EE 3420 Lab Guide 1: Keypad to 7-Segment Display w/ NIOS II

Written by: Grant Seligman, Gabe Garves, and James Starks

Example Overview:

Design a circuit system that takes in a 4-bit input from a 4x4 keypad and output the result to a 7-segment LED display.

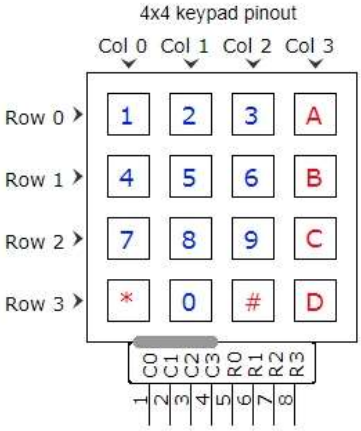


Figure 1

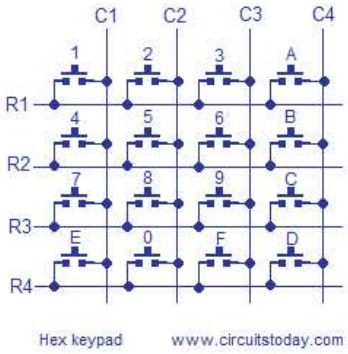


Figure 2

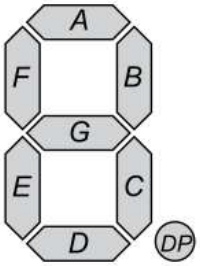


Figure 3

Verilog Breakdown:

To begin, one needs to consider how a 4x4 keypad functions. The rows are outputs from the FPGA, and the columns are inputs to the FPGA. You can switch the inputs to rows and outputs to columns as long as you're consistent. When button 1 is pressed it creates a short between R1 and C1 in **Figure 2**. Sending 0 on R1 and setting the rest of the rows high will isolate R1. Not sure why (maybe phantom power), but the columns will read high until you push a button. Pressing any button along that row will short it to ground and the output will read 0. The code in **Figure 4** checks each row sequentially and if a button is pressed, the column and row connect and a low output is read by the FPGA.

```
1   //Keypad Decoder Module
2   /*
3   AUTHOR: GABE GARVES
4   EDITED BY: GRANT SELIGMAN
5   DATE: 2/25/2020
6   EDITED: 4/13/2020
7   FROM: TXST SENIOR DESIGN PROJECT FALL 2019-SPRING 2020
8   FOR: TEXAS STATE UNIVERSITY STUDENT AND INSTRUCTOR USE
9   */
10  module keypad_decoder(key, row, column, clk);
11      output reg [3:0] key;
12      output reg [3:0] row;
13
14      input wire clk;
15      input wire [3:0] column;
16
17      always @ (posedge clk) begin
18          case (row)
19              // Row 1
20              4'b1110:
21                  case (column)
22                      4'b1110 : key = 4'b0001;
23                      4'b1101 : key = 4'b0010;
24                      4'b1011 : key = 4'b0011;
25                      4'b0111 : key = 4'b1010;
26                      default : row = 4'b1101;
27                  endcase
28              // Row 2
29              4'b1101:
30                  case (column)
31                      4'b1110 : key = 4'b0100;
32                      4'b1101 : key = 4'b0101;
33                      4'b1011 : key = 4'b0110;
34                      4'b0111 : key = 4'b1011;
35                      default : row = 4'b1011;
36                  endcase
37              // Row 3
38              4'b1011:
39                  case (column)
40                      4'b1110 : key = 4'b0111;
41                      4'b1101 : key = 4'b1000;
42                      4'b1011 : key = 4'b1001;
43                      4'b0111 : key = 4'b1100;
44                      default : row = 4'b0111;
45                  endcase
46              // Row 4
47              4'b0111:
48                  case (column)
49                      4'b1110 : key = 4'B1110;
50                      4'b1101 : key = 4'b0000;
51                      4'b1011 : key = 4'b1111;
52                      4'b0111 : key = 4'b1101;
53                      default : row = 4'b1110;
54                  endcase
55              default: row = 4'b1110;
56          endcase
57      end
58  endmodule
```

Figure 4

In order to turn these 4-bit values into actual numbers and letters one has to decode them into an 8-bit binary output for the 7-segment display. **Figure 5** shows how easy this is to do in verilog.

```
/*
AUTHOR: GABE GARVES
DATE: 2/7/2020
EDITOR: GRANT SELIGMAN
DATE: 4/13/2020
FROM: TXST SENIOR DESIGN FALL 2019-SPRING 2020
FOR: TEXAS STATE UNIVERSITY STUDENT AND INSTRUCTOR USE
*/
module seven_seg_decoder(out, in);
  output reg [7:0] out;
  input wire [3:0] in;

// Instead of running of a clock, this aways block waits for and input
// from the keypad. Thus making this module asynchronous.

  always @(in) begin
    case (in)
      // Layout of bits, MSB is DP, then decending G-A
      //               [DP]GFE DCBA
      4'h0 :    out = 8'b0011_1111;
      4'h1 :    out = 8'b0000_0110;
      4'h2 :    out = 8'b0101_1011;
      4'h3 :    out = 8'b0100_1111;
      4'h4 :    out = 8'b0110_0110;
      4'h5 :    out = 8'b0110_1101;
      4'h6 :    out = 8'b0111_1101;
      4'h7 :    out = 8'b0000_0111;
      4'h8 :    out = 8'b0111_1111;
      4'h9 :    out = 8'b0110_0111;
      4'hA :    out = 8'b0111_0111;
      4'hB :    out = 8'b0111_1100;
      4'hC :    out = 8'b0011_1001;
      4'hD :    out = 8'b0101_1110;
      4'hE :    out = 8'b0111_1001;
      4'hF :    out = 8'b0111_0001;
      default : out = 8'b1000_0000;
    endcase
  end
endmodule
```

Figure 5

Lastly for the verilog modules, one doesn't want to cycle through the keypad decoder too fast. This could cause issues with the output signal to the 7-segment decoder. So **Figure 6** is another clock converter that brings the 12MHz clock down to 60Hz.

```
1  // 60Hz clock
2  /*
3  AUTHOR: GABE GARVES
4  DATE: 4/3/2020
5  FROM: TXST SENIOR DESIGN FALL 2019-SPRING 2020
6  FOR: TEXAS STATE UNIVERSITY STUDENT AND INSTRUCTIOR USE
7  */
8
9  module stepdown_clk(clk_out, clk_in);
10     output reg clk_out;
11     input clk_in;
12
13     reg [23:0] counter; //need at least 24 bits
14
15     initial begin
16         #0    counter = 0;
17         #0    clk_out  = 0;
18     end
19
20     always @(posedge clk_in) begin
21         if (counter == 0) begin
22             counter <= 200000; // 12 MHz / 60 Hz = 200,000
23             clk_out <= ~clk_out;
24         end else counter <= counter -1;
25     end
26 endmodule
```

Figure 6

FPGA Implementation:

This task can be done similarly to what you did in EE 2420; where you would take each module and convert them into symbol files and program the FPGA with HDL hardware layer only. But for this lab, we want to incorporate embedded C programming with our FPGA. This basically means that the FPGA will have a hardware layer that is controlled with a software layer and this is done with Intel's own NIOS II softcore CPU.

Start by opening Quartus and creating a new project. Make sure to name the folder you put your project in and every subsequent file in this project without any spaces and use underscores instead. This could be a problem later during compilation and creating the Eclipse Project. Here is the device number 10M08SAU169C8G. If you need guidance on starting a project, refer back to the 3-bit adder lab guide from Digital Logic or ask your instructor.

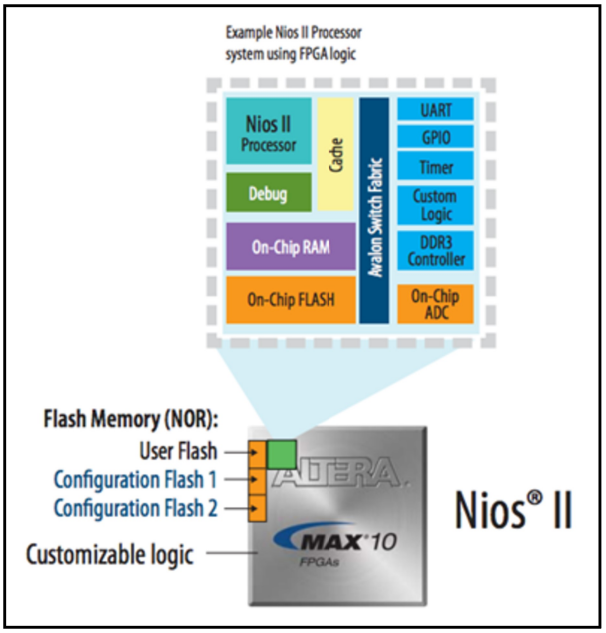


Figure 7

The NIOS II Processor acts like the processor in the KL46Z basically. You program it with C programming to do computation and control I/O. So now you will be able to take any hardware design you create and play with it in software with C programming.

NIOS II Setup:

First launch the Quartus Platform Designer either on the menu bar shown here in **Figure 8** or find it under the Tools tab.

Figure 8

The Platform designer will open up a new tab. Once this is open, you will need to use the IP Catalog to search and add components - if you do not see this, click **View** >> **IP Catalog** to bring it back (**Figure 9**). This IP Catalog window should be on the left-hand side of the Platform Design window by default.

