ECE385
DIGITAL SYSTEMS LABORATORY
SoC Configuration for Lab 9 - AES

Abstract and Goals

In the previous labs, we've already created a full Nios II based system which lets us execute C programs in the software. For lab 9, we will use a Nios II SoC to test a software version of the AES encryption algorithm. This will be entirely written in C, and demonstrated in the Week 1 lab. We will then create a HDL (hardware based) implementation of the AES decryption, which will also receive data from the Nios II, but decrypt AES in hardware. Your lab report should compare the relative performance of the hardware and software encryption and decryption methods.

Set up the System Combining the FPGA with the Nios II Processor:

For the first part, you are to make a SoC setup in Qsys, similar to what you did in lab7. The only differences are:

- Additional PIOs for communication to the AES decoder which you'll design
- The addition of a peripheral called the **JTAG UART**

For the additional PIOs, you will create 4:

Name	Direction	Width
to_hw_port	out	8
to_hw_sig	out	2
to_sw_port	in	8
to_sw_sig	in	2

The "port" PIOs will act as our data bus in and out from the AES decryption hardware block that you will design for Week 2. The "sig" PIOs will act as handshaking. We'll talk more about designing a protocol for communication between software and hardware in time for Week 2.

Next you will add the JTAG UART peripheral. This is found under Interface Protocols->Serial.

INQ. 2

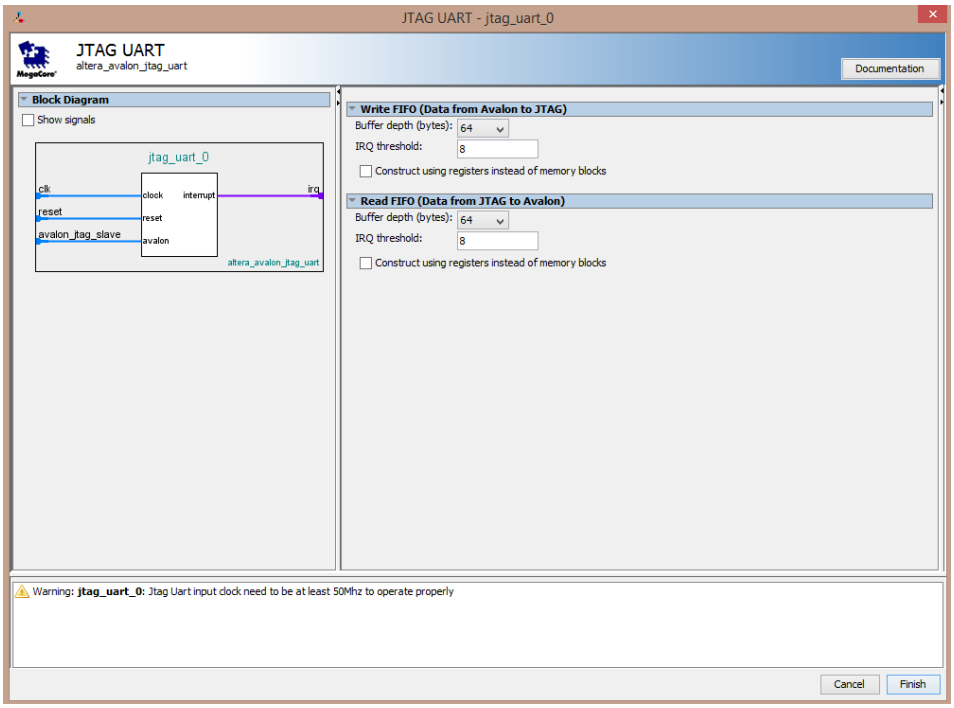
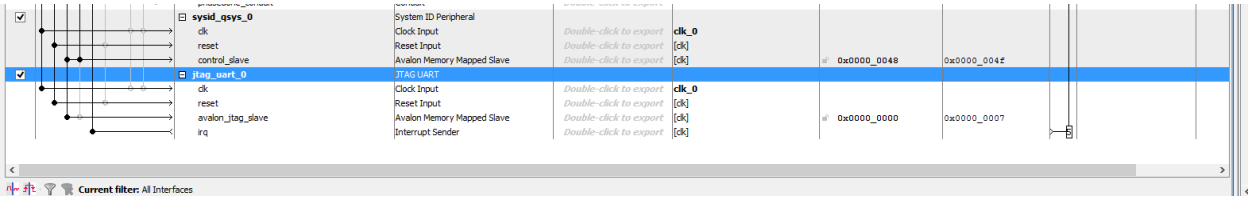


Figure 1

You can simply leave the settings here as a default. What this block does is give you the ability to use console (printf) commands from the Nios II which go through the programming cable via USB. While this is typically not a good user interface for an embedded system (as it requires the programming cable to be connected and the user to have all the Altera software installed), this is an excellent way to debug your software while in development.

Make the connections as shown below:



Making the standard connections for clk/reset/data bus, that we've been doing for the other peripherals. One difference here is that we must assign an **interrupt** for this, which we will assign IRQ (interrupt request) 5. Connect the interrupt controller to IRQ and give it the number 5 (this is on the far right of the row. The reason for using interrupts here is that transmitting or receiving text over the console is in general very slow, and we don't want this procedure to block on the CPU. Therefore, the typical way in which printf is implemented is that control is returned program as soon as printf is called, but an interrupt is set up at the end of transmission for each buffer. This way, the CPU does not have to block while waiting for each of the characters to be transmitted, it is only interrupted whenever the peripheral (the JTAG UART) needs more data.

Hints for Lab 9 (Week 1):

For the first week, you will write code which performs a 128-bit AES encryption in C to execute under the Nios II. For the demo, you will accept a plaintext message through the debug console and your code is expected to print out the ciphertext using the serial console. Remember to include `<stdio.h>` which gives you printf capability using the JTAG-UART peripheral which you added.

In addition, you will be asked to benchmark your code running on the Nios IIe, and come up with a number in MB/s (megabytes encrypted per second). You might need to write a benchmarking routine to compute this number, and the demo point for this will be contingent on coming up with a reasonable routine to do this experiment. Each implementation can have different performance, for this lab, we won't concern ourselves with writing the highest-performing code, but we will use this number to compare it to the hardware decryption which we will do in Week 2. Note that the AES algorithm is largely symmetric, so for the most part, we expect encryption and decryption to have similar performance when running on the same platform (of course, a hardware implementation may have very different performance than a software one).