE to indicate that the current bus cycle is a write cycle

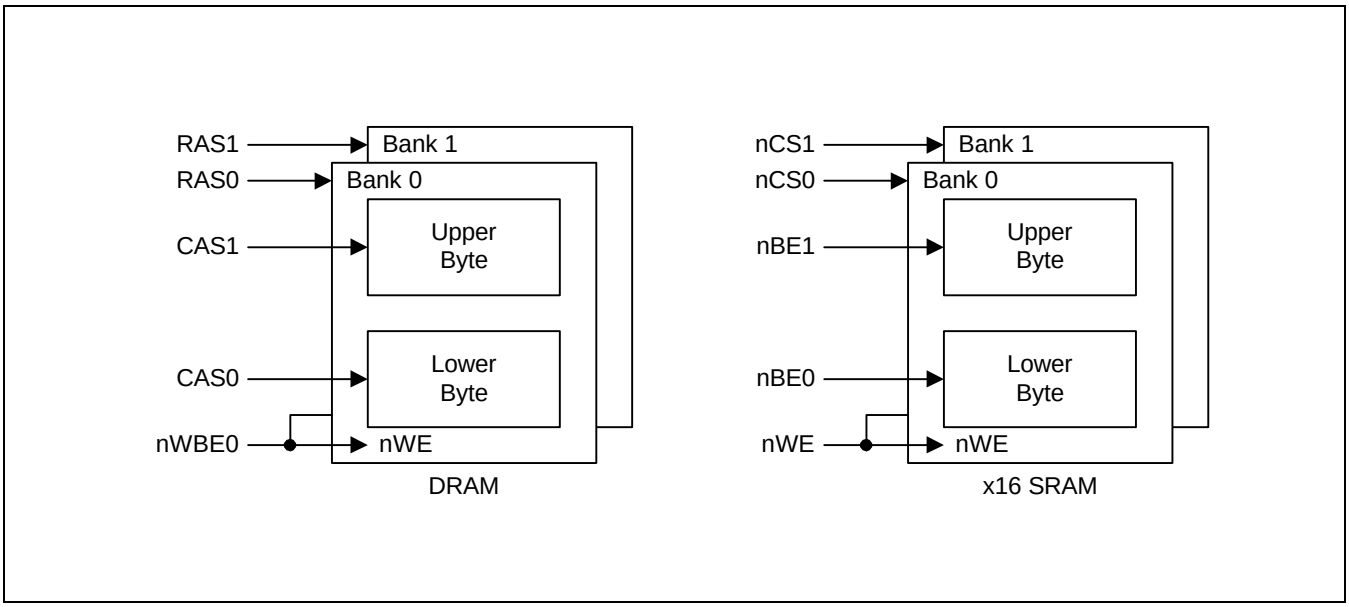


Figure 4-3. DRAM(x16) and SRAM(x16) Bank Configuration (For x16 Data Bus)

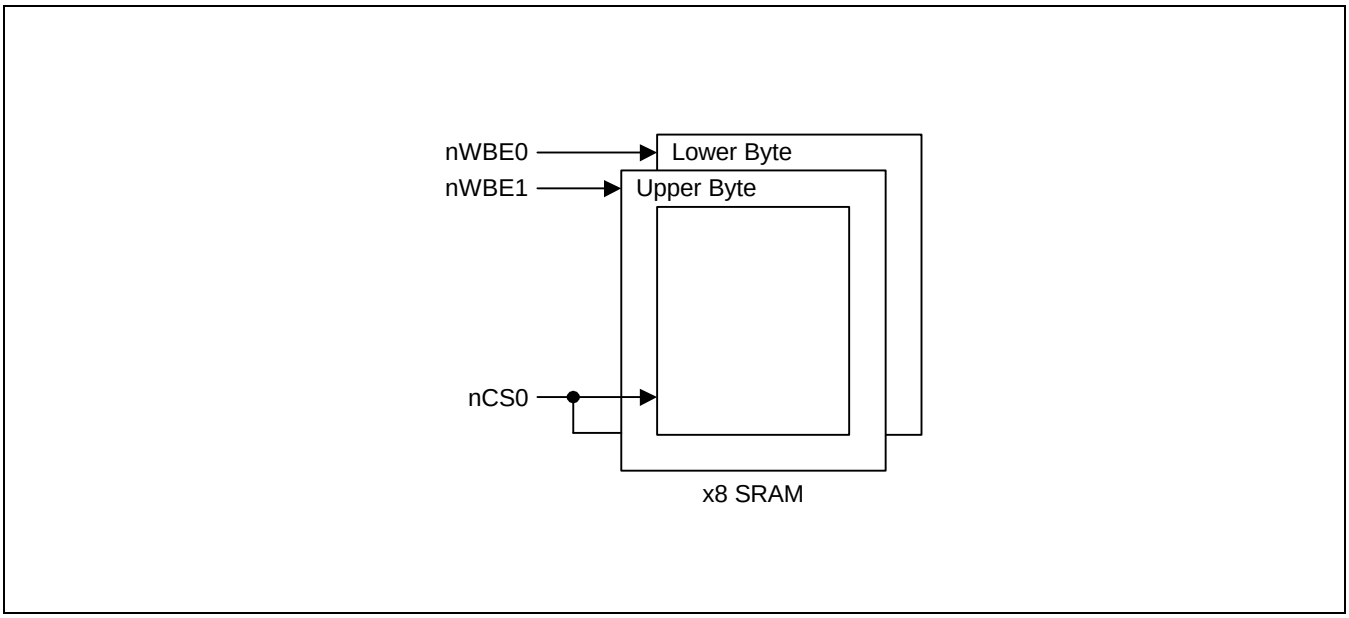


Figure 4-4. Two SRAM(x8) Configuration (For x16 Data Bus)

SYSTEM MANAGER & MEMORY CONTROLLER SPECIAL FUNCTION REGISTERS

SYSTEM REGISTER ADDRESS CONFIGURATION REGISTER (SYSCFG)

The SMRs (System Manager Registers) have the SYSCFG (System Register Address Configuration Register), which determines the start address(base point) of SFR(Special Function Register) files. The SYSCFG contains the start address of SFR. If the reset value of SYSCFG is fff1h, the SYSCFG is mapped to the address of 07FF0000h. To determine the start address, pick up the SYSCFG[14:4] and take 16-bit shift left. In this case, SYSCFG[14:4] is 7FFh and (7FFh << 16) is 07FF0000h, which is the start address of SFR.

Register	Offset Address	R/W	Description	Reset Value
SYSCFG	0x1000	R/W	Special function register to determine the start address	0xfff1

SYSCFG	Bit	Description	Initial State
ST	[0]	**Stall Enable:** When set to 1, Stall operation is enabled. The role of stall option is to insert one cycle wait for the non-sequential access. Originally, this feature was adopted to take care of the internal timing issue. So, we are recommending ST=0 to get the higher performance. 0 = Disable; It is recommended for faster operation 1 = Enable; Insert an internal wait inside the core logic when non-sequential memory accesses occur.	1
CE	[1]	**Cache Enable:** When set to 1, internal Cache will be enabled. When user want to define the internal SRAM, not cache, the cache should be disabled. If the performance is not critical, user can have cache disable option to reduce the current consumption. 0 = Cache disable 1 = Cache enable	0
WE	[2]	**Write Buffer Enable:** When set to 1, the write buffer operation is enabled. To get the higher performance, user should enable the write buffer. The disabling write buffer is for test purpose. 0 = Write buffer operation disable 1 = Write buffer operation enable	0
Reserved	[3]	Reserved	0
SFRSA	[14:4]	**SYSCFG Address (SFRs Start Address):** To determine the start address of SFR, this SFRSA field should be 16-bit left shifted. In other word, the start address of SFR is (SFRSA << 16).	7ff
CM	[16:15]	**Cache Mode:** Internal 4KB memory can be configured as 4KB cache, 2KB Cache/2KB SRAM, or 4KB SRAM. 00 = Half cache enable (2KB cache, 2KB internal SRAM) 01 = Full cache enable (4KB cache) 10 = Disable cache(4KB internal SRAM) 11 = Not used	01

SYSCFG	Bit	Description	Initial State
AME	[17]	**Address Mux Enable:** This bit determines whether or not to use the Multiplexed Address Mode. The Multiplexed Address Mode can generate the address for A[23:16] by using A[15:8] pins. In Normal Mode, the S3C3410X can support the dedicated pins for A[23:16]. In case of Multiplexed Address Mode, A[23:16] pins can be used as I/O ports. Because A[15:8] pins output address data for A[23:16] and A[15:8] by using latch device, as shown in Figure 4-5 and Figure 4-14. This is option for the pin usage because there are many multiplexed pins in S3C3410X. 0 = Normal Mode 1 = Multiplexed Address mode	0
MT0	[19:18]	**Memory Type 0:** This field determines memory type for bank6 00 = ROM/Flash/SRAM 01 = FP DRAM 10 = EDO DRAM 11 = Sync. DRAM	00
MT1	[21:20]	**Memory Type 1:** This field determines memory type for bank7 00 = ROM/Flash/SRAM 01 = FP DRAM 10 = EDO DRAM 11 = Sync. DRAM	00

BANK TIMING CONTROL REGISTER (BANKCONx : nCS0 – nCS5)

Register	Offset Address	R/W	Description	Reset Value
BANKCON0	0x2000	R/W	Bank 0 timing control register (for ROM/Flash)	0x00200070
BANKCON1	0x2004	R/W	Bank 1 timing control register (for ROM/Flash/SRAM)	0x0
BANKCON2	0x2008	R/W	Bank 2 timing control register (for ROM/Flash/SRAM)	0x0
BANKCON3	0x200c	R/W	Bank 3 timing control register (for ROM/Flash/SRAM)	0x0
BANKCON4	0x2010	R/W	Bank 4 timing control register (for ROM/Flash/SRAM)	0x0
BANKCON5	0x2014	R/W	Bank 5 timing control register (for ROM/Flash/SRAM)	0x0

BANKCONx	Bit	Description	Initial State
DBW	[0]	**Data Bus Width:** This bit determines the physical data bus width for bankx (bank1,2,3,4,and 5). The physical data bus width of bank0 depends on the configuration of TEST[1:0] pins. 0 = 8-bit 1 = 16-bit	0
PMC	[2:1]	These bits determines the page mode configuration for ROM access (Single mode, 4 data page mode, 8 data page mode, and 16 data page mode). 00 = 1 Data 01 = 4 Data 10 = 8 Data 11 = 16 Data	00
SM	[3]	In certain x16 SRAM, there are byte selection signals such as LB (Lowe Byte) and UB (Upper Byte). In this case, nWE from S3C3410X should be connected to WE of SRAM and nWBE[1:0] from S3C3410X should be connected to UB/LB of SRAM. 0 = Ordinary 1 = x16 type SRAM	0
Tacc	[6:4]	Determine the number of Access Cycle (Tacc). Please refer the timing diagram. 000 = Disable 001 = 2 Clock 010 = 3 Clock 011 = 4 Clock 100 = 5 Clock 101 = 6 Clock 110 = 7 Clock 111 = 10 Clock	111
Tacp	[8:7]	Determine the number of Page mode access cycle @ page mode (Tacp). Please refer the timing diagram. 00 = 5 Clock 01 = 2 Clock 10 = 3 Clock 11 = 4 Clock	00
Reserved	[9]	Reserved	0

BANKCONx	Bit	Description	Initial State
BAP	[20:10]	**Memory Bank Base Address Pointer:** This 11-bit value corresponds to the upper 11 bits from the total 27-bit system address bus. It indicates the start address of the corresponding memory bank(Bankx), based on 64K-byte units. The base address pointer value is calculated as follows: Base_Address_Pointer = Start_Address / 10000h, which is (BAP[20:10] << 16). If BAP is same with EAP, the corresponding memory bankx will be disabled.	00000000000b
EAP	[31:21]	**Memory Bank End Address Pointer:** This 11-bit value corresponds to the upper 11 bits from the total 27-bit system address bus. To determine the EAP, please refer the below equation : End_Address_Pointer = (End_Address + 1) / 10000h. (End_Address of corresponding memory bank+ 1) is equal to (EAP << 16). In this case, End_Address means the end address of corresponding memory bank by byte address unit, not half-word or word address unit.	00000000000b

BANK TIMING CONTROL REGISTER (BANKCONx: nRAS0 – nRAS1)

Register	Offset Address	R/W	Description	Reset Value
BANKCON6	0x2018	R/W	Bank 6 control register (for FP/EDO/SDRAM/ROM/Flash/SRAM)	0x0
BANKCON7	0x201c	R/W	Bank 7 control register (for FP/EDO/SDRAM/ROM/Flash/SRAM)	0x0

NOTE: BANKCON6 and 7 register can have dual configuration, depending on the MT field in SYSCFG register. In other word, BANKCON6 and 7 have the same configuration with BANKCON1,2,3,4 and 5 when MT=00, or BANKCON6 and 7 have the configuration on DRAM(FP, EDO)/SDRAM when MT=01, 10, and 11.

BANKCONx	Bit	Description	Initial State
Memory Type = ROM or SRAM [MT=00 in SYSCFG]			
DBW	[0]	**Data Bus Width:** This bit determines the physical data bus width for bankx (bank6 and 7) 0 = 8-bit 1 = 16-bit	0
PMC	[2:1]	These bits determines the page mode configuration for ROM access (Single mode, 4 data page mode, 8 data page mode, and 16 data page mode). 00 = 1 Data 01 = 4 Data 10 = 8 Data 11 = 16 Data	00
SM	[3]	In certain x16 SRAM, there are byte selection signals such as LB (Lowe Byte) and UB (Upper Byte). In this case, nWE from S3C3410X should be connected to WE of SRAM and nWBE[1:0] from S3C3410X should be connected to UB/LB of SRAM. 0 = Ordinary 1 = x16 type SRAM	0
Tacc	[6:4]	Determine the number of Access Cycle (Tacc). Please refer the timing diagram. 000 = Disable 001 = 2 Clock 010 = 3 Clock 011 = 4 Clock 100 = 5 Clock 101 = 6 Clock 110 = 7 Clock 111 = 10 Clock	000
Tacp	[8:7]	Determine the number of Page mode access cycle @ page mode (Tacp). Please refer the timing diagram. 00 = 5 Clock 01 = 2 Clock 10 = 3 Clock 11 = 4 Clock	00
Reserved	[9]	Reserved	0

BANKCONx	Bit	Description	Initial State
BAP	[20:10]	**Memory Bank Base Address Pointer:** This 11-bit value corresponds to the upper 11 bits from the total 27-bit system address bus. It indicates the start address of the corresponding memory bank(Bankx), based on 64K-byte units. The base address pointer value is calculated as follows: Base_Address_Pointer = Start_Address / 10000h, which is (BAP[20:10] << 16). If BAP is same with EAP, the corresponding memory bankx will be disabled	00000000000b
EAP	[31:21]	**Memory Bank End Address Pointer:** This 11-bit value corresponds to the upper 11 bits from the total 27-bit system address bus. To determine the EAP, please refer the below equation : End_Address_Pointer = (End_Address + 1) / 10000h. (End_Address of corresponding memory bank+ 1) is equal to (EAP << 16). In this case, End_Address means the end address of corresponding memory bank by byte address unit, not half-word or word address unit.	00000000000b

Memory Type = FP DRAM [MT=01 in SYSCFG] or EDO DRAM [MT=10 in SYSCFG]			
DBW	[0]	**Data Bus Width:** This bit determines the physical data bus width for bankx (bank6 and 7) 0 = 8-bit 1 = 16-bit	0
CAN	[2:1]	Column Address Number for DRAM (FP and EDO DRAM). 00 = 8-bit 01 = 9-bit 10 = 10-bit 11 = 11-bit	00
Tcp	[3]	CAS Pre-charge Time. Please refer the timing diagram. 0 = 1 Clock 1 = 2 Clock	0
Tcas	[6:4]	CAS Pulse Width. Please refer the timing diagram. 000 = 1 Clock 001 = 2 Clock 010 = 3 Clock 011 = 4 Clock 100 = 5 Clock 101 = Not used 110 = Not used 111 = Disable	000
Trc	[7]	RAS to CAS Delay Time. Please refer the timing diagram. 0 = 1 Clock 1 = 2 Clock	0
Trp	[9:8]	RAS Pre-charge Time. Please refer the timing diagram. 00 = 1 Clock 01 = 2 Clock 10 = 3 Clock 11 = 4 Clock	00
BAP	[20:10]	**Memory Bank Base Address Pointer:** This 11-bit value corresponds to the upper 11 bits from the total 27-bit system address bus. It indicates the start address of the corresponding memory bank(Bankx), based on 64K-byte units. The base address pointer value is calculated as follows: Base_Address_Pointer = Start_Address / 10000h, which is (BAP[20:10] << 16). If BAP is same with EAP, the corresponding memory bankx will be disabled	00000000000b
EAP	[31:21]	**Memory Bank End Address Pointer:** This 11-bit value corresponds to the upper 11 bits from the total 27-bit system address bus. To determine the EAP, please refer the below equation : End_Address_Pointer = (End_Address + 1) / 10000h. (End_Address of corresponding memory bank+ 1) is equal to (EAP << 16). In this case, End_Address means the end address of corresponding memory bank by byte address unit, not half-word or word address unit.	00000000000b

Memory Type = Sync. DRAM [MT=11 in SYSCFG]			
DBW	[0]	**Data Bus Width :** This bit determines the physical data bus width for bankx(bank6 and 7) 0 = 8-bit 1 = 16-bit	0
CAN	[2:1]	Column Address Number of SDRAM. 00 = 8-bit 01 = 9-bit 10 = 10-bit 11 = 11-bit	00
Reserved	[6:3]	Reserved	0000
Trc	[7]	RAS to CAS Delay Time. Please refer the timing diagram. 0 = 1 Clock 1 = 2 Clock	0
Trp	[9:8]	RAS Pre-charge Time. Please refer the timing diagram. 00 = 1 Clock 01 = 2 Clock 10 = 3 Clock 11 = 4 Clock	00
BAP	[20:10]	**Memory Bank Base Address Pointer:** This 11-bit value corresponds to the upper 11 bits from the total 27-bit system address bus. It indicates the start address of the corresponding memory bank(Bankx), based on 64K-byte units. The base address pointer value is calculated as follows: Base_Address_Pointer = Start_Address / 10000h, which is (BAP[20:10] << 16). If BAP is same with EAP, the corresponding memory bankx will be disabled	00000000000b
EAP	[31:21]	**Memory Bank End Address Pointer:** This 11-bit value corresponds to the upper 11 bits from the total 27-bit system address bus. To determine the EAP, please refer the below equation: End_Address_Pointer = (End_Address + 1) / 10000h. (End_Address of corresponding memory bank+ 1) is equal to (EAP << 16). In this case, End_Address means the end address of corresponding memory bank by byte address unit, not half-word or word address unit.	00000000000b

EXTERNAL DEVICE CONTROL REGISTERS (EXTCONn)

The S3C3410X can support the connection with two external I/O devices without any additional logic. It is very cost effective, because the additional address decoding logic is not necessary. The smart connection between S3C3410X and external I/O device can improve the cost, PCB size, and reliability of system. Differently from the normal memory banks (8 memory banks in S3C3410X), S3C3410X defines the external I/O banks in SFR (Special Function Register), EXTPORT, EXTDAT0, and EXTDAT1. EXTDAT0 and EXTDAT1 have 64 bytes addressing region, respectively. To read the data from the external I/O device, you should execute the load instruction by issuing the address (SFR start address + I/O bank offset), EXTPORT, EXTDAT0 or EXTDAT1. Then, the data will be latched in the data register. To write the data into the external I/O device, you should execute the store instruction by issuing the address (SFR start address + I/O bank offset), EXTPORT, EXTDAT0 or EXTDAT1. Then, the data will be written into the selected address, and the data will be written into the external I/O device. These operation is automatically executed by the memory controller.

Register	Offset Address	R/W	Description	Reset Value
EXTCON0	0x2030	R/W	Extra device control register 0 (for external output port)	0x0
EXTCON1	0x2034	R/W	Extra device control register 1 (for external chip selection)	0x0

EXTCONx	Bit	Description	Initial State
DW	[1:0]	**Data Bus Width:** This bit determines the physical data bus width for EXTBANKx (External bank0 and 1) 00 = Disable Bank 01 = 8-bit 10 = 16-bit 11 = Not used	00
Tcos	[4:2]	Set-up Time of nECS before nOE. Please refer the timing diagram. 000 = 0 Clock 001 = 1 Clock 010 = 2 Clock 011 = 3 Clock 100 = 4 Clock 101 = 5 Clock 110 = 6 Clock 111 = 7 Clock	000
Tcoh	[10:8]	Hold Time of nECS after nOE. Please refer the timing diagram. 000 = 0 Clock 001 = 0 Clock 010 = 2 Clock 011 = 3 Clock 100 = 4 Clock 101 = 5 Clock 110 = 6 Clock 111 = 7 Clock	000
Tacc	[13:11]	Access Times (nOE low time) 000 = 0 Clock 001 = 2 Clock 010 = 3 Clock 011 = 4 Clock 100 = 5 Clock 101 = 6 Clock 110 = 7 Clock 111 = 8 Clock	000

EXTERNAL OUTPUT PORT REGISTER (EXTPORT)

You can use an external output latch device as an output port without any additional address logic for decoding logic, because S3C3410X have the nWREXP pin. When you write any data to EXTPORT register, nWREXP pin outputs a write strobe signal to interface with an external output latch device for latching the output data. That is, the access signal to the external output latch device is controlled by writing any data to EXTPORT register. For example, if you write 0xff to EXTPORT, memory controller outputs a write strobe signal in nWREXP pin and 0xff in data bus. At this time, the access time is controlled by the extra device control register, EXTCON0 only. (Refer to Figure 4-21)

When MDS mode is selected, this nWREXP pin is used as a write strobe signal for an external output latch, which is emulated for port 7 output.

Register	Offset Address	R/W	Description	Reset Value
EXTPORT	0x203e	R/W	External Port Data register	0x0

EXTERNAL CHIP SELECTION DATA REGISTER (EXTDATX)

You can directly access to the external device as external memory without any additional address decoding logic because S3C3410X have the external device control logic, nECS0 and nECS1 pins. These pins output the external device selection signals. External device is accessed by the memory controller when the data is read/write from/to EXTDAT0 or EXTDAT1. That is, if you read/write the data from/to EXTDAT0 or EXTDAT1, the memory controller automatically outputs nECS0 or nECS1, the selected address and data. For example, if you write 0xff to 0x206c in EXTDAT0, memory controller outputs nECS0 signal, 0x206c in address bus and 0xff in data bus. At this time, the access time of EXTDAT0 and EXTDAT1 are controlled by the extra device control register1, EXTCON1 only. (Refer to Figure 4-21)

When P2.7 or P3.7 is used as a normal input/output mode, nECS0 or nECS1 can not support extra chip selection.

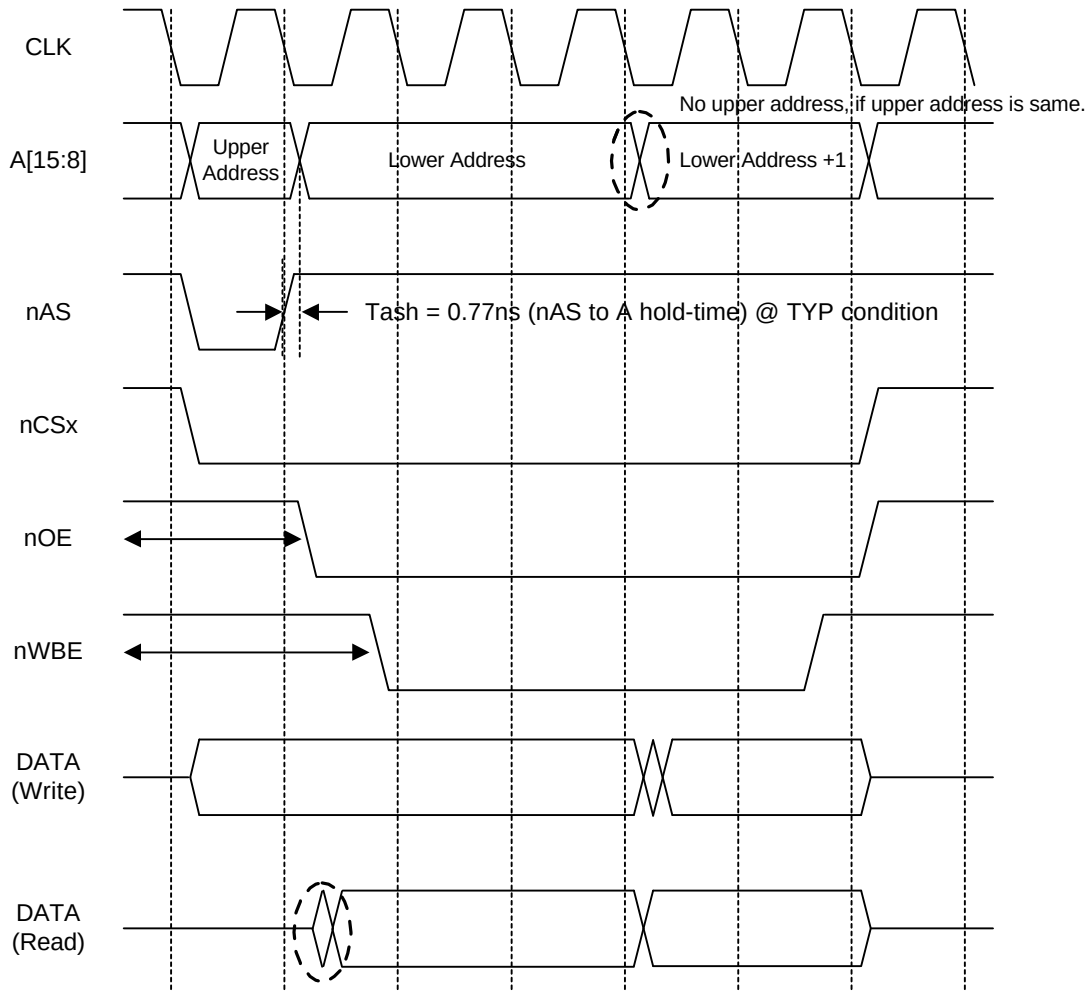
Register	Offset Address	R/W	Description	Reset Value
EXTDAT0	0x202c	R/W	Extra chip selection data register 0	–
	0x206c			
	0x20ac			
	0x20ec			
	0x212c			
	:			
	:			
	0x2fec			
EXTDAT1	0x202e	R/W	Extra chip selection data register 1	–
	0x206e			
	0x20ae			
	0x20ee			
	0x212e			
	:			
	:			
	0x2fee			

DRAM/SDRAM SELF REFRESH CONTROL REGISTER (REFCON)

Register	Offset Address	R/W	Description	Reset Value
REFCON	0x2020	R/W	DRAM/SDRAM refresh control register	0x1

REFCON	Bit	Description	Initial State
VSMR	[0]	**Validity of Special Memory Register (SMR):** Whenever CPU access one of system manager registers(SMR), VSMR bit will be cleared automatically and all memory bank will be disabled. To re-activate the memory bank, VSMR bit should be set to 1 by using STMIA instruction(Data in the CPU registers can be stored into the memory or memory mapped register by single instruction). In other word, user should update the necessary configuration in SMR as well as setting VSMR bit in REFCON register, simultaneously. To do the simultaneous updating, user should use the STMIA instruction, which can transfer the CPU register data into SMR. The last data transfer from CPU register should be data transfer to REFCON register to set the VSMR bit. 0 = Not accessible to memory bank 1 = Accessible to memory bank	1
RC	[11:1]	**Refresh Interval (Refresh Count):** This RC field determine the DRAM refresh period by below equation. **Refresh Period = (2^{11} – refresh count + 1) / MCLK** Ex) If refresh period is 15.6us and MCLK is 33MHz, the refresh count should be as follows: **Refresh Count = 2^{11} + 1 – 33 x 15.6 = 1019 = 1111111011b**	00000000000b
REN	[12]	**Refresh Enable:** If Bank 6 and/or Bank 7 are configured to have DRAM bank by MT[1:0] field in SYSCFG register, this bit has following option. 0 = Disable DRAM refresh. 1 = Enable DRAM refresh. If Bank 6 and/or Bank 7 are configured to have SDRAM bank by MT[1:0] field in SYSCFG register, this bit has following option. 0 = Disable SDRAM auto-refresh. 1 = Enable SDRAM auto refresh.	0
Tch	[15:13]	CAS Hold Time. Please refer the timing diagram. 000 = 1 Clock 001 = 2 Clock 010 = 3 Clock 011 = 4 Clock 100 = 5 Clock Other = Not used	000
Tcsr	[16]	CAS Set-up Time. Please refer the timing diagram. 0 = 1 Clock 1 = 2 Clock	0

MEMORY ACCESS AND I/O TIMING DIAGRAM



When SRAM/ROM interface and enables address mux
A[15:8] outputs upper address during 1-cycle.
At this time, nAS is enabled during 1-cycle.

The nAS signal occurs only in case the current upper address is changed from the previous one.
(That is to say, A[23:16] is different from the previous A[23:16].)

When nAS occurs, nOE and nWBE[1:0] signals are delayed for 1-cycle, but, the nCSx is enabled. When upper address outputs, data(write) is enabled. But data(read) is delayed for 1-cycle because nOE is delayed.

Figure 4-5. S3C3410X Multiplexed Address Mode Timing Diagram

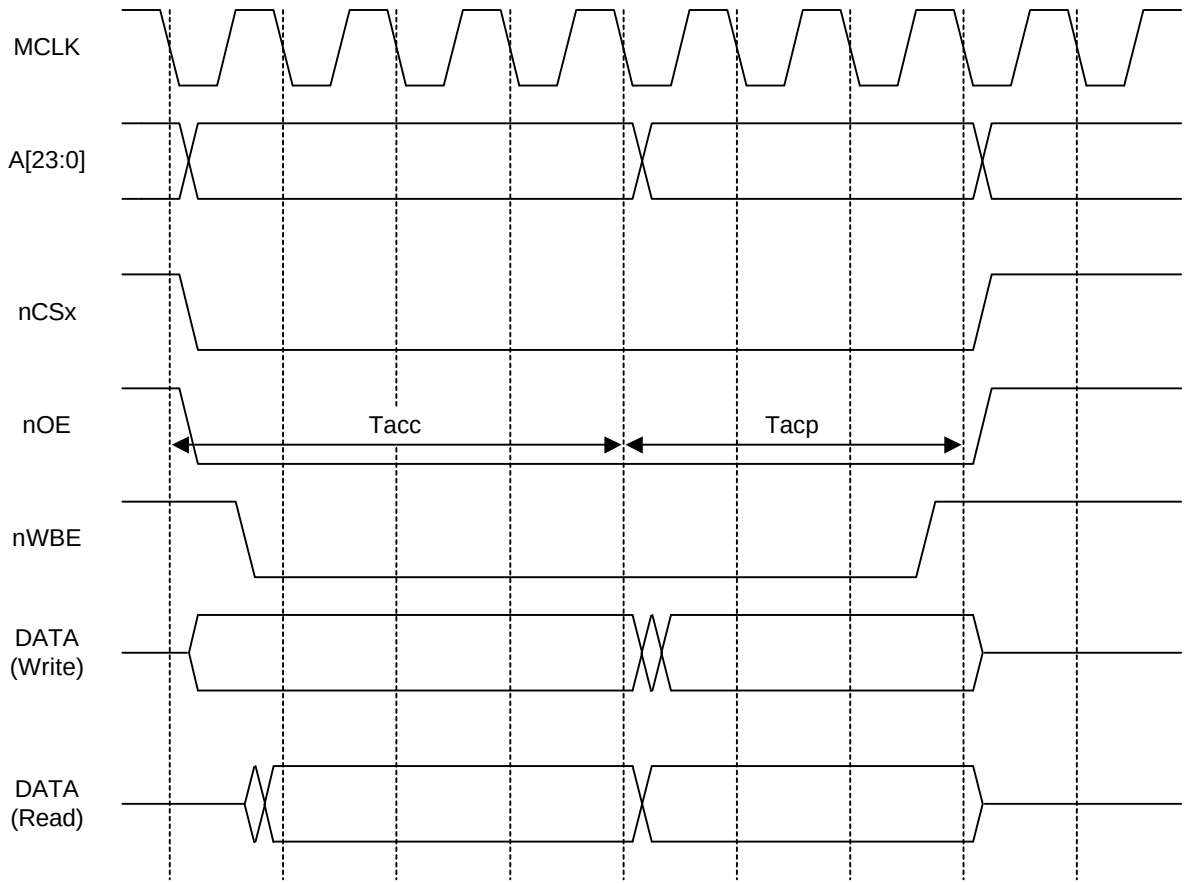


Figure 4-6. S3C3410X nCS Timing Diagram

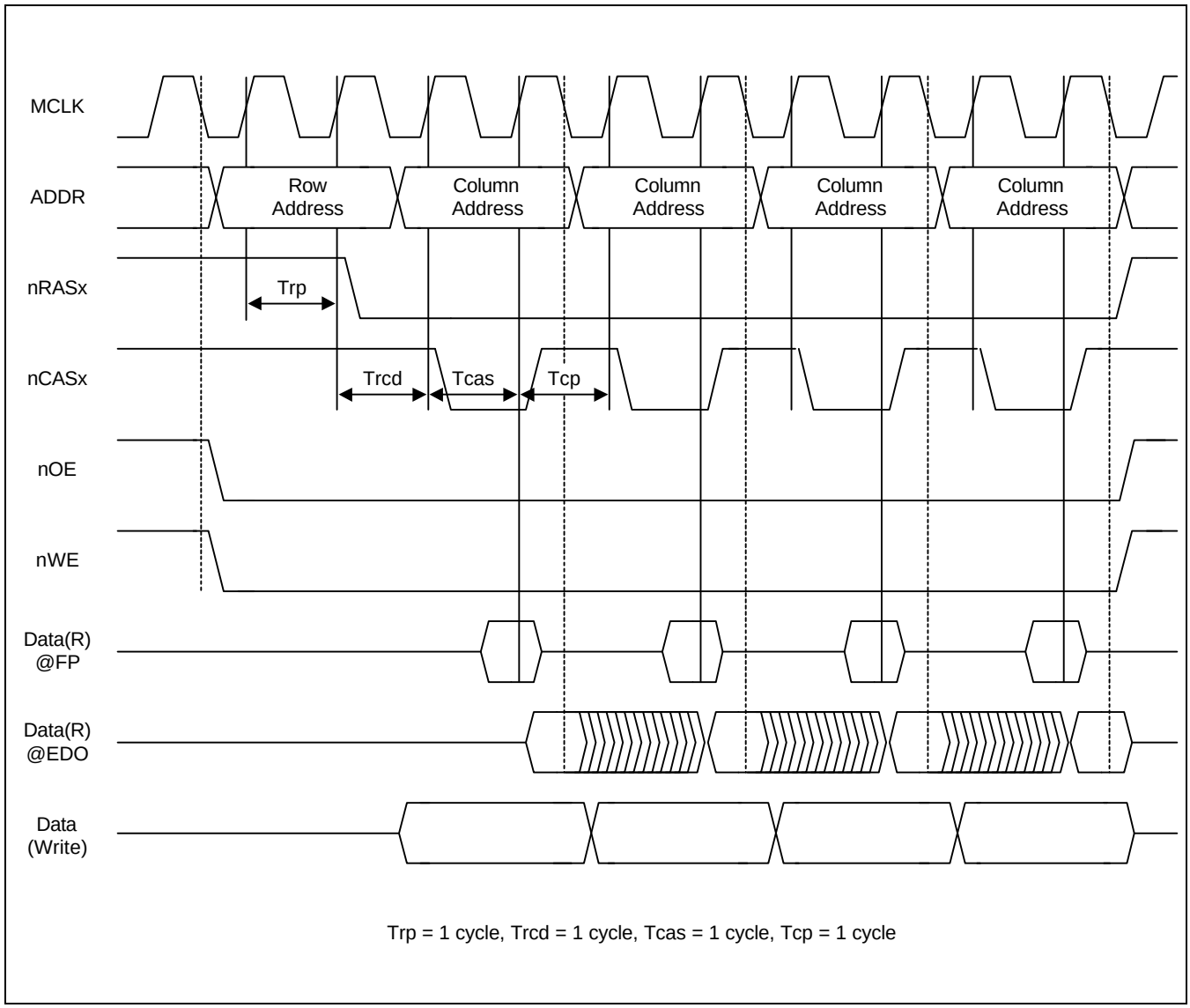


Figure 4-7. S3C3410X DRAM Timing Diagram

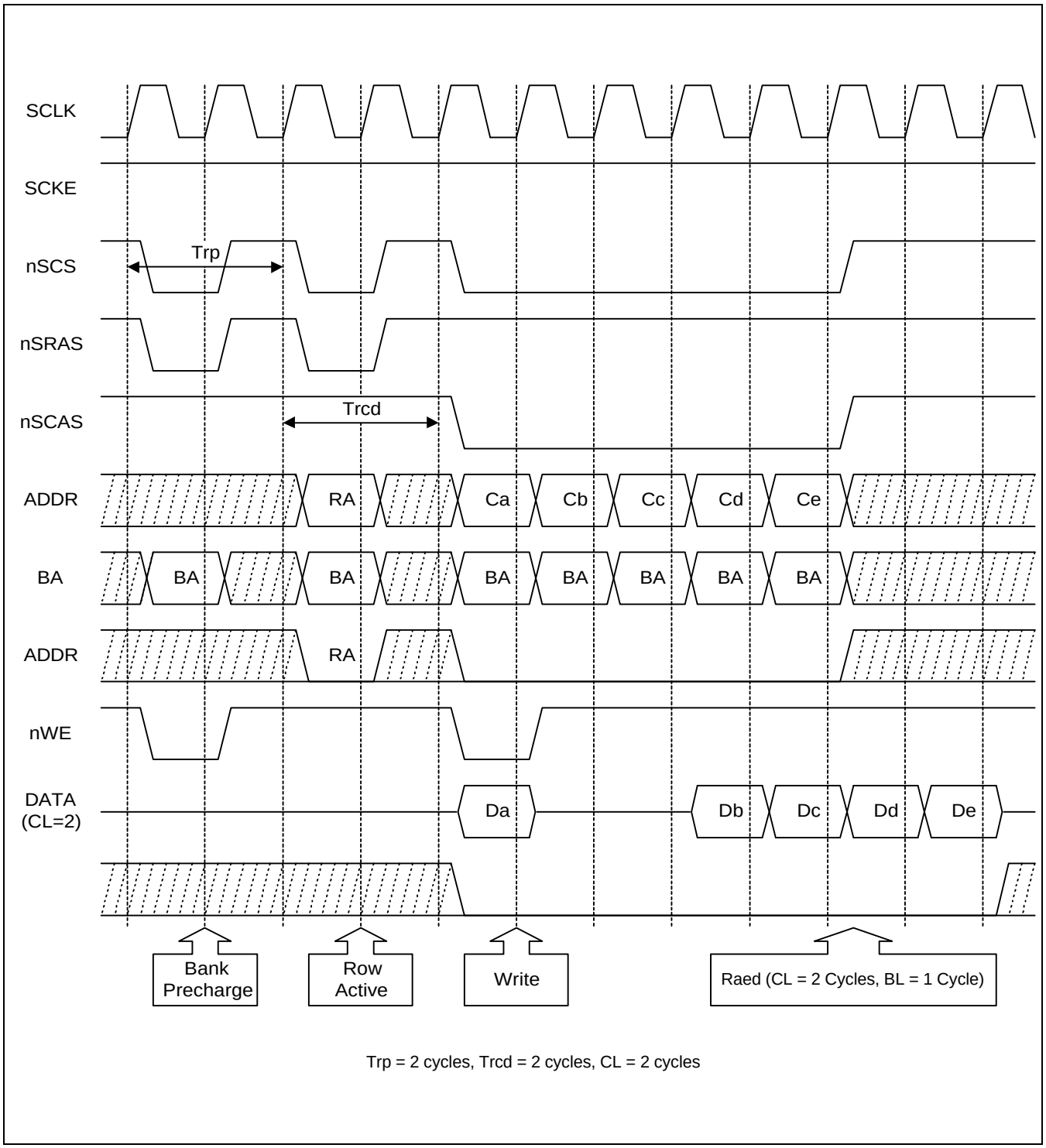


Figure 4-8. S3C3410X SDRAM Timing Diagram

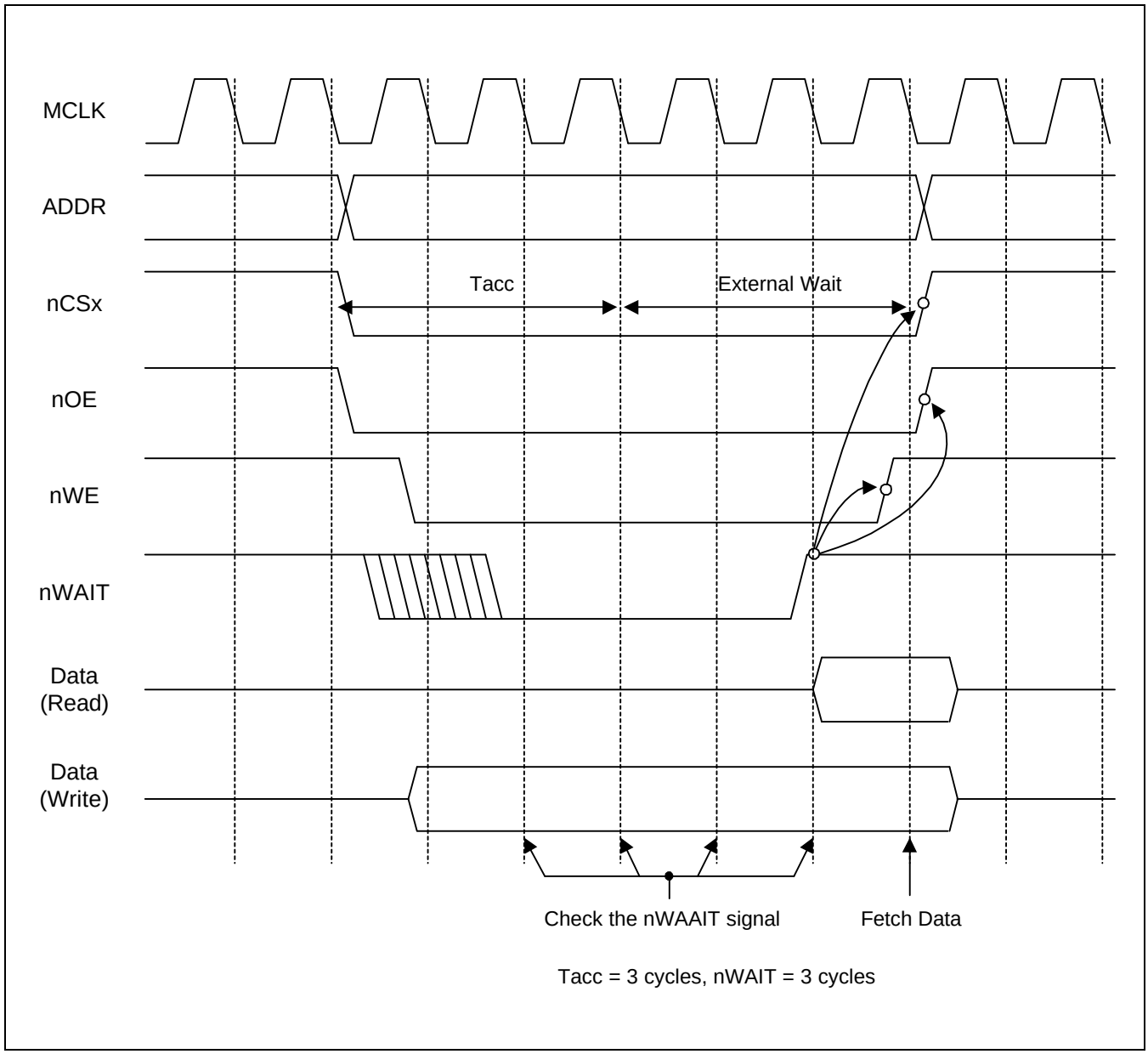


Figure 4-9. S3C3410X nCS Timing Diagram with nWAIT

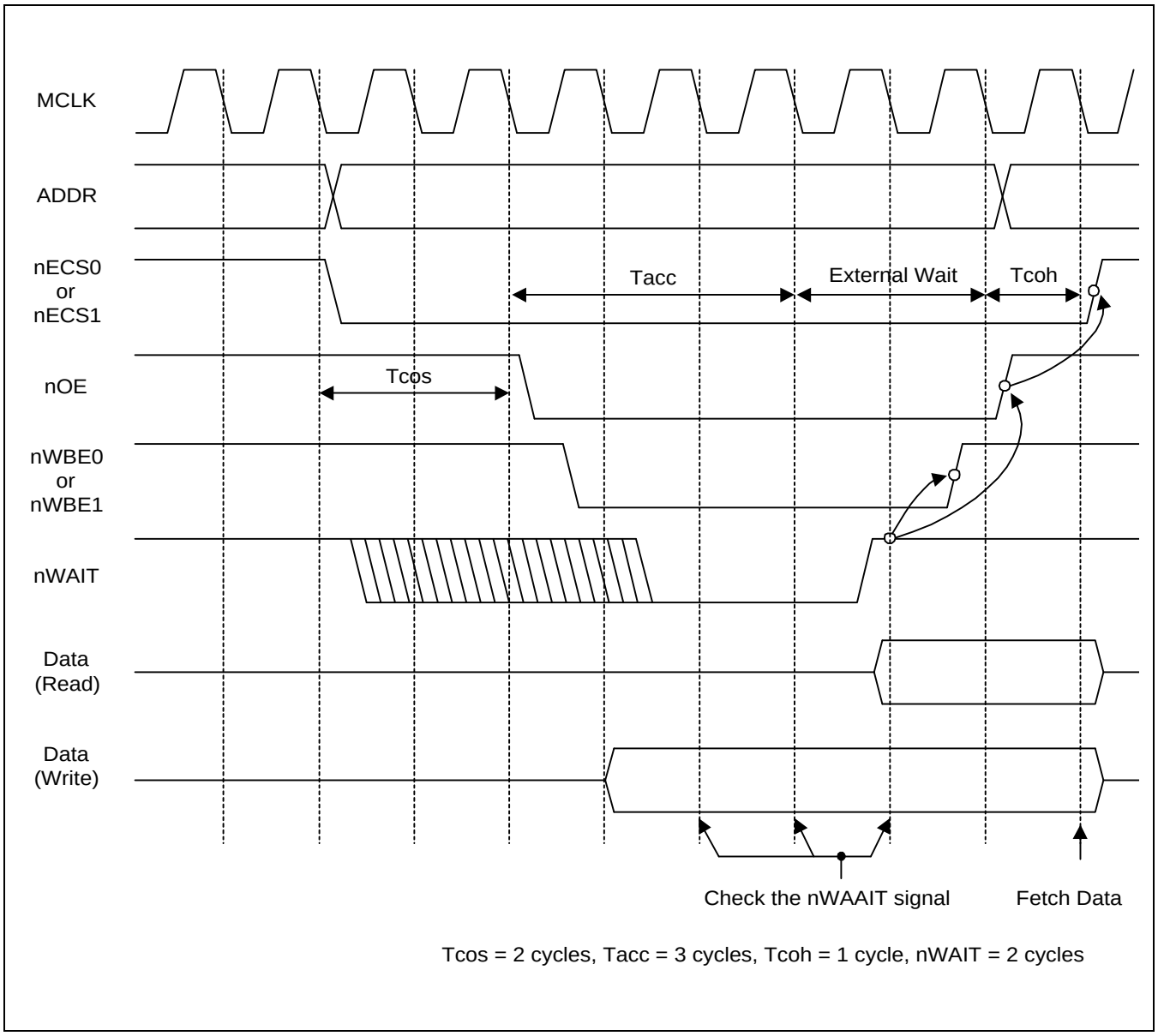


Figure 4-10. S3C3410X nECS Timing Diagram with nWAIT

MEMORY INTERFACE SIGNAL CONNECTION METHOD

Pin Name	Memory Type	Description
nOE	Any type	Memory read strobe signal
nWBE0	Single ×8 Flash ROM/EEPROM/DRAM/SRAM	Memory write strobe signal
	Two ×8 Flash ROM/EEPROM/DRAM/SRAM	Memory lower byte write strobe signal
	Single ×16 SRAM	Memory lower byte signal
nWBE1	Two ×8 Flash ROM/EEPROM/DRAM/SRAM	Memory upper byte write strobe signal
	Single ×16 SRAM	Memory upper byte signal
nWE	×16 SRAM/SDRAM	Memory write strobe signal

MEMORY CONNECTION EXAMPLE

Memory Configuration Type	MCU Pin	Memory Pin
×8/×16 ROM (MCU data bus: ×8/×16)	nOE	nOE
Single x8 Flash ROM/EEPROM or single x8 SRAM	nOE	nOE
(MCU data bus: ×8)	nWBE0	nWE
Two x8 Flash ROM/EEPROM or two ×8 SRAM	nOE	nOE
(MCU data bus: ×16)	nWBE0	nWE of lower byte
	nWBE1	nWE of upper byte
×16 SRAM	nOE	nOE
(MCU data bus: ×16)	nBE0	nLB
	nBE1	nUB
	nSWE	nWE
Single ×8 DRAM	VDD	nOE
(MCU data bus: ×8)	nRAS0	nRAS
	nCAS0	nCAS
	nWBE0	nWE
Two ×8 DRAM	VDD	nOE
(MCU data bus: ×16)	nRAS0	nRAS
	nCAS0	nCAS of lower byte
	nCAS1	nCAS of upper byte
	nWBE0	nWE
Single ×16 DRAM	VDD	nOE
(MCU data bus: ×16)	nRAS0	nRAS
	nCAS0	nLCAS
	nCAS1	nUCAS
	nWBE0	nWE

Memory Configuration Type	MCU Pin	Memory Pin
Two ×16 DRAM (MCU data bus: ×16)	VDD	nOE
	nRAS0	nRAS of lower bank
	nRAS1	nRAS of upper bank
	nCAS0	nLCAS
	nCAS1	nUCAS
	nWBE0	nWE
Single ×8 SDRAM (MCU data bus: ×8)	nOE	nOE
	nSCS	nSCS
	nSRAC	nSRAC
	nSCAS	nSCAS
	DQM0	DQM
	nWE	nWE
	SCKE	SCKE
	SCLK	SCLK
Two ×8 SDRAM (MCU data bus: ×16)	nOE	nOE
	nSCS	nSCS
	nSRAC	nSRAC
	nSCAS	nSCAS
	DQM0	DQM of lower byte
	DQM1	DQM of upper byte
	nWE	nWE
	SCKE	SCKE
	SCLK	SCLK
Single ×16 SDRAM (MCU data bus: ×16)	nOE	nOE
	nSCS	nSCS
	nSRAC	nSRAC
	nSCAS	nSCAS
	DQM0	LDQM
	DQM1	UDQM
	nWE	nWE
	SCKE	SCKE
	SCLK	SCLK