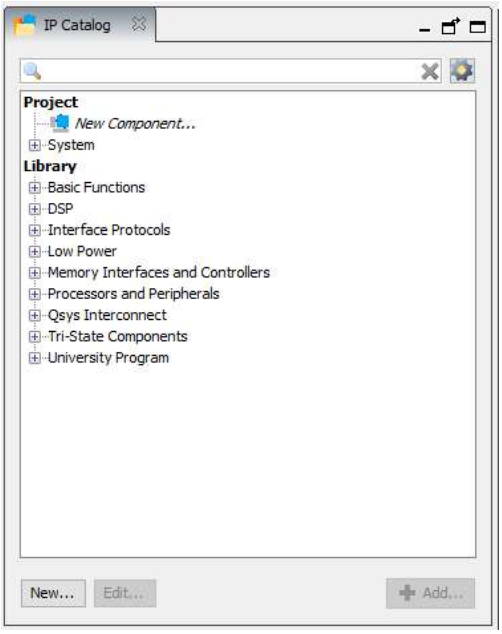


Figure 9

Search and double click "**NIOS II Processor,**" and select **NIOS II/f,** then click **Finish** at the bottom of the window. This will add the NIOS II to the **System Contents** window. See **Figures 10 and 11.**

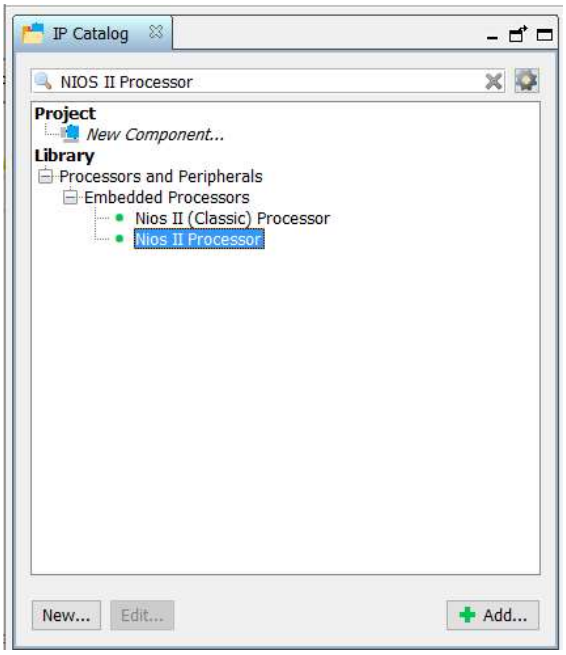


Figure 10

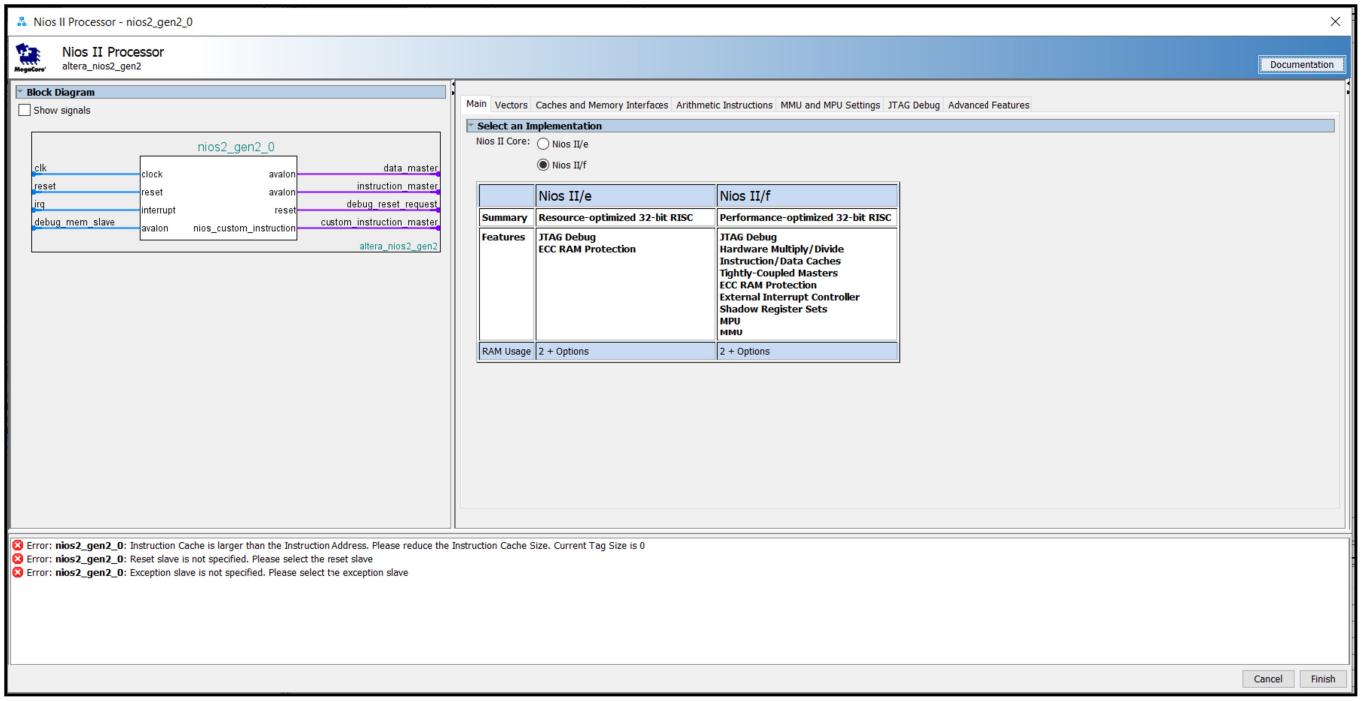


Figure 11

Next, search for and select "**On-Chip Memory (RAM or ROM) Intel FPGA,**" then click finish to add.

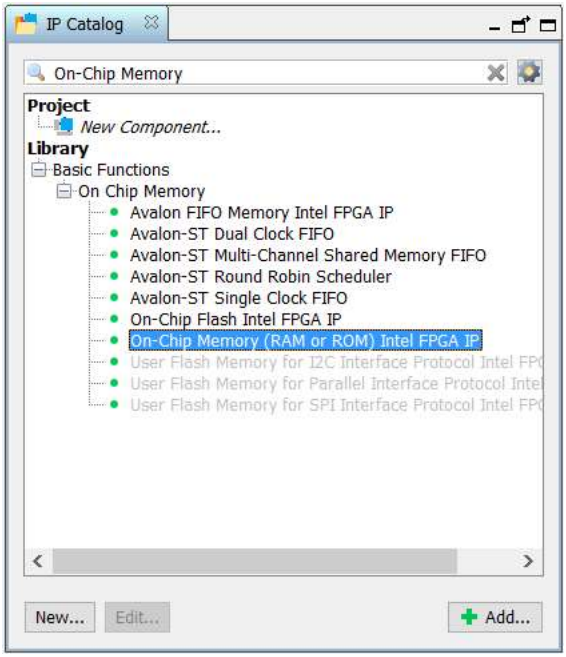


Figure 12

Search and select "**PIO (Parallel I/O) Intel FPGA IP**." This will be how the NIOS processor will receive data from the keypad decoder, so set the **Width** to **4 bits**, and **Direction** to **Input**. See **Figures 13 and 14.**

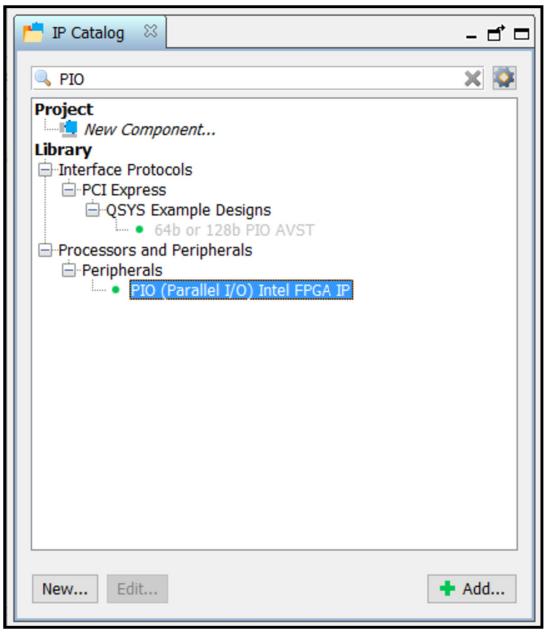


Figure 13

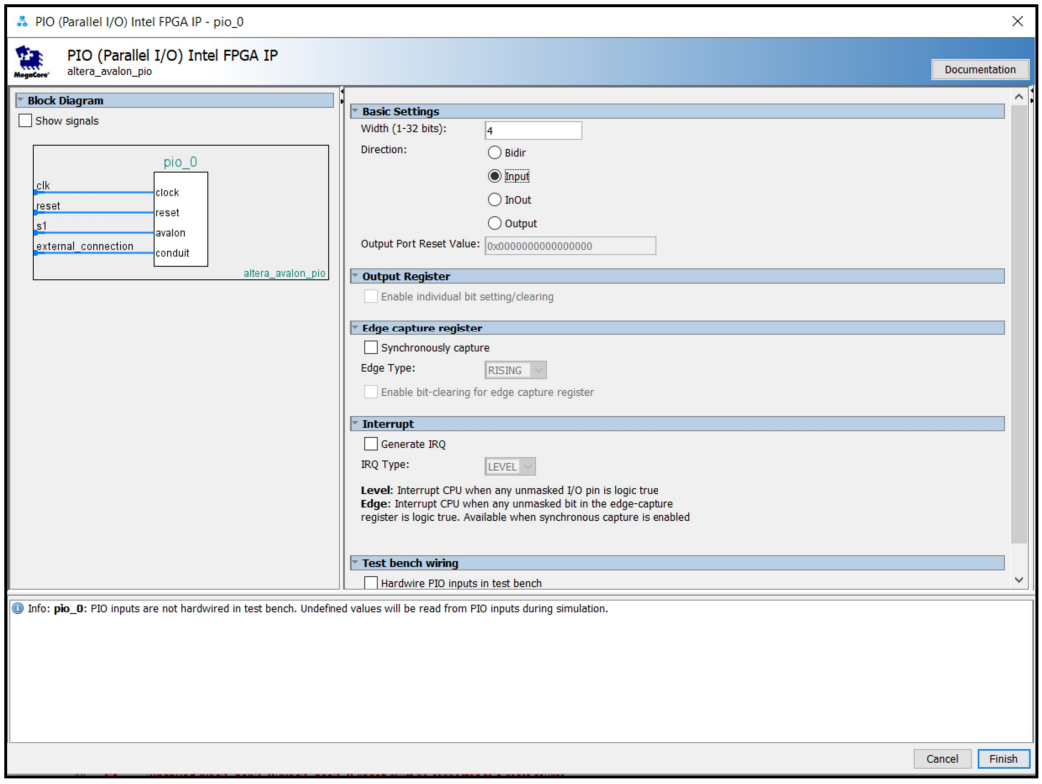


Figure 14

Now, add another **4-Bit PIO** for the seven segment decoder, but this time with the **Direction** set as an **Output.**

Search and add "**JTAG UART Intel FPGA IP,**" click finish to add.

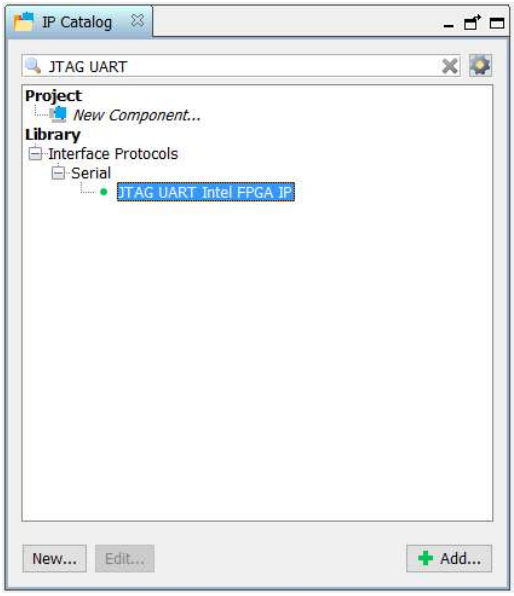


Figure 15

Finally search and add "**System ID Peripheral Intel FPGA IP,**" enter a random number in the **32 bit System ID** field - this is for Eclipse. Eclipse is the C IDE built into Intel Quartus. More on Eclipse will be discussed later.

