



AX215E

SD/MMC/eMMC Card Controller Internal Design Specification

Rev 0.0.1
Dec 2011

AppoTech Limited

Address : Unit 705-707, 7/F, IC Development Centre,
No. 6, Science Park West Ave.
Hong Kong Science Park,
Shatin, N.T. Hong Kong
Telephone : (852) 2607 4090



AX215C SD Card Controller

F
a
x

:

(
8
5
2
)

2
6
0
7

4
0
9
6

w
w
w
.
a
p
p
o
t
e
c
h
.
c
o
m

AppoTech Limited

Address :Unit 705-707, 7/F, IC Development Centre,
No.6, Science Park West Ave.,
Hong Kong Science Park,
Shatin, N.T., Hong Kong
Telephone : (852) 2607 4090
Fax : (852) 2607 4096
www.appotech.com

CPU Features

- 8-bit 8051 CPU, with enhance extend instruction, optimized for SD, NAND Flash applications
- All instructions are single-cycled except branching instructions
- MAX 70 MIPS performance with on chip RC oscillator

Memory Features

- 32K Byte SRAM
 - (code space and xdata space)
- 8k Byte SRAM
 - (code space 、 xdata space and 340 Byte NFC 坏列管理 复用)
- 264 Byte RAM
 - (BCH hardware correction 复用)
- 256 Byte DATA RAM
 - (data ram and xdata space)
- 2K Byte VIA ROM
 - (code space)

General I/O

- 28 GPIO pins
- 6 SDIO pins
- All GPIO pins can be programmable as input or output individually
- All IO pins are internal pull-up selectable individually
- ALL NAND FLASH IO pins can be programmable as **4mA or 8mA or 12mA or 16mA output driving**
- ALL SDIO pins can be programmable as **12mA or 16mA output driving**
- CMOS / TTL level input

SD Interface Features

- Fully supports SD card standard 1.0/1.1/2.0
- Supports SD high capacity standard
- Supports SPI mode
- Supports Command class 0/2/4/5/6/7/8/10
- Speed class up to 6
- Supports host clock up to 50MHz
- Supports bus width X1/X4
- Enhanced ESD protection

MMC/eMMC Interface Features

-
-

NAND Flash Interface Features

- Supports SDR/Toggle DDR1.0/ONFI2.2 interface
- Supports X8/X16 flash
- Supports up to 4 CE
- Supports bad column management 160 byte data mask per page
- Supports data randomized
- Built-in 17/25/31/35/44 bit/page(1K bytes) on-the-fly ECC,优化512 Byte 的情况 Support hardware encode, decode and error correction
- Supports Two-Plane or Interleave NAND flash

Low Power Consumption

- Operating frequency variable 10-90MHz
- Supports Sleep Mode and Idle mode
- Fast wake up during Sleep Mode

Other Features

- Support 1 timer
- Support SPI, UART interface
- Built-in RC Oscillator
- Built-in POR and BOR
- Built-in 3.3V to 1.2V LDO
- Built-in 3.3V to 1.8V LDO

Package

- Die form

Temperature

- Operating temp.: 0°C to +70°C
- Storage temp.: -65°C to +150°C

AX215E Contents

CHAPTER 1 PRODUCT OVERVIEW.....	5
1.1 Description.....	5
1.2 System Architecture.....	6
CHAPTER 2 PIN INFORMATION.....	7
2.1 Pin Assignment.....	7
2.1.1 DIE Form.....	7
2.2 Pin Description.....	9
CHAPTER 3 MEMORY MAPPING AND CPU ARCHITECTURE.....	11
3.1 Basic Structure.....	11
3.2 Interrupt Entry Address Mapping.....	12
3.3 Data Memory – DRAM (DATA).....	12
3.4 General Purpose Register (REGISTER).....	12
3.5 Extended Data Memory – XRAM (XDATA).....	13
3.6 CPU Instruction Set Summary.....	13
3.6.1 Basic Instruction Set.....	13
3.6.2 Extended Instruction Set.....	17
CHAPTER 4 SPECIAL FUNCTION REGISTER (SFR).....	20
4.1 SFR List.....	20
4.2 SFR Description.....	23
CHAPTER 5 INTERRUPT PROCESSING.....	33
5.1 Interrupt Sources and Vectors.....	33
5.2 Interrupt Priorities.....	33
5.3 Interrupt Latency.....	34
CHAPTER 6 CLOCKS AND RESET MANAGEMENT.....	35
6.1 Clock System.....	35
6.1.1 Clock Gating Control.....	35
6.2 Relative Registers.....	36
6.3 Reset System.....	38
CHAPTER 7 PORTS.....	40
CHAPTER 8 TIMER AND SERIAL PORT.....	46
8.1 Timer and Serial Port Register.....	46
8.2 Timer.....	47
8.3 Uart Mode.....	48
8.4 SPI Mode.....	48
8.5 CRC16 Mode.....	49

CHAPTER 9 NAND FLASH CONTROLLER	51
9.1 NAND Flash Controller Register.....	51
9.2 Default State.....	57
9.3 User Define Protocol.....	58
9.4 Cycle Timing.....	62
9.5 Operation Flow.....	65
9.6 Packet size setting	74
CHAPTER 10 BCH-CODEC.....	75
10.1 BCH_Codec Register.....	75
10.2 Encode.....	77
10.3 Decode and correct data.....	78
CHAPTER 11 SD CONTROLLER	81
11.1 SDC Register.....	81
11.2 SD Mode Operation.....	93
11.3 SPI Mode Operation.....	95
11.4 EXINST Opcode Summary.....	97
CHAPTER 12 RANDOM	99
12.1 Random Register	99
12.2 Operation Guide.....	101
CHAPTER 13 ELECTRICAL CHARACTERISTICS.....	103
13.1 Absolute Maximum Ratings.....	103
13.2 Recommended Operating Conditions.....	103
13.3 Electrical Characteristics of 3.3V I/O Cell.....	103
13.4 Electrical Characteristics of RC.....	103
13.5 Electrical Characteristics of LDO.....	104
CHAPTER APPENDIX REVISION HISTORY.....	105

1 Product Overview

1.1 Description

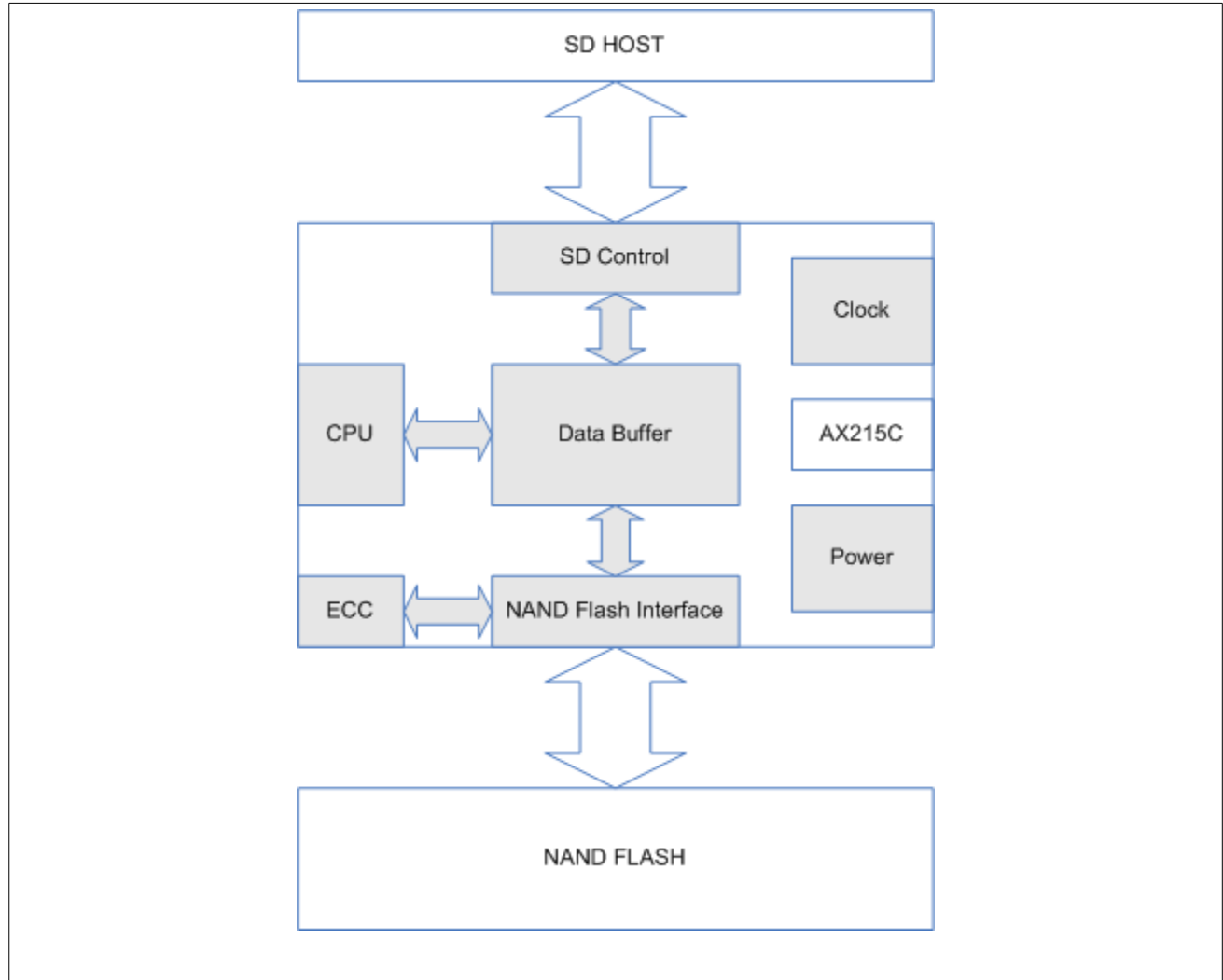
AX215E is a 8-bit 8051 microcontroller. It integrates advanced digital and analog peripherals to accommodate SD card applications.

Features:

- 8-bit 8051 CPU, with enhance extend instruction, optimized for SD, NAND flash control
- All instructions are single-cycled except branching instructions
- Fully supports SD device
- Supports high capacity standard
- Enhanced ESD protection
- Supports 528/2K/4K/8K bytes page NAND Flash
- Supports SLC/MLC/TLC NAND flash
- Supports X8/X16 flash data bus
- Supports ex3 flash data bus
- Supports Toggle DDR 1.0 flash data bus
- Supports ONFI 2.2 flash data bus
- Supports up to 4 flash chips
- Built in 17/25/29/31 bits on-the-fly ECC per sector
- Data protection during data transfer even if unplugged/power off
- On-chip POR and BOR
- On-chip Regulator
- Supports Idle mode and Sleep mode
- Supports MMC/eMMC device interface

1.2 System Architecture

Figure 1-1: System Block Diagram



2.1 Pin Assignment

Figure 2-1: Die Form PAD Location



PAD Name	X-axis (um)	Y-axis (um)
NDAT8		
NDAT9		
NDAT10		
NDAT11		
NDAT12		
NDAT13		
NDAT14		
NDAT15		
NDQS		
VSS		
NDAT0		
NDAT1		
NDAT2		
NDAT3		

NDAT4		
NDAT5		
NDAT6		
VDDCORE		
VDDCORE		
NDAT7		
NRB0		
NRB1		
VCC		
NRE_		
NCE1_		
NCE0_		
VSS		
NCE2_		
NCE3_		
NCLE		
NALE		
NWE_		
NWP_		
SDDAT2		
SDDAT3		
VDD18		
VDD18		
VCCQ		
VCCQ		
VSS		
SDCMD		
VSS		
SDCLK		
SDDAT0		
SDDAT1		
MMCDATA7		
MMCDATA6		
VCCQ		
VSS		
MMCDATA5		
MMCDATA4		
MMC_RSTN		
MMC_SD		
BP_DIS		

2.2 Pin Description

Table 2-2: Pin Description

Pin	Name	Direction	Functions
1	NDAT8	I/O	Flash Data 8 P20
2	NDAT9	I/O	Flash Data 9 P21
3	NDAT10	I/O	Flash Data 10 P22
4	NDAT11	I/O	Flash Data 11 P23
5	NDAT12	I/O	Flash Data 12 P24
6	NDAT13	I/O	Flash Data 13 P25
7	NDAT14	I/O	Flash Data 14 P26
8	NDAT15	I/O	Flash Data 15 P27
9	NDQS	I/O	Flash DQS P37
10	VSS	I	Ground
11	NDAT0	I/O	Flash Data 0 P10
12	NDAT1	I/O	Flash Data 1 P11
13	NDAT2	I/O	Flash Data 2 P12
14	NDAT3	I/O	Flash Data 3 P13
15	NDAT4	I/O	Flash Data 4 P14
16	NDAT5	I/O	Flash Data 5 P15
17	NDAT6	I/O	Flash Data 6 P16
18	VDDCORE	POWER OUT	1.2V core Power out
	VDDCORE	POWER OUT	1.2V core Power out
19	NDAT7	I/O	Flash Data 7 P17
20	NRB0	I/O	Flash Ready and Busy 0 P35
21	NRB1	I/O	Flash Ready and Busy 1 P36
22	VCC	POWER IN	Flash I/O power

23	NRE_	I/O	Flash Read Enable P34
24	NCE1_	I/O	Flash Chip Enable 1 P01
25	NCE0_	I/O	Flash Chip Enable 0 P00
26	VSS	GND	Ground
27	NCE2_	I/O	Flash Chip Enable 2 P02
28	NCE3_	I/O	Flash Chip Enable 3 P03
29	NCLE	I/O	Flash Command Latch Enable P31
30	NALE	I/O	Flash Address Latch Enable P32
31	NWE_	I/O	Flash Write Enable P33
32	NWP_	I/O	Flash Write Protect P30
33	SDDAT2	I/O	SD Data Line 2 P44
34	SDDAT3	I/O	SD Data Line 3 P45
35	VDD18	Power out	Supply power for 1.8V Nand Flash
	VDD18	Power out	Supply power for 1.8V Nand Flash
36	VCCQ	Power in	SD/eMMC power in
	VCCQ	Power	SD/eMMC power
37	VSS	I	Ground
38	SDCMD	I/O	SD Command Line P41
39	VSS	I	Ground
40	SDCLK	I	SD Clock Line P40
41	SDDAT0	I/O	SD Data Line 0 P42
42	SDDAT1	I/O	SD Data Line 1 P43
43	MMCDATA7	I/O	SD Data Line 7 P07
44	MMCDATA6	I/O	SD Data Line 7 P06
45	VCCQ	Power	SD/eMMC power
46	VSS	I	Ground
47	MMCDATA5	I/O	SD Data Line 7 P05
48	MMCDATA4	I/O	SD Data Line 7 P04
49	MMC_RSTN	I/O	MMC H/W reset
50	MMC_SD	I	MMC/SD MODE SEL

51	BP_DIS	I	BYPASS
----	--------	---	--------

3 Memory Mapping and CPU Architecture

3.1 Basic Structure

AX215E CPU is modified and optimized from the standard 8051/8052 architecture. It supports most standard 8051/8052 instructions except the 4 listed below:

MOVX A, @Ri

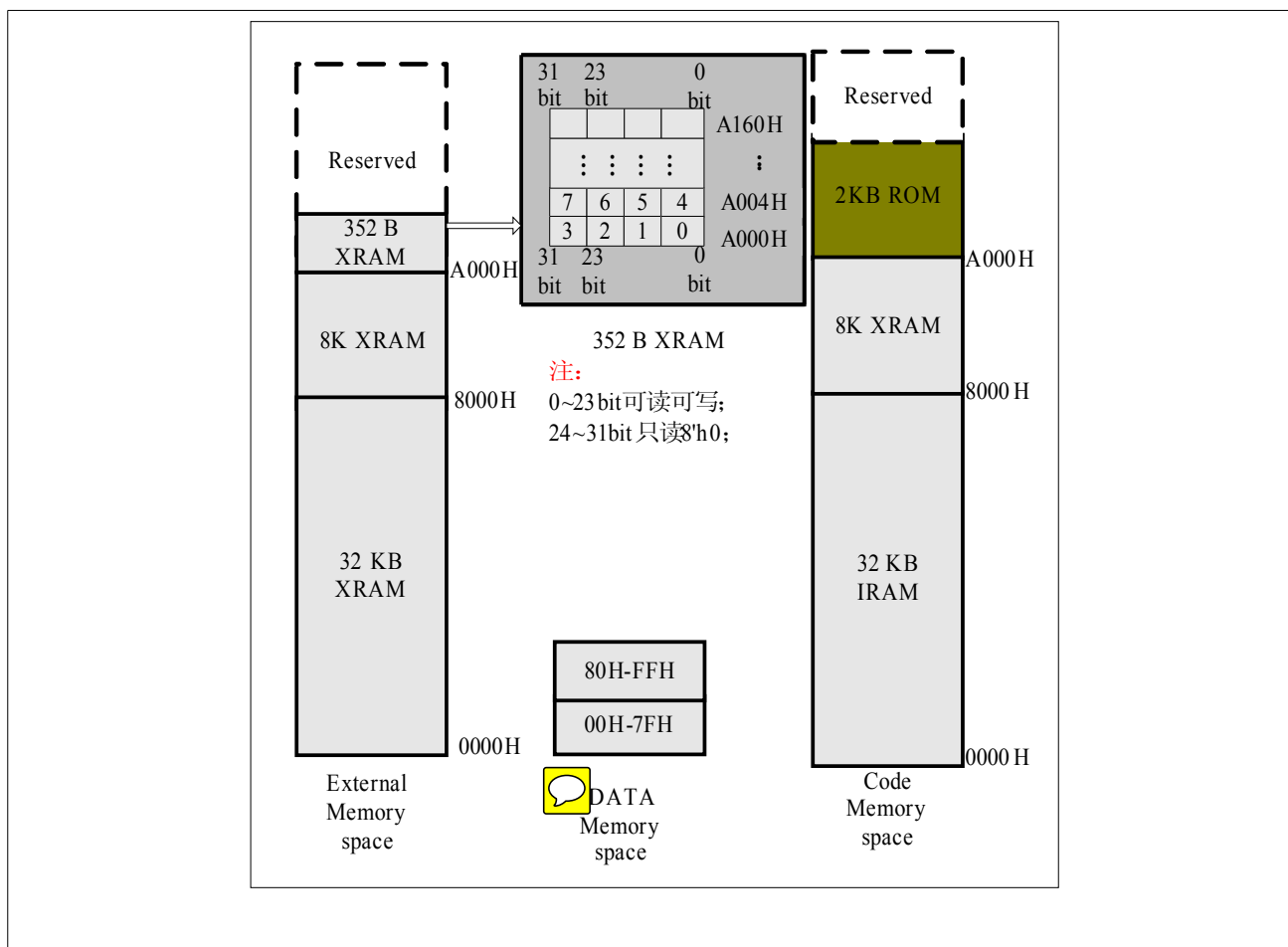
MOVX @Ri, A

MOVC A, @A+PC

DA

As most popular 8051/8052, AX215E supports 2 DPTRs for accessing the external memory space. However, the memory mapping is different, as shown in Figure 3-1.

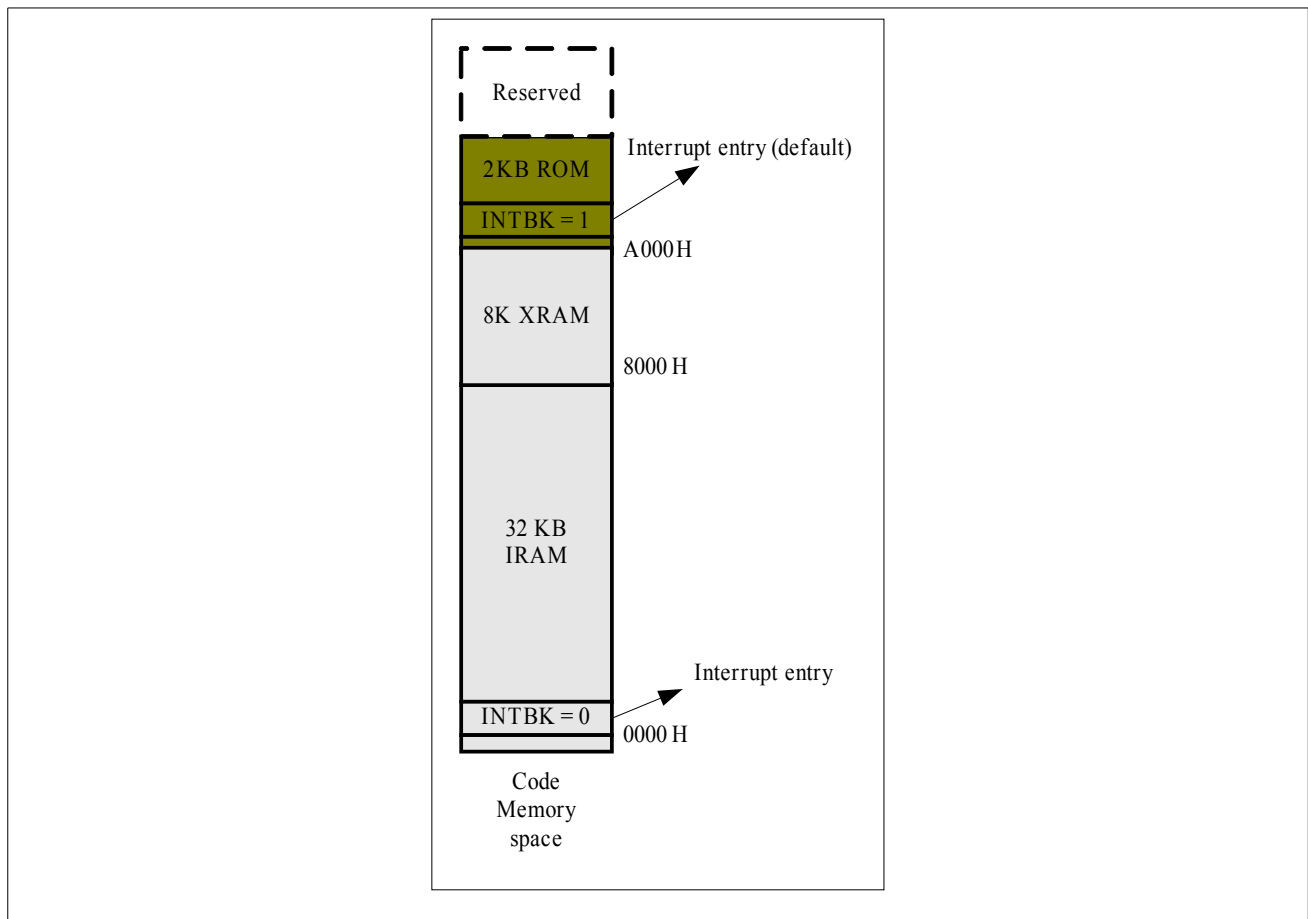
Figure 3-1: Memory Mapping



3.2 Interrupt Entry Address Mapping

After power up, the first instruction is start from A000h. That means CPU will run the boot loader program stored in MASK ROM first, and also the interrupt address is remapping start from A003h. By clearing the interrupt entry bank address control bit INTBK, the interrupt entries address will change start from 0003h as a standard 8051/8052.

Figure 3-2: Interrupt Entry Address Mapping



3.3 Data Memory – DRAM (DATA)

Data memory space (DATA) contains 256 bytes sketchpad memory, special function registers for controlling peripherals. It is also a super set of REGISTER space. To access this complex space, several addressing modes (direct, indirect, register and bit addressing) are provided. For details, please refer to the 8052 standard.

3.4 General Purpose Register (REGISTER)

General purpose registers R0 through R7 link to 32 bytes of internal data memory in the way that allows quick, efficient access. For example, the instruction `MOV A, 00h` using two bytes of code can be replaced by shorthand notation instruction `MOV A, R0` that uses one byte of code only.

These 32 bytes of memory are put into 4 banks. Any one of them within a bank is selected by R0 through R7. Desired register bank is selected using bits RS1 and RS0 in PSW, bit 4 and bit 3 respectively (please refer to Table 3-1 for setting description). This feature eliminates the effort required to backup the registers to stack memory during context switching, in addition provides more registers for complicated algorithms.

Table 3-1: Selection of Register Bank

RS1	RS0	Bank	Mapping Addressing to DATA
0	0	0	00h-07h
0	1	1	08h-0Fh
1	0	2	10h-17h
1	1	3	18h-1Fh

3.5 Extended Data Memory – XRAM (XDATA)

To provide a unified linear memory model, AX215E organizes internal data memory that out of the scope of DATA into extended data memory space (XDATA). Total (40K + 352) bytes of XDATA (Low 32K XRAM, shared with IRAM, and high 8K XRAM, and more high 352 B XRAM) are addressed through instruction *MOVX* using two 16-bit data pointers, DPTR0 and DPTR1.

To facilitate data movement in XDATA, AX215E provides dual data pointers. Two independent data pointers are especially useful when moving, copying and manipulating large data types, such arrays, structures and unions. Based on this feature, AX215E saves many operations in manipulating pointer, comparing to single data pointer classic 8051/8052. The additional data pointer can be selected through register DPCTL. No additional instruction is introduced to distinguish these 2 pointers.

XRAM is divided the three mem , 0x0000~0x7fff as low 32K XRAM address, 0x8000~0x9fff as high 8K XRAM address, 0xa000~0xa160 as more high 352B XRAM address, USB only can DMA low 32K XRAM, BCM only can DMA high 8K XRAM, SD、RCMP、NF and Core olny can DMA low 32K and high 8K XRAM, when more than one dma request the same XRAM at the same time, the high priority DMA request will generate associate ack, the different DMA at the different XRAM, which isn't priority.

Note: 352B XRAM 高的1B(24~31bits)是只读的，读的数是8'b0；低的3B(0~23bits)是可读可写的，所以可用的XRAM只有264B

3.6 CPU Instruction Set Summary

AX215E is a high performance MCU. It executes most instructions in 1 cycle, in contrast to 12 cycles in classic 8051/8052.

3.6.1 Basic Instruction Set

Table 3-2 lists the basic instructions of AX215E, these instruction are compatible with 8051/8052 ISA.

