Project #4 Custom Qsys Computer

Team: Baby Geniuses (Michelle Blum, Christopher Dubois)
March 30, 2017

Overview

Since Qsys is such an invaluable tool for hardware configuration, it is only fitting that we learn to build our own computer in the program. For Project #3, we were tasked to build a computer in Qsys by following the tutorial on the Altera website. This time, we were to generate a computer that is able to accomplish the same task as the computer from the tutorial, but with different specified characteristics. Even though it took us a while to figure out the correct connections and arrange everything into working order, this still took significantly less time than building a computer using Verilog code and symbol blocks to be placed on the Block Diagram File (BDF) from each individual component required for the processor. Also, the resulting files within Quartus appear more compact than the alternative formed by not constructing the computer in Qsys. When we generated the Qsys model, the result was a single Verilog file, which reduces to one symbol block for the BDF. Although the logic may be difficult to follow if one were to look at the Verilog code, the Qsys tool provides an extremely clear visualization of the connections and conduits.

We started by downloading the project template from Moodle. This included all of the inputs and outputs that our final symbol block would display, and their corresponding pin assignments. Now, we were ready to open Qsys and begin forming our processor according to the provided characteristics.The components were relatively easy to define. The primary components were the SDRAM Controller, the Nios-II/f processor, on-chip memory, system ID, interval timer, JTAG_UART, and multiple parallel ports.

The next step was the most challenging part of the assignment for us. By following the warnings generated by our specifications, we were able to see where connections had to be placed and, for the most part, which conduits had to be exported. We got confused by the plethora of slaves that needed to be connected to the few available masters. However, as is explained in more detail in our solutions, eventually, our connections made all of the related errors disappear. The warnings almost had us export all of the necessary conduits, but missed a few. We had to figure out the remaining conduits by using the Media Computer template. Lastly, the base addresses proved to be simple assignments due to Qsys' ability to automatically assign them for our computer, while the IRQ numbers were primarily grabbed from project #3.

We successfully generated our Qsys model and built a BDF file that connected the Nios-II system to the DE2_115 board. The provided template from Moodle verified that the inputs and outputs of our symbol block were correct.

The program that we chose to test our computer with was the same program as last time. This program is intended to count. We used Eclipse to select this program and to connect the program to our configured hardware. After a bit of editing of the Eclipse code, our computer was counting using the red or green LEDs!

This assignment allowed us to further our understanding of the capabilities of Qsys and to see the various possible configurations that can be used in order to accomplish the same task. By creating the computer in Qsys, we learned the usefulness of listing available connections per component and of immediate attempted compilation of the configuration, which generates fairly clear warnings. If it was not for these abilities, Qsys would be much less user-friendly. The way that our general purpose computer (GPC) was designed within the BDF in project #2 is the level of visualization that Qsys would need to provide without the aforementioned two capabilities.

Solutions

Q1/2:

We began the project by downloading the project template so graciously provided by Dr. Horine and opened it in Quartus. In the file, we found all the various pins (input/output) required to complete the project, including some that were never used. From here, we opened up Qsys to begin creating our custom system. We configured the files in the order they appeared on the lab instructions.

We created a System PLL in place of the regular system clock, exporting the ref_clk, ref_reset, and sdram_clock inputs/outputs. We created an SDRAM Controller, which is used to ensure that data is not being simultaneously read from and written to the SDRAM. This is done by ensuring that the timing of the R/W operations are synchronized. We created a NIOS-II/f soft processor, which is used to perform all of the data calculations/operation of the system, attaching both a 2kB instruction cache and a 2kb data cache. We created an On-Chip memory comprised of 32 different 64 kB modules, used as our main memory. We created a System ID component to create a fingerprint unique to this system. We created a Interval Timer to synchronize the timing of all components of the system and avoid contention as much as possible.

We then created 4 parallel port components: one dedicated to the KEY pushbuttons, one dedicated to the SW slide switches, one dedicated to Green LEDs, and another dedicated to Red LEDs. These parallel ports allow for multiple input/output signals to run alongside one another without interrupting the flow of their neighbors. Finally, we added the JTAG_UART component, which serves as the chief communicator between the PC used to design the software and the Field Programmable Gate Array (FPGA) in question.

