Project #3 NIOS-II Qsys Tutorial

Team: Baby Geniuses (Michelle Blum, Christopher Dubois)
February 23, 2017

Overview

Solely using Verilog files and Block Diagram Files (BDFs) to build a functioning computer can be a time consuming and over-complicated process. The Quartus Suite has a built-in tool, called Qsys, that makes the process of building a reduced instruction set computing (RISC) computer processing unit (CPU) significantly faster and easier to understand at first glance. Essentially, Qsys has the ability to configure a processor with no programs in it. The programs are to be decided by the user, meaning that Qsys generates computers using Field Programmable Gate Array (FPGA) logic. Instead of coding each individual Verilog file for the components of the processor, Qsys already has these files prepared and knows the appropriate connections available. For this assignment, we were asked to design our own computer using Qsys and the default counting program that altera assigns to the processor through the tutorial.

We started by carefully following Altera's Hardware Tutorial. This taught us about common components of a processor, including: Nios II/f processor core, on-chip memory, clock, JTAG UART, and input/output pin assignments.

Once the tutorial was complete, we had successfully used the initial system requirements, on the hardware side, to generate a system in Qsys, integrate our Qsys system into the Quartus II project, assign pin locations and clock time, compile the design, and download the design to the DE2-115 board. On the software side, we were able to generate the system in Qsys, use Eclipse to build our programs, and run the software on the DE2-115 board. Since these two sides split after the system is designed, it would be efficient for future work to focus on the hardware and software components simultaneously, eventually arriving at a single unified result.

We actually went through the tutorial fairly quickly, with our system functioning appropriately on our initial attempt. Despite the steps being quite clear, we had to return to the tutorial and thoroughly evaluate each section to even remotely comprehend how Qsys created a RISC CPU in such a short period of time. We went through the steps again and again, and while looking at the final function of our circuit, assigned a reason as to why each section was necessary for completion. The individual steps within the sections, although important, we realized are more software specific and not intertwined with the concept and intent of a system on a programmable chip (SOPC) builder such as Qsys.

By the time we arrived to number 3, we knew that the connections tab was essential in developing our chart of design flow. We were also able to use our understanding of the separate hardware and software

development chains to decide which Altera tools were useful in the movement from Qsys to Quartus to Eclipse. After completing number 3, our most time-consuming activity became examining the .qip and .sopcinfo files to determine their specific purposes. However, there were some key words or lines within the code that aided us in our ultimate decision about the purpose, as is discussed in detail later in this report.

Our last task was to explain the relevance of a board support package (BSP). As computer users, we often forget that the processor not only needs to communicate within itself, but also must be compatible with the operating system (OS). This is accomplished by the BSP, which we found to be an interesting fact and useful reminder of the many factors that a computer creator must consider.

The work that was performed in this assignment allowed us to focus on the components of a processor that change with each device, as opposed to the components that tend to remain fairly steadfast across the board. Qsys would be an extremely useful learning tool even if that given ability to focus was the only purpose it served. This allowed us to see that complex circuits can appear simple if one uses the right tools. For example, if we had designed our general purpose computer (GPC) from project #2 using Qsys, many of the Verilog files would not have been necessary and would have been condensed into a single generated system.

Solutions

Q1/2:

The tutorial gave very specific directions which the programmer should follow to successfully create a Nios II system. It started in Quartus, where we opened a provided project file with two provided inputs (Vcc and a clock), as well as an output bus (LEDG[0..7]). We were then instructed to open Qsys and create a new system to configure. After selecting the target FPGA and choosing the correct clock frequency, we were ready to create the components. We created the following components entirely within Qsys: Nios II/f processor core, on-chip memory, JTAG UART, interval timer, system ID peripheral, and the LED parallel input/output. After creation, we configured each to indicated specifications. At this point, errors and warnings began to show up in the bottom bar, but would easily be solved later in the solution. We then began to create several connections between the components, the clock, and a reset voltage. Once we completed the connections, there were no longer any errors in Qsys. The following

image (**Figure 1**) is a visual rendition of said connections:

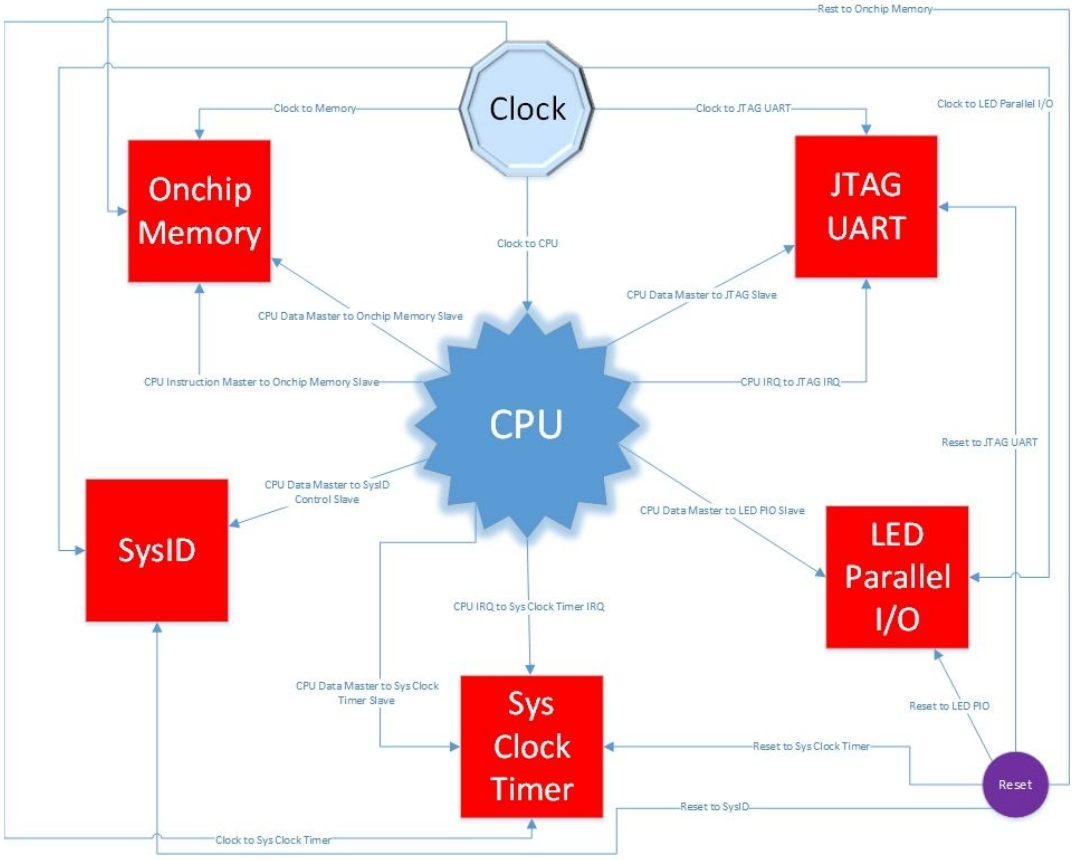

Figure 1: Visual Representation of Qsys Connections

We then were able to generate the system and export it back to Quartus.

In Quartus, we were easily able to import the system file and link the pins to the respective ports. To be safe, we renamed PLD_CLOCKINPUT to CLOCK_50. The next steps were to import the .qip file and assign the pins, two steps we performed without issue. We then had to import an .sdc file, which we realized still had the PLD_CLOCKINPUT name within its code; we promptly replaced it with CLOCK_50. We were then able to compile the project, including the Qsys generated system. We had no errors at this stage. We were then able to download the framework to a DE2-115 board without hassle via the .sof file. There was no visual feedback yet as no program had been loaded into the framework.

Finally, in Nios II SBT for Eclipse, we built a program to be run within the Nios II computer. We opened Eclipse and were able to create a program and BSP from a template - namely, the Binary Count template. After configuring the file name, choosing the file path, and importing the .sopcinfo, we were compilation-ready. We then compiled the project and ran it on the target DE2-115 board, where it quickly counted in binary and delivered the output to the green LEDs, as shown in **Figure 2**. This concluded the tutorial.

