Freescale Semiconductor
Application Note

Document Number: AN3513
Rev. 0, 09/2007

ColdFire ATA Host Controller

by: Juan Mendoza
Freescale 32-bit Consumer and Industrial Operations

1 Introduction

This application note describes the ColdFire® ATA host controller. It covers signal description, functional description, programming model, and initialization.

The ATA host controller is used to connect:

- Hard disks
- CD-ROMs
- DVDs
- Flash storage devices

Features include:

- Programmable timing on the ATA bus. Works with wide range of bus frequencies
- 128-byte FIFO
- FIFO receive, transmit, and end of transmission alarms to DMA unit
- Zero wait-state transfers between DMA bus and FIFO

Contents

freescale™ semiconductor

2 ATA Host Controller

The figure below illustrates the block diagram of the ATA host controller:

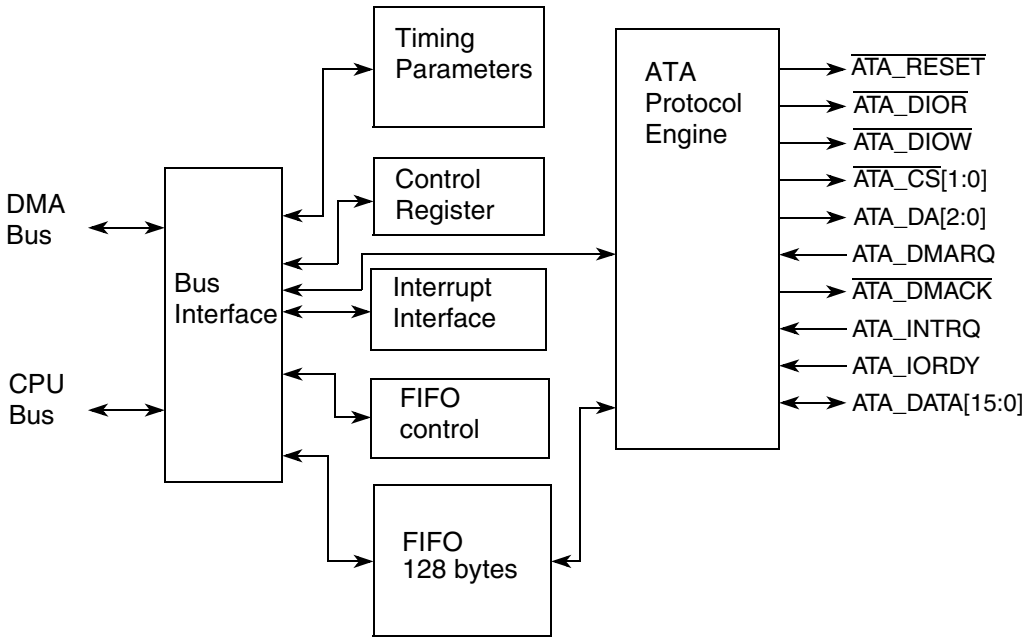

Figure 1. Block Diagram

The ColdFire ATA host controller is fully compatible with: ATA/ATAPI-6 specification (*AT Attachment with Packet Interface Extension*). The ColdFire ATA host controller can control up to two ATA devices as defined in ATA/ATAPI-6 specification and supports all three modes:

- PIO mode 0, 1, 2, 3 and 4 for up to 16.7 MBytes/sec
- Multiword DMA mode 0, 1 and 2 for up to up to 16.7 MBytes/sec
- Ultra DMA modes
  - Modes 0, 1, 2, 3 and 4 with bus clock of 50 MHz or higher for up to 66.7 MBytes/sec
  - Mode 5 with system bus clock of 80 MHz or higher for 100 MBytes/sec

The ColdFire ATA controller is accessible by both the CPU and DMA. Before accessing the ATA bus, the host must program the timing parameters on the ATA bus. A set of timing registers for each transfer mode control the timing on the ATA bus. Most timing parameters are programmable as a number of clock cycles (1 to 255).