Table 3-2: Basic Instruction Set Summary

Mnemonics	Description
-----------	-------------

<i>ADD</i>	<i>A, Rn</i>	Add register to ACC
	<i>A, direct</i>	Add direct byte to ACC
	<i>A, @Ri</i>	Add indirect byte from DRAM to ACC
	<i>A, #data</i>	Add immediate to ACC
<i>ADDC</i>	<i>A, Rn</i>	Add register to ACC with carry
	<i>A, direct</i>	Add direct byte to ACC with carry
	<i>A, @Ri</i>	Add indirect byte from DRAM to ACC with carry
	<i>A, #data</i>	Add immediate to ACC with carry
<i>SUBB</i>	<i>A, Rn</i>	Subtract register to ACC with borrow
	<i>A, direct</i>	Subtract direct byte to ACC with borrow
	<i>A, @Ri</i>	Subtract indirect byte from DRAM to ACC with borrow
	<i>A, #data</i>	Subtract immediate to ACC with borrow
<i>INC</i>	<i>A</i>	Increment ACC
	<i>Rn</i>	Increment register
	<i>direct</i>	Increment direct byte
	<i>DPTR</i>	Increment the selected DPTR
	<i>@Ri</i>	Increment indirect byte of DRAM
<i>DEC</i>	<i>A</i>	Decrement ACC
	<i>Rn</i>	Decrement register
	<i>direct</i>	Decrement direct byte
	<i>DPTR</i>	Decrement the selected DPTR
	<i>@Ri</i>	Decrement indirect byte of DRAM
<i>MUL</i>	<i>AB</i>	Multiply ACC and B
<i>DIV</i>	<i>AB</i>	Divide ACC by B
<i>ANL</i>	<i>A, Rn</i>	AND register to ACC
	<i>A, direct</i>	AND direct byte to ACC
	<i>A, @Ri</i>	AND indirect DRAM to ACC
	<i>A, #data</i>	AND immediate to ACC
	<i>direct, A</i>	AND ACC to direct byte
	<i>direct, #data</i>	AND immediate to direct byte
	<i>C, bit</i>	AND direct bit to carry
	<i>C, /bit</i>	AND complement of direct bit to carry
<i>ORL</i>	<i>A, Rn</i>	OR register to ACC
	<i>A, direct</i>	OR direct byte to ACC
	<i>A, @Ri</i>	OR indirect DRAM to ACC
	<i>A, #data</i>	OR immediate to ACC
	<i>direct, A</i>	OR ACC to direct byte
	<i>direct, #data</i>	OR immediate to direct byte
	<i>C, bit</i>	OR direct bit to carry
	<i>C, /bit</i>	OR complement of direct bit to carry
<i>XRL</i>	<i>A, Rn</i>	XOR register to ACC

	<i>A, direct</i>	XOR direct byte to ACC
	<i>A, @Ri</i>	XOR indirect DRAM to ACC
	<i>A, #data</i>	XOR immediate to ACC
	<i>direct, A</i>	XOR ACC to direct byte
	<i>direct, #data</i>	XOR immediate to direct byte
<i>CLR</i>	<i>A</i>	Clear ACC
<i>CPL</i>	<i>A</i>	Complement ACC
<i>RL</i>	<i>A</i>	Rotate ACC left
<i>RLC</i>	<i>A</i>	Rotate ACC left through carry
<i>RR</i>	<i>A</i>	Rotate ACC right
<i>RRC</i>	<i>A</i>	Rotate ACC right through carry
<i>SWAP</i>	<i>A</i>	Swap nibbles of ACC
<i>MOV</i>	<i>A, Rn</i>	Move register to ACC
	<i>A, direct</i>	Move direct byte to ACC
	<i>A, @Ri</i>	Move indirect DRAM to ACC
	<i>A, #data</i>	Move immediate to ACC
	<i>Rn, A</i>	Move ACC to register
	<i>Rn, direct</i>	Move direct byte to register
	<i>Rn, #data</i>	Move immediate to register
	<i>direct, A</i>	Move ACC to direct byte
	<i>direct, Rn</i>	Move register to direct byte
	<i>direct, direct</i>	Move direct byte to direct byte
	<i>direct, @Ri</i>	Move indirect DRAM to direct byte
	<i>direct, #data</i>	Move immediate to direct byte
	<i>@Ri, A</i>	Move ACC to indirect DRAM
	<i>@Ri, direct</i>	Move direct byte to indirect DRAM
	<i>@Ri, #data</i>	Move immediate to indirect DRAM
	<i>DPTR, #data</i>	Load DPTR with 16-bit constant
	<i>C, bit</i>	Move direct bit to Carry
	<i>bit, C</i>	Move Carry to direct bit
<i>MOVC</i>	<i>A, @A+DPTR</i>	Move code byte relative DPTR to ACC
<i>MOVX</i>	<i>A, @DPTR</i>	Move external data (16-bit address) to ACC
	<i>@DPTR, A</i>	Move ACC to external data (16-bit address)
<i>PUSH</i>	<i>direct</i>	Push direct byte onto stack
<i>POP</i>	<i>direct</i>	Pop direct byte from stack
<i>XCH</i>	<i>A, Rn</i>	Exchange register with ACC
	<i>A, direct</i>	Exchange direct byte with ACC
	<i>A, @Ri</i>	Exchange indirect DRAM with ACC
<i>XCHD</i>	<i>A, @Ri</i>	Exchange low nibble of indirect DRAM with ACC
<i>CLR</i>	<i>C</i>	Clear carry
	<i>bit</i>	Clear direct bit

<i>SETB</i>	<i>C</i>	Set carry
	<i>bit</i>	Set direct bit
<i>CPL</i>	<i>C</i>	Complement carry
	<i>bit</i>	Complement direct bit
<i>JC</i>	<i>rel code</i>	Jump if carry is set
<i>JNC</i>	<i>rel code</i>	Jump if carry is not set
<i>JB</i>	<i>bit, rel code</i>	Jump if direct bit is set
<i>JNB</i>	<i>bit, rel code</i>	Jump if direct bit is not set
<i>JBC</i>	<i>bit, rel code</i>	Jump if direct bit is set and clear bit
<i>ACALL</i>	<i>page code</i>	Absolute subroutine call
<i>LCALL</i>	<i>long code</i>	Long subroutine call
<i>RET</i>		Return from subroutine
<i>RETI</i>		Return from interrupt
<i>AJMP</i>	<i>page code</i>	Absolute jump
<i>LJMP</i>	<i>long code</i>	Long jump
<i>SJMP</i>	<i>rel addr</i>	Short jump (relative address)
<i>JMP</i>	<i>@A+DPTR</i>	Jump indirect relative to DPTR
<i>JZ</i>	<i>rel code</i>	Jump if ACC equals zero
<i>JNZ</i>	<i>rel code</i>	Jump if ACC does not equal zero
<i>CJNE</i>	<i>A, direct, rel code</i>	Compare direct byte to ACC and jump if not equal
	<i>A, #data, rel code</i>	Compare immediate to ACC and jump if not equal
	<i>Rn, #data, rel code</i>	Compare immediate to Register and jump if not Equal
	<i>@Ri, #data, rel code</i>	Compare immediate to indirect and jump if not Equal
<i>DJNZ</i>	<i>Rn, rel code</i>	Decrement Register and jump if not zero
	<i>direct, rel code</i>	Decrement direct byte and jump if not zero
<i>NOP</i>		No operation

Notes:**Rn:**

Register R0-R7 of the currently selected register bank.

@Ri:

Data RAM location addressed indirectly through R0 or R1.

rel code:

8-bit, signed (2s complement) offset relative to the first byte of the following instruction. Used by SJMP and all conditional jumps.

direct:

8-bit internal data location's address. This could be a direct-access Data RAM location (0x00-0x7F) or an SFR (0x80-0xFF).

#data:

8-bit immediate data

#data16:

16-bit immediate data

bit:

Direct-accessed bit in Data RAM or SFR

page code:

11-bit destination address used by ACALL and AJMP. The destination must be within the same 2K-byte page of program memory as the first byte of the following instruction.

long code:

16-bit destination address used by LCALL and LJMP. The destination may be anywhere within the 64K-byte program memory space.

3.6.2 Extended Instruction Set

Base on the basic instruction set, AX215E implements a set of instructions (mainly 32-bit operation) to improve the CPU performance. These instructions are named Extended Instructions (EXINST).

In the EXINST architecture, 5 sets 32-bit register are implemented, they are *ER0*, *ER1*, *ER2*, *ER3*, *ER4*. Each of them is organized by 4 SFRs:

ER0 = {*R03*, *R02*, *R01*, *R00*}

ER1 = {*R13*, *R12*, *R11*, *R10*}

ER2 = {*R23*, *R22*, *R21*, *R20*}

ER3 = {*R33*, *R32*, *R31*, *R30*}

ER4 = {*R43*, *R42*, *R41*, *R40*}

NOTE:

1. These 40 byte registers and R8 can be used as Direct addressing by Basic Instructions
2. *ER4* can only be used by instruction MOV32 *ER4*,*ERn* or MOV32 *ERn*,*ER4*

Most of EXINST operations are based on these *ERs*, and some EXINST also use *ACC*, *B*, PC, or *R8* as source operands.

EXINST are described below:

Table 3-3: Extended Instructions Set Summary

	Mnemonics	Description
MOV32	<i>ERn</i> , EDPI	Fetch 32-bit data from XRAM and store into <i>ERn</i>
	EDPI, <i>ERn</i>	Store 32-bit data from <i>ERn</i> to XRAM
	<i>ERn</i> , <i>ERm</i>	Move 32-bit data from <i>ERm</i> to <i>ERn</i>
	<i>ER4</i> , <i>ERn</i>	Move 32-bit data from <i>ERn</i> to <i>ER4</i>
	<i>ERn</i> , <i>ER4</i>	Move 32-bit data from <i>ER4</i> to <i>ERn</i>
NOT32	<i>ERn</i>	Complement <i>ERn</i>
INC32	<i>ERn</i>	Increment <i>ERn</i>
DEC32	<i>ERn</i>	Decrement <i>ERn</i>
RAN32	EDPI, <i>ERn</i>	Store <i>ERn</i> to XRAM, and <i>ERn</i> updated with 32-bit pseudo random algorithm

ANL32	ERn, EDPi	AND the 32-bit data from XRAM and <i>ERn</i> , store the result into <i>ERn</i>
	EDPi, ERn	AND the 32-bit data from XRAM and <i>ERn</i> , store the result into XRAM
	ERn, ERm	AND <i>ERn</i> and <i>ERm</i> , store the result into <i>ERn</i>
ORL32	ERn, EDPi	OR the 32-bit data from XRAM and <i>ERn</i> , store the result into <i>ERn</i>
	EDPi, ERn	OR the 32-bit data from XRAM and <i>ERn</i> , store the result into XRAM
	ERn, ERm	OR <i>ERn</i> and <i>ERm</i> , store the result into <i>ERn</i>
XRL32	ERn, EDPi	XOR the 32-bit data from XRAM and <i>ERn</i> , store the result into <i>ERn</i>
	EDPi, ERn	XOR the 32-bit data from XRAM and <i>ERn</i> , store the result into XRAM
	ERn, ERm	XOR <i>ERn</i> and <i>ERm</i> , store the result into <i>ERn</i>
ADD32	ERp, EDPi, ERn	ADD the 32-bit data from XRAM and <i>ERn</i> , store the result into <i>ERp</i>
	EDPi, ERn, ERp	ADD <i>ERn</i> and <i>ERp</i> , store the result into XRAM
	ERp, ERn, ERm	ADD <i>ERn</i> and <i>ERm</i> , store the result into <i>ERp</i>
SUB32	ERp, EDPi, ERn	SUB <i>ERn</i> from the 32-bit data from XRAM, store the result into <i>ERp</i>
	EDPi, ERn, ERp	SUB <i>ERp</i> from <i>ERn</i> , store the result into XRAM
	ERp, ERn, ERm	SUB <i>ERm</i> from <i>ERn</i> , store the result into <i>ERp</i>
ROTR32	ERn, ER8	Rotate right <i>ERn</i> R8[4:0] bits, store the result into <i>ERn</i>
ROTL32	ERn, ER8	Rotate left <i>ERn</i> R8[4:0] bits, store the result into <i>ERn</i>
ROTR8	EACC, ER8	Rotate right <i>ACC</i> R8[2:0] bits, store the result into <i>ACC</i>
ROTL8	EACC, ER8	Rotate left <i>ACC</i> R8[2:0] bits, store the result into <i>ACC</i>
CLR32	ERn	Clear <i>ERn</i>
ADDDPi		ADD <i>DPTR</i> and { <i>R8</i> , <i>B</i> }, store the result into <i>DPTR</i>
SUBDPi		SUB { <i>R8</i> , <i>B</i> } from <i>DPTR</i> , store the result into <i>DPTR</i>
INCDPi		<i>DPTR</i> INC 1
DECDPi		<i>DPTR</i> DEC 1
INC4DPi		<i>DPTR</i> INC 4
DEC4DPi		<i>DPTR</i> DEC 4
MUL16	ERn	MUL <i>ERn</i> [15:0] and <i>ERn</i> [31:16], store the result into <i>ERn</i> .
DIV16	ERn	DIV <i>ERn</i> by { <i>R8</i> , <i>B</i> }, store the result into <i>ERn</i> , Store the Remainder to { <i>R8</i> , <i>B</i> }

Notes:**ERn, ERm, ERp:**32-bit register *ER0-ER3*. *n=m=p* is allowed.**ERni, ERmj:**8-bit SFR *R00-R03*, *R10-R13*, *R20-R23*, *R30-R33*. *ni=mj* is allowed.**EDPi:**XRAM addressing pointer *DPTR0* or *DPTR1*.**ER8:**8-bit SFR *R8*.**EACC:**8-bit SFR *ACC*.**Indirect Addressing:**

EXINST support indirect accessing XRAM by using *DPTR0* or *DPTR1*, the width unit of data in XRAM is 32-bit for the other EXINST. That means the LSB2-LSB0 or LSB1-LSB0 of *DPTR* are ignored during these operations. And also, *DPTR* will INC/DEC by 4 or 1 after executed if *DPAID* = 1.

When the data is fetching from XRAM to CPU or storing from CPU to XRAM, little-endian or big-endian data format is switched by *CHEND*. It's illustrated in Figure 3-3.

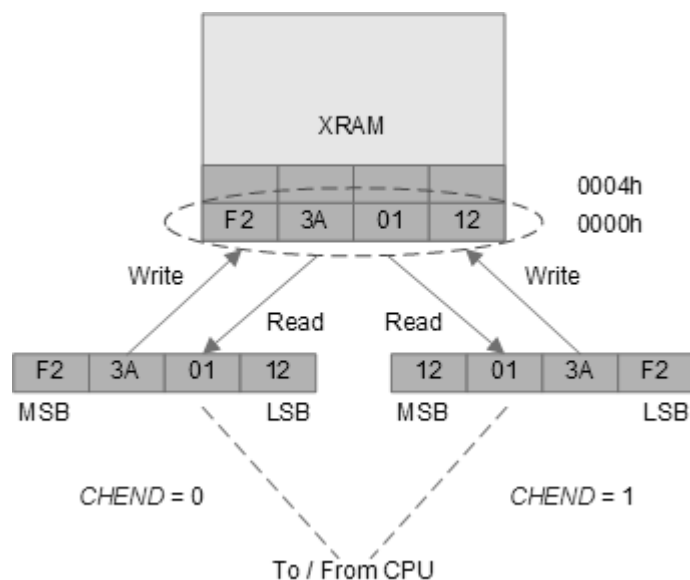


Figure 3-3: Endian Switching

Rotate Operation:

By default, the rotate instruction *ROTR32* support that the data rotates right by *N* bits, the *N* is the immediate data from opcode or value of *R8*. However, *LSHF* and *ASHF* can change the operation as Figure 3-4 shows.

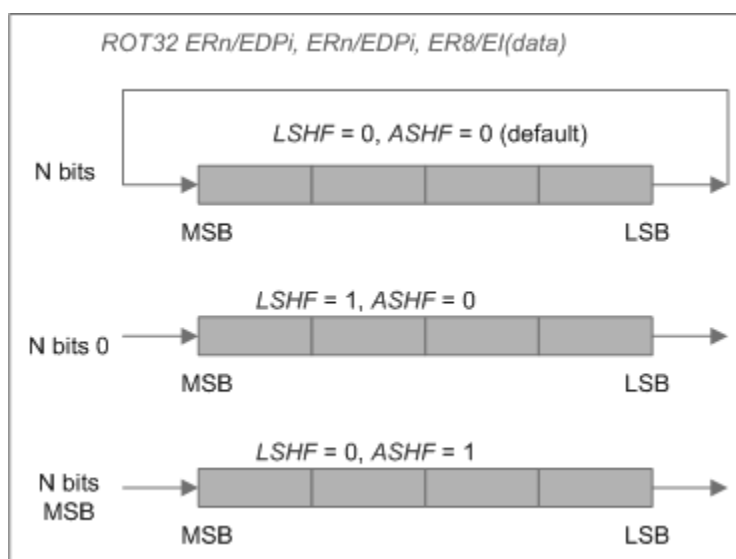


Figure 3-4: Rotate Operation

Note:

When LSHF = 1 and ASHF = 1 at the same time, LSHF is don't care.

4 Special Function Register (SFR)

4.1 SFR List

The unimplemented bits are labelled '-', never write value other than its reset value to it, otherwise unpredictable effects will be resulted. Some registers have undetermined reset value, it is labelled 'x'.

Table 4-1: SFR List


Func.	Name	Addr	Des.	Reset Value
CPU & INT	DPCTL	80h	DPTR Control Register	0000 0000
	SP	81h	Stack Pointer	0111 1111
	DP0L	82h	DPTR0 Low byte	0000 0000
	DP0H	83h	DPTR0 High byte	0000 0000
	DP1L	84h	DPTR1 Low byte	0000 0000
	DP1H	85h	DPTR1 High byte	0000 0000
	IPE	A7h	Interrupt Enable Register 1	0000 0000
	IE	A8h	Interrupt Enable Register	0000 0000
	IP	A9h	Interrupt Priority Register	0000 0000
	R8	B7h	Register 8	xxxx xxxx
	R00	C0h	Register 00	xxxx xxxx
	R01	C1h	Register 01	xxxx xxxx
	R02	C2h	Register 02	xxxx xxxx
	R03	C3h	Register 03	xxxx xxxx
	R40	C4h	Register 40	xxxx xxxx
	R41	C5h	Register 41	xxxx xxxx
	R42	C6h	Register 42	xxxx xxxx
	R43	C7h	Register 43	xxxx xxxx
	R10	C8h	Register 10	xxxx xxxx
	R11	C9h	Register 11	xxxx xxxx
	R12	CAh	Register 12	xxxx xxxx
	R13	CBh	Register 13	xxxx xxxx
	PSW	D0h	Program Status Register	0000 0000
	R20	D8h	Register 20	xxxx xxxx
	R21	D9h	Register 21	xxxx xxxx
	R22	DAh	Register 22	xxxx xxxx
	R23	DBh	Register 23	xxxx xxxx
	ACC	E0h	Accumulator	0000 0000

	B	F0h	Register B	0000 0000
	R30	F8h	Register 30	xxxx xxxx
	R31	F9h	Register 31	xxxx xxxx
	R32	FAh	Register 32	xxxx xxxx
	R33	FBh	Register 33	xxxx xxxx
NAND Flash	NMCTL1	8Eh	NAND Flash Mode Control Register 1	0000 0000
	NTCTL2	8Fh	NAND Flash Mode Control Register 2	0000 0000
	NXCNT0	A6h	NAND Flash DMA Start Address 0 Count	0000 0000
	NFIFO0	9Bh	NAND Flash FIFO 0	xxxx xxxx
	NFIFO1	9Ch	NAND Flash FIFO 1	xxxx xxxx
	NFIFO2	9Dh	NAND Flash FIFO 2	xxxx xxxx
	NFIFO3	9Eh	NAND Flash FIFO 3	xxxx xxxx
	NCEE	91h	NAND Flash CE Enable Register	0000 0000
	NMCTL	92h	NAND Flash Mode Control Register	0011 1000
	NTCTL0	93h	NAND Flash Timing Control 0 Register	1111 1111
	NPCTL	94h	NAND Flash Protocol Control Register	0000 0000
	NXADR0	95h	NAND Flash DMA XRAM Start Address 0 Register	xxxx xxxx
	NXADR1	96h	NAND Flash DMA XRAM Start Address 1 Register	xxxx xxxx
	NCMD1	97h	NAND Flash Command 1 Register	xxxx xxxx
	NTCTL1	98h	NAND Flash Timing Control 1 Register	0000 1111
	NPGSZ0	99h	NAND Flash Page Size 0 Register	xxxx xxxx
	NPGSZ1	9Ah	NAND Flash Page Size 1 Register	xxxx xxxx
BCH Codec	BXADR0	86h	BCH Codec DMA XRAM Start Address 0 Register	xxxx xxxx
	BXADR1	87h	BCH Codec DMA XRAM Start Address 1 Register	xxxx xxxx
	BCTL0	88h	BCH Codec Control Register	0000 xxxx
	BCTL1	89h	BCH Codec Control Register 1	0000 0000
	BECNT	8Ah	BCH Codec ERROR Counter Register	0000 0000
	BXADR2	8Bh	BCH Codec DMA XRAM Start Address 2 Register	0000 0000
	BF0CNT	8Ch	BCH Codec DMA DATA BUFFER 0 Counter Register	0000 0000
	BPSZ	8Dh	BCH-Codec DMA DATA packet size Register	0000 0000
SD	SFLAG	90h	State Flag	000x 00x0
	SHCTL	CDh	SD Host Control	xxxx 001x
	SR1CTL	CEh		0000 0011
	SSTA	CFh	Card State Config	0000 0000
	SDODLY	D1h	Rps delay cfg	0000 0000
	SRCA0	D2h	SD RCA 0 Register	Xxxx xxxx
	SRCA1	D3h	SD RCA 1 Register	Xxxx xxxx
	SDXADR0	D4h	SD Data DMA XRAM Start Address 0 Register	Xxxx xxxx
	SDXADR1	D5h	SD Data DMA XRAM Start Address 1 Register	Xxxx xxxx
	SRXADR0	D6h	SD Response DMA XRAM Start Address 0 Register	Xxxx xxxx
	SRXADR1	D7h	SD Response DMA XRAM Start Address 1 Register	Xxxx xxxx

	SDICTL	DCh	SD data RX config register	0000 0000
	SDOTK	DDh	SD Data Output Token Register	Xxxx xxxx
	SRDL0	DEh	SD Response DMA Length 0 Register	0000 0011
	SRDL1	DFh	SD Response DMA Length 1 Register	0000 0xx0
	SDDL0	E1h	SD Data DMA Length 0 Register	Xxxx xxxx
	SDDL1	E2h	SD Data DMA Length 1 Register	Xxxx xxxx
	SCCTL	E3h	SD Command Control Register	0000 0100
	SRDCTL	E4h	SD Response Delay Control Register	Xxxx xxxx
	SDOCTL	E5h	SD data TX config register	00x0 0000
	SINDX	E6h	SD Command Index Register	00xx xxxx
	SRCTL	E7h	SD Response Control Register	0000 0000
	SPEND	E8h	SD Pending Flag Register	X000 0000
	SARG0	E9h	SD Argument Register0	xxxx xxxx
	SARG1	EAh	SD Argument Register1	xxxx xxxx
	SARG2	EBh	SD Argument Register2	xxxx xxxx
	SARG3	ECh	SD Argument Register3	xxxx xxxx
	SPICNT	ADh	SD Host clock counter	1000 0000
	LAST_CMD	BAH	SD Last CMD Register	xxxx xxxx
	BOOT_CFG	BDH	SD boot mode configure Register	0001 0000
Timer	TCTL	B0h	Timer Control Register	0000 0000
	TCNT	B1h	Timer Counter Register	xxxx xxxx
	TPR	B2h	Timer Period Register	xxxx xxxx
	SBUF	B3h	Serial Port Buffer Register	0000 0000
RAN DOM	CMPCTL	AAh	random compare control	0000 0000
	CMPFIFO	ABh	random compare fifo	0000 0000
	CPERCNT	ACh	random compare error count	0000 0000
	CMPCTL1	A Eh	random compare control 1	- - - - 0000
	CMPSTADR	AFH	compare start addr	0000 0000
	CMPDMAMT	9FH	DMA 32bit data amount	0000 0000
BCM	BCMCNT	B7h	bad column management Counter Register	xxxx xxxx
	BCMDATA	A3h	bad column management data	xxxx xxxx
	BCMCTL	A4h	bad column management Control Register	- - - - -00
	BCMADDR	A5h	bad column management begin addr	xxxx xxxx
GP DMA	GPDMACNT	BBH	General Purpose DMA Counter Register	xxxx xxxx
	GPDMAPT1	BCH	General Purpose DMA port 1	xxxx xxxx
	GPDMAPT0	EDH	General Purpose DMA port 0	xxxx xxxx
	GPDMACON	F1H	General Purpose DMA Control Register	- - - 0 1000
Clock & Port	PCTL	A0h	Power Control Register	1111 1100
	CCTL	A1h	Clock Control Register	0000 0011

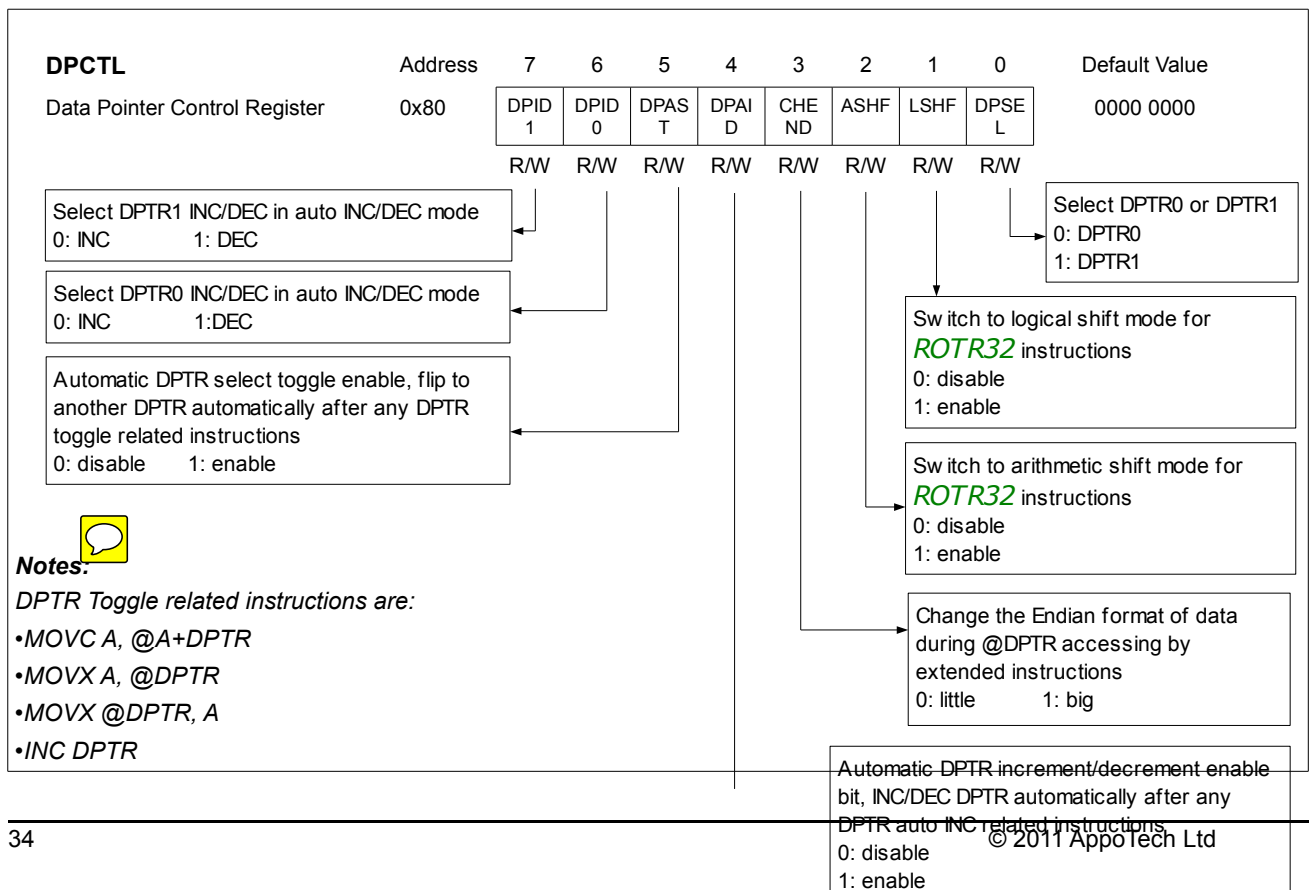
	RCCTL	B4h	RC OSC Control Register	0000 0000
	EXINTR	B5h	External Interrupt Control Register	0000 0000
	WKFLAG	B9h	Wait Flag for Wakup IDLE	0000 0000
	SCDATA	EEh	sfr data register	xxxx xxxx
	SCADDR	EFh	sfr addr register	---x xxxx
	P0	F3h	Port 0 Register	xxxx xxxx
	P1	F4h	Port 1 Register	xxxx xxxx
	P2	F5h	Port 2 Register	xxxx xxxx
	P3	F6h	Port 3 Register	0001 1000
	P4	F7h	Port 4	00xx xxxx
	P0DIR	FCh	Port 0 Direction Register	0000 1111
	P1DIR	FDh	Port 1 Direction Register	1111 1111
	P2DIR	FEh	Port 2 Direction Register	1111 1111
	P3DIR	FFh	Port 3 Direction Register	0110 0000
Other	CHIP_IDH	BFh	CHIP ID register	0000 0000
	CHIP_IDL	BEh	CHIP ID register	0001 1110
	CHIP_DCN	A2h	CHIP DCN register	0000 0000

Notes:

: bit accessible

4.2 SFR Description

Register 4-1: DPCTL – Data Pointer Control Register



•MOV DPTR, #data16

DPTR Auto INC related instructions are:

•MOVC A, @A+DPTR

•MOVX A, @DPTR

•MOVX @DPTR, A

Register 4-2: SP – Stack Pointer Register

SP	Address	7	6	5	4	3	2	1	0	Default Value
Stack Pointer Control Register	0x81	SP								0111 1111
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: SP default value is 80h, that's different from a standard 8051 CPU.

