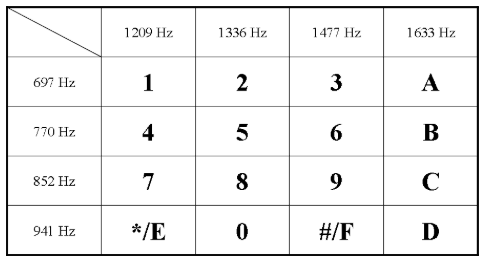
**EE 3420 Lab Guide 2: Keypad to DTMF w/ NIOS**

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

**Example Overview:**

Design a circuit that will take in a keypad input and output to two tone frequencies to a pair of buzzers. We will do this by having the NIOS II softcore CPU take in data from the keypad and send it out to a selector module. Depending on the button pressed the selector module will output the correct tones to a set of buzzers. The tone duration will be controlled by the NIOS by enabling a pio connection to the selector module.

**

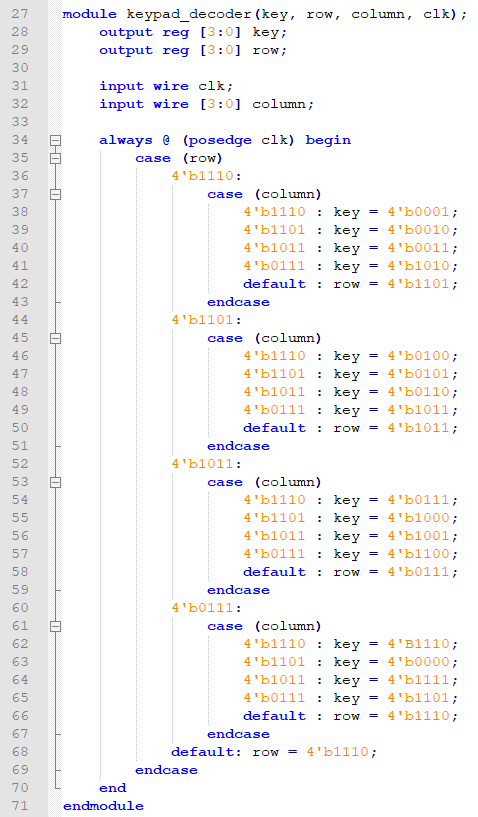
**Figure 1**

*Though these are the frequencies used in this lab, you could actually set up your own frequency values as well. Though you may need to find a speaker set that could handle frequencies under 1000Hz or above 2400Hz.*

**This guide will show you how to set up the tones for keypad row 0. It will be up to you to create the rest of the tone clock divider modules and connect them to the full selector module.**

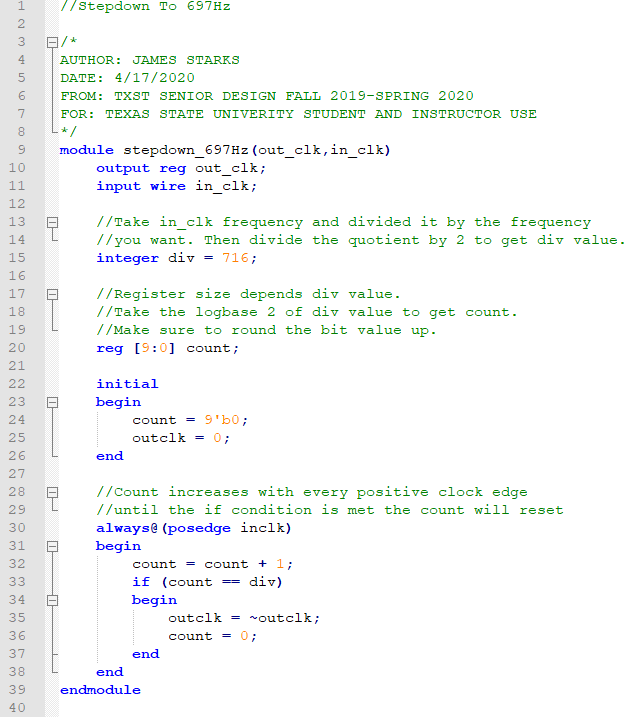
**Verilog Code Breakdown:**

To begin, you will need the keypad module from the last lab. But we have provided it for you here in **Figure 2.**



**Figure 2**

Next we will need to create the tone clock divider modules for row 0. We will take an input clock signal of 12Mhz, put it through a Phase Lock Loop (PLL) to step this signal down to 1Mhz, then write custom modules to step down to the frequencies we need. Basically, we are creating a set of clock converters. We will cover the PLL in the **FPGA Implementation section**, lets first create the clock converters.