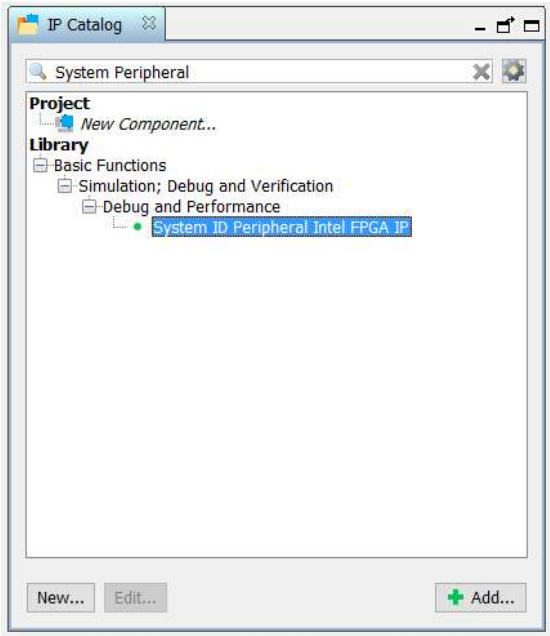


Figure 16

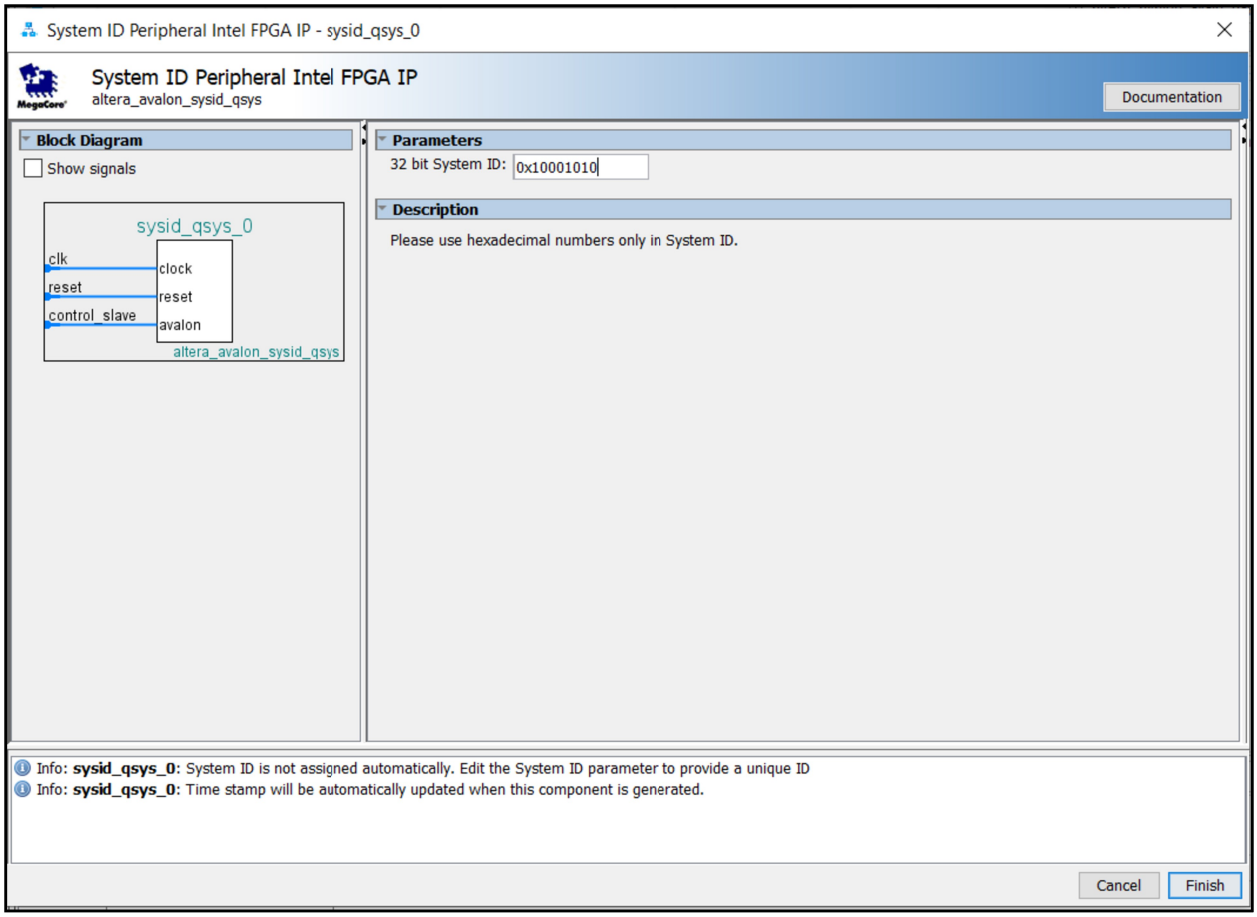


Figure 17

What is nice is that you can drop as many PIOs into NIOS as needed with this custom CPU platform. This beats having to design an entire ALU unit with an Execution Engine, Memory, Instructions, etc. As a note, the NIOS II takes up 39% of the Max10's system memory. Now this won't be an issue for this project but you should keep that in mind if you want to go off and develop your own projects.

Now that everything has been added, you will notice a bunch of errors and warnings in the message window. Before you can generate this system you must resolve all **red** errors and warnings, but **green** info messages are fine. You need to hook up clocks, resets, and the Avalon Memory bus. Look at the **Figure 18** below and copy the connections exactly.

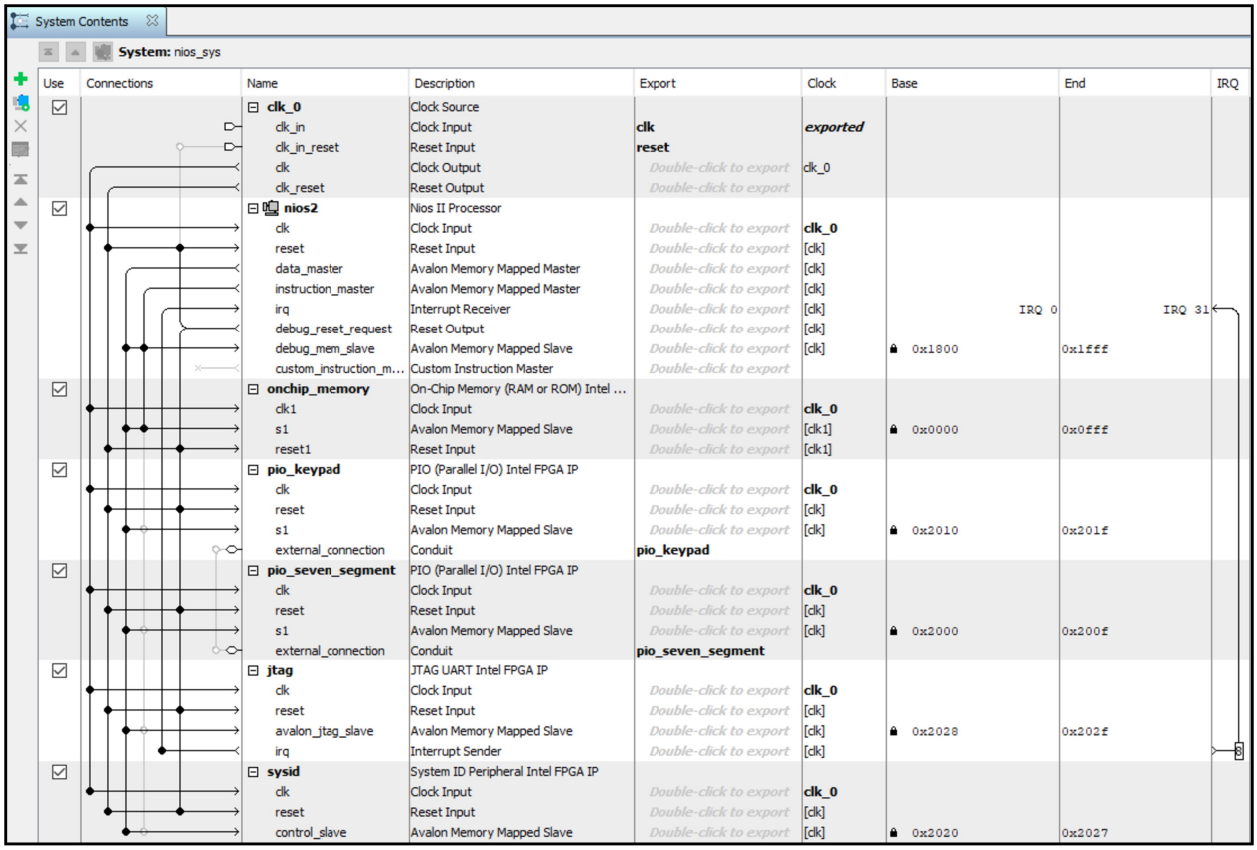


Figure 18

If you followed the instructions correctly, you should have something similar to this on the platform designer. ***Note: It's best to rename components to keep things neat.***

Each module will need a **clock** attached and each module **reset** must be attached to **clk_reset** and **debug_reset_request**. Make sure to double click on **clk_in_reset** in the **Export** column and name it **reset**. This will set up an external reset input for NIOS.

Each module needs the **Avalon Memory Mapped Slave** bus attached to the data master, this is so the Eclipse software application you will write will know where in memory these modules are connected.
Only **On-Chip Memory** will be connected to the **instruction_master** because this is where program instructions will be stored.

The **JTAG Interrupt Sender** also needs to be connected, set the **IRQ** value to **8**, so it does interrupt the program. You can change this value in the **IRQ** column **(Figure 18).**

Now you should see values populate the **Base** and **End** columns in the **System Contents** window. Click on the little **Lock** next to the base address in the **On-Chip Memory** row, this will keep the On-Chip Memory base address at 0. See **Figure 19.**

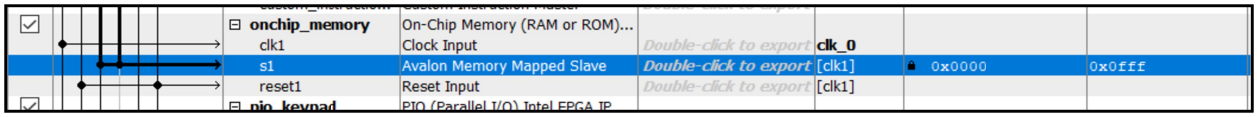


Figure 19

Next, look for the **Conduit** in both **PIO** components and **Double-click to export**. Type in the name of each respective **PIO (Figure 20).**

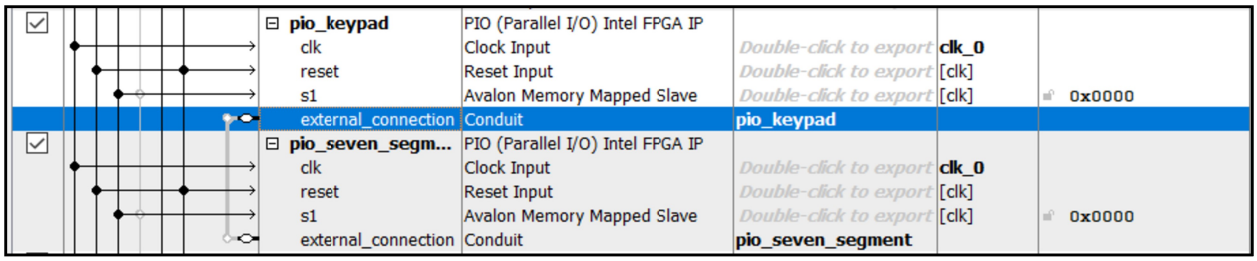


Figure 20

Now you will need to set up the memory map of NIOS. Basically this means that NIOS has sections of 32-bit memory that it can read and write to. Each memory location needs to have its own specific address for NIOS to read and write to it. To prevent or resolve any overlapping memory addresses, at the top menu, click **System** > **Assign Base Addresses**. Now you will see each base address for each component has been set and shows up in the **Base** column.