MEMORY CONFIGURATION EXAMPLES

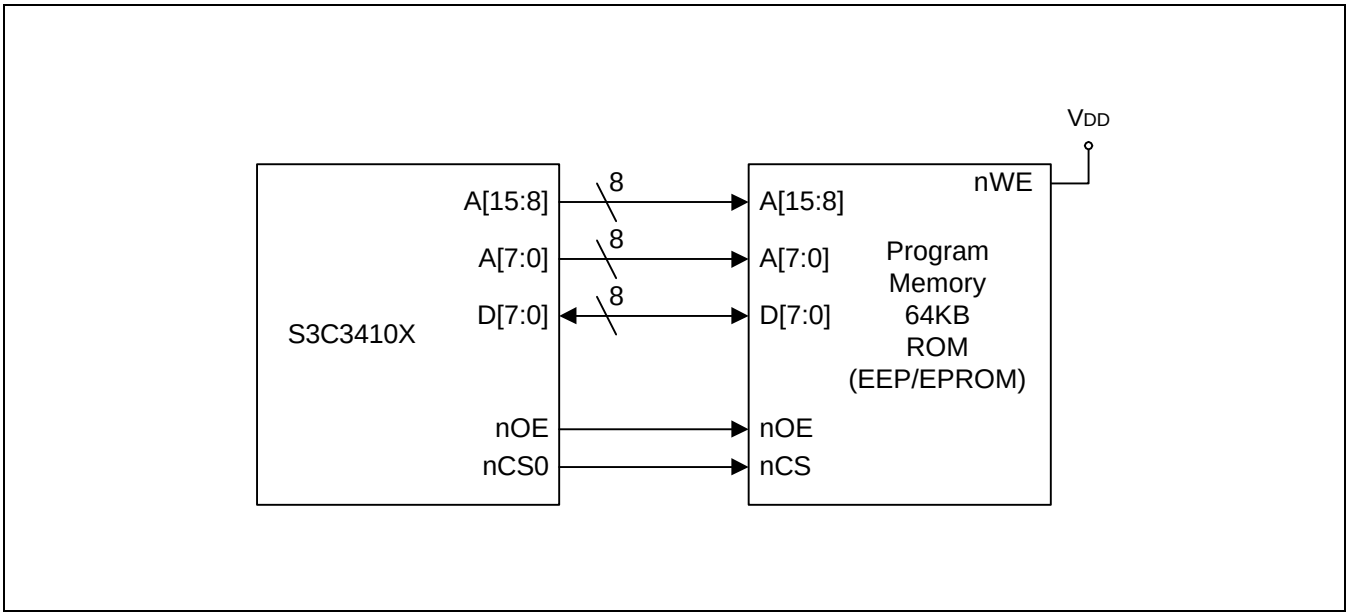


Figure 4-11. 64K × 8 ROM Memory Only

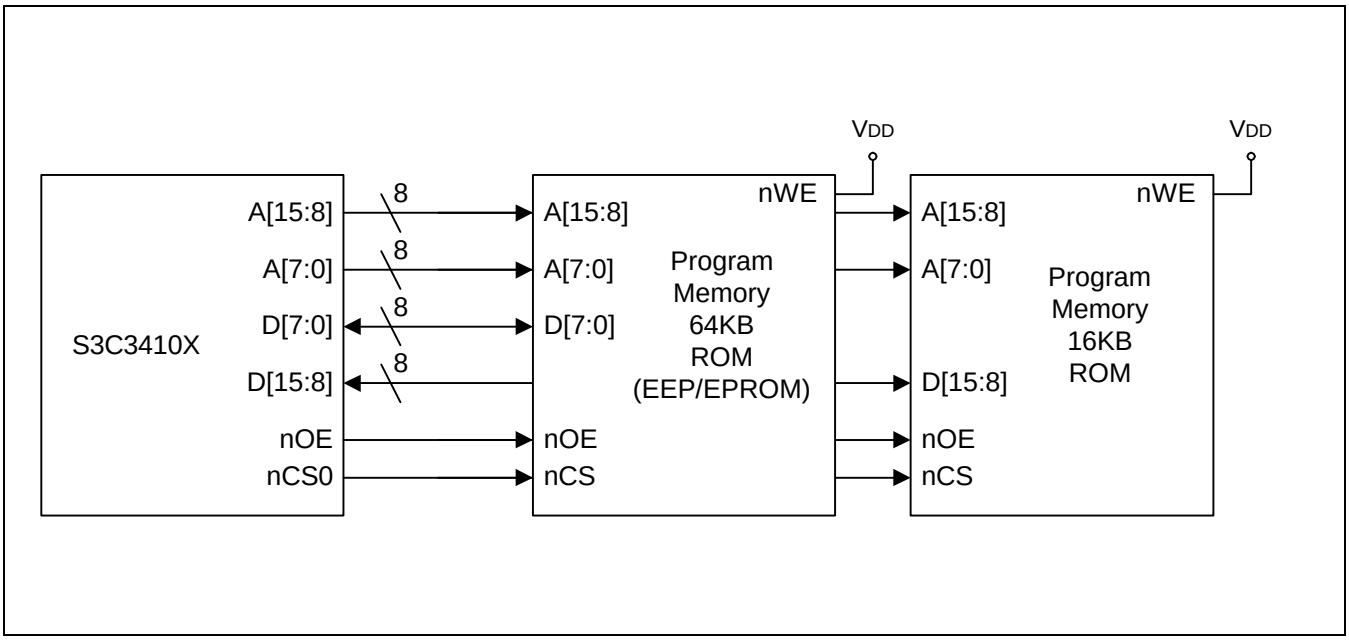


Figure 4-12. 64K × 16 ROM Memory Only

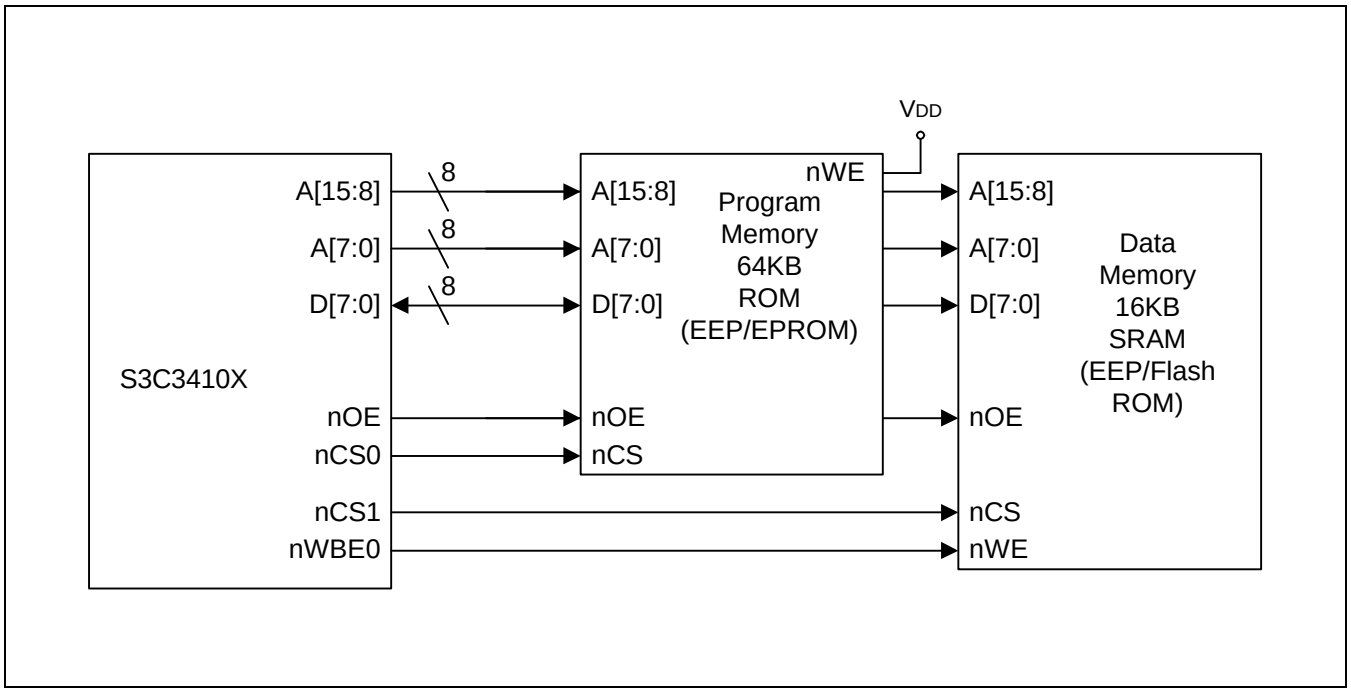


Figure 4-13. 64KB × 8 Program ROM & 64KB × 8 Data Memory

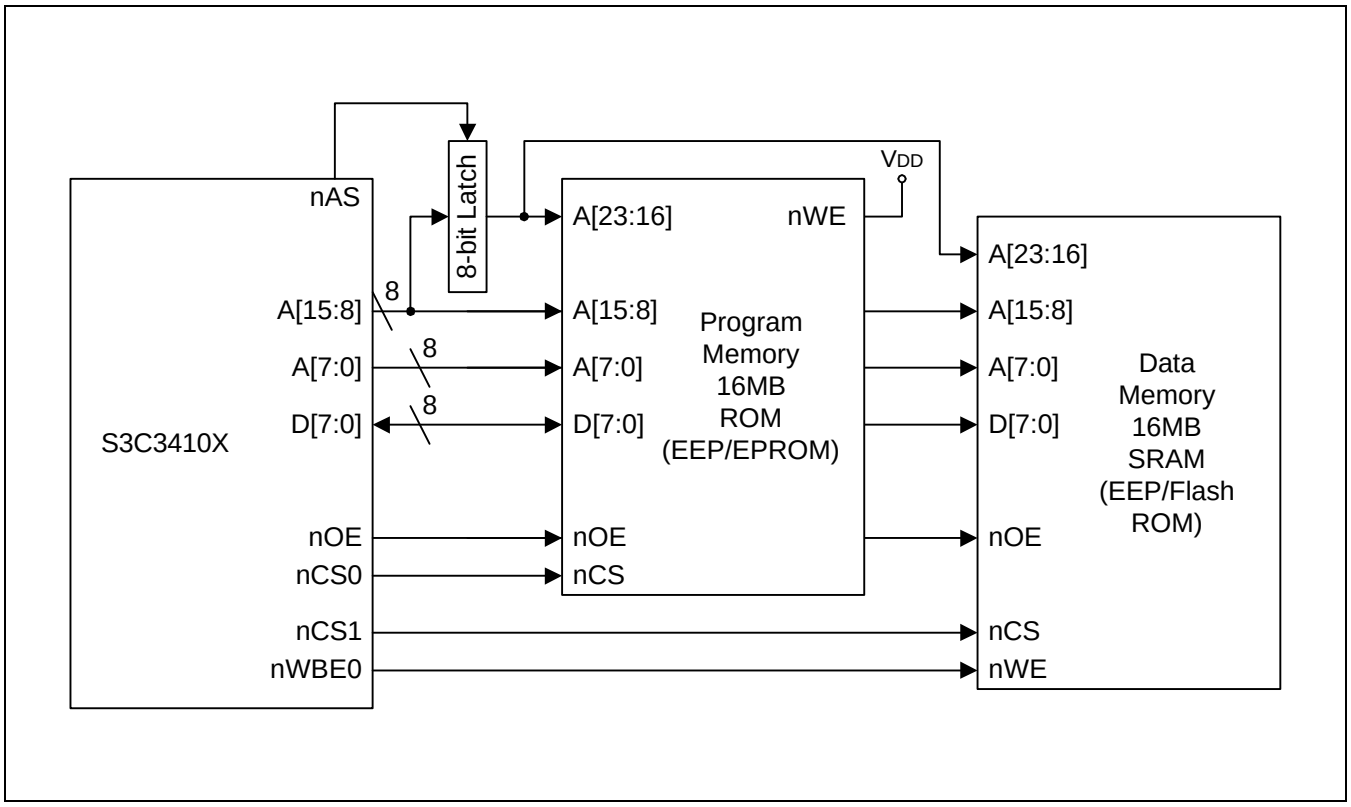


Figure 4-14. 16MB × 8 Program Memory & 16MB × 8 Data Memory

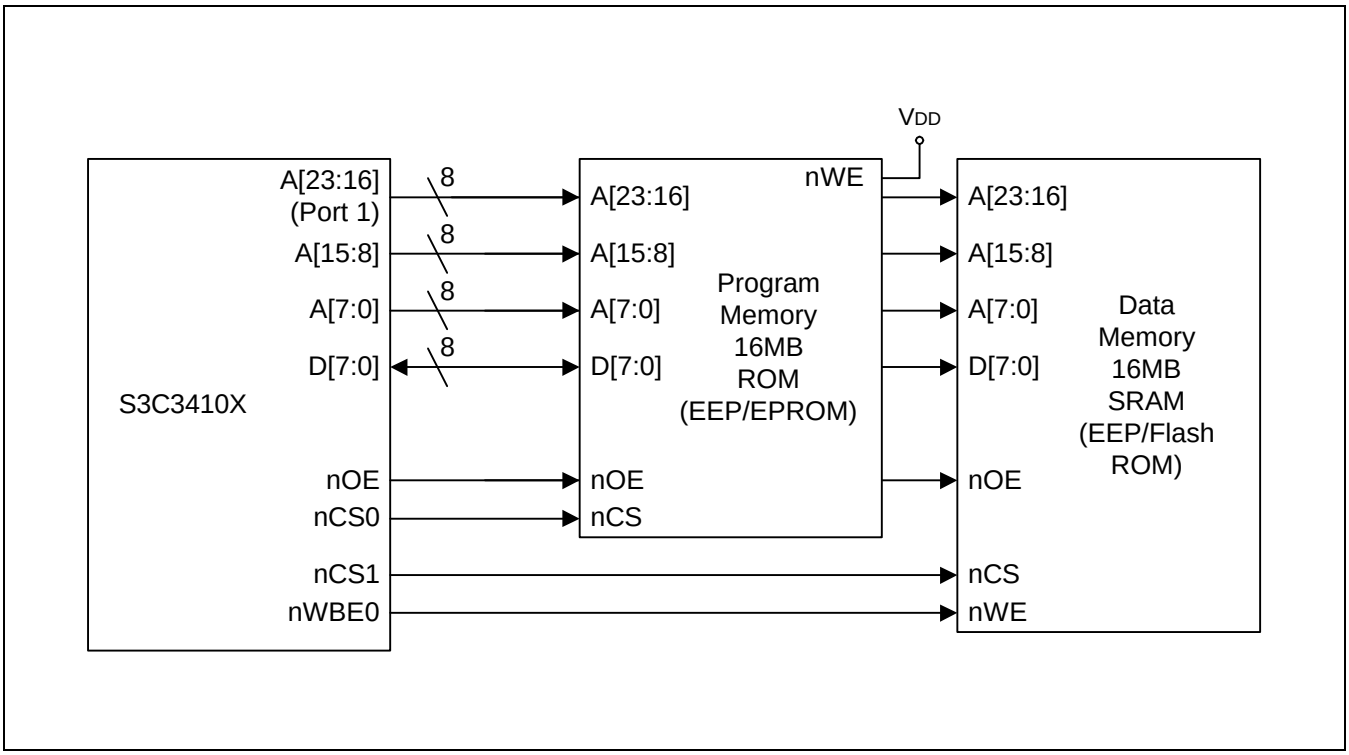


Figure 4-15. 16MB × 8 Program Memory & 16MB × 8 Data Memory

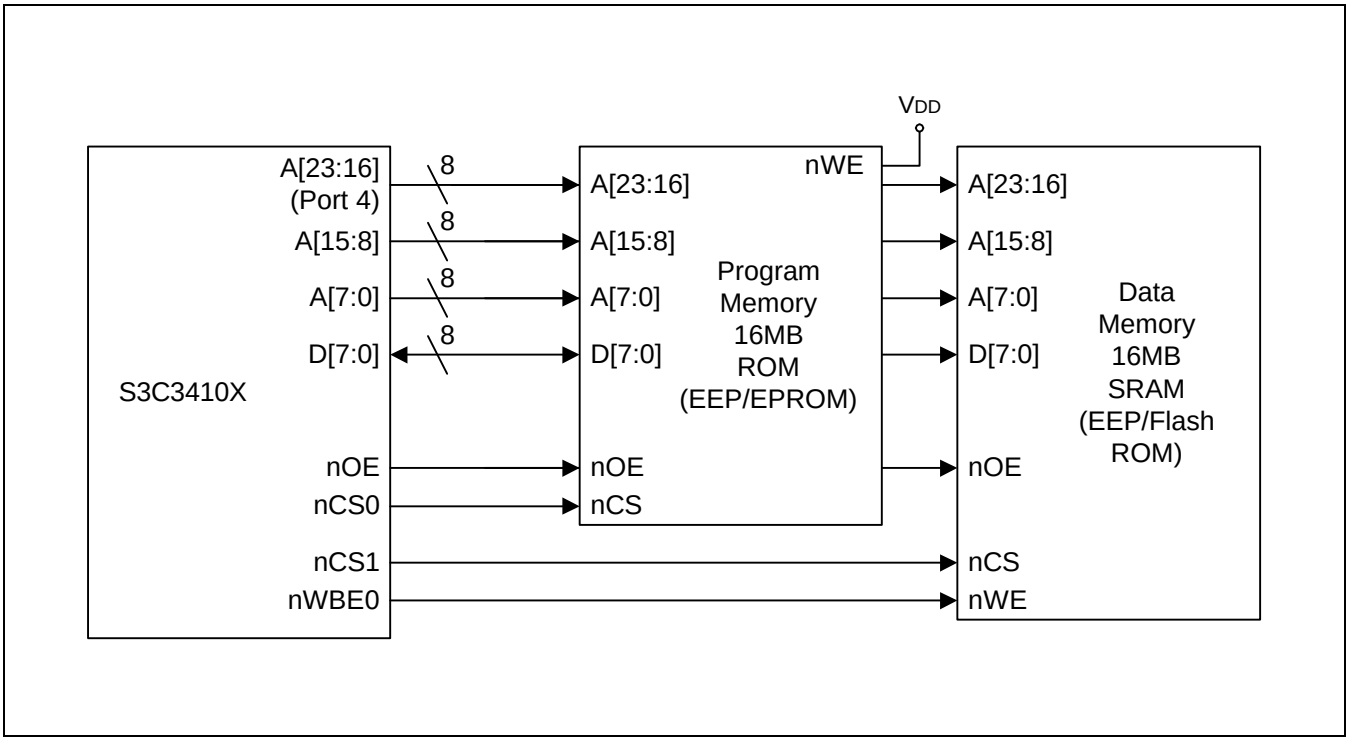


Figure 4-16. 16MB × 8 Program Memory & 16MB × 8 Data Memory

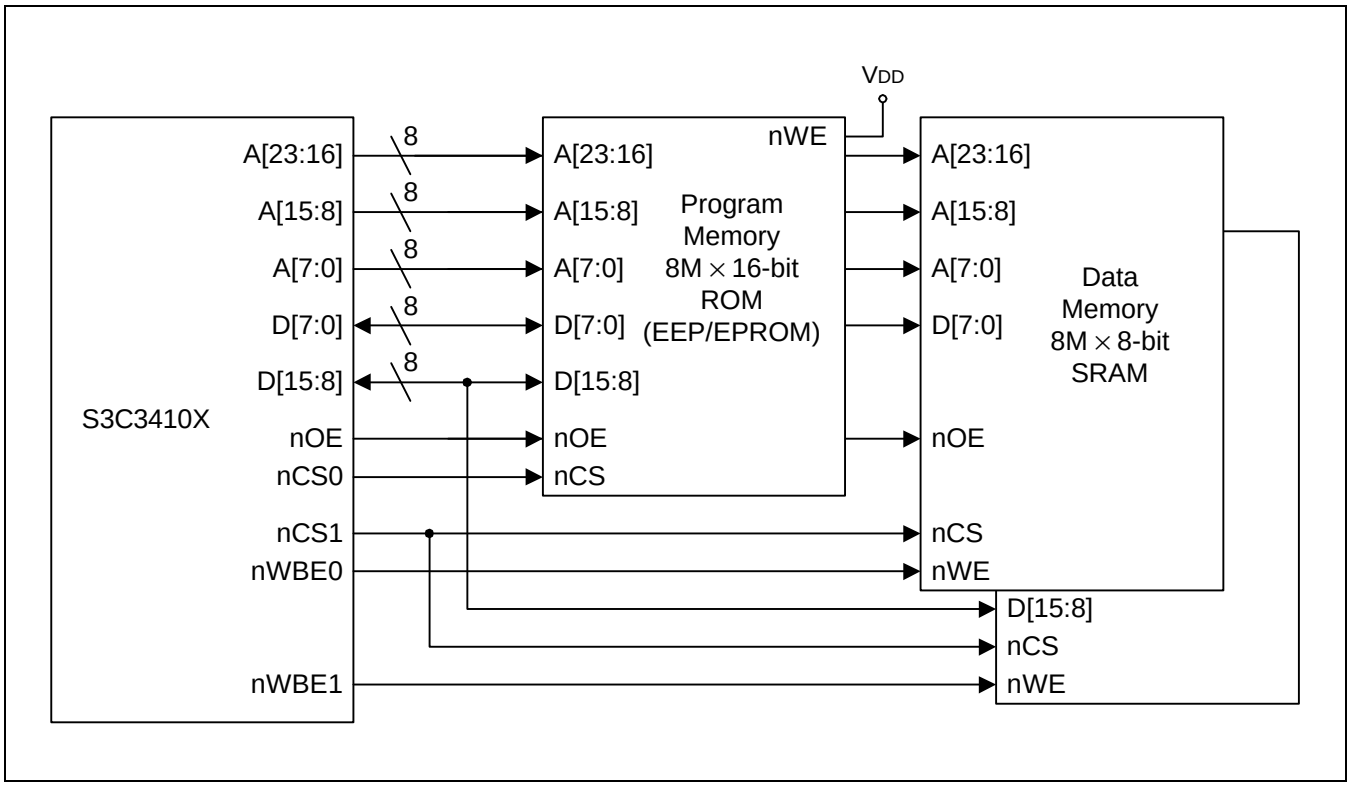


Figure 4-17. 8MB × 16 Program Memory & two 8MB × 8 Data Memory

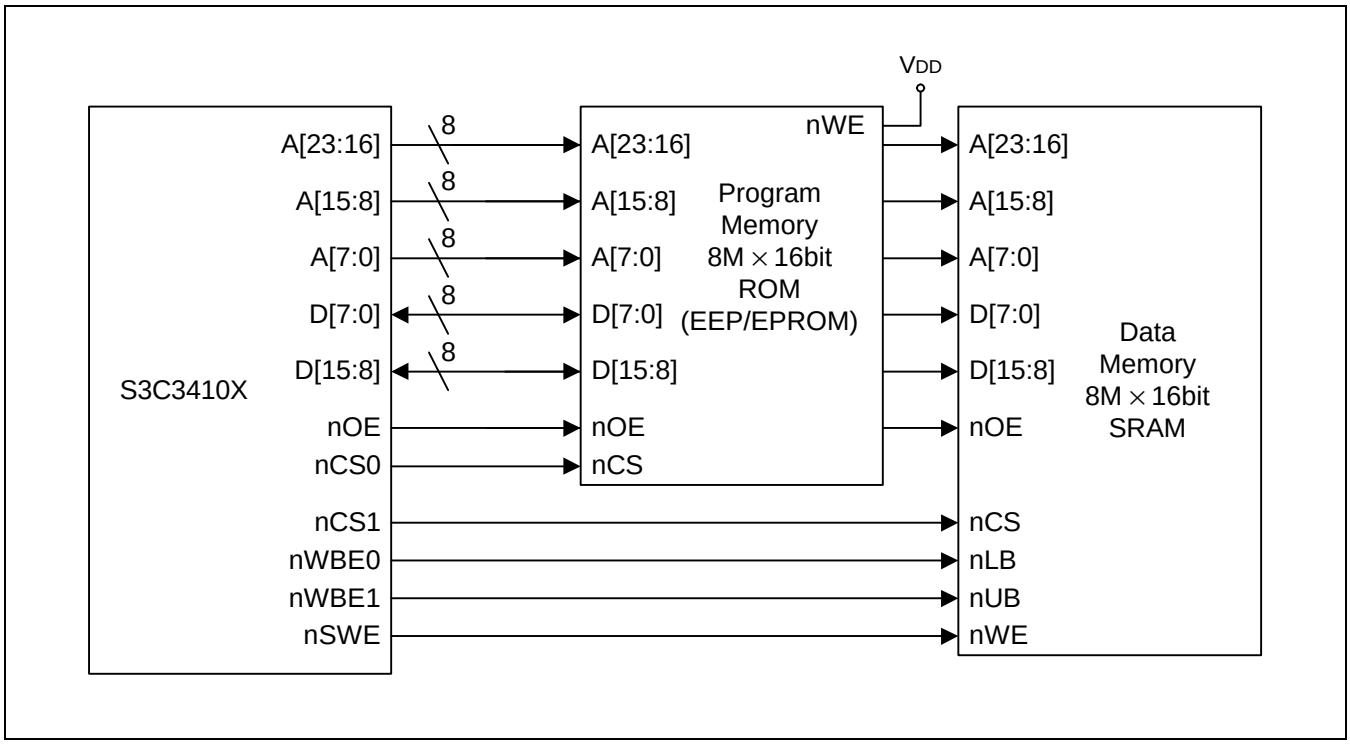


Figure 4-18. 8MB × 16 Program Memory & 8MB × 16 Data Memory (SRAM)

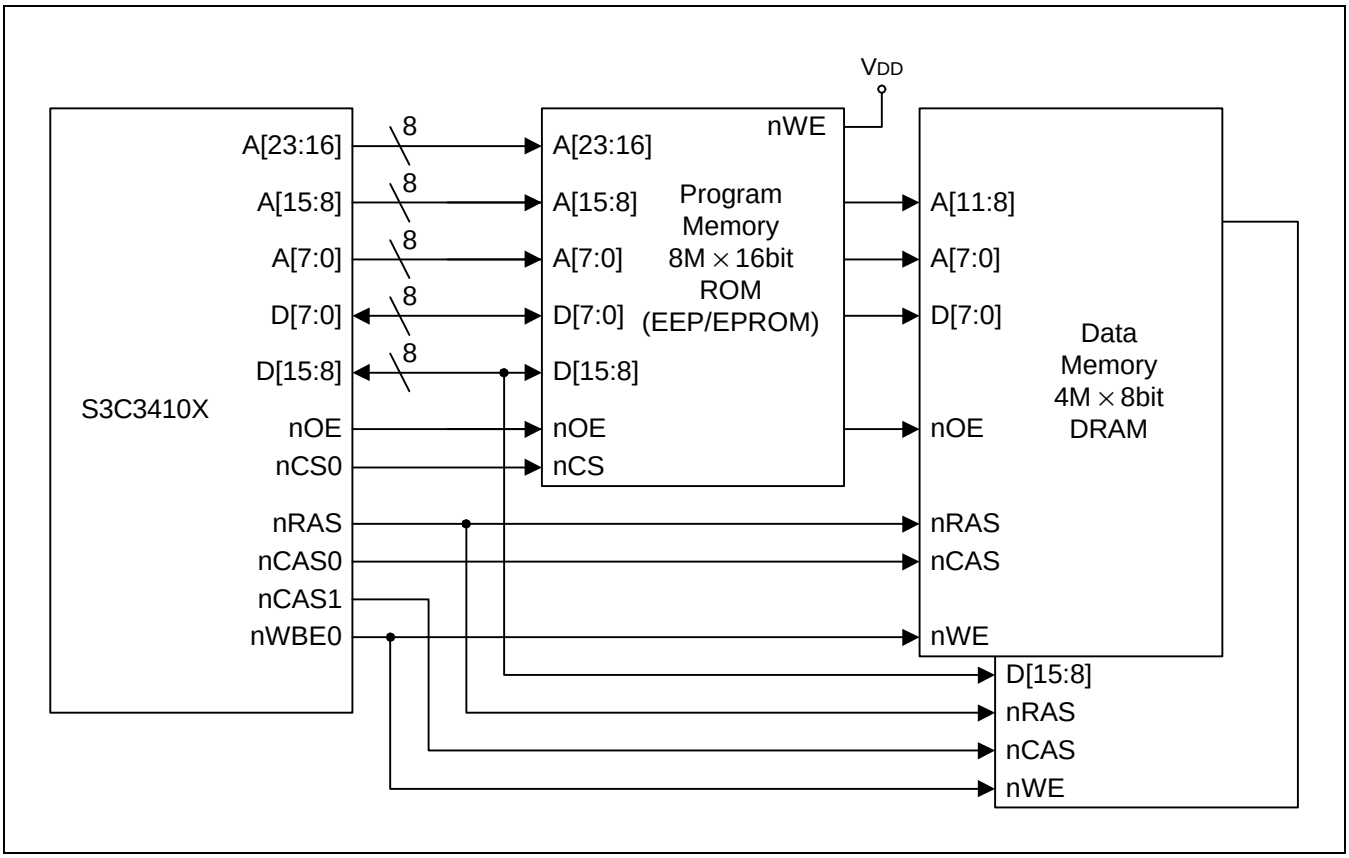


Figure 4-19. 8MB × 16 Program Memory & two 4MB × 8 Data Memory (DRAM)

Figure 4-20. 8MB × 16 Program Memory & 4MB × 16 Data Memory (DRAM)

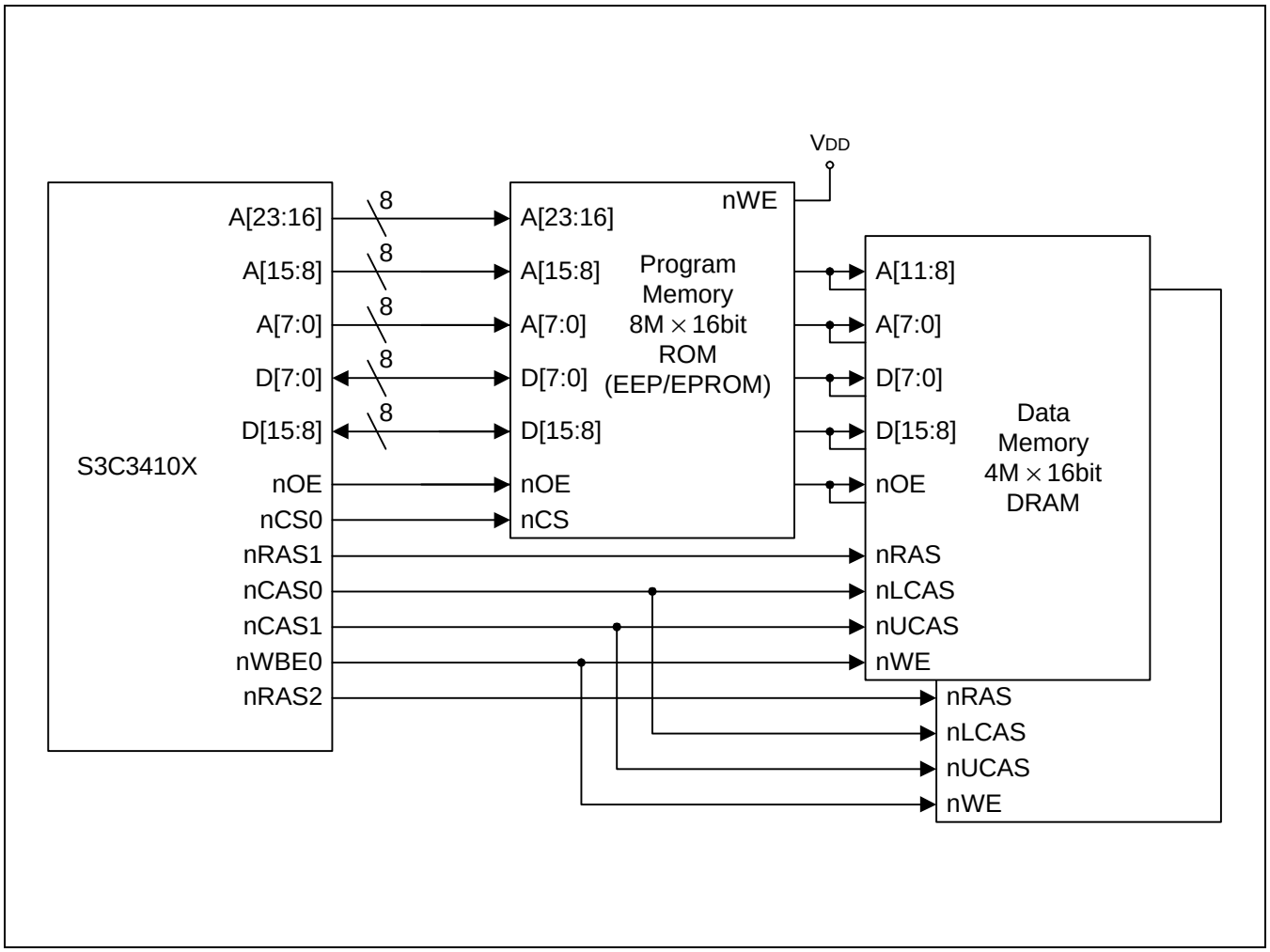


Figure 4-21. 8MB × 16 Program Memory & two 4MB × 16 Data Memory (DRAM)

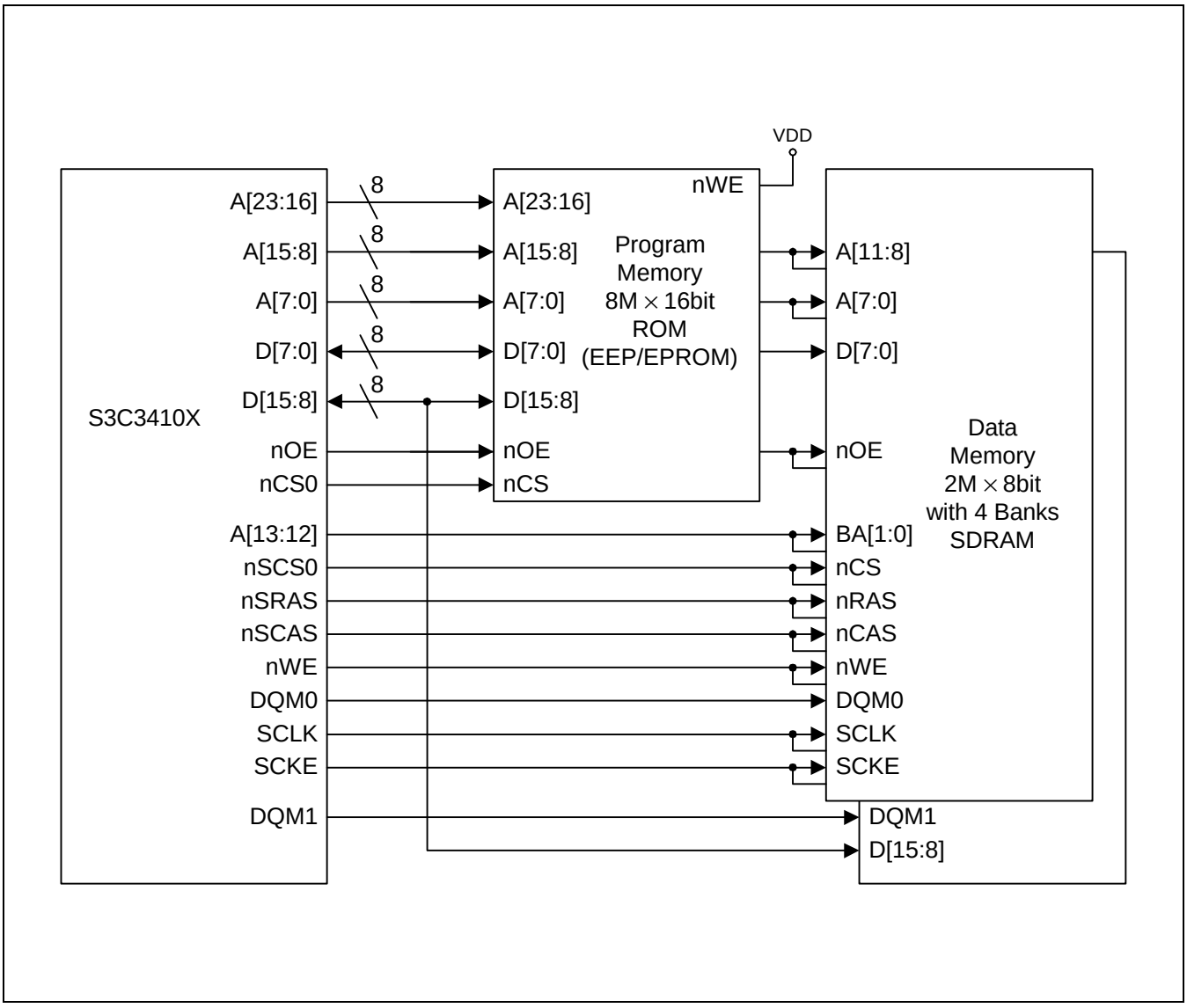


Figure 4-22. 8MB × 16 Program Memory & two 2MB × 8 with 4 Banks SDRAM

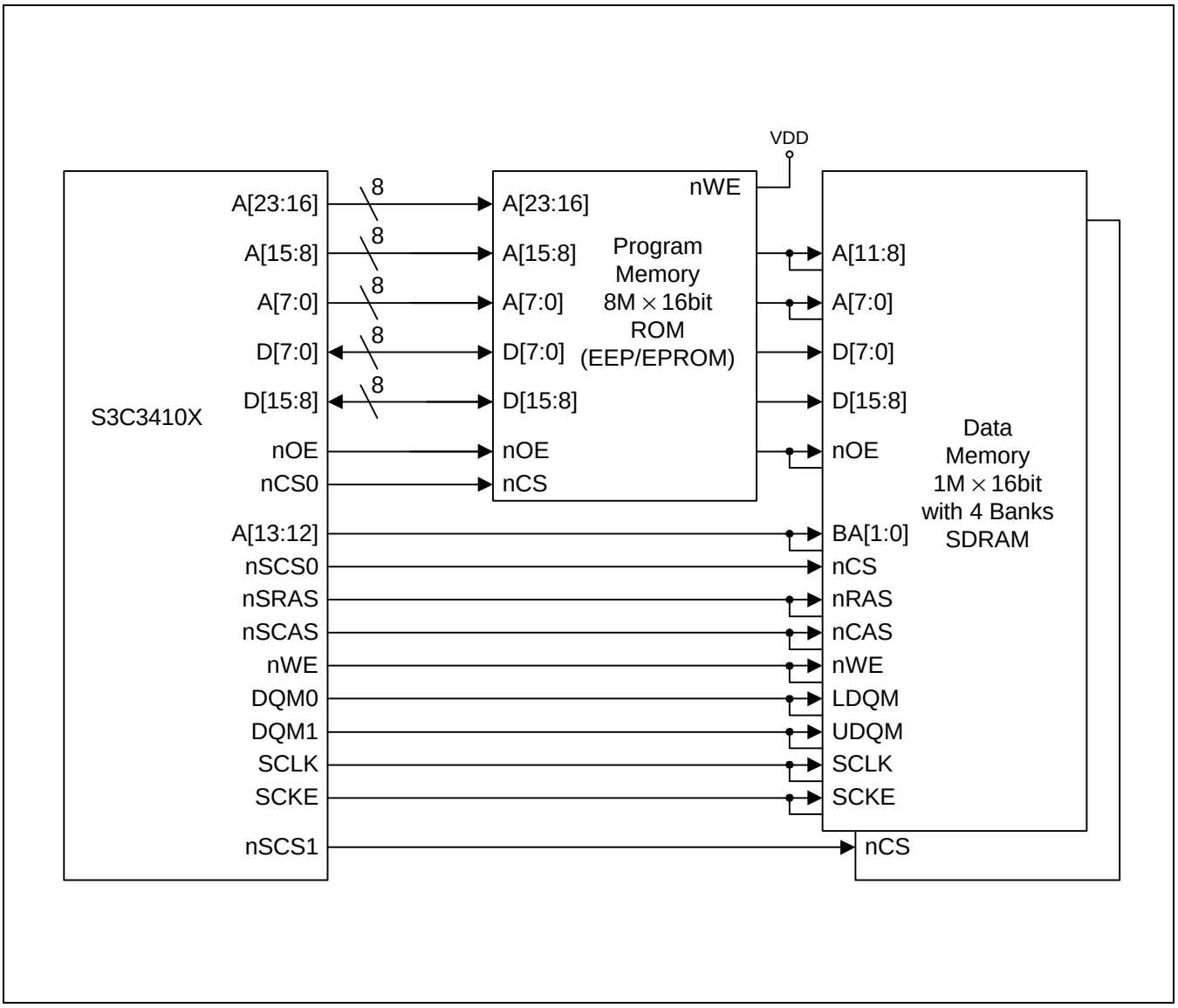


Figure 4-23. 8MB × 16 Program Memory & two 1MB × 16 with 4 Banks SDRAM

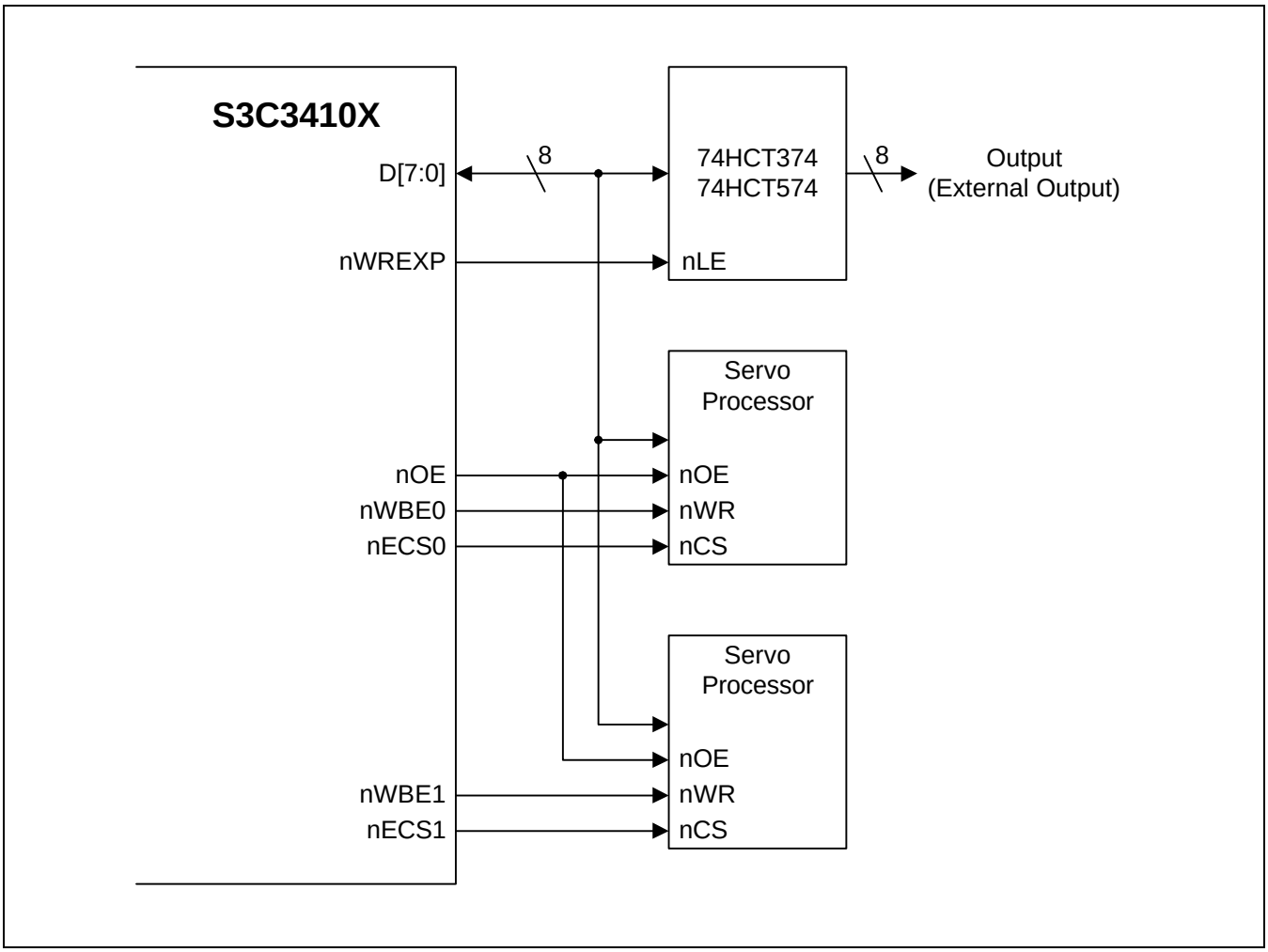


Figure 4-24. nWREXP, nECS0 and nECS1 Application Example in Normal Mode

NOTES

5 UNIFIED CACHE & INTERNAL SRAM

OVERVIEW

The S3C3410X has internal 4K-byte unified (Instruction/Data) cache. The cache architecture is based on two-way set associative and use the LRU(Least Recently Used) as cache replacement policy. To maintain the data coherence between main memory and cache, the cache controller should write the data into the main memory whenever the CPU update the data in cache memory. Because the cache line size is 4 word, there should be four word of memory fetch from main memory when cache miss happens. The cost-effective cache architecture can maintain the good hit ratio by investing the reasonable H/W inside the chip.

The performance difference between cache-on and cache-off is dramatically big. When cache is off, there is always instruction fetch from main memory. If we assume it takes 4 cycles for instruction fetch from main memory, the CPU performance will be dropped to 25% of the case of 10% cache hit due to the only instruction fetch from external memory. The 100% cache hit means that the CPU can fetch the instruction from memory within one cycle, i.e., zero wait. Usually, the user should turn on the cache to get the higher performance. But, if user does not want higher performance, the cache can be turn off to reduce the power consumption. If you turn the cache off and do not use the internal memory as SRAM, the power consumption will be reduced by 40%.

The S3C3410X can support the optional cache configuration. Internal 4KB memory can be configured as 4KB cache memory, 2KB Cache/2KB SRAM, or 4KB SRAM. Users can select these options suitable for their application.

The caching area of external memory can be determined to non-cache region by having the configuration. When the CPU access the non-cacheable region, these data should not be cached. Usually, the program and data area should be in cacheable region to get higher performance. But, the control-purposed data, for example, the data handling by DMA, should be in non-cacheable region. If the control data is in the cacheable region, if some of these data are cached into the cache memory, and if DMA update the data in the external memory of cacheable region, we can not guarantee the data coherence between data in cache memory and in external memory. Summarizing, users should always be aware of the memory allocation for non-cacheable and cacheable region.

The S3C3410X can support the 128MB addressing range and it means that the internal address A[26:0] are only effective even if the CPU can generate the A[31:0] of the internal address. If the S/W generate the address beyond this range, the cache controller and the memory controller will treat this address as special case. The reality is as follow. The cache controller accepts the address of A[27:0] and determine whether this access should be cached, or not when A[27]=0. In other word, the access should be cached if the A[26:0] is corresponding to the cacheable region and should not be cached if the A[26:0] is corresponding to the non-cacheable region. If A[27] = 1, the cache controller treat this access as non-cacheable access even if the A[26:0] is corresponding to cacheable or non-cacheable region. When A[27]=1 and the A[26:0] is corresponding to the cacheable region, the cache controller should treat this access as non-cacheable access and the memory controller should execute the memory access by using A[26:0] address. The cache controller discard the address of A[31:28] and the memory controller also discard the address of A[31:27].

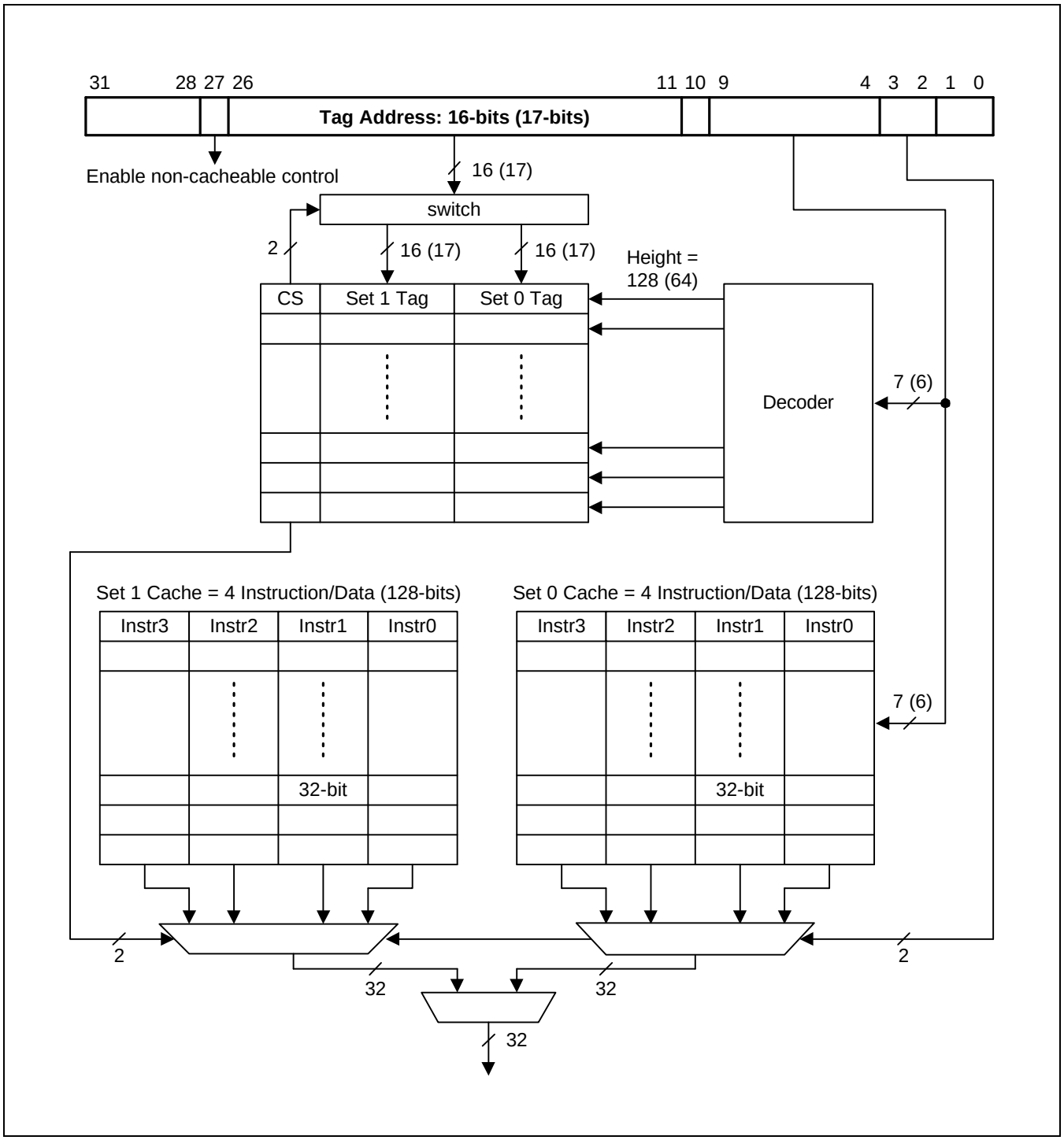


Figure 5-1. Cache Memory Configuration

CACHE OPERATION

CACHE ORGANIZATION

The S3C3410X cache has a 4KB or 2KB cache memory and Tag RAM. The cache architecture consists of 2-way set associative, has 4 word as line size, and uses the LRU replacement policy. To maintain the data coherence between cache and main memory, the S3C3410X supports the WT(Write Through). The Tag RAM has a 2-bit CS(Cache Status) field as well as Tag data for set 0 and 1 as shown in Figure 5-1. Each Tag set has a 16-bits/17-bits Tag address of A[26:11] / A[26:10] for 4KB /2KB cache if the address of A[26:0] is cached in the cache memory. The 2-bit CS indicates the validity of cached data of the corresponding cache memory line. It is also used for the cache replacing algorithm and for selecting the data coming from set 0 and 1. Because the cache consists of 2-way set associative, each set should have 2KB. The one line is 4-word(4×32 = 128bit), and there should be 128 lines in each set. If users specify the 2KB cache, one line is 4-word(4×32 = 128bit) and there should be 64 lines in each set. The TAG and Cache array memory are mapped to the specific address range and users can access these memory by S/W, which will be explained in Cache Flush.

CACHE REPLACE OPERATION

After the system is initialized, the value of CS is set to "00", notifying that the memory content in set 0 and 1 are invalid. When a cache fill occurs, the value of CS is changed to "01" at the specified line, which notifies that the set 0 is only valid. When the subsequent cache fill occurs, the value of CS will be "11" at the specified line, which notifies that the memory content in both set 0 and set 1 are valid. When the memory content in both set 0 and set 1 are valid, there should be cache replacement when the cache miss happens. During the miss cycle, the value of CS should be changed to "10" at the specified line, notifying that the memory content in set 0 will be replaced. After the completion of miss cycle, the value of CS will be changed to "11", again because the specified cache was re-filled. If there happens other miss cycle on the same line, the value of CS should be changed to "01" at the specified line, notifying that the memory content in set 1 will be replaced. After the completion of miss cycle, the value of CS will be changed to "11", again because the specified cache was re-filled. To indicate the Least Recently Used line, there is an internal toggling bit which determines that the recent access was to set 0 or set 1.

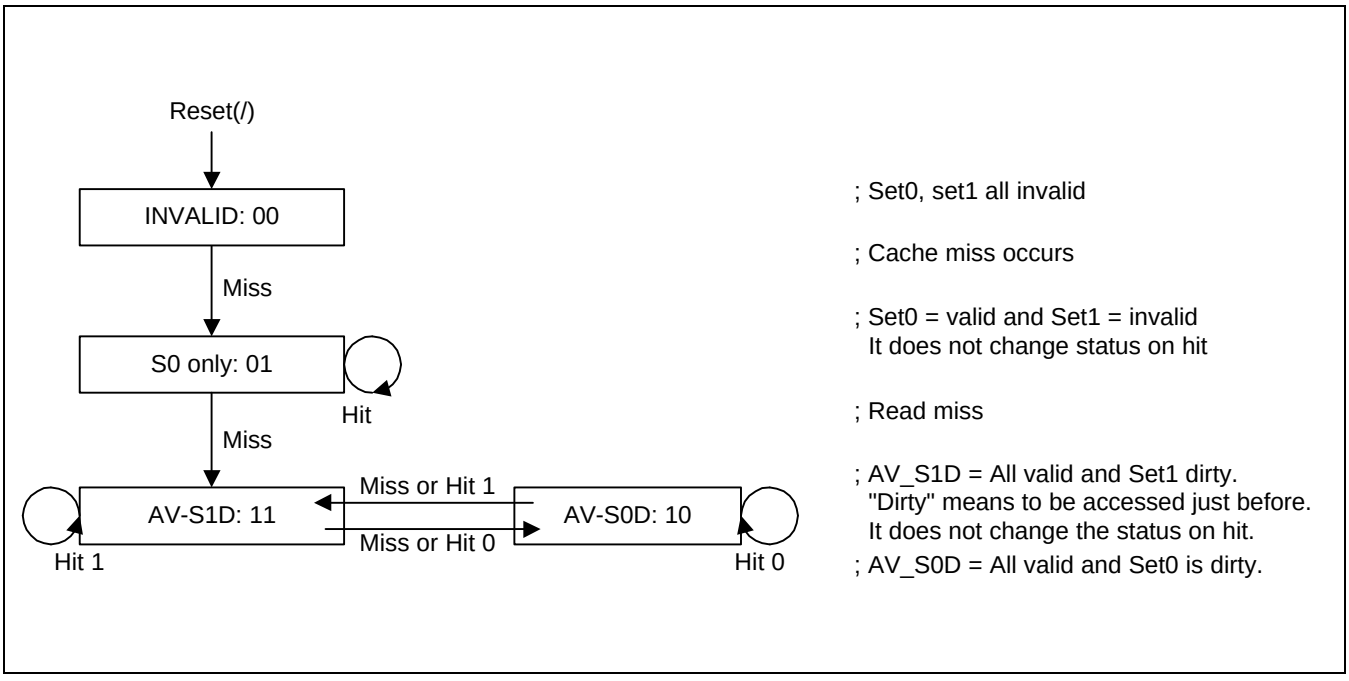


Figure 5-2. CS-Bit Status Diagram

CACHE DISABLE OPERATION

The S3C3410X cache can support the programmable entire-cache-enable/disable mode. Users can enable the cache by setting the value of CE bit in SYSCFG to "1", and disable it by clearing the value of SYSCFG to "0". When the cache disable mode is selected, the instruction and data should always be fetched from the external memory. The S3C3410X can also support the option for cache size of 0KB, 2KB, and 4KB by the Cache Mode bits(SYSCFG[16:15]]). When the reset, the default status is 4KB cache. If users specify the less cache size than 4KB, the remained memory can be used as an internal SRAM.