**Figure 4**

*You will need to create this step down code for each frequency specified for the lab. It’s tedious but that is why copy and paste exists.*

The first thing you need to know is how many divisions out of the 1MHz clock signal it takes to reduce it down to 697Hz. Your numbers may vary from the guides because we tried to get the tones as close as possible to real phone dial tones.

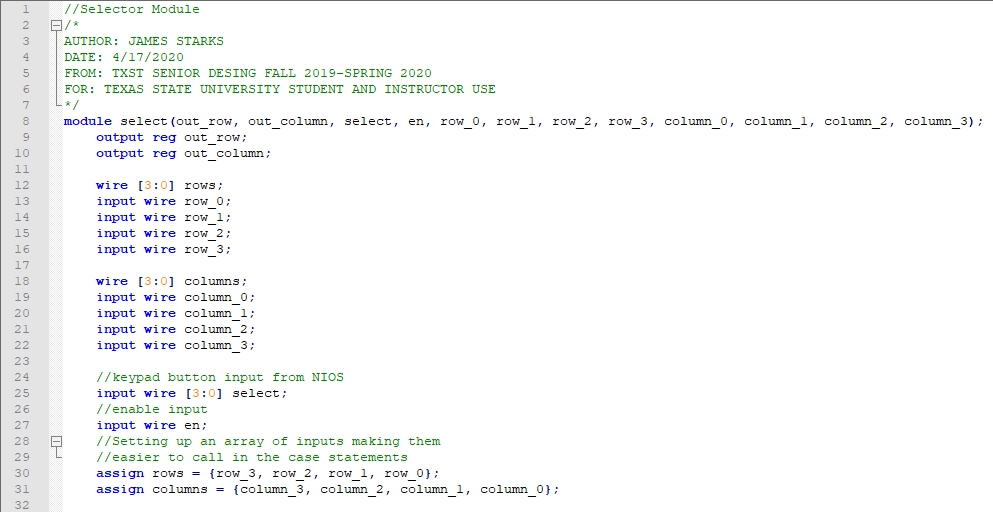
*The 2 in the divisor takes into account that for every clock cycle, there is a positive edge and a negative edge. Since the always block is running on just the positive edge, the code activates at half the clock cycle.*

Next you will have to calculate the number of bits that the counter register has to store in order to reach the division value.

*Make sure to round up your answer for the bits because you can’t store a decimal value of bits, only whole bits.*

**Now go ahead and make modules for the rest of the frequencies as shown in Figure 1. Once you are done creating those modules, you will need to make a 60Hz module. This will be for the keypad and it will be explained later why you need it.**

Next we will need to create a selector module that will have all the clock divider modules, the keypad, and an enable as input and will output the two tones needed to create the tones we want. Then in the next section we will connect everything together with NIOS. **See Figures 5-7** for the selector code.



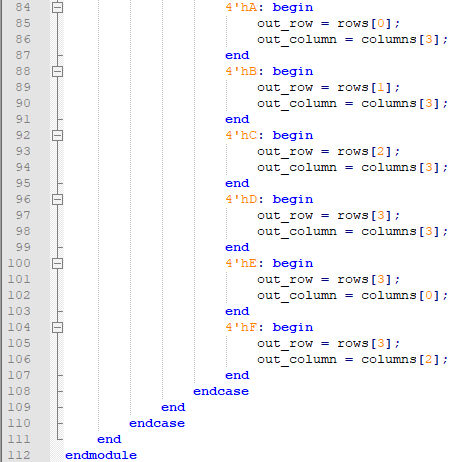
**Figure 5**

*This module is quite large but using case statements is the most efficient in hardware instead of if/if-else/else statements.*

The selector takes in all the clock converter inputs and waits for the enable and keypad value from NIOS. Once the enable is triggered, the keypad value is read and the row and column tones are sent out to the two buzzers. NIOS will flip off the enable after a set amount of time so buzzers don’t run continuously. More on this later.