Next for the memory: Right click the NIOS II component and click **Edit**. Open the **Vectors** tab and set both **Reset** and **Exception Vector Memory** to **onchip_memory.s1**, select finish. Now save this design by going to the top menu bar and click **File > Save**. Name it **NIOS _II_Core** or something unique.

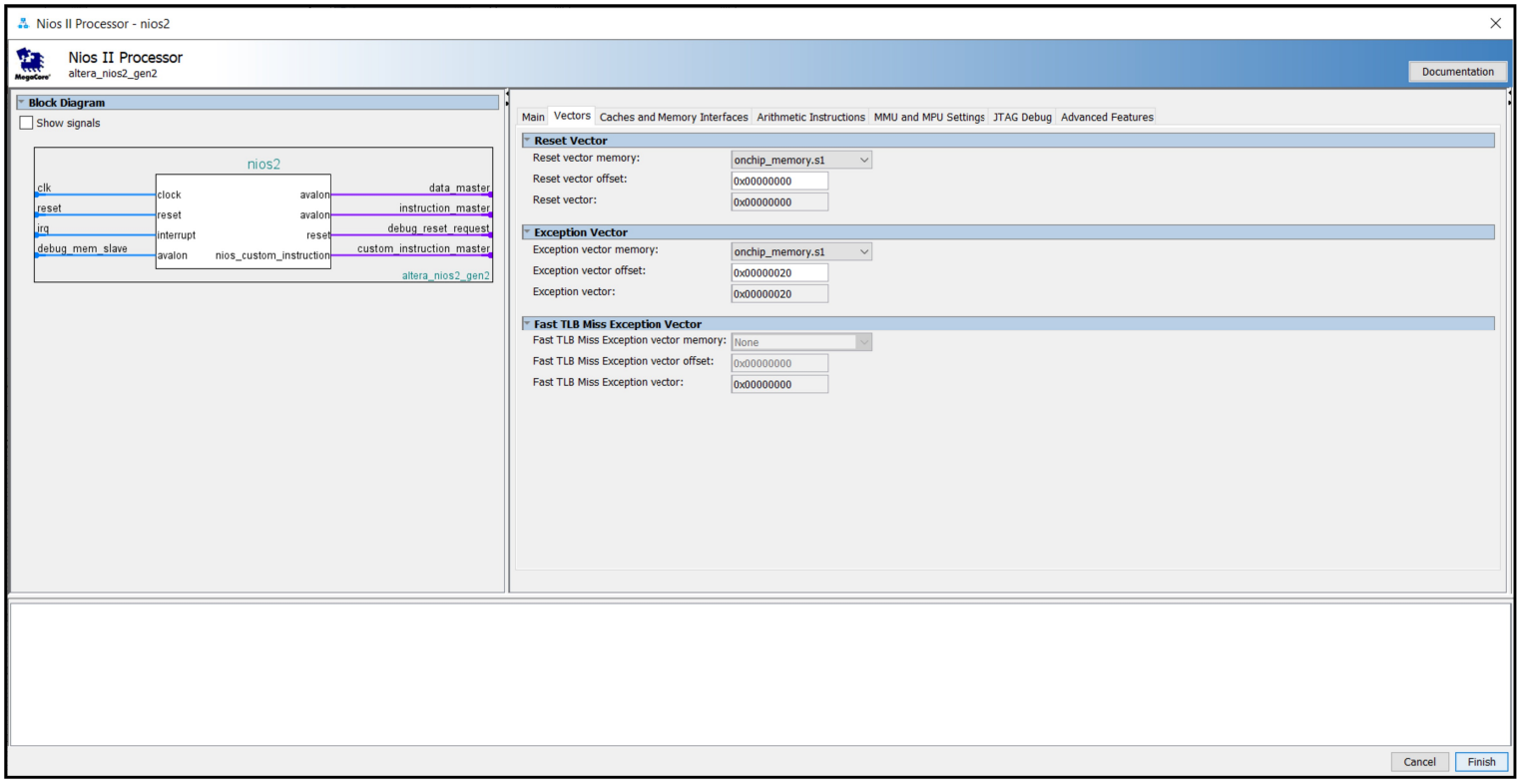


Figure 21

After saving, can now click the **Generate HDL…** button. Before you can generate, again, make sure all errors and warnings are resolved. This will create the HDL files for all the components you just created under one big NIOS II module. You are welcome to look at these files, **BUT DO NOT** edit them unless specified. Also, take **special note** of the **green Info** messages, especially the one pertaining to the **JTAG TCK frequency**, we will address this later. Click finish to exit out of **Platform Designer**.

Now that you have created and generated the NIOS II **.qip** file, it should be in the directory where this project is located. But before you can connect anything, you have to include this **.qip** file into your current Quartus project. Quartus doesn't automatically insert this file into the project when you generate the file. You should get the message window shown in **Figure 22** that reminds you to do this.

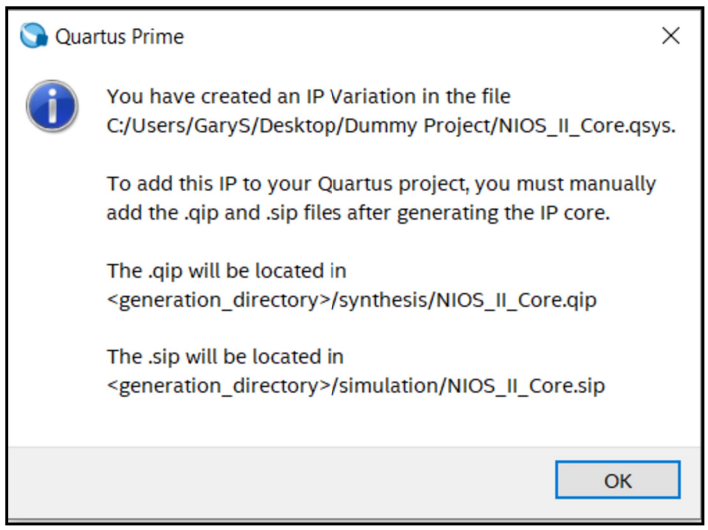


Figure 22

Do this by clicking in the top menu bar: **Project > Add/Remove Files in Project…** and navigating to the **.qip** location. This file should be in the folder you created this Quartus project in, unless you saved it in another directory. **Note:** The **.sip** file isn't necessary for this project and you may not even find it in your project directory. This may be files for ModelSim simulation with NIOS but that is not part of this or any lab you will see.

Connecting NIOS With Verilog Symbol Files

Create a **Block Diagram/Schematic File** by going to the top menu bar click: **File > New… > Block Diagram/Schematic File.** Next, create the symbol files for the keypad_decoder, stepdown_clk, and seven_seg_decoder included verilog modules. To create a **symbol file** click on each verilog module tab in Quartus and do the following for each: **File > Create/Update > Create Symbol Files for Current File.**

Next, add these symbol files to the schematic and also add the newly created NIOS system. Right click anywhere in the **.bdf** file tab in Quartus then click: **Insert > Symbol…** Then find each module and the NIOS file and click on them separately to add them.

Look at **Figure 23** below to see how everything connects. Before compiling make sure to open **Assignments > Device > Device and Pin Options… > Configuration > Configuration mode** and set it to **Single Uncompressed Image with Memory Initialization**, now you should be able to compile the entire Quartus project. See the next section for compilation instructions.

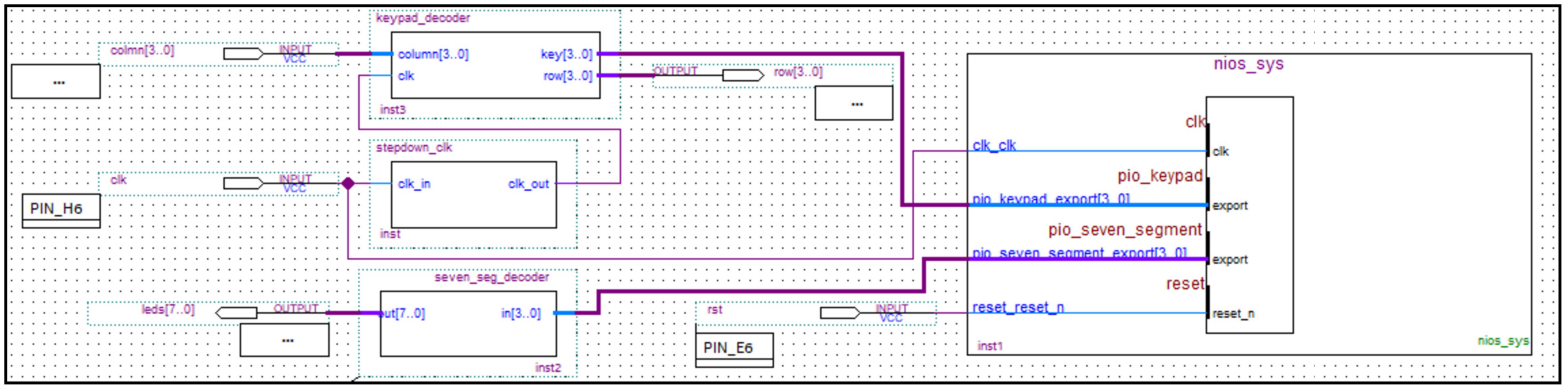


Figure 23

If you have to make changes to any of the verilog files after you create the original symbol file, you will need to update the symbol file again by: ***File > Create/Update > Create Symbol Files For Current File****. Then go to the* ***.bdf*** *file where the current symbol file is located, right click on it, and select* ***Update Symbol or Block.***
If you have to update or change NIOS, you must go back to the ***Platform Designer****, make your changes,* ***save*** *these changes,* ***Generate*** *NIOS again, then lastly* ***Right Click*** *on the* ***nios_sys block diagram*** *in the* ***.dbf*** *file and select* ***Update Symbol or Block...***

Project Compilation and Pin Assignment

A common error that comes up during compilation is the **Top Level Entity** error. Quartus needs a base file to work off of during compilation. There are two ways to fix this: **First,** you can set the entire project as the **Top Level Entity** by clicking in the top bar menu: **Project > Set as Top Level Entity.**

Second, you will have to compile this project twice, if you did not compile the verilog files before creating the **.bdf** file. First, go to the **Files** tab in the **Project Navigator** window (**Figure 24**). **Right click** on the **keypad_decoder.v** and select **Set as Top Level Entity.** Once this compilation is successful, go back to the **Project Navigator** and set the **.bdf** file as the **Top Level Entity.** The **.bdf** file needs to be the final **Top Level Entity**. If this is not the case, then you will have some of the I/O locations missing in the **Pin Assignments** window making it impossible to assign all the necessary pin locations.

