FLASH MEMORY CONTROLLER
A. Ramesh, 04/10/97

Abstract
The flash memory controller has been designed for LSI and FPGA implementations separately. Simulations were done for the ata and pcmcia interfaces. *Card Bus interface is not implemented* yet. Synthesis of the FPGA design was done using Compass and the resulting netlist consists of about 9k gates.

The LSI and FPGA implementation designs are very similar except for some minor differences. The LSI design contains the processor ROM inside, while for the FPGA, the ROM is outside and a special interface is present between them. Also, an additional block of logic (glue_1) serves to program the ROM code simultaneous with the programming of the FPGA itself. This logic works on the "Dclk" input to the FPGA sent from the programming device itself.

The various logic blocks that are present in the design are listed below and their functions specified:
1) sysi_1: This is the main interface to the Host system. The Task File registers are present inside this block. At the start of a transaction, the host system writes relevant data to these task file registers with the command register written at last. These registers are implemented as 5 16-bit registers (sys_0, sys_1, sys_2, sys_3, sys_4) to facilitate easy access to the on-chip processor. When the host reads these registers, it reads each of them through the lower 8-bit of the host data bus. An additional 16-bit register (sys_5) is used for communication with the processor. The configuration registers for the PCMCIA interface is also present in this module. But these are not active except for the SRST bit. These registers can only be written from the host.

2) Bufi_1: This module acts as the interface to the SRAM buffer present in the chip. This can be accessed either by the host or the processor using the "sys_sel" input as a switch. The address to the ram is generated using a counter depending on whether the host or the processor is accessing the buffer.

3) Fmi_1: This module acts as the interface to the Flash memory. The processor can transfer data between the flash memory and the sram buffer using 2 16-bit registers present in this module. The register fm_0 contains the data and is connected to the flash memory data bus. So the processor either reads or writes from this register. Register fm_1 contains the CE's, ALE, CLE outputs for the flash and the R/B# input from the flash. Also, this module identifies the type of interface (ATA, PCMCIA, CARDBUS) using fad(28:27) bits of the flash memory data bus at the rising edge of "nreset".

4) Ecc_1: This logic block generates the four ecc check bytes per sector of data that is transferred between the flash memory and the sram data buffer. These 4 check bytes are stored in two registers "cb20_w" and "cb31_w" which are accessed by the processor to process the data read from the flash.

5) Romi_1: This is interface to the processor rom that is stored outside the FPGA. In the case of LSI design, this block (romu_1) contains the actual rom data also. romu_1 and romi_1 are essentially the same except for the presence of rom.

6) Alu_1, selab_1, idec_1: These are the processor hardware logic blocks.

Simulation:

Simulation was done using Voyager for the case of both ATA and PCMCIA interfaces with the host. This was accomplished using a simulation model of the host (ata_pgen, pc_pgen), SRAM buffer (data_buf) and Flash memory (flash2). The simulation model of the host contained a rom using which the various signals at the host interface were specified at different times during the simulation. Also, for simulation purposes, the processor *generated an Irq14* during a write transaction when the processor is ready to accept data from the host (In normal cases, this is done by the host polling the DRQ bit in the status register). During the simulation, the task file registers were written to, read from the host and the processor. Also, one sector (512 bytes) of data was transferred from the host model to the flash memory during a Write transaction, and then read back from the flash to the host in a subsequent Read transaction. In order to check the ECC logic some wrong data was read back from the flash. For this purpose, 2 different named modules for flash models are present (flash2 and flash2_a). They are essentially the same except that we could generate a faulty data in one module while having the correct data in the other one, thus having only one byte (among 512 bytes) which is in error. (The ECC limitation).

Simulation was done for the cases when there was no error during the read cycle, 1 byte error and 2 bytes error during Read. For the cases of no error and 2 errors, the controller performed as expected. However for the case of *one error*, the calculation of the wrong byte address was not correct and so the wrong data was output to the host. *This needs to be corrected. Simulations for multiple sector transactions, read long, write long commands were not performed.*

For the case of simulation for ATA and PCMCIA interfaces, only the ata_pgen, and pc_pgen have to be switched. Two different test benches “Test bench” & “pc_test_bench” were made for the purpose. The differences between the two are:
1) Reset is active high for PCMCIA, while active low for ata interface.
2) Pc_intf needs to set high for PCMCIA interface.
3) fad (28:27) needs to be changed accordingly.
4) The address output of the pc_pgen is 10 bits and so all 10 bits have to be used for decoding.

