San Jose State University
Department of Electrical Engineering
EE178, Fall 2007, Crabill

Laboratory Assignment #4

Objectives

The objectives of this lab are to design an FSM to control the data path of a speech synthesis circuit, and then modify the system software to say your name. Similar to the previous assignment, the entire hardware source (except the FSM) is provided. You will need to design the FSM, describe it, and then add timing constraints. Only after the hardware is correct will you modify the software. Figure 1 shows a block diagram of the system. The omnipresent clock and reset signals have been omitted.

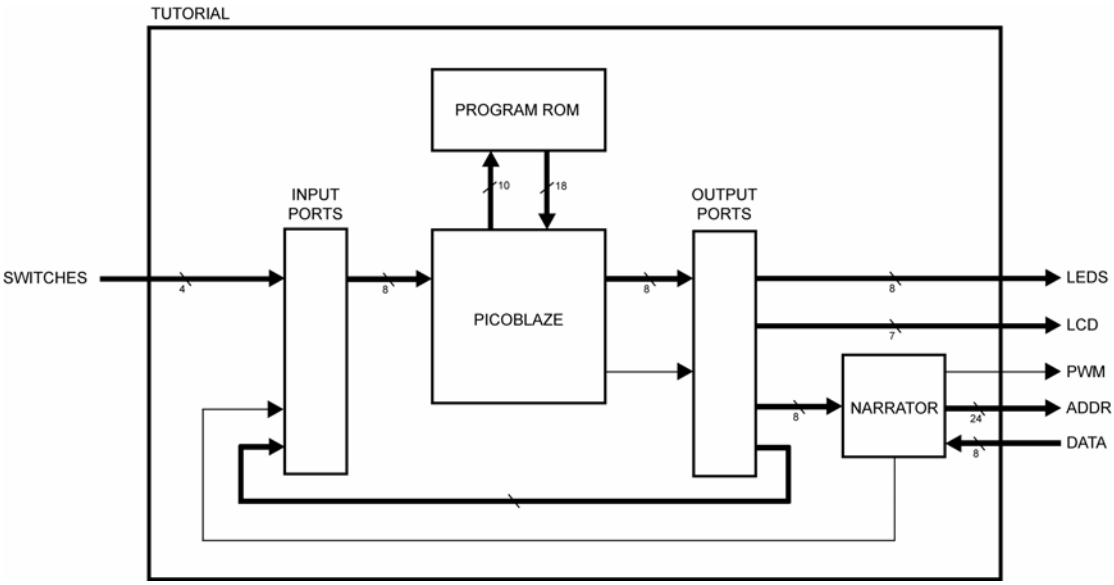

Figure 1: System Block Diagram

You will immediately notice that this system is derived from the previous assignment. The only significant difference is that an additional module, named "narrator" has been instantiated and the software has been modified. The PicoBlaze processor communicates with "narrator" through a single output port (for data) and a single input port (for status). The software executing on the processor can manipulate the "narrator" much faster than humanly possible using switches, buttons, and LEDs.

When you successfully complete this lab, you will have gained experience implementing an FSM. You will also have an appreciation for an ancient speech synthesis technique.

Bibliography

This lab would not be possible without the information and data files provided on the PICTalker project page, published at http://home.alphalink.com.au/~derekw/pictalker/main.htm by Dr. Derek Weston. I would like to thank Dr. Derek Weston and also acknowledge the original source of the data, which is the SPO256 device by General Instruments.

This lab uses the Verilog-HDL version of PicoBlaze; you can read about PicoBlaze on the Xilinx website at http://www.xilinx.com/picoblaze. Additionally, the hardware system and some software routines used in this assignment are derived from PicoBlaze examples for the Spartan-3E Starter Kit board, also obtained from the Xilinx website. In addition, this lab uses *Xilinx Application Note 154, Virtex Synthesizable Delta-*

Sigma DAC. The files you need to complete the lab assignment are posted on the class website; you do not need to download anything from Xilinx.