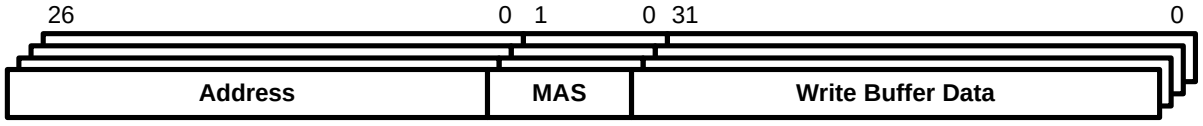
That is to say, if you want to use the internal memory as an internal SRAM, the memory allocation table of the internal SRAM is as follows:

Item	Address	Comment
Internal SRAM	(SFR start address) – (SFR start address + 0x7ff)	2KB
	(SFR start address + 0x800) – (SFR start address + 0xfff)	2KB

The S3C3410X can support the WT(Write Through) to maintain the coherency between the cache and main memory. When ever the CPU updates the cache memory, the cache controller should issue the updating cycle of main memory content through the memory controller, automatically. Users should also be cautious about the data coherency when they specify the cacheable region. For example, if the DMA has the possibility to update the memory content, the memory region should be non-cacheable.

WRITE BUFFER OPERATION

The S3C3410X has four Write Buffer Register to enhance the performance. The role of write buffer is as follows: When the CPU try to write its data into the external memory, the memory controller can not execute the memory cycle if some other master, for example, DMA is using the external bus. In this case, the performance will be degraded if the CPU and memory controller should wait the bus free. To avoid this situation, the S3C3410X has internal four-depth Write Buffer Register. In this case, the CPU should write its data into the Write Buffer Register and execute its next operation. If the bus is free, the Write Buffer Register requests the bus cycle to memory controller. The Write Buffer also need the TAG address of A[26:0] because the Write Buffer should return the accessed data to the CPU when the CPU requests the Read operation again before the data update into the main memory.



[31:0] Write Buffer Data
Data to be written into external memory

[1:0] MAS
00 = 8-bit data mode
01 = 16-bit data mode
10 = 32-bit data mode
11 = Not used

[24:0] Address
Indicates the address of write buffer

Figure 5-3. Write Buffer Configuration

CACHE FLUSHING

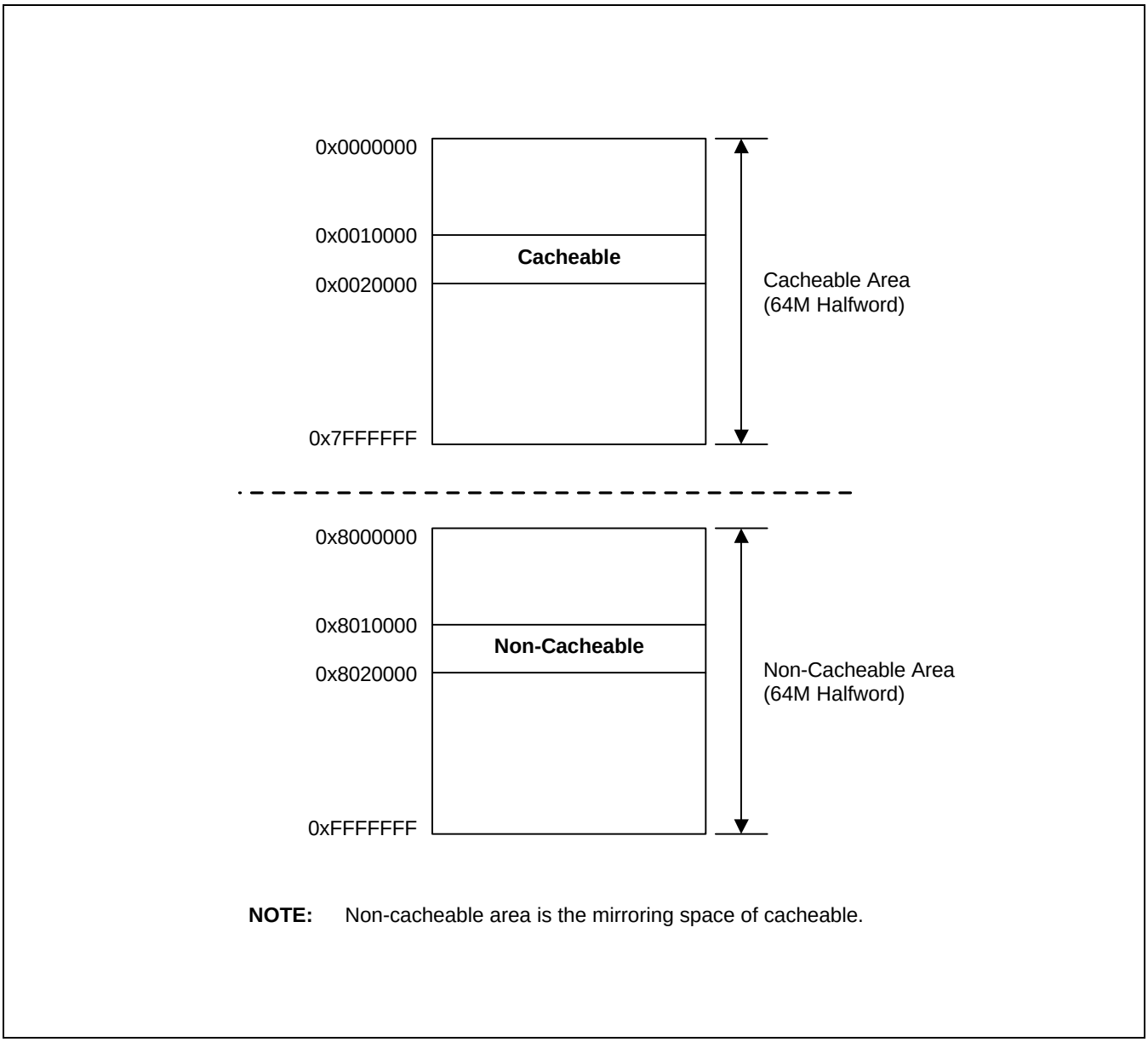
The cache content as well as Tag at the specific line can be accessed by S/W. In S3C3410X, the memory array of set 0 is mapped to the address of 0x10000000 – 0x100007ff, which is 2KB size. Similarly, the memory array of set 1 is mapped to the address of 0x10800000 – 0x108007ff, which is also 2KB size. The Tag array is also mapped to the address of 0x11000000 – 0x110001ff, which is 512B size. As we explained in previous chapter, the width of Tag data is total 36-bit, which consists of 2bit CS, 17/16-bit Tag data for 2KB/4KB for set 0, and 17/16-bit Tag data for 2KB/4KB for set 1. In detail, the 16-bit Tag data(Tag[15:0]) of set 1 and 16-bit Tag data(Tag[15:0]) of set 1 is mapped to the address of 0x11000000. The CS field of the Tag is mapped to the address of 0x11000004. In this case, CS[1] and CS[0] are corresponding to the data bus of D[31] and D[30]. If user specify the 2KB cache size, lower 16-bit Tag data of set 1 and lower 16-bit Tag data of set 0 is mapped to the address of 0x11000000. The remained CS field, upper Tag bit of set 1 and upper Tag bit of set 0 are mapped to the address of 0x11000004. In this case, CS[1], CS[0], Tag[16] for set 1, and Tag[16] for set 0 are corresponding to the data bus of D[31], D[30], D[29], and D[28]. The next line will be corresponding to the address of 0x11000008, 0x11000010, and so on. The memory allocation table of the Tag RAM and Set 0, 1 cache memory is as follows:

Item	Address	Comment
Set 0	0x10000000 – 0x100007ff	2KB
Set 1	0x10800000 – 0x108007ff	2KB
Tag RAM	0x11000000 – 0x110001ff	512B

NOTE: Cache flushing must be executed only in the cache disable mode.

NON-CACHE AREA CONTROL BIT

The S3C3410X can support the 128MB addressing range and it means that the internal address A[26:0] are only effective even if the CPU can generate the A[31:0] of the internal address. If the S/W generate the address beyond this range, the cache controller and the memory controller will treat this address as special case. The reality is as follow. The cache controller accepts the address of A[27:0] and determine whether this access should be cached, or not when A[27]=0. In other word, the access should be cached if the A[26:0] is corresponding to the cacheable region and should not be cached if the A[26:0] is corresponding to the non-cacheable region. If A[27] = 1, the cache controller treat this access as non-cacheable access even if the A[26:0] is corresponding to cacheable or non-cacheable region. When A[27]=1 and the A[26:0] is corresponding to the cacheable region, the cache controller should treat this access as non-cacheable access and the memory controller should execute the memory access by using A[26:0] address. The cache controller discard the address of A[31:28] and the memory controller also discard the address of A[31:27].



NOTE: Non-cacheable area is the mirroring space of cacheable.

Figure 5-4. Non-cacheable Area

NOTES

6 DMA (DIRECT MEMORY ACCESS)

OVERVIEW

The S3C3410X has two general Direct Memory Access channels (DMA0, DMA1) which performs the data transfer between the following source/destination and destination/source without CPU intervention:

- Memory(or Internal SRAM) and Memory (or Internal SRAM)
- UART and Memory (or Internal SRAM)
- SIO and Memory (or Internal SRAM)
- SFR and Memory (or Internal SRAM)
- SFR and SFR (Including UART, SIO, Ext. I/O, Timer1/3)

The on-chip DMA controller can be started by software, by two external DMA requests(nDREQ0, nDREQ1) or by SIO 0, SIO1, UART, Timer1, and Timer3. The DMA operation can also be stopped and restarted by software. The CPU can recognize the completion of DMA operation by software polling or interrupt request from DMA. The source and/or destination address can be increased or decreased during DMA operation and the DMA can support the transfer size by byte, half-word, and word unit.

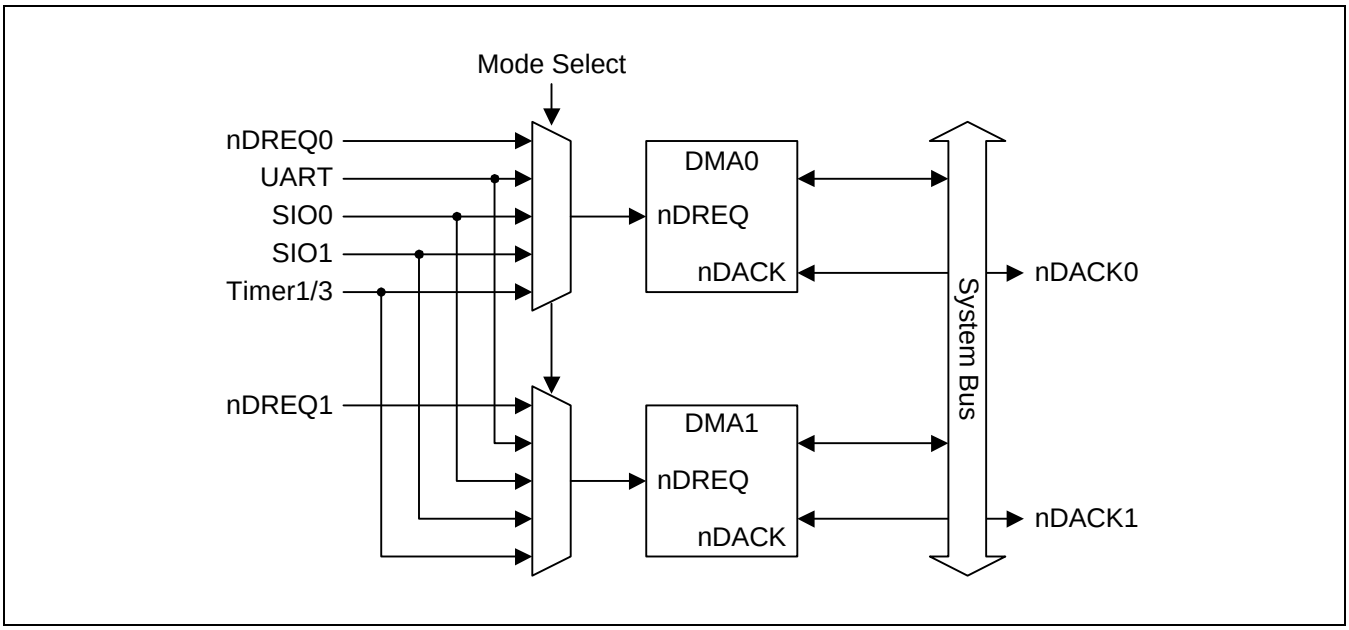


Figure 6-1. DMA0/DMA1 Unit Block Diagram

DMA OPERATION

The DMA operation can be summarized as follows:

- DMA transfer
- Bus arbitration control
- Starting/Stopping DMA transfer

DMA Transfer

The DMA(Direct Memory Access) can transfer the data directly between source and destination. The source or the destination should be memory including internal SRAM, UART, SIO, or other SFR. The external devices can request the DMA service by activating the nDREQ0/1 signal.

The operation of DMA channel should be programmed by configuring the DMA control registers, which contain the control information such as the direction of the source address, or destination address, and transfer size. The UART, SIO, Timer1/3, external devices and software can request DMA service. For example, the UART, SIO, and Timer1/3 can request the DMA service when they are ready to need the DMA operation. For example, the UART can request the DMA service to DMA controller when the UART finish receiving the data from port and ready to send the received data to external memory by using DMA. Differently from internal devices, the external device can activate the nDREQ0/1 signal to request the DMA service to S3C3410X. To make the DMA ready for its operation, users should specify the necessary control information such as source/destination address, transfer size, and transfer count. After the completion of these configuration, user can start the DMA operation by software.

Bus Arbitration Control

Because the DMA operation need the occupation of bus usage, the arbitration should be essential. As well as DMA, the memory controller inside chip need the bus usage. If there happens simultaneous bus request among master devices, there should be arbitration process in S3C3410X. The S3C3410X can do the arbitration process base on fixed priority. The priority of these bus master devices is as follows:

Bus Master Type	Priority
Memory Controller(DRAM/SDRAM refresh)	1
DMA0	2
DMA1	3
Write Buffer	4
CPU Core	5

Starting/Stopping DMA Transfer

The DMA can start its operation of transferring the data when the DMA controller receives the request from the nDREQ signal through external pin, request from UART, request from SIO, or request from Timer1/3. In case of data transfer between memories, the DMA can also start its operation when the user write the start bit(Run bit) in DMA control register. When the entire data transfer specified in DMACNT has been finished, the DMA goes into the idle mode. If users want to perform another DMA operation, the configuration of DMA operation should be programmed again. The users can stop the DMA operation before its complete termination. By clearing the start bit(Run bit), the users can stop the DMA operation even if the specified DMA operation is not finished. When users stop the DMA operation, there will be interrupt generation which depends on the SI(Stop Interrupt) bit in DMA control register. If SI bit is 0 in DMA control register, there will be DMA operation stop without the interrupt generation. If users want to resume the DMA operation, users should re-run the DMA operation by setting the start bit(RE bit) in DMA control register. To guarantee the complete DMA re-run, users should not change the DMA configuration before the re-start.

DATA TRANSFER MODE

Single Step Mode

The single step mode is usually used for test or debugging because the bus mastership can be handed over to other bus master between Read and Write. For the initiation of DMA operation, we need the activation of nDREQ for each Read and Write cycle and there should be separate activation of nDACK for each Read and Write cycle. In other word, we need two times DMA request and two times DMA acknowledge for single DMA operation. For this reason, this kind of DMA operation is too slow and this is only for debugging purpose. During the inactive period of nXDACK, i.e., between Read and Write cycle, the bus controller re-evaluates the bus priority to determine the new bus mastership.

When the DMA request signal goes low, the bus controller can indicate the bus allocation for the DMA operation by lowering the DMA Acknowledge signal if there is not higher priority bus request except this DMA request. During the first low level period of the DMA Acknowledge signal, there will be a DMA read cycle. After the DMA read cycle, there will be a rising of the DMA Acknowledge signal to indicate the end of the DMA read cycle. Simultaneously, the next DMA write cycle will happen if the DMA request signal is still low at the rising edge of DMA acknowledge. But, if the DMA request signal is already high at the rising edge of DMA acknowledge, the next DMA write cycle will be delayed to the new coming activation of DMA Request signal. The Single Step Mode of DMA operation can be initiated by the request from UART or SIO or Timer1/3 as well as nDREQ.

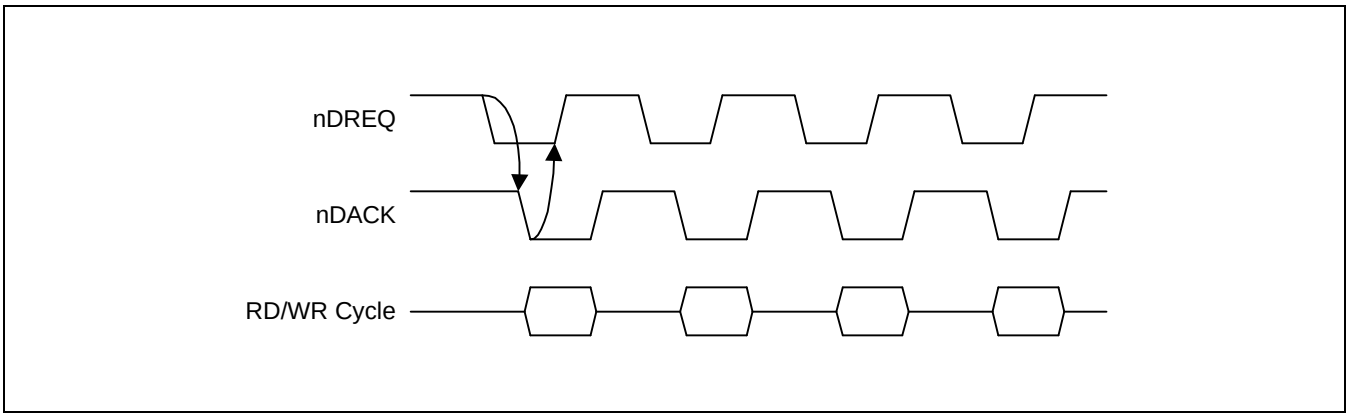


Figure 6-2. External DMA requests (Single Mode)

Block Transfer Mode

The block transfer mode means that the DMA operation will be continued up to the end of transfer count. Usually, the DMA needs the request signal during the unit-by-unit transfer. The block transfer mode need just one time request for whole service of DMA operation, which is shown in Figure 6-3. This transfer mode can monopoly the bus usage if users set the CM(Continuous Mode) bit in DMA control register and it can be harmful for other bus mastership. Therefore, users should be aware of the worst case situation when they need this mode for faster data transfer. If users take the block transfer mode without setting the CM bit, there will be no bus monopoly. It means that the higher bus master can take the bus usage during the block transfer.

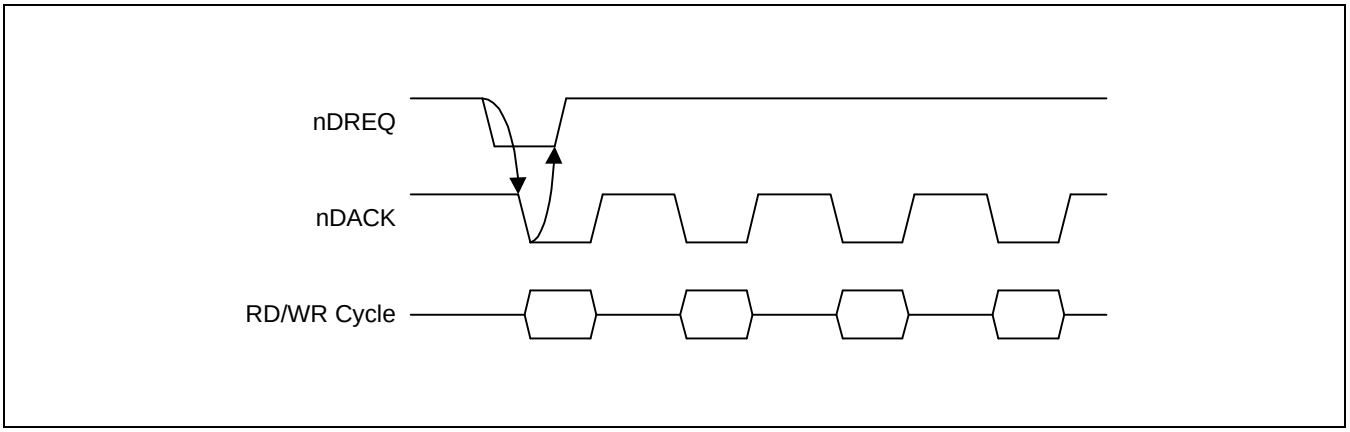


Figure 6-3. External DMA requests (Block Mode)

Demand Mode

The demand mode means there will be continuous DMA transfer cycles as long as the activation of DMA Request signal as shown in figure 6-4. This mode doesn't permit the bus hand-over even though the higher priority bus master request the bus mastership to bus controller during DMA operations. In other word, no other bus master can have the bus mastership during the demand mode. Due to the monopoly of bus mastership in demand mode, we should be aware of the fact that the duration of the demand mode must not exceed the specified maximum time such as the DRAM refresh period.

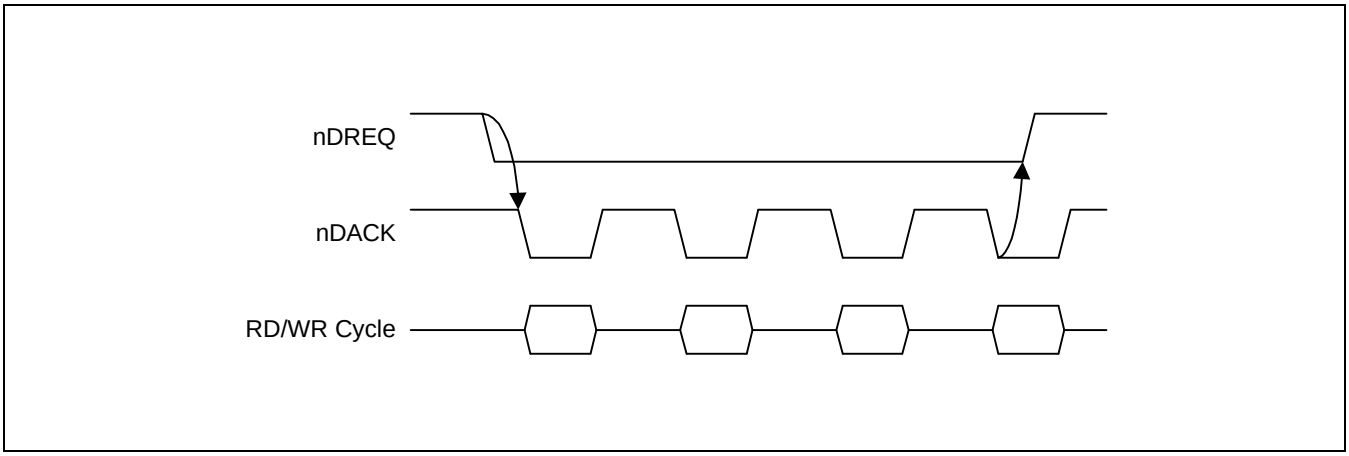


Figure 6-4. External DMA requests (Demand Mode)

DMA SPECIAL FUNCTION REGISTER

DMA CONTROL REGISTERS

Register	Offset Address	R/W	Description	Reset Value
DMACON0	0x300c	R/W	DMA 0 control register	0x0
DMACON1	0x400c	R/W	DMA 1 control register	0x0

DMACONx	Bit	Description	Initial State
RE	[0]	**Run Enable:** This bit determines the enable or disable of DMA operation. To start the DMA operation, this bit should be set. To stop the DMA operation, users can reset this bit. 0 = Disable 1 = Enable	0
BS	[1]	**BUSY Status:** When the DMA start its operation, this read-only status bit is set to "1" automatically. When the DMA is in an idle state, this bit is set to "0". This bit is "read-only".	0
MODE	[3:2]	**Mode Select:** These bits determine the source of DMA initiation. The initiation of DMA operation can be done by S/W, external nDREQ, UART, or SIO/Timer. DMACON0 DMACON1 00 = Software 00 = Software 01 = External nDREQ0 01 = External nDREQ1 10 = UART 10 = UART 11 = SIO, Timer 11 = SIO, Timer	00
DD	[4]	**Destination Address Direction:** This bit determines whether the destination address will be decreased or increased during a DMA operation 0 = Increase address 1 = Decrease address	0
SD	[5]	**Source Address Direction:** This bit determines whether the source address will be decreased or increased during a DMA operation 0 = Increase address 1 = Decrease address	0
DF	[6]	**Destination Address Fix:** This bit determines whether the destination address should be changed during a DMA operation, or not. If users take DF option, the destination address will be fixed. 0 = Increase/Decrease destination address 1 = Do not change destination address (fix)	0

DMACONx	Bit	Description	Initial State
SF	[7]	**Source Address Fix:** This bit determines whether the source address should be changed during a DMA operation, or not. If users take SF option, the source address will be fixed. 0 = Increase/Decrease source address 1 = Do not change source address (fix)	0
SI	[8]	**Stop Interrupt Enable:** The DMA operation can be started by setting RE bit to "1" and can also be stopped by resetting RE bit to "0". When this SI bit is set to "1", and when the DMA operation is forced to stop, there will be "stop interrupt" generation. If this bit is "0", the "stop interrupt" will not be generated. The DMA done interrupt, which is generated after the DMA counter is expired, can not be masked by this bit. 0 = Do not generate the stop interrupt when DMA stops 1 = Generate the stop interrupt when DMA stops	0
BT (note)	[9]	**4 Burst Enable:** When the MODE bit is set to "1" , the DMA operation will be done by the burst transfer mode. The size of burst will depend on TW field in this register. If TW is word unit, there will be four times word transfer. 0 = Normal transfer 1 = 4 Burst transfer	0
Reserved	[10]	Reserved	0
SB	[11]	**Single/Block Mode:** This bit determines the number of external DMA request (nDREQ) that are required for the DMA operation. 0 = One nDREQ initiates a single DMA operation 1 = One nDREQ initiates a block DMA operation	0
TW	[13:12]	**Transfer Width:** This bit determines the transfer data width: byte(8-bit), half-word(16-bit) and word(32-bit). If the transfer width is a byte, source/destination address will be increased/decreased by one(Byte address unit), If it is a half-word, the address will be increased/decreased by two(Half-word address unit). If it is a word, the address will be increased/decreased by four(Word address unit). Note that the "transfer width" is not the physical size of data bus. The physical size of data bus is determined by SMR(System Manager Register) configuration. 00 = Byte(8-bit) 01 = Half-word(16-bit) 10 = Word(32-bit) 11 = Not used	00

DMACONx	Bit	Description	Initial State
CM	[14]	**Continuous Mode:** This bit determines whether the DMA operation should monopoly the system bus, or not until the transfer count value reaches to zero. 0 = Normal operation 1 = Monopoly the system bus until the completion of DMA operation.	0
DM	[15]	**Demand Mode:** To speed up the external DMA operation, set this bit. If this bit is set, the DMA operation will be continuously proceeded while nDREQ is activated. In this case, other higher bus master can not take the bus usage while the operation of this Demand mode. 0 = Normal external DMA mode 1 = Demand mode	0

NOTE: If a DMA is set as four data burst and continuous mode together, four burst mode is ignored, and the continuous mode only is operated. In order to use four burst mode in DMA operation, please be sure that continuous mode is disabled.

DMA SOURCE/DESTINATION ADDRESS REGISTER

These registers contain the 27-bit source/destination address of a DMA channel. Depending on the setting of the DMA control register (DMACONx), these addresses will be increased/decreased or will be fixed without changing.

Register	Offset Address	R/W	Description	Reset Value
DMASRC0	0x3000	R/W	DMA 0 source address register	0x0
DMADST0	0x3004	R/W	DMA 0 destination address register	0x0
DMASRC1	0x4000	R/W	DMA 1 source address register	0x0
DMADST1	0x4004	R/W	DMA 1 destination address register	0x0

DMASRC0	Bit	Description	Initial State
	[26:0]	Initial source address for DMA0	0x0

DMASRC1	Bit	Description	Initial State
	[26:0]	Initial source address for DMA1	0x0

DMADST0	Bit	Description	Initial State
	[26:0]	Initial destination address for DMA0	0x0

DMADST1	Bit	Description	Initial State
	[26:0]	Initial destination address for DMA1	0x0

DMA TRANSFER COUNT REGISTER

These registers contain the 26-bit count value which is the number of DMA transfer. This value is decreased by 1 when one DMA operation is completed regardless of the width of the data which should be transferred. If the DMA operates 4 burst mode, the DMACNT is decreased by 1 when 4 data transfer is completed

Register	Offset Address	R/W	Description	Reset Value
DMACNT0	0x3008	R/W	DMA transfer count register for DMA0	0x0
DMACNT1	0x4008	R/W	DMA transfer count register for DMA1	0x0

DMACNT0,1	Bit	Description	Initial State
	[26:0]	Number of DMA transfer	0x0

NOTES

7 I/O PORTS

OVERVIEW

S3C3410X has 74 multiplexed input/output port pins. There are ten port group, which are eight 8-bit I/O port group, one 2-bit output port group, and one 8-bit input port group:

- Eight 8-bit input/output ports (Port 0, 1, 2, 3, 4, 5, 6 and 7)
- One 2-bit output ports (Port 9)
- One 8-bit input ports (AIN0 – AIN7 / P8.0 – P8.7, Port 8)

Each port can be easily configured by software to meet the various system configuration and design requirement. Users should define the functionality of port before the start of main program. If users does not want to use the multiplexed pin functionality pin, these pin can be configured as simple I/O port. For example, the port 8 can be used as analog input for ADC module or as general input port pins.

Table 7-1. S3C3410X Port Configuration Overview

Port	Configuration Options	Recommend
0	**General I/O port with pull-up resistor:** P0.0, P0.1, P0.2, P0.3 and P0.4 can alternately serve as external capture input or clock input for Timer0, 1, 2, 3 and 4 respectively. P0.5 and P0.6 can alternately serve as PWM or Toggle Out for a Timer3 and 4 respectively. P0.7 can be used as external interrupt input EINT0 or external port write strobe signal(nWREXP).	Bit Programmable
1	**General I/O port:** P1.4 – P1.7 can alternately serve as external interrupt inputs of EINT4 – EINT7, or can be configured as address line of A20 – A23 for external interface. P1.0 – P1.3 can alternately as address line of A16 – A19 for external interface.	Bit Programmable
2	**General I/O port:** P2.0 – P2.6 can be used alternately as chip select signal lines for the external interface. P2.7 can be used as external interrupt input EINT1 or chip select strobes for the extra device(nECS0).	Bit Programmable
3	**General I/O port:** P3.0 – P3.6 can be used alternately as bus control signal lines for the external interface: nWBE0:nBE0:DQM0, nWBE1:nBE1:DQM1, nCAS0:nSRAS, nCAS1:nSCAS, SCKE, SCLK. P3.7 can be used alternately as external interrupt input EINT2 or chip select strobes for the extra device(nECS1).	Bit Programmable
4	**General I/O port:** P4.0 – P4.7 can be configured as data lines, D8 – D15 for the external interface or address line, A16 – A23.	Bit Programmable
5	**General I/O port:** P5.0 and P5.2 can be used alternately as external request input for DMA module: nDREQ0, nDREQ1. P5.1 and P5.3 can be used alternately as external acknowledge output for DMA module: nDACK0, nDACK1. P5.4 and P5.5 can be used alternately as serial data and serial clock for IIC module: IICSDA and IICSCK. P5.6 and P5.7 can be used alternately as input and output for UART module: URXD and UTXD.	Bit Programmable
6	**General I/O port:** P6.0 and P6.4 can be used alternately as serial data input pins for SIO module: SIORXD0 and SIORXD1. P6.2 and P6.6 can be used alternately as serial data output pins for SIO module: SIOTXD0 and SIOTXD1. P6.1 and P6.5 can be used alternately as external clock input/output for SIO module: SIOCLK0 and SIOCLK1. P6.7 can be used as external interrupt input EINT3.	Bit Programmable
7	**General I/O port:** can be used as a real time output by 8-bit or 4-bit unit. If TEST[1:0] bit is set to "10" or "11", P0.0 – P0.4 can be used as JTAG test port: nTCK (P7.0), TMS (P7.1), TDI (P7.2), nTRST (P7.3), TDO (P7.4).	Bit Programmable
8	Analog input channels AIN0 – AIN7, alternately general input port or external interrupt input EINT8(P8.4), EINT9(P8.5), EINT10(P8.6) and EINT11(P8.7).	Bit Programmable
9	**General Output Port:** P9.0 and P9.1 can be used alternately as LCD control signal, LP and DCLK	Bit Programmable

I/O PORT CONTROL REGISTER

PORT DATA REGISTER

All ten port data registers have the identical structure as shown in below:

Table 7-2. Port Data Register Summary

Register Name	Mnemonic	Offset	Reset Value	R/W
Port 0 Data Register	PDAT0	0xb000	0x0	R/W
Port 1 Data Register	PDAT1	0xb001	0x0	R/W
Port 2 Data Register	PDAT2	0xb002	0x0	R/W
Port 3 Data Register	PDAT3	0xb003	0x0	R/W
Port 4 Data Register	PDAT4	0xb004	0x0	R/W
Port 5 Data Register	PDAT5	0xb005	0x0	R/W
Port 6 Data Register	PDAT6	0xb006	0x0	R/W
Port 7 Data Register	PDAT7	0xb007	0x0	R/W
Port 8 Data Register	PDAT8	0xb008	0x0	R
Port 9 Data Register	PDAT9	0xb009	0x0	R/W
Port 7 Buffer Register	P7BR	0xb00b	0x0	R/W

PDATn	Bit	Description	Initial State
I/O Port n Data Register (n = 0 – 9, n = 0 – 7, 9 : R/W, n = 8 : R) : When the port is configured as input port, the port data will reflect the signal on the pin. When the port is configured as output port, the port data in Port Data Register will be given as output on the pin.			
Pn.0	[0]	Port n.0 port data bit (LSB)	0
Pn.1	[1]	Port n.1 port data bit	0
Pn.2	[2]	Port n.2 port data bit	0
Pn.3	[3]	Port n.3 port data bit	0
Pn.4	[4]	Port n.4 port data bit	0
Pn.5	[5]	Port n.5 port data bit	0
Pn.6	[6]	Port n.6 port data bit	0
Pn.7	[7]	Port n.7 port data bit (MSB)	0

Table 7-3. Port Control Register Summary

Register Name	Mnemonic	Offset	Reset Value	R/W
External Interrupt Pending Register	EINTPND	0xb031	0x0	R/W
External Interrupt Control Register	EINTCON	0xb032	0x0	R/W
External Interrupt Mode Register	EINTMOD	0xb034	0x0	R/W
Port 0 Control Register	PCON0	0xb010	0x0	R/W
Port 1 Control Register	PCON1	0xb012	0x0	R/W
Port 2 Control Register	PCON2	0xb014	0x0	R/W
Port 3 Control Register	PCON3	0xb016	0x0	R/W
Port 4 Control Register	PCON4	0xb018	0x0	R/W
Port 5 Control Register	PCON5	0xb01c	0x0	R/W
Port 6 Control Register	PCON6	0xb020	0x0	R/W
Port 7 Control Register	PCON7	0xb024	0x0	R/W
Port 8 Control Register	PCON8	0xb026	0x0	R/W
Port 9 Control Register	PCON9	0xb027	0x0	R/W
Port 0 Pull-up control Register	PUR0	0xb028	0x80	R/W
Port 1 Pull-down control Register	PDR1	0xb029	0xff	R/W
Port 2 Pull-up control Register	PUR2	0xb02a	0xff	R/W
Port 3 Pull-up control Register	PUR3	0xb02b	0xff	R/W
Port 4 Pull-down control Register	PDR4	0xb02c	0xff	R/W
Port 5 Pull-up control Register	PUR5	0xb02d	0x0	R/W
Port 6 Pull-up control Register	PUR6	0xb02e	0x0	R/W
Port 7 Pull-up control Register	PUR7	0xb02f	0x0	R/W
Port 8 Pull-up control Register	PUR8	0xb03c	0x0	R/W

PORT 0 CONTROL REGISTERS (PCON0, PUR0)

Register	Offset Address	R/W	Description	Reset Value
PCON0	0xb010	R/W	Configuration the pins of Port 0	0x0
PUR0	0xb028	R/W	pull-up disable resister for port 0	0x80

PCON0	Bit	Description	Initial State
P0.0	[0]	0 = Schmitt input mode, capture (TCAP0), or clock (TCLK0) input for Timer 0. If the timer mode is configured as capture mode or external timer clock mode, this pin will be used as TCAP0 or TCLK0. If the timer is disabled or use the internal clock as timer input clock, this pin will be configured as input port. 1 = C-MOS push-pull output	0
P0.1	[1]	0 = Schmitt input mode, capture (TCAP1), or clock (TCLK1) input for Timer 1. If the timer mode is configured as Capture mode or external timer clock mode, this pin will be used as TCAP1 or TCLK1. If the timer is disabled or use the internal clock as timer input clock, this pin will be configured as input port. 1 = C-MOS push-pull output	0
P0.2	[2]	0 = Schmitt input mode, capture (TCAP2), or clock (TCLK2) input for Timer 2. If the timer mode is configured as Capture mode or external timer clock mode, this pin will be used as TCAP2 or TCLK2. If the timer is disabled or use the internal clock as timer input clock, this pin will be configured as input port. 1 = C-MOS push-pull output	0
P0.3	[3]	0 = Schmitt input mode or clock (TCLK3) input for Timer 3. If the timer mode is configured as external timer clock mode, this pin will be used as TCLK3. If the timer is disabled or use the internal clock as timer input clock, this pin will be configured as input port. 1 = C-MOS push-pull output	0
P0.4	[4]	0 = Schmitt input mode or clock (TCLK4) input for Timer 4. If the timer mode is configured as external timer clock mode, this pin will be used as TCLK4. If the timer is disabled or use the internal clock as timer input clock, this pin will be configured as input port. 1 = C-MOS push-pull output	0

PCON0	Bit	Description	Initial State
P0.5	[6:5]	00 = Schmitt input mode or capture (TCAP3) input for Timer 3. If the timer mode is configured as Capture mode, this pin will be used as TCAP3. 01 = C-MOS push-pull output 10 = C-MOS push-pull PWM0/TOUT3 output for Timer 3	00
P0.6	[8:7]	00 = Schmitt input mode or capture (TCAP4) input for Timer 4. If the timer mode is configured as Capture mode, this pin will be used as TCAP4. 01 = C-MOS push-pull output 10 = C-MOS push-pull PWM1/TOUT4 output for Timer 4	00
P0.7	[10:9]	00 = Schmitt input mode or external interrupt input (EINT0). If the EINT0 is enabled, this pin will be used as external interrupt request pin. Otherwise, this pin will be used as input port pin. 01 = C-MOS push-pull output 10 = C-MOS push-pull nWREXP output	00

PUR0	Bit	Description	Initial State
P0	[7:0]	Setting the corresponding pull-up resistor of Port 0 0 = Disable pull-up resister 1 = Enable pull-up resister	0x80

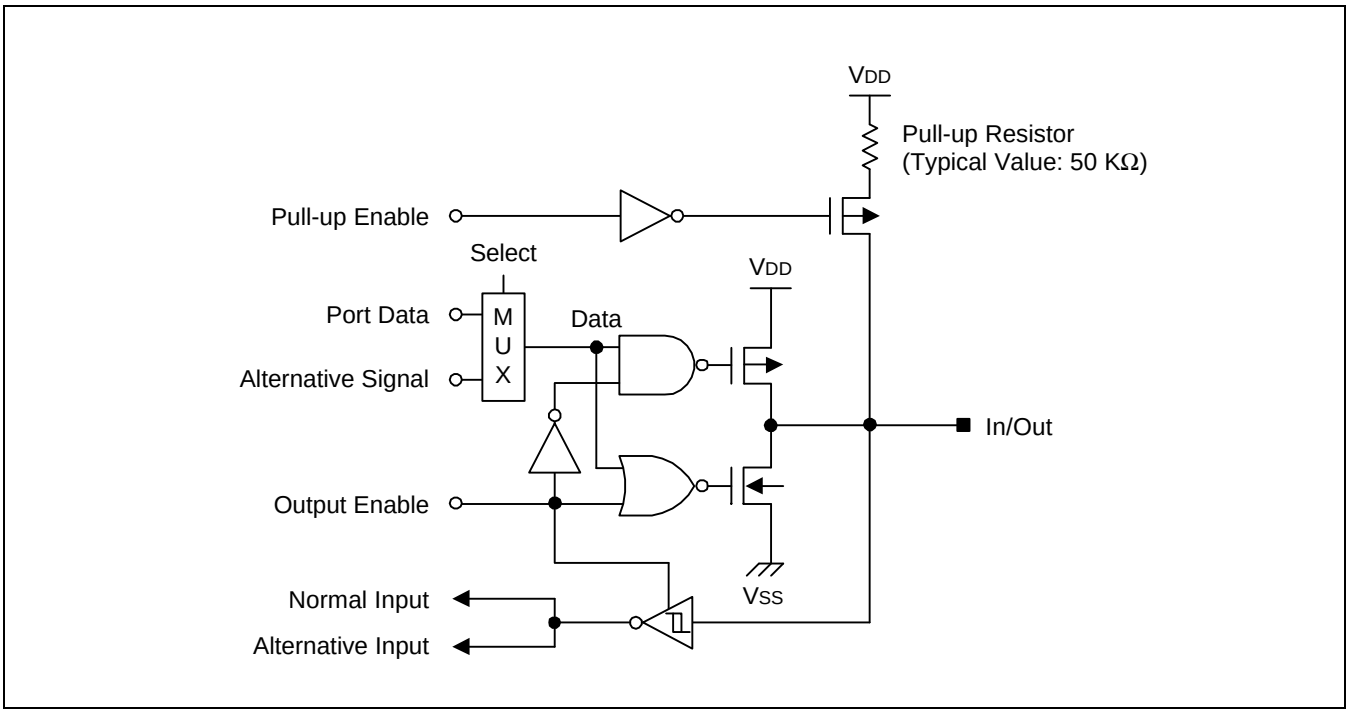


Figure 7-1. Pin Circuit Type 0 (P0.0 – P0.6)

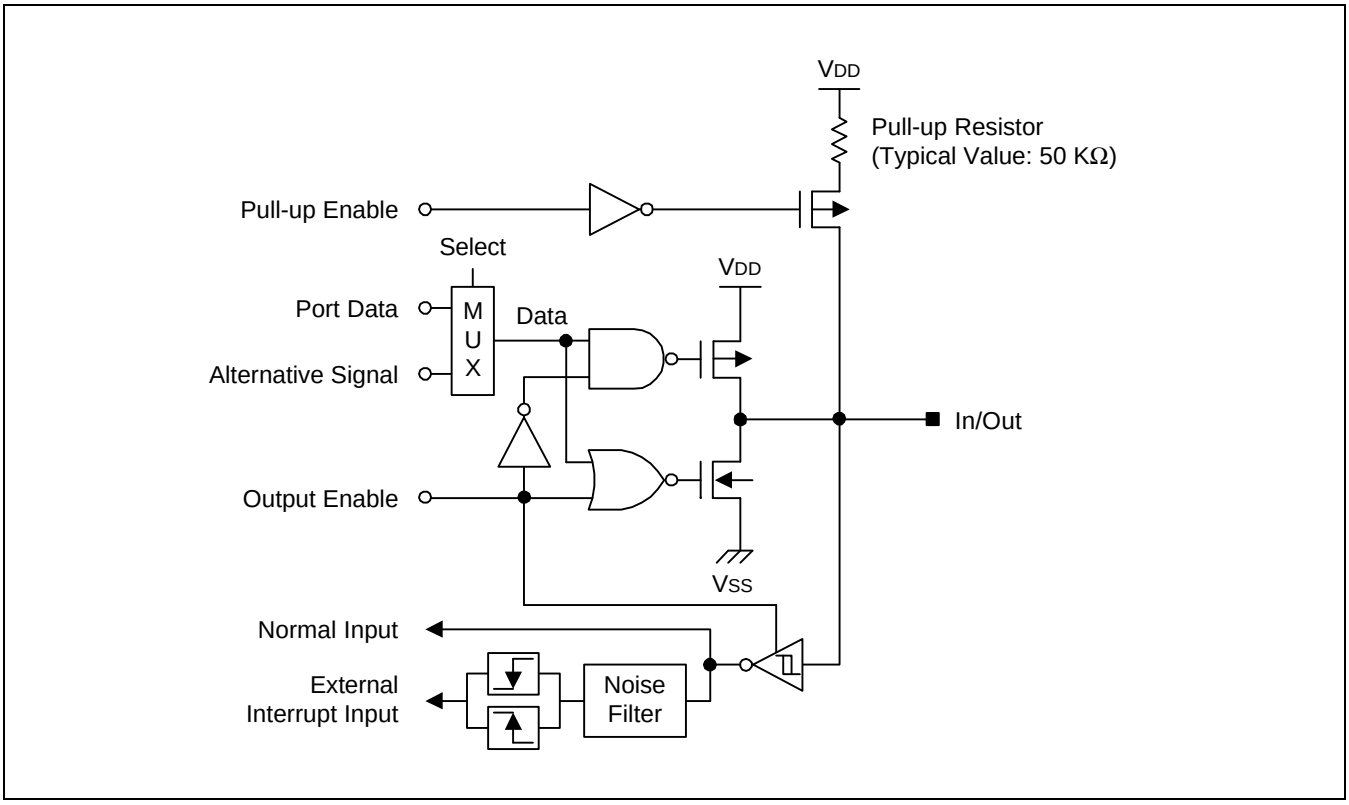


Figure 7-2. Pin Circuit Type 0-1 (P0.7, P2.7, P3.7, P6.7)

PORT 1 CONTROL REGISTERS (PCON1, PDR1)

Register	Offset Address	R/W	Description	Reset Value
PCON1	0xb012	R/W	Configuration the pins of Port 1	0x0
PDR1	0xb029	R/W	pull-down disable resister for port 1	0xff

PCON1	Bit	Description	Initial State
P1.0	[0]	**Setting the corresponding bit of Port 1** 0 = Schmitt input mode 1 = C-MOS push-pull output mode	0
P1.1	[1]	**Setting the corresponding bit of Port 1** 0 = Schmitt input mode 1 = C-MOS push-pull output mode	0
P1.2	[2]	**Setting the corresponding bit of Port 1** 0 = Schmitt input mode 1 = C-MOS push-pull output mode	0
P1.3	[3]	**Setting the corresponding bit of Port 1** 0 = Schmitt input mode 1 = C-MOS push-pull output mode	0
P1.4	[4]	**Setting the corresponding bit of Port 1** 0 = Input or External interrupt input(EINT4). If the EINT4 is enabled, this pin will be used as external interrupt request pin. Otherwise, this pin will be used as input port pin. 1 = C-MOS push-pull output mode	0
P1.5	[5]	**Setting the corresponding bit of Port 1** 0 = Input or External interrupt input(EINT5). If the EINT5 is enabled, this pin will be used as external interrupt request pin. Otherwise, this pin will be used as input port pin. 1 = C-MOS push-pull output mode	0
P1.6	[6]	**Setting the corresponding bit of Port 1** 0 = Input or External interrupt input(EINT6). If the EINT6 is enabled, this pin will be used as external interrupt request pin. Otherwise, this pin will be used as input port pin. 1 = C-MOS push-pull output mode	0
P1.7	[7]	**Setting the corresponding bit of Port 1** 0 = Input or External interrupt input(EINT7). If the EINT7 is enabled, this pin will be used as external interrupt request pin. Otherwise, this pin will be used as input port pin. 1 = C-MOS push-pull output mode	0
MP1	[15:8]	**Setting the Port 1 mode** 0 = Normal input/output mode (P1.0 – P1.7 sets the its corresponding bit of Port 1) 1 = Address bus line mode (Don't care of value P1.0 – P1.7). If this bit is 1, P1.0–P1.7 will be used as address of A16 – A23.	0x00

PDR1	Bit	Description	Initial State
P1	[7:0]	Setting the corresponding pull-down resistor of Port 1 0 = Disable pull-down resister 1 = Enable pull-down resister	0xff

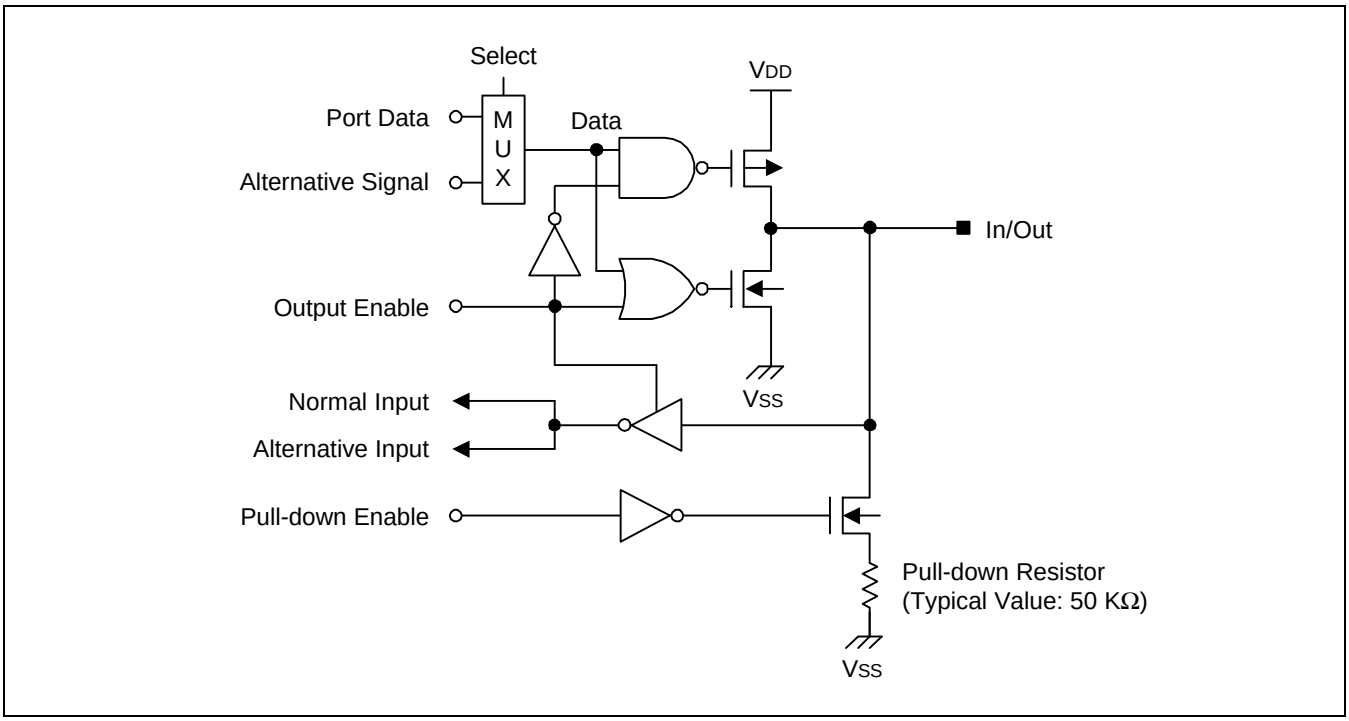


Figure 7-3. Pin Circuit Type 1-1 (Port1.0 – Port1.3)

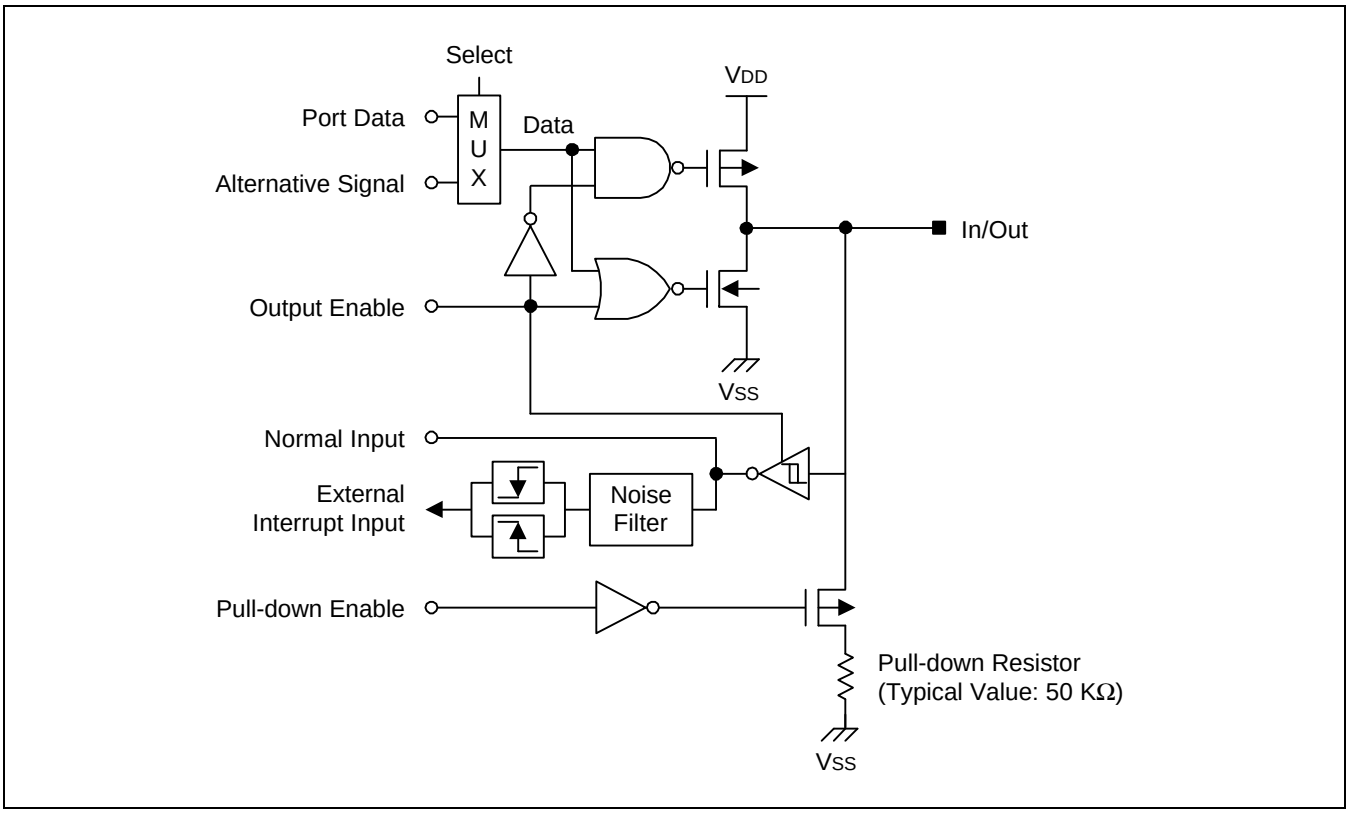


Figure 7-4. Pin Circuit Type 1-2 (Port1.4 – Port1.7)