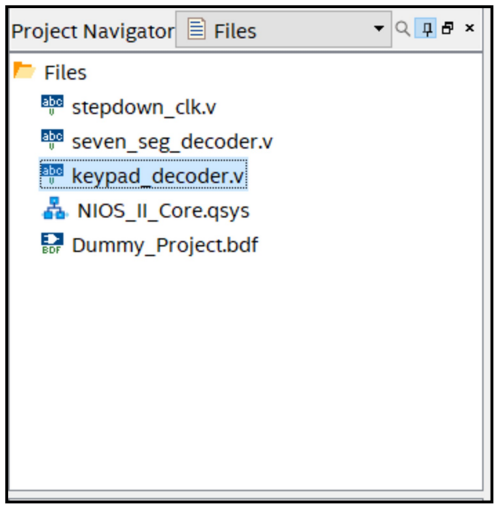


Figure 24

*If any other errors come up, check your **.bdf** file. Usually, there is a missed or wrong bus connection to either a symbol block or the I/O tabs. Make sure to recompile after any changes you have made.*

Now that the project compiles, open the **Pin Planner** under **Assignments** tab in the top menu bar. Connect the clock input to **PIN_H6** (12MHz CLK) and reset input to **PIN_E6** (USER BUTTON) that is on the right side of FPGA in **Figure 25.** ***Note:*** *If the programmer keeps failing try restarting the **Altera JTAG Server**. You can do this in the Windows **Services** application.* ***You will need to ask your instructor for help on this issue if it occurs.***

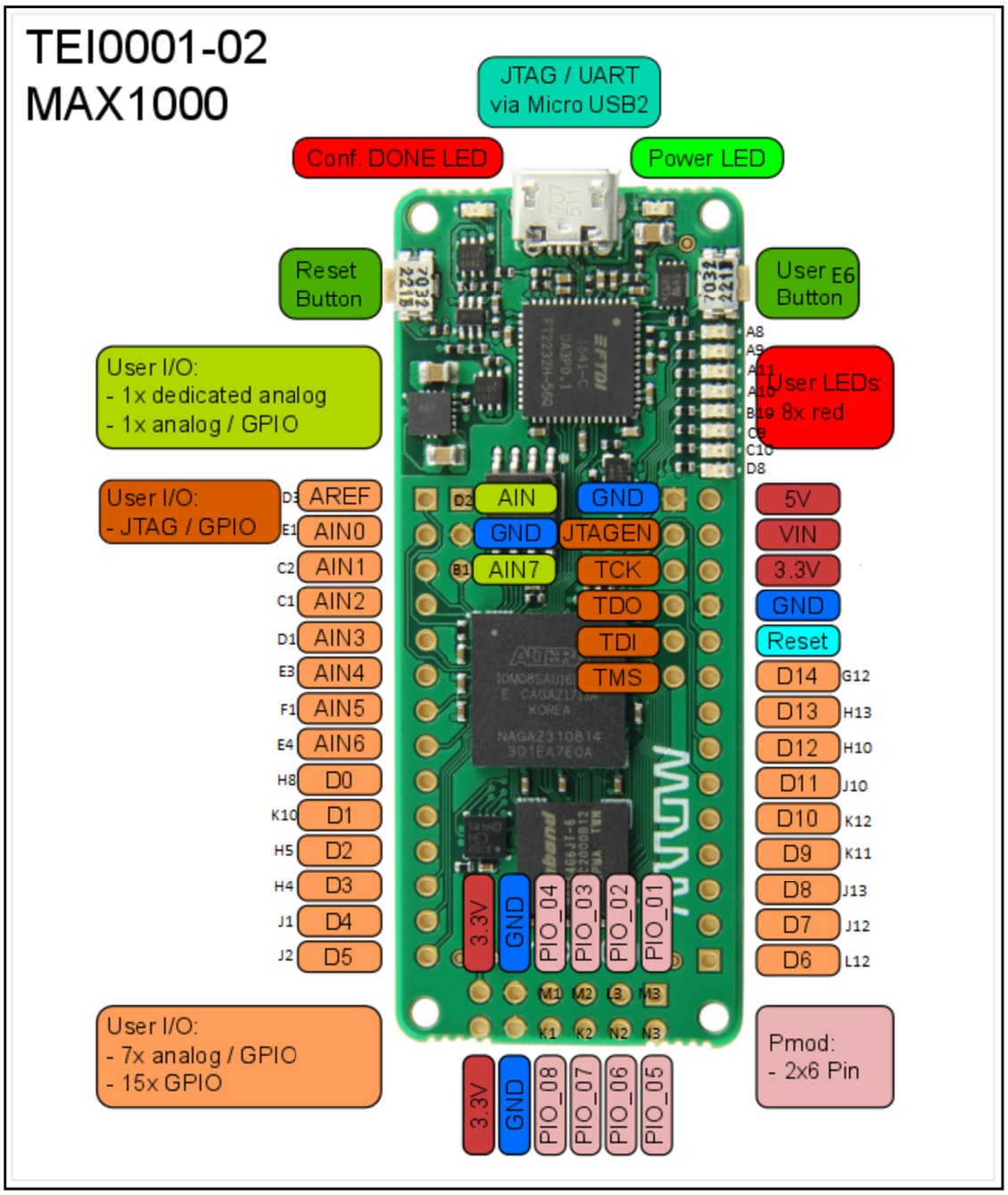
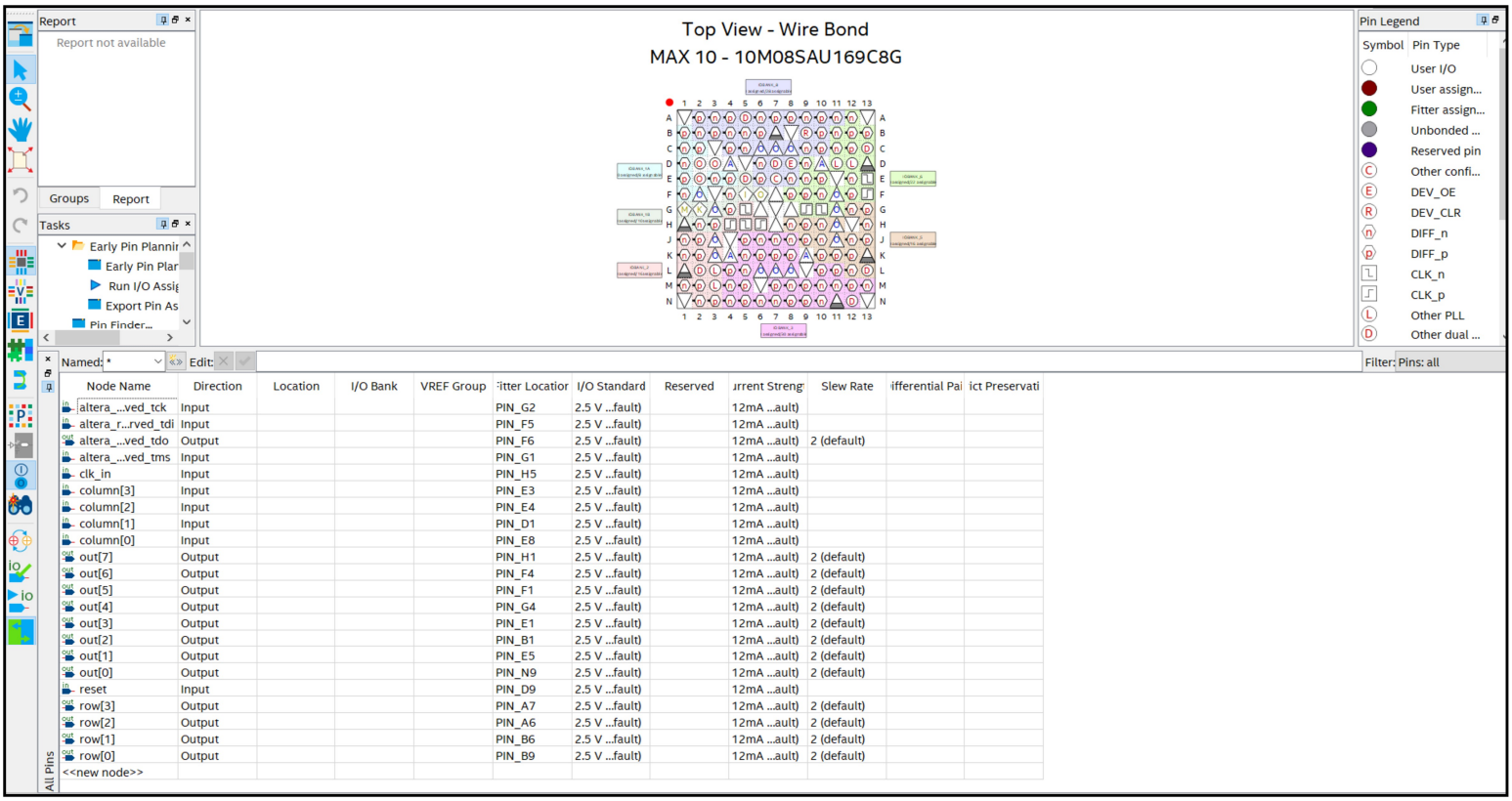


Figure 25

You can choose any of the orange pins for your Digital or Analog I/O. ***Make sure to use the smaller pin labels for the Pin Planner. These are the actual pin names on the FPGA where the orange names are just for your reference. For future reference, if you decide to use one side as analog pins, you can only use that side for analog inputs/outputs. This FPGA doesn't support the use of both Digital and Analog I/O on the same pin rail.*** *Keep this in mind for your own personal projects.*

Now, assign the rest of the **input/output** pins for the keypad and seven segment modules in the **Pin Planner**. See **Figure 26.** You can click and drag each **Node Name** to the pin on the **Top View Diagram.** You should see the set location appear in the **Location** column. Leave the **Altera Nodes** alone, the compilation process sets these pins automatically in the **Fitter Location** column.



