Memories and Control — Some Basics

ENEE 245: Digital Circuits and Systems Laboratory, Fall 2014
Lab 7

Objectives

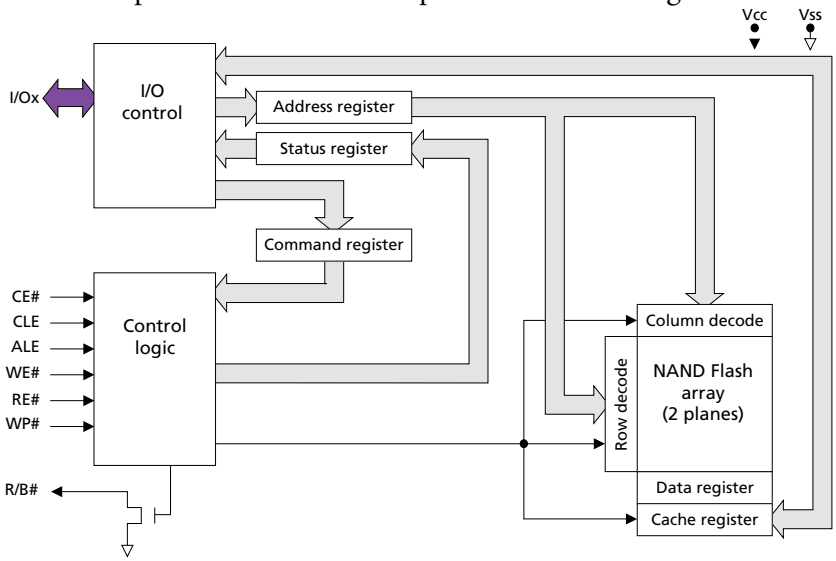
The objectives of this laboratory are the following:

- To learn how to instantiate the block RAMs offered in FPGAs
- To start learning about control of memories

In this lab you will build a simple controller that sends out a series of commands, comprising several timing signals and a combined command/address/data bus. The important things will be keeping track of where in the sequence of operations your controller is, and using an internal data array to convert parallel data to serial data. This array will be implemented by the built-in memory blocks provided in modern FPGAs: the block RAMs.

Flash Command Interface

ONFI-compliant NAND flash chips have the following interface and internals:



The chip exports the following I/O pins to the world (as well as power/ground):

DQ bus	8- or 16-bit I/O, including commands, addresses, and data (only 8-bit bus supported for high-speed DDR interfaces)
CE#	Chip enable, active low
CLE	Command-latch enable
ALE	Address-latch enable
WE#	Write enable, active low
RE#	Read enable, active low
WP#	Write protect, active low
R/B#	Read/Busy (the flash chip's status indicator from internal status register)

The ONFI 4.0 NAND flash interface uses these pins in the following way:

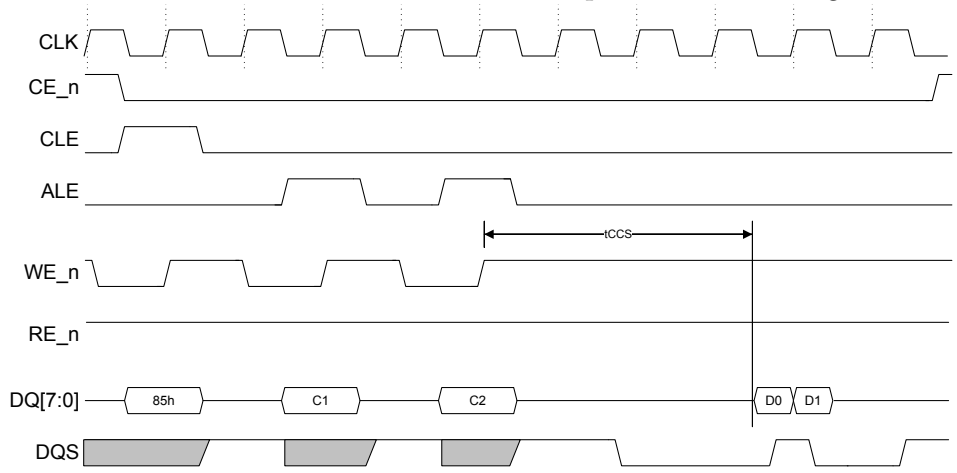


Figure 76 NV-DDR2 and NV-DDR3 data interface command description

This is the latest DDR interface, NV-DDR2/3. The CLK signal at the top is not used in the interface but is shown so you understand that the *controller* might very well use a clock that is faster than the interface between the controller and the flash device. In fact, this is how you should build your controller simply because it is the easiest way to do it.