PORT 2 CONTROL REGISTERS (PCON2, PUR2)

Register	Offset Address	R/W	Description	Reset Value
PCON2	0xb014	R/W	Configuration the pins of Port 2	0x0
PUR2	0xb02a	R/W	pull-up disable resister for port 2	0xff

PCON2	Bit	Description	Initial State
P2.0	[1:0]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Chip select signal(nCS1) output for the external interface	00
P2.1	[3:2]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Chip select signal(nCS2) output for the external interface	00
P2.2	[5:4]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Chip select signal(nCS3) output for the external interface	00
P2.3	[7:6]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Chip select signal(nCS4) output for the external interface	00
P2.4	[9:8]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Chip select signal(nCS5) output for the external interface	00
P2.5	[11:10]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Chip select signal(nCS6) or Row address strobe signal(nRAS0) for DRAM or Chip select signal(nSCS0) for SDRAM. It depends on the configuration on bank 6. If this bank is configured as ROM/SRAM, EDO DRAM, or SDRAM, this signal will behavior as nCS6, nRAS0, or nSCS0.	00
P2.6	[13:12]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Chip select signal(nCS7) or Row address strobe signal(nRAS1) for DRAM or Chip select signal(nSCS1) for SDRAM. It depends on the configuration on bank 7. If this bank is configured as ROM/SRAM, EDO DRAM, or SDRAM, this signal will behavior as nCS7, nRAS0, or nSCS0.	00
P2.7	[15:14]	00 = Input or external interrupt input(EINT1) 01 = C-MOS push-pull output mode 10 = Extra Chip select signal(nECS0) output for the external interface	00

PUR2	Bit	Description	Initial State
P2	[7:0]	Setting the corresponding pull-up resistor of Port 2 0 = Disable pull-up resister 1 = Enable pull-up resister	0xff

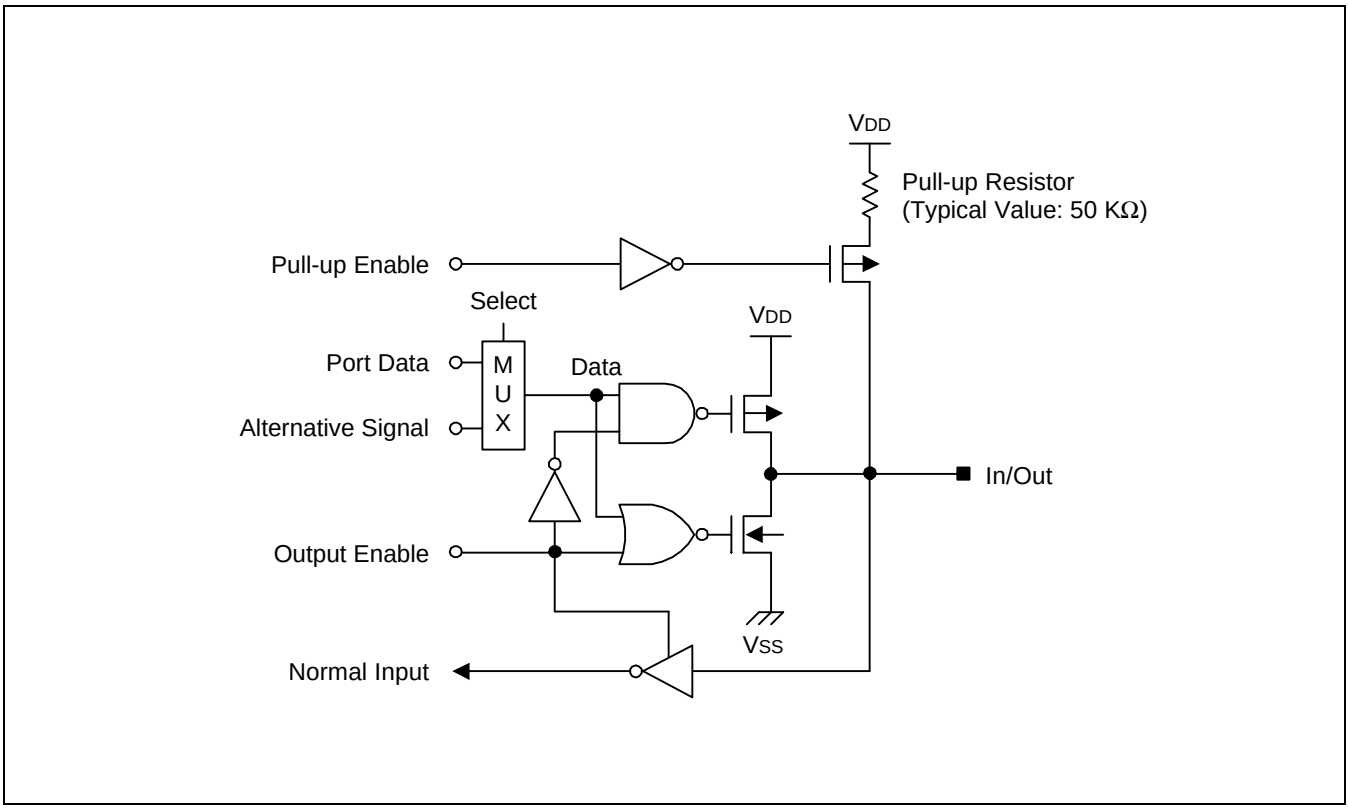


Figure 7-5. Pin Circuit Type 2-1 (P2.0 – P2.6)

PORT 3 CONTROL REGISTERS (PCON3, PUR3)

Register	Offset Address	R/W	Description	Reset Value
PCON3	0xb016	R/W	Configuration the pins of port 3	0x0
PUR3	0xb02b	R/W	pull-up disable resister for port 3	0xff

PCON3	Bit	Description	Initial State
P3.0	[1:0]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Write byte enable(nWBE0) output for external interface or Data Mask(DQM0) output for SDRAM or Byte enable(nBE0) output for x16 SRAM. If the memory bank is configured as SDRAM bank, this port will behavior as Data Mask(DQM0). If the memory bank is configured as x16 SRAM bank, this port will behavior as Byte enable(nBE0). Otherwise, this port will behavior as nWBE0.	00
P3.1	[3:2]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Write byte enable(nWBE1) output for external interface or Data Mask(DQM1) output for SDRAM or Byte enable(nBE0) output for x16 SRAM. If the memory bank is configured as SDRAM bank, this port will behavior as Data Mask(DQM1). If the memory bank is configured as x16 SRAM bank, this port will behavior as Byte enable(nBE0). Otherwise, this port will behavior as nWBE1.	00
P3.2	[5:4]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Column address strobe(nCAS0) output for DRAM or Row address strobe(nSRAS) output for SDRAM. If the memory bank is configured as EDO DRAM. This port will behavior as nCAS0. If the memory bank is configured as SDRAM bank, this port will behavior as nSRAS.	00
P3.3	[7:6]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Column address strobe(nCAS1) output for DRAM or Row address strobe(nSCAS) output for SDRAM. This port will behavior as nCAS1. If the memory bank is configured as SDRAM bank, this port will behavior as nSRAS.	00
P3.4	[9:8]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Write enable(nWE) output for 16-bit SRAM or SDRAM	00
P3.5	[11:10]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Clock Enable(SCKE) output for SDRAM	00
P3.6	[13:12]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Clock output for SDRAM	00
P3.7	[15:14]	00 = C-MOS input mode or external interrupt(EINT2) mode 01 = C-MOS push-pull output mode 10 = Extra chip select(nECS1) output	00

PUR3	Bit	Description	Initial State
P3	[7:0]	Setting the corresponding pull-up resistor of Port 3 0 = Disable pull-up resister 1 = Enable pull-up resister	0xff

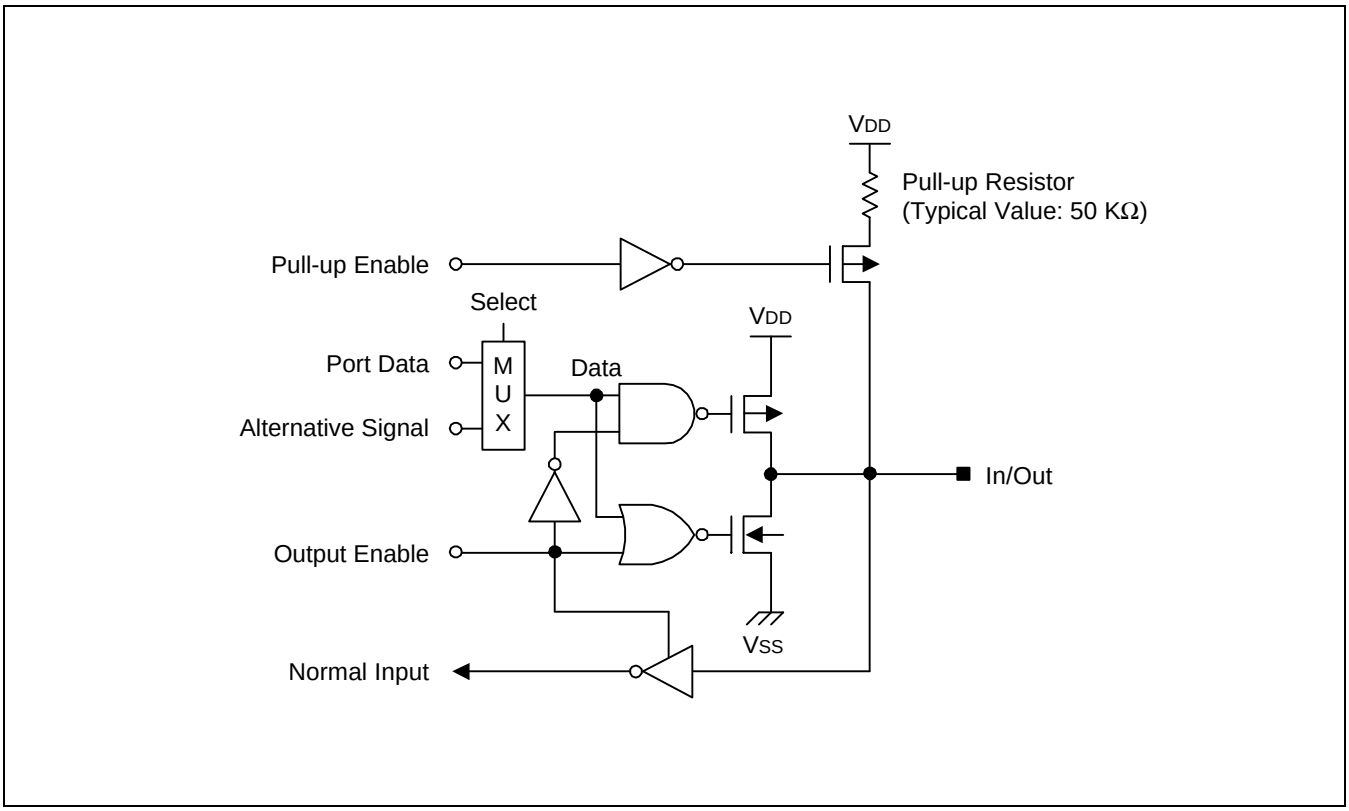


Figure 7-6. Pin Circuit Type 3-1 (P3.0 – P3.6)

PORT 4 CONTROL REGISTERS (PCON4, PDR4)

Register	Offset Address	R/W	Description	Reset Value
PCON4	0xb018	R/W	Configuration the pins of Port 4	0x0
PDR4	0xb02c	R/W	pull-down disable resister for port 4	0xff

PCON4	Bit	Description	Initial State
P4.0	[1:0]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = A16 11 = D8	00
P4.1	[3:2]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = A17 11 = D9	00
P4.2	[5:4]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = A18 11 = D10	00
P4.3	[7:6]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = A19 11 = D11	00
P4.4	[9:8]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = A20 11 = D12	00
P4.5	[11:10]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = A21 11 = D13	00
P4.6	[13:12]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = A22 11 = D14	00
P4.7	[15:14]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = A23 11 = D15	00

PDR4	Bit	Description	Initial State
P4	[7:0]	Setting the corresponding pull-down resistor of Port 4 0 = Disable pull-down resister 1 = Enable pull-down resister	0xff

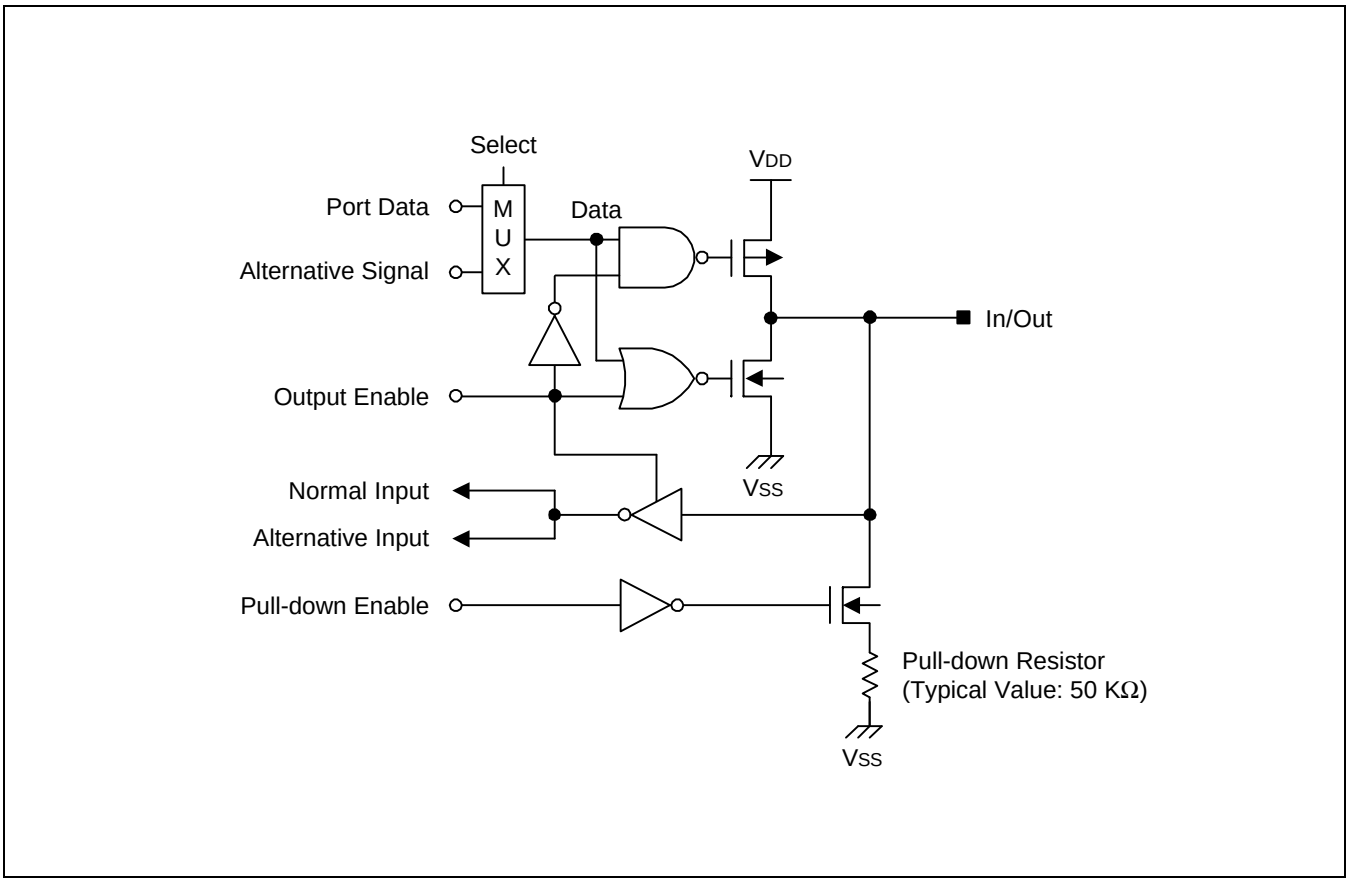


Figure 7-7. Pin Circuit Type 4 (Port 4)

PORT 5 CONTROL REGISTERS (PCON5, PUR5)

Register	Offset Address	R/W	Description	Reset Value
PCON5	0xb01c	R/W	Configuration the pins of Port 5	0x0
PUR5	0xb02d	R/W	pull-up disable resister for port 5	0x0

PCON5	Bit	Description	Initial State
P5.0	[1:0]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = External DMA Request input (nDREQ0)	00
P5.1	[3:2]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = External DMA Acknowledge output (nDACK0)	00
P5.2	[5:4]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = External DMA Request input (nDREQ1)	00
P5.3	[7:6]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = External DMA Acknowledge output (nDACK1)	00
P5.4	[9:8]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Serial data line, SDA for IIC interface (Open-drain type)	00
P5.5	[11:10]	00 = C-MOS input mode 01 = C-MOS push-pull output mode 10 = Serial clock line, SCK for IIC interface (Open-drain type)	00
P5.6	[13:12]	00 = Schmitt input mode or serial input(URXD) for UART. For the case of URXD, the UART should be enabled. Otherwise, this bit will be input port. 01 = C-MOS push-pull output mode 10 = N-ch open-drain output mode	00
P5.7	[16:14]	000 = Schmitt input mode 001 = C-MOS push-pull output mode 011 = N-ch open-drain output mode 101 = C-MOS push-pull serial output(UTXD) for UART 111 = N-ch open-drain serial output(UTXD) for UART	000

PUR5	Bit	Description	Initial State
P5	[7:0]	Setting the corresponding pull-up resistor of Port 5 0 = Disable pull-up resister 1 = Enable pull-up resister	0x0

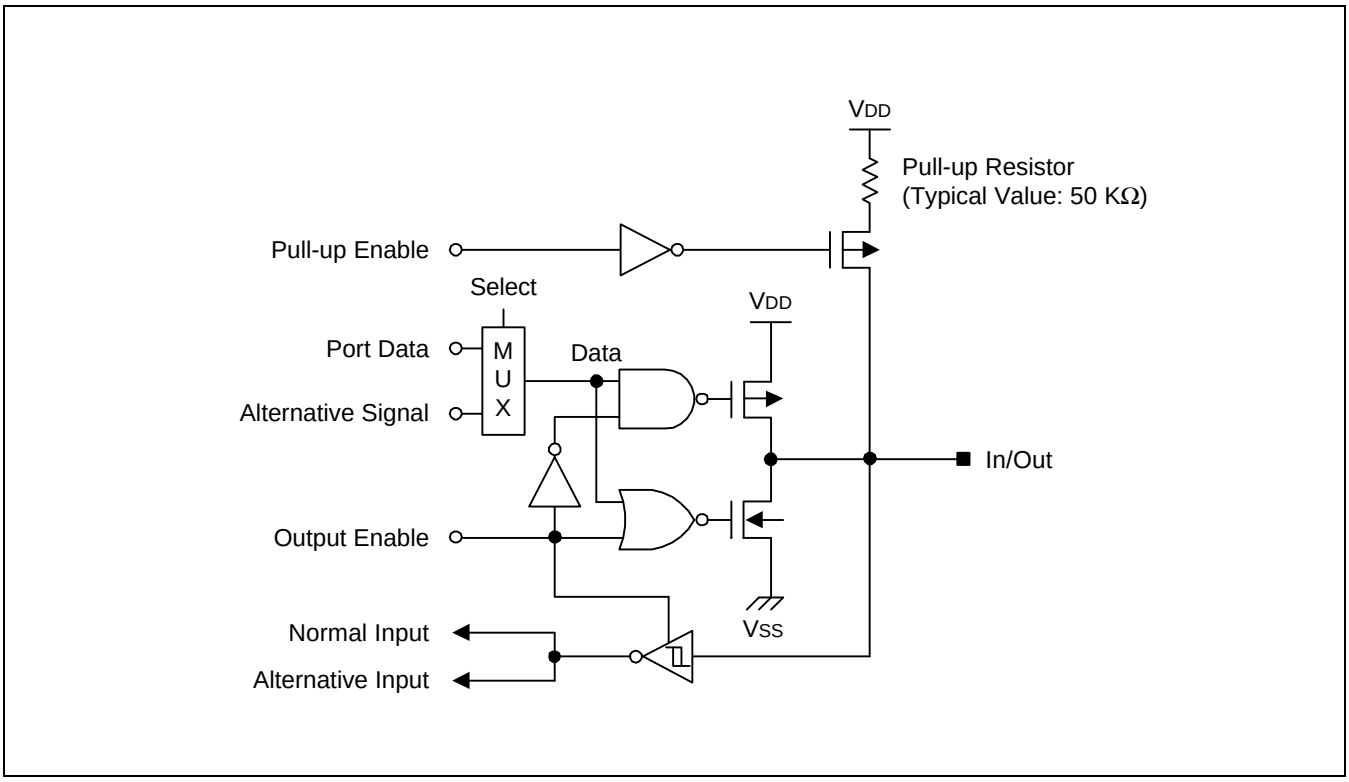


Figure 7-8. Pin Circuit Type 5-1 (P5.0 – P5.5)

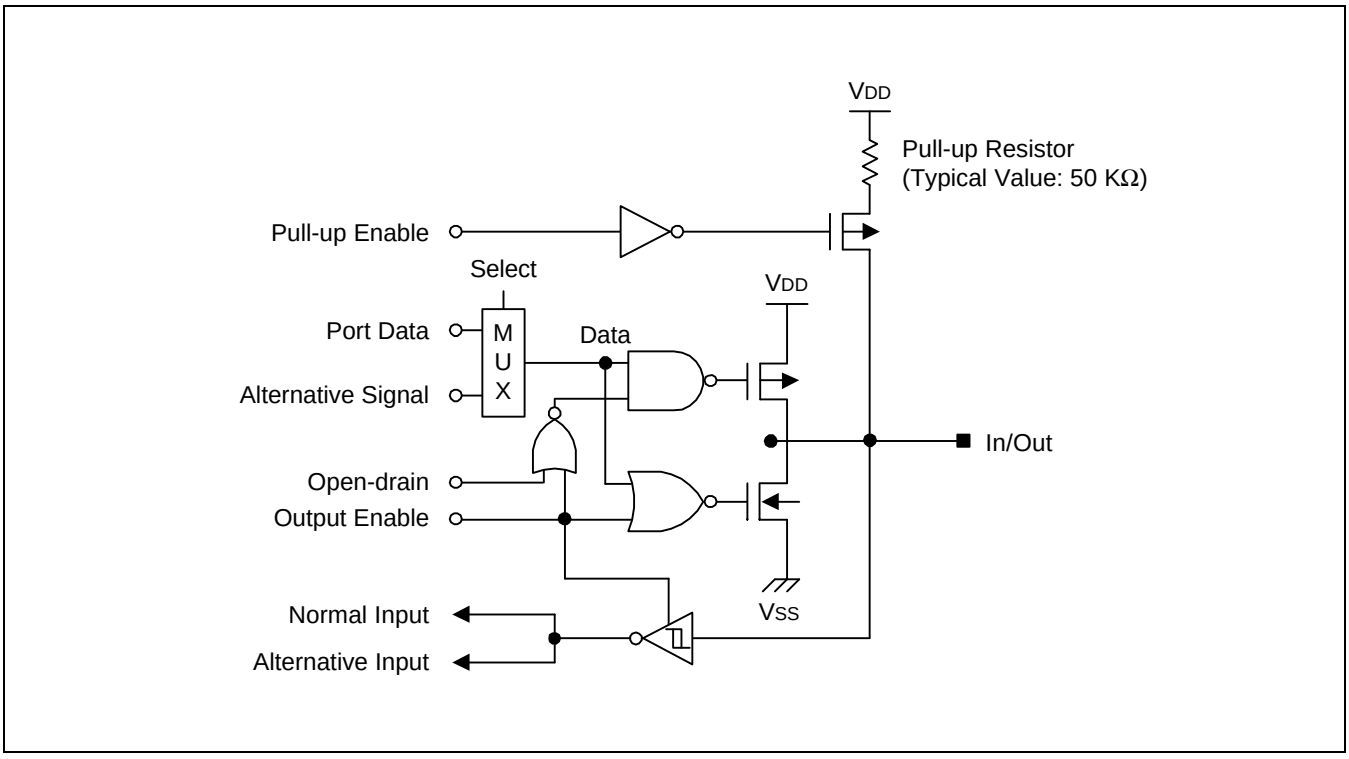


Figure 7-9. Pin Circuit Type 5-9 (P5.6, P5.7, P6.0 – P6.6)

PORT 6 CONTROL REGISTERS (PCON6, PUR6)

Register	Offset Address	R/W	Description	Reset Value
PCON6	0xb020	R/W	Configuration the pins of Port 6	0x0
PUR6	0xb02e	R/W	pull-up disable resister for port 6	0x0

PCON6	Bit	Description	Initial State
P6.0	[1:0]	00 = Schmitt input or serial data input(SIORXD0) mode for SIO. For the case of SIORXD0, the SIO 0 should be enabled. Otherwise, this bit will be input port. 01 = C-MOS push-pull output mode 10 = N-ch open-drain output mode	00
P6.1	[4:2]	000 = Schmitt input or serial clock input(SIOCLK0) mode for SIO For the case of SIOCLK0, the SIO 0 should be enabled. Otherwise, this bit will be input port. 001 = C-MOS push-pull output mode 010 = N-ch open-drain output mode 101 = C-MOS push-pull serial clock output (SIOCLK0) for SIO 0 110 = N-ch open-drain serial clock output (SIOCLK0) for SIO 0	000
P6.2	[7:5]	000 = Schmitt input mode 001 = C-MOS push-pull output mode 010 = N-ch open-drain output mode 101 = C-MOS push-pull serial data output (SIOTXD0) for SIO 0 110 = N-ch open-drain serial data output (SIOTXD0) for SIO 0	000
P6.3	[9:8]	00 = Schmitt input mode 01 = C-MOS push-pull output mode 10 = Wait signal(nWAIT) input for the external interface 11 = Ready signal(nSIORDY) input or output for SIO0,1	00
P6.4	[11:10]	00 = Schmitt input or serial data input (SIORXD1) mode for SIO. For the case of SIORXD1, the SIO 1 should be enabled. Otherwise, this bit will be input port. 01 = C-MOS push-pull output mode 10 = N-ch open-drain output mode	00
P6.5	[14:12]	000 = Schmitt input or serial clock input (SIOCLK1) mode for SIO For the case of SIOCLK1, the SIO 1 should be enabled. Otherwise, this bit will be input port. 001 = C-MOS push-pull output mode 010 = N-ch open-drain output mode 101 = C-MOS push-pull serial clock output (SIOCLK1) for SIO 1 110 = N-ch open-drain serial clock output(SIOCLK1) for SIO 1	000
P6.6	[17:15]	000 = Schmitt input mode 001 = C-MOS push-pull output mode 010 = N-ch open-drain output mode 101 = C-MOS push-pull serial data output (SIOTXD1) for SIO 1 110 = N-ch open-drain serial data output (SIOTXD1) for SIO 1	000
P6.7	[18]	0 = Schmitt input mode or external interrupt (EINT3) input mode. For the case of EINT3, the EINT3 should be enabled. Otherwise, this bit will be input port. 1 = C-MOS push-pull output mode	0

PUR6	Bit	Description	Initial State
P6	[7:0]	Setting the corresponding pull-up resistor of Port 6 0 = Disable pull-up resister 1 = Enable pull-up resister	0x0

PORT 7 CONTROL REGISTERS (PCON7, PUR7)

Register	Offset Address	R/W	Description	Reset Value
PCON7	0xb024	R/W	Configuration the pins of Port 7	0x0
PUR7	0xb02f	R/W	pull-up disable resister for port 7	0x0
P7BR	0xb00b	R/W	Buffer register for storing real time output data	0x0

PCON7	Bit	Description	Initial State
P7.0 (RP0)	[0]	0 = C-MOS input mode 1 = C-MOS push-pull output mode	0
P7.1 (RP1)	[1]	0 = C-MOS input mode 1 = C-MOS push-pull output mode	0
P7.2 (RP2)	[2]	0 = C-MOS input mode 1 = C-MOS push-pull output mode	0
P7.3 (RP3)	[3]	0 = C-MOS input mode 1 = C-MOS push-pull output mode	0
P7.4 (RP4)	[4]	0 = C-MOS input mode 1 = C-MOS push-pull output mode	0
P7.5 (RP5)	[5]	0 = C-MOS input mode 1 = C-MOS push-pull output mode	0
P7.6 (RP6)	[6]	0 = C-MOS input mode 1 = C-MOS push-pull output mode	0
P7.7 (RP7)	[7]	0 = C-MOS input mode 1 = C-MOS push-pull output mode	0
RTO	[9:8]	Setting port 7 as real time output 00 = General I/O port 01 = Low nibble real time output buffer mode 10 = High nibble real time output buffer mode 11 = Byte real time output buffer mode	00
LNS	[10]	Time source of Low nibble real time output 0 = T0 1 = T3	0
HNS	[11]	Time source of High nibble real time output 0 = T0 1 = T3	0

PUR7	Bit	Description	Initial State
P7	[7:0]	Setting the corresponding pull-up resistor of Port 7 0 = Disable pull-up resister 1 = Enable pull-up resister	0x0

NOTE: Port 7 can be used for the realtime buffer output port. The realtime buffer is that P7BR data are output to RP[7:0], when timerD or timer3 match interrupt occurs. At this time, P7 must be configured to C-MOS push-pull output mode.

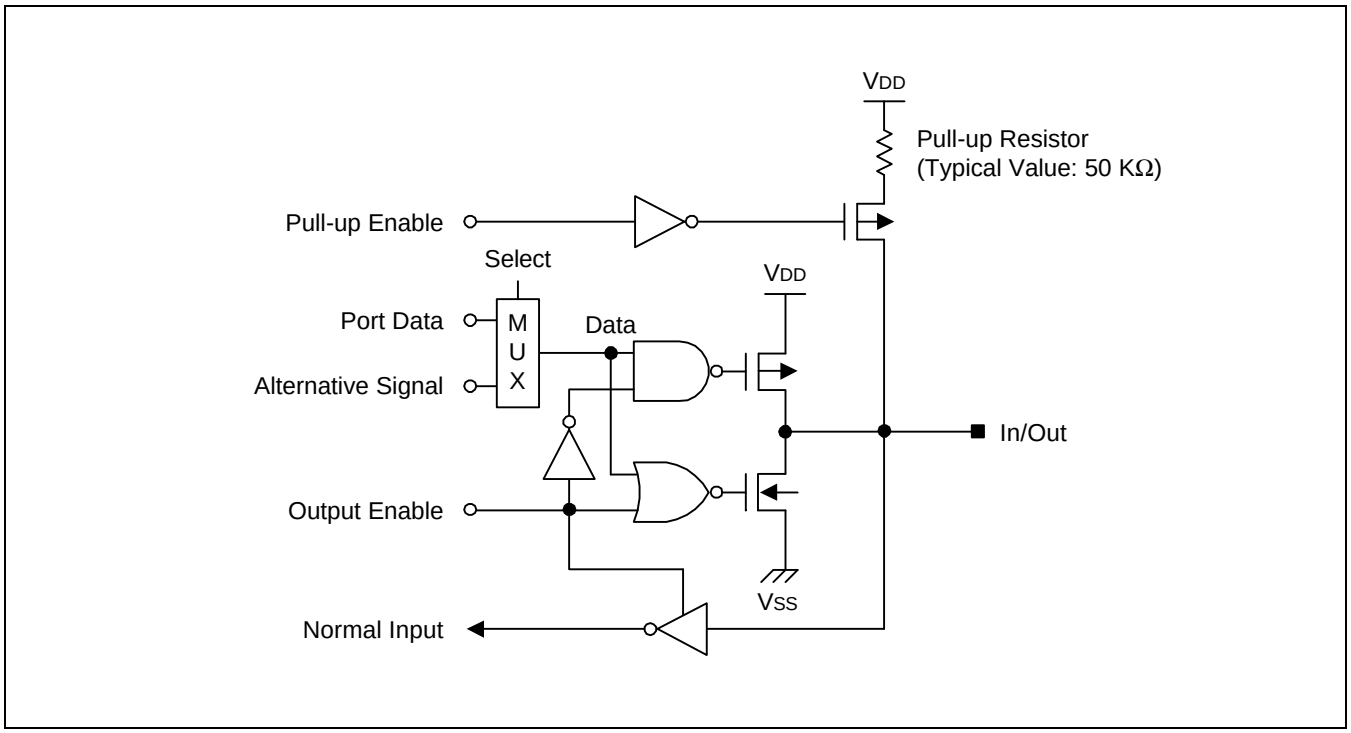


Figure 7-10. Pin Circuit Type 7 (P7.0 – P7.7)

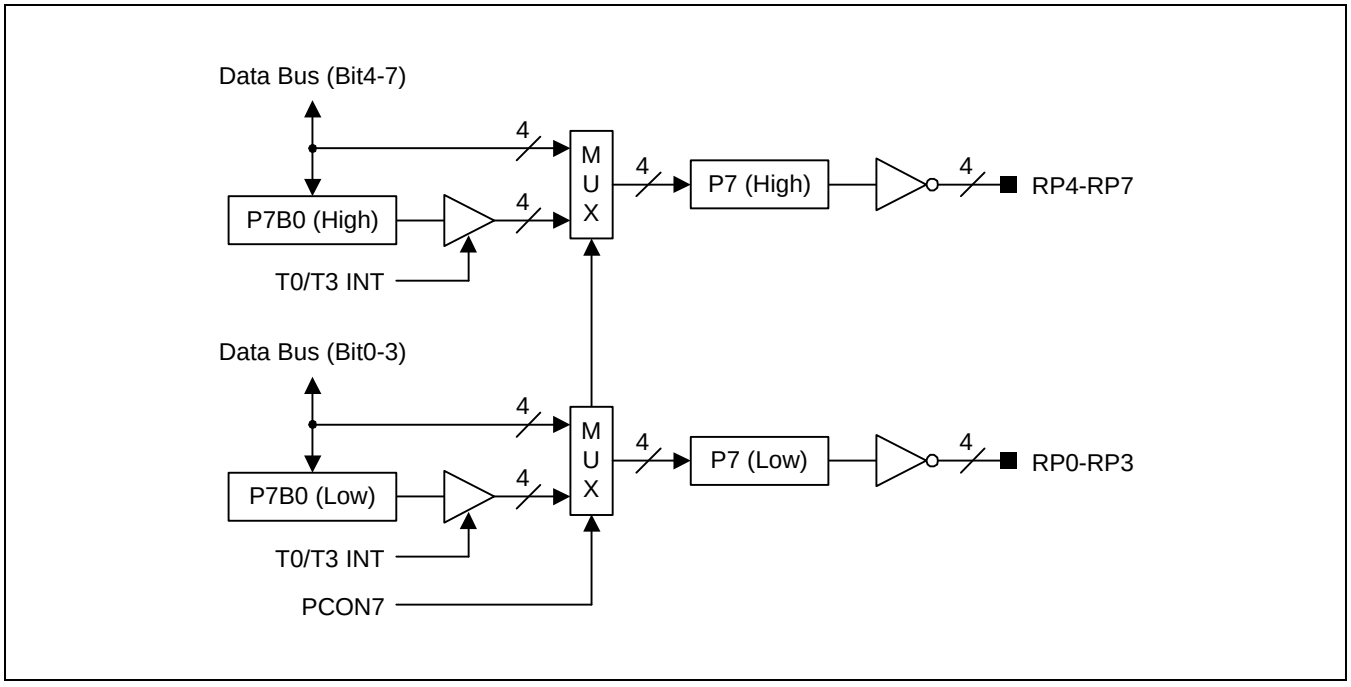


Figure 7-11. Port 7 (Real Time Output)

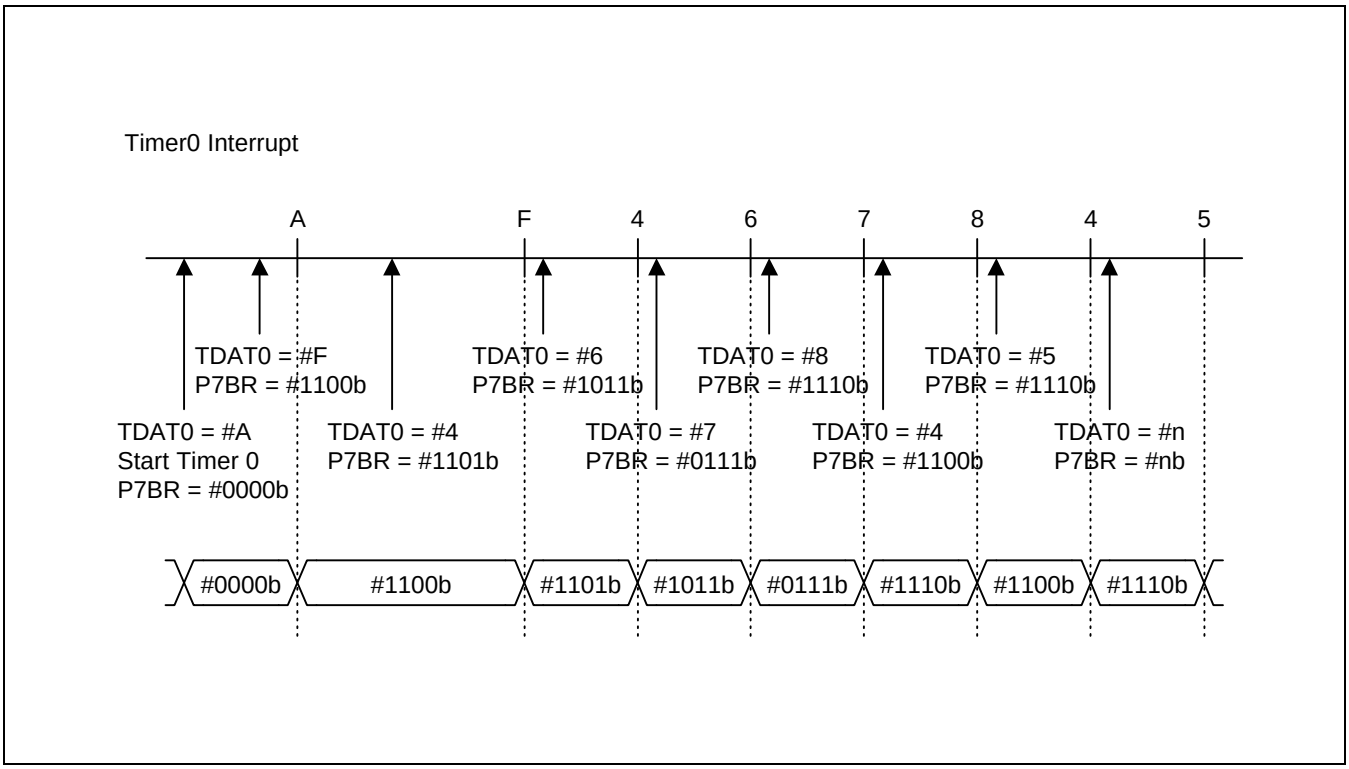


Figure 7-12. Real Time Output Example

PORT 8 CONTROL REGISTERS (PCON8, PUR8)

Register	Offset Address	R/W	Description	Reset Value
PCON8	0xb026	R/W	Configuration the pins of port 8	0x0
PUR8	0xb03c	R/W	Pull-up disable resister for port 8	0x0

PCON8	Bit	Description	Initial State
P8.0	[0]	0 = C-MOS input mode 1 = AIN0	0
P8.1	[1]	0 = C-MOS input mode 1 = AIN1	0
P8.2	[2]	0 = C-MOS input mode 1 = AIN2	0
P8.3	[3]	0 = C-MOS input mode 1 = AIN3	0
P8.4	[4]	0 = C-MOS input mode or External interrupt input mode. For the case of EINT8, the EINT8 should be enabled. Otherwise, this bit will be input port. 1 = AIN4	0
P8.5	[5]	0 = C-MOS input mode or External interrupt input mode. For the case of EINT9, the EINT9 should be enabled. Otherwise, this bit will be input port. 1 = AIN5	0
P8.6	[6]	0 = C-MOS input mode or External interrupt input mode. For the case of EINT10, the EINT10 should be enabled. Otherwise, this bit will be input port. 1 = AIN6	0
P8.7	[7]	0 = C-MOS input mode or External interrupt input mode. For the case of EINT11, the EINT11 should be enabled. Otherwise, this bit will be input port. 1 = AIN7	0

PUR8	Bit	Description	Initial State
P8	[7:0]	Setting the corresponding pull-up resistor of port 7 0 = Disable pull-up resister 1 = Enable pull-up resister	0x0

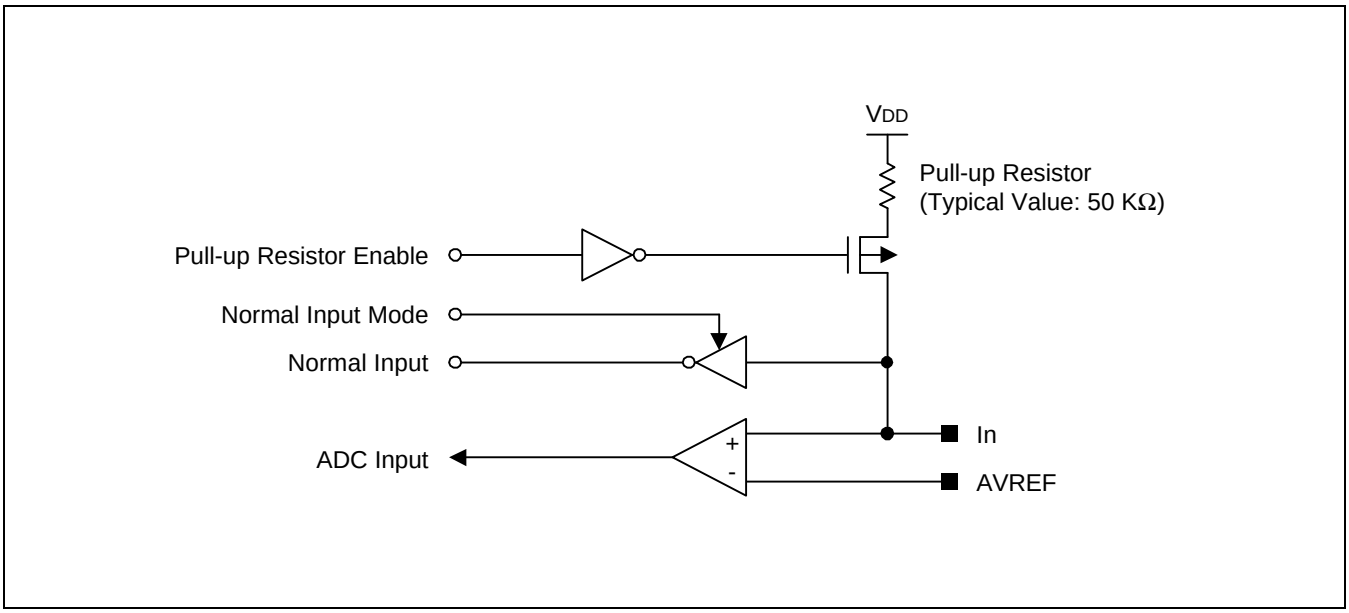


Figure 7-13. Pin Circuit Type 8-1 (P8.0 – P8.3)

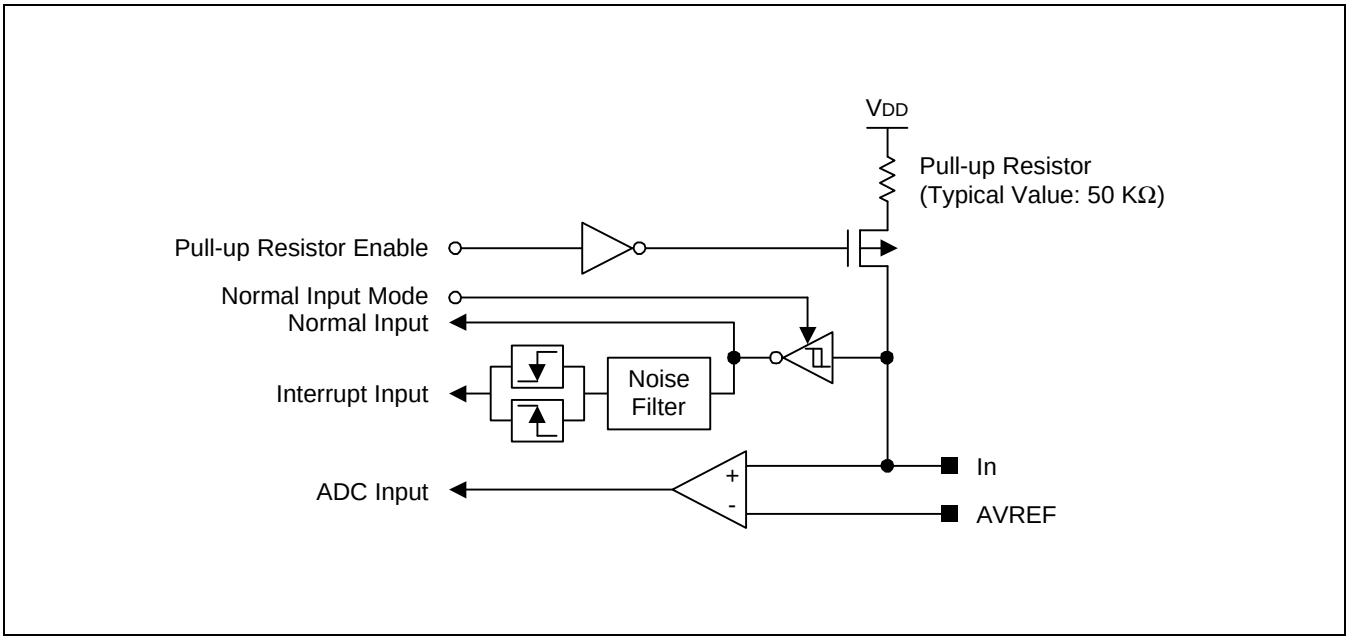


Figure 7-14. Pin Circuit Type 8-2 (P8.4 – P8.7)

PORT 9 CONTROL REGISTERS (PCON9)

Register	Offset Address	R/W	Description	Reset Value
PCON9	0xb027	R/W	Configuration the pins of Port 9	0x0

PCON8	Bit	Description	Initial State
P9.0	[0]	0 = LCD clock output mode (for LP) 1 = C-MOS push-pull output LP: When you write any data to EXTDAT0 or EXTDAT1 register, this signal is generated by the memory controller.	0
P9.1	[1]	0 = LCD line pulse output mode (for DCLK) 1 = C-MOS push-pull output DCLK: When you write any data to EXTPORT register by DMA, this signal is generated by the memory controller.	0

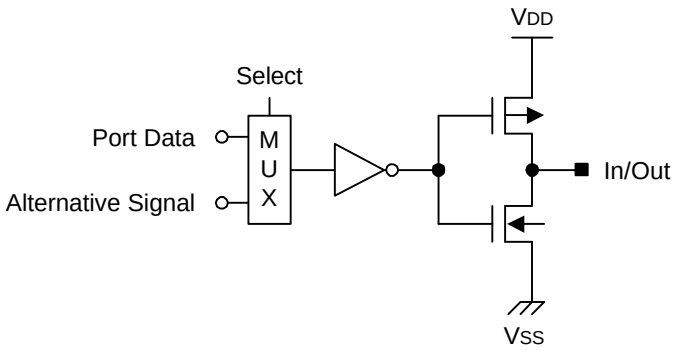


Figure 7-15. Pin Circuit Type 9 (P9.0, P9.1)

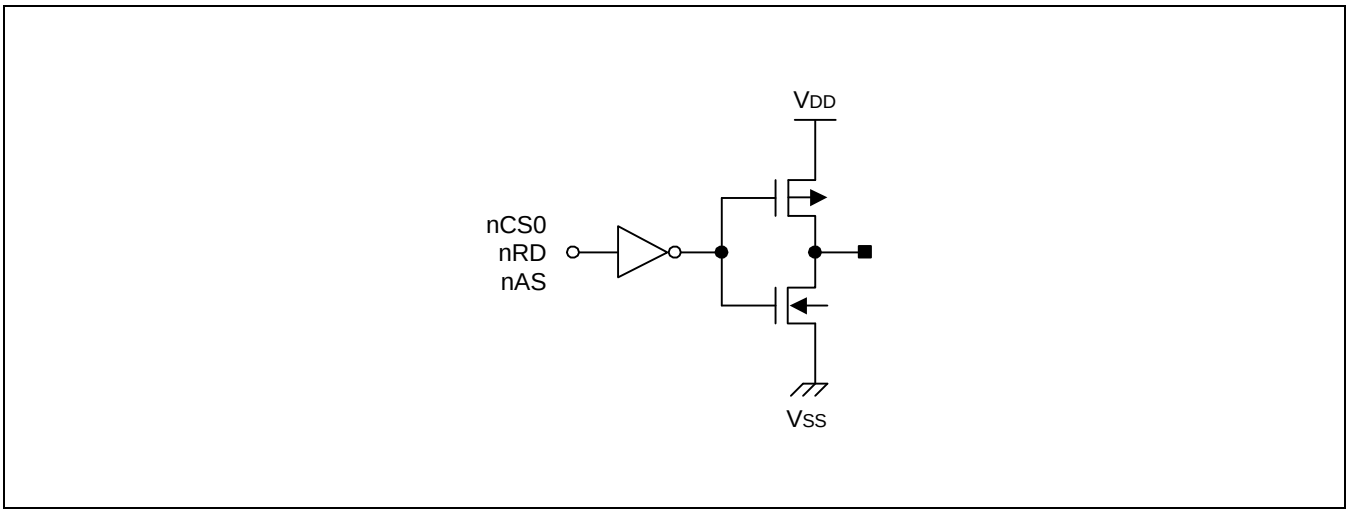


Figure 7-16. Pin Circuit Type 10 (nCS0, nRD and nAS)

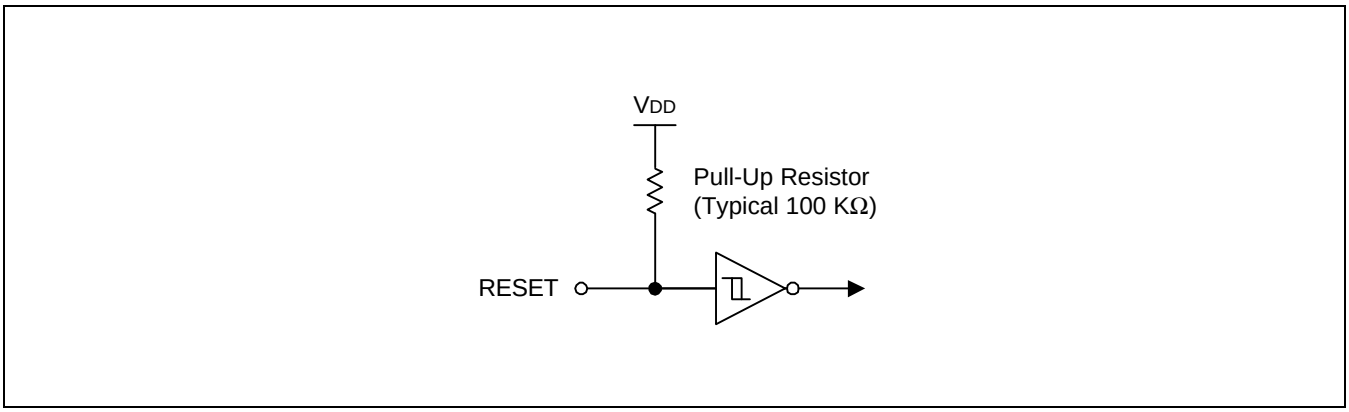


Figure 7-17. Pin Circuit Type 11 (RESET)

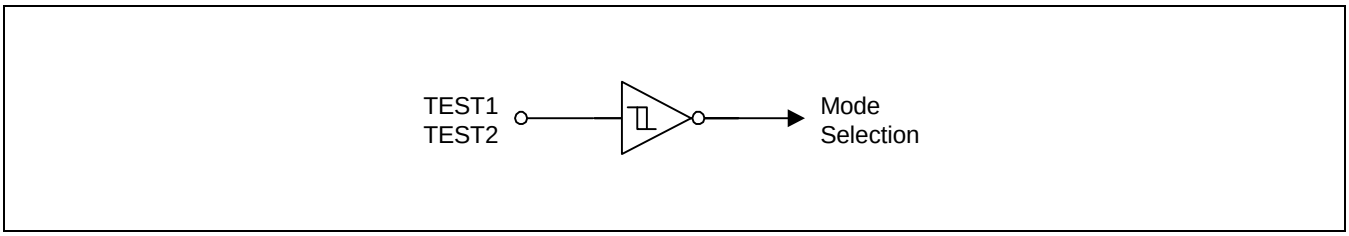


Figure 7-18. Pin Circuit Type 12 (TEST1, TEST2)

EXTERNAL INTERRUPT CONTROL REGISTERS (EINTPND, EINTCON, EINTMOD)

Register	Offset Address	R/W	Description	Reset Value
EINTPND	0xb031	R/W	External interrupt pending register	0x0
EINTCON	0xb032	R/W	External interrupt control register	0x0
EINTMOD	0xb034	R/W	External interrupt mode register	0x0

EINTPND	Bit	Description	Initial State
EINT4	[0]	0 = No interrupt pending *0 = Clear interrupt pending condition (when write)* 1 = External interrupt(EINT4) is pending	0
EINT5	[1]	0 = No interrupt pending *0 = Clear interrupt pending condition (when write)* 1 = External interrupt(EINT5) is pending	0
EINT6	[2]	0 = No interrupt pending *0 = Clear interrupt pending condition (when write)* 1 = External interrupt(EINT6) is pending	0
EINT7	[3]	0 = No interrupt pending *0 = Clear interrupt pending condition (when write)* 1 = External interrupt(EINT7) is pending	0

EINTCON	Bit	Description	Initial State
EINT0	[0]	Setting external interrupt enable of EINT0 0 = Disable external interrupt 1 = Enable external interrupt	0
EINT1	[1]	Setting external interrupt enable of EINT1 0 = Disable external interrupt 1 = Enable external interrupt	0
EINT2	[2]	Setting external interrupt enable of EINT2 0 = Disable external interrupt 1 = Enable external interrupt	0
EINT3	[3]	Setting external interrupt enable of EINT3 0 = Disable external interrupt 1 = Enable external interrupt	0
EINT4	[4]	Setting external interrupt enable of EINT4 0 = Disable external interrupt 1 = Enable external interrupt	0
EINT5	[5]	Setting external interrupt enable of EINT5 0 = Disable external interrupt 1 = Enable external interrupt	0
EINT6	[6]	Setting external interrupt enable of EINT6 0 = Disable external interrupt 1 = Enable external interrupt	0
EINT7	[7]	Setting external interrupt enable of EINT7 0 = Disable external interrupt 1 = Enable external interrupt	0
EINT8	[8]	Setting external interrupt enable of EINT8 0 = Disable external interrupt 1 = Enable external interrupt	0
EINT9	[9]	Setting external interrupt enable of EINT9 0 = Disable external interrupt 1 = Enable external interrupt	0
EINT10	[10]	Setting external interrupt enable of EINT10 0 = Disable external interrupt 1 = Enable external interrupt	0
EINT11	[11]	Setting external interrupt enable of EINT11 0 = Disable external interrupt 1 = Enable external interrupt	0

EINTMOD	Bit	Description	Initial State
EINT0	[2:0]	000 = Falling edge triggered 001 = Rising edge triggered 010 = High level interrupt 011 = Low level interrupt 100 = Both edge triggered	000
EINT1	[5:3]	000 = Falling edge triggered 001 = Rising edge triggered 010 = High level interrupt 011 = Low level interrupt 100 = Both edge triggered	000
EINT2	[8:6]	000 = Falling edge triggered 001 = Rising edge triggered 010 = High level interrupt 011 = Low level interrupt 100 = Both edge triggered	000
EINT3	[11:9]	000 = Falling edge triggered 001 = Rising edge triggered 010 = High level interrupt 011 = Low level interrupt 100 = Both edge triggered	000
EINT4	[14:12]	000 = Falling edge triggered 001 = Rising edge triggered 010 = High level interrupt 011 = Low level interrupt 100 = Both edge triggered	000
EINT5	[17:15]	000 = Falling edge triggered 001 = Rising edge triggered 010 = High level interrupt 011 = Low level interrupt 100 = Both edge triggered	000
EINT6	[20:18]	000 = Falling edge triggered 001 = Rising edge triggered 010 = High level interrupt 011 = Low level interrupt 100 = Both edge triggered	000
EINT7	[23:21]	000 = Falling edge triggered 001 = Rising edge triggered 010 = High level interrupt 011 = Low level interrupt 100 = Both edge triggered	000
EINT8	[25:24]	00 = Falling edge triggered 01 = Rising edge triggered 10 = High level interrupt 11 = Low level interrupt	00
EINT9	[27:26]	00 = Falling edge triggered 01 = Rising edge triggered 10 = High level interrupt 11 = Low level interrupt	00
EINT10	[29:28]	00 = Falling edge triggered 01 = Rising edge triggered 10 = High level interrupt 11 = Low level interrupt	00
EINT11	[31:30]	00 = Falling edge triggered 01 = Rising edge triggered 10 = High level interrupt 11 = Low level interrupt	00

NOTES:

1. Because each external interrupt pins has a 200ns noise filter
2. Because EINTPNDx bits are not cleared automatically, you have to clear this bit by writing "0". (Although these bits are not cleared, the interrupt triggering will operate.)

