San Jose State University
Department of Electrical Engineering
EE178, Spring 2019, Crabill

Laboratory Assignment #6

Objectives

The objectives of this lab are to design an FSM to control the datapath of a speech synthesis circuit. The completed design will output an audio signal to headphones using the audio jack present on the Real Digital Blackboard. Most of the source files are provided. When you successfully complete this lab, you will have developed a piece of intellectual property that you might be able to reuse in the future.

One common digital design technique is to organize a circuit into two sections: a datapath section and a control section. The datapath 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. An analogy for this design technique is shown in Figure 1.

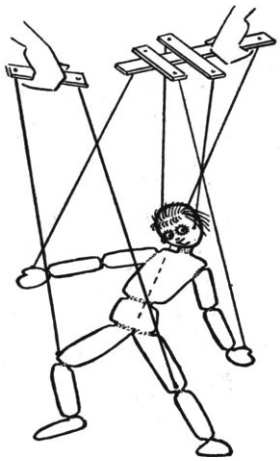

Figure 1: Datapath and Control

Bibliography

This lab draws heavily from documents on the Real Digital website https://www.realdigital.org. I would like to thank Real Digital for making this material available.

This lab would not be possible without the information and data provided on the PICTalker project page (currently off-line) 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. You will find a datasheet for the original SPO256 speech synthesis processor among the files provided for this assignment.

In addition, this lab uses *Xilinx Application Note 154, Virtex Synthesizable Delta-Sigma DAC*. You will find a derivative of this design among the files provided for this assignment.

Design Description and Requirements

In this design, you are not allowed to use latches. You are allowed to use only one clock. The clock must be the 100 MHz clock signal available from the oscillator on the board. You will receive zero points if you do not follow these requirements.

The module has a single clock input. There is one output signal, which is intended to be connected to the on-board audio amplifier.

clk	clock signal, 100 MHz from oscillator
speaker	audio output

The module must drive the speaker output to generate an analog signal as specified in the next section. Create a project using the following files provided for this assignment. Instead of creating source files and copying text, you may consider using the add source capability to directly add files to the project. When adding files, you have the option to copy the files into the project directory – which is advisable, so that the files do not get separate from the project directory.

1. testbench.v	top level testbench for simulation
2. narrator.v	top level design containing a hardware test circuit
3. narrator.xdc	top level design constraint file
4. chatter.v	speech synthesis sub-module
5. dac.v	data converter sub-module
6. fsm.v	sub-module in need of your expertise

FSM Design Considerations

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, code a description of it and proceed to test it in simulation.

In order to succeed with the FSM design, it is critical to understand the datapath. In this case, it has already been designed for you and is shown in Figure 2. A discussion of each element follows the figure.

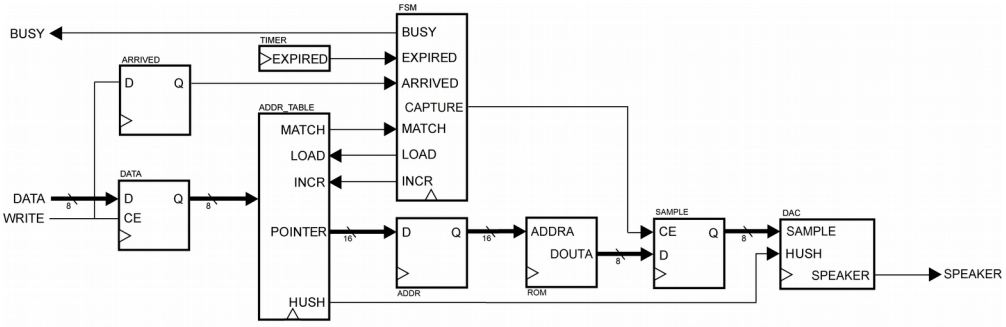

Figure 2: Chatter Datapath

Timer

The datapath 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 and Arrived

The data register exists to hold speech sound selections that are written to the chatter datapath. The write enable signal is used to create a data "arrived" signal 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 in a ROM as sampled audio. The address table contains two 16-bit registers and a table of start and end addresses for each of the 64 different speech sounds stored in the ROM. 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 block 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.