Initial Preparation

Obtain and unzip the file archive from the class website and use the resulting directory as your project folder. You can move the directory anywhere you please as long as there are no spaces in the path to the directory. You will notice this archive has more source files than the previous project. You should also notice a new subdirectory named "s3esk_nor_flash".

Inside the "s3esk_nor_flash" directory are the files you need to program the parallel flash device on your Spartan-3E Starter Kit with the speech data provided in file phonemes.mcs. This step only needs to be done once, and only takes a few minutes. First, read the provided flash programming document for an overview of the process – the provided document assumes you want to put an FPGA programming file into the parallel flash and then set up the board to have the FPGA configure from that device. In this lab, you will actually be programming data into the parallel flash. Do not change the configuration mode of your board; simply program the flash with your data file and you are done.

Before you attempt to program the parallel flash device on your board, you will need to obtain a standard DB9F to DB9M serial cable. A "standard" cable is a pass-through cable, not a loop back cable or a null modem cable. If you don't have the correct cable, borrow one from the instructor during office hours.

Connect your computer to the Spartan-3E Starter Kit using the serial cable and the USB cable, turn on the board power, and launch HyperTerminal by double clicking on the provided session file. Double click on the batch file, which programs the FPGA with a design that enables the transfer of information from the serial port to the parallel flash. Once the FPGA is configured, you should see activity in HyperTerminal. As shown in the flash programming document, use HyperTerminal to first erase the parallel flash, and then program it with the phonemes.mcs data file.

You will also find a datasheet for the original SPO256 speech synthesis processor in the "s3esk_nor_flash" directory. You will need to refer to this when you are ready to modify the program to say your name.

System Description and Requirements

In this system, you are not allowed to use latches. You are allowed to use only one clock and only one asynchronous reset signal. The clock must be the 50 MHz clock signal available from the oscillator on the Spartan-3E Starter Kit board. **You will receive zero points if you do not follow these requirements.**

The following input signals are carried over from the previous assignment:

clk	clock signal, 50 MHz from oscillator
rst	reset signal
switches[3:0]	4-bit switch input

The following output signals are carried over from the previous assignment:

leds[7:0]	8-bit LED output
lcd_db[7:4]	4-bit LCD data bus
lcd_e	LCD enable strobe
lcd_rs	LCD register/ram select
lcd_rw	LCD read/write select

As shown in Figure 1, there are three new signals of significance. These are the PWM audio output, a 24-bit output which provides address information to the parallel flash, and an 8-bit input which receives data from the parallel flash. If you look at the source code, you will also notice a handful of other signals not

shown here. These signals are tied to constants and are present in order to disable other components on the Spartan-3E Starter Kit which may conflict with the operation of the parallel flash device.

You must design your own FSM to control the "narrator" data path, but otherwise implement the system from the provided files. Do not modify them. Additionally, you must modify the provided software to say your name.

System Hardware Project

In the subdirectory for the assembler, there is a test program, software.psm, and a batch file to assemble it. Assemble the program by double clicking on the batch file. The assembler generates an additional source file, software.v. This source file contains a BlockROM initialized with executable code generated by the assembler from software.psm.

Next, create a hardware system using Project Navigator. To do this, you need to create a project and then "add existing source" files:

1. tutorial.v (top level system module)
2. testbench.v (testbench for behavioral simulation of top level system module)
3. kcpsm3.v (the PicoBlaze sub-module)
4. software.v (the PicoBlaze software sub-module)
5. narrator.v (speech synthesis sub-module)
6. dac.v (data converter sub-module)
7. fsm.v (sub-module requiring your expertise)
8. tutorial.ucf (constraint file for the design)

Using Project Navigator, add all the source files to the project in the order listed above. As you add the files, you will notice that Project Navigator actually reads the source and attempts to show you what modules it thinks are missing by marking them with a question mark in the Sources window. When you are done with this step, no question marks should remain. Do not forget to add the constraint file.