8 TIMER (16-BIT TIMERS & 8-BIT TIMERS)

OVERVIEW

The S3C3410X has three 16-bit timers (Timer0, Timer1, and Timer2) and two 8-bit timers (Timer3 and Timer4). The 16-bit timer can operate in interval mode, capture mode, match & overflow mode or DMA mode (Timer1 only). The 8-bit timer can operate in interval mode, capture mode, PWM (Pulse Width Modulation) mode or DMA mode (Timer3 only). The clock source for timer can be an internal or an external clock. Users can enable or disable the timer by setting control bits in the corresponding timer mode register.

The following list summarizes the main features of the general-purpose timers:

- Maximum period of 16-bit Timer is 419.4ms at 40MHz and minimum resolution is 25ns at 40MHz
- Maximum period of 8-bit Timer is 26.2ms at 40MHz and minimum resolution is 50ns at 40MHz
- Programmable clock source for timer, including an external clock
- Input capture capability with programmable trigger edge on input pin
- PWM mode operation (Timer3 and Timer4 only)
- DMA mode operation (Timer1 and Timer3 only)

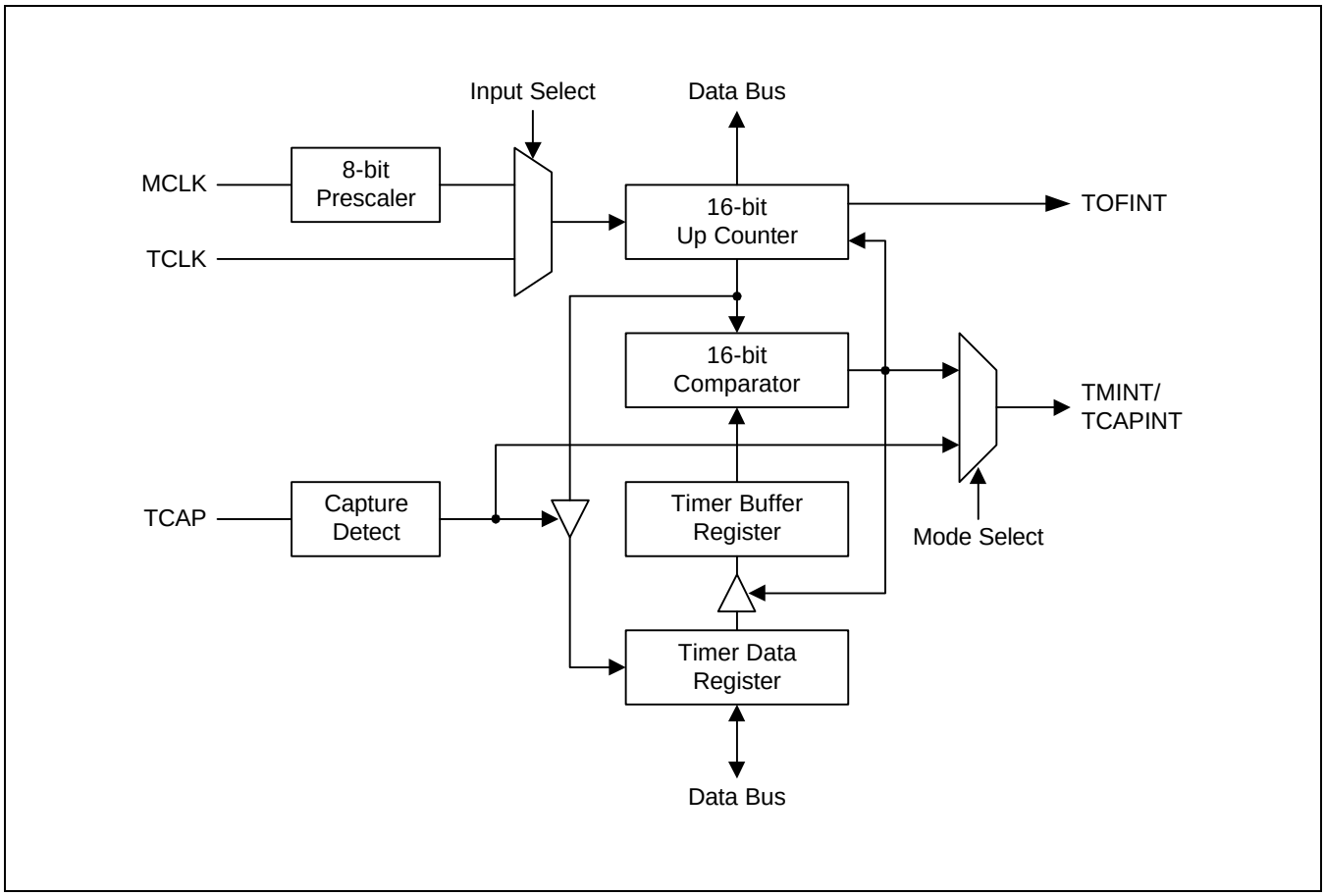


Figure 8-1. 16-Bit Timer Block Diagram

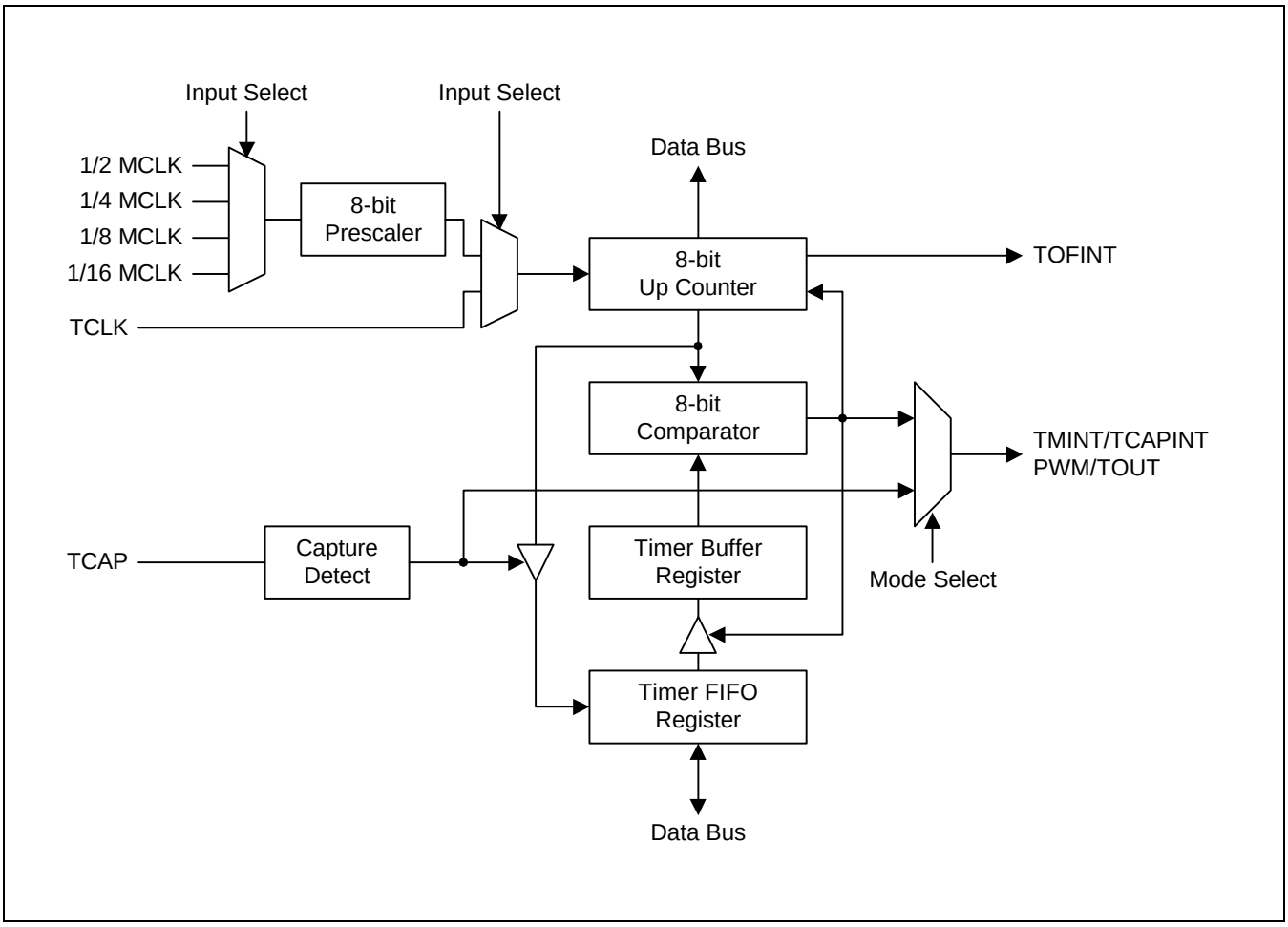


Figure 8-2. 8-Bit Timer Block Diagram

OPERATION DESCRIPTION

16-BIT TIMERS (TIMER0, TIMER1 AND TIMER2)

Interval Mode Operation

In interval mode, a match signal should be generated when the counter value is identical to the value written to the timer data register, TDAT0, TDAT1 and TDAT2. The match signal can generate a timer 0, 1, or 2 match interrupt and clear the counter value. When a match condition happens, the timer output(TOUT0/1/2) will be toggled.

Capture Mode Operation

In capture mode, the timer can perform the capturing operation, which is that the counter value is transferred into the capture register(Timer Data Register) in synchronization with an external trigger. The external triggering signal for capturing operation is a pre-defined valid edge on the capture input pin. When this valid signal happens, the counter value in process should be moved into the capture register(Timer Data Register). By using the capturing function, users can measure the time difference between external events. If a valid trigger signal on the pin does not happen before the overflow, an overflow interrupt will be generated and the counter value will be counted from 0000h, again.

Match & Overflow Mode Operation

In this mode, a match signal can be generated when the counter value is identical to the value written to the timer data register. However, the match signal does not clear the counter even if it can generate a match interrupt as same as the interval mode. Because it does not clear the counter value, the timer can run up to the overflow of counter value and generate an overflow interrupt, also. After the overflow of counter value, the counter value will be counted from 0000h, again.

DMA Mode Operation (Timer 1 Only)

Users can use the DMA to support the Timer 1. The DMA can transfer the data in memory to the TDAT1(Timer Data Register). When the match interrupt happens, the Timer 1 can request the DMA service to transfer the data into the TDAT1 register, again. Before the DMA-based operation, users should configure the control information on DMA, such as TCON1[5:3] to "010", TDAT1, destination address, source address, and so on. This kind of DMA-based timer operation is very helpful to generate the pre-defined timing event.

8-BIT TIMER (TIMER3 AND TIMER4)

Interval Mode Operation

In interval mode, a match signal should be generated when the counter value is identical to the value written to the timer data register, TDAT3 and TDAT4. The match signal can generate a timer match interrupt and clear the counter value. When a match condition happens, the timer output(TOUT3/4) will be toggled.

Capture Mode Operation

In capture mode, the timer can perform the capturing operation, which is that the counter value is transferred into the capture register(Timer Data Register) in synchronization with an external trigger. The external triggering signal for capturing operation is a pre-defined valid edge on the capture input pin. When this valid signal happens, the counter value in process should be moved into the capture register(Timer Data Register). By using the capturing function, users can measure the time difference between external events. If a valid trigger signal on the pin does not happen before the overflow, an overflow interrupt will be generated and the counter value will be counted from 00h, again.

PWM Mode Operation

The timer can be used for generating the PWM(Pulse Width Modulation) signal. Timer3/4 can support the PWM functionality, which is different from Timer0/1/2.

In this mode, a match signal should be generated when the counter value is identical to the written to the timer FIFO register(Timer Data Register). However, because the match signal dose not clear the counter, it can generate an overflow interrupt when the counter value reaches to the ffh. After the overflow of counter value, the timer will count its value from 00h, again. To generate the PWM signal, the PWM output should be "Low" level as long as the counter value is less than or equal(≤) to the value specified in Timer Buffer Register and "High" level as long as the counter value is greater than (>) the value specified in Timer Buffer Register. Because it is 8-bit PWM timer, the one period is equal to $t_{CLK} \times 256$.

The pre-scale value can define the input clock frequency of Timer according to the following equation:

Timer input clock frequency(t_{CLK}) = MCLK / (pre-scale value + 1) : for Timer 0, 1 and 2

Timer input clock frequency(t_{CLK}) = MCLK / (pre-scale value +1) / (divider value) : for Timer 3 and 4

pre-scale value = 0 – 255, divider value = 2, 4, 8, 16

DMA Mode Operation (Timer 3 Only)

Users can use the DMA to support the Timer 1. The DMA can transfer the data in memory to the TDAT3(Timer Data Register). When the match interrupt happens, the Timer 1 can request the DMA service to transfer the data into the TDAT3 register, again. Before the DMA-based operation, users should configure the control information on DMA, such as TCON3[5:3] to "010", TDAT3, destination address, source address, and so on. This kind of DMA-based timer operation is very helpful to generate the pre-defined timing event or the sound using PWM.

TIMER SPECIAL FUNCTION REGISTER

TIMER CONTROL REGISTERS

Register	Offset Address	R/W	Description	Reset Value
TCON0	0x9003	R/W	Timer 0 control register	0x00
TCON1	0x9013	R/W	Timer 1 control register	0x00
TCON2	0x9023	R/W	Timer 2 control register	0x00
TCON3	0x9033	R/W	Timer 3 control register	0x00
TCON4	0x9043	R/W	Timer 4 control register	0x00

TCON0, 1, 2	Bit	Description	Initial State
Reserved	[1:0]	Reserved	00
ICS	[2]	**Timer Input Clock Selection:** This bit can determine the input clock source of Timer. 0 = Internal Clock 1 = External Clock	0
OMS	[5:3]	**Timer Operating Mode Selection:** This field can determine the operating mode of Timer. 000 = Interval mode operation 001 = Match & overflow mode operation 010 = Match & DMA mode operation (Timer1 Only) 100 = Capture on falling edge of TCAP0, 1, and 2 101 = Capture on rising edge of TCAP0, 1, and 2 110 = Capture on rising or falling edges of TCAP0, 1, and 2	000
CL	[6]	**Timer Counter Clear:** This bit can clear the content of timer counter register. 0 = No effect 1 = Clearing the counter register	0
TEN	[7]	**Timer Enable:** This bit can enable or disable of Timer functionality. 0 = Disable (Stop) 1 = Enable (Start)	0

TCON3, 4	Bit	Description	Initial State
CD	[1:0]	**Clock Divider of Internal Clock Source:** This field can determine the divider factor of timer clock source. This bit is only effective when users take the timer clock source as internal CPU clock. In other word, this is effective when ICS bit is set to "0". 00 = 1/16 01 = 1/8 10 = 1/4 11 = 1/2	00
ICS	[2]	**Timer Input Clock Selection:** This bit can determine the input clock source of Timer. 0 = Internal Clock 1 = External Clock	0
OMS	[5:3]	**Timer Operating Mode Selection:** This field can determine the operating mode of Timer. 000 = Interval mode operation 001 = PWM Mode 010 = Match & DMA mode operation (Timer3 Only) 100 = Capture on falling edge of TCAP3, and 4 101 = Capture on rising edge of TCAP3, and 4 110 = Capture on rising or falling edges of TCAP3, and 4	000
CL	[6]	**Timer Counter Clear:** This bit can clear the content of timer counter register. 0 = No effect 1 = Clearing the Counter register	0
TEN	[7]	**Timer Enable:** This bit can enable or disable of Timer functionality. 0 = Disable (Stop) 1 = Enable (Start)	0

NOTES:

1. Timer is continuously operated by one time enabling.
2. If FIFO is enabled in PWM mode, data is not stored into TDAT4. So to store data into TDAT4, FIFO should be disabled.

TIMER FIFO CONTROL REGISTERS

Register	Offset Address	R/W	Description	Reset Value
TFCON	0x904f	R/W	FIFO control register of Timer 4	0x0

TFCON	Bit	Description	Initial State
FEN	[0]	**FIFO Enable:** This bit can determine whether or not to use the FIFO 0 = FIFO disable 1 = FIFO enable	0
FCL	[1]	**FIFO Reset:** This bit can clear the content of Timer FIFO. This bit is automatically cleared after clearing FIFO. 0 = Normal mode 1 = FIFO clearing	0
FTL	[3:2]	**FIFO Trigger Level:** This field can determine the trigger level of FIFO empty interrupt. 00 = Empty 01 = 1 byte 10 = 2 byte 11 = 4 byte	00
FR	[5:4]	**FIFO Repeat:** This field can determine the number of the usage of FIFO data. The seven repeat means that each FIFO data should be used in Timer seven times before taking next data in FIFO. 00 = No effect 01 = One repeat 10 = Three repeat 11 = Seven repeat	00

TIMER FIFO STATUS REGISTERS

Register	Offset Address	R/W	Description	Reset Value
TFSTAT	0x904e	R	FIFO status register of Timer 4	0x0

TFSTAT	Bit	Description	Initial State
FC	[2:0]	**FIFO Count:** Number of data in Timer FIFO	000
FF	[3]	**FIFO Full:** This bit is automatically set to "1" whenever Timer FIFO is full in case of the FIFO enable 0 = 0 ≤ Timer FIFO Data ≤ 4 1 = Full	0

TIMER PRESCALER REGISTERS

Register	Offset Address	R/W	Description	Reset Value
TPRE0	0x9002	R/W	Timer 0 pre-scale register	0xff
TPRE1	0x9012	R/W	Timer 1 pre-scale register	0xff
TPRE2	0x9022	R/W	Timer 2 pre-scale register	0xff
TPRE3	0x9032	R/W	Timer 3 pre-scale register	0xff
TPRE4	0x9042	R/W	Timer 4 pre-scale register	0xff

TPREx	Bit	Description	Initial State
Pre-scale	[7:0]	This field can determines pre-scale value for Timer 0,1,2,3, and 4	0xff

TIMER 0, 1, AND 2 DATA REGISTERS

Register	Offset Address	R/W	Description	Reset Value
TDAT0	0x9000	R/W	Timer 0 data register	0xffff
TDAT1	0x9010	R/W	Timer 1 data register	0xffff
TDAT2	0x9020	R/W	Timer 2 data register	0xffff

TDAT0,1,2	Bit	Description	Initial State
Data	[15:0]	This field can determine the data value for Timer 0,1, and 2	0xffff

TIMER 3 AND 4 DATA REGISTER & FIFO REGISTERS

Register	Offset Address	R/W	Description	Reset Value
TDAT3	0x9031	R/W	Timer 3 data register	0xff
TDAT4	0x9041	R/W	Timer 4 data register	0xff
TFB4	0x904b	R/W	Timer 4 FIFO register @ byte access, FIFO	0x0
TFHW4	0x904a	R/W	Timer 4 FIFO register @ half-word access, FIFO	0x0
TFW4	0x9048	R/W	Timer 4 FIFO register @ word access, FIFO	0x0

TDAT3, 4: Non FIFO mode, Byte access (by STRB)

TDAT3, 4	Bit Size	Description	Initial State
TDATA	[7:0]	Timer data for Timer 3 and 4	0xff

TFB4: Byte access, FIFO Mode

TFB4	Bit Size	Description	Initial State
TDATA	[7:0]	Timer data for Timer 4 FC(FIFO Count) = FC(FIFO Count) + 1	0x0

TFHW4: Half-word access, FIFO Mode

TFHW4	Bit Size	Description	Initial State
TDATA	8 Bit 8 Bit	Timer data0 for Timer 4 Timer data1 for Timer 4 FC(FIFO Count) = FC(FIFO Count) + 2	0x0

TFW4: Word access, FIFO Mode

TFW4	Bit Size	Description	Initial State
TDATA	8 Bit 8 Bit 8 Bit 8 Bit	Timer data0 for Timer 4 Timer data1 for Timer 4 Timer data2 for Timer 4 Timer data3 for Timer 4 FC(FIFO Count) = FC(FIFO Count) + 4	0x0

TIMER 0, 1, AND 2 COUNT REGISTERS

Register	Offset Address	R/W	Description	Reset Value
TCNT0	0x9006	R	Timer 0 count register	0x0
TCNT1	0x9016	R	Timer 1 count register	0x0
TCNT2	0x9026	R	Timer 2 count register	0x0

TDAT0,1,2	Bit	Description	Initial State
CV	[15:0]	This field contains the current timer's count value during the normal operation	0x0

TIMER 3 AND 4 COUNT REGISTERS

Register	Offset Address	R/W	Description	Reset Value
TCNT3	0x9037	R	Timer 3 count register	0x0
TCNT4	0x9047	R	Timer 4 count register	0x0

TDAT3, 4	Bit Size	Description	Initial State
CV	[7:0]	This field contains the current timer's count value during the normal operation	0x0

NOTES

9 UART

OVERVIEW

The UART(Universal Asynchronous Receiver and Transmitter) in S3C3410X can support one asynchronous serial I/O ports. The UART can be operated by the interrupt-based or DMA-based mode. In other words, the UART can generate an interrupt or DMA request to prepare the data to be sent, or to store the received data into the memory. It has two 8-byte FIFOs for receive and transmit. One is for receiving FIFO and the other is for transmitting FIFO. The functionality of UART includes the programmable baud-rate, frame format suitable for infra-red (IrDA ver. 1.0) transmit/receive, programmable number of stop bit insertion, programmable data width of 5, 6, 7, and 8-bit, and parity checking/attaching capability of received/transmitted data.

The UART has a baud-rate generator, transmitter/receiver block and their control unit as shown in Figure 9-1. The baud-rate generator can generate the suitable baud rate for UART by using MCLK. To generate the proper baud rate, users should configure the proper division rate of MCLK in special register in baud rate generator. To support the higher baud rate, the UART in S3C3410X has internal 8-byte FIFO. The data in FIFO should be transferred into Transmitter Shifter for TX and data in Receive Shifter should be moved into FIFO for RX. The TX data in Transmitter Shifter will be shifted out through the UTXD pin for TX case. Also, the data on URXD will also be shifted in Receive Shifter for RX case.

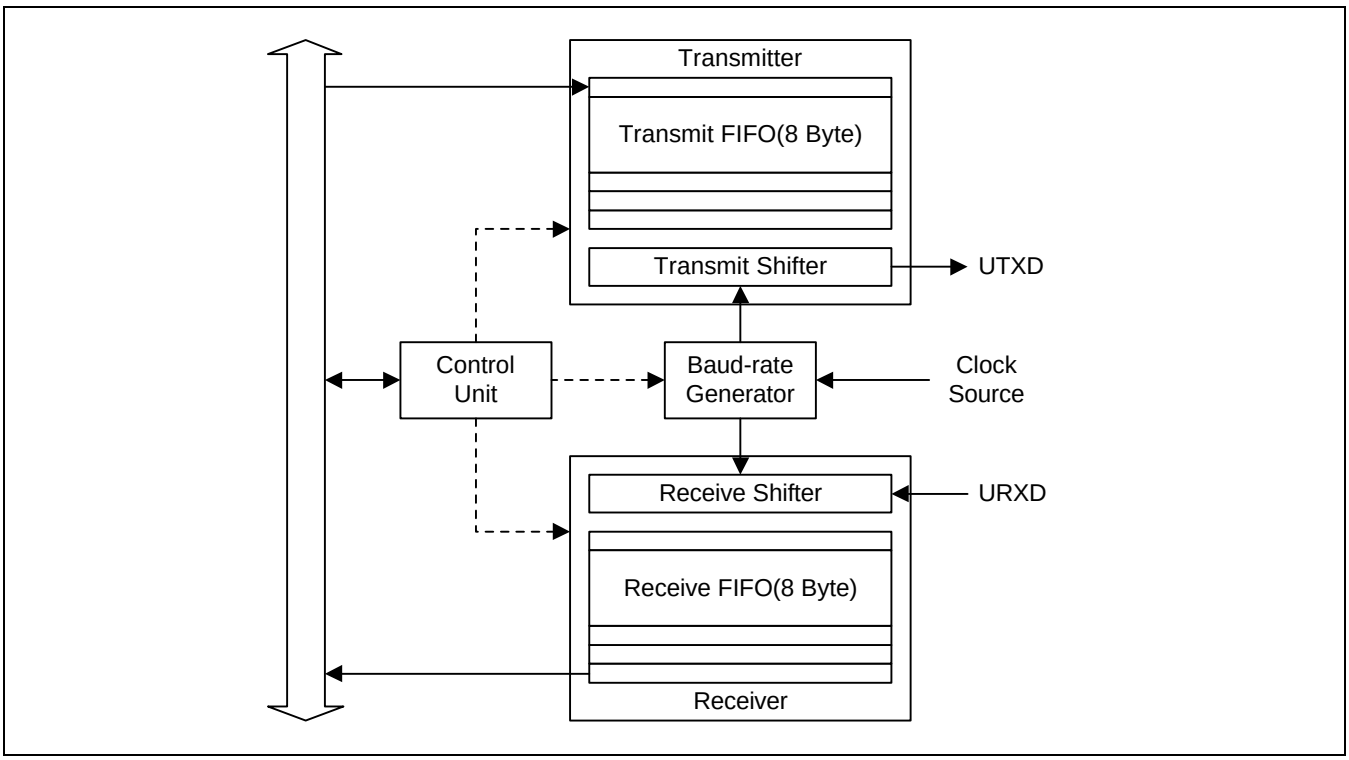


Figure 9-1. UART Block Diagram with FIFO

UART FUNCTION DESCRIPTION

UART OPERATION

The following section describe the operation of UART which include the data transmission, data reception, interrupt generation, baud-rate generation, loopback mode, infra-red mode, and so on.

Data Transmission

The data frame for transmission is programmable. It can have several options regarding to the data size, number of stop bit, parity checking capability, and so on, which can be specified in the Line Control Register(ULCON). Sometimes, users need to send the break condition during the sending the UART frame. The break condition can be realized by writing SBS bit in UCON register. If users write the SBS bit in UCON register during the UART frame sending, the break condition forces the serial output to logic 0 state at least for longer time than one frame transmission after successful sending the current UART frame. This break condition will be automatically cleared after one frame of break time. The UART will send the frame data again after break time. On the receive side, if the receive controller recognize the break condition from Transmitter, there will be break interrupt to CPU.

The data transmission process is shown in Figure 9-2. The transmitter should transfer the data through a path as follows: data source -> transmit holding(transmit FIFO) register -> transmit shifter -> UTXD pin. Two flags(status signals) such as transmit holding(transmit FIFO) register empty and transmitter empty, are used to indicate the status of the transmit holding(transmit FIFO) register and transmitter.

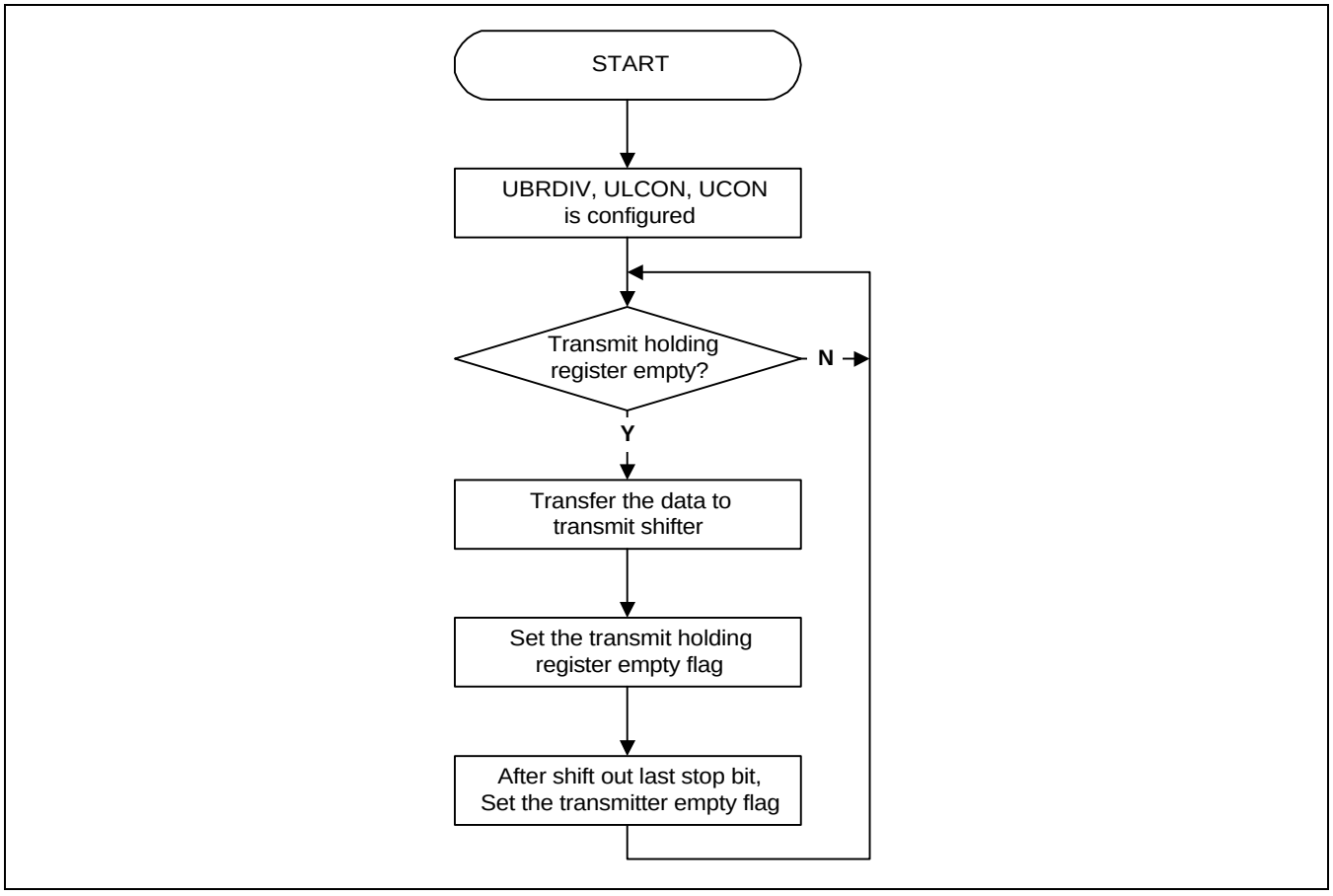


Figure 9-2. UART Data Transmission Process

Data Reception

The RX block of UART can also support several options necessary for UART frame receiving as similar with TX. It can support the option on data size, number of stop bit, parity checking capability, and so on, which can also be specified in the Line Control Register(ULCON). The receiver block of UART can detect the erroneous such as overrun error, parity error, frame error and break condition from TX.

The overrun error indicates that new data has overwritten the previously received data before the previous one has been read. The parity error indicates that the receiver has detected a parity error, which is due to different parity bit from the expectation. The frame error indicates that the received data does not have a valid stop bit in terms of frame boundary. The break condition indicates that the URXD input is held in the logic 0 state at least for longer time than one frame transmission. The receive time-out condition occurs when the UART receiver does not receive the data during the necessary time of 3 half-word (6 bytes) transmission when the Rx FIFO is not empty state in FIFO mode.

The data reception process is shown in Figure 9-3. The receiver transfer data through a path as follows: URXD pin -> receive shifter register -> receive buffer register -> destination. A receive buffer (receive FIFO) full flag as well as several error flags during the reception can be used to indicate the status of the receive buffer (receive FIFO) register.

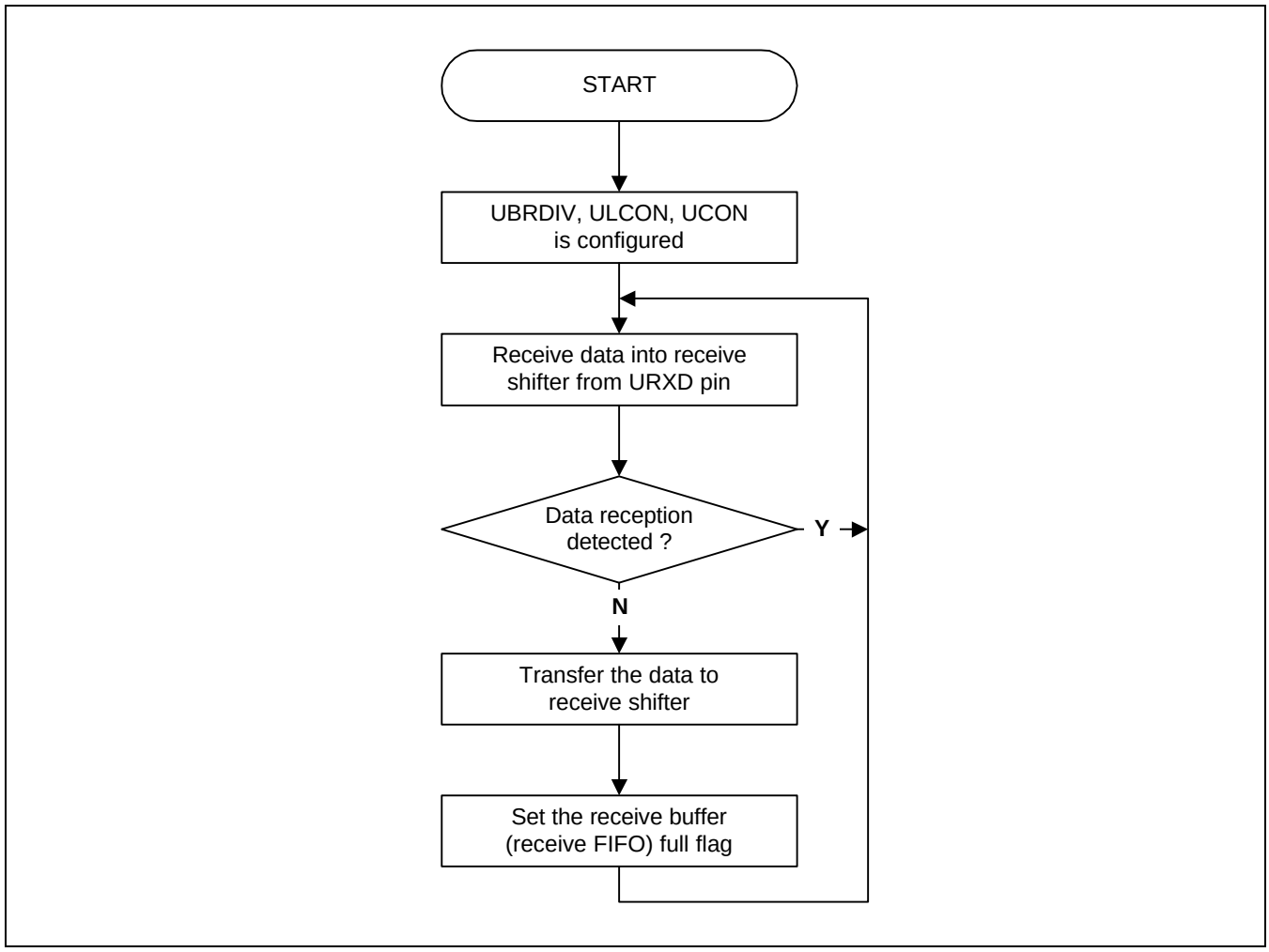


Figure 9-3. UART Data Reception Process

Interrupt / DMA Request Generation

The UART of S3C3410X has eight status signals: overrun error, parity error, frame error, break, receive FIFO ready, receiver time out, transmit FIFO empty and transmitter empty, which are specified in the corresponding UART status register(USTAT).

The overrun error, parity error, frame error and break condition are referred to as the receive status, each of which can cause the receive status interrupt request if the receive status interrupt enable bit is set to one in the control register(UCON). When a receive status interrupt request is detected, users can know the interrupt source by reading the content of UCON register.

When the receiver transfers the data in the receive shifter to the receive FIFO, there will be the activation of the receive FIFO ready status signal, which will cause the receive interrupt if the receive mode in control register is selected as the interrupt mode.

When the transmitter transfers the data in the transmit FIFO to transmit shifter, there will be the activation of the transmit FIFO empty status signal, which will cause the transmit interrupt if the transmit mode in control register is selected as the interrupt mode.

The receive FIFO ready and transmit FIFO empty status signals can also be connected to generate the DMA request signals if the receive/transmit mode is selected as the DMA mode.

Interrupt generation relating with FIFO

Type	FIFO Mode	Non-FIFO Mode
Rx Interrupt	When UART receive one frame data, it should move the data into FIFO. After loading the data into FIFO, it should determine whether an interrupt should be generated by looking the FIFO trigger level, or not.	When UART receives one frame data successfully, it will generate an interrupt.
Tx Interrupt	When UART transmit one frame data, it should extract the data from FIFO. After extracting the data from FIFO, it should determine whether an interrupt should be generated by looking the FIFO trigger level, or not.	When UART transmits one frame data successfully, it will generate an interrupt.
Error Interrupt	When UART detects the frame error, parity error and break condition, it does not generate the interrupt immediately. The UART will move the data into receive FIFO and it will generate the interrupt when the data move to the top of receive FIFO. However, an overrun error as well as receive time-out should generate an interrupt immediately.	All erroneous condition should generate an error interrupt immediately. However, if several interrupt happen simultaneously, the interrupt routine should discriminate the interrupt source by looking the content of UCON register.

Baud Rate Generation

The UART's baud-rate generator provides the serial clock for transmitter and receiver. The source clock for the baud-rate generator should be the S3C3410X's internal system clock. The baud-rate clock is generated by dividing the source clock by 16 and a 16-bit divisor specified by the UART baud-rate divisor register (UBRDIV). The UBRDIV can be determined as follows:

UBRDIV = (round_off) { MCLK / (Transfer rate ´ 16) } – 1

Where the divisor should be from 1 to (2^{16} – 1). For example, if the baud-rate is 115200bps and MCLK is 40MHz, UBRDIV is:

UBRDIV = (int) { MCLK / (Transfer rate ´ 16) + 0.5 } – 1

*= (int) { 40000000 / (115200 * 16) + 0.5 } – 1 = (int) (21.7 + 0.5) – 1*

= 22 – 1 = 21

Loop Back Mode

The S3C3410X UART can support a test mode, so called the loop back mode. In this mode, the transmitted data from UART Tx module is immediately received through UART Rx module via internal connection between Tx and Rx module. This feature allows that the processor can verify the internal transmit/receive data path of UART channel. This mode can be selected by setting the loop back bit in the UART control register(UCON).

Infra Red(IrDA) Mode

The UART in S3C3410X can support the frame of infra-red (IrDA) transmit and receive, which can be selected by setting the infra-red bit in the UART control register (UCON). As shown in Figure 9-4, we should have IrDA Tx Encoder and Rx Decoder, which is different from the normal UART operation mode. By using the specific Decoder/Encoder for IrDA, the signal frame in IrDA is different from the normal signal frame of UART, which is shown in Figure 9-5, 9-6 and 9-7. In IrDA transmit mode, the transmitter should pulse 3/16 duty to represent a zero data as shown in Figure 9-6. In IrDA receive mode, the receiver should detect the 3/16 pulsed period to recognize a zero data as shown in Figure 9-7.

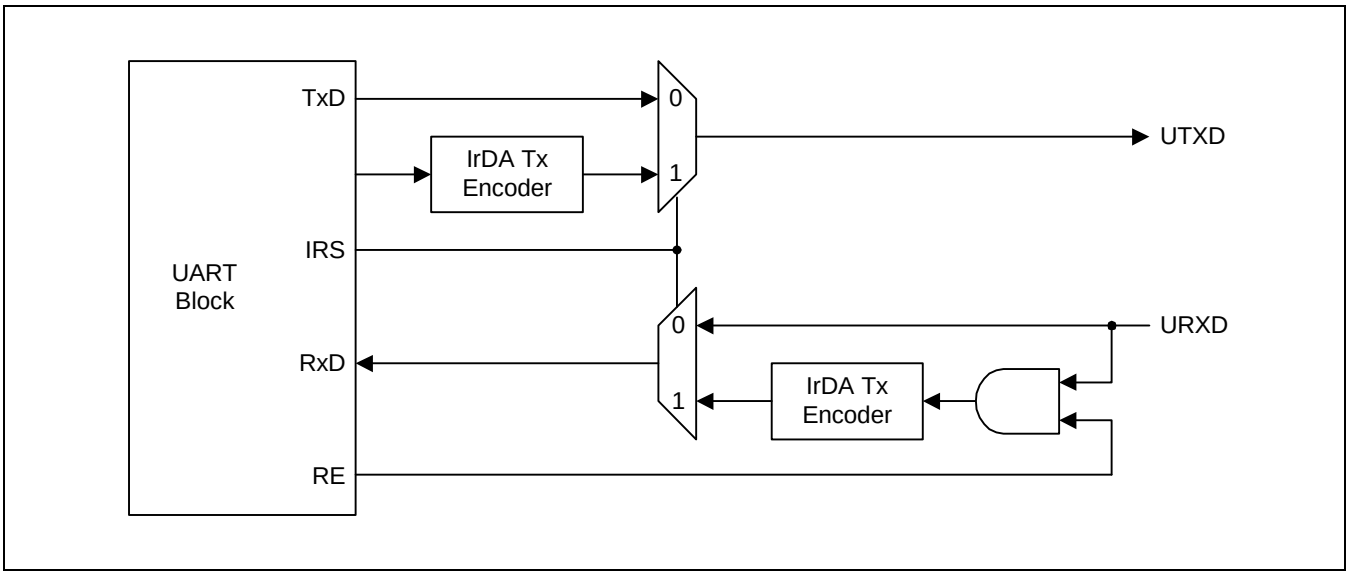


Figure 9-4. IrDA Function Block Diagram

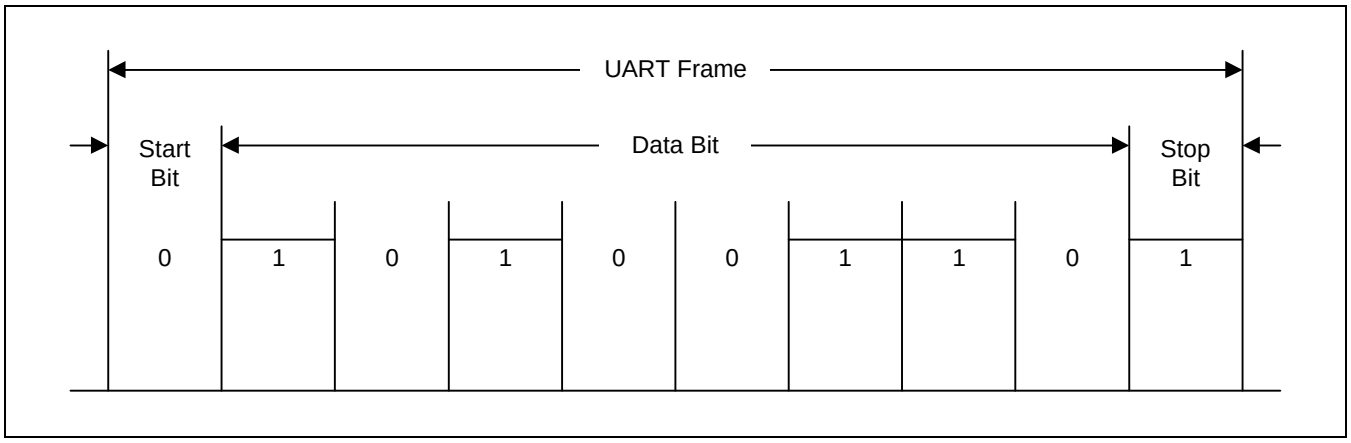


Figure 9-5. Serial I/O Frame Timing Diagram (Normal UART)

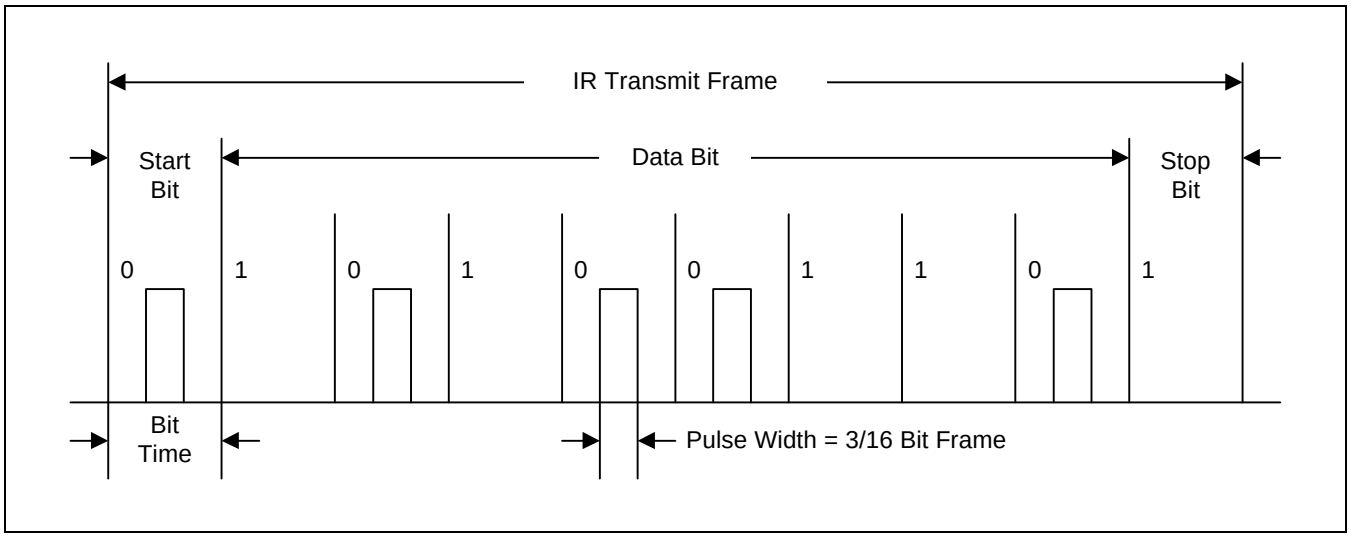


Figure 9-6. Infra Red(IrDA) Transmit Mode Frame Timing Diagram

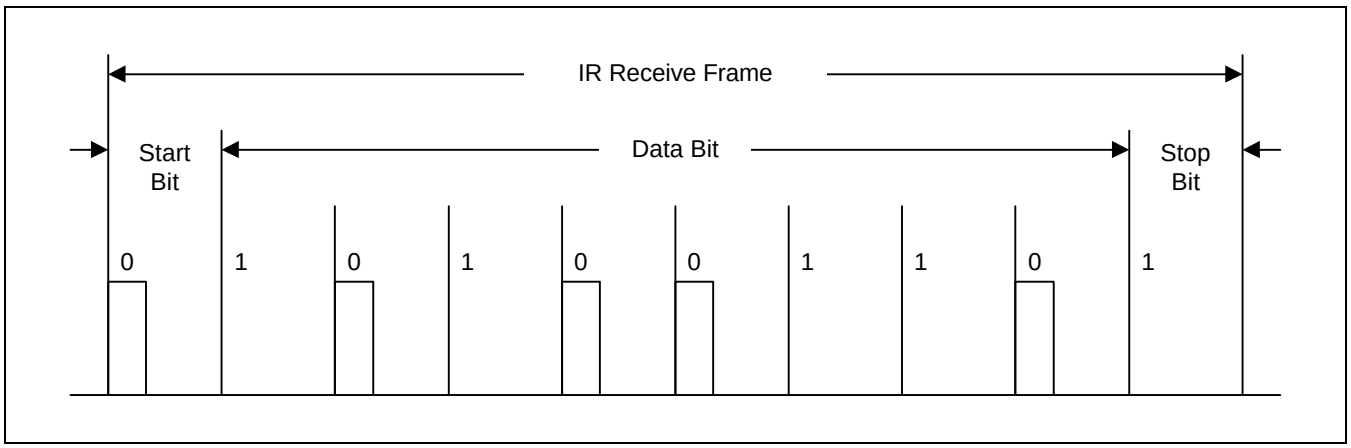


Figure 9-7. Infra Red(IrDA) Receive Mode Frame Timing Diagram

UART SPECIAL FUNCTION REGISTERS

UART LINE CONTROL REGISTER (ULCON)

This is UART line control register, ULCON, is used to control UART block.