Register 4-3: DP0L – Data Pointer 0 Low Byte Register

DP0L	Address	7	6	5	4	3	2	1	0	Default Value
Data Pointer 0 Low Byte Register	0x82	DP0L								0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Register 4-4: DP0H – Data Pointer 0 High Byte Register

DP0H	Address	7	6	5	4	3	2	1	0	Default Value
Data Pointer 0 High Byte Register	0x83	DP0H								0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Register 4-5: DP1L – Data Pointer 1 Low Byte Register

DP1L	Address	7	6	5	4	3	2	1	0	Default Value
Data Pointer 1 Low Byte Register	0x84	DP1L								0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Register 4-6: DP1H – Data Pointer 1 High Byte Register

DP1H	Address	7	6	5	4	3	2	1	0	Default Value
Data Pointer 1 High Byte Register	0x85	DP1H								0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Register 4-7: IPE – Interrupt Enable Register 1

IPE	Address	7	6	5	4	3	2	1	0	Default Value
Interrupt Enable Register 1	0xA7	BOR BEN_	SPIP	TMRI P	EXIP	IPE[3]	SPIE	TMRI E	EXIE	0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Bit	Field	Mode	Description	Setting	
7	BORBEN_	R/W	BORB enable	0: enable	1: disable
6	SPIP	R/W	Serial Port Interrupt Priority Control	0: low	1: high
5	TMRIP	R/W	Timer Interrupt Priority Control	0: low	1: high
4	EXIP	R/W	Port Wakeup Interrupt Priority Control	0: low	1: high
3	IPE[3]	R/W	General purpose sfr or wake up IDLE control	0: disable	1: enable
2	SPIE	R/W	Serial Port Interrupt Enable Bit	0: disable	1: enable
1	TMRIE	R/W	Timer Interrupt Enable Bit	0: disable	1: enable
0	EXIE	R/W	Port Wakeup Interrupt Enable Bit	0: disable	1: enable

Register 4-8: IE - Interrupt Enable Register

IE	Address	7	6	5	4	3	2	1	0	Default Value
Interrupt Enable Register	0xA8	EA	BSIE	BIE	NFIE	SDO	SDI	SDR	SDC	0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Bit	Field	Mode	Description	Setting	
7	EA	R/W	Global Interrupt Enable Bit	0: disable	1: enable
6	BSIE	R/W	ECC Step 1 Interrupt Enable Bit	0: disable	1: enable
5	BIE	R/W	ECC Interrupt Enable Bit	0: disable	1: enable
4	NFIE	R/W	NAND Flash Control Interrupt Enable Bit	0: disable	1: enable
3	SDO	R/W	SD Data Output Interrupt Enable Bit	0: disable	1: enable
2	SDI	R/W	SD Data In Interrupt Enable Bit	0: disable	1: enable
1	SDR	R/W	SD Response Interrupt Enable Bit	0: disable	1: enable
0	SDC	R/W	SD Command Interrupt Enable Bit	0: disable	1: enable

Register 4-9: IP - Interrupt Priority Register

IP		Address	7	6	5	4	3	2	1	0	Default Value
Interrupt Priority Register		0xA9	HWRST_INT_EN	BSIP	BIP	NFIP	SDO	SDI	SDR	SDC	0000 0000
			-	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Bit	Field	Mode	Description					Setting			
7	HWRST_INT_EN	R/W	eMMC Hardware reset interrupt enable					1: enable 0: disable(default)			
6	BSIP	R/W	ECC Step 1 Interrupt Priority Control					0: low 1: high			
5	BIP	R/W	ECC Interrupt Priority Control					0: low 1: high			
4	NFIP	R/W	NAND Flash Control Interrupt Priority Control					0: low 1: high			
3	SDO	R/W	SD Data Output Interrupt Priority Control					0: low 1: high			
2	SDI	R/W	SD Data Input Interrupt Priority Control					0: low 1: high			
1	SDR	R/W	SD Response Interrupt Priority Control					0: low 1: high			
0	SDC	R/W	SD Command Interrupt Priority Control					0: low 1: high			

Register 4-10: WKFLAG - Wait Flag for Wakeup IDLE Register

WKFLAG		Address	7	6	5	4	3	2	1	0	Default Value
Wait Flag for Wakup IDLE Register		0xB9	WIPE3	WBS	WB	WNF	WSO	WSI	WSR	WEX	0000 0000
			W0	W0	W0	W0	W0	W0	W0	W0	
Bit	Field	Mode	Description					Setting			
7	WIPE3	WO						-			
6	WBS	WO	Wait BCH Syndr pending					0: low 1: high			
5	WB	WO	Wait BCH pending					0: low 1: high			
4	WNF	WO	Wait nand flash pending					0: low 1: high			
3	WSO	WO	Wait SD data output pending					0: low 1: high			
2	WSI	WO	Wait SD data input pending					0: low 1: high			
1	WSR	WO	Wait SD response pending					0: low 1: high			
0	WEX	WO	Wait external INT pending					0: low 1: high			

Note: When write WKFLAG register, CPU will enter IDLE.
WKFLAG register is clear by corresponding waiting flag

Register 4-11: R8 – Register 8

R8	Address	7	6	5	4	3	2	1	0	Default Value
Register 8	0xB8	R8								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R8 has no default value, thus it should be initialized before using.

Register 4-12: R00 – Register 00

R00	Address	7	6	5	4	3	2	1	0	Default Value
Register 00	0xC0	R00								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R00 has no default value, thus it should be initialized before using.

Register 4-13: R01 – Register 01

R01	Address	7	6	5	4	3	2	1	0	Default Value
Register 01	0xC1	R01								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R01 has no default value, thus it should be initialized before using.

Register 4-14: R02 – Register 02

R02	Address	7	6	5	4	3	2	1	0	Default Value
Register 02	0xC2	R02								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R02 has no default value, thus it should be initialized before using.

Register 4-15: R03 – Register 03

R03	Address	7	6	5	4	3	2	1	0	Default Value
Register 03	0xC3	R03								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R03 has no default value, thus it should be initialized before using.

Register 4-16: R10 – Register 10

R10	Address	7	6	5	4	3	2	1	0	Default Value
Register 10	0xC8	R10								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R10 has no default value, thus it should be initialized before using.

Register 4-17: R11 – Register 11

R11	Address	7	6	5	4	3	2	1	0	Default Value
Register 11	0xC9	R11								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R11 has no default value, thus it should be initialized before using.

Register 4-18: R12 – Register 12

R12	Address	7	6	5	4	3	2	1	0	Default Value
Register 12	0xCA	R12								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R12 has no default value, thus it should be initialized before using.

Register 4-19: R13 – Register 13

R13	Address	7	6	5	4	3	2	1	0	Default Value
Register 13	0xCB	R13								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R13 has no default value, thus it should be initialized before using.

Register 4-20: PSW – Program Status Register

PSW

Program Status Register

Address	7	6	5	4	3	2	1	0	Default Value
0xD0	CY	AC	EC	RS1	RS0	OV	EZ	P	0000 0000
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Bit	Field	Mode	Description	Setting
7	CY	R/W	Carry flag from bit 7	-
6	AC	R/W	Auxiliary carry flag	-
5	EC	R/W	Extended carry flag, only affect by extend instruction	-
4	RS1	R/W	Register bank selector 00: bank0 01: bank 1	-
3	RS0	R/W	10: bank2 11: bank3	-
2	OV	R/W	Overflow flag	-
1	EZ	R/W	Extended zero flag, only affect by extend instructions	-
0	P	R/W	Parity flag	-

Register 4-21: R20 – Register 20**R20**

Register 20

Address	7	6	5	4	3	2	1	0	Default Value
0xD8	R20								xxxx xxxx
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R20 has no default value, thus it should be initialized before using.

Register 4-22: R21 – Register 21**R21**

Register 21

Address	7	6	5	4	3	2	1	0	Default Value
0xD9	R21								xxxx xxxx
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R21 has no default value, thus it should be initialized before using.

Register 4-23: R22 – Register 22**R22**

Register 22

Address	7	6	5	4	3	2	1	0	Default Value
0xDA	R22								xxxx xxxx
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R22 has no default value, thus it should be initialized before using.

Register 4-24: R23 – Register 23

R23	Address	7	6	5	4	3	2	1	0	Default Value
Register 23	0xDB	R23								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R23 has no default value, thus it should be initialized before using.

Register 4-25: ACC – Accumulator

ACC	Address	7	6	5	4	3	2	1	0	Default Value
Accumulator	0xE0	ACC								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Register 4-26: B – B Register

B	Address	7	6	5	4	3	2	1	0	Default Value
B Register	0xF0	B								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Register 4-27: R30 – Register 30

R30	Address	7	6	5	4	3	2	1	0	Default Value
Register 30	0xF8	R30								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R30 has no default value, thus it should be initialized before using.

Register 4-28: R31 – Register 31

R31	Address	7	6	5	4	3	2	1	0	Default Value
Register 31	0xF9	R31								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R31 has no default value, thus it should be initialized before using.

Register 4-29: R32 – Register 32

R32	Address	7	6	5	4	3	2	1	0	Default Value
Register 32	0xFA	R32								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R32 has no default value, thus it should be initialized before using.

Register 4-30: R33 – Register 33

R33	Address	7	6	5	4	3	2	1	0	Default Value
Register 33	0xF8	R33								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R33 has no default value, thus it should be initialized before using.

Register 4-31: R40 – Register 40

R40	Address	7	6	5	4	3	2	1	0	Default Value
Register 40	0xC4	R40								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R40 has no default value, thus it should be initialized before using.

Register 4-32: R41 – Register 41

R41	Address	7	6	5	4	3	2	1	0	Default Value
Register 41	0xC5	R41								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R41 has no default value, thus it should be initialized before using.

Register 4-33: R42 – Register 42

R42	Address	7	6	5	4	3	2	1	0	Default Value
Register 42	0xC6	R42								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R42 has no default value, thus it should be initialized before using.

Register 4-34: R43 – Register 43

R43	Address	7	6	5	4	3	2	1	0	Default Value
Register 43	0xC7	R43								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note: R43 has no default value, thus it should be initialized before using.

Register 4-35: CHIP_IDH – Register 42

CHIP_IDH	Address	7	6	5	4	3	2	1	0	Default Value
	0xBF	0000_0000								0000_0000
		RO	RO	RO	RO	RO	RO	RO	RO	

Register 4-36: CHIP_IDL – Register 42

CHIP_IDL	Address	7	6	5	4	3	2	1	0	Default Value
	0xBE	0000_1100								0001_1110
		RO	RO	RO	RO	RO	RO	RO	RO	

Register 4-37: CHIP_DCN – Register 42

CHIP_DCN	Address	7	6	5	4	3	2	1	0	Default Value
	0xC0	0000_0000								0000_0000
		RO	RO	RO	RO	RO	RO	RO	RO	

Register 4-38: SCADDR -ADDR_SFR

SCADDR	Address	7	6	5	4	3	2	1	0	Default Value
	0xEF	-	-	-	SCADDR					xxxx_xxxx
					WO	WO	WO	WO	WO	

sfr_addr:

`define adr_p3hd_h 5'h0e

`define adr_p2hd_h 5'h0d

```

`define adr_p1hd_h    5'h0c
`define adr_p0hd_h    5'h0b
`define adr_p4hd      5'h0a
`define adr_aipcon     5'h09
`define adr_p4plp     5'h08
`define adr_p3plp     5'h07
`define adr_p2plp     5'h06
`define adr_p1plp     5'h05
`define adr_p0plp     5'h04
`define adr_p3hd      5'h03
`define adr_p2hd      5'h02
`define adr_p1hd      5'h01
`define adr_p0hd      5'h00

```

Register 4-39: SCDATA

SCDATA	Address	7	6	5	4	3	2	1	0	Default Value
	0xEE	SCDATA								xxxx_xxxx
		WO	WO	WO	WO	WO	WO	WO	WO	

操作方法：

比如你想打开P3口的上拉，具体操作如下：

```
MOV SCADDR,#addr_p3plp
```

```
MOV SCDATA,#DATA
```

如果你想读P3口的上拉值到A，具体操作如下：

```
MOV SCADDR,# addr_p3plp
```

```
MOV A,SCDATA
```

note:

关于驱动电流配置的描述：

p00~p03这四个IO的驱动电流,对应的配置位为：

p03	p02	p01	p00
SCDATA[7:6]	SCDATA[5:4]	SCDATA[3:2]	SCDATA[1:0]

```
MOV SCADDR,#addr_p0hd
```

```
MOV SCDATA,#DATA
```

p10~p13这四个IO的驱动电流,对应的配置位为：

p13	p12	p11	p10
SCDATA[7:6]	SCDATA[5:4]	SCDATA[3:2]	SCDATA[1:0]

```
MOV SCADDR,#addr_p1hd
```

```
MOV SCDATA,#DATA
```

p14~p17这四个IO的驱动电流,对应的配置位为：

p17	p16	p15	p14
SCDATA[7:6]	SCDATA[5:4]	SCDATA[3:2]	SCDATA[1:0]

MOV SCADDR,#addr_p1hd_h

MOV SCDATA,#DATA

p20~p23这四个IO的驱动电流,对应的配置位为：

p23	p22	p21	p20
SCDATA[7:6]	SCDATA[5:4]	SCDATA[3:2]	SCDATA[1:0]

MOV SCADDR,#addr_p2hd_h

MOV SCDATA,#DATA

p24~p27这四个IO的驱动电流,对应的配置位为：

p27	p26	p25	p24
SCDATA[7:6]	SCDATA[5:4]	SCDATA[3:2]	SCDATA[1:0]

MOV SCADDR,#addr_p2hd_h

MOV SCDATA,#DATA

p30~p33这四个IO的驱动电流,对应的配置位为：

p33	p32	p31	p30
SCDATA[7:6]	SCDATA[5:4]	SCDATA[3:2]	SCDATA[1:0]

MOV SCADDR,#addr_p3hd_h

MOV SCDATA,#DATA

p34~p37这四个IO的驱动电流,对应的配置位为：

p37	p36	p35	p34
SCDATA[7:6]	SCDATA[5:4]	SCDATA[3:2]	SCDATA[1:0]

MOV SCADDR,#addr_p3hd_h

MOV SCDATA,#DATA

对应的驱动电流为：

'00	4mA
'01	8mA
'10	12mA
'11	16mA

p04~p07这四个IO的驱动电流,对应的配置位为：(注意：p04~p07这4个IO的驱动只有两种选择)

p07	p06	p05	p04
SCDATA[3]	SCDATA[2]	SCDATA[1]	SCDATA[0]

MOV SCADDR,#addr_p0hd_h

MOV SCDATA,#DATA

对应的驱动电流为：

'0	12mA
'1	16mA

5 GPDMA

GPDMAPT0

Position	7	6	5	4	3	2	1	0
Name	GPDMAPT0L		GPDMAPT0H/ GPDMAPT0L					
Default	X	X	X	X	X	X	X	X
Access	W	W	W	W	W	W	W	W

GPDMAPT0: GPDMA pointer 0

这个寄存器在GPDMA模式时用于指定从memory的哪里开始读数据，即源地址。这个memory的mapping与CPU xdata一样。因为搬运数据时是以32bit为单位，所以这个寄存器有14位。

硬件纠错手动模式时，这个寄存器需要写入错误位置的起始地址，即(0XA000 + BXADR2)

。

操作这个寄存器需要写两次，第一次写高6bit，第二次写低8bit的地址；

GPDMAPT1

Position	7	6	5	4	3	2	1	0
Name	GPDMAPT1L		GPDMAPT1H/ GPDMAPT1L					
Default	X	X	X	X	X	X	X	X
Access	W	W	W	W	W	W	W	W

GPDMAPT1: GPDMA pointer 1

这个寄存器在GPDMA模式时用于指定从memory的哪里开始写数据，即目标地址。这个memory的mapping与CPU xdata一样。因为搬运数据时是以32bit为单位，所以这个寄存器有14位。

操作这个寄存器需要写两次，第一次写高6bit，第二次写低8bit的地址；

GPDMACON

Position	7	6	5	4	3	2	1	0
Name	—	—	—	WKGSEL	EC_DONE	AUTO_EC	MODESEL	
Default	0	0	0	0	1	0	0	0
Access	—	—	—	—	R/W	R/W	R/W	R/W

WKGSEL : wake up group select

0: group0 (DP/DM)

1: group1 (NDAT12/NCE7_)

EC_DONE: Error correct done

0: Not done

1: Done

手动模式时，写1不做任何动作，写0时会kick start error correct。自动模式时，写这bit没作用。

AUTO_EC: Auto Error correct

0: manual error correct

1: auto error correct

MODESEL : Mode Select

00: IDLE mode.

01: General Purpose DMA.当完成所有DMA时，会自动变为IDLE MODE.

10: Error correct mode

GPDMACNT

Position	7	6	5	4	3	2	1	0
Name	GPDMACNTL			GPDMACNTH/ GPDMACNTL				
Default	X	X	X	X	X	X	X	X
Access	W	W	W	W	W	W	W	W

GPDMACNT : DMA的数量。以32bit为单位，需要减1，例如写入0表示DMA 1个32bit数据。当完成所有DMA时，MODESEL会自动变为IDLE MODE。共13bit，最多可以一次过搬运8192个32bit数据。

硬件纠错手动模式时，这个寄存器需要写入错误bit数，即BECNT。

Operation Guide

GPDMA

1.指定源地址；例如逻辑地址0x8000为源地址，因为是以32bit为单位，所以需要右移2位:

```
GPDMAPT0 = ((0X8000>>2)>>8);
```

```
GPDMAPT0 = ((0X8000>>2) & 0X00FF);
```

2.指定目标地址；例如逻辑地址0x4000为目标地址，因为是以32bit为单位，所以需要右移2位:

```
GPDMAPT1 = ((0X4000>>2)>>8);
```

```
GPDMAPT1 = ((0X4000>>2) & 0X00FF);
```

3.指定数量，例如16个32位数，需要减1；

```
GPDMACNT = 0x00;
```

```
GPDMACNT = 0x0F;
```

4.Enable GPDMA mode;

```
GPDMACON = 0X01;
```

5.等待MODESEL变为IDLE MODE。

Error Correct Auto Mode

1.必须在BCH进入decode mode之前Enable EC mode，清零ec_done，选择自动模式:

```
GPDMACON = 0X06;
```

2.正常进行BCH decode流程，bch_done有效之后，必须再等待ec_done有效之后再bch_done清0(bch_done被清零的时候同时也会清零ec_done):

```
while((GPDMACON & 0X08) != 0X08);
```

BCTL0 &= ~0X80;

3.继续进行decode 流程，每次等待bch_done的时候都按第二步进行即可。

4.完成所有数据的decode之后，需要退出ec_mode

GPDMACON = 0X00;

Error Correct Manual Mode

1.必须在BCH进入decode mode之前Enable EC mode，选择手动模式：

GPDMACON = 0X0A;

2.正常进行BCH decode流程，第一次bch_done有效之后，存在错误且不大于可纠错范围，则写上错误位置起始地址和错误数量，并kick start error correct

if((BCTL0 & 0X60) == 0X20)

{

GPDMAPT0 = 0XA0 >> 2;

GPDMAPT0 = BXADR2 >> 2;

GPDMACNT = 0X00;

GPDMACNT = BECNT;

GPDMACON &= ~0X08;//kick start

}

BCTL0 &= ~0X80;

3.如果只有一包数据，等待ec_done即可完成decode流程。连续decode请跳过这步，直接看step4

while((GPDMACON & 0X08) != 0X08);

GPDMACON = 0X00;

4.继续进行decode 流程，下一个bch_done有效的时候，如果存在错误（不管是否大于可纠错范围），则必须确认上一包数据已经硬件纠错完成才能clear bch_done和kick start第二次纠错。

if((BCTL0 & 0X20) == 0X20)//have error

{

while((GPDMACON & 0X08) != 0X08);//wait before error correct done

if((BCTL0 & 0X40) != 0X40)//not too many error

{

GPDMAPT0 = 0XA0 >> 2;

GPDMAPT0 = BXADR2 >> 2;

GPDMACNT = 0X00;

GPDMACNT = BECNT;

GPDMACON &= ~0X08;//kick start

}

}

BCTL0 &= ~0X80;

5.循环进行step3，完成所有数据decode后，跳到step3，等硬件完成最后一包数据的纠错即可结束。

6 Interrupt Processing

AX215E extends the interrupt system to support totally 11 interrupt sources with two priority levels. Each interrupt source has one or more associated interrupt-pending flag(s) located in an SFR. When a peripheral or external source meets a valid interrupt condition, the associated interrupt-pending flag is set to logic 1.

If interrupts are enabled for the source, an interrupt request is generated when the interrupt-pending flag is set. As soon as execution of the current instruction is complete, the CPU backs up the location of the next instruction to STACK and then begins execution of an interrupt service routine (ISR). Each ISR must end with an RETI instruction, which returns program execution to the next instruction that would have been executed if the interrupt request had not occurred. If interrupts are not enabled, the interrupt-pending flag is ignored by the hardware and program execution continues as normal. (The interrupt-pending flag is set to logic 1 regardless of the interrupt's enable/disable state.)

Each interrupt source can be individually enabled or disabled through the use of an associated interrupt enable bit in the Interrupt Enable SFRs. However, interrupts must first be globally enabled by setting the EA bit (IE.7) to logic 1 before the individual interrupt enables are recognized.

Setting the EA bit to logic 0 disables all interrupt sources regardless of the individual interrupt-enable settings. Note that interrupts which occur when the EA bit is set to logic 0 will be held in a pending state, and will not be serviced until the EA bit is set back to logic 1.

Some interrupt-pending flags are automatically cleared by the hardware when the CPU vectors to the ISR. However, most are not cleared by the hardware and must be cleared by software before returning from the ISR. If an interrupt-pending flag remains set after the CPU completes the return-from-interrupt (RETI) instruction, a new interrupt request will be generated immediately and the CPU will re-enter the ISR after the completion of the next instruction.

6.1 Interrupt Sources and Vectors

The CPU supports 11 interrupt sources. Software can simulate an interrupt by setting some interrupt-pending flags to '1'. If interrupts are enabled for the flag, an interrupt request will be generated and the CPU will vector to the ISR address associated with the interrupt-pending flag. MCU interrupt sources, associated vector addresses, priority order, and control bits are summarized in Table 6-1. Refer to the data sheet section associated with a particular on-chip peripheral for information regarding valid interrupt conditions for the peripheral and the behaviour of its interrupt-pending flag(s).

6.2 Interrupt Priorities

Each interrupt source can be individually programmed to one of two priority levels: low or high. A low priority interrupt service routine can be pre-empted by a high priority interrupt. A high priority interrupt cannot be pre-empted. Each interrupt has an associated interrupt priority bit in an SFR (IP, IPE) used to configure its priority level. Low priority is the default. If two interrupts are recognized simultaneously, the interrupt with the higher priority is serviced first. If both interrupts have the same priority level, a fixed priority order is used to arbitrate, given in Table 6-1.

Table 6-1: Summary of Interrupts

INT Source	Vector	Priority	Pending Flag	Enable Bit	Priority Bit
HWRST(eMMC)	0053h	1 (Highest)	hw_rst_pnd	IP.7	
SD CMD	0003h	2	SPND.0	IE.0	IP.0

SD RPS	000Bh	3	SPND.1	IE.1	IP.1
SD DI	0013h	4	SPND.2	IE.2	IP.2
SD DO	001Bh	5	SPND.3	IE.3	IP.3
NAND Flash Controller	0023h	6	NTCTL1.7	IE.4	IP.4
BCH-Codec done	002Bh	7	BCTL0.7	IE.5	IP.5
BCH-Codec step1 done	0033h	8	BCTL1.7	IE.6	IE.6
IO wake up	003bh	9	EXINTR.3	IPE.0	IPE.4
Timer	0043h	10	TCTL.7	IPE.1	IPE.5
Serial Port	004Bh	11 (Lowest)	TCTL.3	IPE.2	IPE.6

6.3 Interrupt Latency

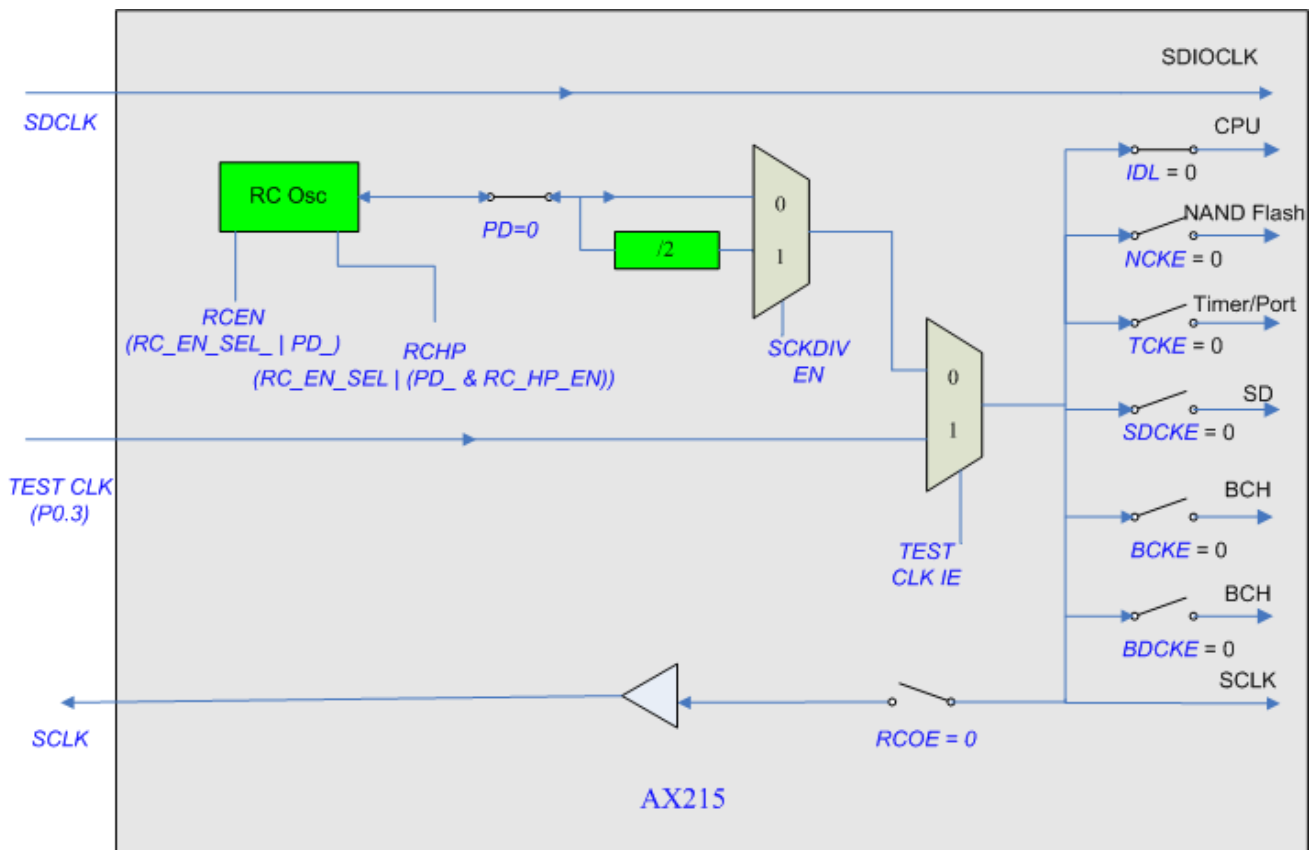
Interrupt response time depends on the state of the CPU when the interrupt occurs. Pending interrupts are sampled and priority decoded each system clock cycle. Therefore, the response time is 4–6 system clock cycles: 1 clock cycle to detect the interrupt, 2 clock cycles to back up the location of next instruction, and 1–3 clock cycles to complete the fetch and to execute the first instruction of ISR, depending on the instruction length. If an interrupt is pending when a RETI is executed, a single instruction is executed before serving the pending interrupt. Therefore, the maximum response time for an interrupt (when no other interrupt is currently being serviced or the new interrupt is of greater priority) occurs when the CPU is performing an RETI instruction followed by a 3-byte instruction as the next instruction. In this case, the response time is 10–12 system clock cycles: 1 clock cycle to detect the interrupt, 3 clock cycles to execute the RETI, 3 clock cycles to fetch and complete the following 3-byte instruction, 2 clock cycles to back up the location of next instruction and 1–3 clock cycles to complete the fetch the first instruction of ISR. If the CPU is executing an ISR for an interrupt with equal or higher priority, the new interrupt will not be serviced until the current ISR completes, including the RETI and following instruction.