FSM Design Considerations

One common digital design technique is to organize a circuit into two sections: a data path section and a control section. The data path section accepts control signals that tell it what to do, and generates status signals that indicate status, conditions, or events. The control section (typically an FSM) is responsible for generating the control signals – and if status signals exist, they are used by the FSM to make decisions. A good analogy for this design technique is shown in Figure 2.

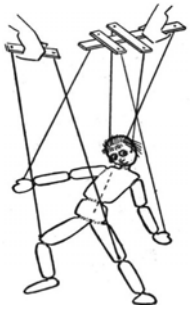

Figure 2: Your turn as John Malkovich

Before you begin writing any code, you must sit down with scratch paper and draw a state diagram of a finite state machine that will satisfy the design requirements. Once you have a possible solution, write a description of it in Verilog-HDL and proceed to test it in simulation.

In order to have any chance of success with the FSM design, it is critical to understand the data path. In this case, it has already been designed for you and is shown in Figure 3. A discussion of each element follows the figure.

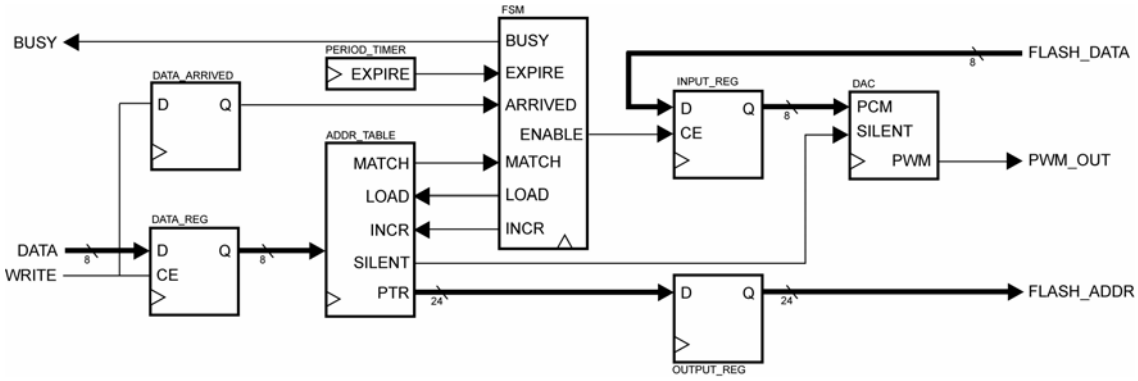

Figure 3: Narrator Data Path

Interval Timer

The data path timer is a simple counter that tracks clock cycles. The purpose of this function is to count the time elapsed between data samples provided to the data converter. In other words, this timer sets the sample rate of the output waveform. The output is asserted for one clock cycle when the timer reaches its terminal count; the terminal count also resets the timer and the timer begins counting a new interval.

Data Register and Strobe

The data register is essentially the implementation of an 8-bit output port for the PicoBlaze processor. In addition, the write enable signal is used to create a “data arrived” strobe which is asserted for one clock cycle after data has been captured by the data register.

Address Table and Pointers

The data written into the data register specifies a speech sound to be played by the circuit. The actual sound information is stored externally, in the parallel flash, as sampled audio (essentially the contents of a .wav file). The address table contains two 24-bit registers and a table of start and end addresses for each of the 64 different speech sounds stored in the parallel flash. The table has an additional bit to force certain sounds to be silent (this is used to implement verbal pauses of several different durations).

The first register is used to hold the start/current address. This register is loadable and can also be incremented. The second register is used to hold the end address. This register is only loadable. When this block is instructed to load new values, both the start/current and end address registers are updated based on the value in the data register. When instructed to increment, the component increments the register holding the start/current value. The logic is constantly comparing the two address registers, and when the start address has been incremented to reach the end address, the module indicates that an address match has occurred.

Output Register

The output register is simply a 24-bit register for the outbound address to the parallel flash. The purpose of this register is not immediately apparent. You might ask, "Why does it even need to exist at all?" It exists not for the sake of functionality, but for the sake of improved and repeatable pin-to-pin I/O timing.