Register	Offset Address	R/W	Description	Reset Value
ULCON	0x5003	R/W	UART line control register	0x0

ULCON	Bit	Description	Initial State
WL	[1:0]	**Word Length:** The word length indicates the number of data bit to be transmitted or received per frame 00 = 5-bits 01 = 6-bits 10 = 7-bits 11 = 8-bits	00
SB	[2]	**Number of Stop Bit:** The number of stop bit per frame should be specified by using SB. 0 = One stop bit per frame 1 = Two stop bit per frame	0
PMD	[5:3]	**Parity Mode:** The parity mode can specify the parity mode. When the parity mode is enabled, the parity generation for Tx and parity checking for Rx will be performed automatically during the Tx and Rx operation of UART. 0xx = No parity 100 = Odd parity 101 = Even parity	000
IRM	[6]	**Infra-Red Mode :** The Infra-Red mode can determine whether or not to be use the Infra-Red mode. 0 = Normal mode operation 1 = Infre-Red Tx/Rx mode	0

UART CONTROL REGISTER (UCON)

Register	Offset Address	R/W	Description	Reset Value
UCON	0x5007	R/W	UART control register	0x0

UCON	Bit	Description	Initial State
RM	[1:0]	**Receive Mode:** This field can determine the operation mode of UART. The UART can be operated by DMA as well as interrupt mechanism. If the interrupt is not enabled, it is polling mode. 00 = Disable 01 = Interrupt request or polling mode 10 = DMA0 request 11 = DMA1 request	00
TM	[3:2]	**Transmit Mode:** This field can determine the operation mode of UART. The UART can be operated by DMA as well as interrupt mechanism. If the interrupt is not enabled, it is polling mode. 00 = Disable 01 = Interrupt request or polling mode 10 = DMA0 request 11 = DMA1 request	00
SBS	[4]	**Send Break Signal:** Setting this bit can cause the UART to send a break signal. 0 = Normal transmit 1 = Send break signal	0
LBM	[5]	**Loop Back Mode:** Setting loop back bit to "1" can cause the UART to enter loop back mode. The Loop Back Mode means the internal connection between Tx and Rx module for test purpose. 0 = Normal operation 1 = Loopback mode	0
RSIE	[6]	**Rx Status Interrupt Enable:** This bit enables the UART to generate an interrupt if an exception, such as a break, frame error, parity error, or overrun error occurs during a receive operation. 0 = Do not generate receive status interrupt 1 = Generate receive status interrupt	0
RXTOEL	[7]	**Rx Time Out Enable:** Enable/Disable Rx time out interrupt. This bit is only effective when the FIFO is enabled. 0 = Disable 1 = Enable	0

UART STATUS REGISTER (USTAT)

The UART status register, USTAT, is a read-only register which is used to monitor the status during the operation of serial I/O in the UART

Register	Offset Address	R/W	Description	Reset Value
USTAT	0x500b	R	UART status register	0xc0

USTAT	Bit	Description	Initial State
OE	[0]	**Overrun Error:** This bit is automatically set to "1" whenever an overrun error occurs during the receive operation. 0 = No overrun error during receive 1 = Overrun error	0
PE	[1]	**Parity Error:** This bit is automatically set to "1" whenever an parity error occurs during the receive operation. 0 = No parity error during receive 1 = Parity error	0
FE	[2]	**Frame Error:** This bit is automatically set to "1" whenever an frame error occurs during the receive operation. 0 = No frame error during receive 1 = Frame error	0
BD	[3]	**Break Detect:** This bit is automatically set to "1" to indicate that a break signal has been received. 0 = No break received 1 = Break received	0
RTO	[4]	**Receiver Time Out:** This bit is automatically set to "1" whenever a receiver time out occurs during the receive operation. 0 = No receiver time out during receive 1 = Generate receiver time out	0
RFDR	[5]	**Receive FIFO Data Ready / Receive Buffer Data Ready:** This bit is automatically set to "1" whenever the receiver is ready to receive the data through the URXD pin. 0 = Completely empty 1 = 1-byte ≤ Rx FIFO Data ≤ 8-byte @ FIFO mode The buffer register has a received data @ Non FIFO mode	0
TFE	[6]	**Transmit FIFO Empty / Transmit Holding Register Empty:** This bit is automatically set to "0" whenever the transmitter has the valid data for sending. 0 = 1-byte ≤ FIFO ≤ 8-byte @ FIFO mode The holding register is not empty @ Non FIFO mode 1 = Empty	1
TSE	[7]	**Transmit Shift Register Empty:** This bit is automatically set to "1" whenever the transmit shift register does not have a valid data for sending. 0 = Not empty 1 = Transmit holding & shifter register empty	1

UART FIFO CONTROL REGISTER (UFCON)

Register	Offset Address	R/W	Description	Reset Value
UFCON	0x500f	R/W	UART FIFO control register	0x0

UFCON	Bit	Description	Initial State
FE	[0]	**FIFO Enable:** This bit can determine whether or not to use the FIFO mode. 0 = FIFO Disable 1 = FIFO enable	0
RFR	[1]	**Receive FIFO Reset:** To reset the receive FIFO, user should set RFR bit. 0 = Normal mode 1 = Rx FIFO reset	0
TFR	[2]	**Transmit FIFO Reset:** To reset the transmit FIFO, user should set TFR bit. 0 = Normal mode 1 = Tx FIFO reset	0
Reserved	[3]	Reserved	0
RFTL	[5:4]	**Receive FIFO Trigger Level for Interrupt Generation:** This field can determine the interrupt trigger level of receive FIFO. 00 = 2 byte 01 = 4 byte 10 = 6 byte 11 = 8 byte	00
TFTL	[7:6]	**Transmit FIFO Trigger Level for Interrupt Generation:** This field can determine the interrupt trigger level of transmit FIFO. 00 = Empty 01 = 2 byte 10 = 4 byte 11 = 6 byte	00

UART FIFO STATUS REGISTER (UFSTAT)

Register	Offset Address	R/W	Description	Reset Value
UFSTAT	0x5012	R	UART FIFO control register	0x0

UFSTAT	Bit	Description	Initial State
RFC	[2:0]	**Receive FIFO Count:** This field can indicate the number of current data in Receive FIFO.	000
TFC	[5:3]	**Transmit FIFO Count:** This field can indicate the number of current data in Transmit FIFO.	000
RFF	[6]	**Receive FIFO Full:** This bit is automatically set to "1" whenever receive FIFO is full during receive operation. 0 = 0 ≤ Rx FIFO Data ≤ 7 byte 1 = Full	0
TFF	[7]	**Transmit FIFO Full:** This bit is automatically set to "1" whenever transmit FIFO is full during transmit operation. 0 = 0 ≤ Rx FIFO Data ≤ 7 byte 1 = Full	0
EIF	[8]	**Error in FIFO:** This bit represent that there is not valid data in the receive FIFO. 0 = All data in receive FIFO are valid 1 = Some data in receive FIFO is not valid.	0

UART TRANSMIT HOLDING REGISTER & FIFO REGISTERS

Register	Offset Address	R/W	Description	Reset Value
UTXH	0x5017	W	UART transmit holding register	0x0
UTXH_B	0x5017	W	UART transmit FIFO register @ byte access	0x0
UTXH_HW	0x5016	W	UART transmit FIFO register @ half-word access	0x0
UTXH_W	0x5014	W	UART transmit FIFO register @ word access	0x0

UTXH	Bit	Description	Initial State
TXDATA	[7:0]	This field represents the data to be transmitted through TX module in UART. When users write the data in this register, the transmit holding register empty bit(TFE) in the status register should be set to "0". This bit is for preventing the overwriting on transmitted data that may already be existed in the UTXH register. Users should update the UTXH after checking TFE bit. Whenever the UTXH is written with new value, the transmit register empty bit(TFE) will be automatically cleared to "0"	0x0

UTXH_B : Byte Access, FIFO Mode

UTXH_B	Bit	Description	Initial State
TXDATA0	[7:0]	Transmit data for UART; When users write the byte data in this register, FIFO_COUNT = FIFO_COUNT + 1	0x0

UTXH_HW : Half-word Access, FIFO Mode

UTXH_HW	Bit	Description	Initial State
TXDATA0 TXDATA1	[7:0] [15:8]	Transmit data0 for UART Transmit data1 for UART When users write the half-word data in this register, FIFO_COUNT = FIFO_COUNT + 2	0x0 0x0

UTXH_W : Word Access, FIFO Mode

UTXH_W	Bit	Description	Initial State
TXDATA0 TXDATA1 TXDATA2 TXDATA3	[7:0] [15:8] [23:16] [31:24]	Transmit data0 for UART Transmit data1 for UART Transmit data2 for UART Transmit data3 for UART When users write the word data in this register, FIFO_COUNT = FIFO_COUNT + 4	0x0 0x0 0x0 0x0

UART RECEIVE BUFFER REGISTER & FIFO REGISTERS

Register	Offset Address	R/W	Description	Reset Value
URXH	0x501b	W	UART receive buffer register	–
URXH_B	0x501b	W	UART receive FIFO register @ byte access	–
URXH_HW	0x501a	W	UART receive FIFO register @ half-word access	–
URXH_W	0x5018	W	UART receive FIFO register @ word access	–

URXH	Bit	Description	Initial State
RXDATA	[7:0]	This field represents the data to be received through RX module in UART. When users read the data in this register, the receive buffer data ready bit(RFDR) in the status register should be set to "0". This bit is for preventing the reading the invalid received data in the URXH register before successful reception. Users should read the URXH after checking RFDR bit. Whenever the URXH is read, the receive buffer data ready bit(RFDR) in the status register will be automatically cleared to "1"	–

URXH_B : Byte Access, FIFO Mode

URXH_B	Bit	Description	Initial State
RXDATA0	[7:0]	Receive data for UART; When users read the byte data in this register, FIFO_COUNT = FIFO_COUNT + 1	–

URXH_HW : Half-word Access, FIFO Mode

URXH_HW	Bit	Description	Initial State
RXDATA0 RXDATA1	[7:0] [15:8]	Receive data0 for UART Receive data1 for UART When users read the half-word data in this register, FIFO_COUNT = FIFO_COUNT + 2	–

URXH_W : Word Access, FIFO Mode

URXH_W	Bit	Description	Initial State
RXDATA0 RXDATA1 RXDATA2 RXDATA3	[7:0] [15:8] [23:16] [31:24]	Receive data0 for UART Receive data1 for UART Receive data2 for UART Receive data3 for UART When users read the word data in this register, FIFO_COUNT = FIFO_COUNT + 4	–

UART BAUD RATE DIVISOR REGISTER (UBRDIV)

The value in the baud rate divisor register, UBRDIV, can be used to determine the UART Tx/Rx clock rate(baud rate) as follows:

UBRDIV = (round_off) { MCLK / (transfer rate ´ 16) } – 1

Where the divisor should be from 1 to (2^{16} – 1). For example, if the baud-rate is 115200bps and MCLK is 40MHz, UBRDIV is:

UBRDIV = (int) { MCLK / (Transfer rate ´ 16) + 0.5 } – 1

*= (int) { 40000000 / (115200 * 16) + 0.5 } – 1 = (int) (21.7 + 0.5) – 1*

= 22 – 1 = 21

Register	Offset Address	R/W	Description	Reset Value
UBRDIV	0x501e	R/W	Baud rate divisor register for UART	0x0

UBRDIV	Bit	Description	Initial State
UBRDIV	[15:0]	Baud rate divisor value	0x0

10 SIO (SYNCHRONOUS I/O)

OVERVIEW

The S3C3410X SIO(Synchronous I/O) can interface with various types of external devices which requires the serial data transfer/receive. The SIO module can transmit or receive 8-bit serial data at a frequency determined by its corresponding control register. To ensure the flexible data transmission rate, users can select an internal or external clock source.

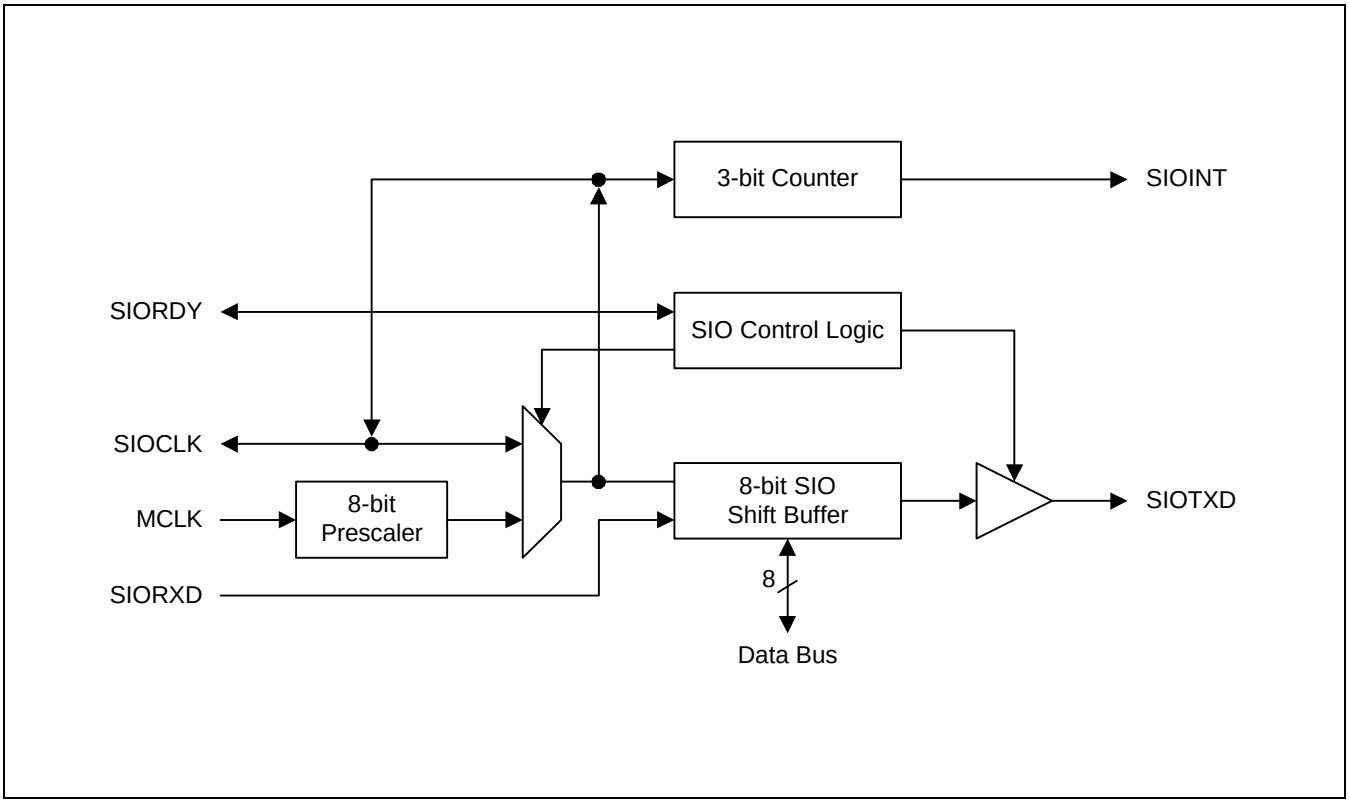


Figure 10-1. SIO Function Block Diagram

SIO FUNCTION DESCRIPTION

NON-DMA MODE OPERATION

Transmit and Receive By Synchronous Serial Line

The 8-bit data can be transmitted and received through SIO port. The serial output data can go out through a serial output pin(SIOTXD), and the serial input data can come through a serial input pin(SIORXD). In this case, the data should be sent and received synchronously by serial clock pin(SIOCLK). After transmitting or receiving data, the SIO interrupt request will be activated if users enable an interrupt for SIO TX and RX. Because of the separate hardware for TX and RX, the dual operation of TX and RX is possible. If users want to use the receiving operation, users can treat the receive data as dummy one.

The TX and RX rate can be controlled by having the appropriate configuration in the SIOCON and SBRDR registers. The clock source of serial interface can be an internal or external clock. In other words, the TX and RX rate are programmable and users can determine the rate by having a suitable configuration in the SIOCON and SBRDR registers.

Programming Procedure

When users write a byte data into the SIODAT register, the SIO will start to transmit a data if the SIO run bit is set and the transmit mode bit is enabled.

To program the operation of SIO modules, please take following steps:

1. Configure the multiplexed I/O pins as SIO related ones(SIOTXD, SIORXD, SIOCLK, and SIORDY).
2. Configure the SIOCON register to have a necessary functionality of the serial I/O module.
3. For the operation of interrupt mode in SIO, configure the interrupt mode in SIOCON register and enable interrupt-relating bits in interrupt controller.
4. In case of interrupt mode for TX, users should write a data to be transmitted in SIODAT, first. To start the transmission, users should write SB(SIO Start) bit in SIOCON register. After transmission, there should be SIO interrupt. In case of interrupt mode of RX, there will be SIO interrupt after receipt. To start the receiving operation, users should write SB bit in SIOCON register.
5. Go to step 4 if users need interrupt-based SIO operation more.

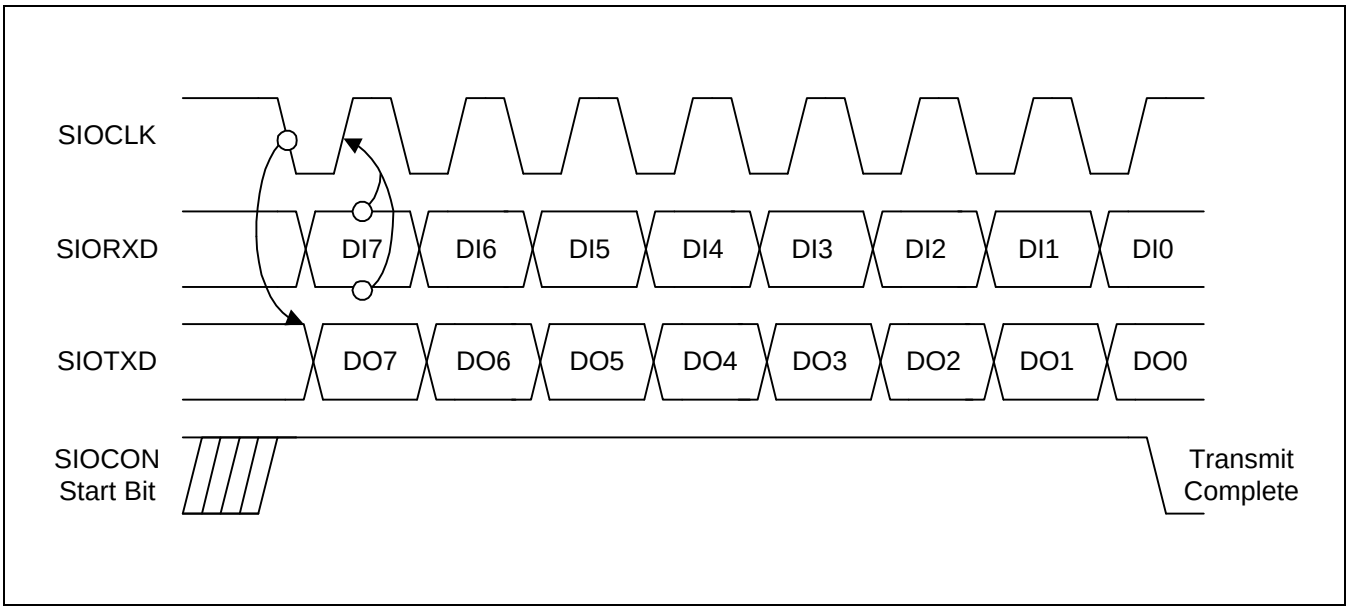


Figure 10-2. Synchronous I/O Timing Transmit/Receive Mode (Tx at Falling)

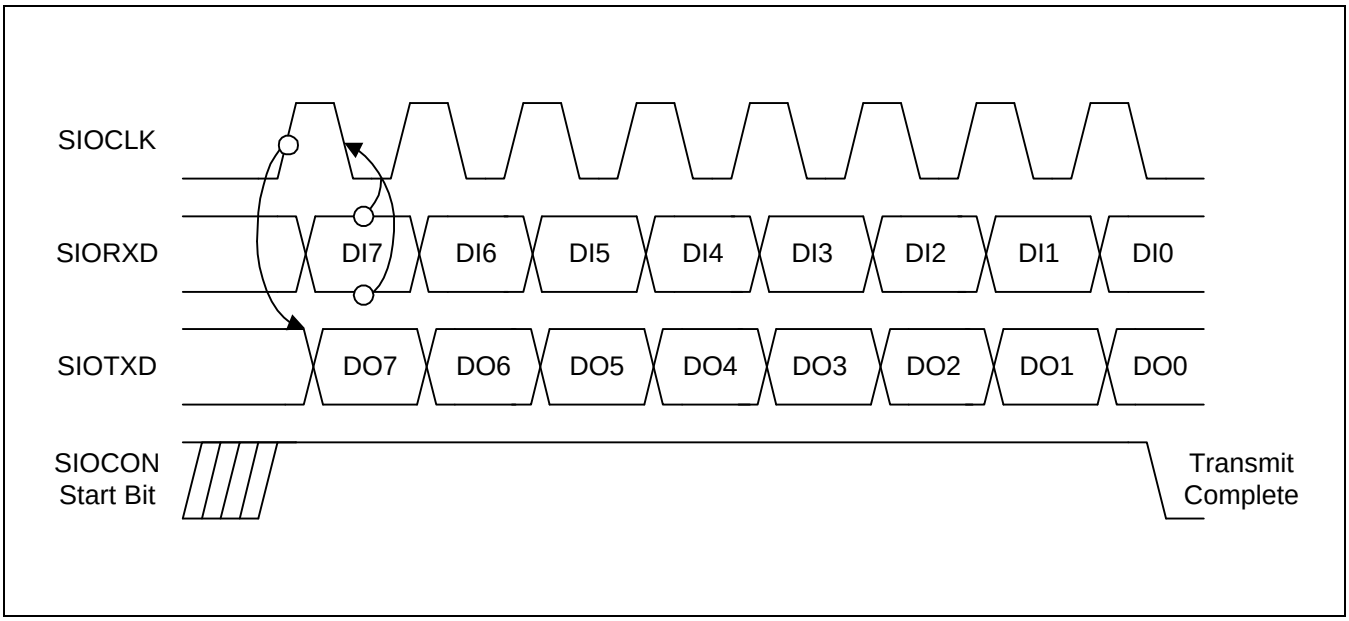


Figure 10-3. Synchronous I/O Transmit/Receive Mode (Tx at Rising)

DMA MODE OPERATION

Hand-Shaking Mode (Flag Run Mode)

In case of DMA-base SIO operation, there can be consecutive frame TX and RX. Between frames(for example, TX frame and next TX frame), SIO can watch the SIORDY signal to check whether receiving SIO is busy, or not. This is Hand-Shaking Mode. If RX SIO is not ready, TX SIO can not send the frame and should wait until RX SIO is available. To indicate the busy state, SIORDY signal should below.

Non Hand-Shaking Mode (Auto Run Mode)

The Non-Hand-Shaking Mode means that SIO can transmit its data without watching SIORDY signal during the consecutive operation. In stead of watching SIORDY signal, users can have programmable duration between frames. The duration can be specified in IVTCNT register in SIO module.

Steps for Transmit by DMA

1. For the operation of DMA mode for TX in SIO, have a suitable configuration in DMA-related register in DMA controller, first. The TRS bit in SIOCON register should be 1.
2. Configure the DMA mode in SIOCON register.
3. The SIO requests DMA service
4. The SIO transmits the data (The first transmitted data is in the first source address of DMA)
5. Go to step 3 until DMA count is "0"

Steps for Receive by DMA

1. For the operation of DMA mode for RX in SIO, have a suitable configuration in DMA-related register in DMA controller, first. The TRS bit in SIOCON register should be 0.
2. Configure the DMA mode in SIOCON register. The SB(SIO Start) bit should also be set.
3. The SIO requests the DMA service after 8-bit data has been received.
4. Go to step 4 until DMA count is "0"

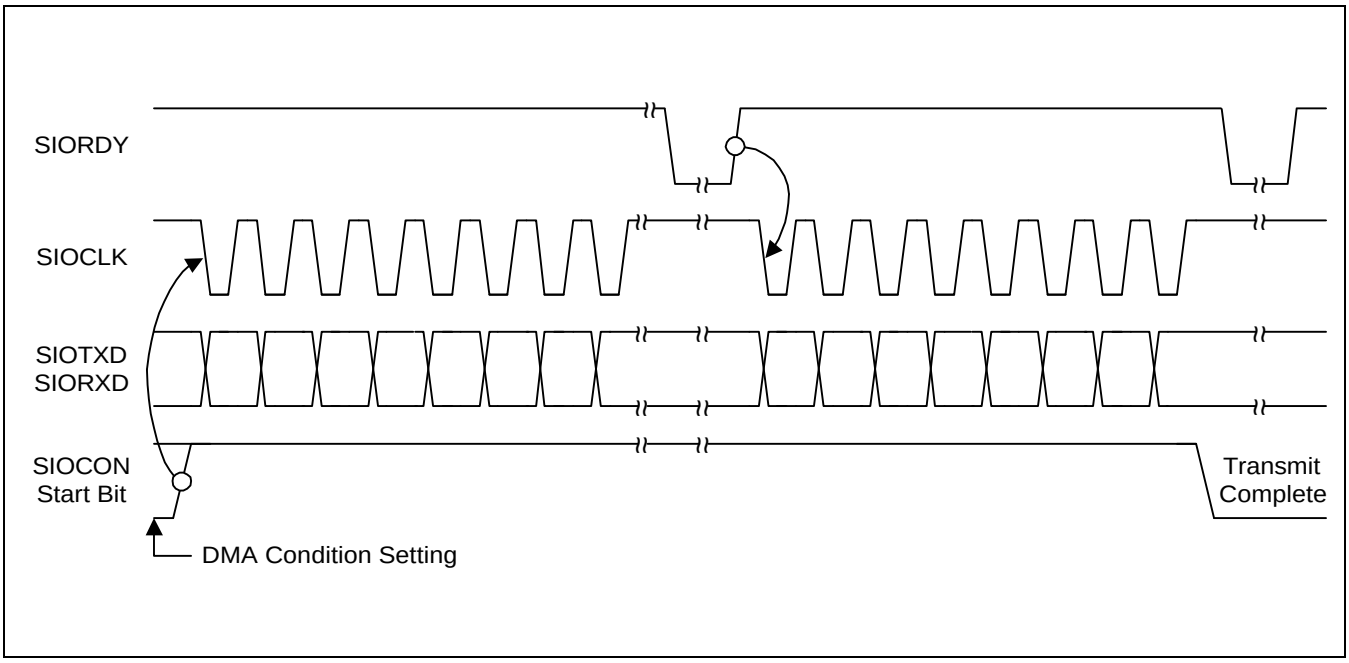


Figure 10-4. Synchronous I/O Timing in Hand-shaking Mode (Flag Run Mode)

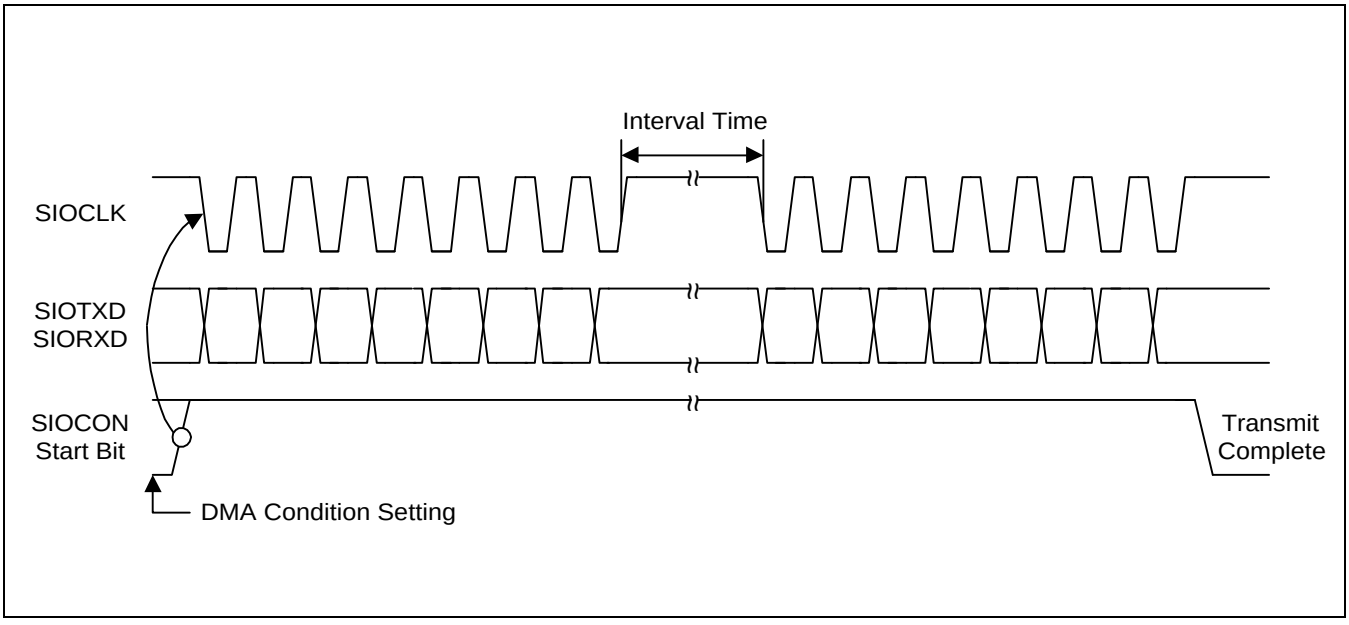


Figure 10-5. Synchronous I/O Timing in Non Hand-shaking Mode (Auto Run Mode)

SYNCHRONOUS I/O SPECIAL FUNCTION REGISTERS

SYNCHRONOUS I/O CONTROL REGISTERS (SIOCON0, SIOCON1)

There are two control register for synchronous I/O interface module, SIOCON0 and SIOCON1. To determine the SIO operation, users should configure a necessary option on this register.

Register	Offset Address	R/W	Description	Reset Value
SIOCON0	0x6003	R/W	Synchronous I/O 0 control register	0x0
SIOCON1	0x7003	R/W	Synchronous I/O 1 control register	0x0

SIOCONx	Bit	Description	Initial State
MODE	[1:0]	**SIO Mode Selection:** This field can determine the SIO operation mode for SIO TX and RX. 00 = Disable 01 = SIO interrupt 10 = DMA0 request 11 = DMA1 request	00
HSE	[2]	**Hand-Shaking Mode Enable:** In DMA-based SIO operation, users can have Hand-Shaking or Non-Hand-Shaking mode. In Hand-Shaking mode, SIO controller should watch SIORDY signal before the sending of next frame. In Non-Hand-Shaking mode, users can have programmable duration between consecutive frames. 0 = Non hand-shaking mode (Auto run mode) 1 = Hand-shaking mode (Flag run mode)	0
SB	[3]	**SIO Start:** To start SIO operation, users should set this bit. 0 = No action 1 = Clear 3-bit counter and start shift	0
CES	[4]	**Clock Edge Select:** This bit can determine what clock edge should be used for serial transmission. If this bit is set to "0" for transmission, the transmitting operation is executed at the falling edge, and the receiving operation is executed at the rising edge. 0 = Falling edge clock to Tx 1 = Rising edge clock to Tx	0
TRS	[5]	**Transmit/Receive Selection:** This bit can decide that the current SIO is configured as receiver only, or transmitter/receiver. The receiver only mode is just for DMA-based receive operation. For DMA-based TX and interrupt-based TX/RX operation, users should have TRS as 1. In case of TX/RX mode, SIO can receive the data from external device without S/W control. In this case, users should determine whether the received data is valid, or not. 0 = Received only mode 1 = Transmit/Receive mode	0
DD	[6]	**Data Direction:** This bit can control whether MSB should be transmitted first or LSB is transmitted first. 0 = MSB mode 1 = LSB mode	0
CS	[7]	**SIO Clock Source Selection:** This bit can determine the clock source for SIO. 0 = Internal Clock 1 = External Clock	0

SIO DATA REGISTERS (SIODAT0, SIODAT1)

To send the data by SIO, users should write the data in the SIO data register (SIODAT0, SIODAT1) before sending. As well as transmission, the received data can also be stored. Even if the receive operation can not be controlled by S/W, the users should decide whether the received data is valid, or not. For example, this decision can be made by the interrupt service routine. If users do not want the received data, users should think the data in SIO data register as dummy one.

Register	Offset Address	R/W	Description	Reset Value
SIODAT0	0x6002	R/W	Synchronous I/O 0 data register	0x0
SIODAT1	0x7002	R/W	Synchronous I/O 1 data register	0x0

SIODATx	Bit	Description	Initial State
SIODATA	[7:0]	**SIO Data:** The field represent the data to be transmitted or received over the SIO channel.	0x0

SIO BAUD RATE PRESCALER REGISTERS (SBRDR0, SBRDR1)

The value stored in the baud rate divisor register (SBRDR0, SBRDR1), allows users to determine the SIO clock rate (baud rate) as follows:

Baud rate = CKIN / { 2 ´ (divisor value + 1) }

Register	Offset Address	R/W	Description	Reset Value
SBRDR0	0x6001	R/W	Synchronous I/O 0 baud rate pre-scale register	0x0
SBRDR1	0x7001	R/W	Synchronous I/O 1 baud rate pre-scale register	0x0

SIODATx	Bit	Description	Initial State
Pre-scale	[7:0]	The field has a pre-scale value for generating the baud rate.	0x0

SIO INTERVAL COUNTER REGISTERS (ITVCNT0, ITVCNT1)

In case of the non hand-shaking mode(auto run mode), users should have the duration between consecutive frames. The duration between frames can be calculated by below formula.

Intervals time(between 8-bit data) = (ITVCNT + 1) ´ 256 ´ 2 / CKIN

Register	Offset Address	R/W	Description	Reset Value
ITVCNT0	0x6000	R/W	Synchronous I/O 0 interval counter register	0x00
ITVCNT1	0x7000	R/W	Synchronous I/O 1 interval counter register	0x00

ITVCNTx	Bit	Description	Initial State
Count value	[7:0]	This field contains the interval time value for the DMA non hand-shaking mode.	0x00

11 INTERRUPT CONTROLLER

OVERVIEW

In S3C3410X, there are 35 interrupt sources. Among them, 23 interrupt sources are coming from internal peripheral devices like the DMA controller, UART, SIO, etc. Other 8 interrupt sources are coming from external interrupt request pins like EINT0, EINT1, EINT2, EINT3, EINT8, EINT9, EINT10, and EINT11. The other 4 are coming from external interrupt request pins like EINT4, EINT5, EINT6, and EINT7. Because these 4 external interrupt requests should be OR-ed internally, we consider these external interrupt request sources as one interrupt request source to CPU. In other word, the total interrupt request sources to CPU is 32, not 35.

Even if there are many interrupt request sources, the ARM7TDMI core can only recognize all interrupt as two kinds of interrupt: a normal interrupt request(IRQ) and a fast interrupt request(FIQ). Therefore, all interrupt sources in S3C3410X should be categorized as either IRQ or FIQ.

The multiple interrupt sources should be controlled by three kind of information in special registers in interrupt controller. These are INTMOD, INTPND, and INTMSK register. The role of three registers in interrupt controller is as follow.

In S3C3410X, the interrupt controller can support the interrupt base vector address as well as programmable priority. To reduce the interrupt latency, the interrupt controller in S3C3410X can assign the hard-wired vector address for each interrupt source. The total 32 interrupt request sources to CPU can have the programmable priority. This kind of programmable priority can make users to have more intelligent interrupt handling.

- **Interrupt Mode Register:** Defines the interrupt mode for each interrupt source, which is IRQ or FIQ. By having the configuration for each interrupt source in this register, users can allocate all interrupt sources as IRQ or FIQ mode interrupt.
- **Interrupt Pending Register:** In CPU core, there is PSR(Processor Status Register) register, which has several field including the interrupt relating I-Flag and F-Flag. As mentioned above, the CPU can accept two kinds of interrupt even if there are many interrupt sources in S3C3410X. That is why all interrupt sources in S3C3410X should be categorized into two mode, which is IRQ mode and FIQ mode. In this case, if CPU is running the service for certain interrupt, and if this interrupt has IRQ mode, the other interrupt sources with IRQ mode can not be serviced until the completion of current service. These interrupt should be pending in IPR(Interrupt Pending Register). In case of FIQ mode, other FIQ interrupt request can not take CPU while the current FIQ service is running as same as IRQ case. Therefore, the FIQ interrupt request should be pending in IPR as same as IRQ. If IRQ interrupt service is running, the FIQ interrupt can take the CPU for service because FIQ has higher priority than IRQ. In other word, ARM CPU can support two level interrupt architecture. The pending interrupt service can start whenever the I-Flag or F-Flag should be cleared to "0". The service routine should clear the pending bit, also.
- **Interrupt Mask Register:** If this mask bit is set, the corresponding interrupt request should be disabled. Users can select the interrupt enable or disable by using this register. For masking(Disable the interrupt), the corresponding mask bit should be "0".

INTERRUPT SOURCE

In S3C3410X, there are 35 interrupt sources. Among them, 23 interrupt sources are coming from internal peripheral devices like the DMA controller, UART, SIO, etc. Other 8 interrupt sources are coming from external interrupt request pins like EINT0, EINT1, EINT2, EINT3, EINT8, EINT9, EINT10, and EINT11. The other 4 are coming from external interrupt request pins like EINT4, EINT5, EINT6, and EINT7. Because these 4 external interrupt requests should be OR-ed internally, we consider these external interrupt request sources as one interrupt request source to CPU. In other word, the total interrupt request sources to CPU is 32, not 35.

Sources	Description	Number
EINT0	External interrupt 0	0
EINT1	External interrupt 1	1
INT_URX	UART receive interrupt	2
INT_UTX	UART transmit interrupt	3
INT_UERR	UART error interrupt	4
INT_DMA0	DMA0 interrupt	5
INT_DMA1	DMA1 interrupt	6
INT_TOF0	Timer 0 overflow interrupt	7
INT_TMC0	Timer 0 match/capture interrupt	8
INT_TOF1	Timer 1 overflow interrupt	9
INT_TMC1	Timer 1 match/capture interrupt	10
INT_TOF2	Timer 2 overflow interrupt	11
INT_TMC2	Timer 2 match/capture interrupt	12
INT_TOF3	Timer 3 overflow interrupt	13
INT_TMC3	Timer 3 match/capture interrupt	14
INT_TOF4	Timer 4 overflow interrupt	15
INT_TMC4	Timer 4 match/capture interrupt	16
INT_BT	Basic Timer interrupt	17
INT_SIO0	SIO 0 interrupt	18
INT_SIO1	SIO 1 interrupt	19
INT_IIC	IIC interrupt	20
INT_RTCA	RTC alarm interrupt	21
INT_RTCT	RTC time interrupt(SEC/MIN/HOUR)	22
INT_TF	Timer4 FIFO interrupt	23
EINT2	External interrupt 2	24
EINT3	External interrupt 3	25
EINT4/5/6/7	External interrupt 4/5/6/7	26
INT_ADC	ADC interrupt	27
EINT8	External interrupt 8	28
EINT9	External interrupt 9	29
EINT10	External interrupt 10	30
EINT11	External interrupt 11	31

NOTE: EINT4, EINT5, EINT6 and EINT7 are sharing the same interrupt request line. So, the ISR(Interrupt Service Routine) can discriminate the interrupt request source by reading the EINTPND register because it has 4-bit for the interrupt source of EINT4, EINT5, EINT6, and EINT7. The EINTPND has to be cleared by writing "0" in ISR

INTERRUPT CONTROLLER SPECIAL FUNCTION REGISTERS

INTERRUPT MODE REGISTER (INTMOD)

Each bit in INTMOD register can determine the interrupt mode of each interrupt request. In case of FIQ mode, this bit should be "1". Otherwise, it means the IRQ mode interrupt. The FIQ mode has higher priority than IRQ mode. During the service of IRQ, the FIQ mode interrupt can occupy the CPU for its service.

Register	Offset Address	R/W	Description	Reset Value
INTMOD	0xc000	R/W	Interrupt mode register 0 = IRQ mode 1 = FIQ mode	0x0

INTMOD	Bit	Description		Initial State
EINT0	[0]	0 = IRQ mode	1 = FIQ mode	0
EINT1	[1]	0 = IRQ mode	1 = FIQ mode	0
INT_URX	[2]	0 = IRQ mode	1 = FIQ mode	0
INT_UTX	[3]	0 = IRQ mode	1 = FIQ mode	0
INT_UERR	[4]	0 = IRQ mode	1 = FIQ mode	0
INT_DMA0	[5]	0 = IRQ mode	1 = FIQ mode	0
INT_DMA1	[6]	0 = IRQ mode	1 = FIQ mode	0
INT_TOF0	[7]	0 = IRQ mode	1 = FIQ mode	0
INT_TMC0	[8]	0 = IRQ mode	1 = FIQ mode	0
INT_TOF1	[9]	0 = IRQ mode	1 = FIQ mode	0
INT_TMC1	[10]	0 = IRQ mode	1 = FIQ mode	0
INT_TOF2	[11]	0 = IRQ mode	1 = FIQ mode	0
INT_TMC2	[12]	0 = IRQ mode	1 = FIQ mode	0
INT_TOF3	[13]	0 = IRQ mode	1 = FIQ mode	0
INT_TMC3	[14]	0 = IRQ mode	1 = FIQ mode	0
INT_TOF4	[15]	0 = IRQ mode	1 = FIQ mode	0
INT_TMC4	[16]	0 = IRQ mode	1 = FIQ mode	0
INT_BT	[17]	0 = IRQ mode	1 = FIQ mode	0
INT_SIO0	[18]	0 = IRQ mode	1 = FIQ mode	0
INT_SIO1	[19]	0 = IRQ mode	1 = FIQ mode	0
INT_IIC	[20]	0 = IRQ mode	1 = FIQ mode	0
INT_RTCA	[21]	0 = IRQ mode	1 = FIQ mode	0

INTMOD	Bit	Description		Initial State
INT_RTCT	[22]	0 = IRQ mode	1 = FIQ mode	0
INT_TF	[23]	0 = IRQ mode	1 = FIQ mode	0
EINT2	[24]	0 = IRQ mode	1 = FIQ mode	0
EINT3	[25]	0 = IRQ mode	1 = FIQ mode	0
EINT4/5/6/7	[26]	0 = IRQ mode	1 = FIQ mode	0
INT_ADC	[27]	0 = IRQ mode	1 = FIQ mode	0
EINT8	[28]	0 = IRQ mode	1 = FIQ mode	0
EINT9	[29]	0 = IRQ mode	1 = FIQ mode	0
EINT10	[30]	0 = IRQ mode	1 = FIQ mode	0
EINT11	[31]	0 = IRQ mode	1 = FIQ mode	0

INTERRUPT PENDING REGISTER (INTPND)

In CPU core, there is PSR(Processor Status Register) register, which has several field including the interrupt relating I-Flag and F-Flag. As mentioned above, the CPU can accept two kinds of interrupt even if there are many interrupt sources in S3C3410X. That is why all interrupt sources in S3C3410X should be categorized into two mode, which is IFQ mode and FIQ mode. In this case, if CPU is running the service for certain interrupt, and if this interrupt has IRQ mode, the other interrupt sources with IRQ mode can not be serviced until the completion of current service. These interrupt should be pending in IPR(Interrupt Pending Register). In case of FIQ mode, other FIQ interrupt request can not take CPU while the current FIQ service is running as same as IRQ case. Therefore, the FIQ interrupt request should be pending in IPR as same as IRQ. If IRQ interrupt service is running, the FIQ interrupt can take the CPU for service because FIQ has higher priority than IRQ. In other word, ARM CPU can support two level interrupt architecture. The pending interrupt service can start whenever the I-Flag or F-Flag should be cleared to "0". The service routine should clear the pending bit, also.

Register	Offset Address	R/W	Description	Reset Value
INTPND	0xc004	R/W	Interrupt pending register. Indicates the interrupt request status of each source. 0 = The interrupt has not been requested (when reading) *0 = Clear pending bit (when writing)* 1 = The interrupt source has asserted the interrupt request (when reading) *1 = No effect, keeping current status, '0' or '1'. (when writing)*	0x0

INTPND	Bit	Description		Initial State
EINT0	[0]	0 = Not requested	1 = Requested	0
EINT1	[1]	0 = Not requested	1 = Requested	0
INT_URX	[2]	0 = Not requested	1 = Requested	0
INT_UTX	[3]	0 = Not requested	1 = Requested	0
INT_UERR	[4]	0 = Not requested	1 = Requested	0
INT_DMA0	[5]	0 = Not requested	1 = Requested	0
INT_DMA1	[6]	0 = Not requested	1 = Requested	0
INT_TOF0	[7]	0 = Not requested	1 = Requested	0
INT_TMC0	[8]	0 = Not requested	1 = Requested	0
INT_TOF1	[9]	0 = Not requested	1 = Requested	0
INT_TMC1	[10]	0 = Not requested	1 = Requested	0
INT_TOF2	[11]	0 = Not requested	1 = Requested	0
INT_TMC2	[12]	0 = Not requested	1 = Requested	0
INT_TOF3	[13]	0 = Not requested	1 = Requested	0
INT_TMC3	[14]	0 = Not requested	1 = Requested	0
INT_TOF4	[15]	0 = Not requested	1 = Requested	0
INT_TMC4	[16]	0 = Not requested	1 = Requested	0
INT_BT	[17]	0 = Not requested	1 = Requested	0

INTPND	Bit	Description		Initial State
INT_SIO0	[18]	0 = Not requested	1 = Requested	0
INT_SIO1	[19]	0 = Not requested	1 = Requested	0
INT_IIC	[20]	0 = Not requested	1 = Requested	0
INT_RTCA	[21]	0 = Not requested	1 = Requested	0
INT_RTCT	[22]	0 = Not requested	1 = Requested	0
INT_TF	[23]	0 = Not requested	1 = Requested	0
EINT2	[24]	0 = Not requested	1 = Requested	0
EINT3	[25]	0 = Not requested	1 = Requested	0
EINT4/5/6/7	[26]	0 = Not requested	1 = Requested	0
INT_ADC	[27]	0 = Not requested	1 = Requested	0
EINT8	[28]	0 = Not requested	1 = Requested	0
EINT9	[29]	0 = Not requested	1 = Requested	0
EINT10	[30]	0 = Not requested	1 = Requested	0
EINT11	[31]	0 = Not requested	1 = Requested	0

INTERRUPT MASK REGISTER (INTMSK)

The interrupt mask register has interrupt mask bits for all interrupt source. When an interrupt source mask bit is "0", the corresponding interrupt can not be serviced by the CPU when the corresponding interrupt request is generated. If the mask bit is "1", the interrupt service can be done.

Register	Offset Address	R/W	Description	Reset Value
INTMSK	0xc008	R/W	Interrupt mask register. Each bit can disable or enable the corresponding interrupt request. 0 = Interrupt service is masked or disabled. 1 = Interrupt service is available	0x0

INTMSK	Bit	Description		Initial State
EINT0	[0]	0 = Masked	1 = Service available	0
EINT1	[1]	0 = Masked	1 = Service available	0
INT_URX	[2]	0 = Masked	1 = Service available	0
INT_UTX	[3]	0 = Masked	1 = Service available	0
INT_UERR	[4]	0 = Masked	1 = Service available	0
INT_DMA0	[5]	0 = Masked	1 = Service available	0
INT_DMA1	[6]	0 = Masked	1 = Service available	0
INT_TOF0	[7]	0 = Masked	1 = Service available	0
INT_TMC0	[8]	0 = Masked	1 = Service available	0
INT_TOF1	[9]	0 = Masked	1 = Service available	0
INT_TMC1	[10]	0 = Masked	1 = Service available	0
INT_TOF2	[11]	0 = Masked	1 = Service available	0
INT_TMC2	[12]	0 = Masked	1 = Service available	0
INT_TOF3	[13]	0 = Masked	1 = Service available	0
INT_TMC3	[14]	0 = Masked	1 = Service available	0
INT_TOF4	[15]	0 = Masked	1 = Service available	0
INT_TMC4	[16]	0 = Masked	1 = Service available	0
INT_BT	[17]	0 = Masked	1 = Service available	0
INT_SIO0	[18]	0 = Masked	1 = Service available	0
INT_SIO1	[19]	0 = Masked	1 = Service available	0
INT_IIC	[20]	0 = Masked	1 = Service available	0
INT_RTCA	[21]	0 = Masked	1 = Service available	0

INTMSK	Bit	Description		Initial State
INT_RTCT	[22]	0 = Masked	1 = Service available	0
INT_TF	[23]	0 = Masked	1 = Service available	0
EINT2	[24]	0 = Masked	1 = Service available	0
EINT3	[25]	0 = Masked	1 = Service available	0
EINT4/5/6/7	[26]	0 = Masked	1 = Service available	0
INT_ADC	[27]	0 = Masked	1 = Service available	0
EINT8	[28]	0 = Masked	1 = Service available	0
EINT9	[29]	0 = Masked	1 = Service available	0
EINT10	[30]	0 = Masked	1 = Service available	0
EINT11	[31]	0 = Masked	1 = Service available	0

INTERRUPT VECTOR BASE ADDRESS

To reduce the interrupt latency, the S3C3410X can support the concept of interrupt vector base address. The interrupt vector base address means the start address of corresponding service routine. In other word, as soon as CPU recognize the interrupt request, there will be Branch to fixed hardwired vector. But, because the CPU can support just two interrupt mode of FIQ and IRQ, we need special technique to assign the specific base vector address for all interrupt source. To do this, the interrupt controller should give the Branch Instruction (Branch to the fixed hardware vector address) to CPU as soon as the CPU recognize the interrupt request. Because the interrupt controller can know the interrupt mode as well as source, the interrupt controller can give the specific vector address to CPU by H/W. Following table shows the interrupt base vector address for each interrupt source.

Sources	Address	Sources	Address
EINT0	0x80	INT_TMC4	0xc0
EINT1	0x84	INT_BT	0xc4
INT_URX	0x88	INT_SIO0	0xc8
INT_UTX	0x8c	INT_SIO1	0xcc
INT_UERR	0x90	INT_IIC	0xd0
INT_DMA0	0x94	INT_RTCA	0xd4
INT_DMA1	0x98	INT_RTCT	0xd8
INT_TOF0	0x9c	INT_TF	0xdc
INT_TMC0	0xa0	EINT2	0xe0
INT_TOF1	0xa4	EINT3	0xe4
INT_TMC1	0xa8	EINT4/5/6/7	0xe8
INT_TOF2	0xac	INT_ADC	0xec
INT_TMC2	0xb0	EINT8	0xf0
INT_TOF3	0xb4	EINT9	0xf4
INT_TMC3	0xb8	EINT10	0xf8
INT_TOF4	0xbc	EINT11	0xfc

INTERRUPT PRIORITY REGISTER (INTPRI)

INTPRIn are the interrupt priority register to determine the priority of interrupt sources. There should be 32 grade priorities for interrupt sources because there are total 32 interrupt request sources in S3C3410X. It means that the 5-bit register can determine all the priorities of 32 interrupt request sources. So, each INTPRIn is divided into four part to set the priority of the each interrupt source, that is, INTPRI0 is divided into PRIORITY3, PRIORITY2, PRIORITY1, and PRIORITY0. Lower number has the higher priority than the higher number, that is, PRIORITY0 has the higher priority than PRIORITY3. So, for determining the priority of interrupt sources, first set the interrupt number in PRIORITYn only. For example, if PROIRITY0 have 0x11, which is basic timer interrupt number, then basic timer interrupt have the highest priority in all interrupt sources.

As previously mentioned, there are two kinds of interrupt mode in CPU. One is FIQ and the other is IRQ. The FIQ has the higher priority than IRQ in CPU. So, if you want to set the priority of FIQ and IRQ, you must set the priority of FIQ higher than that of IRQ. In summary, the FIQ must have the higher priority than IRQ, interrupt source in low number priority register has the higher priority than interrupt source in high number priority register. In addition, all 32 PRIORITYn should have a different interrupt source number.

Register	Offset Address	R/W	Description	Reset Value
INTPRI0	0xc00c	R/W	Interrupt priority register 0	0x03020100
INTPRI1	0xc010	R/W	Interrupt priority register 1	0x07060504
INTPRI2	0xc014	R/W	Interrupt priority register 2	0x0b0a0908
INTPRI3	0xc018	R/W	Interrupt priority register 3	0x0f0e0d0c
INTPRI4	0xc01c	R/W	Interrupt priority register 4	0x13121110
INTPRI5	0xc020	R/W	Interrupt priority register 5	0x17161514
INTPRI6	0xc024	R/W	Interrupt priority register 6	0x1b1a1918
INTPRI7	0xc028	R/W	Interrupt priority register 7	0x1f1e1d1c

Register		[28:24]		[20:16]		[12:8]		[4:0]
INTPRI0	EN	PRIORITY3	X	PRIORITY2	X	PRIORITY1	X	PRIORITY0
INTPRI1	X	PRIORITY7	X	PRIORITY6	X	PRIORITY5	X	PRIORITY4
INTPRI2	X	PRIORITY11	X	PRIORITY10	X	PRIORITY9	X	PRIORITY8
INTPRI3	X	PRIORITY15	X	PRIORITY14	X	PRIORITY13	X	PRIORITY12
INTPRI4	X	PRIORITY19	X	PRIORITY18	X	PRIORITY17	X	PRIORITY16
INTPRI5	X	PRIORITY23	X	PRIORITY22	X	PRIORITY21	X	PRIORITY20
INTPRI6	X	PRIORITY27	X	PRIORITY26	X	PRIORITY25	X	PRIORITY24
INTPRI7	X	PRIORITY31	X	PRIORITY30	X	PRIORITY29	X	PRIORITY28

INTPRI0	Bit	Description	Initial State
EN	[31:29]	000 = Disable interrupt priority other = Enable interrupt priority	000
PRIORITY N	5-bit	The priority number for interrupt request source N.	
X	3-bit	Do not care field.	

NOTES:

1. To use the programmable priority, set EN to 000b, then the priority should be determined by SW.
2. The PRIORITYn determines the priority of the corresponding interrupt source.
3. The highest priority is PRIORITY0, and the lowest priority is PRIORITY31.

NOTES

12 A/D CONVERTER

OVERVIEW

The 10-bit CMOS A/D converter in S3C3410X has a 8-channel analog input multiplexer, auto offset calibration comparator, high resolution R-string DAC, clock generator, successive approximation register(SAR), ADC control register(ADCCON), and tri-state output register (ADCDAT). This well trimmed ADC architecture can give high accurate conversion result. In addition to accurate conversion result, users can have the power down mode of ADC to reduce the power consumption when users do not use the ADC.

FEATURE

- Resolution: 10-bit
- Differential Linearity Error: ± 1 LSB
- Integral Linearity Error: ± 1 LSB
- Maximum Conversion Rate: 500KSPS
- Low Power Consumption: 3.3 mW (Typical) @ normal operation mode
  330 nW (Typical) @ standby mode
- No missing code

A/D CONVERTER OPERATION

BLOCK DIAGRAM

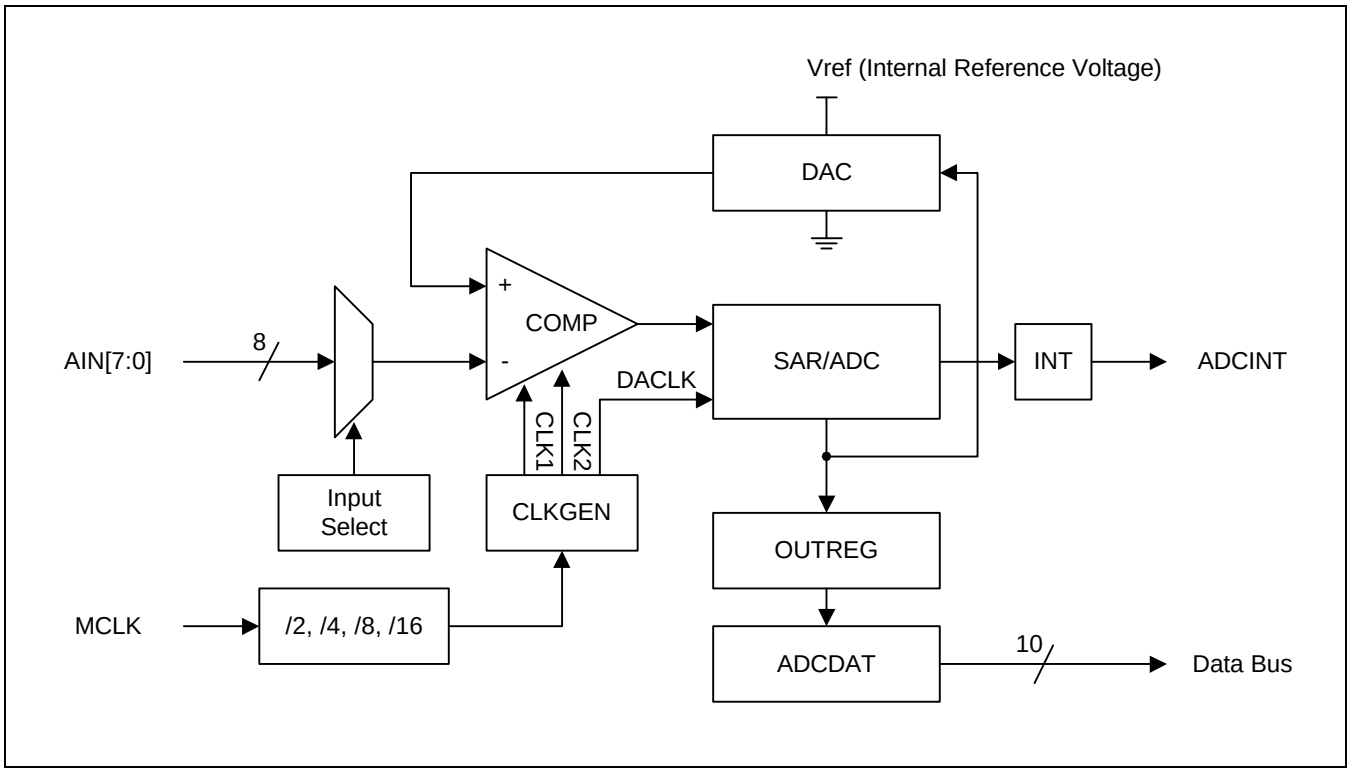


Figure 12-1. A/D Converter Block Diagram

FUNCTION DESCRIPTION

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. The conversion process of basic SAR-type ADC is as follow. After sampling analog input, the MSB is switched on to generate the D/A output of half reference voltage 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 and next bit is switched on to generate the D/A output contributing quarter reference voltage. It means that there will be comparison between analog input and the 0.75 reference voltage. When the input signal is smaller than the output signal of the D/A converter, then the MSB is off and next bit is switched on to generate the D/A output contributing quarter reference voltage. It means that there will be comparison between analog input and the 0.25 reference voltage. and a comparison will be performed. By having this comparison algorithm, we need 10 times comparison to determine the 10bit result. Usually, SAR algorithm need n-step to determine the n-bit, which has linear complexity.