7 Clocks and Reset Management

7.1 Clock System

AX215E possesses a configurable clock system aiming to provide balance between performance and power consumption in different applications. It provides fine-grain clock gating to shutdown unused part completely. The organization of clock system (default state) is illustrated in Figure 7-1.

Figure 7-1: Clock System (default state)



7.1.1 Clock Gating Control

There are 6 bits in PCTL to control the clock gating for the 6 main parts in AX215E, they are IDLE for CPU, TCKE for Timer, Serial Port and general IO, SDCKE for SD controller, NCKE for NAND Flash controller, BCKE and BDCKE for BCH-Codec. PD and HPEN is the performance control bit of RC oscillator, and PD is also the output control bit of RC oscillator.

Idle Mode:

Setting IDLE will force the CPU into a idle mode, CPU stops running instruction until any interrupts happen (EA = 1 is must). The interrupt will wake up the CPU and CPU will go to the ISR of this interrupt source.

Power Down Mode:

Power down mode aggressively turn the RC oscillator to low power mode and gate the output (RC oscillator is the global clock of the whole system). At this moment, there is no activity in any module extra RC oscillator.

Only SD interrupt can resume the clocks and deactivate power down mode. There are 2 signals can wake up the system during power down mode, they are SDCMD and SDCLK.

When wake up function is enabled before power down, any falling edge occurred on the selected signal will re-enable the oscillator and then the system is waken up. CPU will go to the ISR if PIE is enable, otherwise CPU goes on running the next instruction before power down. Before power down, it's recommended that all the ports are output high to void the floating leakage. To enable the wake up function, the operation steps listed below are recommended:

```
//select wake up pin and trigger edge
ORL CCTL, #0x40
//enable wake up function
ORL PCTL, #0x10
//insert 4 cycles delay
NOP
NOP
NOP
NOP
//clear wake up pending
ANL PCTL, #0x7F
//insert 4 cycles delay
NOP
NOP
NOP
NOP
//enable port wake up interrupt if needed
ORL IE1, #0x01
ORL IE, #0x81
```

7.2 Relative Registers

Register 7-1: PCTL – Power Control Register

PCTL

Power Control Register

Address	7	6	5	4	3	2	1	0	Default Value
0xA0	POF	SDCKE	BCKE	BDCKE	NCKE	INTBK	PD	IDL	1111_1100
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Bit	Field	Mode	Description	Setting
7	POF	R/W	Power on flag	-
6	SDCKE	R/W	SD clock enable	0: Disable 1: Enable
5	BCKE	R/W	BCH clock enable	0: Disable 1: Enable
4	BDCKE	R/W	BCH decode clock enable	0: Disable 1: Enable
3	NCKE	R/W	Nand flash clock enable	0: Disable 1: Enable
2	INTBK	R/W	Interrupt entry bank address select	0: Low Addr 1: High Addr
1	PD	R/W	Power down mode	0: Disable 1: Enable
0	IDL	R/W	IDLE mode	0: Idle disable 1: Enable

Register 7-2: CCTL – Clock Control Register

CCTL

Clock Control Register

Address	7	6	5	4	3	2	1	0	Default Value
0xA1	SDRST_	NRST_	BRST_	MROM_CE	TSCKIE	HPE_N	TCKE	SCKDIVE	0000_0011
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

SD reset control

0: reset
1: reset release

NAND Flash reset control

0: reset
1: reset release

BCH reset control

0: reset
1: reset release

MROM CE control

0: enable
1: disable

Source clock divide 2

0: no divide
1: divide 2

Timer clock enable

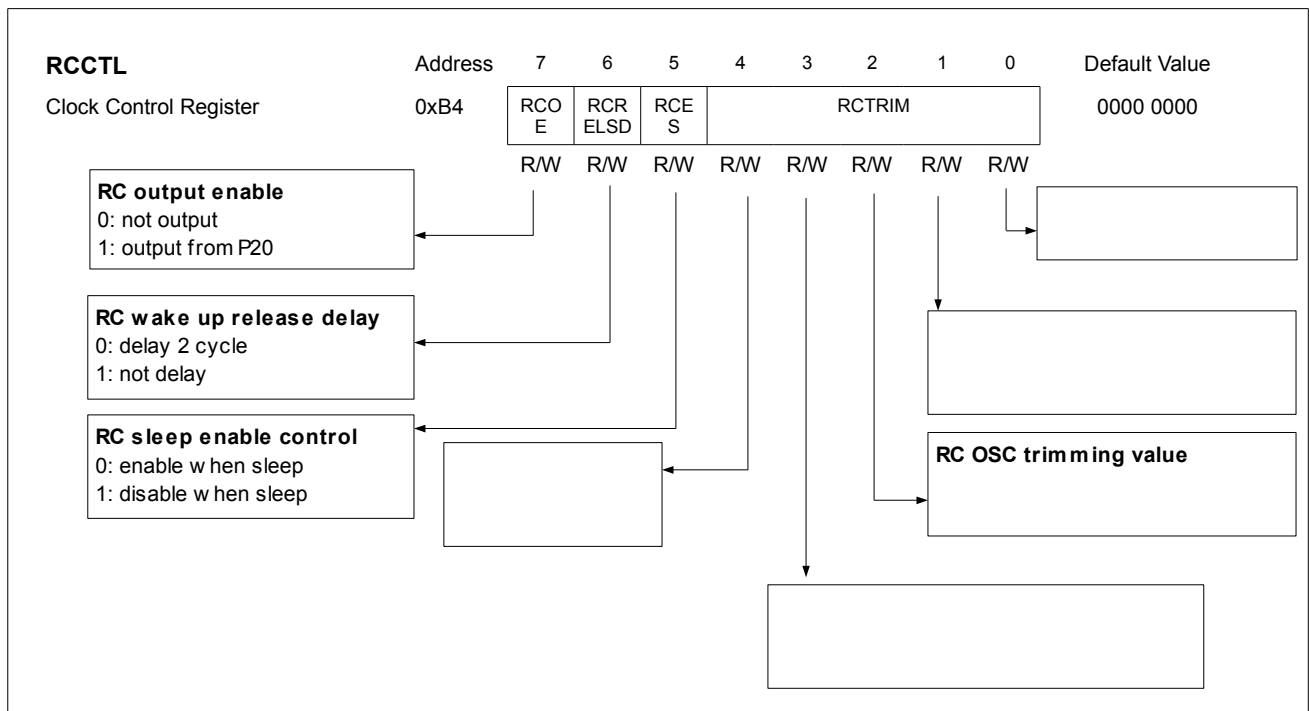
0: disable
1: enable

RC OSC high performance enable

0: RC OSC working on 10MHz
1: RC OSC working on 70MHz

Normal mode Test CLK input select

0: sysclk select RC CLK
1: sysclk select input test CLK

Register 7-3: RCCTL – RC Clock Control Register

Register 7-4: EXINTR – External Interrupt Control Register

EXINTR	Address	7	6	5	4	3	2	1	0	Default Value
External Interrupt Control Register	0xB5	0	boot_pat_en	HWRSTEN	MMC	wk_pnd	wken	wkedg	wkps	0000_0000
		R/W	R/W	R/W	R/W	RO	R/W	R/W	R/W	

boot_pat_en: boot_pattern enable bit

0 : boot pattern disable.

1 : boot_pattern enable.

HWRSTEN: eMMC Host hardware reset enable

0 = disable

1 = enable

MMC: MMC device

0 = SD

1 = MMC

(NOTE: EXINTR 【4】 和SD模块的BOOT_CFG 【7】 的区别，BOOT_CFG 【7】 (MMC_SD只读) 为上电硬件检测MMC/SD信息，若上电后硬件检测到为0，表示为MMCdevice，需要把EXINTR 【4】 一直设为1，否则不需要。)

wk_pnd: wake up pending

wken: wake up enable

0: unable

1:enable

wkedg: wake up pin edge select

0: rise edge

1: fall edge

wkps: wake up pin select

0: SD clk pin

1: SD Command pin

7.3 Reset System

Only POR (power on reset) will reset the whole system of AX215E. It's illustrated in Figure 6-2.

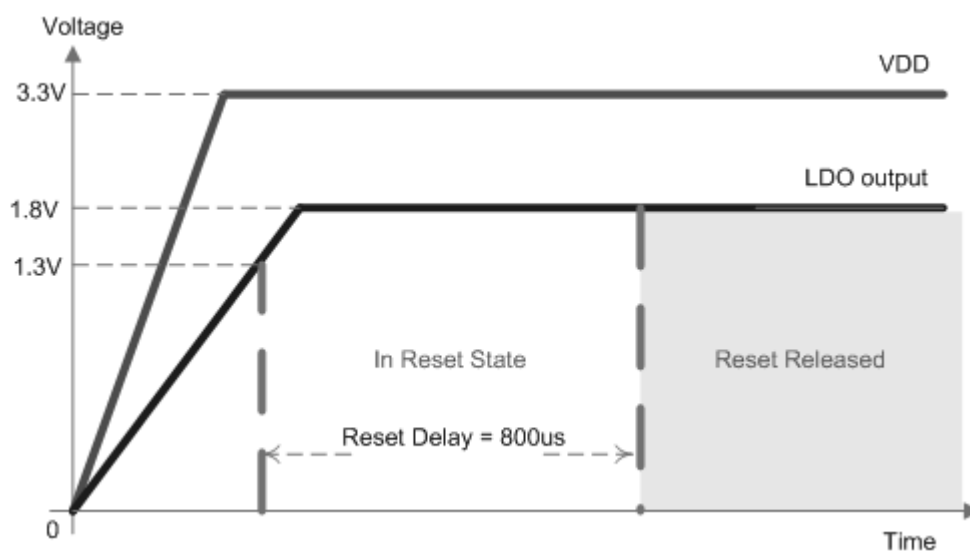


Figure 6-2: Reset Timing

8 Ports

AX215E has 52 IO pins. Figure 7-1 shows the structure of the IO.

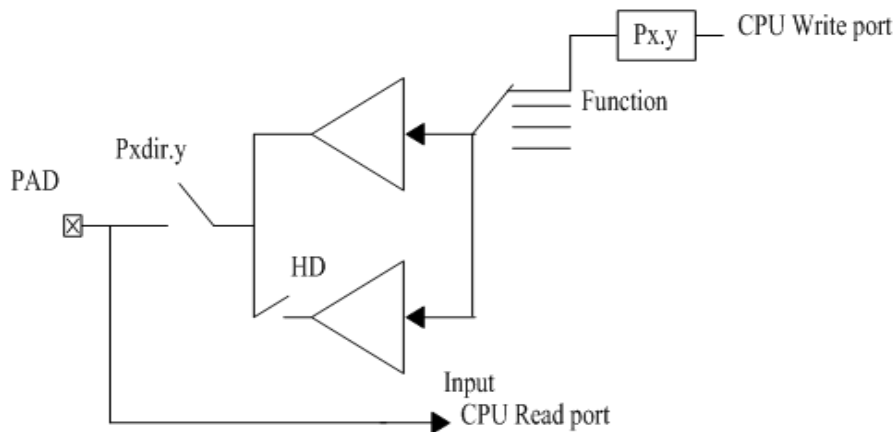


Figure 7-1: IO Structure

From Figure 7-1, user can see that you can set the port current driven strength, CPU will not have the correct value when read the port immediately after port register changed because the delay of synchronizer. Thus in this situation, 3 or more cycles delay should be inserted during write and read ports as the example codes shown below:

```
//configure the port's direction
```

```
//P1.7-4 output
```

```
//P1.3-0 input
```

```
MOV P1DIR, #0x0f
```

```
//write the value for output
```

```
//the value of port output buffer is 0x70
```

```
MOV P1, #0x70
```

```
//change the output
```

```
//the value of P1.3~0 output buffer are unknown because it is changed by ORL, CPU calculates the value read from port pin and then write the result back to port output buffer.
```

```
ORL P1, #0x10
```

```
//inserts 3 or more cycles delay if read is need
```

```
NOP
```

```
NOP
```

```
NOP
```

```
//read port
```


MOV A, P1

Register 8-1: P0PLP – Port 0 Pull-up Register

P0PLP	Address	7	6	5	4	3	2	1	0	Default Value
Port 0 Pull-up Register	0xED	P0PLP								1111 1111
		RW	RW	RW	RW	RW	RW	RW	RW	

Default All Port0 pins are pull-up

Register 8-2: P1PLP – Port 1 Pull-up Register

P1PLP	Address	7	6	5	4	3	2	1	0	Default Value
Port 1 Pull-up Register	0xEE	P1PLP								0000 0000
		RW	RW	RW	RW	RW	RW	RW	RW	

Register 8-3: P2PLP – Port 2 Pull-up Register

P2PLP	Address	7	6	5	4	3	2	1	0	Default Value
Port 2 Pull-up Register	0xEF	P2PLP								0000 0000
		RW	RW	RW	RW	RW	RW	RW	RW	

Register 8-4: P3PLP – Port 3 Pull-up Register

P3PLP	Address	7	6	5	4	3	2	1	0	Default Value
Port 3 Pull-up Register	0xF1	P3PLP								1000 0000
		RW	RW	RW	RW	RW	RW	RW	RW	

Register 8-5: P4PLP – Port 4 Pull-up Register

P4PLP	Address	7	6	5	4	3	2	1	0	Default Value
Port 4 Pull-up Register	0xF2	0	P4PLP							0010 0000
		-	RW	RW	RW	RW	RW	RW	RW	

P4plp[6]: uart output port select

0: uart output from NCE2

1: uart output from SDDAT2

Register 8-6: P0 – Port 0 Register

P0	Address	7	6	5	4	3	2	1	0	Default Value
Port 0 Register	0xF3	P0								xxxx xxxx
		RW	RW	RW	RW	RW	RW	RW	RW	

P0[7]: MMCDATA[7]

P0[6]:MMCDATA[6]

P0[5]:MMCDATA[5]

P0[4]:MMCDATA[4]

Register 8-7: P1 – Port 1 Register

P1	Address	7	6	5	4	3	2	1	0	Default Value
Port 1 Register	0xF4	P1								xxxx xxxx
		RW	RW	RW	RW	RW	RW	RW	RW	

Register 8-8: P2 – Port 2 Register

P2	Address	7	6	5	4	3	2	1	0	Default Value
Port 2 Register	0xF5	P2								xxxx xxxx
		RW	RW	RW	RW	RW	RW	RW	RW	

Register 8-9: P3 – Port 3 Register

P3	Address	7	6	5	4	3	2	1	0	Default Value
Port 3 Register	0xF6	P3								0001 1000
		RW	RW	RW	RW	RW	RW	RW	RW	

Register 8-10: P4 – Port 4 Register

P4	Address	7	6	5	4	3	2	1	0	Default Value
Port 4 Register	0xF7	0	0	P4						00xx xxxx
		RO	RO	RO	RO	RO	RO	RO	RO	

P4[5]: SD_DAT[3]

P4[4]:SD_DAT[2]

P4[3]:SD_DAT[1]

P4[2]:SD_DAT[0]

P4[1]:SD_CMD

P4[0]:SD_CLK

Register 8-11: P0DIR – Port 0 Direction Register

P0DIR	Address	7	6	5	4	3	2	1	0	Default Value
Port 0 Direction Register	0xFC	P0DIR								1111 1111
		-	-	-	-	RW	RW	RW	RW	

Register 8-12: P1DIR – Port 1 Direction Register

P1DIR	Address	7	6	5	4	3	2	1	0	Default Value
Port 1 Direction Register	0xFD	P1DIR								1111 1111
		RW	RW	RW	RW	RW	RW	RW	RW	

Register 8-13: P2DIR – Port 2 Direction Register

P2DIR	Address	7	6	5	4	3	2	1	0	Default Value
Port 2 Direction Register	0xFE	P2DIR								1111 1111
		RW	RW	RW	RW	RW	RW	RW	RW	

Register 8-14: P3DIR – Port 3 Direction Register

P3DIR	Address	7	6	5	4	3	2	1	0	Default Value
Port 3 Direction Register	0xFF	P3DIR								1110 0000
		RW	RW	RW	RW	RW	RW	RW	RW	

Register 8-15: P0HD – Port hard drive control

P0HD	Address	7	6	5	4	3	2	1	0	Default Value
Port 0 hard drive control Register	0xBC	P0HD								0000 0000
		RW	RW	RW	RW	RW	RW	RW	RW	

P0xHD : P0x hard drive control

0: Hard drive disable;

1: Hard drive enable;

Register 8-16: P1HD – Port1 hard drive control

P1HD	Address	7	6	5	4	3	2	1	0	Default Value
Port 1 hard drive control Register	0xBB	P1HD								0000 0000
		RW								

P1xHD : P1x hard drive control

0: Hard drive disable;

1: Hard drive enable;

Register 8-17: P2HD – Port2 hard drive control

P2HD	Address	7	6	5	4	3	2	1	0	Default Value
Port 2 hard drive control Register	0xBA	P2HD								0000 0000
		RW								



P2xHD : P2x hard drive control

0: Hard drive disable;

1: Hard drive enable;

Register 8-18: P3HD – Port3 hard drive control

P3HD	Address	7	6	5	4	3	2	1	0	Default Value
Port 2 hard drive control Register	0xB8	P3HD								0000 0000
		RW								

P3xHD : P3x hard drive control

0: Hard drive disable;

1: Hard drive enable;

Register 8-19: P4HD – Port4 hard drive control

P4HD	Address	7	6	5	4	3	2	1	0	Default Value
Port4 hard drive control Register	--	P4HD								0000 0000
		RW								

P4xHD : P4x hard drive control

0: Hard drive disable;

1: Hard drive enable;

9 Timer and Serial Port

9.1 Timer and Serial Port Register

Register 9-1: TCTL – Timer Control Register

TCTL	Address	7	6	5	4	3	2	1	0	Default Value
Timer Control Register	0xB0	TOF	TPSR			SPKS / SPTD	EX_EN	mode select		0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Timer over flow flag
0: not over/clear
1: over

Timer pre-scaler
000: no pre-scaler
001: 1-bit pre-scaler
010: 2-bit pre-scaler
011: 3-bit pre-scaler
100: 4-bit pre-scaler
101: 5-bit pre-scaler
110: 6-bit pre-scaler
111: 7-bit pre-scaler

Write 1 to it for kicking start Serial Port TX, write 0 to it for clearing the TX done flag. When TX is done, CPU can read 1 from this bit, otherwise 0 is return for the read value
0: not done
1: done/kick start

Mode select:
00: timer mode
01: UART mode
10: SPI mode
11: CRC16I mode

Use external CLK for timer or serial port enable
0: not use
1: use

Register 9-2: TCNT – Timer Counter Register

TCNT	Address	7	6	5	4	3	2	1	0	Default Value
Timer Counter Register	0xB1	TCNT								0000 0000
		RW	RW	RW	RW	RW	RW	RW	RW	

Register 9-3: TPR – Timer Period Register

TPR	Address	7	6	5	4	3	2	1	0	Default Value
Timer Period Register	0xB2	TPR								0000 0111
		RW	RW	RW	RW	RW	RW	RW	RW	

Register 9-4: SBUF – Serial Port RX/TX Buffer Register

SBUF	Address	7	6	5	4	3	2	1	0	Default Value
Serial Port TX Buffer Register	0xB3	SBUF								xxxx xxxx
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

9.2 Timer

The timer is a 8-bit system clock base timer with configurable 7-bit pre-scaler and period register. This timer can be used as a timer or the baud rate generator of serial port. Its block diagram is illustrated in Figure 8-1.

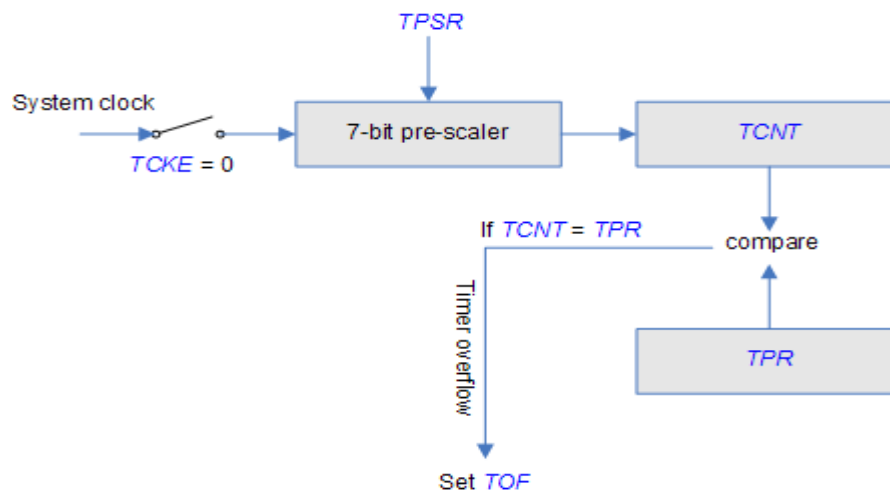


Figure 8-1: Timer Block Diagram

The Timer overflow rate calculation is :

$$\text{Overflow Rate} = \text{TPSR} * (\text{TPR} + 1)$$

To enable the timer, it's recommended to follow the example code list below:

```

//enable the clock to timer
ORL CCTL, #0x02
//select pre-scaler 7-bit mode
MOV TCTL, #0x70
//configure the period register
MOV TPR, #0xFF
//clear the timer counter
MOV TCNT, #0x00
  
```

```
//clear the timer over flow flag
CLR TOF
//check the timer over flow flag
JNB TOF, $
```

9.3 Uart Mode

The serial port can be used to transmit data in UART protocol. It's a timer base serial port. The timing waveform is shown in Figure 8-2.

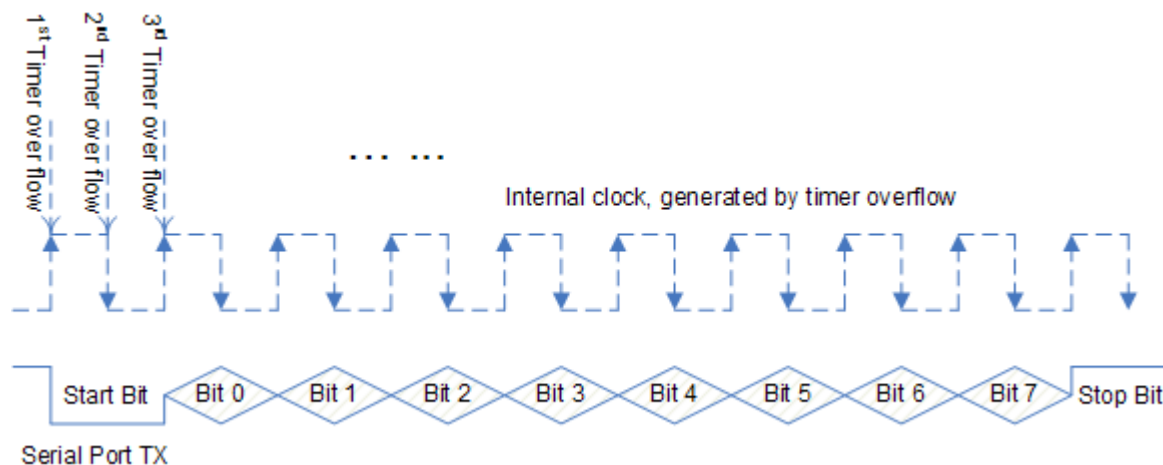


Figure 8-2: Serial Port UART TX Timing

The Serial Port TX/RX is MUXed with pin NCE2 . The the baud rate calculation is:

Baud Rate = $2 * (\text{Overflow Rate}) = 2 * \text{TPSR} * (\text{TPR} + 1)$



Note: UART MODE 有两组io输出，NCE2_ (P02)和SDDAT2 (P44)。通过配置寄存器 P4plp[6]: uart output port select , 0: uart output from NCE2 ; 1: uart output from SDDAT2。

Operation example code listed below:

```
//select UART mode
MOV TCTL, #0x01
//configure the period register
MOV TPR, #0x03
//write the data to be send in serial port buffer
MOV SBUF, #0xAA
//kick start the TX
SETB SPKS
//wait the TX done flag be set
JNB SPTD, $
//clear the TX done flag
```


CLR SPTD

9.4 SPI Mode

The serial port can be used to transmit data in SPI protocol. It's a timer base serial port. The timing waveform is shown in Figure 8-2.

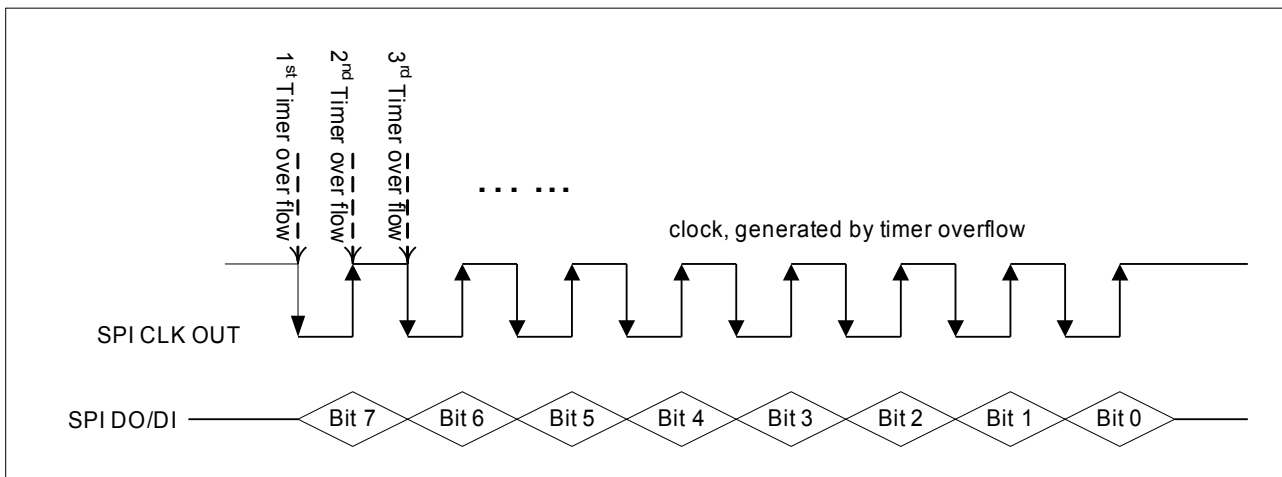


Figure 8-2: Serial Port SPI TX/RX Timing

The SPI TX is MUXed with pin NALE. The SPI RX is MUXed with pin NWP. The SPI CLK is MUXed with pin NCLE. The SPI CLK period calculation is:

$$\text{SPI CLK period} = 2 * (\text{Overflow Rate}) = 2 * \text{TPSR} * (\text{TPR} + 1)$$

Operation example code listed below:

```
//select SPI mode
MOV TCTL, #0x02
//configure the period register
MOV TPR, #0x03
//write the data to be send in serial port buffer
MOV SBUF, #0xAA
//kick start the TX/RX
SETB SPKS
//wait the TX/RX done flag be set
JNB SPTD, $
//read RX data
MOV A, SBUF
//clear the TX/RX done flag
CLR SPTD
```

9.5 CRC16 Mode

Operation example code listed below:

//enable the clock to timer

ORL CCTL, #0x02

//select CRC16 mode

MOV TCTL, #0x03

//clear CRC16 Low buffer

MOV TCNT, #0x00

//clear CRC16 High buffer

MOV TPR, #0x00

//CRC16

MOV SBUF, #0x13

NOP

//read CRC16 Result

MOV A, TCNT

MOV A, TPR

10 NAND Flash Controller

The NAND Flash controller supports Configurable User Define Protocol for NAND Flash accessing.