In the PCMCIA interface test bench, attribute memory was not implemented. So, reading the attribute memory from the host using “noe” was *not simulated.*

Synthesis:

Synthesis was performed using Compass (ASIC synthesizer) and the entire hardware consumes about 9k logic gates. EDIF files were made for the implementation of the logic using an FPGA. It was found that making .pcl files for the various modules are more effective in reducing the gates than synthesizing and optimizing directly from the top level. Each of the inner modules need to be synthesized, optimized and then the topmost level is optimized for least gate count. A batch file was also generated for this purpose (fpga.sh)

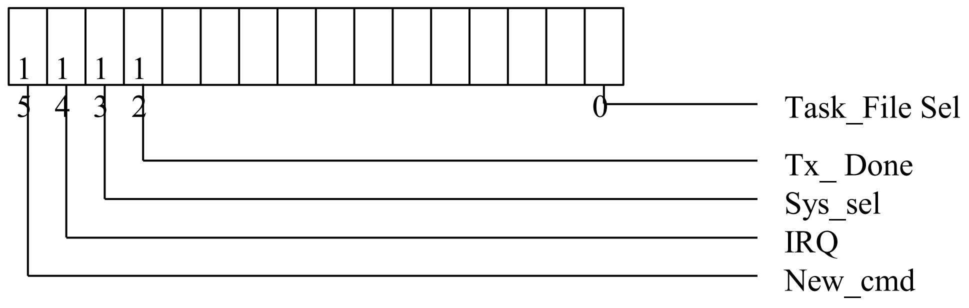
Things Accomplished:
1) Obtained hardware for the Flash controller.
2) Testing of the logic for simple transactions have been successful.

Things to be done:
1) Address calculation of the ECC logic needs to be checked and corrected if needed.
2) Card Bus interface needs to be implemented.
3) Simulations for multiple sector read/write need to be done.

Task File registers:

Registers	**High Byte (15 : 8)**	**Low Byte (7 : 0)**	**Host**	**Proc.**
Sys_0	Sector Number	Sector Count	W	R
Sys_1	Cylinder High	Cylinder Low	W	R
Sys_2	Command	Device / Head	W	R
Sys_3	Features	Device Control	W	R
Sys_4	Status	Error	R	W
Sys_0a	Sect. Num. (updated)	Sect. Cnt. (updated)	R	R/W
Sys_1a	Cyl. High (updated)	Cyl. Low (updated)	R	R/W
Sys_2a	XXXXXXXX	Dev./head (updated)	R	R/W

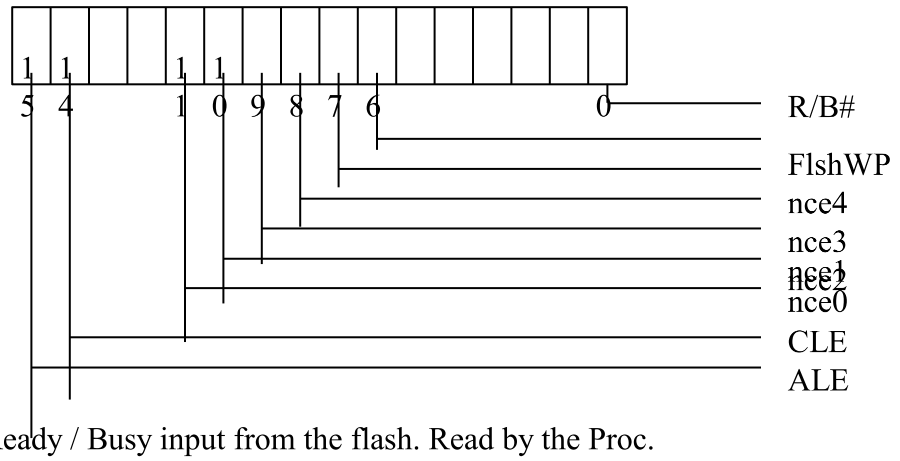
Register Sys_5:



Task_File Sel:	0 : Updated register access (eg. sys_0a) 1 : Original register access (eg. sys_0)
Tx_done:	Set after 256 transactions bet host and buffer. Proc. resets it.

Sys_sel	:	0 : Host access to Buffer 1 : Flash access to Buffer
IRQ:		Interrupt to the host by the proc. Reset by the proc.
New_cmd:		Set every time new cmd is recieved. Reset by the proc.

Register fm_1:



R/B# : Ready / Busy input from the flash. Read by the Proc.

Others : Output to the Flash. Written by the Proc.

Achalu Ramesh joined ACC Micro (Auctor Corporation later on) in October 1996 just before graduating Iowa state university with MSEE in 1997. He took charge of the debug work of flash memory controller LSI I designed handling an UNIX workstation and the design tools, Voyager and Compass (ACC Micro did not have superb Verilog/Synopsis).

We made a breadboard of flash memory controller LSI system implementing an Altera FPGA and Samsung flash memory chips kindly gifted by Samsung as flash memory targetted.

Just before Ramesh started working at Intel Folsom in April 1997, he made this job report.

Downloadable Files