Q3/4:

Assigning the base addresses was as easy as pressing a button in the toolbar (System > Assign Base Addresses), while we assigned similar IRQ numbers to the last project. We used the Media Computer template as a guide to our own conduit exports and, most importantly, system connections. After we had initially created all of the components of the system, there were a myriad of errors and warnings, almost entirely regarding connections. We were generally able to make the connections purely from learning the last project, but the new components threw us off a bit. We were able to compare our project's connections to that of the Media Computer and were able to quickly shake off those final warnings. We then used the Media Computer to determine the necessary exports to get the system running. Below is **Figure 1**, a detailed diagram illustrating the connections amongst the system components.

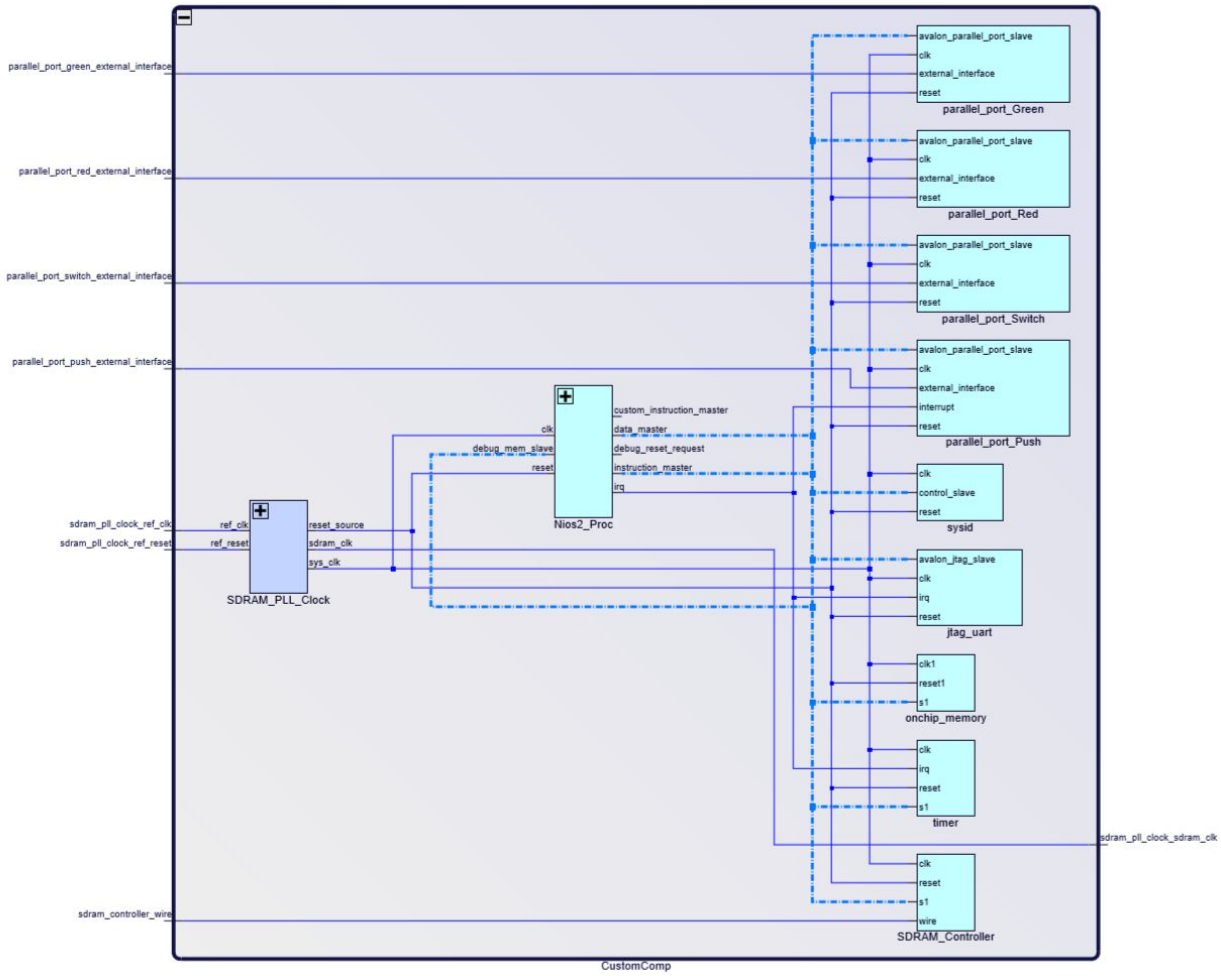

Figure 1 - the system connections shown in full graphic detail

We were then able to save and generate the system so that it could be imported to Quartus.