10.1 NAND Flash Controller Register

Register 10-1: NFIFO0 – NAND Flash FIFO 0 Register

NFIFO0	Address	7	6	5	4	3	2	1	0	Default Value
NAND Flash FLFO 0 Register	0x9B	NFIFO0								0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

NFIFO is the FIFO to store the NAND Flash command0, address data which will be sent. NFIFO0 has no default value, thus it should be initialized before using.

Register 10-2: NFIFO1 – NAND Flash FIFO 1 Register

NFIFO1	Address	7	6	5	4	3	2	1	0	Default Value
NAND Flash FLFO 1 Register	0x9C	NFIFO1								0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

NFIFO is the FIFO to store the NAND Flash command0, address data which will be sent. NFIFO1 has no default value, thus it should be initialized before using.

Register 10-3: NFIFO2 – NAND Flash FIFO 2 Register

NFIFO2	Address	7	6	5	4	3	2	1	0	Default Value
NAND Flash FLFO 2 Register	0x9D	NFIFO2								0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

NFIFO is the FIFO to store the NAND Flash command0, address data which will be sent. NFIFO2 has no default value, thus it should be initialized before using.

Register 10-4: NFIFO3 – NAND Flash FIFO 3 Register

NFIFO3	Address	7	6	5	4	3	2	1	0	Default Value
NAND Flash FLFO 3 Register	0x9E	NFIFO3								0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

NFIFO is the FIFO to store the NAND Flash command0, address data which will be sent. NFIFO3 has no default value, thus it should be initialized before using.

Register 10-5: NCEE – NAND Flash CE Enable Register

NCEE	Address	7	6	5	4	3	2	1	0	Default Value
NAND Flash CE Enable Register	0x91					NCE 3E	NCE 2E	NCE 1E	NCE 0E	xxxx 0000
						R/W	R/W	R/W	R/W	

Enable pin NCE0NCE3 for NAND Flash CE

0: disable
1: enable

Notes:
When NCExE is set, the corresponding CE pin is controlled by NAND Flash controller, it can not be affected by port control registers P0DIR and P0 any more.

Register 10-6: NMCTL – NAND Flash Mode Control Register

NMCTL	Address	7	6	5	4	3	2	1	0	Default Value
NAND Flash Mode Control Register	0x92	LATS EL1	LATS EL0	NRB S1	NRB S0	NCE ACT	reserve	NBU SM1	NBU SM0	0011 1000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

NAND Flash RE data sample select

00: RE rising

01: reserve

10: 1 clk delay after RE rising

11: half clk delay after RE rising

NAND Flash Data bus mode select

00: Low 8-bit bus, NDAT0-7 valid

01: 16-bit bus, NDAT0-15 valid

10: High 8-bit bus, NDAT8-15 valid

11: 8-bit 2 bus, NDAT0-15 valid

Detail in Port mapping

00: NRB0

01: NRB1

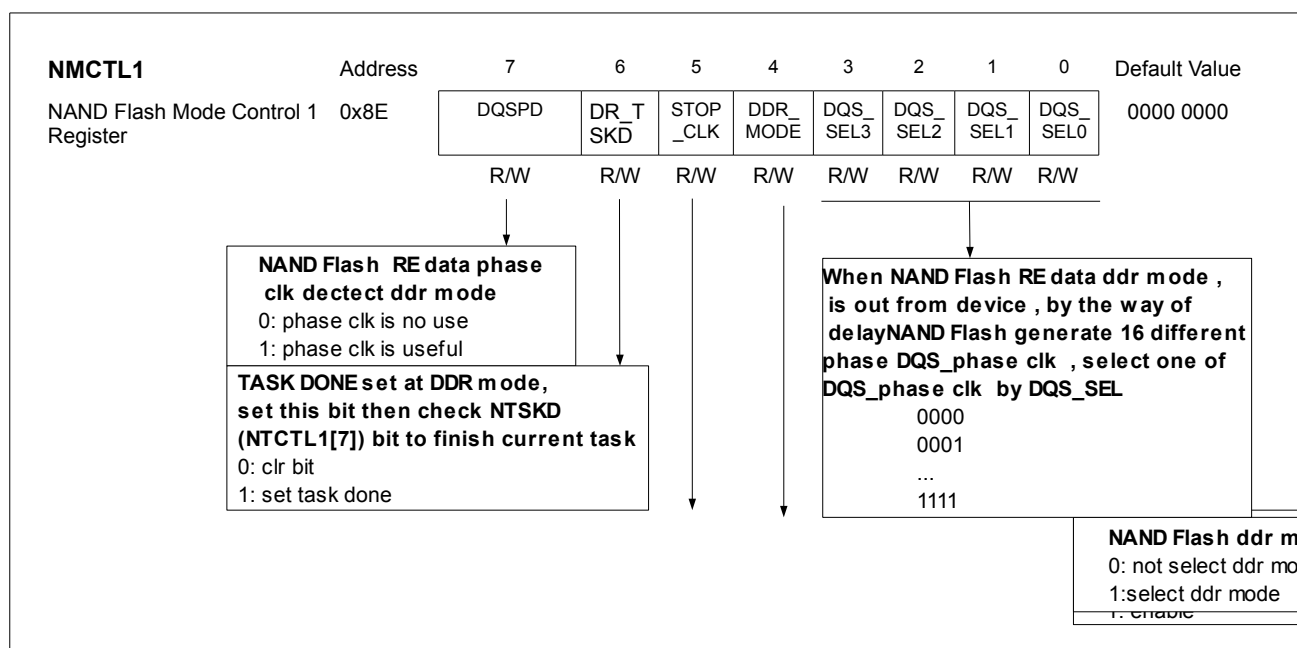
1x: RB don't care

NAND Flash CE active enable

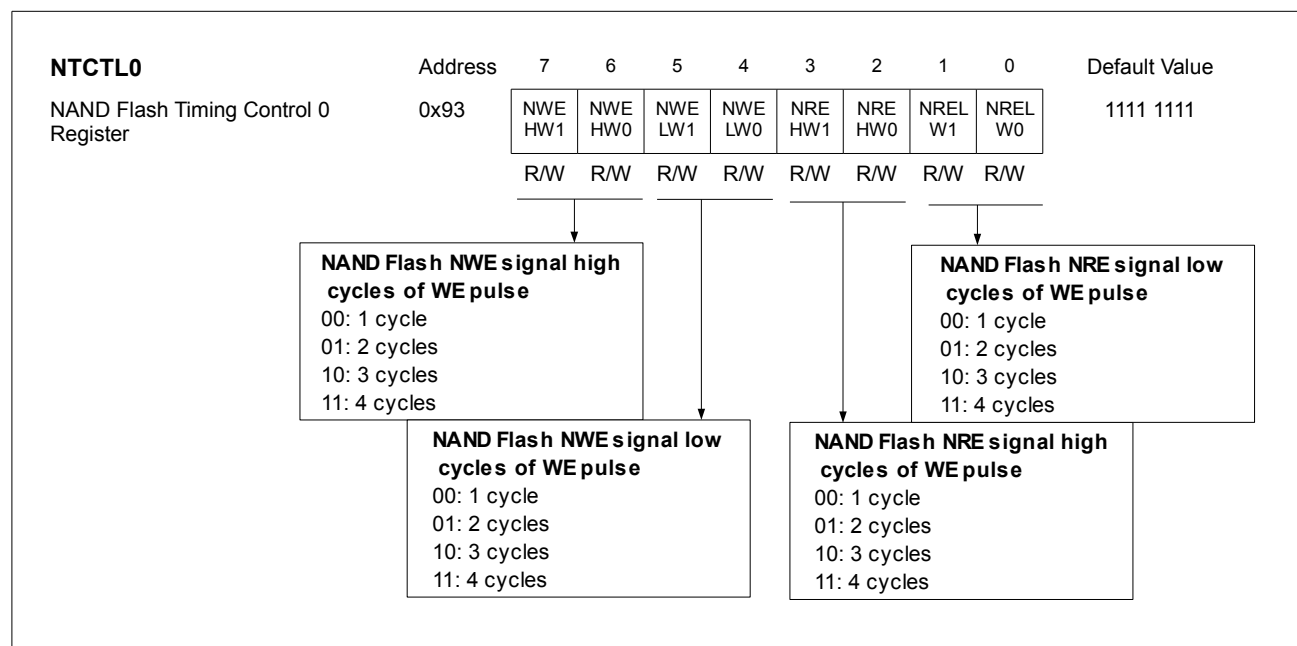
0: disable, CE always active

1: enable, CE active during task only

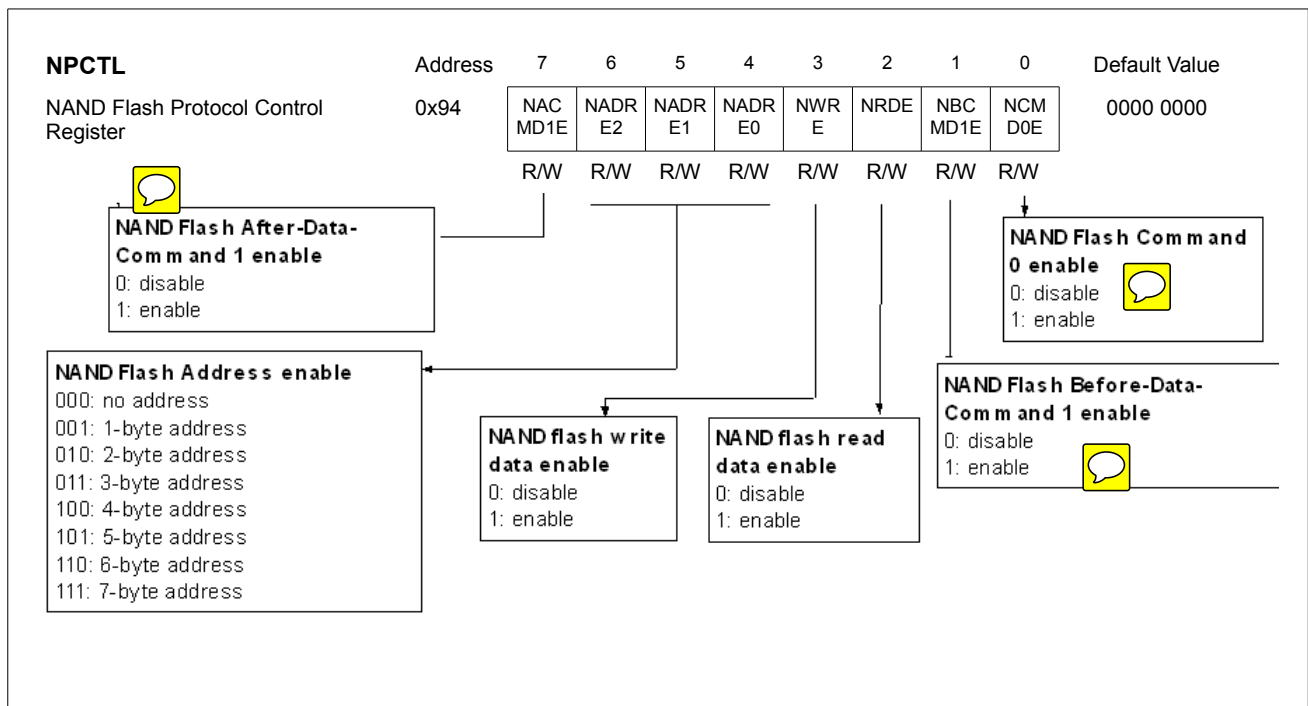
Register 10-7: NMCTL1 – NAND Flash Mode Control 1 Register



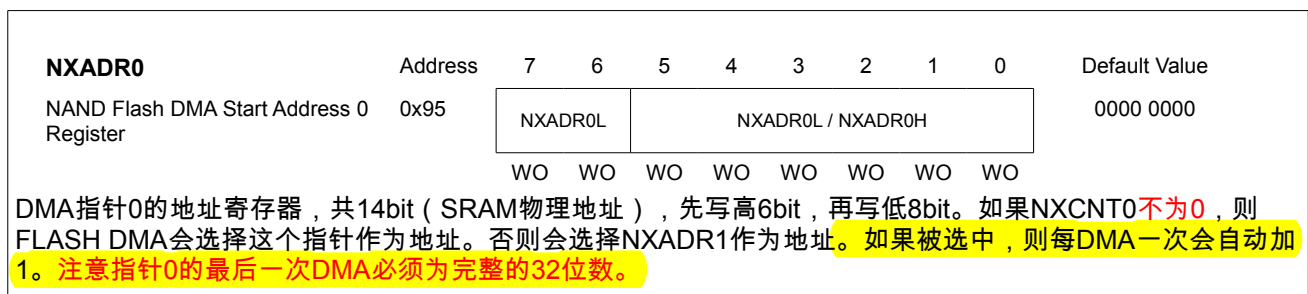
Register 10-8: NTCTL0 – NAND Flash Timing Control 0 Register



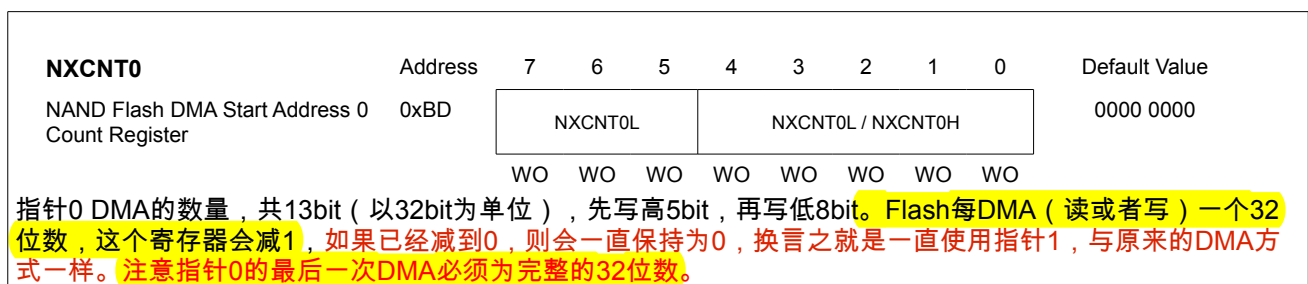
Register 10-9: NPCTL – NAND Flash Protocol Control Register



Register 10-10: NXADR0 – NAND Flash DMA Start Address 0 Register



Register 10-11: NXCNT0 – NAND Flash DMA Start Address 0 Count



Register 10-12: NXADR1 – NAND Flash DMA Start Address 1 Register

NXADR1	Address	7	6	5	4	3	2	1	0	Default Value
NAND Flash DMA Start Address 1 Register	0x96	NXADR1L		NXADR1L / NXADR1H						0000 0000
		-----		WO	WO	WO	WO	WO	WO	

DMA指针1的地址寄存器，共14bit（32bit物理地址），先写高6bit，再写低8bit。如果NXCNT0为0，则FLASH DMA会选择这个指针作为地址。否则会选择NXADR0作为地址。如果被选中，则每DMA一次会自动加1。如果NXCNT0已经减到0，则会一直保持为0，换言之就是一直使用指针1，与原来的DMA方式一样。

Register 10-13: NACMD1 – NAND Flash After-Data-Command 1 Register

NACMD1	Address	7	6	5	4	3	2	1	0	Default Value
NAND Flash After-Data-Command 1 Register	0x97	NACMD1								0000 0000
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Register 10-14: NTCTL1 – NAND Flash Timing Control 1 Register

NTCTL1	Address	7	6	5	4	3	2	1	0	Default Value
NAND Flash Timing Control 1 Register	0x98	NTSK D	NWB DLY2	NWB DLY1	NWB DLY0	NHLD 1	NHLD 0	NSTP 1	NSTP 0	0000 1111
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

NAND Flash Task Done Flag
0: not done
1: done

NAND Flash Wait cycles to detect WB falling
000: 32 cycles
001: 4 cycles
010: 8 cycles
011: 12 cycles
100: 16 cycles
101: 20 cycles
110: 24 cycles
111: 28 cycles

NAND Flash hold time width
00: 0 cycle
01: 1 cycles
10: 2 cycles
11: 3 cycles

NAND Flash setup time width
00: 0 cycle
01: 2 cycles
10: 4 cycles
11: 6 cycles

Register 10-15: NTCTL2 – NAND Flash Timing Control 1 Register 2

NTCTL2

NAND Flash Mode Control Register 2

Address

0xAD

7	6	5	4	3	2	1	0
ddrck div1	ddrc kdiv0	tdqsd 1	tdqsd 0	tcad3	tcad2	tcad1	tcad0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Default Value

0000 0000

DDR clock divide

00: systemclock div 2 10: systemclock div 6
 01: system clock div 4 11: system clock div 8

Nwait for Tdqspre or Tdqs post delay time

00: 8clock 10: 4clock
 01: 2clock 11: 6 clock

Between CMD, ADDRESS, DATA delay in onfi sync mode
CMD to ADDRESS delay for random data input in toggle mode

0000: 32clock
 0001: 2 clock
 0010: 4 clock
 0011: 6 clock
 ...

Note: In normal or DDR toggle mode, counter clocking by system clock
 In ONFI SYNC DDR mode, counter clocking by DDR clock

Register 10-16: NPGSZ0 – NAND Flash Page Size 0 Register**NPGSZ0**

NAND Flash Page Size 0 Register

Address

0x99

7	6	5	4	3	2	1	0
NPGSZ0							
W	W	W	W	W	W	W	W

Default Value

0000 0000

**Register 10-17: NPGSZ1 – NAND Flash Page Size 1 Register****NPGSZ1**

NAND Flash Page Size 1 Register

Address

0x9A

7	6	5	4	3	2	1	0
--	NPGSZ1						
--	W	W	W	W	W	W	W

Default Value

--00 0000

10.2 Default State

To ensure the NAND Flash is in the right state after power up, the IO group of NAND Flash interface have a special state after reset:

Table 11-1: NAND Flash IO Default State

Pin	State	Description
NCE0-3	Input With Pull-up at nomal mode	Do not select any flash
NDAT0-15	Input High Impedance	
NCLE	Output Low	Command disable
NALE	Output Low	Address disable
NRE	Output High	Read disable
NWE	Output High	Write disable
NWP	Output Low	Enable flash write protect mode
NRB0	Input High Impedance	NAND flash RB0 status input
NRB1	Input High Impedance	NAND flash RB1 status input
NDQS	Pin direction is control by ddr_mode	DQS input or output (used for DDR interface)

The NAND Flash is default in write protect state to prevent the error write operation during power up. The example codes of exiting write protect mode are listed below:

//exit the flash write protect mode

ORL P3, #0x01

10.3 User Define Protocol

The controller supports a flexible mode to accessing flash with the command and address organized by user configuration.

Table 11-2: NAND Flash controller mode configuration

Device support Mode	Bus mode	configuration
Support toggle ddr flash device	Low 8_bit mode	DDR_MODE =1 & nmctl[0] = 0
Support ONFI flash device work at sync mode	Low 8_bit mode	DDR_MODE =1 & nmctl[0] = 1
Support tradition flash device (include ONFI flash work at async mode)	Low 8_bit mode	DDR_MODE =0 & nmctl[1:0] = 2'b00
	16bit mode	DDR_MODE =0 & nmctl[1:0] = 2'b01
	High 8_bit mode	DDR_MODE =0 & nmctl[1:0] = 2'b10
	2-8bit mode	DDR_MODE =0 & nmctl[1:0] = 2'b11

Figure 10-1~ 10-6show the command format.

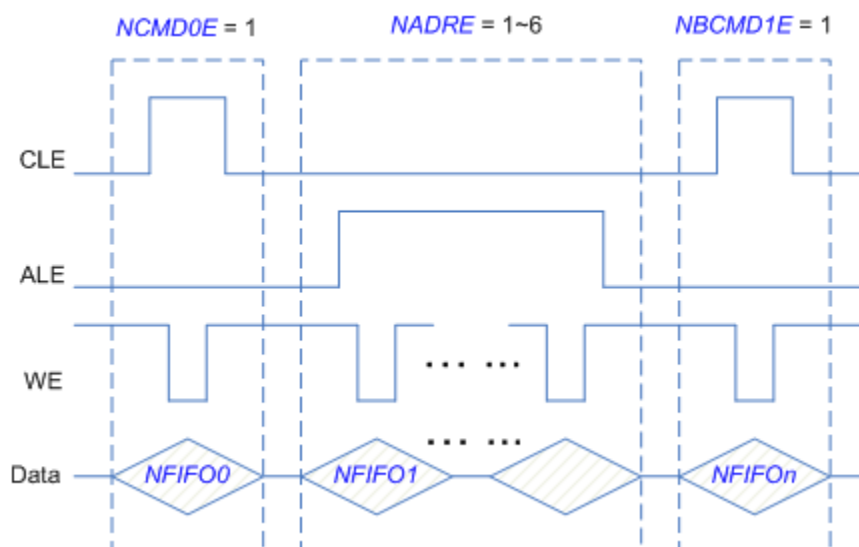


Figure 10-1: User Define Command & Address Part Format

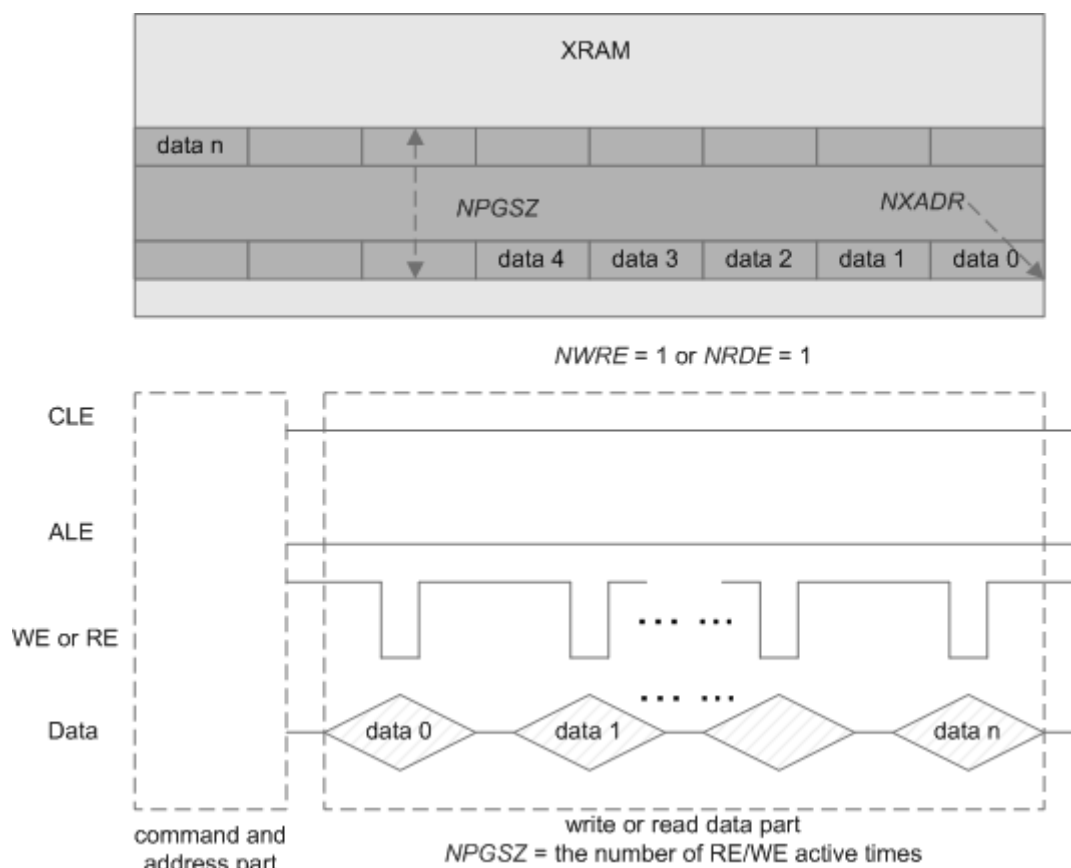


Figure 10-2: User Define Data Part Format

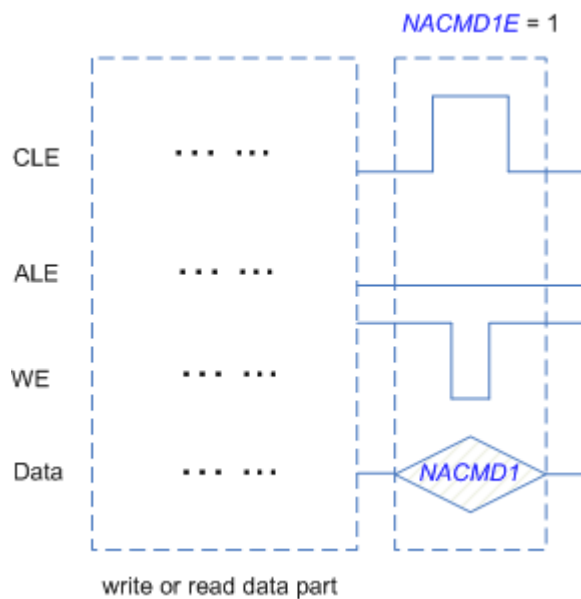


Figure 10-3: User Define Command1 Part Format

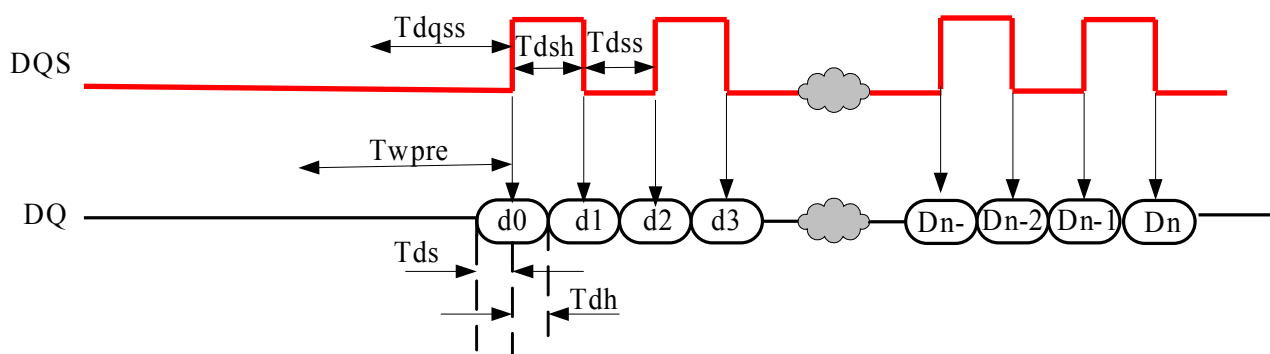


Figure 10-4: User Define write data to device at ddr mode

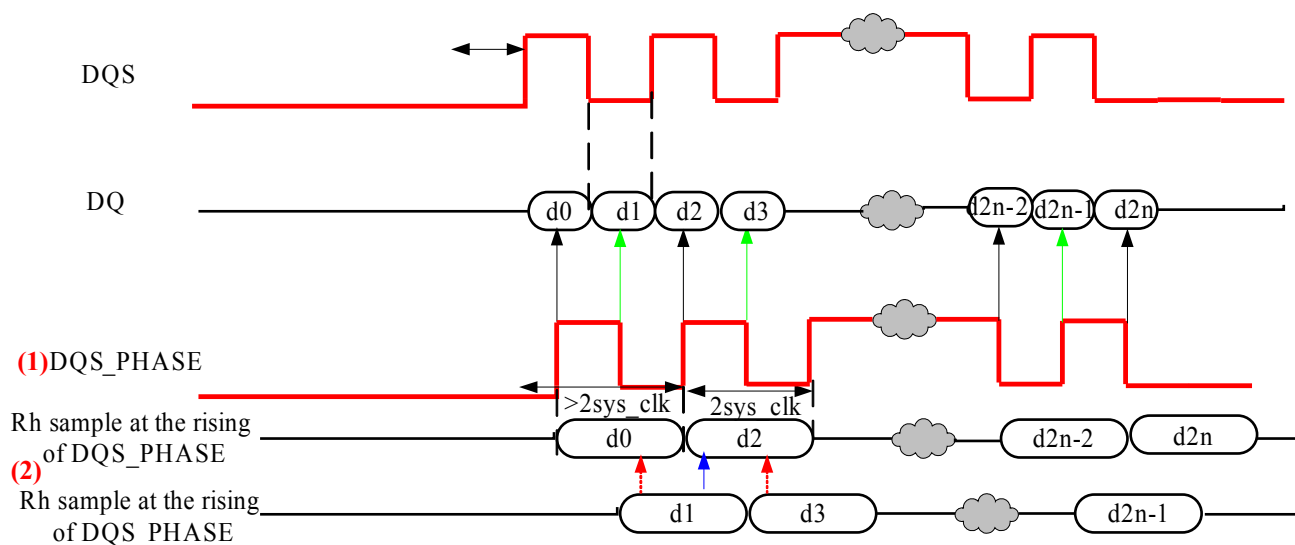


Figure 10-5: User Define read data from device at ddr mode

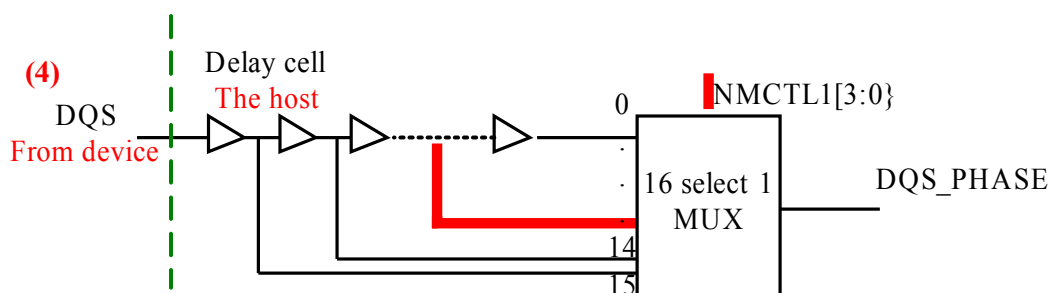


Figure 10-6: User Define DQS phase clk at ddr mode

In the command part, NBCMD1E and NACMD1E should not be enabled both. In the data part, NWRE and NRDE should not be enabled both.