2.1 External Signal Description

Table 1. External Signals

Name	Function	Reset State	I/O
$\overline{\text{ATA\_RESET}}$	ATA bus reset signal	0	O
$\overline{\text{ATA\_DIOR}}$	ATA bus read strobe	1	O
$\overline{\text{ATA\_DIOW}}$	ATA bus write strobe	1	O
$\overline{\text{ATA\_CS}}$[1:0]	ATA bus chip selects	1	O
ATA_DA[2:0]	ATA bus address lines	0	O
ATA_DMARQ	ATA bus DMA request	—	I/O
$\overline{\text{ATA\_DMACK}}$	ATA DMA acknowledge	1	O
ATA_INTRQ	ATA bus interrupt request	—	I/O
ATA_IORDY	ATA bus I/O channel ready	—	O
ATA_DATA[15:0]	ATA data bus (little endian)	Hi-Z	Three-state I/O
ATA_BUFFER_EN	ATA buffer direction control signal	0	O

- $\overline{\text{ATA\_RESET}}$ — When asserted, the ATA bus is in reset. After power-on, the ATA bus is in reset.
- $\overline{\text{ATA\_DIOR}}$ — During PIO and Multiword DMA transfer, this signal functions as a read strobe. During Ultra DMA in burst, this signal functions as HDMARDY. During Ultra DMA out burst, this signal functions as host strobe.
- $\overline{\text{ATA\_DIOW}}$ — During PIO and Multiword DMA transfer, this signal functions as a write strobe. During Ultra DMA burst, this signal functions as STOP and is asserted when the host wants to terminate a running UDMA transfer.
- $\overline{\text{ATA\_CS}}$[1:0] — The chip selects for the ATA bus.
- ATA_DA[2:0] — The address lines of the ATA bus.
- ATA_DMARQ — Device asserts to transfer data using Multiword DMA or Ultra DMA mode.
- $\overline{\text{ATA\_DMACK}}$ — The host negates this signal when it grants the DMA request.
- ATA_INTRQ — The device asserts this signal to interrupt the host CPU.
- ATA_IORDY — This signal has three functions:
  - IORDY — Active low wait during PIO cycles
  - DDMARDY — Active low device ready during Ultra DMA out transfers
  - DSTROBE — Device strobe during Ultra DMA in transfers
- ATA_DATA[15:0]) — This is the bidirectional, three-state ATA data bus.
- ATA_BUFFER_EN — ATA buffer directional control signal.

3 ATA Functional Description

The ATA interface provides two ways to communicate with the ATA peripherals connected to the ATA bus. After programming the appropriate timing parameters, two protocols can be active at the same time on the ATA bus.

- PIO Mode — Can be performed at any time by the host CPU or the host DMA to the ATA bus. During a PIO transfer the incoming bus cycle is translated to an ATA PIO bus cycle by the ATA protocol engine. During PIO mode, the FIFO is not active, so no buffering of data occurs and the host CPU/DMA cycle is stalled until the ATA bus read data is available, or the bus data can be put on the ATA bus during a write. PIO mode is a slow protocol, mainly intended to program the ATA disk drive, but can be used to transfer data to/from the disk drive.
- DMA Mode — Two different DMA protocols are supported on the ATA bus: Ultra DMA mode (UDMA) and Multiword DMA mode (MDMA). Selection is made using the ATA host control register (ATA_CR). Data is transferred between the ATA bus and the FIFO, no direct interface exists between the ATA bus and the CPU/DMA. The ColdFire ATA host controller initiates DMA transfers by:
  1. Writing the appropriate command sequence to the device registers for DMA transfer request.
  2. After the command register is written, command execution begins.
  3. The device will then request either a transfer or receive request to the DMA.
- The ColdFire ATA host controller must be programmed to accept the DMA request from the device. Once started, data transfer is organized between the ATA bus and the FIFO. The FIFO informs the ColdFire DMA unit when it needs to be refilled or emptied. In either case, it sends an alarm flag to the ColdFire DMA unit. When the ColdFire DMA unit receives the FIFO transmit alarm, it must write some data to the FIFO (typically 32bytes). When the ColdFire DMA unit receives the FIFO receive alarm, it should read some data from the FIFO (typically 32 bytes). The FIFO level at which the alarms are produced is programmable. For completeness, a third alarm to the ColdFire DMA unit is provided. This alarm signals the end of the transfer. The end of transfer has two possible outcomes.
  - If the microprocessor is reading data from the device and there are less than FIFO_ALARM bytes remaining in the FIFO, the host must transfer the bytes remaining in the FIFO to memory. The number of halfwords remaining in the FIFO are reported in the FIFO_FILL register.
  - If the microprocessor is writing data to the device, no FIFO manipulation is required after transfer completion.