One way to achieve a small clock-to-out in a Xilinx FPGA is to take advantage of the I/O output flip flops. These flip flops require direct connection between the flip flop output and the output driver in the I/O block. The address register therefore cannot be placed in I/O output flip flops because a loadable counter needs to feed back its current value to compute the next state. Therefore, the address register is followed by another register which can be placed into the I/O output flip flops.

Assume the pin-to-pin clock to out is 10 ns.

Parallel Flash

The parallel flash is not explicitly shown in Figure 3. For our purposes, consider the parallel flash to be a large ROM, which is a combinational circuit. The input is an address, the output is data. Figure 4 shows a read waveform reprinted from the Intel device datasheet.

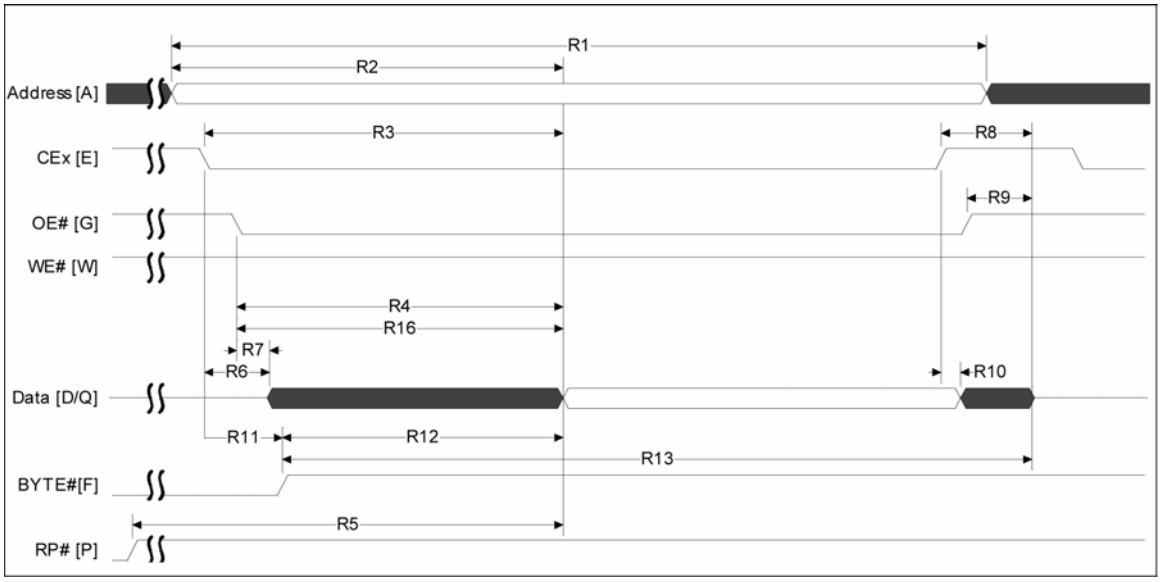

Figure 4: Flash Read Cycle Timing

The "interesting" information in Figure 4 is the minimum read cycle time (R1) and the maximum address to output delay (R2). The data sheet indicates both of these parameters are 75 ns. This is important because it means that the data path must actually wait several cycles of the 50 MHz clock for the parallel flash to return valid data.

Input Register

The input register is simply an 8-bit register with clock enable for the inbound data from the parallel flash. The purpose of this register is not immediately apparent. You might ask, "Why does it even need to exist at all?" This register exists for the sake of functionality, and also for the sake of improved and repeatable pin-to-pin I/O timing.

One way to achieve a small input setup in a Xilinx FPGA is to take advantage of the I/O input flip flops. These flip flops require direct connection between the input buffer in the I/O block and the flip flop input.

The flip flops in the data converter may not be placed in I/O input flip flops because there is combinational logic on the signal path. Therefore, the data converter is preceded by another register which can be placed into the I/O input flip flops.