**Figure 6**

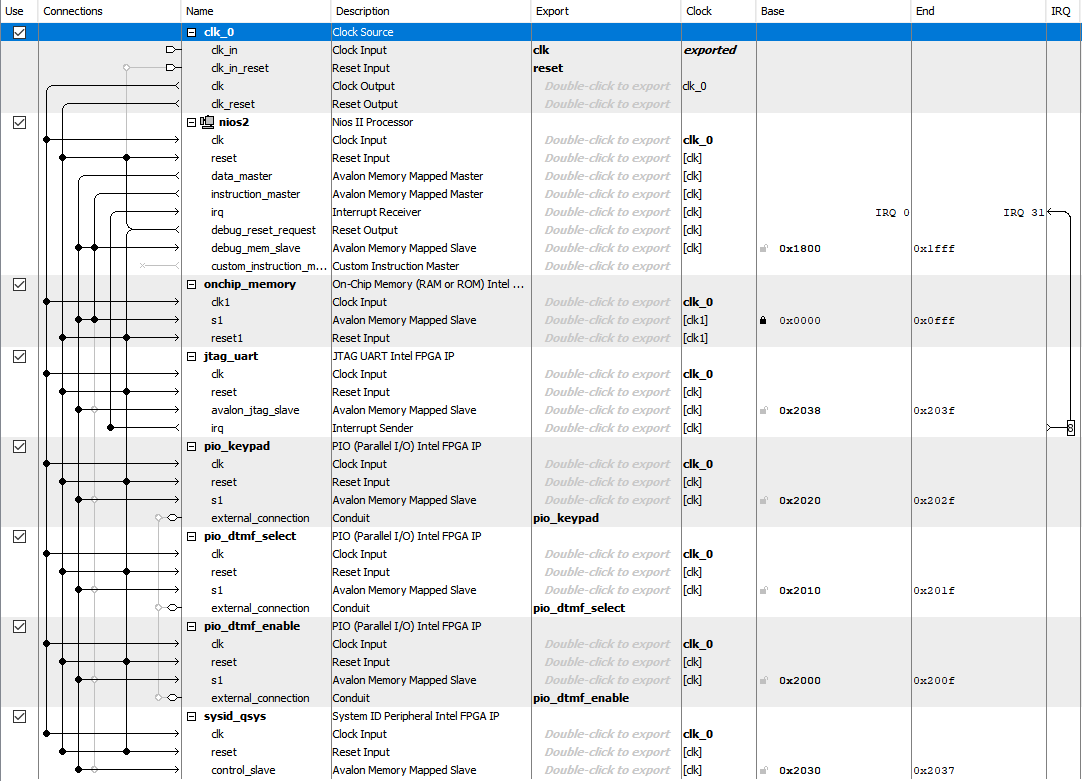


**Figure 7**

**Once all your modules have been made or copied into Quartus. Make each one into a Symbol File and then continue onto the FPGA Implementation section.**

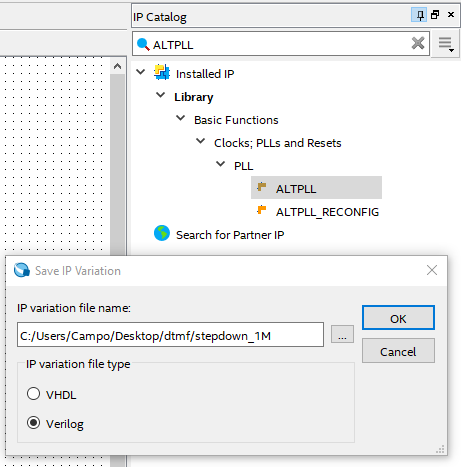
**FPGA Implementation:**

For this project you will need to create a NIOS platform with 3 **PIO** modules, one for a **4-bit** input from the **keypad decoder** and another 4-bit output to the **DTMF tone selector**. The final **PIO** will be used to control the **enable** pin on the **tone selector**. See **Figure 5** to see how to configure the Platform Designer with the NIOS processor. Make sure to resolve all errors and warnings before generating.