Q5:

After our Qsys project was generated, we were able to open the result in Quartus. The result was a single Verilog file that was practically unreadable. Lucky for us, we knew the exact function of this Verilog file from the Qsys display. We created a symbol file from this Verilog code and placed it into our BDF, which can be seen in **Figure 2.**

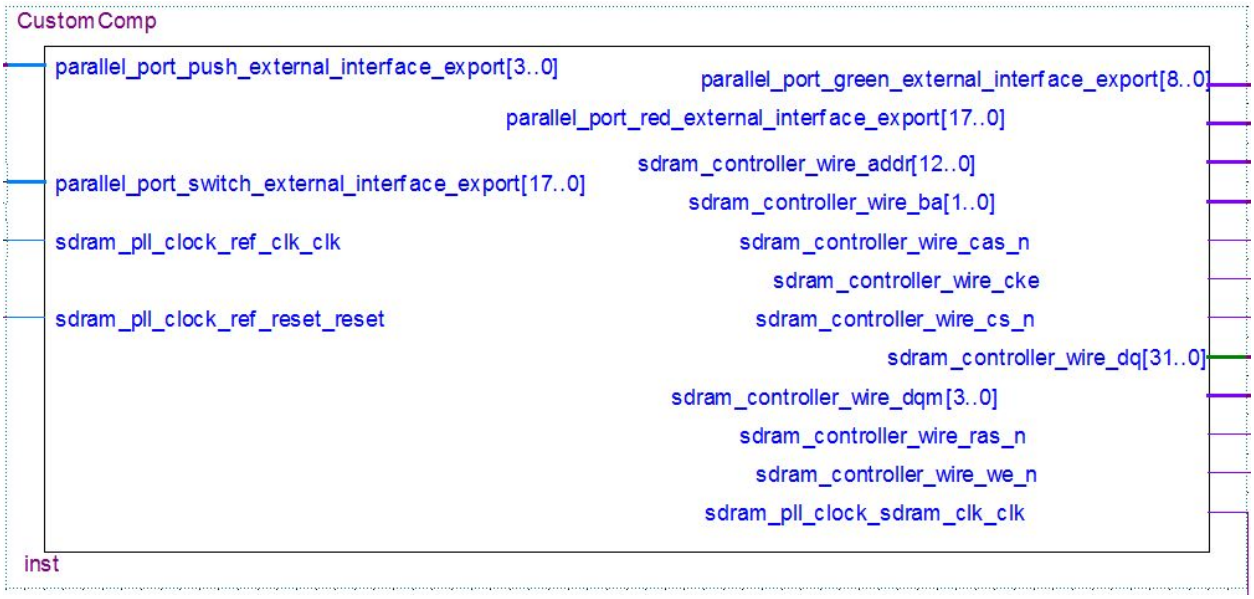

Figure 2 - the symbol file created from the Qsys Verilog file

The connections came intuitively, since the names of the inputs and outputs provided in the template appeared similarly to the names of the inputs and outputs of our symbol. However, we did have a few inputs and outputs from the template left over. We anticipated that this would happen, but then realized that we had too many leftover. This issue was immediately apparent to us because we had leftover clock inputs. How was our computer to be driven if it lacked any clock? Our initial thought was that we had to export the clock conduits in order to have them appear, but we did not know which conduits to export. We ended up using the Media Computer to see the three clock-related conduits that were exported. After exporting these additional conduits, we re-generated our symbol block and were relieved to see the appropriate clock names.

The final product can be seen in either **Figure 3** or **Figure 4**. In the end, we only had two components remaining from the original provided template. They were an input of JTAG_UART and an output of JTAG_UART.