Node Name	Direction	Location	I/O Bank	VREF Group	itter Locatior	I/O Standard	Reserved	rrent Streng	Slew Rate	ifferential Pai	ict Preservati
altera_...ved_tck	Input				PIN_G2	2.5 V ...fault)		12mA ...ault)			
altera_r...rved_tdi	Input				PIN_F5	2.5 V ...fault)		12mA ...ault)			
altera_...ved_tdo	Output				PIN_F6	2.5 V ...fault)		12mA ...ault)	2 (default)		
altera_...ved_tms	Input				PIN_G1	2.5 V ...fault)		12mA ...ault)			
clk_in	Input				PIN_H5	2.5 V ...fault)		12mA ...ault)			
column[3]	Input				PIN_E3	2.5 V ...fault)		12mA ...ault)			
column[2]	Input				PIN_E4	2.5 V ...fault)		12mA ...ault)			
column[1]	Input				PIN_D1	2.5 V ...fault)		12mA ...ault)			
column[0]	Input				PIN_E8	2.5 V ...fault)		12mA ...ault)			
out[7]	Output				PIN_H1	2.5 V ...fault)		12mA ...ault)	2 (default)		
out[6]	Output				PIN_F4	2.5 V ...fault)		12mA ...ault)	2 (default)		
out[5]	Output				PIN_F1	2.5 V ...fault)		12mA ...ault)	2 (default)		
out[4]	Output				PIN_G4	2.5 V ...fault)		12mA ...ault)	2 (default)		
out[3]	Output				PIN_E1	2.5 V ...fault)		12mA ...ault)	2 (default)		
out[2]	Output				PIN_B1	2.5 V ...fault)		12mA ...ault)	2 (default)		
out[1]	Output				PIN_E5	2.5 V ...fault)		12mA ...ault)	2 (default)		
out[0]	Output				PIN_N9	2.5 V ...fault)		12mA ...ault)	2 (default)		
reset	Input				PIN_D9	2.5 V ...fault)		12mA ...ault)			
row[3]	Output				PIN_A7	2.5 V ...fault)		12mA ...ault)	2 (default)		
row[2]	Output				PIN_A6	2.5 V ...fault)		12mA ...ault)	2 (default)		
row[1]	Output				PIN_B6	2.5 V ...fault)		12mA ...ault)	2 (default)		
row[0]	Output				PIN_B9	2.5 V ...fault)		12mA ...ault)	2 (default)		
<<new node>>											

Figure 26

Once the pins are set, you can exit Pin Planner (which saves automatically) and **compile the project once more.** You can now launch the Quartus Programmer Tool  and download the configuration to the Max10 as either a **.sof** or **.pof** file. Since licensing requires the FPGA remain connected while using any NIOS IPs, you must leave this window open while using the NIOS. See **Figure 27.** Make sure the **Arrow-USB-Blaster** is set in the **Hardware Setup** window so Quartus can communicate with the Max10. Make sure the check box in the **Program/Configure** column is checked and select **Start.** If the program loaded successfully, you should see the **Progress Bar** filled in 100%,

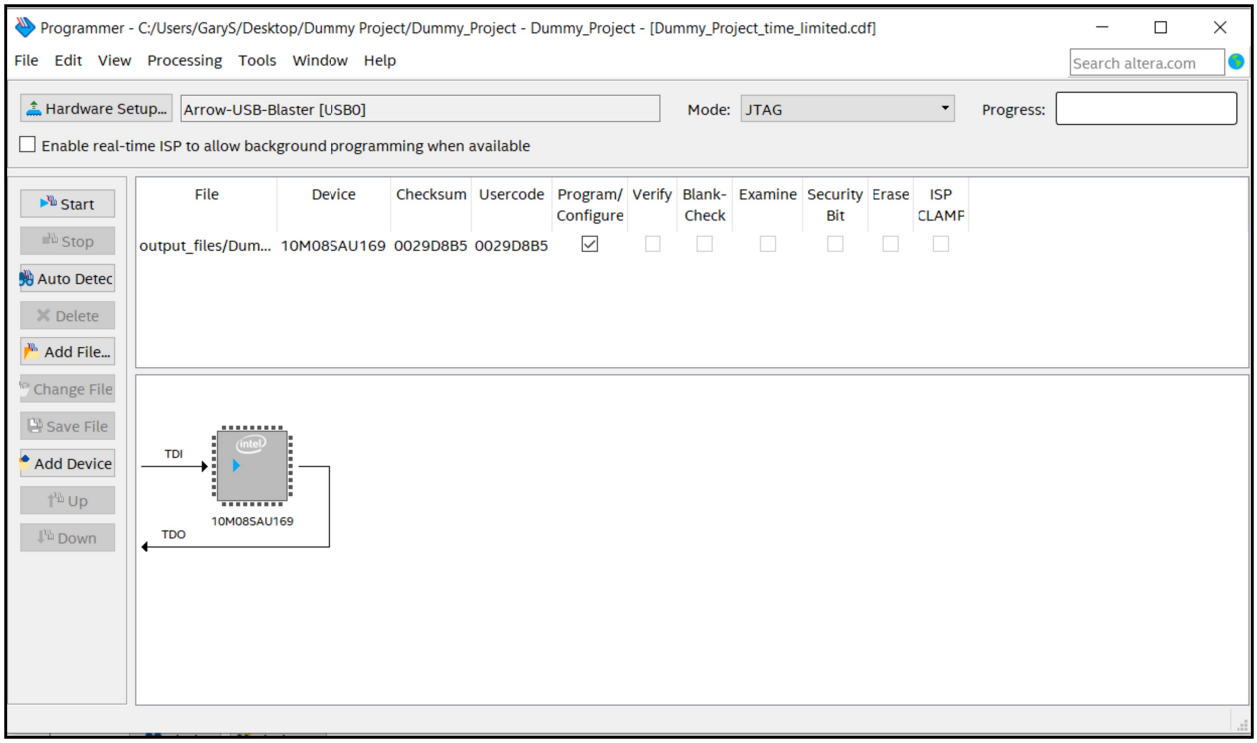


Figure 27

Writing an Application for NIOS:

Now you're going to create the C software layer on top of the hardware abstraction layer you just created and programmed into the Max10. Once you finish with this lab, you can officially say you have developed your first embedded system.
Keep the FPGA connected to your computer and then click on **Tools** in the top bar menu. Then launch **NIOS II Software Build Tools for Eclipse**, select the create a location of the new workspace folder. This will be the folder where your Quartus is located. You should see this window similar to **Figures 28 and 29.**

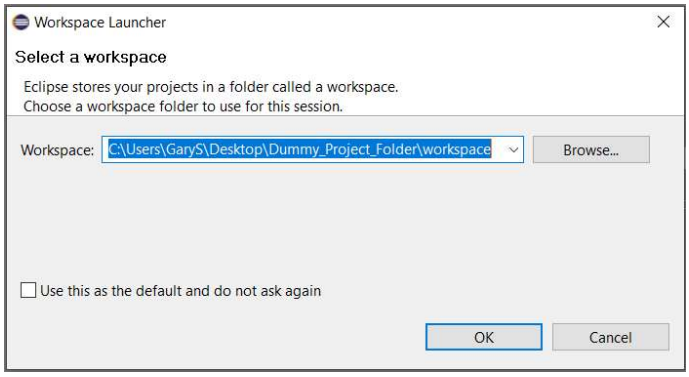


Figure 28

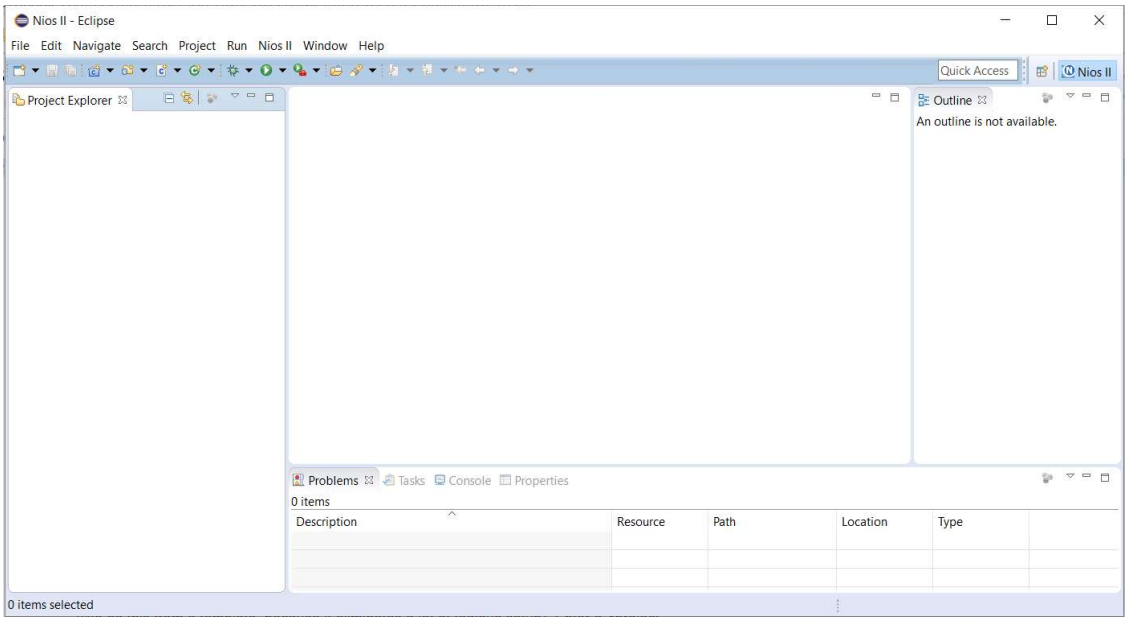


Figure 28

Create a new **NIOS II Application**: **File > New > Nios II Application and BSP from Template** (We do this from a template, because it eliminates a lot of tedious setup). Insert the **.sopcinfo** file in the **SOPC Information File Name** slot. This file contains all the hardware configuration information Eclipse needs to be compatible with this Max10. Next, enter a <project name> in the project name field, select "**Hello World Small**" in **Templates** and click **Finish**.