Comparator and DAC (Digital to Analog Converter)

The CMOS comparator can produce the digital output as the result of comparison between analog input and reference voltage by the assumed digital code. This comparator need internal non-overlapping clock to reduce the error effect during the conversion process. Especially, the D/A converter consists of 128 resistor strings and switches array to cover 7-bit resolution. So, the comparator should perform the comparison with 3-bit resolution. The D/A converter can generate 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 analog ground which should be applied to the comparator and the D/A converter block.

Clock Divider and Clock Generator (CLKGEN)

The clock divider block of the A/D converter can generate the necessary clock for ADC. The clock rate for ADC can eventually indicate the conversion speed of ADC. To get the reliable accuracy of ADC, users should configure the proper clock rate of ADC. The ADC clock can be achieved by slowing down the MCLK. The CKSEL field in ADCCON register can select the necessary clock for ADC. These options are MCLK/2, MCLK/4, MCLK/8, or MCLK/16. Internally, ADC should generate the necessary clock based on master clock. For example, we need non-overlapping clock for the operation of ADC.

NOTE: The maximum frequency into CLKGEN is 20MHz. In other word, the MCLK/X should be less than 20MHz.

A/D Conversion Time

The number of cycle of CLKGEN (Input clock of clock generator for ADC) which needs for complete conversion of 10-bit resolution, is 45.

So, the minimum A/D conversion time at MCLK=40MHz is calculated as follows if users take the division factor of 2:

40MHz / 2 (divide 2 frequency) / 45 = 444.4KHz = 2.25us

If MCLK is 25MHz, the minimum A/D conversion time is calculated as follows:

25MHz / 2 / 45 = 277.8KHz = 3.6us

NOTE: In the above calculated A/D conversion time, the CPU access time is omitted. If the CPU access time is considered, the maximum conversion rate will be about 500KSPS.

Standby Mode

When users need the reduction of power for ADC, users can set the STBY "1". In this case, the A/D converter should be kept in standby mode without an A/D conversion operation and it can eliminate the power consumption, also

NOTE: If STBY is applied during A/D conversion, the FLAG bit goes HIGH immediately.

A/D CONVERTER SPECIAL REGISTERS

A/D CONVERTER CONTROL REGISTER (ADCCON)

Register	Offset Address	R/W	Description	Reset Value
ADCCON	0x8002	R/W	A/D Converter control register	0x140

ADCCON	Bit	Description	Initial State
ADEN	[0]	**A/D Enable:** This bit can determine enable/disable for ADC. 0 = No operation 1 = Start A/D conversion. This bit will be cleared automatically after conversion start.	0
ASEL	[3:1]	**Analog Input Select:** This field can determine the channel of analog input. 000 = AIN0 001 = AIN1 010 = AIN2 011 = AIN3 100 = AIN4 101 = AIN5 110 = AIN6 111 = AIN7	000
CLKSEL	[5:4]	**Clock Source:** This field can determine the input clock of clock generator (CLKGEN) 00 = MCLK / 16 01 = MCLK / 8 10 = MCLK / 4 11 = MCLK / 2	00
STBY	[6]	**Standby Mode:** System power down mode select. 0 = Normal operation 1 = Power down mode	1
MODE	[7]	**Conversion Mode:** 10-bit/8-bit mode. 0 = 10-bit operation 1 = 8-bit operation	0
FLAG	[8]	A/D converter state flag (Read Only) 0 = A/D conversion in process 1 = End of A/D conversion	1

A/D CONVERTER DATA REGISTER (ADCDAT)

When A/D conversion is finished, the conversion result can be read from the ADCDAT register. The ADCDAT register should be read after the conversion is finished.

Register	Offset Address	R/W	Description	Reset Value
ADCDAT	0x8006	R/W	A/D Converter data register	–

ADCDAT	Bit	Description	Initial State
DATA	[9:0]	A/D converter output	–

NOTE: If MODE bit in the ADCCON register set to "1", ADCDAT[1:0] isn't a valid value. Instead, the maximum conversion rate will be 650KSPS.

NOTES

13 BASIC TIMER & WATCHDOG TIMER

OVERVIEW

The S3C3410X has internal Basic Timer/Watchdog Timer. This kind of timer can be used to resume the controller operation when it is disturbed due to noise, system error, or other kinds of malfunction. To have a configuration on Watchdog Timer, the overflow signal from 8-bit Basic Timer should be fed to the clock input of 3-bit Watchdog Timer as shown in below figure. User can enable or disable the Watchdog Timer by software, i.e., by controlling the configuration in BTCON register. If users do not want to use the configuration of Watchdog Timer, the 8-bit Basic Timer can only be used as a normal interval timer to request the interrupt service. Also, it works to signal the end of the required oscillation interval after a reset or Stop mode release. For example, the Basic Timer can give the overflow signal to necessary logic blocks after a reset or release from Stop mode. In this case, the overflow signal from Basic Timer can guarantee the necessary time delay for stable clock from external oscillator circuit.

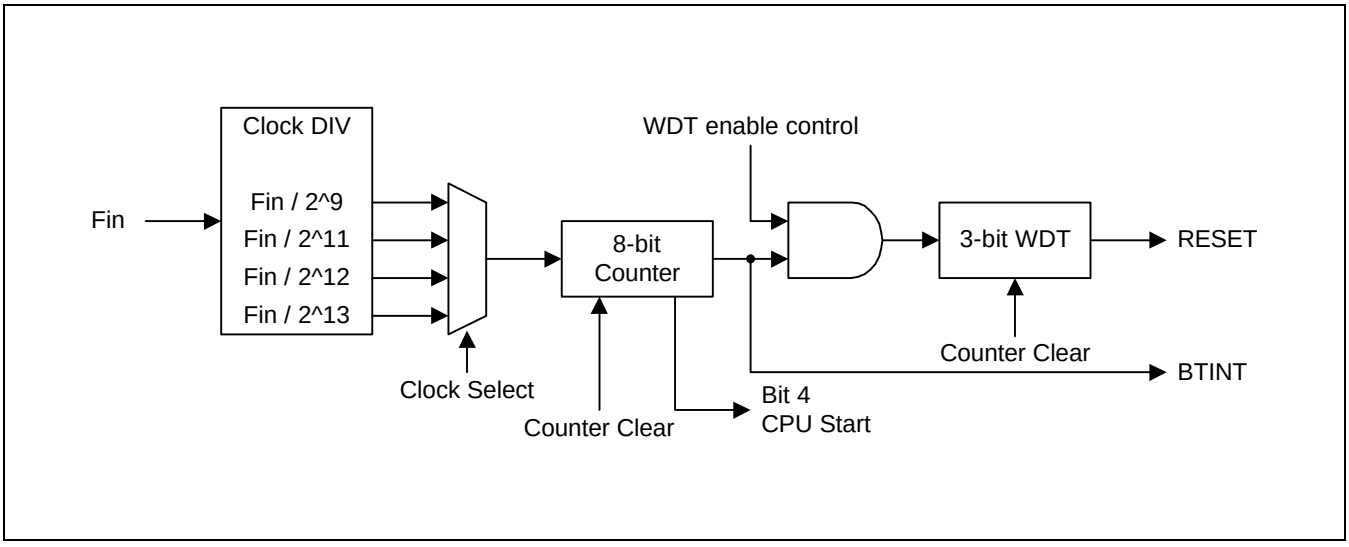


Figure 13-1. Basic Timer Block Diagram

FUNCTION DESCRIPTION

Interval Timer Function

The primary function of Basic Timer is to measure the elapsed time between events. The standard time interval is equal to 256 basic timer clock pulses, which is an overflow signal from 8-bit Basic Timer.

The content of 8-bit counter register, BTCNT, increases it content every when a clock signal is detected which corresponds to the frequency selected by BTCON. The BTCNT continues its counting until an overflow occurs, i.e., the content reaches to 255. An overflow can cause the BT interrupt pending flag to be set, which signals that the designated time interval has elapsed. In this case, when an interrupt request is generated, BTCNT is cleared to all zero, and the counting continues from 00h, again.

Oscillation Stabilization Using Interval Timer Function

Users can use the Basic Timer to have programmable delay time, which is necessary for stabilizing the clock signal from oscillator circuit after reset or Stop mode release.

When the S3C3410X is in Stop mode, the reset or external interrupt request can wake up the S3C3410X. Please understand that the oscillator circuit is in disable state when the S3C3410X is in Stop mode. In case of wake-up by reset, the oscillator should start first. Because the default clock division ratio is Fin / 2^13, the Fin / 2^13 clock will be fed to the 8-bit Basic Timer. When an overflow occurs from Bit 4 of BTCNT register(Not using 8-bit, but 4-bit of Basic Timer), this kind of overflow signal can release the clock blocking to CPU. In other word, the normal clock can be bed to S3C3410X when an overflow of Bit 4 in Basic Timer. In case of wake-up by external interrupt request, the only difference from reset, is clock division ratio. While we should use the default value of clock division ratio for the case of wake-up by reset, we use the pre-defined value of clock division ration before entering Stop mode for the case of wake-up by external interrupt request. In any case, the CPU can resume its operation when normal clock can be fed to the blocks in S3C3410X.

In summary, please take following sequence for releasing S3C3410X from Stop mode:

1. When S3C3410X is in Stop mode, the escape from Stop mode can be made by a power-on reset or an external interrupt. At same time, the oscillator can start its oscillation.
2. In case of wake-up by power-on reset, the Basic Timer will increase its content(BTCNT) at the rate of Fin / 2^13, which is the default rate of clock division ration. In case of wake-up by external interrupt request, the Basic Timer will increase its content(BTCNT) at the rate of preset value, which is written before entering into Stop mode.
3. The normal clock from oscillator will be delayed to be fed to all logic blocks inside S3C3410X until the 4^{th} bit of Basic Timer is generated. It means that user can use the Basic Timer to guarantee the stable clock from oscillator, i.e., waiting up to stable oscillation.
4. When the normal clock can be fed to S3C3410X, the S3C3410X can resume the operation.

Watchdog Timer Operation

The Basic Timer can also be used as a "Watchdog" Timer to recover the S3C3410X from the unexpected program sequence, that is, system or program operation error due to external factor. For example, the external noise can cause this kind of situation, which means that the CPU is running the unexpected code sequence, i.e., malfunction of CPU. To recover the CPU from the unexpected sequence, the Watchdog Timer should reset the CPU in case of malfunction. But, during normal sequence, the instruction which clear the Watchdog before the overflow of Watchdog Timer (Within a given period) should be executed at the proper points in a program. If this instruction can be executed in certain circumstance, it means the overflow of Watchdog Timer and it can generate the internal reset signal generation to restart the CPU from the beginning. In summary, an operation of Watchdog Timer is as follows:

- Each time BTCNT overflows, an overflow signal should be sent to the Watchdog Timer Counter, WDTCNT.
- If WDTCNT overflows, system reset should be generated.

NOTE

A reset signal can clear the BTCON as 0x0000. This value can enable the Watchdog Timer because it is not 0xA5(Please understand the Watchdog Timer can be disable when its content(WDTE field in BTCON[15:8] register) is 0xA5). For normal program sequence, the application program should prevent the overflow. To do this, the WDTCNT value should be cleared(by writing a "1" to WDTC bit of the Basic Timer Control Register(BTCON[0])) before the overflow occurs.

BASIC TIMER DURATION

The Basic Timer Counter, BTCNT, can be used to specify the time-out duration, and is a free-running 8-bit counter. Please keep below table as reference for duration of timer. This is the case when the external clock is 40Mhz.

Clock Source	Resolution	Interval Time
Fin / 2^9 (Fin = 40MHz)	12.8 us	2^9 $\times$ 2^8 / Fin = 3.277 ms
Fin / 2^{11} (Fin = 40MHz)	51.2 us	$2^{11} \times 2^8$ / Fin = 13.107 ms
Fin / 2^{12} (Fin = 40MHz)	102.4 us	$2^{12} \times 2^8$ / Fin = 26.214 ms
Fin / 2^{13} (Fin = 40MHz)	204.8 us	$2^{13} \times 2^8$ / Fin = 52.429 ms

WATCHDOG TIMER DURATION

The Watchdog Timer Counter, WTCNT, can be used to specify the time-out duration and is a free-running 3-bit counter. To enable Watchdog Timer, user should write the data in BTCON[15:8] register except 0xA5. In case of 0xA5, it will disable the Watchdog Timer. After writing certain value in BTCON[15:8] except 0xA5, there will be a system reset if the overflow occurs.

Clock Source	Basic Timer Interval	Watchdog Timer Interval Time
Fin / 2^9 (Fin = 40MHz)	3.277 ms	$2^9 \times 2^8 \times 2^3$ / Fin = 26.214 ms
Fin / 2^{11} (Fin = 40MHz)	13.107 ms	$2^{11} \times 2^8 \times 2^3$ / Fin = 104.858 ms
Fin / 2^{12} (Fin = 40MHz)	26.214 ms	$2^{12} \times 2^8 \times 2^3$ / Fin = 209.715 ms
Fin / 2^{13} (Fin = 40MHz)	52.429 ms	$2^{13} \times 2^8 \times 2^3$ / Fin = 419.430 ms

BASIC TIMER & WATCHDOG TIMER SPECIAL REGISTERS

BASIC TIMER CONTROL REGISTER (BTCON)

The basic timer control register contains Watchdog counter enable bits, clock input bits, and counter clear bit.

Register	Offset Address	R/W	Description	Reset Value
BTCON	0xa002	R/W	Basic Timer Control register	0x0

BTCON	Bit	Description	Initial State
WDTC	[0]	**Watchdog Timer Clear:** This bit can clear the Watchdog Timer Counter. When this bit is set, the Watchdog Timer Counter will be cleared to all zero.	0
BTC	[1]	**Basic Timer Clear:** This bit can clear the Basic Timer Counter. When this bit set, the Basic Timer Counter will be cleared to all zero.	0
CS	[3:2]	**Clock Source Select:** This field can select a clock source. 00 = Fin / 2^{13} 01 = Fin / 2^{12} 10 = Fin / 2^{11} 11 = Fin / 2^{9}	00
Reserved	[7:4]	Reserved	0000
WDTE	[15:8]	**Watchdog Timer Enable:** This field can control to enable or disable a Watchdog Timer Counter. When this field is 0xA5, Watchdog Timer Counter will be stopped. The other value except 0xA5 can enable a Watchdog Timer Counter, and make a system reset when the overflow signal occurs.	00000000

BASIC TIMER COUNT REGISTER (BTCNT)

Register	Offset Address	R/W	Description	Reset Value
BTCNT	0xa007	R	Basic Timer count register	0x0

BTCNT	Bit	Description	Initial State
CV	[7:0]	Count value	0x00

NOTES

14 IIC-BUS INTERFACE

OVERVIEW

The S3C3410X RISC microprocessor can support a multi-master IIC-bus serial interface. A dedicated serial data line(SDA) and a serial clock line(SCL) can carry information between bus masters and peripheral devices which are connected to the IIC-bus. The SDA and SCL lines are bi-directional.

In multi-master IIC-bus mode, multiple S3C3410X RISC microprocessor can receive or transmit the serial data to or from slave devices. The master S3C3410X which can initiate a data transfer over the IIC-bus, is responsible for terminating the transfer. Standard bus arbitration procedure is used in this IIC-bus in S3C3410X.

When the IIC-bus is free, the SDA and SCL lines should be both at High level. A High-to-Low transition of SDA can initiate a Start condition. A Low-to-High transition of SDA can initiate a Stop condition while SCL remains steady at High Level.

The Start and Stop conditions can always be generated by the master devices. A 7-bit address value in the first data byte which is put onto the bus after the Start condition is initiated, can determine the slave device which the bus master device has selected. The 8th bit determines the direction of the transfer (read or write).

Every data byte that is put onto the SDA line should be total eight bits. The number of bytes which can be sent or received during the bus transfer operation is unlimited. Data is always sent from most-significant bit (MSB) first and every byte should be immediately followed by an acknowledge (ACK) bit.

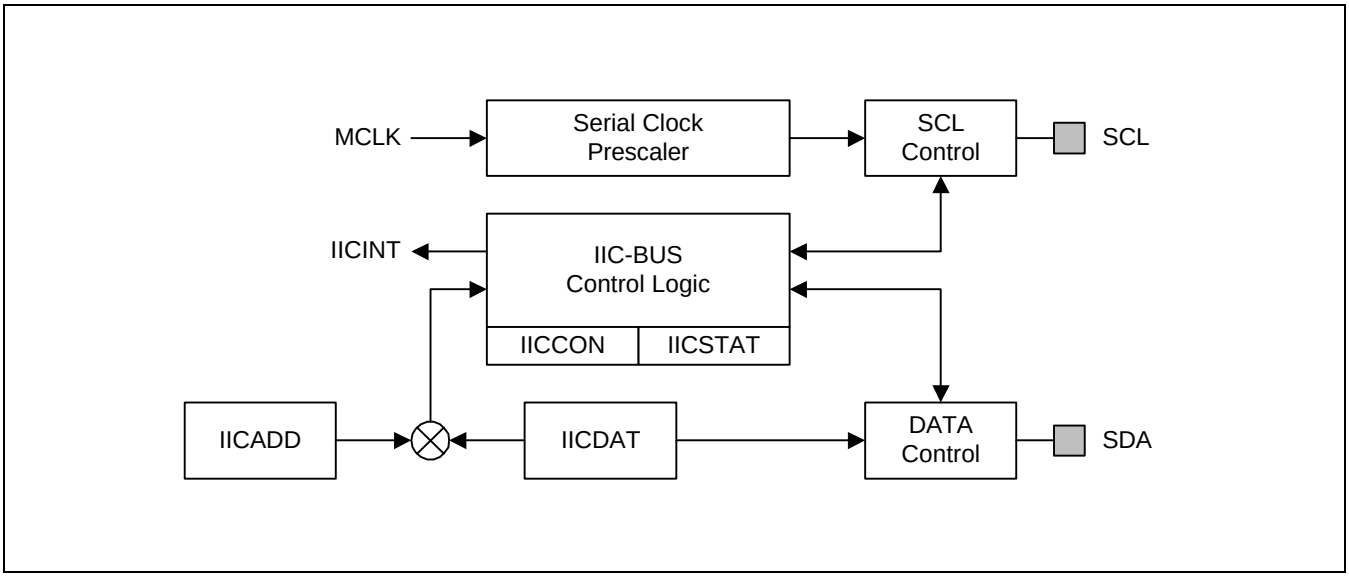


Figure 14-1. IIC-Bus Block Diagram

IIC_BUS OPERATION

THE IIC-BUS INTERFACE

The S3C3410X IIC-bus interface has four operation modes:

— Master transmitter mode
— Master receive mode
— Slave transmitter mode
— Slave receive mode

Functional relationships among these operating modes are described below.

START AND STOP CONDITIONS

When the IIC-bus interface is in inactive state, it is usually in slave mode. In other word, the state of interface should be in slave mode before detecting a Start condition on the SDA line. (A Start condition can be initiated by having a High-to-Low transition of the SDA line while the clock signal of SCL is High) When the state of interface is changed into the master mode, it can initiate a data transfer on the SDA line as well as generating the SCL signal.

A Start condition can initiate a one-byte serial data transfer over the SDA line and stop condition can indicate the termination of data transfer. A Stop condition is a Low-to-High transition of the SDA line while SCL is High. Start and Stop conditions are always generated by the master. The IIC-bus is busy when a start condition is generated. A few clocks after a stop condition, the IIC-bus will be free, again.

When a master initiates a Start condition, it should send slaver address to give a notice to the slaver device. The one byte of address field consist of a 7-bit address and a 1-bit transfer direction indicator (that is, write or read). If bit 8 is 0, it indicate a write operation(transmit operation). If bit 8 is 1, it indicate a request for data read(receive operation).

The master will finish the transfer operation by transmitting a Stop condition. If the master want to continue the data transmission the bus, it should generate another Start condition as well as slave address. In this way, the read-write operation can be performed in various format.

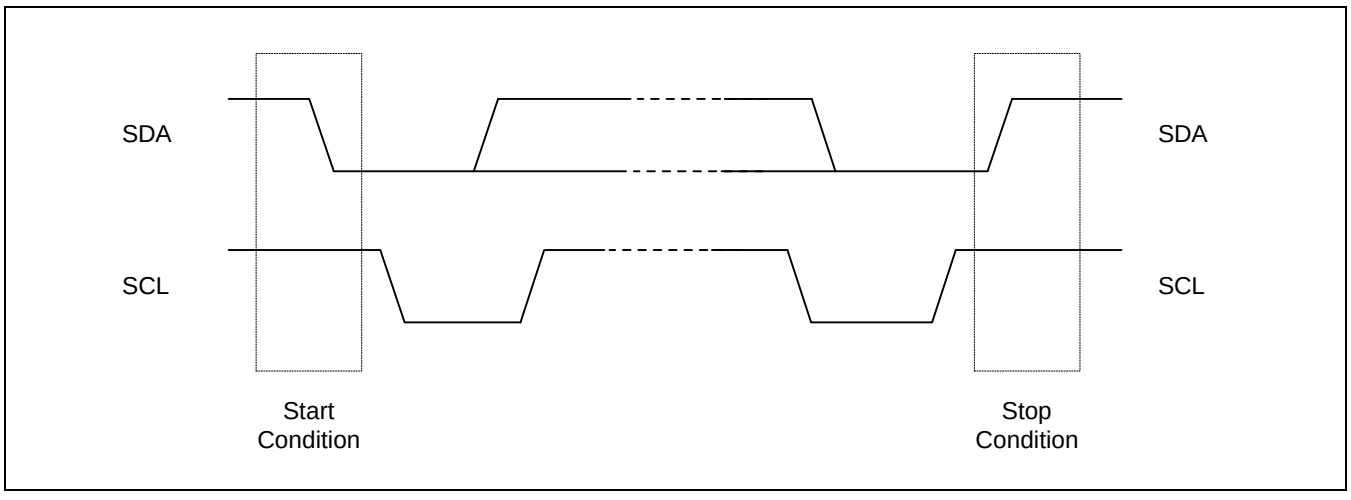
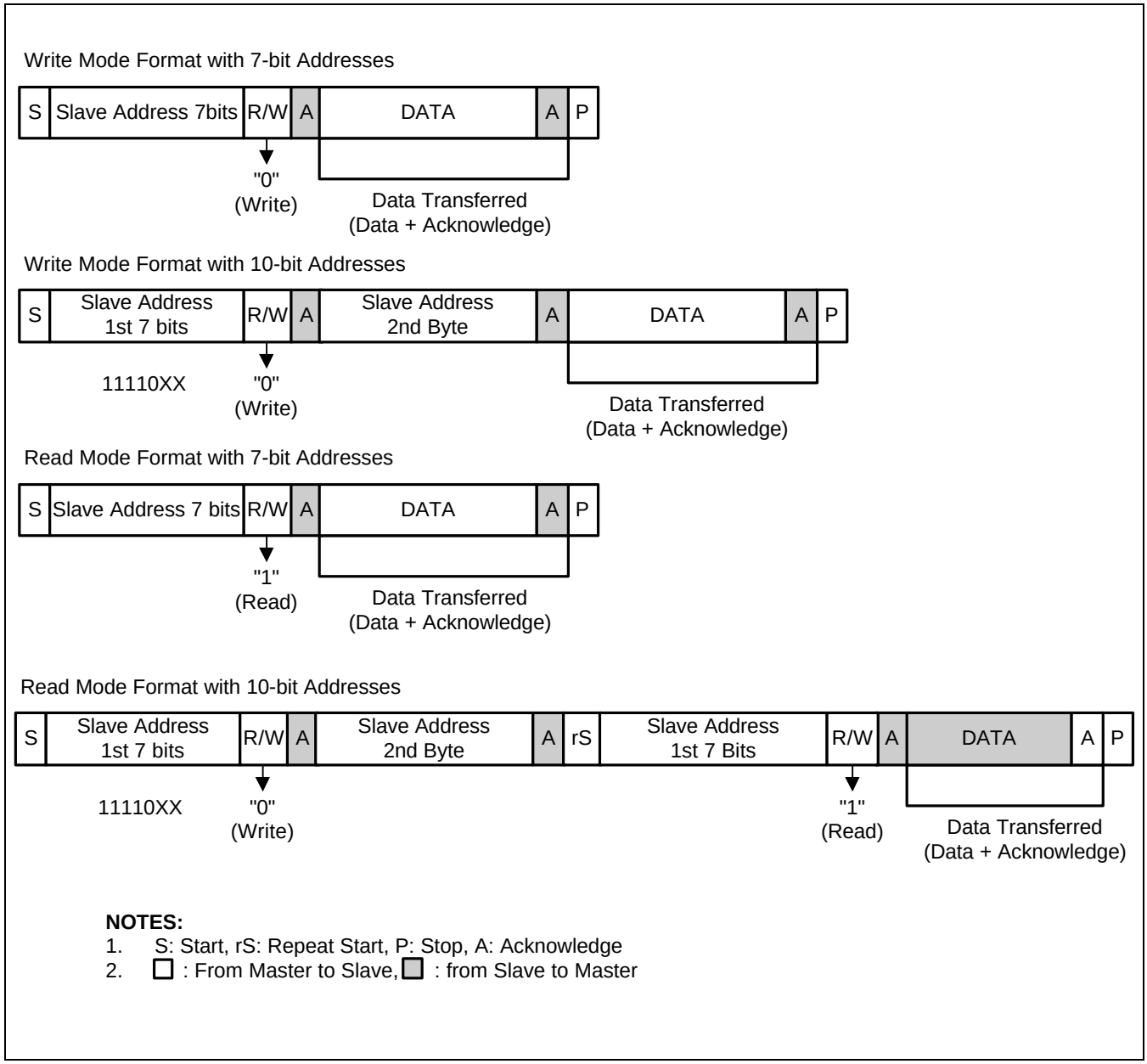


Figure 14-2. Start and Stop Condition

DATA TRANSFER FORMAT

Every byte put on the SDA line should have eight bits in length. The number of bytes which can be transmitted per transfer is unlimited. The first byte following a start condition should have the address field. The address field can be transmitted by the master when the IIC-bus is operating in master mode. Each byte should be followed by an acknowledge (ACK) bit. The MSB bit of Serial data and addresses are always sent first.



NOTES:
1. S: Start, rS: Repeat Start, P: Stop, A: Acknowledge
2. ☐ : From Master to Slave, ▩ : from Slave to Master

Figure 14-3. IIC-Bus Interface Data Format

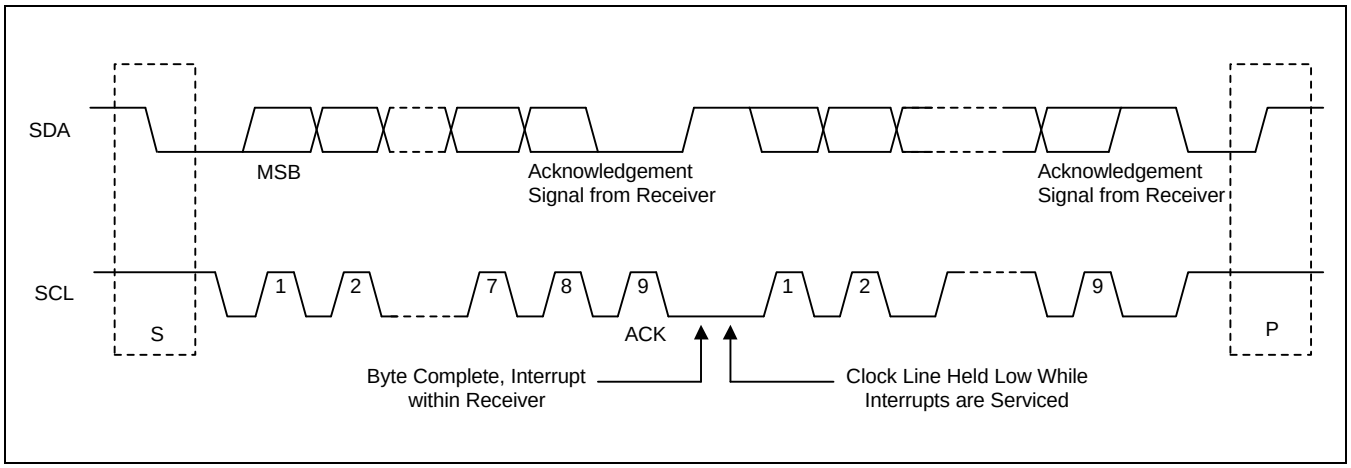


Figure 14-4. Data Transfer on the IIC-Bus

ACK SIGNAL TRANSMISSION

To finish a one-byte transfer operation completely, the receiver should send an ACK bit to the transmitter. The ACK pulse should occur at the ninth clock of the SCL line. Eight clocks are required for the one-byte data transfer. The clock pulse required for the transmission of the ACK bit, should be generated by the master.

The transmitter should release the SDA line by making the SDA line High when the ACK clock pulse is received. The receiver should also drive the SDA line Low during the ACK clock pulse so that the SDA is Low during the High period of the ninth SCL pulse.

The ACK bit transmit function can be enable or disable by software (IICSTAT). However, the ACK pulse on the ninth clock of SCL is required to complete a one-byte data transfer operation.

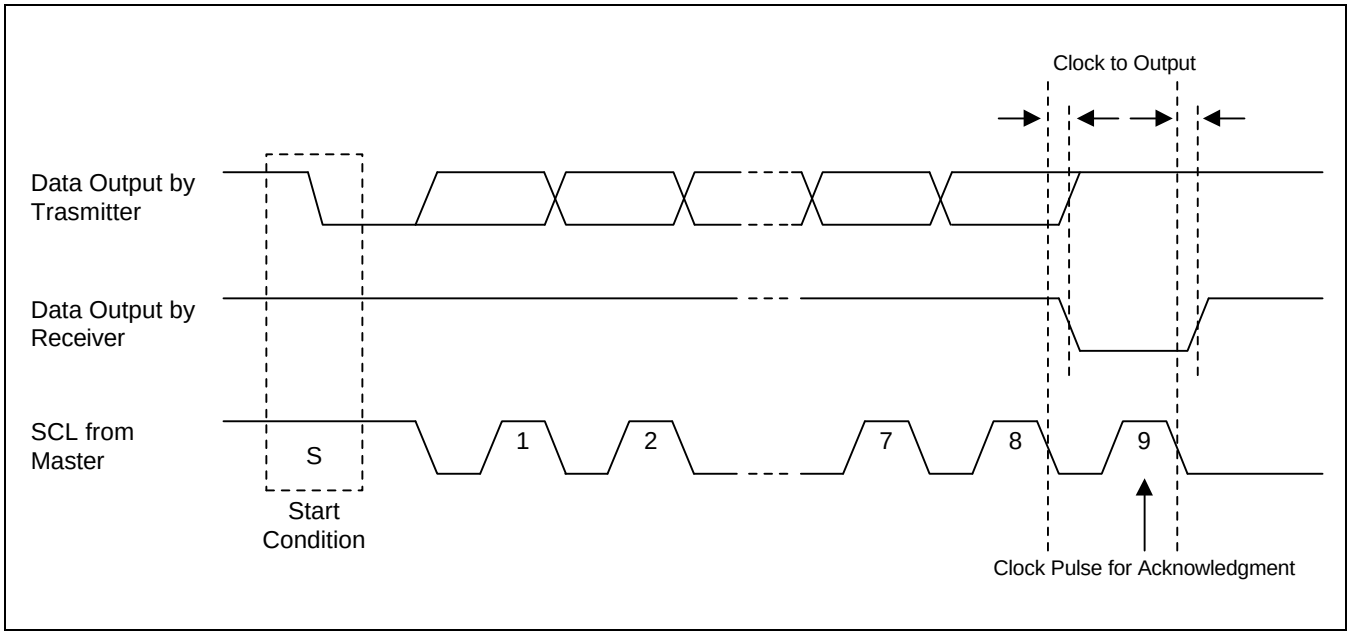


Figure 14-5. Acknowledge on the IIC-Bus

READ-WRITE OPERATION

In case of transmitter mode, after a data was transferred, the IIC-bus interface will wait until IICDS(IIC-bus Data Shift Register) is written by a new date. Until the new data is written, the SCL line will be held low. After the new data is written to IICDS register, the SCL line will be released. The S3C3410X should wait the interrupt to know the completion of transmission of current data. After getting the interrupt request, the CPU should write a new data into IICDS, again.

In case of receive mode, after a data is received, the IIC-bus interface will wait until IICDS register is read. Until the new data is read out, the SCL line will be held low. After the new data is read out from IICDS register, the SCL line will be released. The S3C3410X should wait the interrupt to know the completion of reception of new data. After getting the interrupt request, the CPU should read data from IICDS.

BUS ARBITRATION PROCEDURES

Arbitration takes place on the SDA line to prevent the contention on the bus between two masters. If a master with a SDA High level detects another master with a SDA active Low level, it will not indicate a data transfer because the current level on the bus does not correspond to its own. The arbitration procedure will be extended until the SDA line will be High.

But, in case of simultaneous lowering of the SDA line from masters, each master should evaluate whether or not the mastership is allocated to itself. For the purpose of evaluation, each master should detect the address bits. While each master generate the slaver address, it should also detect the address bit on the SDA line because the lowering of SDA line is stronger than maintaining High on the line. For example, one master generate Low as first address bit, while the other master is maintaining High. In this case, both master will be detect Low on the bus because Low is stronger than High even if first master is trying to maintain High on the line. In this case, Low-generating master as first address bit will get the mastership and High-generating master as first address bit should withdraw the mastership. If both master generate Low as first address bit, there should be arbitration for second address bit, again. This arbitration will be continued up to the end of last address bit.

ABORT CONDITION

If a slave receiver can not acknowledge the confirmation of the slave address, it should hold the level of the SDA line High. In this case, the master should generate a Stop condition to abort the transfer.

If a master receiver is involved in the aborted transfer, it should signal the end of the slave transmit operation. It does this by canceling the generation of an ACK in the master Rx mode. The slave transmitter should then release the SDA to allow a master to generate a Stop condition.

IIC-BUS INTERFACE SPECIAL REGISTERS

MULTI-MASTER IIC-BUS CONTROL REGISTER (IICCON)

Register	Offset Address	R/W	Description	Reset Value
IICCON	0xe000	R/W	IIC-bus control register	0x0

IICCON	Bit	Description	Initial State
Reserved	[0]	Reserved	0
BSSF	[1]	**Busy Signal Status Flag:** When CPU has read this bit, the "0" status indicates that IIC-Bus is idle and the "1" status means IIC-Bus is busy. In case of writing to this bit, the "0" write operation asserts the Stop signal on IIC-Bus interface and the "1" asserts the Start signal on IIC-Bus interface.	0
MS	[3:2]	**Mode Selection:** This field determines which IIC mode is currently able to read/write data from/to IICDAT 00 = Slave receive mode 01 = Slave transmit mode 10 = Master receive mode 11 = Master transmit mode	00
ACKE(1)	[4]	**Acknowledge Enable:** This bit determines whether IIC-Bus acknowledge is enabled or disabled. 0 = Disable ACK generation 1 = Enable ACK generation	0
BE	[5]	**IIC-Bus Enable:** This bit determines whether IIC-Bus data output is enabled or disabled. 0 = Disable Rx/Tx 1 = Enable Rx/Tx	0
Reserved	[6]	This bit should be to set "0"	0
Reset	[7]	If "1" is written to this bit, the IIC bus controller is reset to its initial state	0

NOTE: Interfacing EEPROM, the ACK generation may be disabled in order to generate the STOP condition in Rx mode.

MULTI-MASTER IIC-BUS STATUS REGISTER (IICSTAT)

Register	Offset Address	R/W	Description	Reset Value
IICSTAT	0xe001	R/W	IIC-bus status register	0x0

IICSTAT	Bit	Description	Initial State
LRBSF	[0]	**Last-received Bit Status Flag:** This bit is automatically set to "1" whenever an ACK signal is not received during a last bit receive operation. When the last receive bit is zero, this is as same meaning as the detection of an ACK signal. In this case, Last-Received Bit Status Flag will be cleared.	0
GCSF	[1]	**General Call Status Flag:** This bit is automatically set to "1" whenever "00000000b", General Call Value is issued as the received slave address. When the Start/Stop condition is detected, this bit of General Call Status Flag will be cleared.	0
MACSF	[2]	**Master Address Call Status Flag:** This bit is automatically set to "1" whenever the received slave address matches the address value in IICADD register. This bit will be cleared after Start/Stop condition is detected.	0
ASF	[3]	**Arbitration Status Flag:** This bit is automatically set to "1" to indicate that a bus arbitration has been failed during IIC-Bus interface. This bit is also set to "0" to indicate the successful arbitration for IIC-Bus interface	0
INTFLAG	[4]	**Interrupt Pending Flag:** This bit is IIC-bus Tx/Rx interrupt pending flag. It is impossible to write "1" into this bit. If this bit is read as "1", IICSCL is tied to "L" and IIC is stopped. To resume the operation, clear this bit by writing "0". 0 = 1) No interrupt pending (when read) 2) Clear pending condition (when write) 1 = 1) Interrupt is pending (when read) 2) N/A (when write)	0

NOTE: A IIC-bus interrupt occurs 1) when a 1-byte transmit or receive operation is terminated, 2) when a general call or a slave address match occurs, or 3) if bus arbitration fails. To measure the setup time of IICSDA before rising edge of IICSCL, IICDS has to be written before clearing IIC interrupt pending flag bit by the setup time in Tx mode.

MULTI-MASTER IIC-BUS ADDRESS REGISTER (IICADD)

Register	Address	R/W	Description	Reset Value
IICADD	0xe003	R/W	IIC-Bus transmit/receive address register	0x0

IICADD	Bit	Description	Initial State
Reserved	[0]	Reserved	0
SA	[7:1]	**Slave Address:** 7-bit slave address, latched from the IIC-bus: When serial output enable=0 in the IICCON register, IICADD is write-enabled. You can read the IICADD value at any time, regardless of the current serial output enable bit (IICCON) setting.	0000000b

MULTI-MASTER IIC-BUS TRANSMIT/RECEIVE DATA SHIFT REGISTER (IICDS)

Register	Address	R/W	Description	Reset Value
IICDS	0xe002	R/W	IIC-Bus transmit/receive data shift register	0x0

IICDS	Bit	Description	Initial State
DS	[7:0]	**Data Shift:** 8-bit data shift register for IIC-bus Tx/Rx operation: When serial output enable = 1 in the IICCON, IICDS is write-enabled. You can read the IICDS value at any time, regardless of the current serial output enable bit (IICCON) setting	0x0

MULTI-MASTER IIC-BUS PRESCALER REGISTER (IICPS)

Register	Address	R/W	Description	Reset Value
IICPS	0xe004	R/W	IIC-Bus Prescaler register	0xff

IICPS	Bit Size	Description	Initial State
PS	[7:0]	**Prescaler Value:** This prescaler value is used to generate clock of the IIC-Bus clock. The system clock is divided by (16 × (prescaler value +1)) to make clock of the IIC block. If the prescaler value is zero, IIC operation may work incorrectly.	0xff

MULTI-MASTER IIC-BUS PRESCALER COUNTER REGISTER (IICPCNT)

Register	Address	R/W	Description	Reset Value
IICPCNT	0xe005	R/W	IIC-Bus Prescaler Counter register	0x0

IICPCNT	Bit	Description	Initial State
PCNT	[7:0]	**Prescaler Counter Value:** This 8-bit value is the value of the prescaler counter. It is read(in test mode only) to check the counter's current value	0x0

NOTES

15 POWER MANAGEMENT

OVERVIEW

The Power Management Block in S3C3410X can manage the optimal power consumption for the given task by selecting the optimal operation mode. The Power Management scheme in S3C3410X consists of five categories, which are Normal, Slow, Idle, DMA Idle, and Stop mode. The key scheme of this power down mode is to distribute the clock or slow-down clock to necessary block for the given task. By selecting optimal clocking strategy, we can reduce the power consumption by getting rid of unnecessary power consumption for the given task.

S3C3410X has five power-down modes. The following section describes each power-down mode. The transition between the modes is not allowed freely. For available transitions among these modes, refer to Figure 15-1.

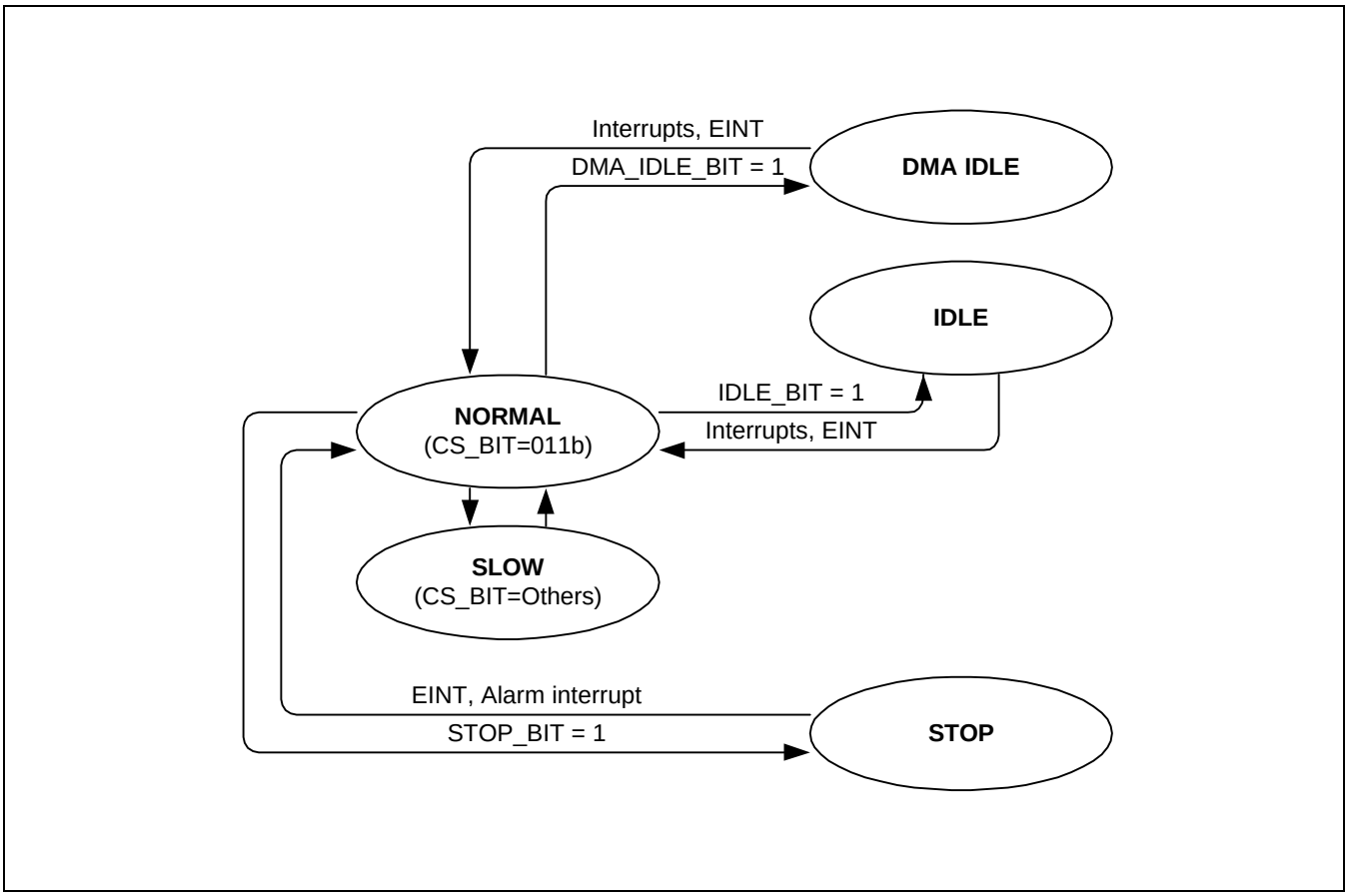


Figure 15-1. Power Management State Machine

POWER MANAGEMENT OPERATION

NORMAL MODE

In Normal mode, all peripherals and the basic blocks: such as CPU core, bus controller, memory controller, interrupt controller, and clock controller, should work normally. In Normal mode, the power consumption will be maximized.

SLOW MODE

The Slow mode can reduce power consumption by slowing down operation frequency. The operating frequency is divide by n of MCLK. The divide ratio is determined by CS bits in the SYSCON register.

IDLE MODE

IDLE mode is invoked by the setting SYSCON[1] to "1". In IDLE mode, the operation of CPU is halted by disconnecting the clock to CPU while some peripherals remain active.

These are two ways to escape from IDLE mode:

1. Execute a reset. All system and peripheral control registers are reset to their default value and the contents of all data registers are retained. The reset automatically selects a slow clock (1/16) because SYSCON[5:3] are cleared to "000b". if interrupt masked, a reset is the only way to escape from IDLE mode.
2. Any active interrupt happens, causing IDLE mode to be released. The interrupt routine will be serviced by active CPU. After the interrupt is serviced, the CPU will return to the next instruction after instruction used for IDLE mode entrance.

DMA IDLE MODE

The DMA Idle mode can be invoked by the setting SYSCON[2] to "1". In DMA Idle mode, CPU operation will be stopped while some peripherals remain active. This is same as IDLE mode. The difference between IDLE and DMA IDLE mode is that any external DMA request can wake up CPU and make CPU sleep by the corresponding DMA Acknowledge. Consequently, user can make CPU alive only during the DMA operation. This mode is effective when there are infrequent external DMA request based on Single DMA Request/Acknowledge. This mode is not effective for internal DMA request, Demand, or Block transfer mode. The main reason for this mode is that we can save the power consumption during DMA operation by sleeping CPU when user want DMA operation without CPU operation and when there are infrequent external DMA request.

These are three ways to release DMA Idle mode:

1. Execute a reset. It is the same that Idle mode.
2. Any active interrupt happens, causing DMA Idle mode to be released. It is as same as IDLE mode.
3. The external DMA request makes DMA Idle mode to be released and Acknowledge makes CPU in IDLE mode.

STOP MODE

Entering STOP Mode

STOP mode is invoked by the setting SYSCON[0] to "1". In STOP mode, the operation of the CPU and all peripherals should be halted. That is, the on-chip main oscillator stops and the supply current is reduced to less than 1uA. All system functions stop when the clock "freezes", but data stored in the internal register file is retained. STOP mode can be released by two ways: by a reset or by an external interrupt or alarm interrupt.

NOTE

Do not use STOP mode if you are using an external clock source because Xin input must be restricted internal to VSS to reduce current leakage. Also, do not use STOP mode if program control execute in DRAM memory because of DRAM leakage current. Therefore, please confirm status before the STOP mode: STOP command in located Non-DRAM memory.

Wake-Up from STOP Mode

The S3C3410X can be escaped from STOP mode by external interrupt or by a reset.

Using RESET to Release STOP Mode: The STOP mode should be released when the **RESET** signal is released. All system and peripheral control registers are reset to have their default hardware values and the contents of data registers should be retained. A reset operation automatically selects a slow clock(MCLK/16) because SYSCON[5:3] are cleared to "000b". After the programmed oscillation stabilization interval has elapsed, the CPU starts the system initialization routine by fetching the program instruction stored in location 0x0.

Using an External Interrupt to Release STOP Mode: External interrupts and alarm interrupt can be used to release STOP mode. Which interrupt you can use to release STOP mode in a given situation depends on the current microcontroller's operating mode. The external interrupts of EINT0 - EINT11 and alarm interrupt in the S3C3410X can be used to release STOP mode :

Please note the following conditions for STOP mode release:

- If user want to release STOP mode by using an external interrupt or alarm interrupt, the current values in system and peripheral control registers should be unchanged. User can also program the duration of the oscillation stabilization interval. To do this, user should make the appropriate control and clock setting before entering STOP mode.
- When the STOP mode is released by external interrupt or alarm interrupt, the SYSCON[5:3] setting remains unchanged and the selected clock value is used.
- The external interrupt should be serviced when the STOP mode release interrupt occurs. After interrupt service routine, the program sequence should be return to the instruction after the instruction for STOP mode entrance.

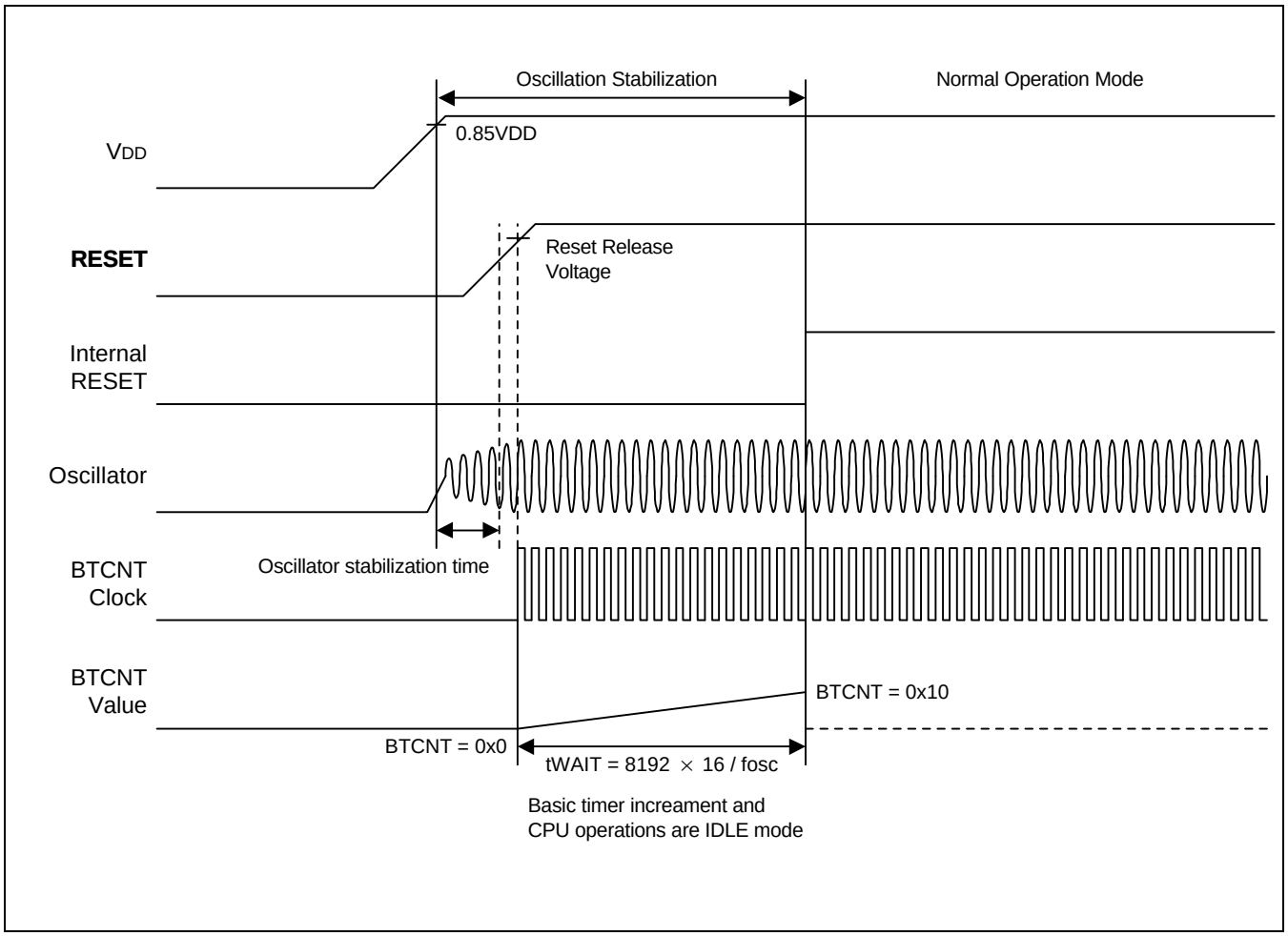
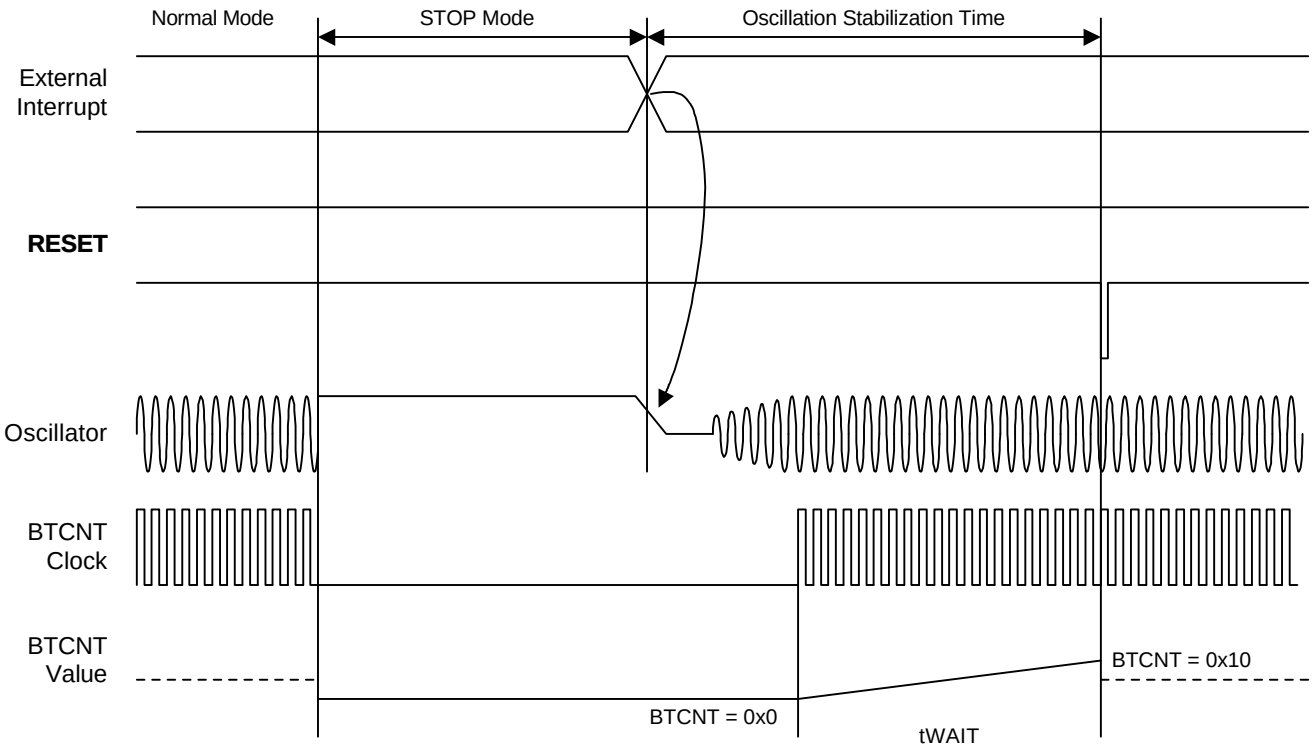


Figure 15-2. Oscillation stabilization Time on RESET

NOTE: Duration of the stabilization wait time, tWAIT, when it is released by a Power-on reset is ($2^{13} \times 16$ / fosc).