A handful of commands that you will want to implement in the future:

- Reset command
- Read ID commands
- Read Parameter Page command
- Page Read commands
- Block Erase commands
- Page Write commands (called “Page Program”)

These commands take the following forms. In this lab, you will be emulating the *timing*, but you will be using different *command-code numbers*, because you will use a 2-bit data bus for simplicity.

Reset Command

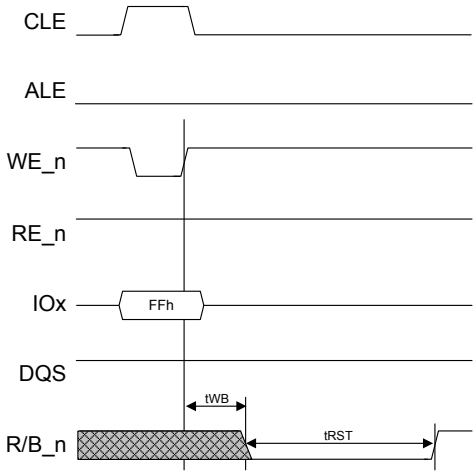


Figure 77 Reset timing diagram

Read ID — Address 0x00

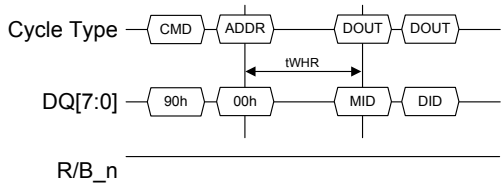


Figure 81 Read ID timing diagram for manufacturer ID

MID Manufacturer ID for manufacturer of the part, assigned by JEDEC.

DID Device ID for the part, assigned by the manufacturer.

Read ID — Address 0x20

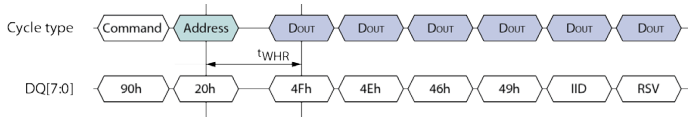


Figure 80 Read ID timing diagram for ONFI signature

IID Power on Interface ID, 01h for NV-DDR3, 00h for SDR

RSV 6th byte is reserved for future use

Read Parameter Page

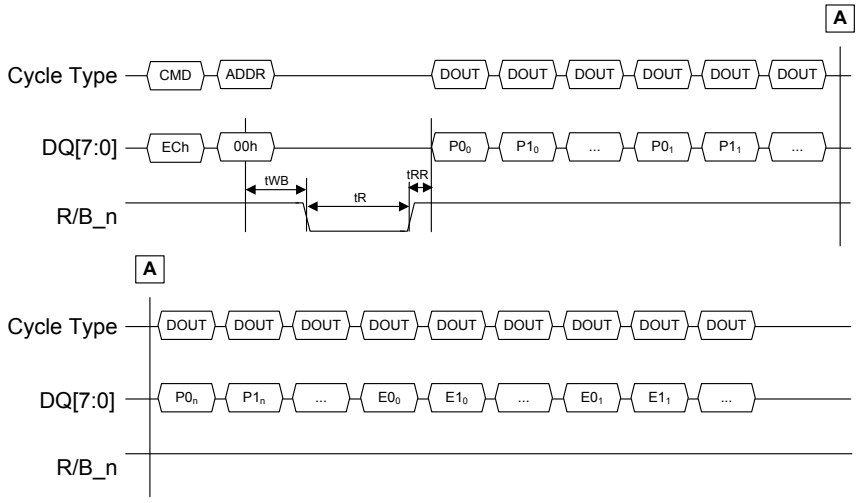


Figure 85 Read Parameter Page command timing

Page Read — Note that *Page Read* has two command cycles, during which the controller sends the values 0x00 and 0x30.