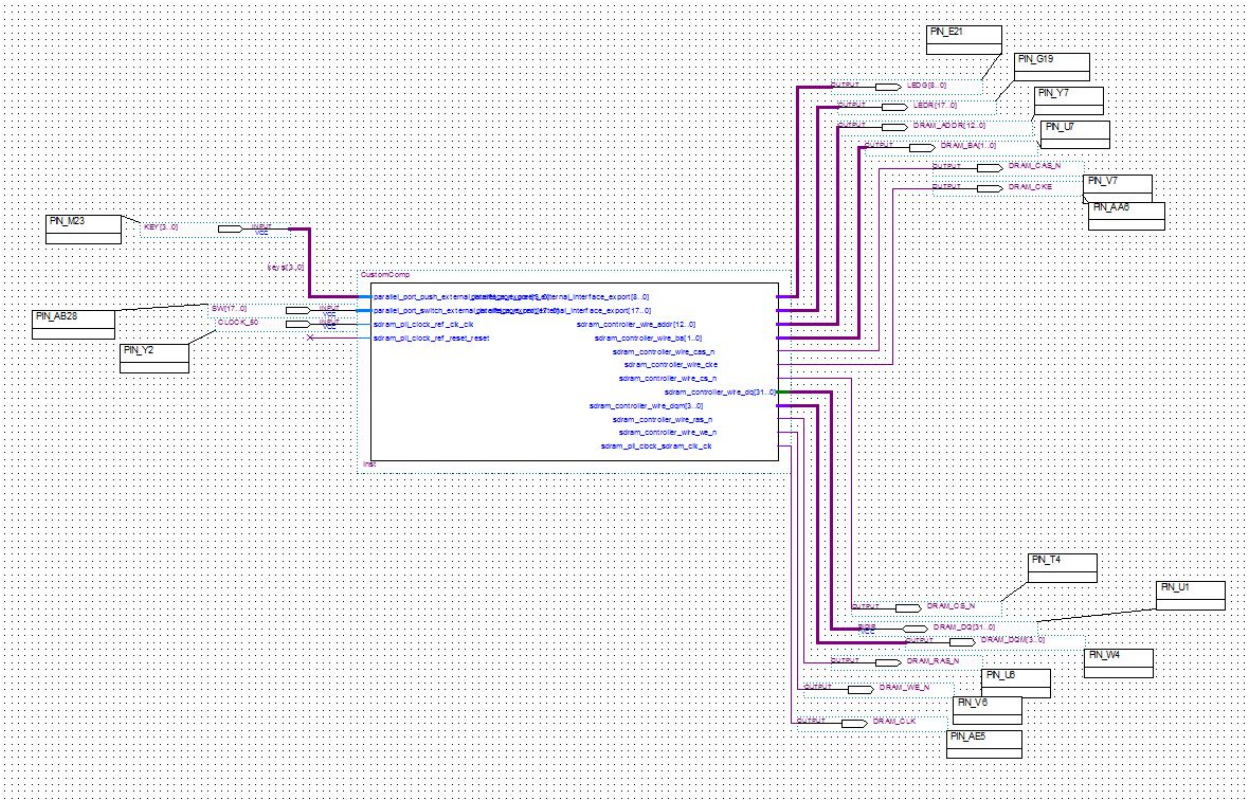

Figure 3 - the full BDF, complete with pin connections/assignments

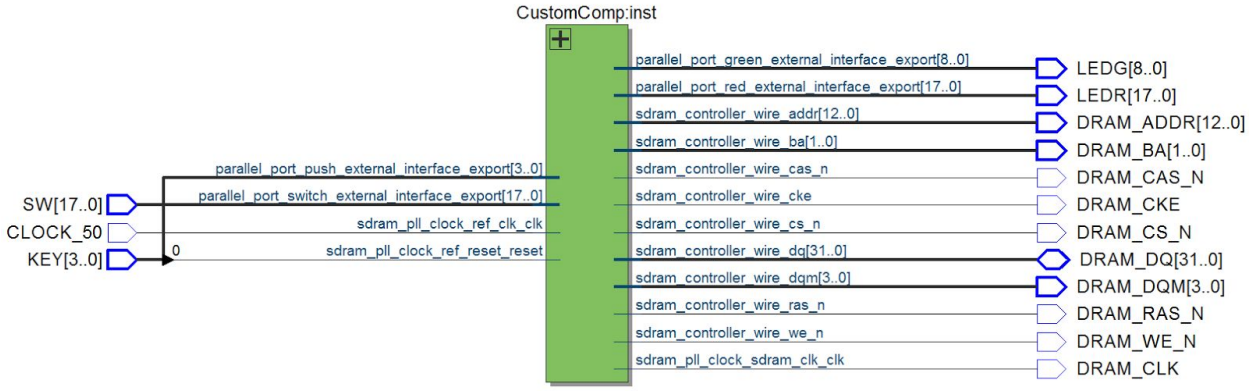

Figure 4 - a condensed version of the BDF (RTL) that shows the inputs and outputs in clearer fashion

Since our custom computer did not require any part of the JTAG_UART component to be exported from Qsys, the leftovers finally made sense. With this newly instilled confidence, we compiled our BDF and loaded the project onto the DE2_115 board.

Q6:

We were now ready to move into the software aspect of the project. We switched to Nios II Software Build Tools for Eclipse, the IDE used to write software for Nios II systems. Here, we created a new

project by importing the .sopcinfo file generated by Qsys and basing it on the Count Binary template that we used in the last project. We created both the software and the BSP from this process and started preparing the software for use. After configuring the BSP correctly, we then compiled the project to be run on the DE2-115 board. We then pressed run and observed the results.

Q7:

We arrived at the moment of truth; the time to test our computer. Now that the Qsys file had been connected to the DE2_115 board and our BSP and sample application had been created and selected, we had all of the necessary steps completed in order to see results. We ran the computer with much hope, and saw nothing. Absolutely nothing showed up on the DE2_115 board. This led to confusion because we knew that our computer was linked to the board, as completed in question 5, and we knew that the pin assignments were correct. The