All transfers between the FIFO and CPU or DMA are zero wait-state transfers. When a PIO access is performed during a running DMA transfer, the DMA transfer pauses, the PIO access finishes, and the DMA transfer resumes again.

The ColdFire ATA controller offers a buffer directional control signal, ATA_BUFFER_EN, which is used to control the direction of any bidirectional buffer requiring direction control. TIME_ON and TIME_OFF are timing registers that control data flow to avoid bus contention when switching the buffer on or off.

3.1 Endianess

The ATA interface operates in big endian mode. When the DRIVE_DATA register is accessed, the bytes to/from the ATA bus are swapped. The byte order in 16-bit or 32-bit registers is:

- bits [7:0] : byte 0
- bits [15:8] : byte 1
- bits [23:8] : byte 2
- bits [31:24] : byte 3

This requires that when reading device information via the DRIVE_DATA register, a byte swapping mechanism be implemented in software. For example:

```
uint16 readDriveRegisterData(void)
{
     uint16 temp, returnValue;
     return Value = DRIVE_DATA; returnValue = (*(vuint16*)(0x900000A0));
     temp = (returnValue & 0xff00) >> 8;
     returnValue = returnValue << 8;
     return (returnValue | temp);
}
```

For data written by the CPU/DMA, no byte swap is necessary.

3.2 Sector Addressing

The addressing of data sectors recorded on the device's media is performed by logical sector address. In accordance with ATA/ATAPI-6 specification the ColdFire ATA controller supports 27-bit and 48-bit logical block addressing (LBA). In standards ATA/ATAPI-5 and earlier, a cylinder/head/sector addressing (CHS) translation was defined. This translation is obsolete, but may be implemented as defined in ATA/ATAPI-5.

3.3 Programming Model

The ColdFire ATA host controller has four groups of registers:

- ATA host controller registers — This set of registers contains the:
  - — ATA_CR — The ATA control register allows access to ATA reset/enable, FIFO control, DMA request control, and handshake enabling/disabling.
  - — ATA_ISR — The ATA interrupt status register reports status of DMA requests, controller, FIFO, and interrupts.
  - — ATA_IER — The ATA interrupt enable register allows for the enabling/disabling of various FIFO and controller interrupts.
  - — ATA_ICR — The ATA interrupt clear register allows for the clearing of select FIFO error interrupts.
  - — FIFO_ALARM — The ATA FIFO alarm register holds the value at which the FIFO requests the DMA.
- ATA timing registers — This set contains the registers that hold the timing values for PIO, MDMA, and UDMA modes. The timings for each particular mode are defined by the ATA/ATAPI-6

specification. Each register contains a number of clock cycles as calculated by dividing the timing constraint from the specification by the period of the system bus.

- TIME_ON — t_{ON} parameter for buffer control to prevent bus contention
- TIME_OFF — t_{OFF} parameter for buffer control to prevent bus contention
- PIO timing registers
  - TIME_1 — t_1
  - TIME_2W — t_2 write
  - TIME_2R — t_2 read
  - TIME_AX — t_A
  - TIME_PIORDX — t_{RD}
  - TIME_4 — t_4
  - TIME_9 — t_9
- MDMA timing registers
  - TIME_M — t_M
  - TIME_JN — t_N and t_J
  - TIME_D — t_D
  - TIME_K — t_k
- UDMA timing registers
  - TIME_ACK — t_{ACK}
  - TIME_ENV — t_{ENV}
  - TIME_RPX — t_{RP}
  - TIME_ZAH — t_{ZAH}
  - TIME_MLIX — t_{MLI}
  - TIME_DVH — t_{DVH}
  - TIME_DZFS — t_{DZFS}
  - TIME_DVS — t_{DVS}
  - TIME_CVH — t_{CVH}
  - TIME_SS — t_{SS}
  - TIME_CYC — t_{CYC} and t_{2CYC}

  ColdFire offers a buffer signal, ATA_BUFFER_EN, for direction control of any buffer requiring such control. To avoid bus contention when switching direction on the buffer, the ATA controller has two timing registers that control the timing of the data bus with respect to the ATA_BUFFER_EN signal.
  - TIME_ON - Time after ATA_BUFFER_EN signal has asserted to start driving the ATA bus.
  - TIME_OFF - Time before ATA_BUFFER_EN signal is deasserted to stop driving the ATA bus.