10.4 Cycle Timing

The timing of flash IO pins are controlled by the flash timing control register NTCTL0 and NTCTL1. Figure 10-7 ~ 10-12 illustrate these timing waveform below.

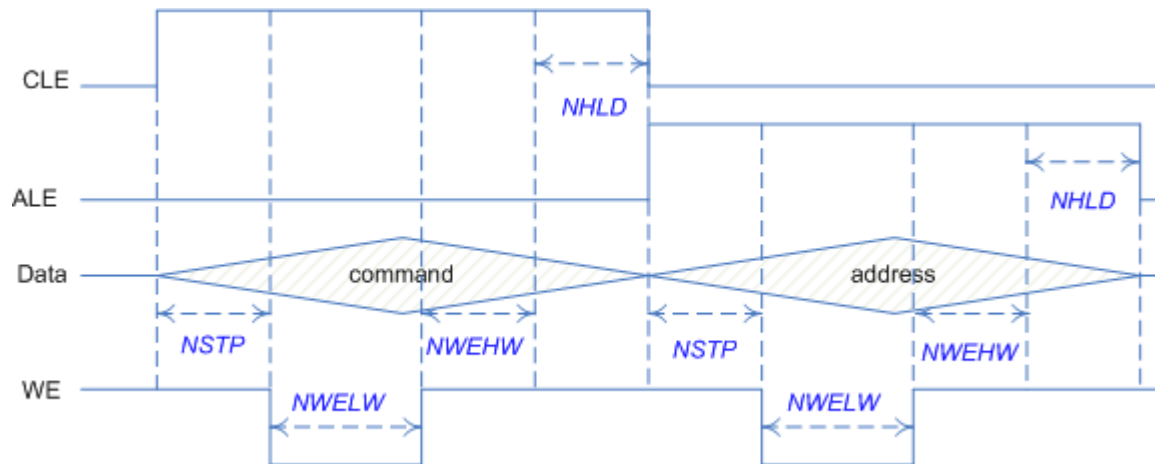


Figure 10-7: CLE / ALE Timing

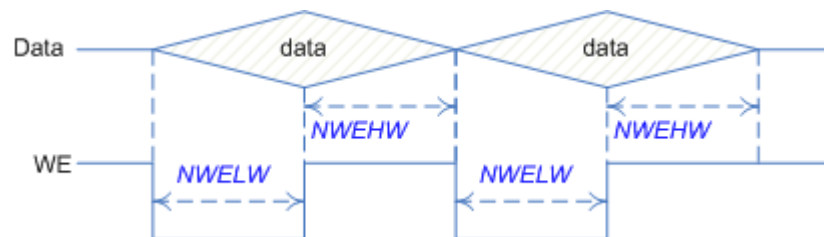


Figure 10-8: Write Data Timing

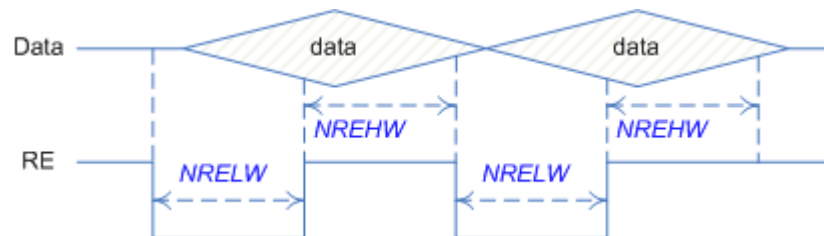


Figure 10-9: Read Data Timing

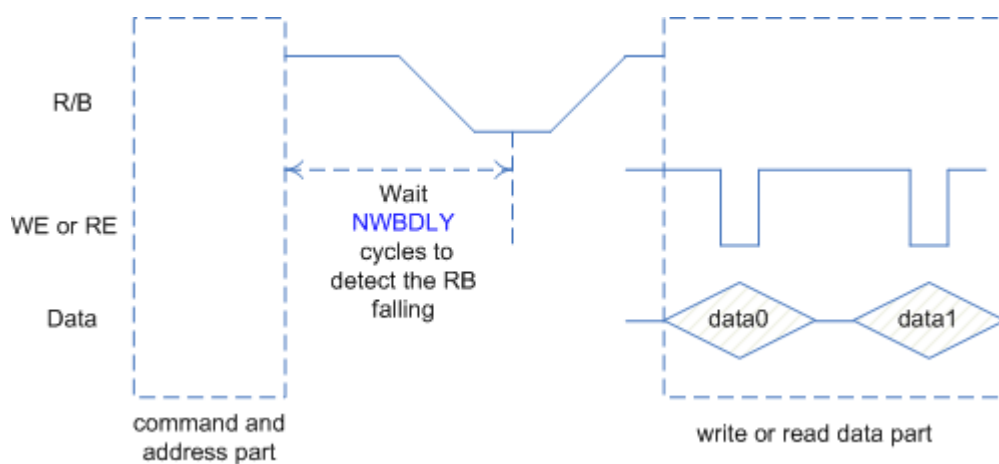


Figure 10-10: Wait R/B Falling Timing 1

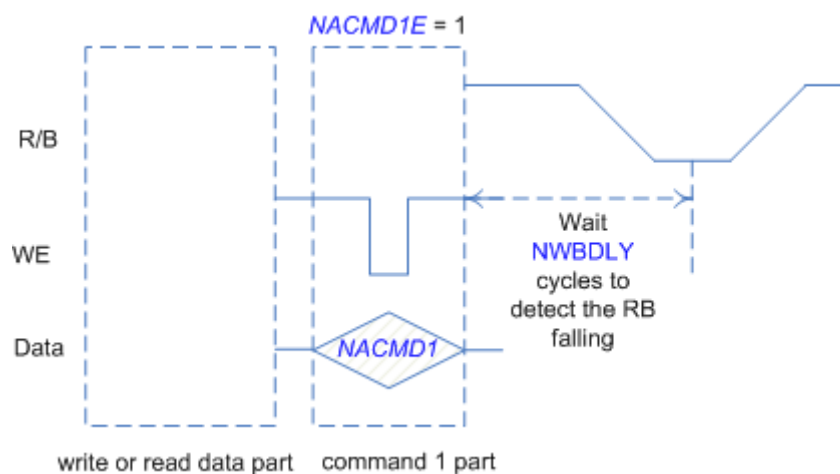


Figure 10-11: Wait R/B Falling Timing 2

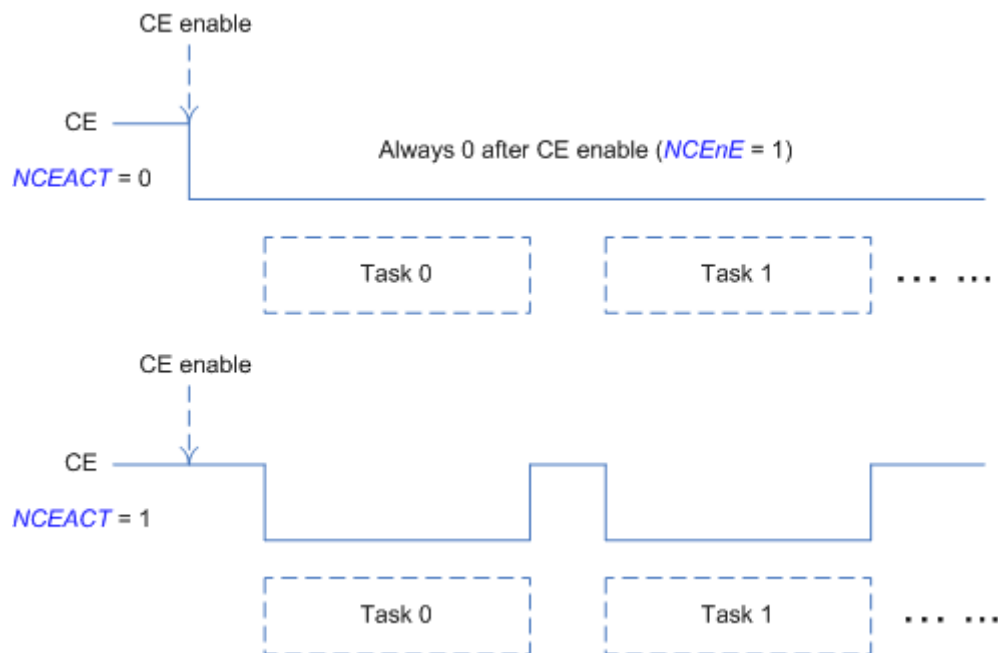


Figure 10-12: CE Timing

figure 9-11 NRE out

figure 9-12 NWE out

figure 9-13 read data sample timing

10.5 Operation Flow

Before operating the NAND Flash controller to access the flash, the IOs connected to flash must be configured:

```
#define NFIFO4 NFIFO0
#define NFIFO5 NFIFO1
#define NFIFO6 NFIFO2
#define NFIFO7 NFIFO3
//initial the IO
MOV P3PLP, #0x20
MOV P3, #0x18
...
...
...
```

The 2nd step is to enable the NAND Flash controller's clock if needed:

```
ORL PCTL, #0x08
```

Then release the NAND Flash controller's reset:

```
ORL CCTL, #0x40
```

Configure the operation mode and cycle timing (please reference to the NAND Flash's data sheet):

```
//select bus mode (low 8-bit )
```

```
//enable CE active mode
```

```
//select RB0
```

```
MOV NMCTL, #0x08
```

```
//select timing
```

```
MOV NTCTL0, #0xFF
```

```
MOV NTCTL1, #0x7F
```

```
//select NCE0 enable
```

```
MOV NCEE, #0x01
```

When all the initialization above are done, the controller is ready for real accessing task. Accessing flash examples:

RESET Task

```
//command 0 = 0xFF(Reset)
```

```
MOV NFIFO0, #0xFF
```

```
//enable command 0 only, kick-start the task
```

```
MOV NPCTL, #0x01
```

```
//wait task done
```

```
JNB NTSKD, $
```

```
//clear task done flag
```

```
CLR NTSK
```

READ STATUS Task

```
//不使用NXADR0，全部通过NXADR1来完成DMA
```

```
//configure DMA start address1 (0x4000)
```

```
// xram_start_addr1 = {0x10, 0x00, 2'b00};
```

```
MOV NXCNT0, #0X00//如果可以确保上次NXCNT0已经为0时，可以省略这两步
```

```
MOV NXCNT0, #0X00
```

```
MOV NXADR1, #0x10
```

```
MOV NXADR1, #0x00
```

```
//configure the number of read byte (0x0010)
```

```
MOV NPGSZ0, #0x10
```

```
MOV NPGSZ1, #0x00
```

```
//command 0 = 0x70(Read Status)
MOV NFIFO0, #0x70
//enable command 0 and read, kick-start the task
MOV NPCTL, #0x05
//wait task done
JNB NTSKD, $
//read the status value
....
//read data from sram
...
...
//clear task done flag
CLR NTSK
```

BLOCK ERASE Task

```
//command 1 = 0xD0 (start erase)
MOV NFIFO4, #0xD0
//address = 0x030201 (3 byte address)
MOV NFIFO3, #0x03 // third address
MOV NFIFO2, #0x02 // second address
MOV NFIFO1, #0x01 // first address
//command 0 = 0x60 (block erase)
MOV NFIFO0, #0x60
//enable command 0, address (3 byte) and before-data-command 1, kick-start the task
MOV NPCTL, #0x33
//wait task done
JNB NTSKD, $
//clear task done flag
CLR NTSK
```

PAGE PROGRAM Task

```
//configure the page size (4K data)
MOV NPGSZ0, #0x00
MOV NPGSZ1, #0x10
//configure the data buffer start address in XRAM, the data should be ready in the buffer
// xram_start_addr0 = {0x04, 0x00, 2'b00};
```

```

// xram_start_addr1 = {0x06, 0x00, 2'b00};
//当addr0 完成0x200次DMA(32bit, 即2K Byte)后会指向 xram_start_addr1完成剩下的DMA。
MOV NXADR0, #0x04
MOV NXADR0, #0x00
MOV NXCNT0, #0X02
MOV NXCNT0, #0X00
MOV NXADR1, #0x06
MOV NXADR1, #0x00
//address = 0x00_02_00_00_00 (5 byte)
MOV NFIFO5, #0x00
MOV NFIFO4, #0x02
MOV NFIFO3, #0x00
MOV NFIFO2, #0x00
MOV NFIFO1, #0x00
//command 0 = 0x80 (page program)
MOV NFIFO0, #0x80
//command 1 = 0x10 (start program)
MOV NACMD1, #0x10
//enable command 0, address (5 byte), write and after-data-command 1, kick-start the task
MOV NPCTL, #0xD9
//wait task done
JNB NTSKD, $
//clear task done flag
CLR NTSK

```

PAGE READ Task

```

//configure the page size (4K data)
MOV NPGSZ0, #0x00
MOV NPGSZ1, #0x10
//configure the data buffer start address in XRAM
// xram_start_addr1 = {0x04, 0x00, 2'b00};
//因为NXCNT0为0, 所以会一直使用 xram_start_addr1完成DMA。
MOV NXCNT0, #0X00
MOV NXCNT0, #0X00
MOV NXADR1, #0x04
MOV NXADR1, #0x00

```

```
//command 1 = 0x30 (start read)
MOV NFIFO6, #0x30
//address = 0x00, 0x02, 0x00, 0x00, 0x00 (5 byte)
MOV NFIFO5, #0x00
MOV NFIFO4, #0x02
MOV NFIFO3, #0x00
MOV NFIFO2, #0x00
MOV NFIFO1, #0x00
//command 0 = 0x00 (page read)
MOV NFIFO0, #0x00
//enable command 0, address (5 byte), read and before-data-command 1, kick-start the task
MOV NPCTL, #0x57
//wait task done
JNB NTSKD, $
//read the data from the data buffer
...
//clear task done flag
CLR NTSK
```

STOP NAND FLASH Task

```
//set task done flag
SETB NTSK
NOP
NOP
//clear task done flag
CLR NTSK
```

WRITE CMD0 and ADDRESS ONLY Task

```
//address = 0x00_02_00_00_00 (5 byte)
MOV NFIFO5, #0x00
MOV NFIFO4, #0x02
MOV NFIFO3, #0x00
MOV NFIFO2, #0x00
MOV NFIFO1, #0x00
//command 0 = 0x80 (page program)
MOV NFIFO0, #0x80
```

//enable command 0, address (5 byte), kick-start the task

MOV NPCTL, #0x51

//wait task done

JNB NTSKD, \$

//clear task done flag

CLR NTSK

WRITE DATA ONLY Task

//enable write data, kick-start the task

MOV NPCTL, #0x08

//wait task done

JNB NTSKD, \$

//clear task done flag

CLR NTSK

WRITE CMD1 ONLY Task

//command 1 = 0x10 (start program)

MOV NACMD1, #0x10

//enable CMD1, kick-start the task

MOV NPCTL, #0x81

//wait task done

JNB NTSKD, \$

//clear task done flag

CLR NTSK

If the NAND Flash controller interrupt is enable (IE.4= 1), an interrupt is generated when task is done (NTSKD = 1).

2. operate follow for device ddr mode (including toggle ddr and ONFI source synchronous mode)

(1)toggle ddr mode

Before operating the NAND Flash controller to access the flash, the IOs connected to flash must be configured:

#define NFIFO4 NFIFO0

#define NFIFO5 NFIFO1

#define NFIFO6 NFIFO2

#define NFIFO7 NFIFO3

//initial the IO

MOV P3PLP, #0x20

ORL P3DIR, #0x20

...

...

...

The 2nd step is to enable the NAND Flash controller's clock if needed:

ORL PCTL, #0x08

Then release the NAND Flash controller's reset:

ORL CCTL, #0x40

Configure the operation mode and cycle timing (please reference to the toggle ddr Flash's data sheet):

//select bus mode (low 8-bit & toggle ddr mode)

//enable CE active mode

//select RB0

MOV NMCTL, #0x08

MOV NMCTL1 , #0X10

//select timing

MOV NTCTL0, #0xFF

MOV NTCTL1, #0x7F

MOV NTCTL2, #0x3f

//select NCE0 enable

MOV NCEE, #0x01

When all the initialization above are done, the controller is ready for real accessing task. Accessing flash examples:

RESET Task

//command 0 = 0xFF(Reset)

MOV NFIFO0, #0xFF

//enable command 0 only, kick-start the task

MOV NPCTL, #0x01

//wait task done

JNB NTSKD, \$

//clear task done flag

CLR NTSK

DQS phase clk select Task

//before read device ,we need select a good DQS phase clk to sample data

MOV R0 , #0x00

```
MOV R1 , #0x10    // DQS_SEL= nmctl[3:0]
```

```
sel_phase_clk:
```

```
    MOV  NMCTL1 , R0          //change the DQS_SEL , select ddr mode
```

```
    ORL   NMCTL1 , #(0x01<<4)
```

```
    LCALL read_task          //a read task ,after task done
```

```
// check DQS phase clk pending bit , when detect nmctl[7] = 0 , DQS phase select done
```

```
    MOV A , NMCTL1
```

```
    ANL A , #0x80
```

```
    JZ  get_phase_clk
```

```
    INC  R0                  // if nmctl[7] = 1 , change DQS_SEL
```

```
    DJNZ R1 , sel_phase_clk
```

```
get_phase_clk:
```

```
    MOV A , R0              //get the useful DQS phase clk number ,when nmctl1[7]=0
```

```
//divide 2 : DQS_SEL=0000,...,r0, detect a good DQS phase clk , so DQS_SEL= r0 /2
```

```
    RR  A
```

```
    ANL A , #~0x80
```

```
    MOV NMCTL1 , A
```

```
    ORL NMCTL1 , #(0x01<<4) //set the right dqs phase clk , select DDR mode
```

```
.....
```

after get the right DQS phase clk ,the page read write task , cmd and adress task are the same as above
tradiction flash's

```
.....
```

(2) ONFI source synchronous mode

Before operating the NAND Flash controller to access the flash, the IOs connected to flash must be configured:

```
#define NFIFO4 NFIFO0
```

```
#define NFIFO5 NFIFO1
```

```
#define NFIFO6 NFIFO2
```

```
#define NFIFO7 NFIFO3
```

```
//initial the IO
```

```
MOV P3PLP, #0x20
```

```
ORL P3DIR, #0x20
```

...
...
...

The 2nd step is to enable the NAND Flash controller's clock if needed:

ORL PCTL, #0x08

Then release the NAND Flash controller's reset:

ORL CCTL, #0x40

RESET Task

//command 0 = 0xFF(Reset)

MOV NFIFO0, #0xFF

//enable command 0 only, kick-start the task

MOV NPCTL, #0x01

//wait task done

JNB NTSKD, \$

//clear task done flag

CLR NTSK

select ONFI synchronous mode

//set ONFI feature , please reference to the ONFI data sheet

MOV NPGSZ0 , #0X04 ;4 parameters

MOV NPGSZ1 , #0X00

;org 0x200 (0x11 ,0x00,0x00,0x00)4 byte data as parameters

MOV NXCNT0, #0X00

MOV NXCNT0, #0X00

MOV NXADR1, #0x02

MOV NXADR1, #0x00

MOV NFIFO0 , #0XEF ; CMD

MOV NFIFO1 , #0X01 ; ADR

MOV NPCTL , #0X19

//wait task done

JNB NTSKD, \$

//clear task done flag

CLR NTSK

//select bus mode (low 8-bit & onfi ddr mode)

```

//enable CE active mode
//select RB0
MOV NMCTL, #0x09
MOV NMCTL1 , #0X10
//select timing
MOV NTCTL0, #0xFF
MOV NTCTL1, #0x7F
MOV NTCTL2, #0x3f
//select NCE0 enable
MOV NCEE, #0x01

```

DQS phase clk select Task

```

//before read device ,we need select a good DQS phase clk to sample data
MOV R0 , #0x00
MOV R1 , #0x10 // DQS_SEL= nmctl[3:0]

sel_phase_clk:
    MOV NMCTL1 , R0 //change the DQS_SEL , select ddr mode
    ORL NMCTL1 , #(0x01<<4)
    LCALL read_task //a read task ,after task done
// check DQS phase clk pending bit , when detect nmctl[7] = 0 , DQS phase select done
    MOV A , NMCTL1
    ANL A , #0x80
    JZ get_phase_clk

    INC R0 // if nmctl[7] = 1 , change DQS_SEL
    DJNZ R1 , sel_phase_clk
get_phase_clk:
    MOV A , R0 //get the useful DQS phase clk number ,when nmctl1[7]=0
//divide 2 : DQS_SEL=0000,...,r0, detect a good DQS phase clk , so DQS_SEL= r0 /2
    RR A
    ANL A , #~0x80
    MOV NMCTL1 , A
    ORL NMCTL1 , #(0x01<<4) //set the right dqs phase clk , select DDR mode

```

.....

after get the right DQS phase clk ,the page read write task , cmd and adress task are the same as above

tradiction flash's

.....

10.6 Packet size setting

NAND FLASH Controller can work on 8-bit or 16-bit data bus mode. It is different setting from 8-bit mode to 16-bit mode.

When read/write n byte work on 8-bit data bus mode, packet size set n.

//configure the number of read/write byte (if n = 0x0110)

MOV NPGSZ0, #0x10

MOV NPGSZ1, #0x01

When read/write n word work on 16-bit data bus mode, packet size set (2*n - 1).

//configure the number of read/write byte (if n = 0x0110, (2*n - 1) = 0x021f)

MOV NPGSZ0, #0x1f

MOV NPGSZ1, #0x02

11 bad column management

BCMADDR



Position	7	6	5	4	3	2	1	0
Name	BCMADDRL				BCMADRH/ BCMADRL			
Default	X	X	X	X	X	X	X	X
Access	W	W	W	W	W	W	W	W

BCMADDR: BCM begin address

这个寄存器在坏列管理模式时用于指定从memory的哪里开始取坏列地址，因为这个寄存器表示的是32X2048的memory的物理地址，又因为BCM用的是16bit的数，所以这个地址为12位。（地址范围为0X000~0XFFF，最低位表示用高16bit（1）还是低16bit（0））

操作这个寄存器需要写两次，第一次写高4bit，第二次写低8bit的地址；

BCMCTL

Position	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—		BCMEN
Default	0	0	0	0	0	0		0
Access	—	—	—	—	—	—		R/W

BCMEN: 坏列管理使能。注意不需要坏列管理时，这bit必须清零。

0: Disable

1: Enable

BCMCNT

Position	7	6	5	4	3	2	1	0
Name	BCMCNT							
Default	X	X	X	X	X	X	X	X
Access	W	W	W	W	W	W	W	W

BCMCNT：该次操作中坏列的数量（需要减1，写0表示有1个坏列）。每出现一个坏列，BCMCNT会减1。如果BCMCNT已经为0，则再出现一个坏列后，BCMEN会disable。

注意：当使能BCMEN时，flash controller中的NPGSZ必须加上这个CNT的值，例如原本NPGSZ是1024个byte，如果这次的坏列数量是4个坏列（即4byte），则NPGSZ的值要改为1028个byte，否则会有4个byte不会写到flash中。

BCMDATA

Position	7	6	5	4	3	2	1	0
Name	BCMDATA							
Default	X	X	X	X	X	X	X	X
Access	W	W	W	W	W	W	W	W

BCMDATA: 将会被写入到坏列去的数据。

Operation Guide

Write bad column address (坏列地址写入)

如图1所示，坏列地址存储在一块32 X 2048的memory中，和XDATA空间中地址为0X8000~0X9FFF的memory为同一块memory。

当需要将坏列地址写进这个memory时，直接通过CPU写入即可。

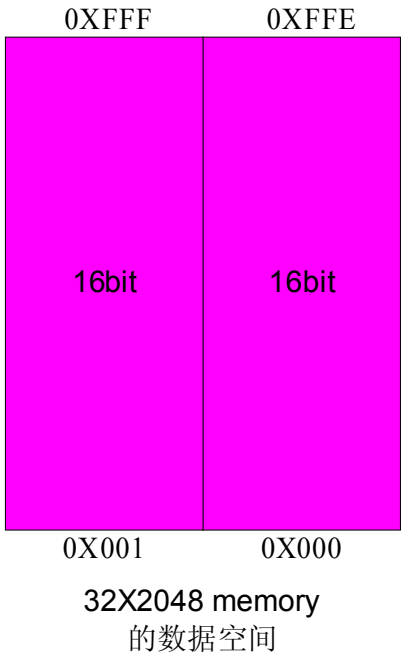


图1：存储坏列地址的memory的数据空间

其中每个16bit的格式如下：

Byte0 BIT	Not use					Bad column address										
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

注意：这个坏列地址指的是该次操作中的第几个byte，例如往flash写数据，NF_CTL会自动通过DMA从SRAM拿数据，如果第三个byte将会被写入坏列中，则这个坏列地址必须写入2，这样原来的第3 byte就会被BCMDATA代替，第3byte会写进在下一个正常的flash空间中。16Bit和8x2 flash也一样，均是以byte为单位来表示坏列地址。

bad column manage (坏列管理)

坏列管理的主要运作方法如下：

(注意BCH的kick start必须放在以下步骤之前)

1.写入坏列地址，具体步骤见上，不需要更新坏列地址时可跳过这步；

2.指定从memory哪里开始读坏列地址；例如0x801为第一个坏列地址(对应的XDATA逻辑地址为 $((0X801 \ll 1) + 0X8000)$):

```
MOV BCMADDR, #0X08;
```

```
MOV BCMADDR, #0X01;
```

3.指定该次操作中坏列的数量，例如有10个坏列（需要减1）：

```
MOV BCMCNT, #0X09;
```

4.指定写入到坏列的数据；

```
MOV BCMDATA, #0XAA;
```

5.Enable bad column manage，选择读或者写flash；例如读flash；

```
MOV BCMCTL, #0X03;
```

6.配置和kick start NFC；

.....