**Figure 5**

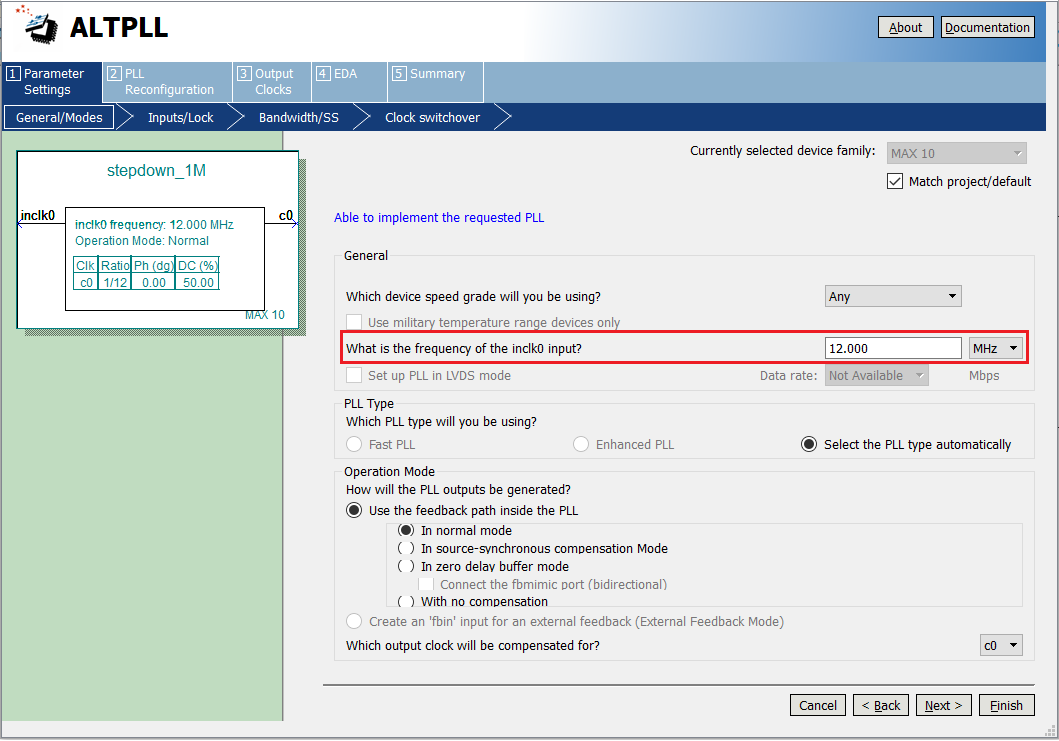
Now we must add a PLL to the block diagram to help with tone generation. We will use a built in PLL to drop the clock from 12MHz to 1MHz. This will help improve the reliability of the clock dividers that will be used to generate a tone.

Search in the IP catalog for “**ALTPLL**” and launch the wizard. Follow the images below to see which settings to apply.