In most cases, it is not necessary to use clock enable controls on the I/O input flip flops. However, because the parallel flash is relatively slow, this design must actually wait several cycles for the data to become valid after an address change. The idea is to only enable the I/O input flip flops to sample the data when it is known to be valid. Otherwise, your circuit will be processing garbage data some of the time.

Assume the pin-to-pin input setup is 10 ns, and the input hold is 0 ns.

Data Converter

The data converter is a single-bit pulse width modulation circuit derived from a Xilinx application note. The converter is provided with an 8-bit signed binary number and generates a pulse train which (when properly filtered) reaches a voltage proportional to the input value.

Control Module (FSM)

The control module will contain a finite state machine of your own design. This finite state machine may be either Mealy or Moore; it doesn't matter as long as it satisfies the project requirement. However, if you find you need more than 16 states, you may want to rethink your design. The interface between the control module and the data path consists of these signals:

busy	FSM output: indicates the FSM is not idle.
load	FSM output: indicates the address table should load new information.
incr	FSM output: indicates the address table should increment the pointer.
enable	FSM output: indicates the input register should capture data from parallel flash.
expire	FSM input: indicates the timer has counted a full sample period.
arrived	FSM input: indicates new data has arrived from the processor.
match	FSM input: indicates the pointer has reached its final address.

Behaviorally speaking, the FSM must sit idle until data arrives. After data arrives, the FSM must command the address table to load the address pointer and ending address register based on the newly arrived data. At this point, the FSM is ready to fetch data from the parallel flash; it should do so every time the timer expires, and then increment the address pointer. After some number of data fetches, the address table will indicate the current address matches the end address. This should cause the FSM to stop fetching data and return to the idle state. Any time the FSM is not idle, it should indicate that it is busy.

Don't forget to consider the performance of the parallel flash interface when designing the FSM. The total delay from address output to data input should be budgeted as 10 ns + 75 ns + 10 ns = 95 ns. That is five clock cycles of a 50 MHz clock. So, when you load or increment the address value, you cannot enable the input registers to sample the returned data unless *at least* that much time has elapsed.

System Functional Simulation

After you have designed the FSM, you should perform some minimal functional simulation of the system. This is important for two reasons. First, it will give you confidence your system is working properly before you implement it. Second, if the system does not behave as expected when you download it, you will have a mechanism to quickly create additional test cases to help debug the problem. The instructor will not help you debug software problems (incorrect system behavior) unless you are able to run a simulation. A simple

test bench is included with the downloadable support package for this lab. Feel free to enhance the basic test bench as you see fit.

If you didn't already observe this in the previous assignment, you will find that you need to simulate for a very long time in order to capture activity that takes place. This is because the program initially sets up the LCD display. For debugging, you can use a modified program that skips the LCD display initialization (either comment it out, or add a JUMP instruction to bypass it).

Another approach to speed up simulation is to craft an additional test bench, associated directly with a sub-module, and simulate the sub-module directly. For instance, you could simulate your FSM alone, or possibly simulate the "narrator" alone. How much time you spend in verification and what you elect to verify are entirely your choices.

System Synthesis and Implementation

When you are ready to try the design in hardware, synthesize the system as you have done before. Do not forget to check the synthesis report. You should expect to see the PicoBlaze module generate a fair number of warnings about "simulation mismatch" which may be safely ignored.

For this design, you are required to add timing constraints for static timing analysis. After selecting the constraint file in Project Navigator, use the "Create Timing Constraints" process to launch the constraints editor. Enter the constraints as shown in Figure 5.