only part that we had not tested with our computer was the counting sample application provided by Altera. As it turns out, the counting application was not correctly referring to the red or green LEDs of the DE2_115 board. We had to open the Eclipse code and edit the section shown in **Code Block 1** below to view the output on the board. With the quick switch of the wording in the code from Green LEDs to Red LEDs, we could make the computer count on either set.

```
static void count_led()
{
#ifdef PARALLEL_PORT_RED_BASE
    IOWR_ALTERA_AVALON_PIO_DATA(
      PARALLEL_PORT_RED_BASE,
        count
        );
#endif
}
```

Code Block 1 - count_led(): a function from the BlinkLEDs program used to write the binary values to the LEDs

Q8:

The SDRAM Controller is primarily used to synchronize the actions of the SDRAM (Synchronized Dynamic Random Access Memory) and to ensure that there is no bus contention in terms of read/write operations. **Figure 5** shows the SDRAM controller of our custom computer in detail:

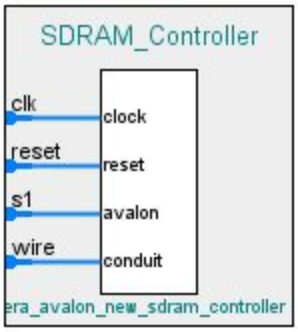

Figure 5 - Components of SDRAM Controller

Base addresses refer to the first address reserved for a component, usually the first in a series of components. For example, in reference to the SW slide switches, the address of SW[0] would be the base address, and the first in a series of addresses.

A conduit essentially is a bus with various lanes used to transfer vital information between components in the system. Below, in **Figure 6**, is a picture of a bus used in our system.

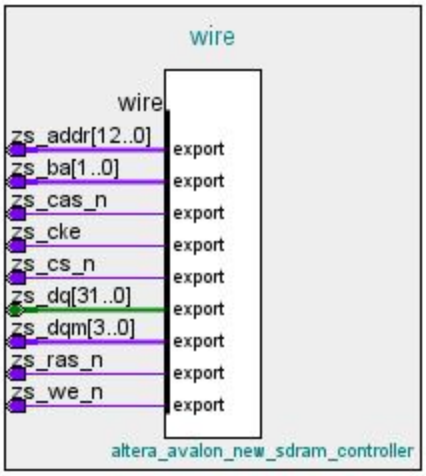

Figure 6 - Components of a Conduit