CS_BIT	tWAIT	tWAIT (When fosc is 40MHz)
00b	$2^{13} \times 16$ / fosc	3.28 ms
01b	$2^{12} \times 16$ / fosc	1.64 ms
10b	$2^{11} \times 16$ / fosc	0.82 ms
11b	$2^{9} \times 16$ / fosc	0.205 ms

Figure 15-3. Oscillation Stabilization Time on STOP Mode Release

NOTE: Duration of the stabilization wait time, tWAIT, it is released by an interrupt is determined by the setting in basic timer control register, BTCNT

POWER MANAGEMENT SPECIAL FUNCTION REGISTERS

SYSTEM CONTROL REGISTER (SYSCON)

The system control register is used to control the system operation of the chip.

Register	Offset Address	R/W	Description	Reset Value
SYSCON	0xd003	R/W	System control register	0x0

SYSCON	Bit	Description	Initial State
STOP	[0]	**STOP Control:** This bit value determines whether STOP mode is enabled or disabled. 0 = Normal operation 1 = Entering STOP mode	0
IDLE	[1]	**IDLE Control:** This bit value determines whether IDLE mode is enabled or disabled. 0 = Normal operation 1 = Entering IDLE mode	0
DMA_IDLE	[2]	**DMA IDLE Control:** This bit value determines whether DMA_IDLE mode is enabled or disable. 0 = Normal operation 1 = Entering DMA_IDLE mode	0
CS	[5:3]	**Clock Select:** This field determines frequency of system clock. 000 = MCLK / 16 001 = MCLK / 8 010 = MCLK / 2 011 = MCLK 100 = MCLK / 1024	000
GIE	[6]	**Global Interrupt Enable:** This bit control to enable or disable the interrupt 0 = No requested 1 = Requested	0

16 RTC (REAL TIME CLOCK)

OVERVIEW

The RTC(Real Time Clock) unit can be operated by the backup battery although the system power is turned off. The RTC can transmit 8-bit data to 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 and also can perform the alarm function

FEATURE

- BCD number: second, minute, hour, date, day, month, year
- Leap year generator
- Alarm function: alarm interrupt.
- Year 2000 problem is removed.
- Independent power pin (RTCVDD)
- RTC Time interrupt (SEC/MIN/HOUR)

REAL TIME CLOCK OPERATION

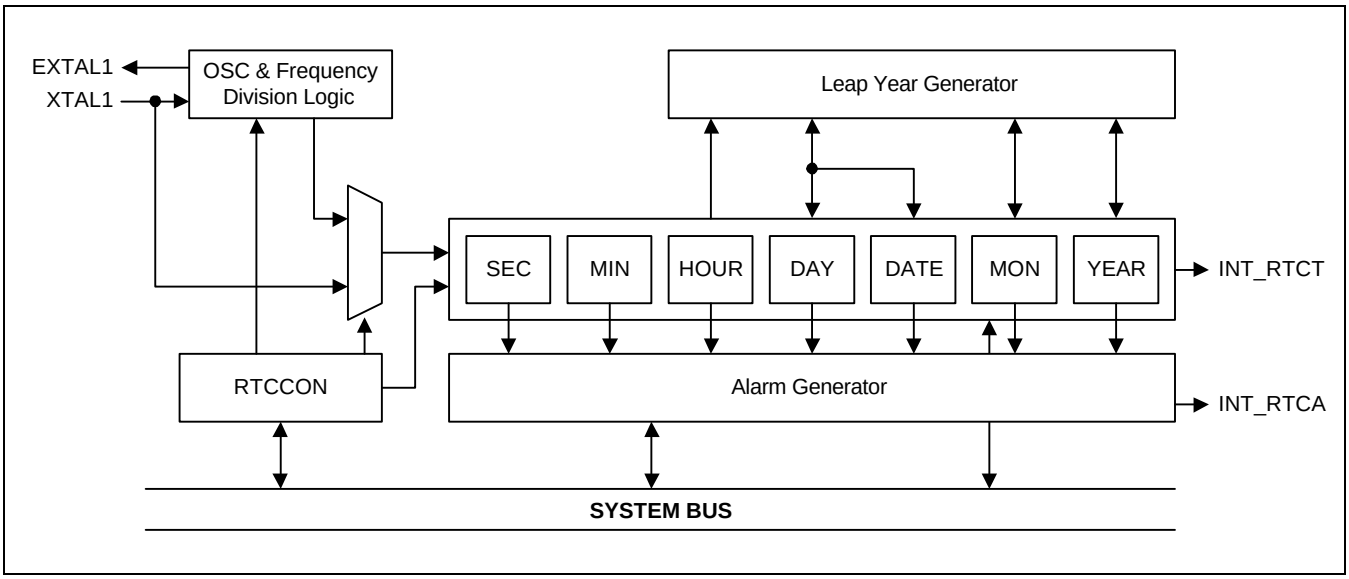


Figure 16-1. Real Time Clock Block Diagram

LEAP YEAR GENERATOR

This block can determine whether the last date of each month is 28, 29, 30, or 31, based on data from BCDDAY, BCDMON, and BCDYEAR. This block can also consider the leap year in deciding the last date. An 8-bit counter can only represent 2 BCD digits, so it cannot decide whether 00 year is a leap year or not. For example, it can not discriminate between 1900 and 2000. To solve this problem, the RTC block in S3C3410X has hard-wired logic to support the leap year in 2000. Please note 1900 is not leap year while 2000 is leap year. Therefore, two digits of 00 in S3C3410X denote 2000, not 1900.

SAFE READ OF SEC, MIN, HOUR, DAY, MONTH, AND YEAR

It is required to set bit 0 of the RTCCON register to read and write the register in RTC block. To display the sec., min., hour, day, month, and year, the CPU should read the data in BCDSEC, BCDMIN, BCDHOUR, BCDDAY, BCDDATE, BCDMON, and BCDYEAR register in RTC block. But, there may be one second deviation because of multiple register read. For example, when user read registers from BCDYEAR to BCDMIN register, we assume that the result was 1959(Year), 12(Month), 31(Date), 23(Hour) and 59(Minute). When user read BCDSEC register, if the result is value from 1 to 59(Second), there is no problem. But, if the result is 0 sec., there will be possibility for year, month, data, hour, and minute to be changed into 1960(Year), 1(Month), 1(Date), 0(Hour) and 0(Minute) because of one second deviation as above-mentioned. In this case, user should read from BCDYEAR to BCDSEC again if BCDSEC is zero.

BACKUP BATTERY OPERATION

The RTC logic can be driven by the backup battery, which supplies the power through the RTCVDD pin into RTC block, even if the system power is off. In this case of power-off, the interfaces of the CPU and RTC logic should be blocked and the backup battery only drives the oscillation circuit and the BCD counters to minimize power dissipation.

ALARM FUNCTION

The RTC can generate an alarm signal at a specified time in the power down mode or normal operation mode. In normal operation mode, the alarm interrupt (INT_RTCA) is activated. The RTC alarm register, RTCALM, can determine the alarm enable/disable and the condition of the alarm time setting.

RTC TIMER INTERRUPT OPERATION

The RTC generates an time interrupt at each sec/minute/hour/day in normal operation mode. In normal operation mode, the RTC time interrupt (INT_RTCT) is activated. The RTC time interrupt control register, RINTCON, determines the RTC time (SEC/MIN/HOUR/DAY) interrupt enable.

REAL TIME CLOCK SPECIAL REGISTERS

REAL TIME CLOCK CONTROL REGISTER (RTCCON)

The RTCCON register consists of 4 bits such as RTCEN which controls the read/write enable of the BCD registers, CLKSEL, CNTSEL, and CLKRST for testing.

RTCEN bit can control all interfaces between the CPU and the RTC, so it should be set to 1 in an RTC control routine to enable data read/write after a system reset. Also before power off, the RTCEN bit should be cleared to 0 to prevent an inadvertent writing into RTC registers.

Register	Offset Address	R/W	Description	Reset Value
RTCCON	0xa013	R/W	RTC control register	0x0

RTCCON	Bit	Description	Initial State
RTCEN	[0]	RTC read/write enable 0 = Disable, 1 = Enable	0
CLKSEL	[1]	BCD clock select 0 = XTAL $1/2^{15}$ divided clock 1 = Reserved (XTAL clock)	0
CNTSEL	[2]	BCD count select 0 = Merge BCD counters 1 = Reserved (Separate BCD counters)	0
CLKRST	[3]	RTC clock count reset 0 = No reset, 1 = Reset	0

RTC ALARM CONTROL REGISTER (RTCALM)

RTCALM register can determine the alarm enable/disable and the alarm time. Note that the RTCALM register can generate the alarm signal through INT_RTCA.

Register	Offset Address	R/W	Description	Reset Value
RTCALM	0xa012	R/W	RTC alarm control register	0x0

RTCALM	Bit	Description	Initial State
SECEN	[0]	Second alarm enable 0 = Disable, 1 = Enable	0
MINEN	[1]	Minute alarm enable 0 = Disable, 1 = Enable	0
HOUREN	[2]	Hour alarm enable 0 = Disable, 1 = Enable	0
DAYEN	[3]	Day alarm enable 0 = Disable, 1 = Enable	0
MONREN	[4]	Month alarm enable 0 = Disable, 1 = Enable	0
YEAREN	[5]	Year alarm enable 0 = Disable, 1 = Enable	0
ALMEN	[6]	Alarm global enable 0 = Disable, 1 = Enable	0

ALARM SECOND DATA REGISTER (ALMSEC)

Register	Offset Address	R/W	Description	Reset Value
ALMSEC	0xa033	R/W	Alarm second data register	0x59

ALMSEC	Bit	Description	Initial State
SECDATA	[6:4]	BCD value for alarm second from 0 to 5	101
	[3:0]	from 0 to 9	1001

ALARM MIN DATA REGISTER (ALMMIN)

Register	Offset Address	R/W	Description	Reset Value
ALMMIN	0xa032	R/W	Alarm minute data register	0x59

ALMMIN	Bit	Description	Initial State
MINDATA	[6:4]	BCD value for alarm minute from 0 to 5	101
	[3:0]	from 0 to 9	1001

ALARM HOUR DATA REGISTER (ALMHOUR)

Register	Offset Address	R/W	Description	Reset Value
ALMHOUR	0xa031	R/W	Alarm hour data register	0x23

ALMHOUR	Bit	Description	Initial State
HOURDATA	[5:4]	BCD value for alarm hour from 0 to 2	10
	[3:0]	from 0 to 9	0011

ALARM DAY DATA REGISTER (ALMDAY)

Register	Offset Address	R/W	Description	Reset Value
ALMDAY	0xa037	R/W	Alarm day data register	0x31

ALMDAY	Bit	Description	Initial State
DAYDATA	[5:4]	BCD value for alarm day, from 0 to 28, 29, 30, 31 from 0 to 3	11
	[3:0]	from 0 to 9	0001

ALARM MON DATA REGISTER (ALMMON)

Register	Offset Address	R/W	Description	Reset Value
ALMMON	0xa036	R/W	Alarm month data register	0x12

ALMMON	Bit	Description	Initial State
MONDATA	[4]	BCD value for alarm month from 0 to 1	1
	[3:0]	from 0 to 9	0010

ALARM YEAR DATA REGISTER (ALMYEAR)

Register	Offset Address	R/W	Description	Reset Value
ALMYEAR	0xa035	R/W	Alarm year data register	0x99

ALMYEAR	Bit	Description	Initial State
YEARDATA	[7:0]	BCD value for year from 00 to 99	0x99

BCD SECOND REGISTER (BCDSEC)

Register	Offset Address	R/W	Description	Reset Value
BCDSEC	0xa023	R/W	BCD second register	Undef.

BCDSEC	Bit	Description	Initial State
SECDATA	[6:4]	BCD value for second from 0 to 5	–
	[3:0]	from 0 to 9	–

BCD MINUTE REGISTER (BCDMIN)

Register	Offset Address	R/W	Description	Reset Value
BCDMIN	0xa022	R/W	BCD minute register	Undef.

BCDMIN	Bit	Description	Initial State
MINDATA	[6:4]	BCD value for minute from 0 to 5	–
	[3:0]	from 0 to 9	–

BCD HOUR REGISTER (BCDHOUR)

Register	Offset Address	R/W	Description	Reset Value
BCDHOUR	0xa021	R/W	BCD hour register	Undef.

BCDHOUR	Bit	Description	Initial State
HOURDATA	[5:4]	BCD value for hour from 0 to 2	–
	[3:0]	from 0 to 9	–

BCD DAY REGISTER (BCDDAY)

Register	Offset Address	R/W	Description	Reset Value
BCDDAY	0xa027	R/W	BCD day register	Undef.

BCDDAY	Bit	Description	Initial State
DAYDATA	[5:4]	BCD value for hour from 0 to 3	–
	[3:0]	from 0 to 9	–

BCD DATE REGISTER (BCDDATE)

Register	Offset Address	R/W	Description	Reset Value
BCDDATE	0xa020	R/W	BCD date register	Undef.

BCDDATE	Bit	Description	Initial State
DATEDATA	[2:0]	BCD value for date from 1 to 7	–

BCD MONTH REGISTER (BCDMON)

Register	Offset Address	R/W	Description	Reset Value
BCDMON	0xa026	R/W	BCD month register	Undef.

BCDMON	Bit	Description	Initial State
MONDATA	[4]	BCD value for month from 0 to 1	–
	[3:0]	from 0 to 9	–

BCD YEAR REGISTER (BCDYEAR)

Register	Offset Address	R/W	Description	Reset Value
BCDYEAR	0xa025	R/W	BCD year register	Undef.

BCDMON	Bit	Description	Initial State
YEARDATA	[7:0]	BCD value for year from 00 to 99	–

RTC TIME INTERRUPT PENDING REGISTER (RINTPND)

Register	Offset Address	R/W	Description	Reset Value
RINTPND	0xa010	R/W	RTC Time interrupt pending register	0x0

RINTPND	Bit	Description	Initial State
INT_SEC	[0]	0 = No interrupt pending *0 = Clear interrupt pending condition (when write)* 1 = RTC SEC interrupt is pending	0
INT_MIN	[1]	0 = No interrupt pending *0 = Clear interrupt pending condition (when write)* 1 = RTC MIN interrupt is pending	0
INT_HOUR	[2]	0 = No interrupt pending *0 = Clear interrupt pending condition (when write)* 1 = RTC HOUR interrupt is pending	0
INT_DAY	[3]	0 = No interrupt pending *0 = Clear interrupt pending condition (when write)* 1 = RTC DAY interrupt is pending	0

RTC TIME INTERRUPT CONTROL REGISTER (RINTCON)

Register	Offset Address	R/W	Description	Reset Value
RINTCON	0xa011	R/W	RTC Time interrupt control register	0x0

RINTCON	Bit	Description	Initial State
INT_SEC	[0]	Setting RTC Time interrupt enable of SEC 0 = Disable 1 = Enable	0
INT_MIN	[1]	Setting RTC Time interrupt enable of MIN 0 = Disable 1 = Enable	0
INT_HOUR	[2]	Setting RTC Time interrupt enable of HOUR 0 = Disable 1 = Enable	0
INT_DAY	[3]	Setting RTC Time interrupt enable of DAY 0 = Disable 1 = Enable	0

17 ELECTRICAL DATA

ABSOLUTE MAXIMUM RATINGS

Table 17-1. Absolute Maximum Rating

Symbol	Parameter	Rating	Unit
V_{DD}	DC Supply Voltage	– 0.3 to 3.8	V
V_{IN}	DC Input Voltage	– 0.3 to V_{DD} + 0.3	V
I_{IN}	DC Input Current	± 10	mA
T_A	Operating Temperature	0 to 70	°C
T_{STG}	Storage Temperature	– 40 to 125	°C
$RTCV_{DD}$	Battery Voltage for RTC	2.5 to V_{DD}	V

D.C. ELECTRICAL CHARACTERISTICS

Table 17-2. DC Electrical Characteristics

(V_{DD} = 3.3 ± 0.3V, T_A = 0 to 70 °C)

Symbol	Parameters	Conditions	Min	Type	Max	Unit
V_{IH}	High level input voltage					V
	LVCMOS Interface		2.0			
	Schmitt-trigger Interface		2.3			
	RESET		$V_{DD} \times 0.8$			
	XTAL, EXTAL		$V_{DD} - 0.3$			
V_{IL}	Low level input voltage					V
	LVCMOS Interface				0.8	
	Schmitt-trigger Interface				0.8	
	RESET				$V_{DD} \times 0.2$	
	XTAL, EXTAL				0.4	
VT	Switching threshold	LVCMOS		1.4		V
VT+	Schmitt trigger, positive-going threshold	LVCOMS			2.3	V
VT-	Schmitt trigger, negative-going threshold	LVCMOS	0.8			V
I_{IH}	High level input current					uA
	Input buffer	$V_{IN} = V_{DD}$	–10		10	
	Input buffer with pull-up		10	60	90	
I_{IL}	Low level input current					uA
	Input buffer	$V_{IN} = V_{SS}$	–10		10	
	Input buffer with pull-up		–90	-60	–10	
V_{OH}	High level output voltage					V
	Type B4, B8 (1)	I_{OH} = -1 uA	$V_{DD} - 0.05$			
	Type B4	I_{OH} = -4 mA	2.4			
	Type B8	I_{OH} = -8 mA	2.4			
V_{OL}	Low level output voltage					V
	Type B4, B8 (1)	I_{OH} = 1 uA			0.05	
	Type B4	I_{OH} = 4 mA			0.4	
	Type B8	I_{OH} = 8 mA			0.4	
I_{OZ}	Tri-state output leakage current	$V_{OUT}=V_{SS}$ or V_{DD}	–10		10	uA
I_{OS}	Output short circuit current	V_{DD}=3.6V, $V_O=V_{DD}$			210	mA
		V_{DD}=3.6V, $V_O=V_{SS}$	–170			mA
C_{IN}	Input capacitance (2)	Any Input and bi-directional buffers			4	pF
C_{OUT}	Output capacitance (2)	Any Output buffer			4	pF

NOTES:

1. Type B4 means 4mA output driver cell, and Type B8 means 8mA output driver cells.
2. This value excludes package parasitic.

Typical Quiescent Supply Current on VDD @ Normal Mode

- Test Condition 1 : Cache off, Write buffer off, and ROM access

	10 MHz	20 MHz	30MHz	40MHz
3.0 V	9.06	16.52	23.62	32.96
3.3 V	10.40	18.72	27.38	37.46
3.6 V	11.80	20.96	31.06	42.24

- Test Condition 2 : Cache on, Write buffer on, and cache hit operation at SDRAM interface

	10 MHz	20 MHz	30MHz	40MHz
3.0 V	14.74	27.12	39.68	54.24
3.3 V	16.72	30.66	45.58	61.54
3.6 V	18.90	34.38	51.42	69.18

- Test Condition 3 : Cache on, Write buffer on, and DMA transfer(SDRAM to SDRAM, Half-word transfer mode)

	10 MHz	20 MHz	30MHz	40MHz
3.0 V	22.0	42.0	62.0	84.0
3.3 V	25.0	47.0	70.0	94.0
3.6 V	28.0	53.0	79.0	104.0

Typical Quiescent Supply Current on VDD @ IDLE Mode

	10 MHz	20 MHz	30MHz	40MHz
3.0 V	3.26	4.78	6.16	9.64
3.3 V	3.94	5.54	7.78	11.30
3.6 V	4.62	6.44	9.28	13.14

Typical Quiescent Supply Current on VDD @ STOP Mode

	10 MHz	20 MHz	30MHz	40MHz
3.0 V	14 uA			
3.3 V	15 uA			
3.6 V	16 uA			

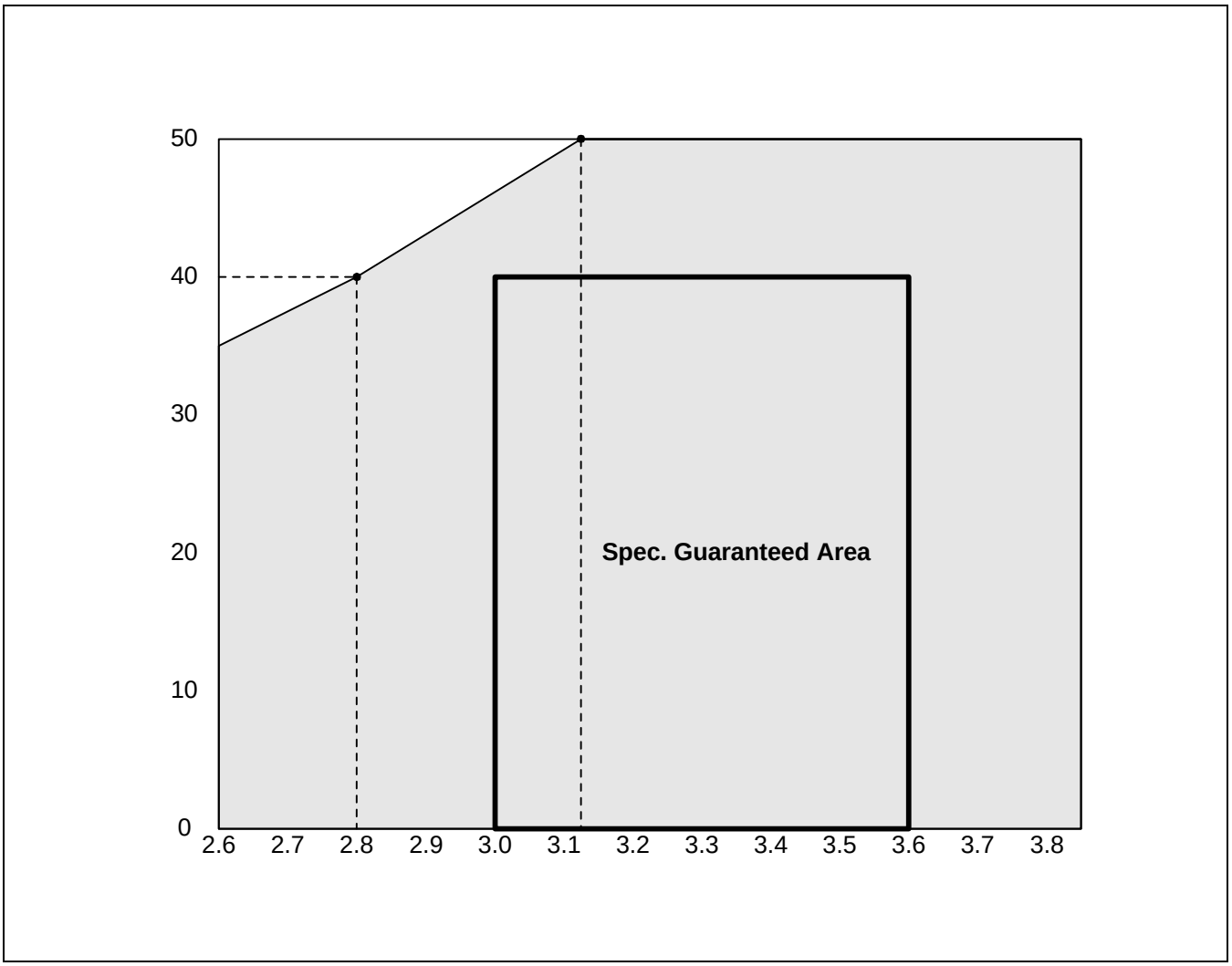


Figure 17-1. Typical Operating Voltage Range

A.C. ELECTRICAL CHARACTERISTICS

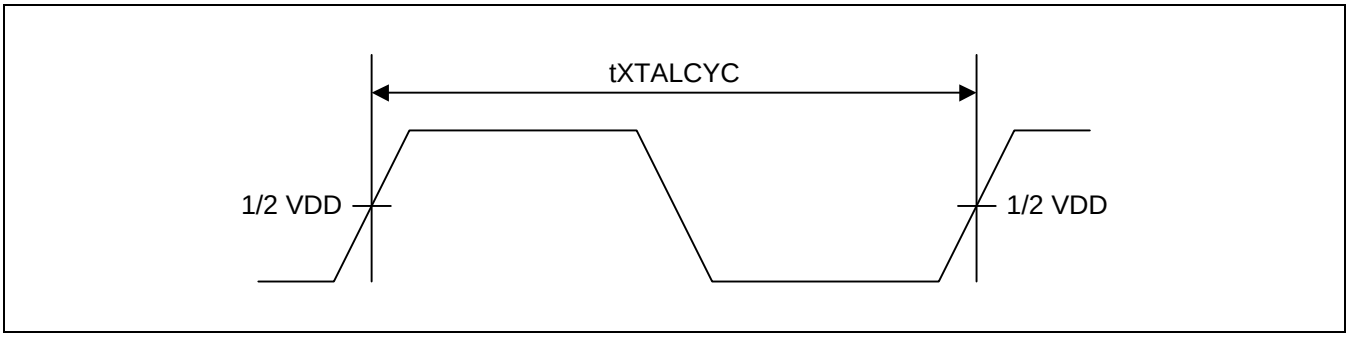


Figure 17-2. EXTAL0 Clock Timing

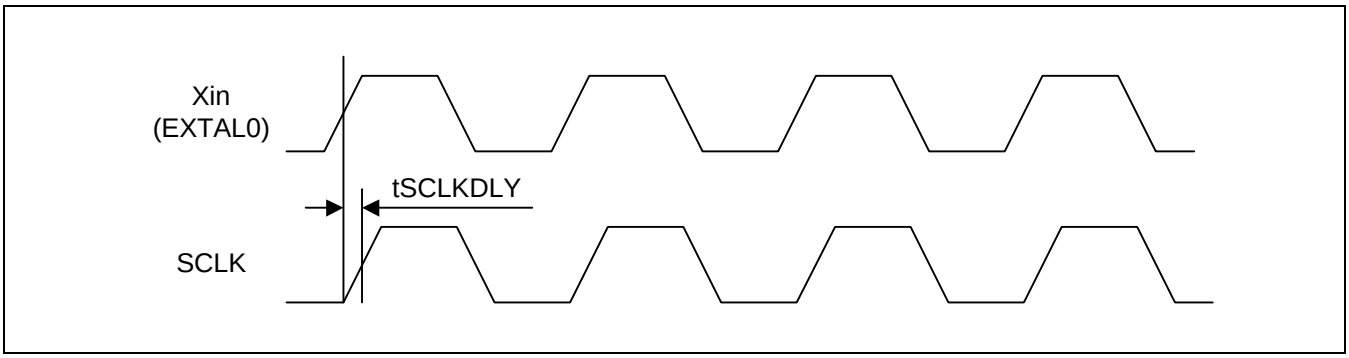


Figure 17-3. Xin(EXTAL0)/SCLK Timing

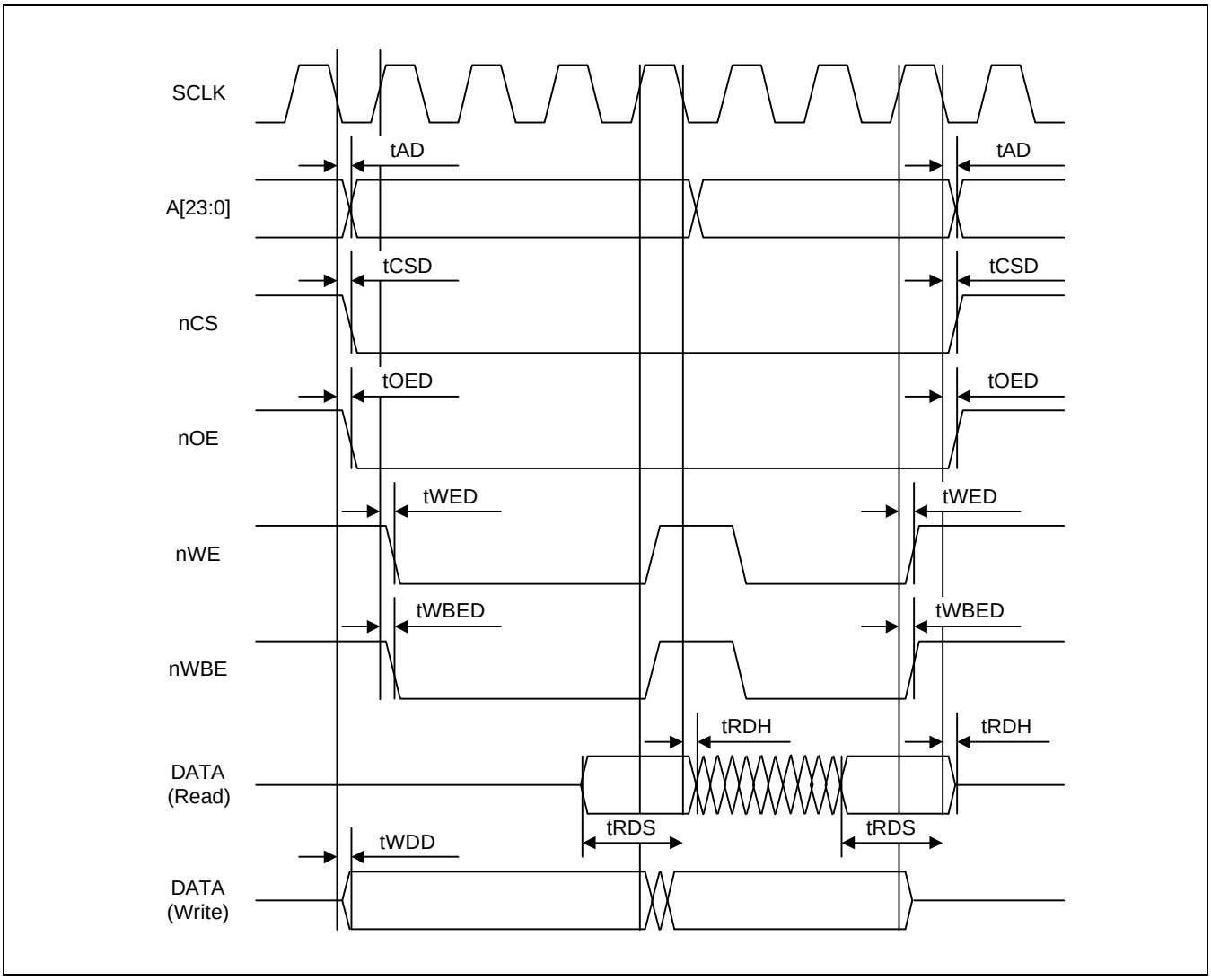


Figure 17-4. ROM/SRAM Bus Timing

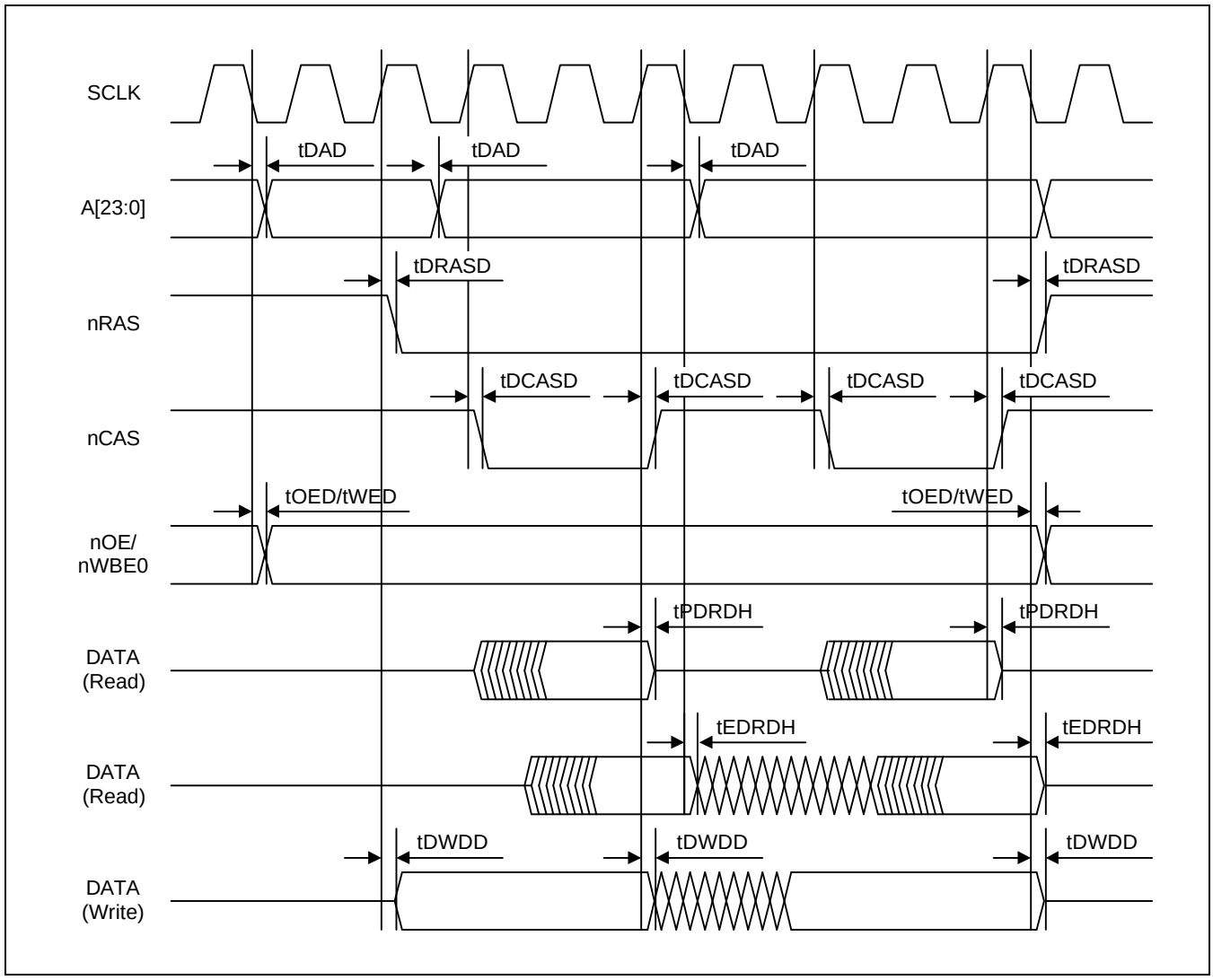


Figure 17-5. DRAM (Fast Page/EDO) Bus Timing

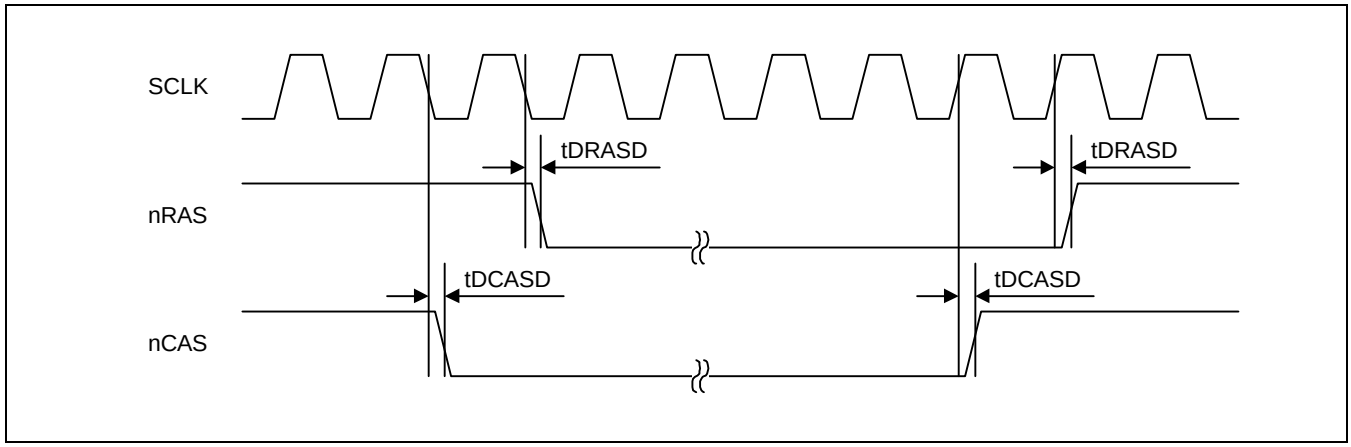


Figure 17-6. DRAM CBR Refresh Timing

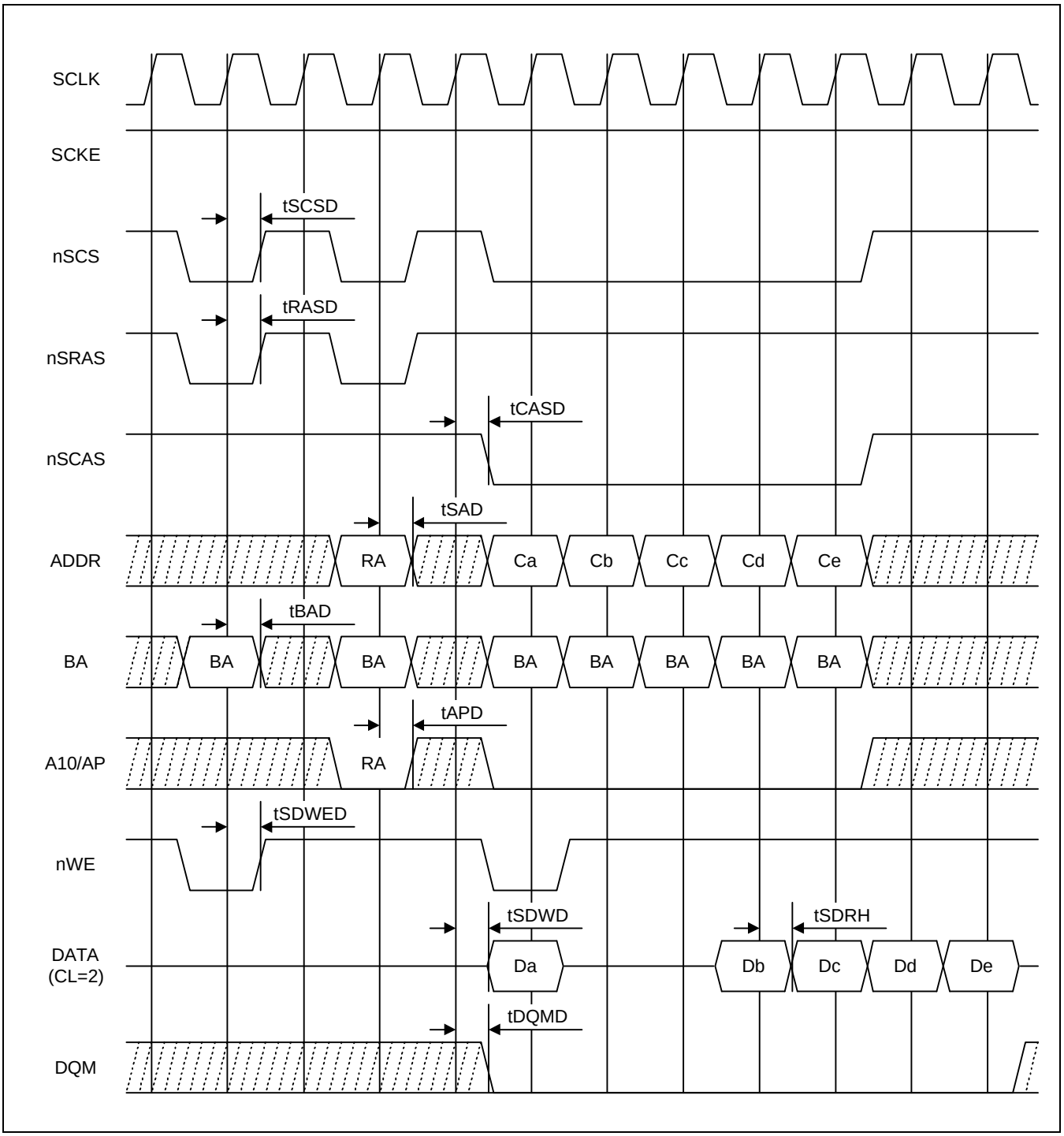


Figure 17-7. SDRAM Bus Timing (Single Write and Burst Read)

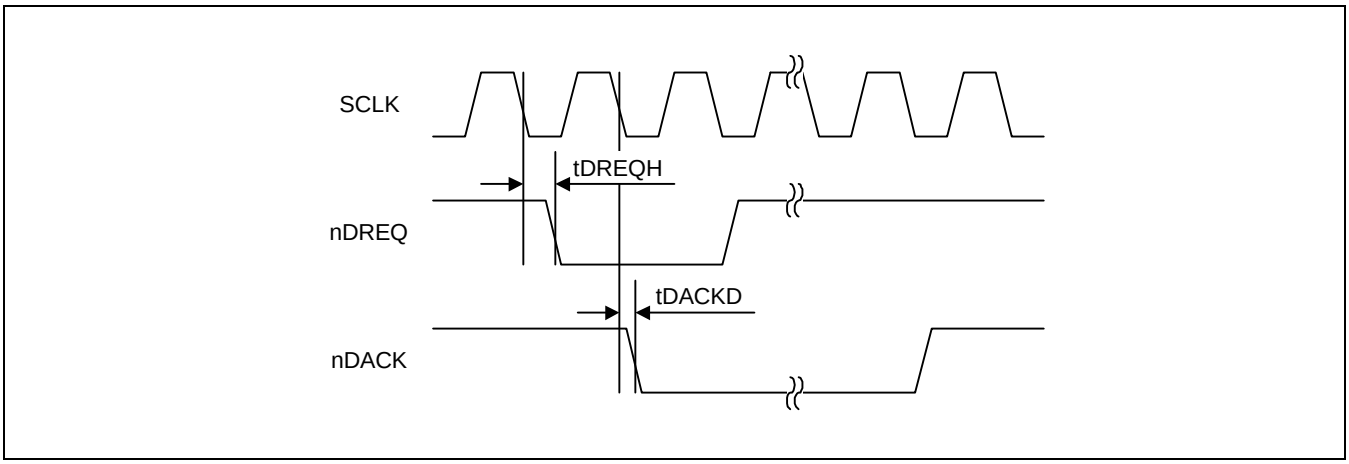


Figure 17-8. External DMA Timing

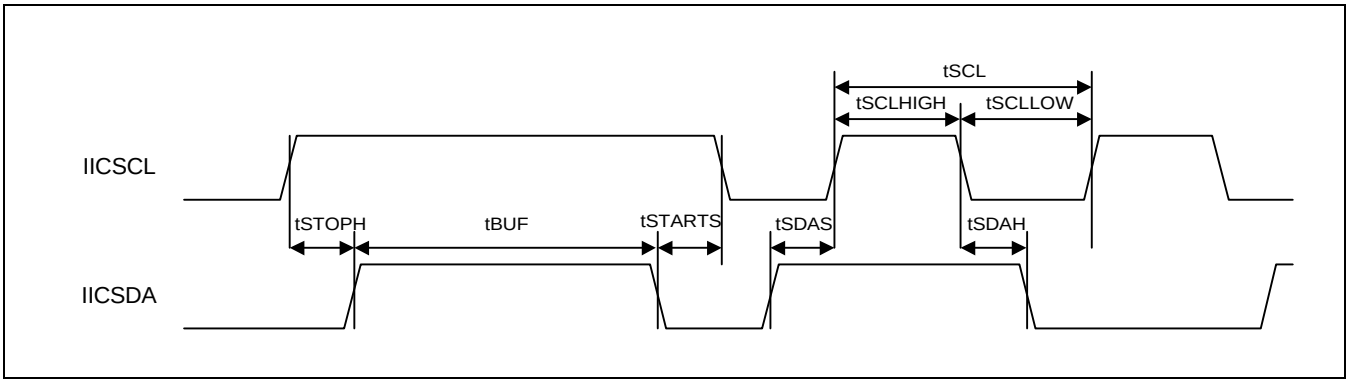


Figure 17-9. IIC Interface Timing

Table 17-3. Clock Timing

(V_{DD} = 3.3 ± 0.3V, T_A = 0 to 70 °C, Operating Frequency = 40 MHz)

Parameter	Symbol	Min	Typ	Max	Unit
Crystal clock input frequency	fXTAL			40	MHz
Crystal clock input cycle time	tXTALCYC	25			ns
Xin to SCLK delay time	tSCLKDLY			17.2	ns

Table 17-4. DMA Controller Timing

(V_{DD} = 3.3 ± 0.3V, T_A = 0 to 70 °C, Operating Frequency = 40 MHz)

Parameter	Symbol	Min	Typ	Max	Unit
nDREQ hold time	tDREQS		4.46		ns
nDACk delay time	tDACKD		2.62		ns

Table 17-5. IIC Interface Timing

(V_{DD} = 3.3 ± 0.3V, T_A = 0 to 70 °C, Operating Frequency = 40 MHz)

Parameter		Symbol	Min	Typ	Max	Unit
SCL high level pulse width	100 KHz	tSCLHIGH	4.0			us
	400 KHz		0.6			
SCL low level pulse width	100 KHz	tSCLLOW	4.7			us
	400 KHz		1.3			
Bus free time between STOP and START	100 KHz	tBUF	4.7			us
	400 KHz		1.3			
START hold time	100 KHz	tSTAH	4.0			us
	400 KHz		0.6			
SDA hold time	100 KHz	tSDAH	0			us
	400 KHz		0		0.9	
SDA setup time	100 KHz	tSDAS	250			ns
	400 KHz		100			
STOP hold time	100 KHz	tSTOS	4.7			us
	400 KHz		0.6			

Table 17-6. Memory Interface Timing

(V_{DD} = 3.3 ± 0.3V, T_A = 0 to 70 °C, Operating Frequency = 40 MHz)

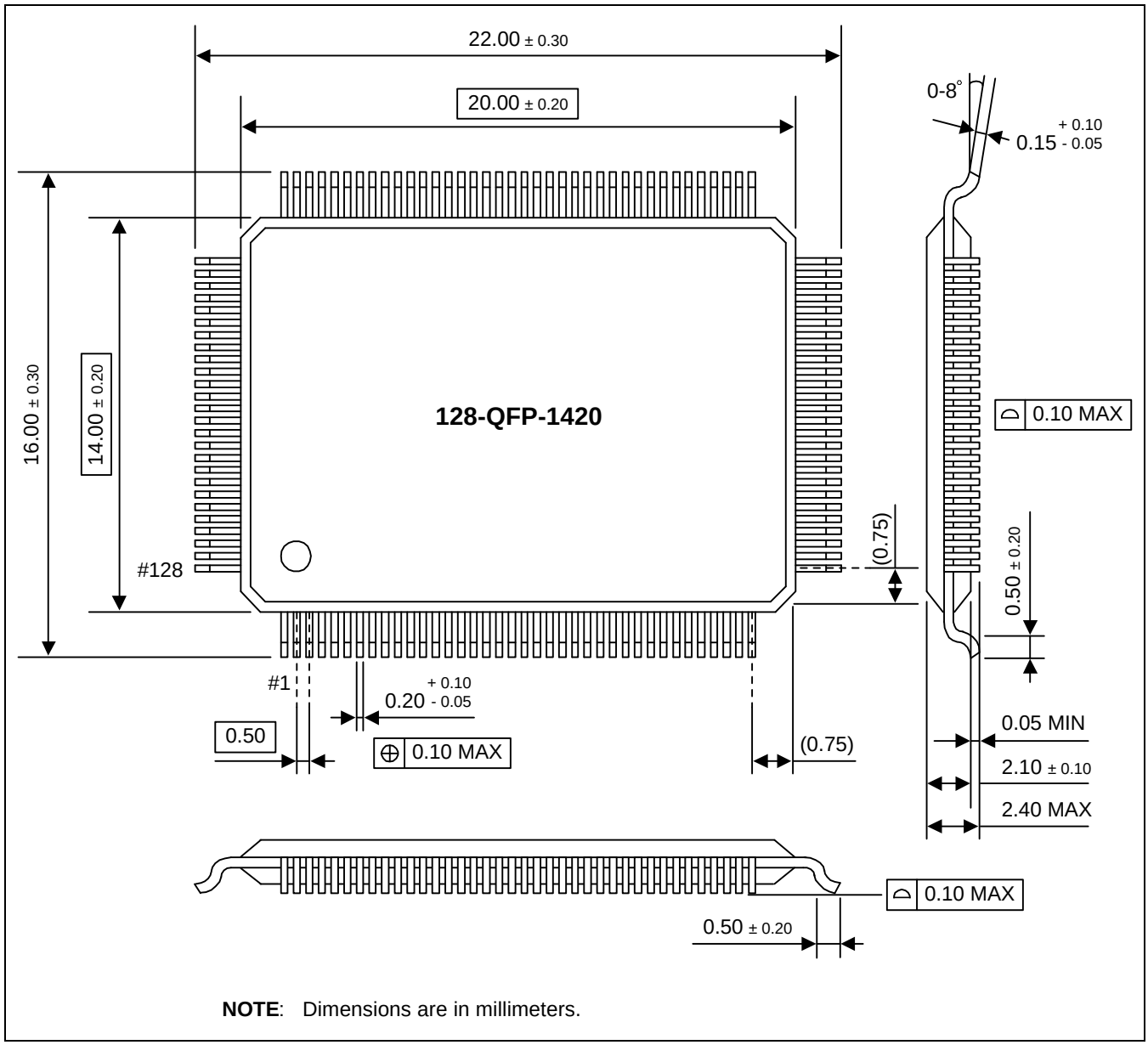
Parameter	Symbol	Min	Typ	Max	Unit
ROM/SRAM address delay time	t_{AD}			13.41	ns
ROM/SRAM chip select delay time	t_{CSD}			10.87	ns
ROM/SRAM read enable delay time	t_{OED}			10.44	ns
ROM/SRAM write enable delay time	t_{WED}			12.83	ns
ROM/SRAM write byte enable delay time	t_{WBED}			12.50	ns
ROM/SRAM read data setup time	t_{RDS}		5.14		ns
ROM/SRAM read data hold time	t_{RDH}	0			ns
ROM/SRAM write data delay time	t_{WDD}			14.30	ns
DRAM column address delay time	t_{DCAD}			12.03	ns
DRAM row address delay time	t_{DRAD}			11.28	ns
DRAM RAS delay time	t_{DRASD}			11.76	ns
DRAM CAS delay time	t_{DCASD}			12.40	ns
DRAM read enable delay time	t_{DOED}			10.37	ns
DRAM write enable delay time	t_{DWED}			12.02	ns
DRAM(FP) read data hold time	t_{PDRDH}	2.18			ns
DRAM(EDO) read data hold time	t_{EDRDH}	2.79			ns
DRAM write data delay time	t_{DWDD}			11.23	ns
SDRAM chip select delay time	t_{SCSD}			16.25	ns
SDRAM SRAS delay time	t_{RASD}			14.89	ns
SDRAM SCAS delay time	t_{CASD}			15.00	ns
SDRAM address delay time	t_{SAD}			13.27	ns
SDRAM bank address delay time	t_{BAD}			13.14	ns
SDRAM A10 address delay time	t_{APD}			13.20	ns
SDRAM data write delay time	t_{SDWD}			14.30	ns
SDRAM data read hold time	t_{SDRH}	0			ns
SDRAM write enable delay time	t_{SDWED}			14.83	ns
SDRAM DQM delay time	t_{DQMD}			14.48	ns
SDRAM SCKE delay time	t_{SCKED}			14.59	ns
nWAIT setup time	t_{WAIT}	10			ns

NOTES

18 MECHANICAL DATA

OVERVIEW

The S3C3410X is available in a 128-QFP-1420 package.



NOTE: Dimensions are in millimeters.

Figure 18-1. 128-QFP-1420 Package Dimensions

NOTES