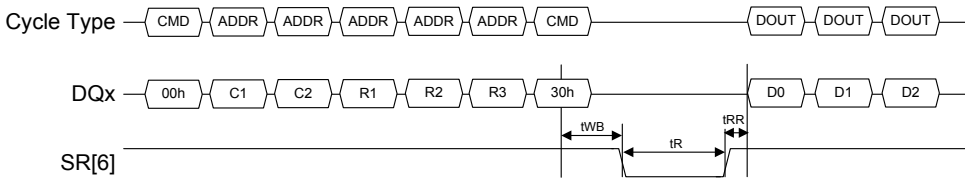


Figure 93 Read timing

C1-C2 Column address of the page to retrieve. C1 is the least significant byte.

R1-R3 Row address of the page to retrieve. R1 is the least significant byte.

Dn Data bytes read from the addressed page.

Block Erase — Note that *Block Erase*, like *Page Read,* has two command cycles, during which the controller sends the values 0x60 and 0xD0. Also note that it only sends a row address, as opposed to a combined row+column address.

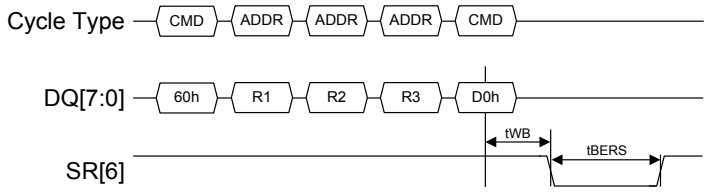


Figure 87 Block Erase timing

Page Program

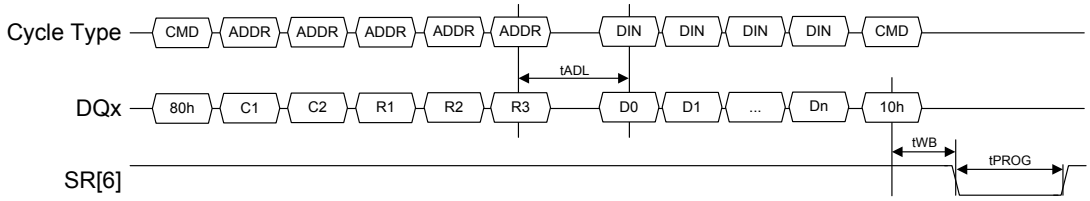


Figure 97 Page Program timing

C1-C2 Column address of the starting buffer location to write data to. C1 is the least significant byte.

R1-R3 Row address of the page being programmed. R1 is the least significant byte.

D0-Dn Data bytes/words to be written to the addressed page.

On the FPGA board there are four buttons and 8 switches. You will use these as follows:

- You will use the 8 switches to identify address values: your controller will read the address from the switches, interpreting switches 0–3 to indicate the column address (the "C1" and "C2" bytes in the various commands, which will each be 2 bits in your design), and switches 4–7 to indicate the row address (the "R1," "R2," and "R3" bytes in the various commands, which will each be 2 bits in your design).
- You will use the buttons to identify commands: your controller will respond to the press of a button by reading the address from the switches and issue the appropriate command signals.
  - ➡ Button 1: 0b01 *Reset* command (command only, no address)
  - ➡ Button 2: 0b10 *Page Read* command (use switches 0–3 to indicate the column address and switches 4–7 to indicate the row address)

  You do not need to implement any other commands at this time.

You will use a 2-bit data bus (as opposed to 8-bit), which means that you will use the 2-bit command codes provided above instead of the real ones in ONFI flash devices, and you will break the row and column addresses into 2-bit chunks and send those chunks over multiple cycles. You will use the Block RAM facility to convert the "parallel" address data, which is read from the 8-bit switch bus at the time of command initiation, to the serial form that is driven out on the command bus (the 8-bit addresses will be sent out as four different 2-bit values).

Block RAM in the FPGA

Your FPGA has quite a few libraries that provide already-created circuits for you to use in your designs. The library documentation is on the course website. For this lab, you will use the Block RAM facility, which is extremely powerful. Because we only want to get the main idea and not necessarily talk to a real flash device, we will just use a narrow bus and a small burst length.

Nonetheless, the general facility that you use and get familiar with as part of this lab is extremely useful and will become an important tool in your future designs.