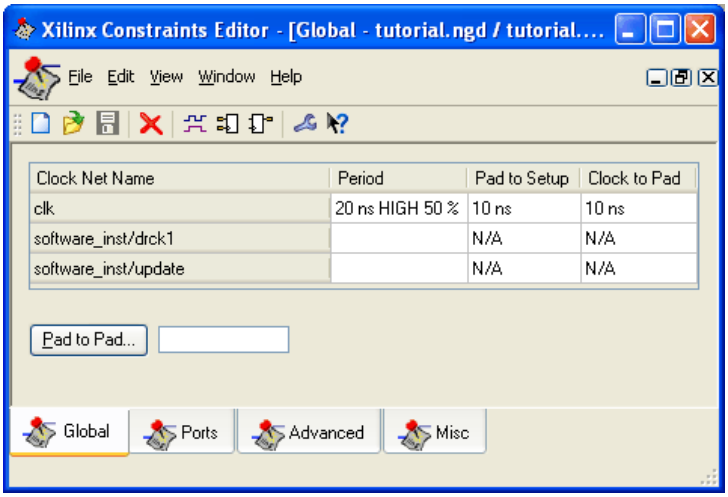

Figure 5: Timing Constraint Entry

These constraints tell the implementation tools that your design must run at 50 MHz and that the input and output timing is bounded. The specific values used for the input and output timing were selected to be realistic but not aggressive – and are based on the assumptions presented earlier where the data path elements were discussed. If your design fails to meet the timing requirements, consult the instructor for advice.

You may be curious why there are three clock signals shown in Figure 5. The extra two signals are contained in the software ROM code and clock a circuit that we are not using. They may be ignored for now. If you have more than what is shown, something is wrong – consult the instructor.

Once you have saved the constraint file, implement the system to make sure no errors occur. Generate a programming file and try the design in hardware. Figure 6 shows where to attach your audio output device. It is the same connection point you used for the speaker in a previous assignment.

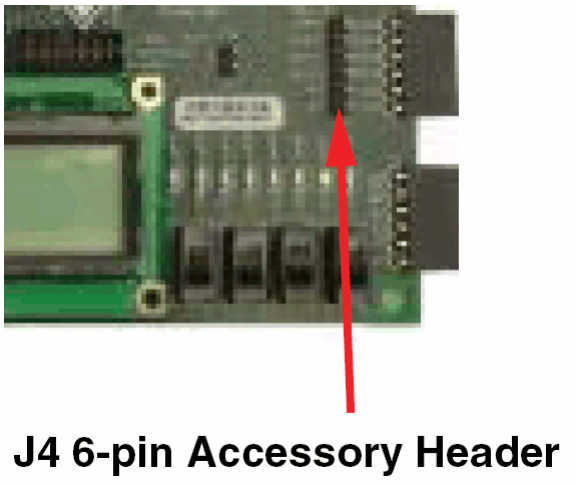

Figure 6: Speaker Connection Location

When your design begins to run, you should hear it say "EE178". To hear the phrase again, press the rotary knob to assert the system reset signal. This will cause the processor to restart execution of the program at the first instruction and say the phrase again.

System Software Development

Now you must modify the PicoBlaze assembly program to say your name instead of the test phrase. Review the SPO256 data sheet to understand how intelligible speech is crafted from strings of smaller speech units. The test program provides an example of how to program the design to utter a string of these smaller speech units.

Modify the program to say your name. Unless you have a very simple name, this process will involve trial and error. Don't forget to run the assembler and re-implement the design each time you modify the program – otherwise, the updated software code will not be reflected in your hardware project. In a later lab, we will learn how to try out code modifications without having to re-implement the design.

System Hardware Verification

Generate a programming file and proceed to verify the system in hardware. The "verification" of your work is somewhat subjective. Does it sound like the hardware is saying your name?

When you have completed the software modifications, generate a programming file for the PROM and then load your system into the PROM.

Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. You are required to demonstrate a working design which has been programmed into the PROM. Within nine hours of your demonstration, you are required to submit your entire project directory in the form of a compressed ZIP archive. Use WinZIP to archive the entire project directory, and name the archive l4_yourlastname.zip. That is "l4" as in Lab 4, **not** "14" as in fourteen. For example, if I were to make a submission, it would be l4_crabill.zip. Then email the archive to the instructor. Only WinZIP archives will be accepted. If your archive is too large, you may remove the subdirectories in the project folder.

Demonstrations must be made on the due date. If your circuit is not completely functional by the due date, you should demonstrate and submit what you have to receive partial credit.