- ATA FIFO registers — FIFO 16-bit and 32-bit data access registers.
  - FIFO_DATA32 — 32-bit wide data port to/from FIFO

  - FIFO_DATA16 — 16-bit wide data port to/from FIFO
  - FIFO_FILL — Reports number of halfwords present in FIFO
- ATA drive registers — Access to the registers physically located on the ATA drive
  - DRIVE_DATA — Drive data register read/write used for host PIO data transfer
  - DRIVE_FEATURES — If read, error register is read. If written, features register is written
  - DRIVE_SECTOR_COUNT — Read/write drive sector count register
  - DRIVE_LBA_LOW — Read/write drive LBA_LOW register
  - DRIVE_LBA_MID — Read/write drive LBA_MID register
  - DRIVE_LBA_HIGH — Read/write drive LBA_HIGH register
  - DRIVE_DEV_HEAD — Read/write register for selecting device 0 or device 1
  - DRIVE_COMMAND_STATUS — Read returns status, will write command register.
  - DRIVE_ALT_STATUS/CONTROL — If read, register contains same information as DRIVE_STATUS. If written, register programs DRIVE_CONTROL register.

4 Initialization

ColdFire ATA host controller and device initialization covers three different groups of steps:

1. ATA module initialization
   - Enable functionality of the ATA pins
   - Reset the ATA module for interface programming
   - Query device for status and capabilities
2. PIO initialization
   - Set PIO timing registers
   - Enable IORDY in ATA host configuration register (for PIO–3 and above)
3. MDMA/UDMA initialization
   - Set either MDMA or UDMA timing registers

4.1 ATA Host Controller Initialization

1. Enable ATA functionality on the external pins. This involves modifying the pin assignment registers (PAR) on various ports on the ColdFire device. Initialization varies amongst specific ColdFire parts. The MCF5445x is used in the example that follows.

```
/*****************************************
* Configure port ATA for primary ATA functionality
* ATAL: ATA_RESET, ATA_DMARQ, ATA_IORDY
* ATAH: ATA_BUFFER_EN, ATA_CS[1:0], ATA_DA[2:0]
*****************************************/
MCF_GPIO_PAR_ATA = 0x07E7;
/*****************************************
* Configure port FEC1 for ATA functionality
* ATA_DATA 5,6,7,8,9,11,13,15 on FEC1H
* ATA_DATA[2:1], ATA_DATA10, ATA_DATA0, ATA_DATA[4:3], ATA_DATA 14,12
*****************************************/
```

```
MCF_GPIO_PAR_FEC = 0x10;
/*******************************************
* configure port FECI2C for ATA functionality
* ATA_DIOR, ATA_DIOW
*******************************************/
MCF_GPIO_PAR_FECI2C = 0x0A00;
/*******************************************
* configure port PCI for ATA functionality
* ATA_DMACK, ATA_INTRQ
*******************************************/
MCF_GPIO_PAR_PCI= 0x8080;
```

2. Execute power-on/hardware reset protocol. Executed immediately after power-on reset (POR) or after a hardware reset. The host interrupt controller setup varies amongst specific ColdFire microprocessors.

```
uint32 i;
/*********************************************
* Enable ATA interrupt at interrupt controller
*********************************************/
MCF_INTC1_IMRH &= (~MCF_INTC_IMRH_INT_MASK54);       //allow interrupts from ATA
MCF_INTC1_ICR54 = 0x7;     //Set level 7
asm_set_ipl(0);            // Set status register IPL to allow all interrupts

MCF_ATA_CR = 0x0;          //assertATAReset
wait_uS(25);               //wait 25uS as specified by ATA/ATAPI-6
MCF_ATA_CR = 0x40;         //deassertATAReset
wait_uS(20000);            //wait 2ms as specified by ATA/ATAPI-6

MCF_ATA_IER |= 0x08;       //Enable ATA interrupt from device
```