We will instantiate the RAM32X2S block, which is a 32-entry RAM that is 2 bits wide, running off the positive edge of the clock. In general, your FPGA has numerous block RAMs, each up to 2KB in size, so this really is just a taste of what the FPGA can do. It is instantiated this way:

```
RAM32X2S #(
      .INIT_00(32'h00000000), //INITforbit0ofRAM
      .INIT_01(32'h00000000)  //INITforbi1tofRAM
) instance_name (
      .O0(O0),          // RAM data[0] output
      .O1(O1),          // RAM data[1] output
      .A0(A0),          // RAM address[0] input
      .A1(A1),          // RAM address[1] input
      .A2(A2),          // RAM address[2] input
      .A3(A3),          // RAM address[3] input
      .A4(A4),          // RAM address[4] input
      .D0(D0),          // RAM data[0] input
      .D1(D1),          // RAM data[1] input
      .WCLK(WCLK),      // Write clock input
      .WE(WE)           // Write enable input
);
```

If you put this into your code, giving it a unique instance name (i.e., replace "instance_name" with whatever you want to call it), then the synthesis tools will grab one of these from the Xilinx library and instantiate it onto your FPGA.

The format is that the top parenthetical (preceded by a # sign) is optional and, if present, represents the initialization values; the bottom parenthetical is not optional and represents the wire connections to the module. It has a 5-bit address input (for 32 different locations), and the ability to both read and write simultaneously over a 2-bit input port (D0 and D1) and a 2-bit output port (O0 and O1). For instance, this would make it possible to build a 1-in/1-out register file. In this lab, we will only use the output port, and we will initialize the data to a random value, whatever you want.

For your controller, you will instantiate one of these and operate over a 2-bit data bus instead of an 8-bit data bus. This will reduce the number of wires that you will have to deal with when hooking your FPGA up to the DLA. You only need to implement the following six signals:

DQ bus	2-bit I/O, including commands, addresses, and data
CE#	Chip enable, active low
CLE	Command-latch enable
ALE	Address-latch enable
WE#	Write enable, active low
RE#	Read enable, active low

These should be mapped as follows in the User Constraints File:

```
NET "ceb"     LOC = "B4"; # Bank = 0, Pin name = IO_L24N_0, Type = I/O, Sch name = R-IO1
NET "cel"     LOC = "A4"; # Bank = 0, Pin name = IO_L24P_0, Type = I/O, Sch name = R-IO2
NET "ale"     LOC = "C3"; # Bank = 0, Pin name = IO_L25P_0, Type = I/O, Sch name = R-IO3
NET "web"     LOC = "C4"; # Bank = 0, Pin name = IO, Type = I/O, Sch name = R-IO4
```

```
NET “dq[0]"   LOC = “B6"; # Bank = 0, Pin name = IO_L20P_0, Type = I/O, Sch name = R-IO5
NET "dq[1]"   LOC = “D5"; # Bank = 0, Pin name = IO_L23N_0/VREF_0, Type = VREF, Sch name = R-
IO6
```

When one of the buttons is pressed, read the address value from the switches into the Block RAM as separate 2-bit values. This will make it simpler at the later point to send the addresses out 2 bits at a time. Pad the top of the ROW ADDRESS with a “00” value.

Pre-Lab Preparation

Design your controller. Your goal is to create timing diagrams like the ones above. Use the Xilinx design facility to generate waveforms for each command, and bring these as your pre-lab write-up.

In-Lab Procedure

Bring flash drives to store your data.

Ask the TA questions regarding any procedures about which you are uncertain.

Complete the following tasks:

- **Program your system onto the FPGA board.**
- Connect the FPGA to the break-out board.
- Run the system and use the DLA to observe the 6 bits of the controller output. Save the DLA’s output for your post-lab report.
- Look at the detailed RTL schematics produced by the software; save it for your post-lab report.
- Look at the timing report that gives the pin-to-pin delays for input/output combinations of every pin. Save these reports and tables for your post-lab report.

Post-Lab Report

Write up your code, schematics, and lab procedures. Demonstrate the correctness of your designs through your DLA output and note any differences between what you simulated and how the circuits behaved in the lab.

Regarding the RTL schematics produced by the software—how did the design software synthesize your code? Where did it choose poorly, and how could it have done better? Could you have better specified your design to get more efficient results?

Regarding the timing report that gives the pin-to-pin delays for input/output combinations of every pin—what does the design software say for the timing? How fast is each component? How fast could you, in theory, run your design?