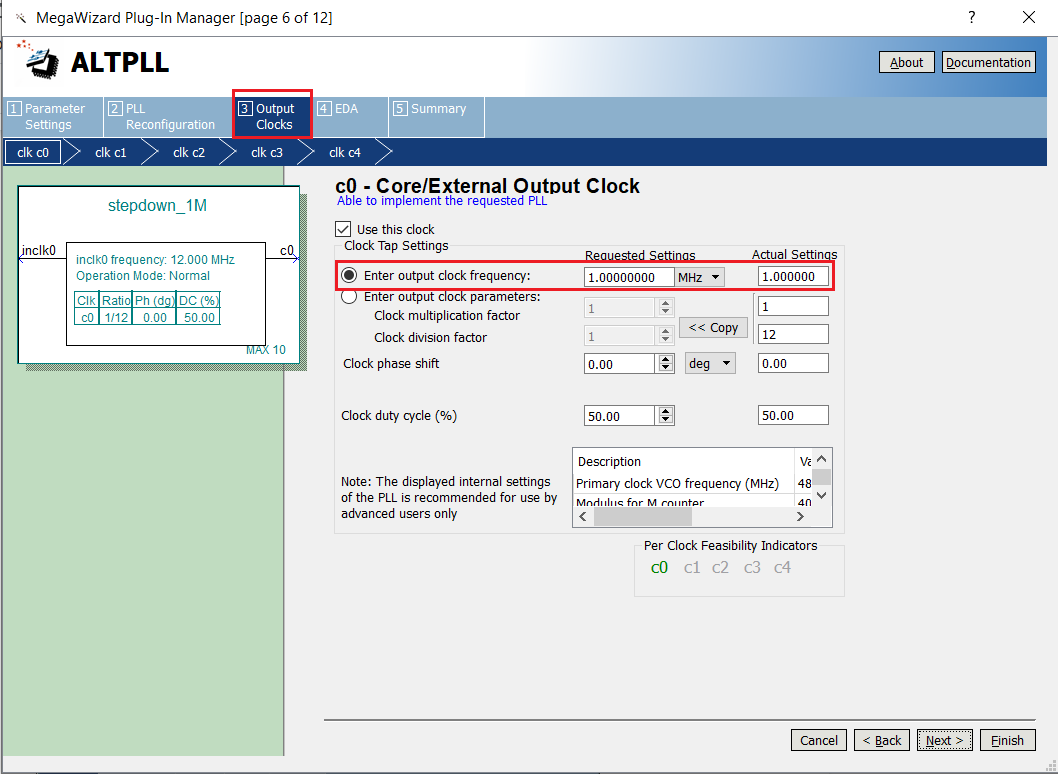
**Figure 6**

**Figure 7** will be the first window you will see. For the PLL, you will need an input clock signal. This will be set to 12Mhz because we will pull from the Max1000’s on-board clock.

****

**Figure 7**

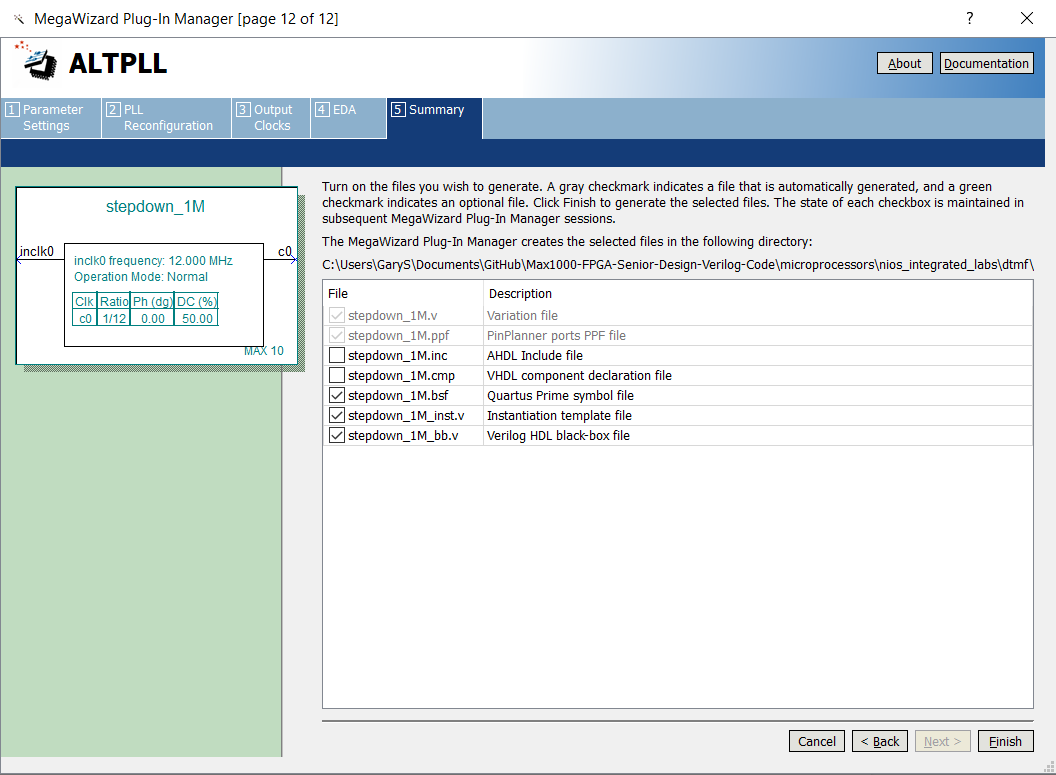
Next you will need to set the output clock. For this lab, it will be 1MHz. As a note, the lowest the PLL can step down to is 10kHz. You are welcome to drop it down to 10KHz but you will have to recalculate your values for your clock converter modules. To set the output, select **Output Clocks** tap at the top, select the **“Enter output clock frequency”,** and type in the desired frequency as shown in **Figure 8.**