3. Query device for capabilities. After power-on/hardware reset the host should issue an IDENTIFY DEVICE and/or IDENTIFY PACKET DEVICE command to determine the current status and features implemented by the device(s). Because the ATA host controller operates in big endian mode, it is necessary to byte swap the device information.

```
uint16 deviceIdInfo[256];          //512 byte sector/buffer

setTimingRegistersForPIOMode(0);   // Set PIO mode 0
MCF_ATA_CR = 0x41;                 // Enable IORDY at Host, needed for PIO-3 and above

while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
DRIVE_DEV_HEAD = 0xE0; //Select Drive 0;
while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
while((DRIVE_ALT_STATUS_CONTROL & 0x40) != 0x40);    //wait for DRDY bit set
DRIVE_COMMAND_STATUS = 0xEC; //Write IDENTIFY_DEVICE command
/*wait for interrupt to signal completion of command device ready for buffer read*/

/*****************************************************
* ATA interrupt handler
*****************************************************/
__interrupt__
void ata_interrupt(void)
{
        uint8 DevStatus;
        uint16 temp;

        DevStatus = DRIVE_COMMAND_STATUS;
/*****************************************************
```

```
* Read Device info
*****************************************************/
        for(i=0;i<256;i++)
        {
                deviceIdInfo[i] = DRIVE_DATA;
                temp = deviceIdInfo[i] >> 8;
                deviceIdInfo[i] = deviceIdInfo[i] << 8;
                deviceIdInfo[i] = deviceIdInfo[i] | temp;
        }
}
```

4.1.1 PIO Initialization

Enabling PIO mode protocol requires the execution of the SET FEATURES command. The DRIVE_SECTOR_COUNT and DRIVE_FEATURES registers are written with subcommand information used by the SET FEATURES command. The ATA interrupt is enabled, allowing the device to signal acceptance of command. The following example initializes device and host for PIO Mode 4.

```
while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
writeDriveRegister(DRIVE_DEV_HEAD, 0xE0);            //Select Drive 0
while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
while((DRIVE_ALT_STATUS_CONTROL & 0x40) != 0x40);    //wait for DRDY bit set
ATA_IER = 0x08;
DRIVE_SECTOR_COUNT = 0x08 | 0x04;                    // IORDY, PIO Mode 4 enabled
DRIVE_FEATURES = 0x03;                               // Set transfer mode according
                                                     to DRIVE_SECTOR_COUNT register
DRIVE_COMMAND_STATUS = 0xEF;                         //Execute SET_FEATURES command

/*wait for interrupt to signal command was executed and accepted*/
setTimingTimingRegistersForPIOMode(4); //
```

4.1.2 UDMA/MDMA Initialization

Enabling UDMA/MDMA Mode protocol requires the execution of the SET FEATURES command as well. The DRIVE_SECTOR_COUNT and DRIVE_FEATURES registers are written with subcommand information used by the SET FEATURES command. The ATA interrupt is enabled, allowing the device to signal acceptance of command. The following example initializes device and host for MDMA mode 2.

```
#define MDMA 0x20
#define UDMA 0x40
#define MODE0 0x0
#define MODE1 0x1
#define MODE2 0x2
#define MODE3 0x3
#define MODE4 0x4
#define MODE5 0x5

while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
DRIVE_DEV_HEAD = 0xE0;                               //Select Drive 0;
while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
while((DRIVE_ALT_STATUS_CONTROL & 0x40) != 0x40);    //wait for DRDY bit set
ATA_IER = 0x08;
/*********************************************************
* Set subcommand in Features register
* Set transfer mode based on value in Sector Count Register
```

```
* Write command code 0xEF SET Features Section 8.46 ATAPI-6 Spec
*************************************************************/
DRIVE_SECTOR_COUNT = MDMA | MODE2;                    //MDMA mode 2
DRIVE_FEATURES = 0x03;
DRIVE_COMMAND_STATUS = 0xEF;
/*wait for interrupt to signal command was executed and accepted*/
/*************************************************************
* use setTimingRegistersForUDMAMode(4);     // Set UDMA mode timing parameters
* to set timing parameters for UDMA mode.
*************************************************************/
setTimingRegistersForMDMAMode(2);           // Set MDMA Mode 2 timing parameters
```

4.2 Data Access

The following examples demonstrate simple read or write transfers using all three transfer protocols. The examples transfer one sector using LBA.