Address

The address register is a 16-bit register for the address to the ROM, and acts as a pipeline stage.

Read Only Memory

The ROM is 64Kx8 and stores 8-bit sampled audio. You will need to create this ROM using the Vivado IP Catalog. Open the IP Catalog and find the Block Memory Generator, under the Memories & Storage Elements section, as shown in Figure 3.

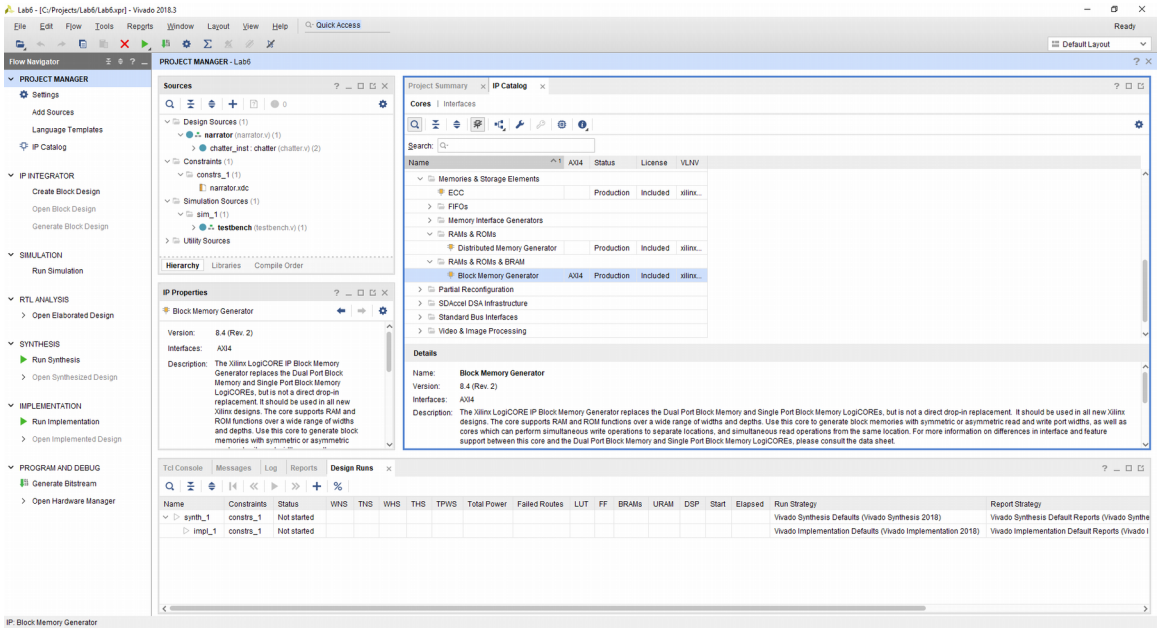

Figure 3: Block Memory Generator in Vivado IP Catalog

Refer to Figure 4, Figure 5, Figure 6, and Figure 7 – take your time, be accurate, and only click OK after you have reviewed all tabs.