**Figure 8**

*A single PLL module can have up to 5 different clock outputs. PLLs are very useful if you need a very stable clock frequency. They can also be used to step up the original on-board clock frequency but for this lab, this is unnecessary.*

To finish, click on the **Summary** tab at the top of the PLL menu and make sure the three file options are selected as shown in **Figure 9.** Then select **Finish.**

****

**Figure 9**

*The PLL manager will create all the necessary files including the symbol file needed to use the PLL in the* ***Block Diagram file (.bdf).***

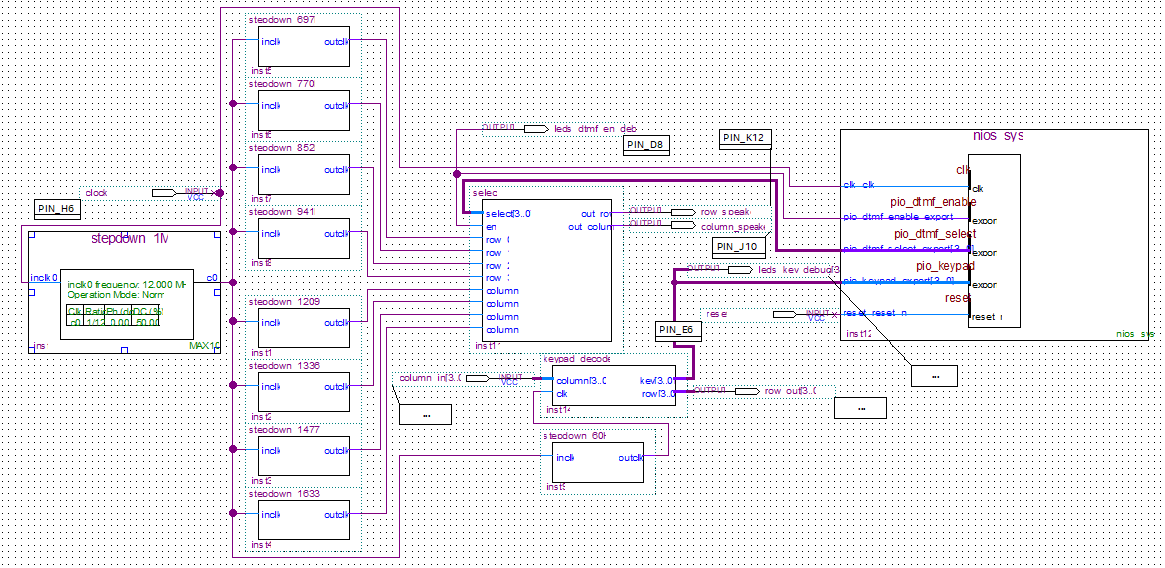
Here is a list of files you will have to **include** in your project and **create symbol files** of...

* select.v
* keypad\_decoder
* stepdown\_60Hz.v
* stepdown\_697Hz.v
* stepdown\_770Hz.v
* stepdown\_852Hz.v
* stepdown\_941Hz.v
* stepdown\_1209Hz.v
* stepdown\_1336Hz.v
* stepdown\_1477Hz.v
* stepdown\_1633Hz.v

Now you will need to create a **Block Diagram** file so you can drop in all of your symbol files. You will first start off by connecting the output of the PLL into the input of each separate verilog clock divider. Then the output of each clock divider will go into the selector row and column inputs. Each of these stepped down frequencies will then be sent to the selector module which uses the output from the NIOS to select two frequencies to play.