4.2.1 PIO Data Out Protocol

After programming the PIO timing registers and having selected the desired device (device 0 or device 1), the ColdFire ATA controller programs the device to execute the WRITE SECTOR(S) command to write data to the device using the selected PIO mode. The device sets the DRQ bit to signal to the ATA host controller that it is ready for data. The ATA IRQ line is asserted to signal that all data has been transferred.

```
while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
writeDriveRegister(DRIVE_DEV_HEAD, 0xE0);            //Select Drive 0;
while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
while((DRIVE_ALT_STATUS_CONTROL & 0x40) != 0x40);    //wait for DRDY bit set
DRIVE_SECTOR_COUNT = 0x01;                           //number of sectors to write
DRIVE_LBA_LOW = 0x01;
DRIVE_LBA_MID = 0x00;
DRIVE_LBA_HIGH = 0x00;
DRIVE_COMMAND_STATUS = 0x30;                         // WRITE SECTOR(S) command
while((DRIVE_ALT_STATUS_CONTROL & 0x08) != 0x08);    //wait for DRQ bit set
for(i=0;i<256;i++)
        DRIVE_DATA = i;
/*****wait for interrupt to signal completion of data transfer********/
```

4.2.2 PIO Data In Protocol

The PIO data read protocol is analogous to the PIO data out protocol. The microprocessor programs the device to launch READ SECTOR(S) command to read data from the device using the selected PIO mode. The device asserts its IRQ line to signal that it is ready to transfer data to the host.

```
while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
writeDriveRegister(DRIVE_DEV_HEAD, 0xE0);            //Select Drive 0;
while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
while((DRIVE_ALT_STATUS_CONTROL & 0x40) != 0x40);    //wait for DRDY bit set
DRIVE_SECTOR_COUNT = 0x01;                           // number of sectors to be read
DRIVE_LBA_LOW = 0x01;
DRIVE_LBA_LOW = 0x00;
DRIVE_LBA_HIGH = 0x00;
DRIVE_COMMAND_STATUS = 0x20;
/*****wait for interrupt to signal device ready for reading by host*****/
```

```
/*****************************************************
* ATA interrupt handler
*****************************************************/
__interrupt__
void ata_interrupt(void)
{
        uint8 DevStatus;

        DevStatus = DRIVE_COMMAND_STATUS;
        /*******************************************************
        * Read Data
        *******************************************************/
        for(i=0;i<256;i++)
                deviceIdInfo[i] = DRIVE_DATA;      //does not implement byte swap
}
```

4.2.3 MDMA/UDMA Data Read

After programming the respective MDMA or UDMA timing registers, programming the DMA module to accept DMA requests from the device, and having selected the desired device (device 0 or device 1), the ColdFire ATA controller sends the READ DMA command using PIO mode to the device. This causes the device to request a DMA transfer. The device asserts its IRQ line to signal end of transfer. At the end of transfer the CPU or DMA must read any remaining bytes left in the FIFO.

NOTE

There may be less than FIFO_ALARM bytes remaining so transfer is not automatic by the DMA. The FIFO_FILL register reports the number of halfwords remaining in the FIFO.

```
uint16 sectorInfo[256]; //512 byte sector

FIFO_ALARM = 0x20; //Set FIFO alarm to half the size of the 128 byte-buffer
ATA_CR = 0x40; //Reset FIFO
/*********************************************************************
* FIFO normal operation, FIFO empty by DMA, DMA burst, IORDY, MDMA
* if you want UDMA write 0xDD
*********************************************************************/
ATA_CR = 0xD9;
ATA_IER = 0x88;// allow DMA request and interrupt from device
/*********************************************************************
* Setup the eDMA for ATA RX, channel 14
* *******************************************************************/
MCF_EDMA_CR = MCF_EDMA_CR_ERCA;             //Round Robin priority
MCF_EDMA_ERQ = MCF_EDMA_ERQ_ERQ14;          //Allow ATA RX - 14
MCF_EDMA_EEI = MCF_EDMA_EEI_EEI14;          //Error interrupt enable for ATA RX
MCF_EDMA_TCD14_SADDR = MCF_EDMA_TCD_SADDR_SADDR(0x90000018); //Set the source Address
MCF_EDMA_TCD14_ATTR = MCF_EDMA_TCD_ATTR_SSIZE_32BIT | MCF_EDMA_TCD_ATTR_DSIZE_32BIT;
MCF_EDMA_TCD14_SOFF = MCF_EDMA_TCD_SOFF_SOFF(0x0);
MCF_EDMA_TCD14_NBYTES = MCF_EDMA_TCD_NBYTES_NBYTES(0x20);      //32 byte
MCF_EDMA_TCD14_SLAST = 0x0;                                   //No adjustment
MCF_EDMA_TCD14_DADDR = MCF_EDMA_TCD_DADDR_DADDR((uint32)&sectorInfo);
MCF_EDMA_TCD14_CITER_ELINK = 0x0;
MCF_EDMA_TCD14_CITER = 0x10;
MCF_EDMA_TCD14_DOFF = 0x4;
MCF_EDMA_TCD14_DLAST_SGA = 0x0;
```

```
MCF_EDMA_TCD14_BITER = 0x10;
MCF_EDMA_TCD14_BITER_ELINK = 0x0;
MCF_EDMA_TCD14_CSR  = 0x0;

while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
writeDriveRegister(DRIVE_DEV_HEAD, 0xE0);            //Select Drive 0;
while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
while((DRIVE_ALT_STATUS_CONTROL & 0x40) != 0x40);    //wait for DRDY bit set
DRIVE_SECTOR_COUNT = 0x01;                           //sectors to be read
DRIVE_LBA_LOW = 0x1;
DRIVE_LBA_MID = 0x00;
DRIVE_LBA_HIGH = 0x00;
DRIVE_COMMAND_STATUS = 0xC8;
/*****wait for interrupt to signal end of dma transfer*****/
```