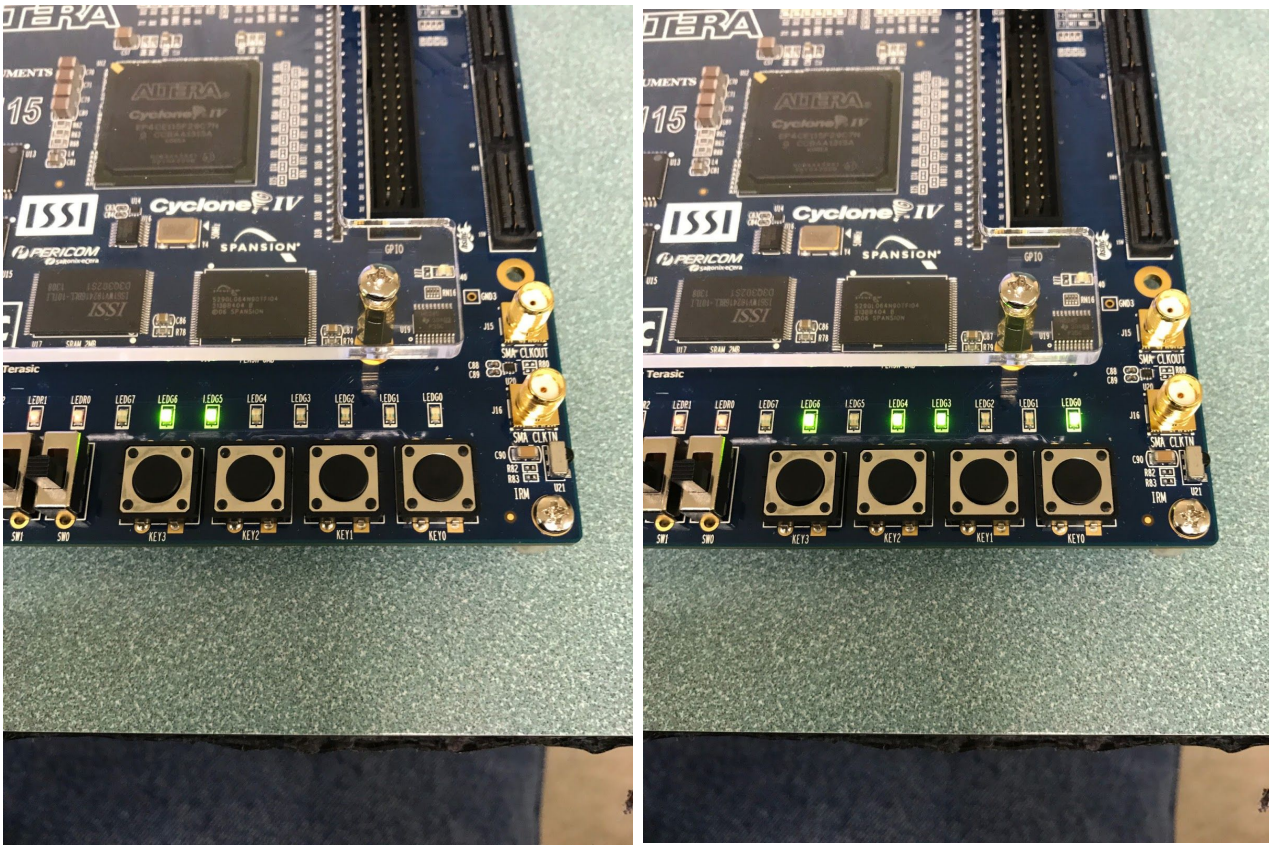

Figure 2: Images depicting the Binary Count program in action

Q3:

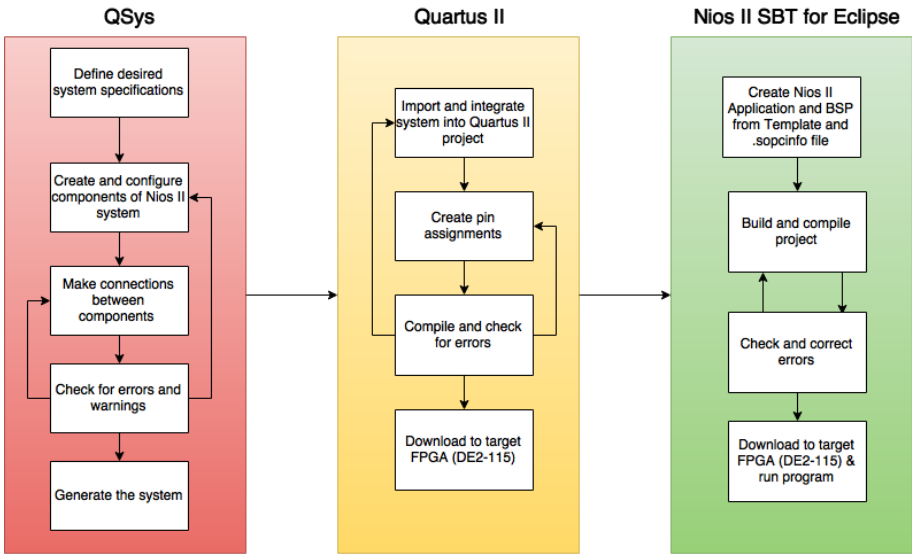

Figure 3: Design Flow

Figure 3 shows the general design flow of creating a Nios II computer system on a DE2-115 FPGA board. The design flow can be divided distinctly between the three tools used to create the system: QSys, Quartus II, and Nios II. On the Qsys side, the process begins with a desired list of specifications. The engineer must determine the balance between economy and quality, as well as determine what their hardware should be capable of performing. After this is decided, the engineer is then responsible for creating and configuring the components to run the system in a way that fits these predetermined realistic standards. This includes memory size, cache size, and clock frequency. It is then necessary to make the connections between the components and the clock so that they may work in tandem. At this point, it is up to the engineer to check for errors of any kind before attempting to generate the system. If there are no apparent errors, the system may be generated; if there are errors, they must be rectified before generation.

The next tool is Quartus II, which uses the information generated from Qsys to integrate into its own ecosystem. First, the engineer must import the system to Quartus so that it may be properly used. From here, the system can be integrated into a BDF with proper I/O and pin assignments. This way, the system may properly function on the FPGA board. At this point, the code can be compiled; success is dependant on the accuracy of the connections and their intended places. Errors can be easily fixed once the user sees a misconnected component, and the code can then be recompiled until all the bugs are out. The system is then finally downloaded to the FPGA board.

The final step is Eclipse, which is where specific programs are created to run on the system. In order to create an appropriate program, the engineer must write the new code or insert a template and import its .sopcinfo file from the Qsys generation. The .sopcinfo file informs Eclipse of the specifications and the program is built accordingly. In this particular case, the program was built directly from a template, but in general, the software can be any combination of original and template code. The engineer must then build and compile the software into a set of machine-level instructions. If there are errors, they must be corrected before the compilation is successful. The software may then be downloaded to the FPGA and run for debugging purposes. If all works completely as intended, then this ends the process; however, this rarely happens. The engineer is responsible for continually refining and improving the software until it runs without error.

Q4:

The purpose of Qsys is to generate and configure a computer system based around the Nios II Processor. It uses a library of Verilog files related to common processor components to simplify the user's experience. This way, the user only needs to define the properties and connections of these common components in order to generate a functioning system.

When one is forming a complicated circuit, it would be useful to use Qsys in order to form the general functionality of the computer, and to only write function specific Verilog files for special computer capabilities. Instead of "reinventing the wheel", the user can start their project at a new development point.

We thought that one of the most useful features of Qsys was its ability to list available connections between components. Instead of inserting a module into a system and wondering where the appropriate location lies, Qsys ensures that the user is aware that there are a limited amount of logical options. Therefore, Qsys not only aides its users by providing pre-existing components, but also can provide connection information for new components that users add to the library.

Q5:

The .qip is used as a configuration file, containing the information and location pertaining to each component of the Qsys synthesis process. The extension name is short for Quartus IP file. This configuration can be used and reused easily because instead of having to manually import all components individually into Quartus, the .qip file points to them and their specifications easily. In a way, it functions similarly to the .qsf file in that it configures global assignments for various system components.

Q6:

The .sopcinfo file is a XML file that gives all necessary information about the created system to Eclipse in order to correctly configure the software to run. Coding within the file is fairly repetitive because it simply states the properties for each parameter. For example, for the parameter “deviceFamily”, it should be read as a string and it is set to enabled or is being actively used. Actually, the extension of the file gives a hint towards what is contained inside, considering that we were working on a SOPC and this file passes information about the system. Looking through the file, it is easy to see separate settings and parameters that we ourselves worked on in Qsys.

Q7:

BSP is a package of files that modifies the given program for compatibility with the given hardware. Exploring the BSP folder, we found C header files, including system.h, which defined a countless number of parameters relating hardware and software; this includes the Device Family, CPU implementation version (*f* or *e*), and Cache Size. This folder also included drivers for the target hardware, as well as a visually-assisted BSP description guide (summary.html).