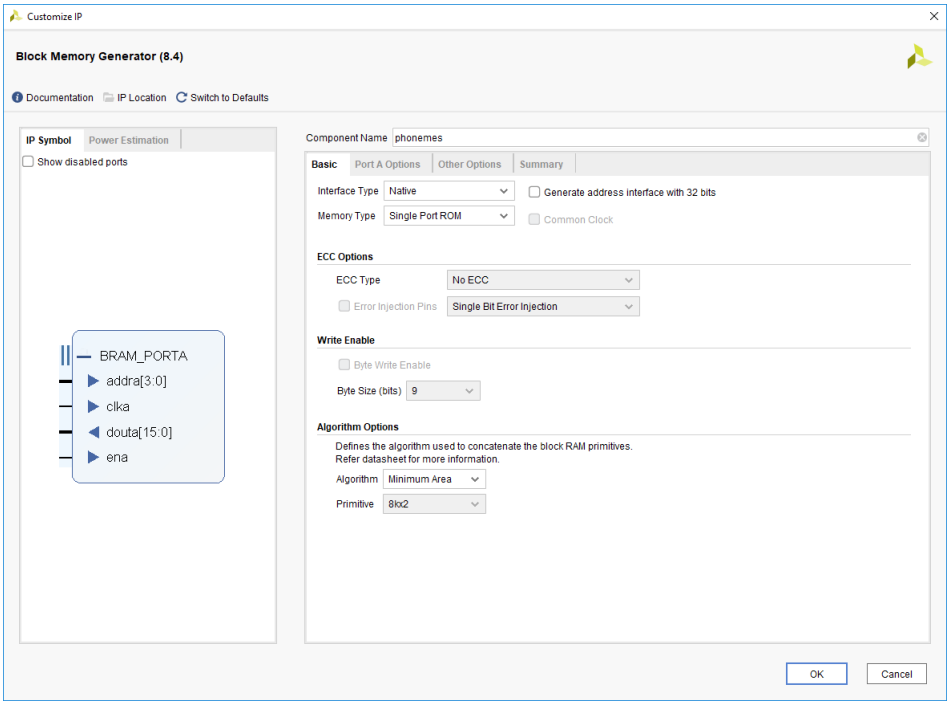

Figure 4: Block Memory Generator, Basic Tab

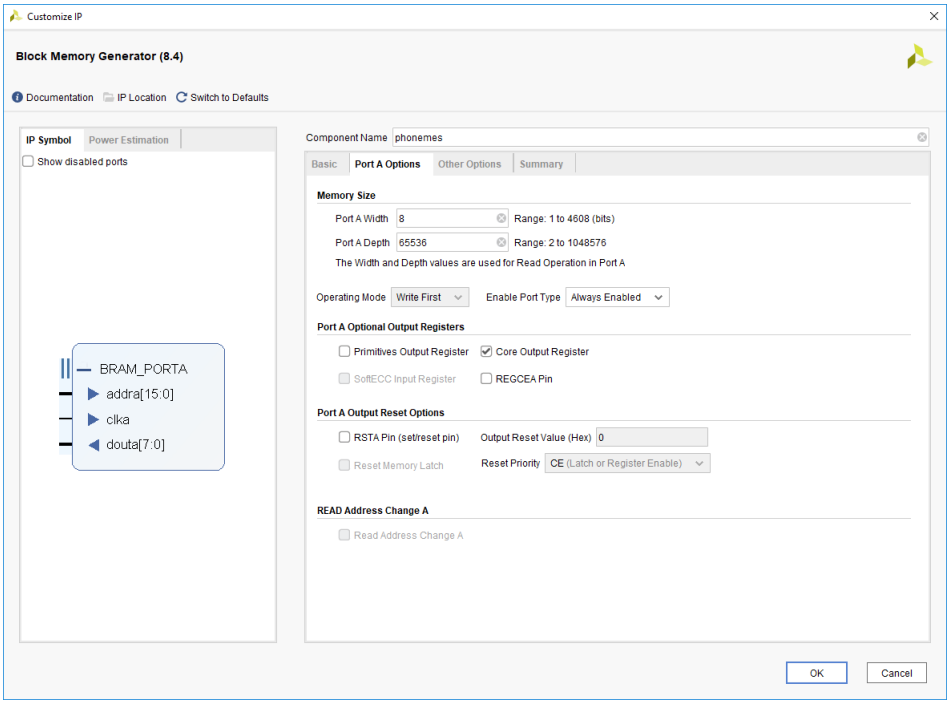

Figure 5: Block Memory Generator, Port A Options

The most significant of the other options shown in Figure 6 is the specification of a coefficient file which contains the values to be stored in the ROM. You will find a phonemes.coe file among the files provided for this assignment.