7.等待NFC task done，disable BCM。

.....

```
MOV BCMCTL, #0X00;
```

执行过程为：

每次write flash时，会有一个计数器，每写1byte，计数器加1，然后将计数器的值与当前的坏列地址作比较，两者相等时则判断当前为坏列，这时则将BCMDATA的值替换掉原来的数据，原来的数据则顺延到下一次写。同时，从memory读出下一个坏列地址。

Read flash类似，遇到坏列地址时，会丢弃对应那byte。

完成一次连续的操作，计数器会清零。这时需要重新写BCM begin address和BCMCNT

。

12 BCH-Codec

The BCH-Codec supports 17/25/31/35/44-bit BCH encode/decode algorithms.

17-bit mode has $17 * 14 = 238$ redundant bits = 29 Byte + 6bit.

25-bit mode has $25 * 14 = 350$ redundant bits = 43 Byte + 6bit.

31-bit mode has $31 * 14 = 434$ redundant bits = 54Byte + 2bit.

35-bit mode has $35 * 14 = 490$ redundant bits = 61 Byte + 2bit.

44-bit mode has $44 * 14 = 616$ redundant bits = 77 Byte + 0bit.

It is needed to release BCH reset in CCTL.5, enable bclk and bdclk (for decode mode) in PCTL before using this module.

BCH encode 和decode 的数据长度都是需要设置为8bit 的倍数，且encode 的长度+ 冗余数据长度 = decode 数据长度，当冗余长度不是8的倍数，又满足decode长度为8的倍数时，encode 数据长度不会是8 的倍数，此时硬件会自动根据不同的bit mode 选择，填0以满足要求。

例如选择31 bit mode 时，冗余为 54Byte + 2bit，即encode 数据长度只能是整数byte + 6 bit data，这样 encode 的长度+ 冗余数据长度 = decode 数据长度 才满足8 bit 的倍数，此时硬件会自动在encode 时填入2 bit 0，以满足encode 数据需求

BCH_Codec Register

Register 11-1: BXADR0 – BCH-Codec DMA Start Address 0 Register

BXADR0	Address	7	6	5	4	3	2	1	0	Default Value
BCH-Codec DMA Start Address 0 Register	0X86	BXADR0								0000 0000
		W	W	W	W	W	W	W	W	

Note: BCH data buffer 0 DMA start address (from XRAM). Write 14bit address to this register twice, high 6 bit first, then low 8 bit. Start address is integer and divided 4 of XRAM physical address. Auto increase.

encode: DMA address from XRAM

decode: select data from XRAM: DMA address from SRAM

select data from NFC: no use

Register 11-2: BXADR1 – BCH-Codec DMA Start Address 1 Register

BXADR1	Address	7	6	5	4	3	2	1	0	Default Value
BCH-Codec DMA Start Address 1 Register	0X87	BXADR1								0000 0000
		W	W	W	W	W	W	W	W	

Note: BCH data buffer 1 DMA start address (from XRAM). Write 14bit address to this register twice, high 6 bit first, then low 8 bit. Start address is integer and divided 4 of XRAM physical address. Auto increase.

encode: DMA address from XRAM

decode: select data from XRAM: DMA address from SRAM

select data from NFC: no use



Register 11-3: BXADR2 – BCH-Codec DMA Start Address 2 Register

BXADR2	Address	7	6	5	4	3	2	1	0	Default Value
BCH-Codec DMA Start Address 1 Register	0X8B	BXADR1								0000 0000
		W	W	W	W	W	W	W	W	

For encode mode:

Write register only:

BCH redundant data DMA address to XRAM. Write 14bit address to this register twice, high 6 bit first, then low 8 bit. Start address is integer and divided 4 of XRAM physical address. Auto increase.

Encode 冗余数据处理：

1，直接写入flash mode 时，冗余数据直接送到flash controller，不保存

2，非写入flash mode 时，冗余数据保存到BXADR2 指向地址，可以通过CPU读取这些数据

For decode mode：

Read register only:

Indicate BCH error position data start address (base address is A000H): A000 + BXADR2;

decode 错误数据位置可以通过CPU读取，或使用硬件自动纠错模块处理数据

Register 11-2: BPSZ – BCH-Codec DMA DATA packet size Register

BPSZ	Address	7	6	5	4	3	2	1	0	Default Value
BCH-Codec data packet size	0X8D	BPSZ								0000 0000
		W	W	W	W	W	W	W	W	

Note: BCH encode or decode data byte length packet size. Write 12bit counter to this register twice,high 4 bit first, then low 8 bit.

Encode mode: encode data 长度，以byte 为单位，剩余几bit不足1byte 的，硬件自动根据不同的bit mode 填入相应的0来进行编码，所以要按足1 byte计算

Decode mode: decode data 长度，一定是整数byte data

Register 11-2: BCNT0 – BCH-Codec DMA DATA buffer 0 counter Register

BF0CNT	Address	7	6	5	4	3	2	1	0	Default Value
BCH-Codec data buffer0 CNT	0X8C	BF0CNT								0000 0000
		W	W	W	W	W	W	W	W	

Note: BCH encode or decode data buffer0 byte counter. Write 10bit counter to this register twice,high 2 bit first, then low 8 bit.

设置要求： $\text{BFOCNT} \times 4 \leq \text{BPSZ}$ ，BFOCNT为0 或需要的byte数除以4

当选择使用一个data buffer 时， $\text{BFOCNT} = 0$ ；

当选择使用两个data buffer 时，BFOCNT 为data buffer 0 的数据长度，且数值是需要的byte数除以4。

Register 11-3: BCTL0 - BCH-Codec Control 0 Register

BCTL0	Address	7	6	5	4	3	2	1	0	Default Value
BCH-Codec Control 0 Register	0X88	BDONE	BFAIL	BHERR	RDTDMAEN	rev	BTSEL			0000 0000
		R/W	R	R	R/W	R/W	R/W			

Bit	Field	Mode	Description	Setting
7	BDONE	R/W	BCH done flag	Read: 0: bch not done 1: bch done Write: 0: clear bit 1: no effect
6	BFAIL	RO	BCH fail	0: data correct-able 1: data correct fail
5	BHERR	RO	BCH has error	0: BCH detect no error 1: BCH detect error
4	RDTDMAEN	R/W	BCH redundant decode DMA enable	0: BCH redundant bit not DMA to SRAM 1: BCH redundant bit DMA to SRAM
3	rev	R/W		
2:0	BTSEL	R/W	BCH correct bit select	000: 17 bit 001: 25 bit 010: 31 bit 011: 35 bit 1xx: 44 bit

Note:

Register 11-4: BCTL1 – BCH-Codec Control 1 Register

BCTL1	Address	7	6	5	4	3	2	1	0	Default Value
BCH-Codec Control 1 Register	0X89	BSDONE	BDMAMUX	BDCSEL	BMODE	BEPADRTG	-	-	BKS	0000 0000
		R/W	R/W	R/W	R/W	R/W	-	-	WO	

Bit	Field	Mode	Description	Setting
7	BSDONE	R/W	BCH syndrome done	Read: 0: bch syndrome not done 1: bch syndrome done Write: 0: clear bit 1: no effect
6	BDMAMUX	R/W	BCH data DMA select	Encode mode : 0: data from SRAM, redundant data save to SRAM 1: data from SRAM, encode data and redundant data save to NFC decode mode 0: data from SRAM, error position save to SRAM 1: data from NFC, error position save to SRAM
5	BDCSEL	R/W	BCH decode length select	0: 8864 1: 4768
4	BMODE	R/W	BCH mode	0: encode mode 1: decode mode
3	BEPADRTG	R/W	BCH error position buffer auto toggle	0: disable toggle, always use A000H buffer 1: enable toggle, auto loop from A000H, A0B0H; can be read low 8bit by bxadr2 in decode mode
2-1	-	-	-	-
0	BKS	WO	BCH kick start	0: no effect 1: kick start

Register 11-7: BECNT – BCH-Codec Error Counter Register

BECNT	Address	7	6	5	4	3	2	1	0	Default Value
BCH-Codec Error Counter Register	0X8A	BECNT								xxxx xxxx
		R	R	R	R	R	R	R	R	

note: BECNT is validly respond to BFAIL = 0 and BHERR = 1.

Encode

To encode one package, it is needed to set as follows:

Encode and save redundant data to SRAM:

- 1, set DMA data buffer start address, redundant data DMA start address,buffer size
- 2, set encode mode, data save to SRAM and select T(bit correction ability), initial packet size and data buffer0 counter(if using one data buffer, BF0CNT = 0), (BPSZ = encode byte)(eg. BPSZ = 1024 + 7, BF0CNT = 1024 (using two data buffer))
- 3, kick start encode
- 4, check done BCTL1.7
- 5, clear done

Example for encode:

```
// parameter define
#define ECBCHPSZ (1024 + 7)
#define BCHBF0CNT 1024
#define BCHBFXADR0 0x1000
#define BCHBFXADR1 0x5000
#define BCHBFXADR2 0x6000
#define TSEL 4
#define BL1K 0
// configure packet size, data buffer start address, buffer 0 counter
MOV BPSZ, #high (ECBCHPSZ)
MOV BPSZ, #low (ECBCHPSZ)
MOV BF0CNT, #high (BCHBF0CNT>>2)
MOV BF0CNT, #low (BCHBF0CNT>>2)
MOV BXADR0, #high (BCHBFXADR0>>2)
MOV BXADR0, #low (BCHBFXADR0>>2)
MOV BXADR1, #high (BCHBFXADR1>>2)
MOV BXADR1, #low (BCHBFXADR1>>2)
// redundant data write to SRAM start address
MOV BXADR2, #high (BCHBFXADR2>>2)
MOV BXADR2, #low (BCHBFXADR2>>2)
// select 44bit mode
MOV BCTL0, #(TSEL<<0)
// select encode mode, 1K byte sector decode length, redundant data save to SRAM
MOV BCTL1, #((0x00<<6) | (BL1K<<5) | (0x00<<4))

// kick start
ORL BCTL1, #(0x01<<0)
// wait encode finish
WAITD:
MOV A, BCTL1
JNB ACC.7, WAITD
//clear done
ANL BCTL1, #~(0x01<<7)
CLR BCTL0.7
```

Encode and save redundant data to NFC:

- 1, set DMA data buffer start address, redundant data DMA start address, buffer size

- 2, set encode mode, data save to NFC and select T(bit correction ability), initial packet size and data buffer0 counter(if using one data buffer, BF0CNT = 0), (BPSZ = encode byte)(eg. BPSZ = 1024 + 7, BF0CNT = 1024 (using two data buffer))
- 3, set NFC controller
- 4, kick start encode
- 5, check done BCTL1.7
- 6, clear done

Decode and correct data

Decode:

Decode one data packet from SRAM:

- 1, set DMA data start address, buffer size
- 2, set decode mode, data from SRAM and select T(bit correction ability), initial packet size and data buffer0 counter(if using one data buffer, BF0CNT = 0), (BPSZ = decode byte)(eg. BPSZ = 1024 + 7+77, BF0CNT = 1024 (using two data buffer), T = 44 bit mode)
- 3, kick start decode
- 4, check bchdone, and if there have errors, correct and clear bch syndrome done then bch done

Example for decode:

```
// configure packet size, data buffer start address, buffer 0 counter
MOV BPSZ,    #high (ECBCHPSZ)
MOV BPSZ,    #low (ECBCHPSZ)
MOV BF0CNT,  #high (BCHBF0CNT>>2)
MOV BF0CNT,  #low (BCHBF0CNT>>2)
MOV BXADR0,  #high (BCHBFXADR0>>2)
MOV BXADR0,  #low (BCHBFXADR0>>2)
MOV BXADR1,  #high (BCHBFXADR1>>2)
MOV BXADR1,  #low (BCHBFXADR1>>2)

// select 44bit mode
MOV BCTL0, #(TSEL<<0)
// select decode mode, 1K byte sector decode length, redundant data save to SRAM
MOV BCTL1, #((0x00<<6) | (BL1K<<5) | (0x01<<4))

// kick start
ORL BCTL1, #(0x01<<0)
// wait decode stage1 finish
WAITSD:
    MOV A, BCTL1
    JNB ACC.7, WAITSD
// wait decode finish
WAITD:
    JNB BCTL0.7, WAITD
// check error flag
CALL CHECKERR
//clear stage 1 done
ANL BCTL1, #~(0x01<<7)
//clear BCH done
CLR BCTL0.7
```

Decode one data packet from NFC:

- 1, set DMA data start address, buffer size
- 2, set decode mode, data from NFC and select T(bit correction ability), initial packet size and data buffer0 counter(if using one data buffer, BF0CNT = 0), (BPSZ = decode byte)(eg. BPSZ = 1024 + 7+77, BF0CNT = 1024 (using two data buffer), T = 44 bit mode)
- 3, kick start decode
- 4, check bchdone, and if there have errors, correct and clear bch syndrome done then bch done

Decode several data packets:

- 1, set DMA data start address, buffer size
- 2, set decode mode, data from SRAM and select T(bit correction ability), initial packet size and data buffer0 counter(if using one data buffer, BF0CNT = 0), (BPSZ = decode byte)(eg. BPSZ = 1024 + 7*77, BF0CNT = 1024 (using two data buffer),T = 44 bit mode)
- 3, kick start 1st decode
- 4, wait bch 1st syndrome done, if done, reconfigure DMA address ect. Then kick start 2nd decode. Repeat step 4 one more time (wait bch 2nd syndrome done,and kick start 3rd decode)
- 5, check bchdone (first time for 1st data packet), and if there have errors, correct and clear bch done
- 6, wait bch syndrome done, if done(first time for 3rd data packet syndrome done), reconfigure next DMA address ect. Then kick start next decode again.
- 7, repeat step 5 and step 6, if decode data packets no finish.
- 8, repeat step 5 two more times if all data had DMA input to BCH (wait last 2 data packet decode done)
- 9, decode finish

note:

During series decode several data packets , can not change BCTL0, BCTL1, BPSZ AND BF0CNT SFR configure.

Check error flag and correct data:

Before correct data, check BHERR bit first,

If 0, no error occur, return

If 1, BCH detect error

Then check BFAIL bit,

If 1, Too many errors occur, BCH can not correct those errors, return

If 0, BCH can correct those errors

Correct data:

- 1, using CPU to correct data

- 1,read error position start address from BCH controller

- 2,read error position from SRAM

- 3, read error data from SRAM

- 4,correct data

Eg:

```
data_reg[error_position]=data_reg[error_position]^error_data;
```

```
// check error flag
```

```
CHECKERR:
```

```
JNB BCTL0.5, NO_ERR_DONE
```

```
JB BCTL0.6, TOO_M_ERR
```

```
CALL BCH_XOR_ERR
```

```
TOO_M_ERR:
```

```
NO_ERR_DONE: // no error or too many error
```

```
RET
```

```
// correct error data
```

```
BCH_XOR_ERR:
```

```
// get error bit count
```

```
MOV R7, BECNT
```

```
// get error position start address in SRAM
```

```
MOV DPH, #0XA0
```

```
MOV DPL, BXADR2
```

```
BCH_XOR_BIT:
```

```
// DPTR auto inc
```

```
SETB DPAID
```

```
// get error data byte address
```

```
MOVX A, @DPTR
```

```
MOV DP1L, A
```

```
MOVX A, @DPTR
```

```
MOV DP1H, A
```

```
// get error data
```

```

MOVX A, @DPTR
MOV  R8, A
INC  DPTR // NOP INC
// change DPTR0 to DPTR1, disable DPTR auto inc
SETB DPSEL
CLR  DPAID
// get & correct data
MOVX A, @DPTR
XRL  A, R8
MOVX @DPTR, A
CLR  DPSEL
BCH_NEXT_ERR:
  DJNZ R7, BCH_XOR_BIT
RET

```


- 2 , using hardware to correct data
 - 1 , set mem_ctl to BCH correct data mode
 - 2 , kick start correct data
 - 3 , check correction done

13 SD Controller

13.1 SDC Register

1.SPND:SD pending flag register (Addr: 0xE8)

Position	7	6	5	4	3	2	1	0
Name	SPI_Mode	HW_RST_PND	Align	Dato_sts	Dat_Tx_Pend	Dat_Rx_Pend	Rps_Tx_Pend	Cmd_Rx_Pend
Default	x	0	0	0	0	0	0	0
Access	RO	RO	R/W	RO	R/W	R/W	R/W	R/W

SPI_Mode: SPI Mode 

0: not spi mode,maybe SD mode or MMC mode

1: spi mode

HW_RST_PND : HW reset pending

0 : not pending

1: HW reset pending

Align: If DMA write data bytes can't divide by 4,you can set this bit to align the data before writing into flash.

0 = you need align the data by software

1 = can align the data auto

Dato_sts: Stop data TX

0 = Not Stop

1 = Stop Data TX Operation

Dat_Tx_Pend: Data TX Pending

0 = Not Pending

1 = Data TX Pending

Dat_Rx_Pend: Data RX Pending

0 = Not Pending

1 = Data RX Pending

Rps_Tx_Pend: Response TX Pending Or Response Fail

0 = Not Pending

1 = Rspnse TX Pending

Cmd_Rx_Pend: Command RX Pending

0 = Not Pending

1 = Command RX Pending

2.SCCTL: SD Command Control Register (Addr:0xE3)

Position	7	6	5	4	3	2	1	0
Name	Cmd_CRC_	SPI_CRC_C	Out_Rise	Cmd_CRC_e	Cmd8_rcv	DUAL_VOL	Manual	Get_Nxt

	Chk_Bit	hk		rr			_Rps	_Cmd
Default	0	0	0	0	0	1	0	0
Access	R/W	RO	R/W	RO	RO	-R/W	R/W	W0

Cmd_CRC_Chk_Bit: Command CRC Check enable

0 = Command CRC Check enable

1 = Command CRC Check disable

SPI_CRC_Chk: SPI CRC option ON/OFF(SPI CMD59)

0 = OFF

1 = ON

Out_Rise: Data out at Posedge or Negedge

0 = Posedge

1 = Negedge

Cmd_CRC_err: Command CRC Error

0 = Cmd RX Correctly

1 = Cmd RX Uncorrectly

Cmd8_rcv: Cmd8 auto response or manual response

0 = Cmd8 response by software

1 = Cmd8 auto response by hardware

DUAL_VOL: whether emmc device support dual voltage

0: EMMC not support dual voltage

1: EMMC support dual voltage

Manual_Rps: Manual Response Mode enable

0 = Auto Response By Hardware

1 = Response By Software

Get_Nxt_Cmd: Get Next Command

0 = Not Get Next Command

1 = Get Next Command 

3.SRCTL: SD Response Control Register (Addr: 0xE7)

Position	7	6	5	4	3	2	1	0
Name	Rps_St_Byte_Clk	Rps_Cmd_Idx_Clk	Rps_End_Byte_Clk		Rps_Busy_E nSPI	Rps_CRC_Chk_Bit	No_Arg_Clk	Sdrps_Start
Default	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	WO

Rps_St_Byte_clk: Response Start Byte

0 = Response Start Byte is Cmd_index

1 = No Response start byte

Rps_Cmd_Idx_Clk:

If Rps_St_Byte_clk is 0:

0 = Response Start Byte is Cmd_index

1 = Response Start Byte is 0011_1111

If Rps_St_Byte_clk is 1:

No Response start byte

Rps_End_Byte_Clk:

Value	0_0	0_1	1_0	1_1
End_byte	CRC7	CRC7 for CID/CSD	FF	No end byte

Rps_Busy_EnSPI: Busy signal is transmitted after transmission of response (for SPI mode)

0 = Disable

1 = Enable

Rps_CRC_Chk_Bit: Rps CRC check enable

0 = Enable

1 = Disable

No_Arg_Clk: No argument

0 = Has argument

1 = No argument

Sdrps_Start: Rps TX start

0 = Not start

1 = Start

4. **SDICTL**: SD data RX config register (Addr: 0xDC)

Position	7	6	5	4	3	2	1	0
Name	Fbdat	Dat_CRC_err	Busy_Src		Busy_Cmd_en	Cmd12_en	Clr_Busy	Rdat_en
Default	0	0	0	0	0	0	0	0
Access	R/W	RO	RO	RO	R/W	R/W	WO	R/W

Fbdat: Dataout bit sel

0 = One bit data out

1 = Four bits data out

Dat_CRC_err: Data TX CRC error

0 = Correct

1 = Data CRC error

Busy_Src:

00 = Valid reception

01 = Invalid reception



1x = Busy Cmd reception

Busy_Cmd_en: Busy Cmd Enable

0 = Disable

1 = Enable



Cmd12_en: Cmd12 stop data transmission enable

0 = Disable

1 = Enable



Clr_Busy: Clear busy data state

0 = Unclear

1 = Clear



Rdat_en: Rx data enable

0 = Disable

1 = Enable

sddi_cfg = {fbdat, dat_crc_err, busy_src, busy_cmd_en, cmd12_en, 1'h0, rdat_en};

5.SDOCTL: SD data TX config register (Addr:0xE5)

Position	7	6	5	4	3	2	1	0
Name	Spi_out_busy	Spi_Cmd_Dis	Rca_Mat	Key_Rx_Enb	Sdc_Start	Dat_Sts_Val	Dat_Sts_Werr	Wdat_start
Default	0	0	X	0	0	0	0	0
Access	R/W	R/W	RO	R/W	R/W	R/W	R/W	W0

Spi_out_busy: At SPI mode, card is at busy state, once clear busy, card jumps out of busy state immediately.
Note: This bit is used for the card that can't clear busy after host stop clk.

0 = Once clear busy, after several clk, card jumps out of busy state.

1 = Once clear busy, card jumps out of busy state immediately.

Spi_Cmd_Dis: Disable Cmd RX (Only Used in SPI MODE)

0 = Enable Cmd RX

1 = Disable



Rca_Mat: Cmd argument[31:16] match

0 = Dismatch

1 = Match

Key_RX_Enb: Argument Rx enable

0 = Diable

1 = Enable

Sdc_Start: SD Cmd start



0 = Not start

1 = Start

Dat_Sts_Val: Data: Data response token control about 'staus'

For SPI mode, if spi_dat_crc_werr = 0:

If host care data crc result, you should set this bit as "0"

if Status of data response token is 101, it means data CRC error

if Status of data response token is 010, it means Data Rx without CRC error

if host don't care data crc result, you should set this bit as "1"

Status of data response token is fixed with 010, Data accepted regardless of data crc error

Dat_Sts_Werr: Data response token control about 'staus'

For SPI mode, if spi_dat_crc_werr = 1, status of data response token is 110, Data rejected due to a write error

Wdat_start: Data TX start

0 = Not start

1 = Start

sddo_cfg = {1'b0, spi_cmd_dis, rca_mat, key_rx_enb, sdc_start, dat_sts_val, dat_sts_werr, 1'h0};

6.SFLAG: (Addr:0x90)

Position	7	6	5	4	3	2	1	0
Name	Busy_src		Card_sta_inv	Cmd_sts_dec[0]	Dat_crc_err	Rps_sts	Rca_mat	Cmd_crc_err
Default	0	0	0	X	0	0	X	0
Access	RO	RO	RO	RO	RO	RO	RO	RO

Busy_Src:

00 = Valid reception

01 = Invalid reception

1x = Busy Cmd reception

Card_sta_inv: Card state invalid

0 = Card state valid

1 = Card state invalid

Cmd_sts_dec[0]: CMD END

0 = Cmd state is 1,2,or 3

1 = Cmd state is 0 ,it means cmd end

Dat_crc_err: Data TX CRC error

0 = CRC correct

1 = Error

Rps_sts: Card states valid or Cmd CRC correct

0 = Valid and correct



1 = No Response or Invalid or wrong

Rca_Mat: Cmd RCA match

0 = Dismatch

1 = Match

Cmd_crc_err: Cmd CRC error

0 = CRC correct

1 = error

7. SHCTL: SD Host Control (Addr:0xCD)

Position	7	6	5	4	3	2	1	0
Name	Cmd_Sts_Dec				bus_width8	Force_Sdmod	CMD7_busy_en	Di_dat0_clk
Default	X	X	X	X	0	0	1	X
Access	RO	RO	RO	RO	R/W	R/W	R/W	RO

--	--	--	--	--	--	--	--	--

Cmd_Sts_Dec: Cmd states decode

0001 = Cmd state is 00

0010 = Cmd state is 01

0100 = Cmd state is 10

1000 = Cmd state is 11

Bus_width8:MMC bus width

0: 1bit or 4 bits bus

1: 8bits bus

Force_Sdmod: Force the card to change to SD Mode

0 = Keep current Mode

1 = Force to SD Mode

CMD7_busy_en: CMD7 busy enable //CMD7 need to send busy signal when RCA match.

0: needn't to send busy

1: need to send busy(default)

Di_dat0_chk: RX Data0

sdhost_cfg = {cmd_sts_dec, Bus_width8, force_sdmod, cmd7_rca_nchk, di_dat0_clk};

8:SR1CTL: (Addr:0xCE)

Position	7	6	5	4	3	2	1	0
Name	Reserved			App_cmd	Stop_dat_auto	Reserved	R6_e	R1_e
Default	0	0	0	0	0	0	1	1
Access	-	-	-	R/W	R/W	-	R/W	R/W

App_cmd: Application CMD

0 = Not Application CMD

1 = Application CMD

Stop_dat_auto: Stop data transmit auto

R6_e: Response6 enable

0 = unable



1 = enable

R1_e: Response1 enable

0 = unable



1 = enable

9.SSTA: Card State Config (Addr:0xCF)

Position	7	6	5	4	3	2	1	0
Name	Reserved			Card_sta_update	Card_sts			
Default	0	0	0	0	0			

Access	-	-	-	WO	R/W	R/W	R/W	R/W
--------	---	---	---	----	-----	-----	-----	-----



Card_sta_update: Update Card status

0 : Card will go to "Card_sts" state regardless of current state.

1 : CMD_CRC_err = 0, Card will go to next Card state.

CMD_CRC_err = 1, Card will keep current Card state.

