Physics 623

FPGA II: Construction of a ROM

Aug. 5, 2008

1 Objective

A 32 word 12 bit read-only memory (ROM) is constructed in the Spartan II FPGA and is used to drive three seven-segment displays with a programmed pattern. The ROM is constructed using the Xilinx CORE generator and the CORE memory editor. The ROM addresses are set by a five bit binary counter which is driven with the test board external 100 MHz clock oscillator. The clock frequency must be stepped down to make the display visible using, for example, the MSB of a 25 bit binary counter. We will program the ROM with the first 32 prime numbers, i.e. 2 through 131 and read it out with the seven-segment displays.

2 Procedure Outline

The top level source will again be a schematic diagram. Since 32 prime numbers takes you into the hundreds, we have to use all three seven-segment displays and three seven-segment decoders. We will also again use a count stepdown for the external 100 MHz clock. The 32x12 ROM requires five address lines so we will need a five bit counter to sequentially address the ROM. While the ROM can be built directly with VHDL coding, we will use a Xilinx tool called the CORE generator which is used to build preconfigured logic functions. **ISE 8.1** or higher is required to support the CORE generator. The required counters and seven segment display decoders will be constructed in VHDL language (easily done after working through the WebPack tutorial) and once circuit symbols are created, all components can be entered into the circuit schematic.

2.1 Xilinx Cores

The CORE generator can be used to produce devices ranging in complexity from simple arithmetic operators and delay elements to complex building-blocks such as digital signal decoders, processing filters, multiplexers, transformers, FIFOs, and memories.

3 Core Implementation

The Core to be built is treated in the Project Navigator like another type of new source. The Cores are built using the CORE generator which can be run either inside or outside the Project navigator. To start the CORE generator from within Xilinx ISE select $NewSource$ and $IP(Coregen\&ArchitecturalWizard)$. The next window will present a choice of Core types. Choose $Memories$ & $StorageElements$, then $RAMS$ & $ROMs$, and then $Distributed\ Memory\ v7.1$ to make a ROM. After executing $Finish$ a $DistributedMemory$ window will open up which is a tool to construct the memory. Choose ROM, specify the required Depth and Width, go to $Next$, and then execute $Generate$. A .xco file will be generated, but remember that the ROM at this point is empty. Now you will run the Memory Editor to load data into the ROM. With the .xco file highlighted, click on $ManageCores$. From the window that opens up, load your

CORE .cgp Project File and under *Tools* select the *MemoryEditor*. Choose a memory block name and again specify the depth and width but you will now also be able to edit the memory contents. Select **2** for the Address Radix (since you will address the ROM in binary) and select **16** for the Data Radix since you will write the memory contents in Hex. Each memory entry will contain three Hex characters corresponding to 12 bits. For example, entering the prime 5 will be entered as **005** and the prime 131 will be entered as **131**. When the memory contents have been entered, save the memory definition and execute *Generate* under the File menu item. Select *Making a .coefile*. Exit the Memory Editor after saving all the files. You now have created a memory definition but the data is not yet in the ROM. Double click on the .xco file name and the *DistributedMemory* window will open up showing the parameters of the ROM. Click on *Next* two times and load the memory defintion file created earlier. You can check the memory contents by clicking on *ShowCoefficients*. The CORE generation procedure will also have made a ROM circuit symbol which will be available from the schematic window.

4 Creating the Bitmap

Open the schematic window and wire up the circuit. All the parts are available from the *Symbols* window. Rename the Clock input to **CLK**. No buffer is required for the Clock input but an **obuf** part is required for each output. The obufs must be renamed to match the 7 bit wide branch by renaming them to *obufname*(6:0). See the writeup circuit diagram. Check the schematic by running *checkschematic* and save the file.

The next step is to assign the circuit pins. Create a new constraint file (.ucf) by adding a new source of the constraint type. Highlighting the file name will show a *AssignPackagePins* in the *Processes* window. Clicking on this will open the PACE pin assignment editor from which you enter the required pin assignments. Save the file. You will find the Pin Assigments in the table at the end of this report.

Now Synthesize, Translate, Map, Place and Route as before. If you are successful, click on *GenerateProgrammingFile* and a .bit file will be created. Remenber to use the JTAG clock with the USB cable or the CCLK clock with the parallel port cable. The .bit file is than downloaded into the FPGA with the XESS GXSLOAD tool.

5 Downloading the Circuit

The binary configuration bit file is now ready to be downloaded to the actual FPGA chip. Follow the following steps for downloading.

1. Make sure the circuit boards are connected to a +9V plug-in power supply.
2. Make sure the circuit board is connected to the appropriate PC port using either the parallel port or USB serial cable.
3. You will then use one of several available **GXSTOOLS** depending on which function you are trying to implement.
    - **GXSTEST**: This utility lets the user test an XS Board for proper functioning.

- **GXSLOAD**: This utility lets the user download FPGA configuration files to the FPGA. Choose the appropriate download port (either the Parallel or USB0 port) depending on which download cable you are using.
- **GXSPORT**: This utility lets the user send logic inouts to an XS Board by toggling the data pins of the parallel port.
- **GXSSETCLK**: This utility allows you to set an integer divisor for the on-board 100 MHz clock.

6 Questions

1. The ROM will be made up out of the Spartan II Block RAM. Our chip has eight Block RAM cells each of which is a fully synchronous dual-port 4096-bit RAM giving 32 Kbits of total Block RAM. The rest of the circuit has registers that are made up out of the D Flip-Flops in the CLB. Estimate the number of CLBs required to implement the circuit and compare your estimate to the information in the Project Status Report.

2. Determine the maximum clock frequency for the circuit (It better be above 50 MHz). The required information is in the Post-Place Static Timing report.

3. Now we will look at the performance of the ROM by filtering out everything but the inputs and outputs of the ROM block. Run the Timing Analyzer and once the window is open select **Analyze** and **Against User Specific Paths by Defining Endpoints**. Enter the nets corresponding to the Sources and Destinations of the ROM and run the Timing Analyzer.
   What is the worst case access time of the ROM?
   What is the total worst case delay including nets?

7-Segment LED Display Pin Assignments

LED Decoder Output	FPGA Pin	7-Segment LED
S0	P67	XSA Board
S1	P39	
S2	P62	
S3	P60	
S4	P46	
S5	P57	
S6	P49	
S0	P47	Rightmost XStend
S1	P40	
S2	P28	
S3	P29	
S4	P27	
S5	P42	
S6	P48	
S0	P64	Leftmost XStend
S1	P65	
S2	P76	
S3	P50	
S4	P51	
S5	P54	
S6	P56	
CLK	P88	XSA Board Clock

7 Circuit Diagram