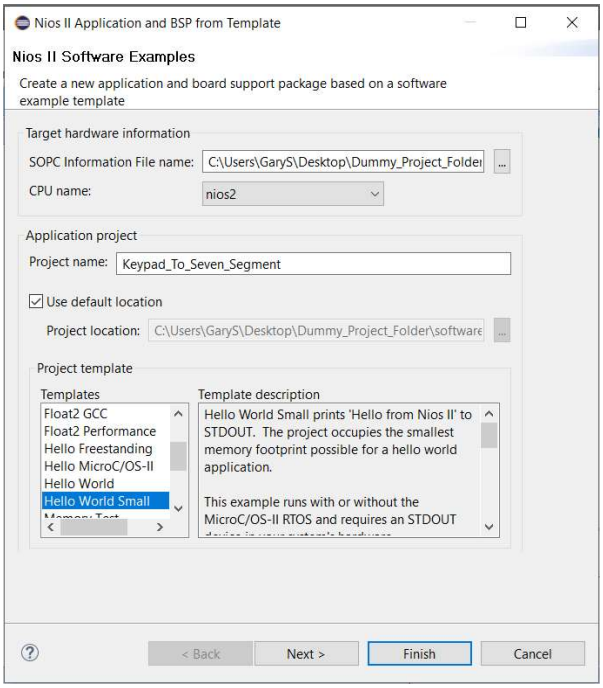


Figure 29

This will generate two projects **Keypad_To_Seven_Segment** and **Keypad_To_Seven_Segment_bsp** in the **Project Explorer** window like in **Figure 30.**

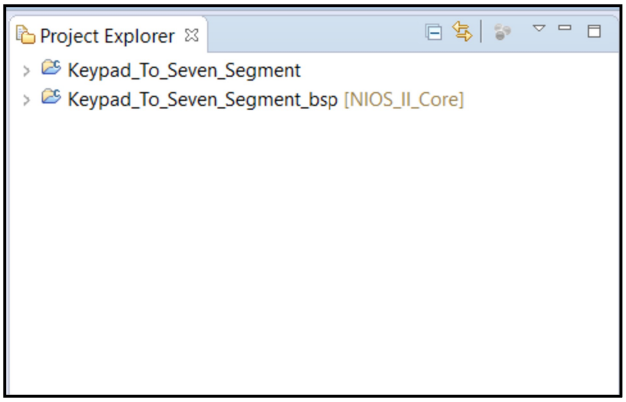


Figure 30

Right Click Keypad_To_Seven_Segment_bsp > NIOS II > Generate BSP (Wait for this to finish). Now you can **Right Click Keypad_To_Seven_Segment** and select **Build Project**. Make sure everything builds correctly before writing your own application. If no errors occur, you should see something similar to **Figure 31** in the **Console. NOTE:** If you happen to build your project before generating the BSP, the program will come up with some errors when it tries to run. To fix this, **Right Click** on the **Keypad_To_Seven_Segment_bsp** and select **Clean Project.** Next do the same to the **Keypad_To_Seven_Segment.** Then generate the **bsp** again and then **build** the **Keypad_To_Seven_Segment**.

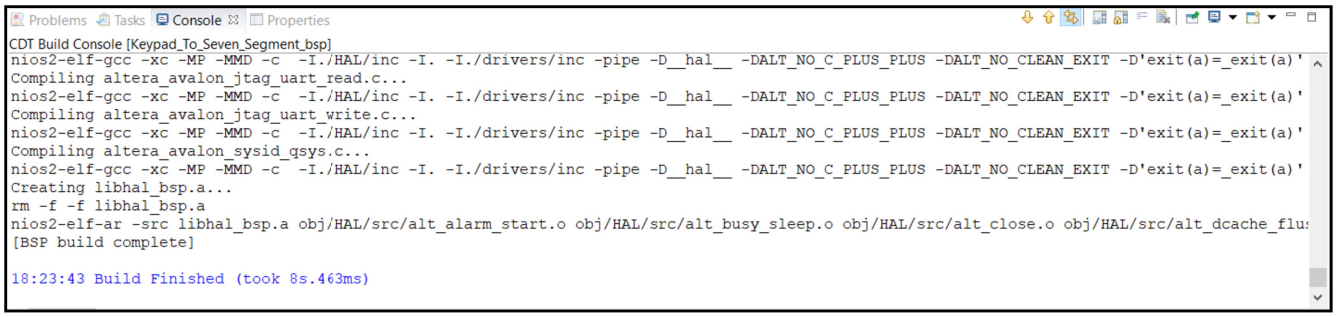


Figure 31

Writing the C Program for NIOS:

Since we built this off a template, the main C program you will edit will be located in the **Project Explorer** under <project name>. as **helloworld_small.c**, feel free to change this name to what you like by **Right Click > Rename**. **Double Click** this file to open it.

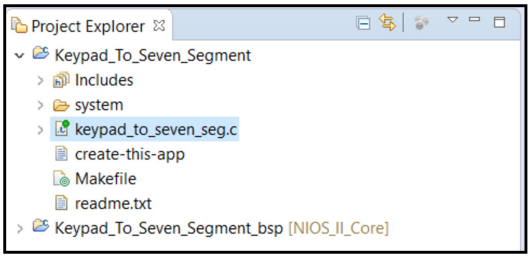


Figure 32

main.c

```
 * FOR: TEXAS STATE UNIVERSITY STUDENTS AND INSTRUCTOR USE
*/

/*
 * This simple program reads the input pins from the keypad and then writes
 * this input into an address in NIOS'system memory register.
 * Then NIOS will write this value from memory out to the
 * seven segment decoder module. Where it will send the output
 * to the pins that you assigned.
*/
//including the libraries needed for the project to run
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"

int main()
{
    // Variables used to capture incoming data
    unsigned char key = 0;
    unsigned char old_key = 0;

  // Main loop to check for keypad input
  while (1)
  {

    /*
     * Instead of PIO_KEYPAD_BASE, you could input the exact address value
     * You will find this 0x_______ address in the Platform Designer under
     * the Base column.
     * Example: key = IORD_ALTERA_AVALON_PIO_DATA(0x2010, <data>);
     *
     */

      // Read value at PIO_KEYPAD_BASE address into key variable
      key = IORD_ALTERA_AVALON_PIO_DATA(PIO_KEYPAD_BASE);

      // Write key variable to PIO_SEGMENT_BASE address
      IOWR_ALTERA_AVALON_PIO_DATA(PIO_SEVEN_SEGMENT_BASE, key);

      // Check if new key has been pressed
      if(old_key == key) continue;
      else
      {
          // Update the key value and print it to the Console
          old_key = key;
          printf("%i\t", key);
      }
  }
  return 0;
}
```

Figure 33

Now change the template and type in the code in **Figure 33.** Looking at the code in **Figure 33**, we can see it checks data in the **PIO_KEYPAD_BASE** memory location and stores that value into the key variable. Values are provided by the keypad decoder, and required for the seven segment decoder are only 1 nyble, so we will use unsigned (doesn't have to be unsigned) chars since they are only 1 byte (No need to waste memory).

The next line will write the key value to **PIO_SEVEN_SEGEMENT_BASE**, which is connected to the seven segment decoder. Lastly, the code checks to see if a new key was pressed, if so display it to the console. **Note: these PIO<module name>_BASE defines are located in <project name>_bsp under system.h.** You will need to use these because they let the **IORD & IOWR Altera Avalon PIO Data functions** know where to read and write data. Do not edit anything in the BSP project, it will more than likely break something if you do.
Before you download your C program application to the FPGA, you must lower the JTAG TCK. Navigate to ..\intelFPGA_lite\18.1\nios2eds and launch **Nios II Command Shell.bat**. This will more than likely be located in the **C\:** directory. Ask your instructor if you cannot find it on your lab computer.

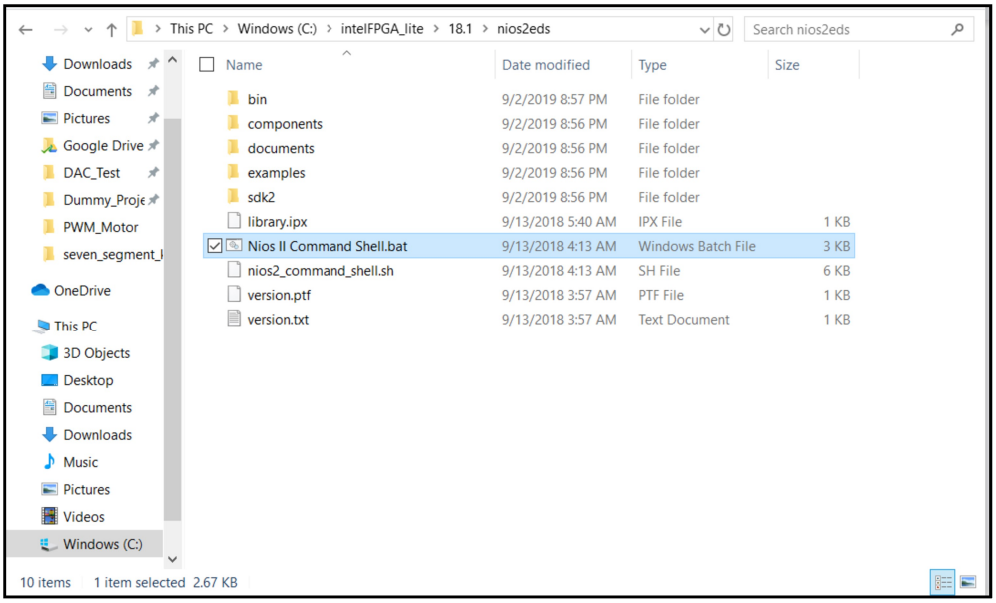


Figure 34

Find the number of the USB connector by typing in the command ***jtagconfig -n***, in most cases it's 1. Now type **jtagconfig --setparam 1 JtagClock 5M** to lower the jtag test clock speed (**Figure 35**). Then type **exit** to close the console.

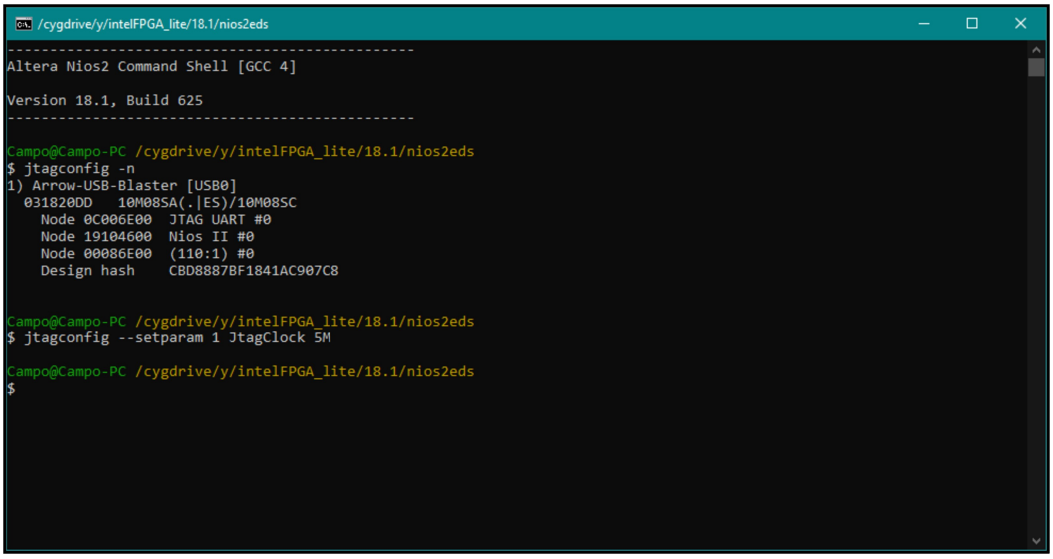


Figure 35

The reason you have to become a console wizard is because of the fact that most development boards come with a 50MHz clock or faster. This Max1000 board has a 12MHz clock instead. So the Jtag TCK, which is the Jtag clock, needs to be less than half of the internal clock of the device in order to run correctly. So you have to set it to 5MHz to meet the software system requirements.