Card_sts: Card status

0000 = Idle

0001 = Ready

0010 = Ident

0011 = Standby

0100 = Tran

0101 = Data

0110 = Receive data

0111 = Program

1000 = Disconnect

1001 = bus_test

1010 = sleep

1011 = irq

1100 = Inactive

Others: Error

10.SDODLY: Rpsdo_cfg (Addr:0xD1)

Position	7	6	5	4	3	2	1	0
Name	Rpsdo_dly							Rpsdo_dly_e
Default	0							0
Access	R/W							R/W

Rpsdo_dly: Before Data start bit TX Rpsdo_dly * 8 delay cycle

Rpsdo_dly_e: Delay enable

0 = unable

1 = enable

11:SRCA0: (Addr:0xD2)

Position	7	6	5	4	3	2	1	0
----------	---	---	---	---	---	---	---	---

Name	Rcal
Default	XXXX_XXXX
Access	R/W

Rcal: RCA low byte

12.SRCA1: (Addr:0xD3)

Position	7	6	5	4	3	2	1	0
Name	Rcah							
Default	XXXX_XXXX							
Access	R/W							

Rcah: RCA high byte

13.SDOTK: Data TX start token(For SPI data out only) (Addr:0xDD)

Position	7	6	5	4	3	2	1	0
Name	Do_token							
Default	XXXX_XXXX							
Access	R/W							

Do_token: Data TX start token

For single block read:

Only first byte: Start Block

1	1	1	1	1	1	1	0
---	---	---	---	---	---	---	---

14.SDXADR0: (Addr:0xD4)

Position	7	6	5	4	3	2	1	0
Name	Mem_adr_dat_sdc[7:0]							
Default	XXXX_XXXX							
Access	WO							

Mem_adr_dat_sdc[7:0]: DMA addr 7 to 0 for data

15.SDXADR1: (Addr:0xD5)

Position	7	6	5	4	3	2	1	0
Name	Reserved				Mem_adr_dat_sdc[12:8]			
Default	XXXX				XXXX			
Access	-				WO			

Mem_adr_dat_sdc[12:8]: DMA addr 12 to 8 for data

16.SRXADR0: (Addr:0xD6)

Position	7	6	5	4	3	2	1	0
Name	Mem_adr_cmd_sdc[7:0]							
Default	XXXX_XXXX							
Access	WO							

Mem_adr_cmd_sdc[7:0]: DMA addr 7 to 0 for Cmd

17:SRXADR1: (Addr:0xD7)

Position	7	6	5	4	3	2	1	0
Name	Reserved				Mem_adr_cmd_sdc[12:8]			
Default	XXXX				XXXX			
Access	-				WO			

Mem_adr_cmd_sdc[12:8]: DMA addr 12 to 8 for Cmd

18.SRDL0: (Addr:0xDE)

Position	7	6	5	4	3	2	1	0
Name	Dma_no_bytel_cmd							
Default	0000_0000							
Access	WO							

Dma_no_bytel_cmd: Dma_no_bytel_cmdbytes for Cmd reception and Rps transmission

19:SRDL1: (Addr:0xDF)

Position	7	6	5	4	3	2	1	0
Name	Apu	Force_cmd12	Tokchk		Dat_not_mat	Reserved		Dma_no_byteh_cmd
Default	0	0	0		0	x		0
Access	WO	WO	WO		WO	X		WO

Apu: Cmd output enable

0 = Cmd output disable

1 = Cmd output enable

Force_cmd12: Force the card to transmtion

0 = Keep transmtion

1 = Stop transmtion

Tokchk: Token check

0x = Check Multiple block write start token for SPI Mode

x0 = Check single block read ,write and Multiple block read start token for SPI Mode

11 = Stop tran token

Dat_not_mat: Data out not match

0 = match

1 = Not match

Dma_no_byteh_cmd: Cmd reception and Rps transmission bytes
={ Dma_no_byteh_cmd , Dma_no_bytel_cmd} +1

20.SDDL0: (Addr:0xE1)



Position	7	6	5	4	3	2	1	0
Name	Dma_no_bytel_dat							
Default	XXXX_XXXX							
Access	WO							

21:SDDL1: (Addr:0xE2)

Position	7	6	5	4	3	2	1	0
Name	Reserved							Dma_no_bytel_cmd
Default	XXXX_XXXX							
Access	-							WO

Dma_no_byteh_dat: has {{dma_no_byteh_dat,dma_no_bytel_dat}+1}bytes data for rx or tx

Data in memory after RX:

For example dma_no_byte = n



When $n\%4 =$

1: the last 1 byte data in memory : {data_n,24'h0}

2: the last 2 bytes data in memory :{data_n,data_(n-1),16'h0}

3: the last 3 bytes data in memory :{data_n,data_(n-1),data_(n-2),8'h0}

0: the last 4 bytes data in memory :{data_n,data_(n-1),data_(n-2),data_(n-3)}

22:SRDCTL: (Addr:0xE4)

Position	7	6	5	4	3	2	1	0
Name	Rps_dly_en	Rps_dly_NO						
Default	XXXX_XXXX							
Access	WO							

Rps_dly_no: Rps delay "Rps_dly_no + 4" SD clock after Cmd received

Rps_dly_en: Rps delay enable

0 = disable

1 = enable

23.SINDX: (Addr:0xE6)

Position	7	6	5	4	3	2	1	0
Name	Reserved		CMD_index					

Access	RO	W/R						
--------	----	-----	--	--	--	--	--	--

DONE:When write to SD_CLK_COUNTER , DONE will turn “0”,hardware starts to count SD_CLK untill equal to SD_CLK_COUNTER,then DONE will turn “1”.

SD_CLK_COUNTER:SD_CLK_COUNTER*2=SD host CLK cycles you want to count.Mean that if you want to

count “M” SD host clk cycles,you should set SD_CLK_COUNTER equal half of “M”。 eg : if you want to count 40 SD host clk cycles, you need only set SD_CLK_COUNTER equal 20. When SD_CLK_COUNTER = 7f,the maximum SD host clk cycles you can count up to FE(254 SD host clk cycles).

IF you change the SD_CLK_COUNTER when hardware is counting, it will start to count from “0”untill equal to

the latest SD_CLK_COUNTER.

29:**Last_CMD**: (Addr:0xBA)

Position	7	6	5	4	3	2	1	0
Name		HCS	Last_cmd					
Default		x	x	x	x	x	x	x
Access	0	RO	RO	RO	RO	RO	RO	RO

HCS: High capacity device

0 = less than or equal 2GB

1 = more than 2GB

Last_cmd:Host last sent CMD and device responded correctly.

30:**BOOT_CFG**: (Addr:0xBD)

Position	7	6	5	4	3	2	1	0
Name	MMC_SD	Boot_dat_start	Boot_dat_end	BOOT_EN	Cmd_low_m74	Cmd0_rst	Cmd0_f0	Cmd0_fa
Default	0	0	0	1	0	0	0	0
Access	RO	RO	RO	R/W	RO	RO	RO	RO

MMC_SD: Hard ware detect SD and MMC

0 = MMC device

1 = SD device

Boot_dat_start: device begin to send boot data

0 = device can't start to send boot data.

1 = device can start to send boot data

Boot_dat_end: device stop sending boot data.

0 = go on sending

1 = stop sending

BOOT_EN:boot enable

0 = boot unable

1 = boot enable

Cmd_low_m74: CMD line low more than 74 CLK

0 = less than 74 CLK

1 = more than 74 CLK

Cmd0_rst:CMD0 with arg = 32'h0

Cmd0_f0:CMD0 with arg = 32'hf0f0_f0f0

Cmd0_fa:CMD0 with arg = 32'hfff_fffa

13.2 SD Mode Operation

1. SD MODE R1 Response Operation:

// Config DMA Address

MOV SRXADR0, #21H

MOV SRXADR1, #01H

// Config DMA Length 4 Bytes

MOV SRDL0, #3H

MOV SRDL1, #0H

// Config R1 Response Enable

MOV SR1CTL, #01H

// Kick Start Response Transmission

MOV SRCTL, #01H

2. SD MODE R2 Response Operation:

// Config DMA Address

MOV SRXADR0, #21H

MOV SRXADR1, #01H

// Config DMA Length 15 Bytes

MOV SRDL0, #14

MOV SRDL1, #0H

// Kick Start Response Transmission, CRC7 for CID/CSD

MOV SRCTL, #51H

3. SD MODE R3 Response Operation:

// Config DMA Address

MOV SRXADR0, #21H

MOV SRXADR1, #01H

// Config DMA Length 4 Bytes

MOV SRDL0, #3

MOV SRDL1, #0H

// Kick Start Response Transmission

MOV SRCTL, #61H

4. SD MODE R6 Response Operation:

```
// Config DMA Address
MOV SRXADR0, #21H
MOV SRXADR1, #01H
// Config DMA Length 4 Bytes
MOV SRDL0, #3
MOV SRDL1, #0H
// Config R6 Response Enable
MOV SR1CTL, #02H
// Kick Start Response Transmission
MOV SRCTL, #01H
```

5. SD MODE R7 Response Operation:

```
// Config DMA Address
MOV SRXADR0, #21H
MOV SRXADR1, #01H
// Config DMA Length 4 Bytes
MOV SRDL0, #3
MOV SRDL1, #0H
// Kick Start Response Transmission
MOV SRCTL, #01H
```

6. SD MODE Data Output Operation:

```
// Config DMA Address
MOV SDXADR0, #21H
MOV SDXADR1, #01H
// Config DMA Length 64 Bytes
MOV SDDL0, #63
MOV SDDL1, #00
// Kick Start Data Transmission
ORL SDOCTL, #(1 << 0)
```

7. SD MODE Data Recieve Operation:

```
// Config DMA Address
MOV SDXADR0, #21H
MOV SDXADR1, #01H
// Config DMA Length 64 Bytes
MOV SDDL0, #63
MOV SDDL1, #00
// Kick Start Data Recieve
```

```
ORL SDICTL,#(1 << 0)
```

13.3 SPI Mode Operation

1. SPI MODE R1 Response Operation:

```
// Config DMA Address
```

```
MOV SRXADR0,#21H
```

```
MOV SRXADR1,#01H
```

```
// Config DMA Length 1 Bytes
```

```
MOV SRDL0,#0
```

```
MOV SRDL1,#0
```

```
// Kick Start Response Transmission
```

```
MOV SRCTL,#11110001B
```

2. SPI MODE R2 Response Operation:

```
// Config DMA Address
```

```
MOV SRXADR0,#21H
```

```
MOV SRXADR1,#01H
```

```
// Config DMA Length 2 Bytes
```

```
MOV SRDL0,#1
```

```
MOV SRDL1,#0
```

```
// Kick Start Response Transmission
```

```
MOV SRCTL,#11110001B
```

3. SPI MODE R3 Response Operation:

```
// Config DMA Address
```

```
MOV SRXADR0,#21H
```

```
MOV SRXADR1,#01H
```

```
// Config DMA Length 5 Bytes
```

```
MOV SRDL0,#4
```

```
MOV SRDL1,#0
```

```
// Kick Start Response Transmission
```

```
MOV SRCTL,#11110001B
```

4. SPI MODE R7 Response Operation:

```
// Config DMA Address
```

```
MOV SRXADR0,#21H
```

```
MOV SRXADR1,#01H
```

```
// Config DMA Length 5 Bytes
```

```
MOV SRDL0,#4
```

```
MOV SRDL1,#0
```

```
// Kick Start Response Transmission
```

```
MOV SRCTL,#11110001B
```

5. SPI MODE Data Transmission and receive operation is the same as SD mode

13.4 EXINST Opcode Summary

Table 13-1: EXINST Opcode

Mnemonics		Opcode			Flags
		1st Byte	2nd Byte	3rd Byte	
MOV32	ERn, EDPi	1010 0101	1110 nn0i		EZ
	EDPi, ERn	1010 0101	1110 nn1i		
	ERn, ERm	1010 0101	0100 nnmm		
	ER4, ERn	1010 0101	1001 nn00		
	ERn, ER4	1010 0101	1001 nn10		
NOT32	ERn	1010 0101	0001 nn00		
INC32	ERn	1010 0101	0001 nn10		EZ
DEC32	ERn	1010 0101	0001 nn11		EZ
RAN32	EDPi, ERn	1010 0101	0000 nn1i		
ANL32	ERn, EDPi	1010 0101	0010 nn0i		EZ
	EDPi, ERn	1010 0101	0010 nn1i		
	ERn, ERm	1010 0101	0011nnmm		
ORL32	ERn, EDPi	1010 0101	0100 nn0i		EZ
	EDPi, ERn	1010 0101	0100nn1i		
	ERn, ERm	1010 0101	0101 nnmm		
XRL32	ERn, EDPi	1010 0101	0110nn0i		EZ
	EDPi, ERn	1010 0101	0110 nn1i		
	ERn, ERm	1010 0101	0111 nnmm		
ADD32	ERp, EDPi, ERn	1101 0100	00pp nn0i		EZ, EC
	EDPi, ERn, ERp	1101 0100	00pp nn1i		
	ERp, ERn, ERm	1101 0100	01pp nnmm		
SUB32	ERp, EDPi, ERn	1101 0100	10pp nn0i		EZ, EC
	EDPi, ERn, ERp	1101 0100	10pp nn1i		
	ERp, ERn, ERm	1101 0100	11pp nnmm		
ROTR32	ERn, ER8	1010 0101	1101 nn00		
ROTL32	ERn, ER8	1010 0101	1101 nn01		
ROTR8	EACC, ER8	1010 0101	1000 0000		
ROTL8	EACC, ER8	1010 0101	1000 0001		
CLR32	ERn	1010 0101	0001 nn01		
ADDDPi		1010 0101	0000 010i		
SUBDPi		1010 0101	0000 110i		

INCDPi		1010 0101	0000 000i		
DECDPi		1010 0101	0000 100i		
INC4DPi		1010 0101	1101 0i11		
DEC4DPi		1010 0101	1101 1i11		
MUL16	ERn	1010 0101	1100 nn00		EZ
DIV16	ERn	1010 0101	1101 nn10		

Notes:

ERn, ERm, ERp:



32-bit register *ER0-ER3*. n=m=p is allowed.

ERni, ERmj:

8-bit SFR *R00-R03, R10-R13, R20-R23, R30-R33*. ni=mj is allowed.

EDPi:

XRAM addressing pointer *DPTR0* or *DPTR1*.

ER8:

8-bit SFR *R8*.

EACC:

8-bit SFR *ACC*.

14 Random data generator

CMPCTL

Position	7	6	5	4	3	2	1	0
Name	MODESEL			CPLAST	CPLOAD	CPDISEL	LBCNT	
Default	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

MODESEL : Mode Select

000: IDLE mode.

001: Compare mode. Calculate the different bit amount.

011: NandFlash read SRAM XOR mode. While NandFlash module reads data from SRAM, the data will XOR with a random value.

100: DMA mode, RCMP module reads data from SRAM , and XOR a random value, then write the data to the

same address with read address . When DMA done, DMA mode will return to the IDLE mode.

101: NandFlash write SRAM XOR mode. While NandFlash module writes data to SRAM, the data will XOR with a random value.

110: Bad Column Search mode.可以将错误的byte地址写到SRAM中, 16Bit flash和8x2flash 同样, 均是以byte为单位来表示地址。

(注意: 使能这个功能且读flash速度最快时, 不能有除nandflash外 DMA优先级比BCS高的模块在和BCS使用同一块memory, 优先级比BCS高的模块有USB, nandflash, BCH, sd host; 而且NFC也不能与优先级比它高的DMA同时使用同一块memory, 优先级比NFC高的模块有USB)

CPLAST: Last compare enable (While CPDISEL is 0, this bit do not work).

0: Disable;

1: Enable;

CPLOAD: Load the random data every clock (While CPDISEL is 0, this bit do not work).

0: Disable;

1: Enable;

CPDISEL: Compare data source select.如果是bad column search mode时, 这位必须为0.

0: Compare data from nand flash.

1: Compare data from ER0



LBCNT: Last compare byte amount select.如果这时有坏列管理(BCM), 则这个寄存器应该填有效数据的最后一次DMA的byte的数量。例如从flash读出的数据有9byte, 但是坏列有2个, 则有效数据为7byte, 这时LBCNT应该填3。

00: 4 byte

01: 1 byte

10: 2 byte

11: 3 byte



CMPCCTL1

Position	7	6	5	4	3	2	1	0
Name	—	—	—	—	BCSDONE	LASTPT	TESTEN	FIXEN
Default	—	—	—	—	0	0	0	0
Access	—	—	—	—	R	R	R/W	R/W

BCSDONE : bad column search done flag. 退出 Bad Column Search mode后这个寄存器会清零。

0:not done

1:done

LASTPT : 最后一次坏列地址DMA是否为完整的32bit

0:最后一次坏列地址DMA为完整的32bit;

1:最后一次坏列地址DMA为单个16bit。这时最后一个坏列地址会被copy 1次, 凑够32bit写到sram中。

退出 Bad Column Search mode后这个寄存器会清零。

TESTEN : DMA test enable. Only used in DMA mode.

0: Disable, DMA mode work normally.

1. Enable, DMA mode will not read data from SRAM, and only write the random data to SRAM

FIXEN : Fix data enable. It can be used for all mode.

0: Disable fixes data, the data will be random

1. Enable fixes data, the data will not change

CMPSTADR

Position	7	6	5	4	3	2	1	0
Name	STADDRL		STADDRH/ STADDRL					
Default	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

DMA start address: It is used for DMA mode. Must write the high address first (13~8bits), and then write the (7~0bits). This address is equal to (XRAM address >> 2).

读这个寄存器时，第一次读高6bit(13~8bit)，第二次读低8bit(7~0bits)。每完成1次DMA，这个寄存器会自动加1。（SRAM物理地址）

CMPDMAMT

Position	7	6	5	4	3	2	1	0
Name	CMPDMAMTL			CMPDMAMTH/ CMPDMAMTL				
Default	0	0	0	0	0	0	0	0
Access	W	W	W	W	W	W	W	W

DMA 32bit data amount: It is used for DMA mode. Must write the high bit first (10~8bits), and then write the (7~0bits). This SFR is equal to the 32bit data amount -1. If you want to DMA one data, you can write 0 to this SFR.

CPERCNT

Position	7	6	5	4	3	2	1	0
Name	Compare counter							
Default	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

CPERCNT: This count will calculate the different bits amount that is between FIFO data and ER0 or FIFO data and NandFlash data. 在bad column search mode时，这个寄存器的功能跟compare mode一样。

CMPFIFO

Position	7	6	5	4	3	2	1	0
Name	Compare FIFO							
Default	0	0	0	0	0	0	0	0
Access	W	W	W	W	W	W	W	W

Write this SFR to configure the seed, must write 4 times. Firstly write the 7~0bits, and then write the 15~8bits, and then write the 23~16 bits, finally write the 31~24 bits;

Operation Guide

Compare mode

1. Write the 32bits seed to the CMPFIFO : (The seed is 0x12345678 in this example)
CMPFIFO = 0x78;
CMPFIFO = 0x56;
CMPFIFO = 0x34;
CMPFIFO = 0x12;
2. If need another function, you can change CMPCTL or CMPCTL1.(This example is not change)
3. Enable the compare mode , and select the last compare byte amount :
CMPCTL = 0x20 ;
4. If finish compare , read the CMPCNT, and then select IDLE mode to clear CMPCNT.

NandFlash read SRAM XOR mode

1. Write the 32bits seed to the CMPFIFO : (The seed is 0x12345678 in this example)
CMPFIFO = 0x78;
CMPFIFO = 0x56;
CMPFIFO = 0x34;
CMPFIFO = 0x12;
2. Write the XOR data amount: (This example is 0x0100(32bit data amount).)
unsigned int temp;
temp = 0x0100 - 0x01;
CMPDMAMT = (temp >> 8);
CMPDMAMT = temp & 0x00FF;
3. Enable the NandFlash read SRAM XOR mode :
CMPCTL = 0x60 ;
4. Then the NandFlash module read data from SRAM will XOR a random value.

NandFlash Write SRAM XOR mode

1. Write the 32bits seed to the CMPFIFO : (The seed is 0x12345678 in this example)
CMPFIFO = 0x78;
CMPFIFO = 0x56;
CMPFIFO = 0x34;
CMPFIFO = 0x12;
2. Write the XOR data amount: (This example is 0x0100(32bit data amount).)
unsigned int temp;
temp = 0x0100 - 0x01;
CMPDMAMT = (temp >> 8);
CMPDMAMT = temp & 0x00FF;
3. Enable the NandFlash write SRAM XOR mode :
CMPCTL = 0xA0 ;
4. Then the NandFlash module write data to SRAM will XOR a random value.

DMA mode

1. Write the 32bits seed to the CMPFIFO : (The seed is 0x12345678 in this example)
 CMPFIFO = 0x78;
 CMPFIFO = 0x56;
 CMPFIFO = 0x34;
 CMPFIFO = 0x12;
2. Write the DMA start address: (This example is 0x1000(XRAM address))
 unsigned int temp;
 temp = 0x1000 >> 2;
 CMPSTADR = (temp >> 8);
 CMPSTADR = temp & 0x00FF;
3. Write the DMA amount: (This example is 0x0100(32bit data amount).)
 unsigned int temp;
 temp = 0x0100 - 0x01;
 CMPDMAMT = (temp >> 8) ;
 CMPDMAMT = temp & 0x00FF;
4. Enable the DMA mode :
 CMPCTL = 0x80 ;
5. Wait the MODESEL clear;

DMA test mode

1. Write the 32bits seed to the CMPFIFO : (The seed is 0x12345678 in this example)
 CMPFIFO = 0x78;
 CMPFIFO = 0x56;
 CMPFIFO = 0x34;
 CMPFIFO = 0x12;
2. Write the DMA start address: (This example is 0x1000(XRAM address))
 unsigned int temp;
 temp = 0x1000 >> 2;
 CMPSTADR = (temp >> 8);
 CMPSTADR = temp & 0x00FF;
3. Write the DMA amount: (This example is 0x0100(32bit data amount).)
 unsigned int temp;
 temp = 0x0100 - 0x0001;
 CMPDMAMT = (temp >> 8) ;
 CMPDMAMT = temp & 0x00FF;
4. Enable the DMA test mode :
 CMPCTL1 = 0x02;
5. Enable the DMA mode :
 CMPCTL = 0x80 ;
6. Wait the MODESEL clear;

Bad column search mode

(注意：使能这个功能且读**flash**速度最快时，不能有除**nandflash**外 **DMA**优先级比**BCS**高的模块在和**BCS**使用同一块**memory**，优先级比**BCS**高的模块有**USB**，**nandflash**，**BCH**，**sd host**；而且**NFC**也不能与优先级比它高的**DMA**同时使用同一块**memory**，优先级比**NF**高的模块有**USB**)

1. Write the 32bits seed to the CMPFIFO : (The seed is 0x12345678 in this

example)

```
CMPFIFO = 0x78;
```

```
CMPFIFO = 0x56;
```

```
CMPFIFO = 0x34;
```

```
CMPFIFO = 0x12;
```

2. Write the DMA start address: (This example is 0x1000(逻辑地址)),坏列地址将会被写入到sram这里去

```
unsigned int temp;
```

```
temp = 0x1000 >> 2;//0X400
```

```
CMPSTADR = temp >> 8;
```

```
CMPSTADR = temp & 0x00FF;
```

3. Enable the compare mode , and select the last compare byte amount 1 byte;(注意, BCS mode只支持nandflash DMA到SRAM的情况, 即CPDISEL必须为0)

```
CMPCTL = 0xC1 ;
```

4. Read nandflash and wait it done;

5. wait BCSDONE:

```
while(~(CMPCTL1 & 0X08));
```

6. 读该次操作总错误bit数(可选)

```
i= CPERCNT;
```

7. 读当前的SRAM address和LASTPT:

```
temp = CMPSTADR;
```

```
temp = (temp << 8) | CMPSTADR;
```

```
i= CMPCTL1 & 0X04;
```

假如得出temp = 0x402且LASTPT为0,则说明出现了4个错误的byte, 这4个错误的byte按出现的先后顺序, 其地址和错误bit数(1~8)会分别存放在SRAM中, 如下图:

0X402	EMPTY	EMPTY
0X401	ErrByte3(16BIT)	ErrByte2(16BIT)
0X400	ErrByte1(16BIT)	ErrByte0(16BIT)

假如得出temp = 0x402且LASTPT为1,则说明出现了3个错误的byte, 这3个错误的byte按出现的先后顺序, 其地址和错误bit数(1~8)会分别存放在SRAM中, 如下图:

0X402	EMPTY	EMPTY
0X401	ErrByte2(16BIT)	ErrByte2(16BIT)
0X400	ErrByte1(16BIT)	ErrByte0(16BIT)

其中每个16bit的格式如下:

注意Address指的是这个Byte在这次操作中的DMA序列, NF_CTL每读够4个byte就会往SRAM DMA一次, 例如第一次DMA的第3byte错了, 则这个地址为2, 如果第二次DMA的第一个byte错了, 则地址为4, 依此类推。16Bit flash和8x2flash同样, 均是以byte为单位来表示地址。

Error bit amount表示这个byte有多少bit错了, 0表示1bit错, 7表示8bit错。

Byte0	Error Bit Amount		Address															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
BIT																		

8. Disable Bad Column Mode:

```
CMPCTL = 0X00
```

15 Electrical Characteristics

15.1 Absolute Maximum Ratings

Symbol	Parameter	MIN	MAX	Unit
VDD	DC Power Supply	-0.3	+3.6	V
Vin	Input Voltage	VSS-0.3	VDD+0.3	V
Vout	Output Voltage	VSS-0.3	VDD+0.3	V
Ta	Operating Temperature	0	+75	°C
Tst	Storage Temperature	-40	+125	°C

15.2 Recommended Operating Conditions

Symbol	Parameter	MIN	TYP	MAX	Unit
VDD	Supply Voltage	3.0	3.3	3.6	V
VDD18	Supply Voltage	1.62	1.8	1.98	V
Vin	Input Voltage	0	-	VDD	V
Tamb	Ambient Temperature	0	-	+70	°C

15.3 Electrical Characteristics of 3.3V I/O Cell

Symbol	Parameter	CONDITIONS	Limits			Unit
			MIN	TYP	MAX	
VDD	Power Supply	3.3V I/O	3.0	3.3	3.6	V
Vil	Input Low Voltage	LVTTL			0.8	V
Vih	Input High Voltage		2			V
Vol	Output Low Voltage	I _{ol} =~8/16mA			0.4	V
Voh	Output High Voltage	I _{oh} =~8/16mA	2.4			V
Rpu	Input Pull-up Resistance	Vin = 0		30K		ohm

15.4 Electrical Characteristics of RC

Symbol	Parameter	MIN	TYP	MAX	Unit
RC output	Low power output		10		MHz (trim value = 5'b10000)
	High performance output		70		
Tamb	Ambient Temperature	0	-	+70	°C

15.5 Electrical Characteristics of LDO

Symbol	Parameter	MIN	TYP	MAX	Unit
VDDIO	Supply Voltage	2.0	3.3	3.6	V
VDD18	LDO output	1.62	1.8	1.98	V

BORB	BOR trigger Voltage		1.9		V
Ivdd18	LDO output driving			150	mA
Tamb	Ambient Temperature	0	-	+70	°C

Appendix Revision History

Date	Version	Revised items	Revised by
11年12月22日	0.0.1	1.	zhangjing jing_zhang@buildwin.com.cn
		1.	
		1.	
		1.	
		1.	

The information in this document is believed to be accurate in all respects at the time of publication but is subject to change without notice. AppoTech assumes no responsibility for errors and omissions, and disclaims responsibility for any consequences resulting from the use of information included herein. Additionally, AppoTech assumes no responsibility for the functioning of undescribed features or parameters. AppoTech reserves the right to make changes without further notice. AppoTech makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does AppoTech assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. AppoTech products are not designed, intended, or authorized for use in applications intended to support or sustain life, or for any other application in which the failure of the AppoTech product could create a situation where personal injury or death may occur. Should Buyer purchase or use AppoTech products for any such unintended or unauthorized application, Buyer shall indemnify and hold AppoTech harmless against all claims and damages.