4.2.4 MDMA/UDMA Data Write

The DMA write protocol is analogous to the DMA read protocol. The ColdFire ATA controller sends the WRITE DMA command using PIO mode to the device causing the device to request a DMA transfer. The DMA request and interrupt request must be enabled. The device asserts its IRQ line to signal end of transfer.

NOTE

No FIFO manipulation is necessary at the end of transfer.

```
uint16 sectorInfo[256]; //512 byte sector

FIFO_ALARM = 0x20; //Set FIFO alarm to half the size of the 128 byte-buffer
ATA_CR = 0x40; //Reset FIFOuint16 sectorInfo[256]; //512 byte sector
/*********************************************************************
* FIFO normal operation, FIFO empty by DMA, DMA burst, IORDY, MDMA
* if you want UDMA write 0xEE
*********************************************************************/
ATA_CR = 0xEA;
ATA_IER = 0x88;
/*********************************************************************
* Setup the eDMA for ATA TX, channel 15
* *******************************************************************/
MCF_EDMA_CR = MCF_EDMA_CR_ERCA;              //Round Robin priority
MCF_EDMA_ERQ = MCF_EDMA_ERQ_ERQ15;           //Allow ATA TX
MCF_EDMA_EEI = MCF_EDMA_EEI_EEI15;           //Error interrupt enable for ATA TX
MCF_EDMA_TCD15_SADDR =  MCF_EDMA_TCD_SADDR_SADDR((uint32)&sectorInfo);
MCF_EDMA_TCD15_ATTR = MCF_EDMA_TCD_ATTR_SSIZE_32BIT | MCF_EDMA_TCD_ATTR_DSIZE_32BIT;
MCF_EDMA_TCD15_SOFF = MCF_EDMA_TCD_SOFF_SOFF(0x4);            //Size offset 4 bytes
MCF_EDMA_TCD15_NBYTES = MCF_EDMA_TCD_NBYTES_NBYTES(0x20);//32 byte or 8 longwords
MCF_EDMA_TCD15_SLAST = 0x0;
MCF_EDMA_TCD15_DADDR = MCF_EDMA_TCD_DADDR_DADDR(0x90000018);
MCF_EDMA_TCD15_CITER_ELINK = 0x0;
MCF_EDMA_TCD15_CITER = 0x10;
MCF_EDMA_TCD15_DOFF = 0x0;
MCF_EDMA_TCD15_DLAST_SGA = 0x0;
MCF_EDMA_TCD15_BITER = 0x10; //Make sure DMA does not transfer more than sector size
MCF_EDMA_TCD15_BITER_ELINK = 0x0;
MCF_EDMA_TCD15_CSR  = 0x0;
while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
writeDriveRegister(DRIVE_DEV_HEAD, 0xE0);            //Select Drive 0;
```

```
while((DRIVE_ALT_STATUS_CONTROL & 0x80) == 0x80);    //wait for BSY bit clear
while((DRIVE_ALT_STATUS_CONTROL & 0x40) != 0x40);    //wait for DRDY bit set
DRIVE_SECTOR_COUNT = 0x01;                   //number of sectors to be written
DRIVE_LBA_LOW = 0x01;
DRIVE_LBA_MID = 0x00;
DRIVE_LBA_HIGH = 0x00;
DRIVE_COMMAND_STATUS = 0xCA;
/*****wait for interrupt to signal end of dma transfer*****/
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3513
Rev. 0
09/2007

freescale™
semiconductor