Back in Eclipse, **Right Click** <project name> and **Run As > NIOS II Hardware**.

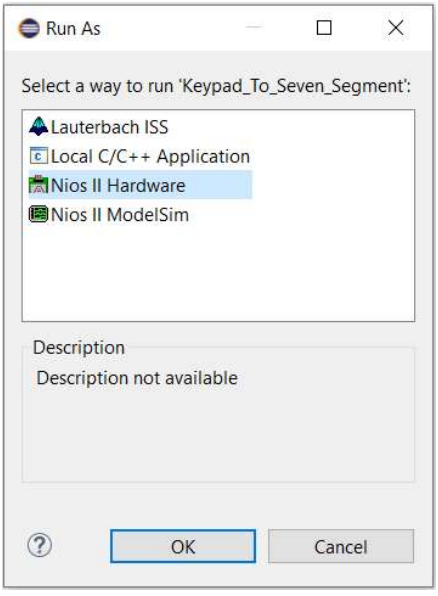


Figure 36

The **Run Configurations** window will come up as shown in **Figure 37.**

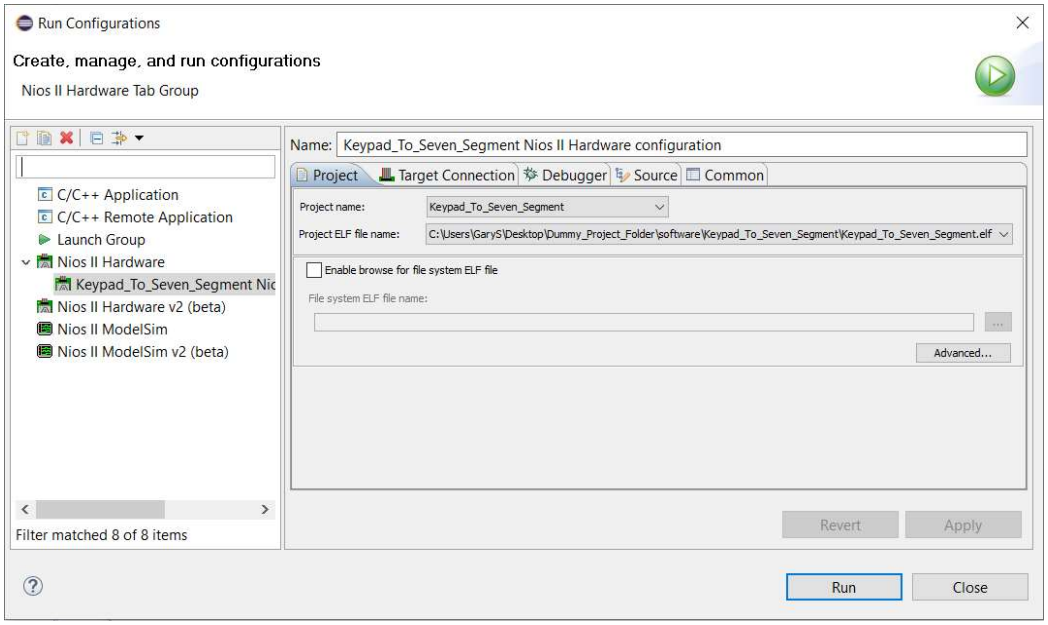


Figure 37

Under the **Target Connections** tab click **Refresh Connections** and the Arrow-USB-Blaster Cable should populate. If the **Run** button does not become available, check off **Ignore mismatched system ID** and **system timestamp**. See **Figures 38.** Then click on **Refresh Connections**.

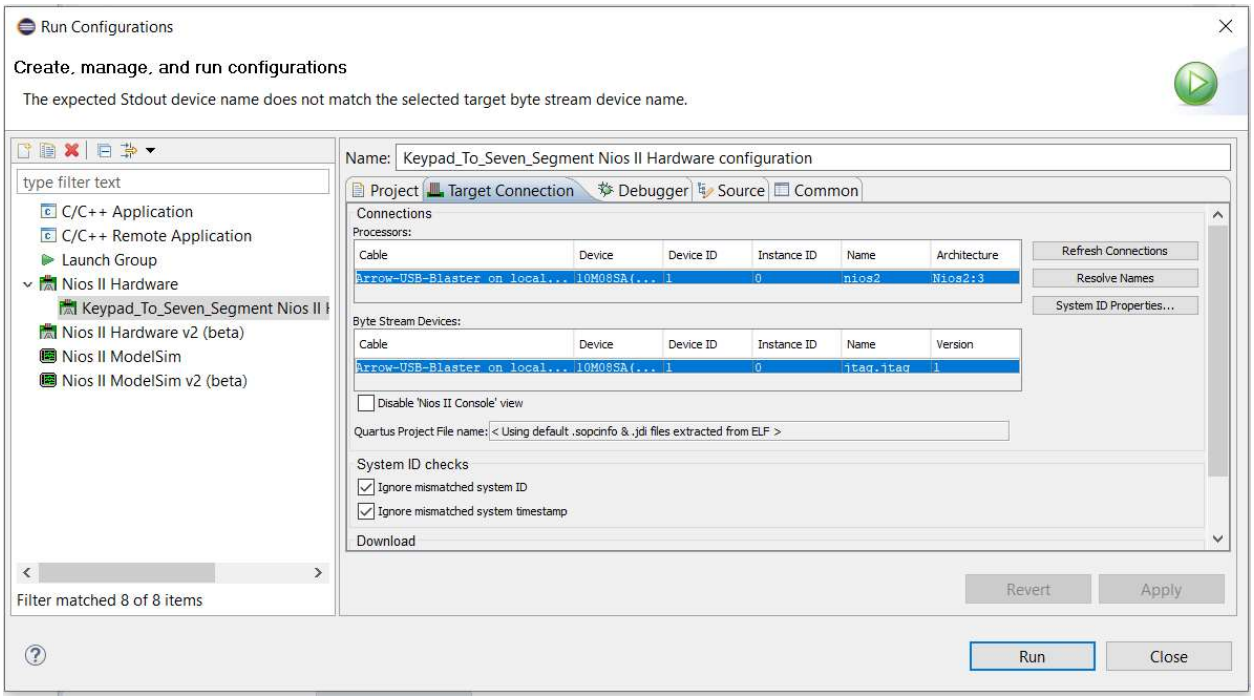


Figure 38

Lastly click **Apply**, then **Run**. If everything is successful you will see something similar in the **Console** output (**Figure 39**).

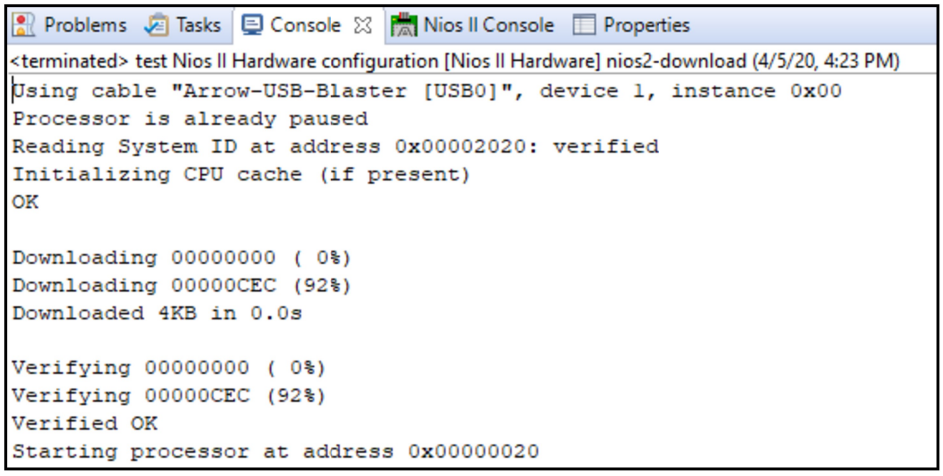


Figure 39

If all was done correctly you should be able to press any key and view it in the **NIOS II Console** in **Figure 40** and in **Figure 41** the seven segment when pressing the "*" key!

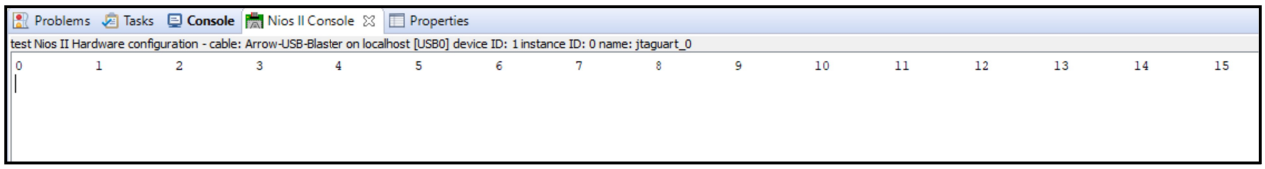


Figure 40

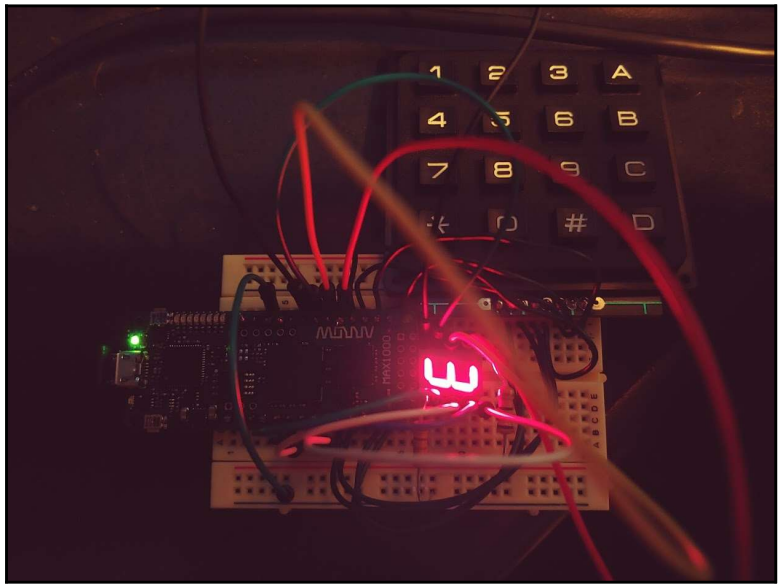


Figure 41

References:

[1] “Membrane Keypad 4X4 Matrix,” *ProtoSupplies*. https://protosupplies.com/product/membrane-keypad-4x4-matrix/ (accessed Apr. 12, 2020).
[2] “Interfacing hex keypad to 8051. Circuit diagram and assembly program. Simple circuit using minimum components,” *Electronic Circuits and Diagrams-Electronic Projects and Design*, Mar. 08, 2013. http://www.circuitstoday.com/interfacing-hex-keypad-to-8051 (accessed Apr. 12, 2020).
[3] “Seven-segment display,” *Wikipedia*. Apr. 06, 2020, Accessed: Apr. 12, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Seven-segment_display&oldid=949464691.