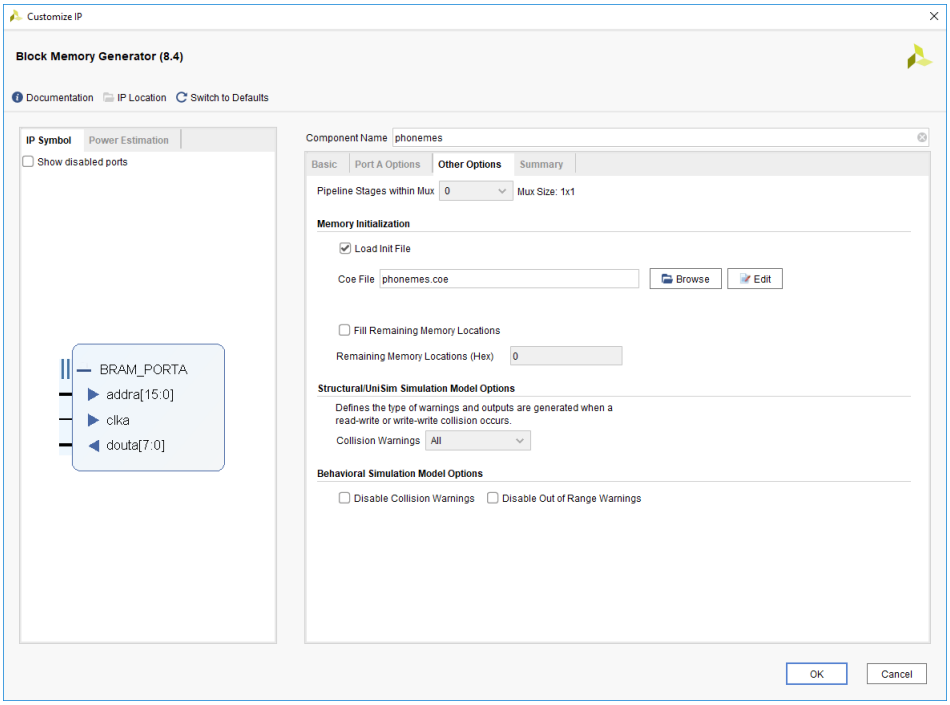

Figure 6: Block Memory Generator, Other Options

In Figure 7, notice the ROM read latency in the summary. Based on the settings in the previous figures, the ROM read latency should be 2 clock cycles.

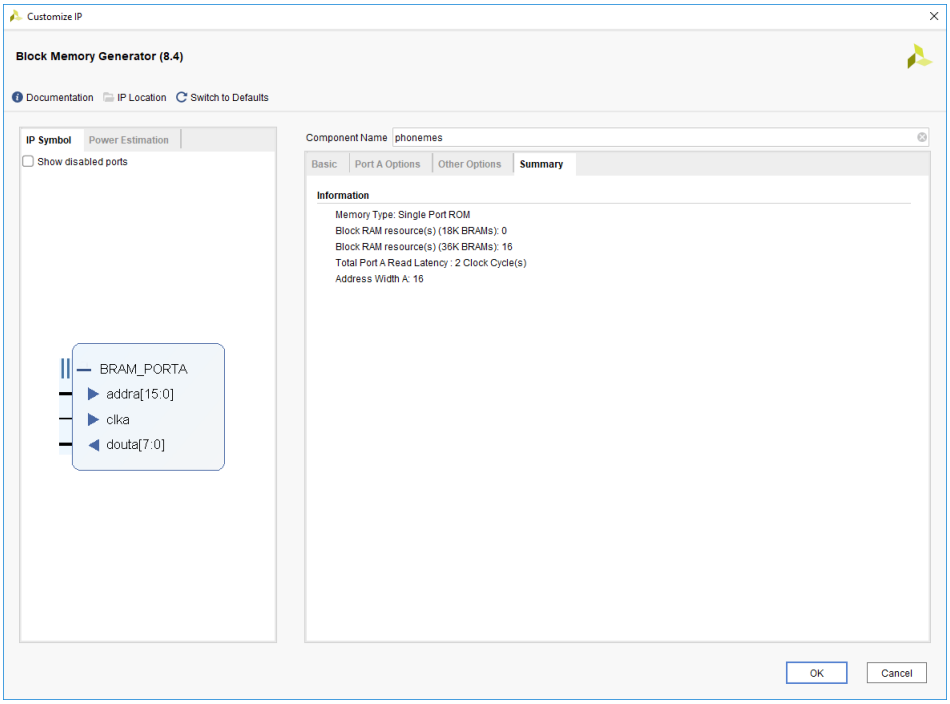

Figure 7: Block Memory Generator in Vivado IP Catalog

Clicking OK begins output products generation, which you have experienced in a previous lab. Accept the defaults for the output products generation. If Vivado asks to create a directory for the output products, grant it permission.

As noted earlier, the ROM read latency should be 2 clock cycles. Figure 8 illustrates the relationship between address inputs to the ROM and the data outputs from the ROM. Address "1" corresponds to data "A", address "2" corresponds to data "B", and so on…

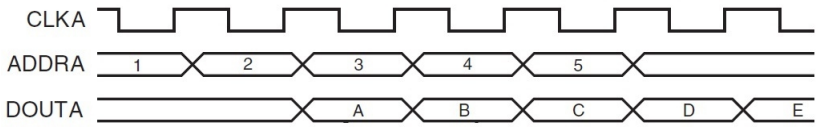

Figure 8: Block Memory Generator in Vivado IP Catalog

Sample

The sample register is an 8-bit register for the data from the ROM, and acts as a pipeline stage.

Data Converter

The data converter is a single-bit pulse width modulation circuit. It is provided an 8-bit signed binary value and generates a pulse train which (when filtered) reaches a voltage proportional to the input value.

Finite State Machine

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 finite state machine and the datapath 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
capture	FSM output: indicates the sample register should capture data from ROM
expired	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

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 based on the data. At this point, the FSM is ready to fetch data from the ROM; it should do so every time the timer expires, and then increment the address pointer. After some number of data fetches, the current address will match the end address. This should cause the FSM to stop fetching data and return to idle. Anytime the FSM is not idle, it should indicate busy.

Don't forget to consider the ROM read latency when designing the FSM. When you load or increment the address value, you cannot enable the capture register until you know the ROM data is ready.

Testing the Design

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 problems unless you are able to run a simulation. A simple test bench is provided. Feel free to enhance the basic test bench as you see fit.

Synthesizing and Implementing the Design

Synthesize and implement your design exactly as you did in the tutorial. Do not forget to check the reports. As a general practice, you will want to review all errors and warnings. These point to areas of concern that you should either address or justify. If the design fails one or more timing specifications the reports will indicate this is the case.

Test your design in hardware, you should hear it repeat "EE178" in a nasal, robotic voice. Does the circuit behave as you expect? If it does not, seek assistance.

WARNING! The audio is extremely loud. Do not wear the headphones, but instead bring them only as close to your ears as necessary to hear the output of the circuit. With stereo headphones, the audio output will be on one channel only.

Customizing the Design for Demonstration

You must modify the narrator code to say your first and last name instead of "EE178". Review the SPO256 data sheet to understand how speech is crafted from strings of smaller speech units. Unless you have a very simple name, this process will involve trial and error, and if you have a long name, you may need to expand the counter in the hardware test circuit. Once you are confident it works properly, demonstrate your final result to the instructor.

Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. You are required to demonstrate a working design. You are also required to submit an archive of your project in the form of a ZIP file. Use the Vivado "File" → "Project" → "Archive..." option to create the ZIP file. Name the archive lab6_yourlastname_yourfirstname.zip.

You will give your demonstration in class on the due date, and submit your archive to the instructor through Canvas before the end of that day. If your circuit is not completely functional by the due date, you should demonstrate and turn in what you have accomplished to receive partial credit.