The 60Hz clock divider module will be connected to the keypad module. This will allow the keypad to scan for any button inputs 15 times a second. The keypad module has some issues running at higher frequencies so it is best to run it under 100Hz.

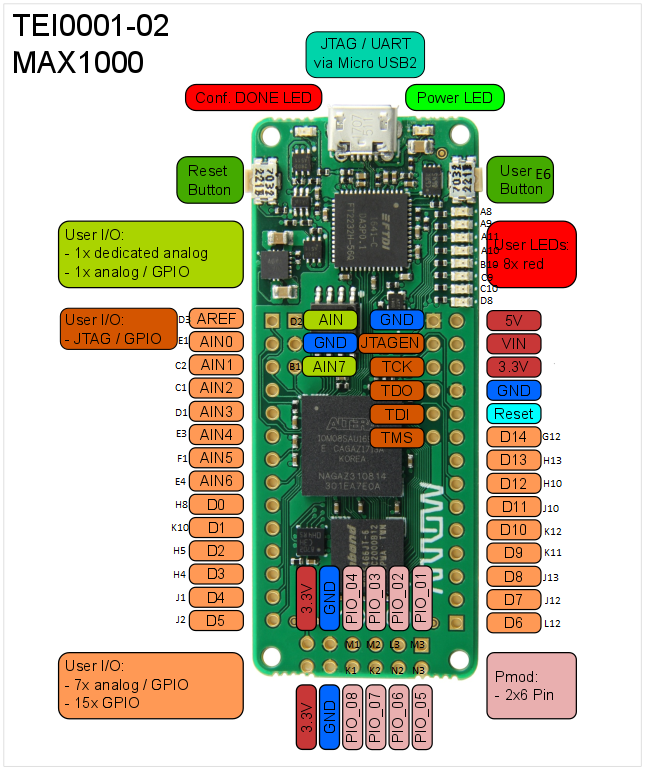
Now go ahead and add **NIOS** into your **Block Diagram** file. Connect the keypad[3..0], select[3..0], the 12MHz clock and reset to NIOS. Then connect the **DTMF enable** from NIOS to **en** input on the selector. Use **Figure 10** as your connection guide.

**Figure 10**

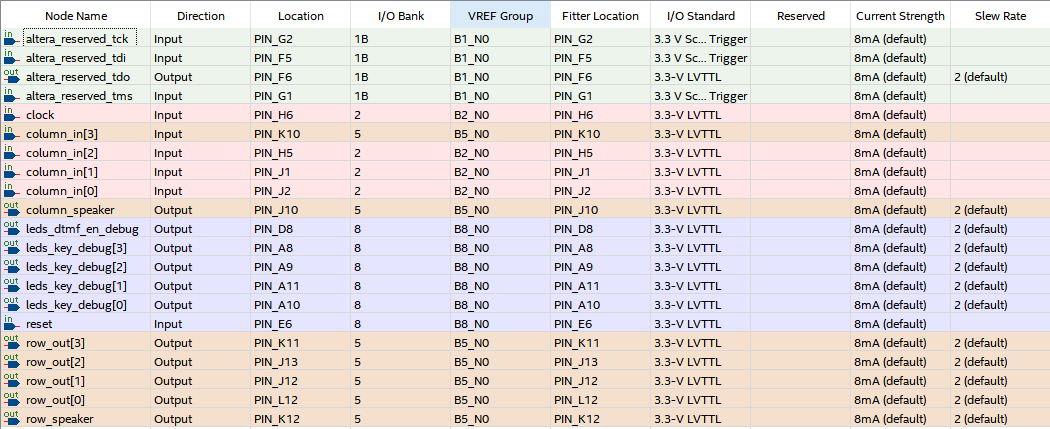
Before compiling make sure these additional Quartus settings are applied…

* **Assignments>>Device>>Device and Pin Options>>Configuration** set **Configuration mode** to “**Single Uncompressed Image w/ Memory Initialization**.”
* Also in the **Device and Pin Options** window set **Unused Pins** to “**As input tri-stated**,” and under **Voltage**, **Default I/O standard** to “**3.3-V LVTTL**.”
* **Assignments>>Settings>>Operating Settings and Conditions** and set both **VCCA voltage** and **VCC\_ONE voltage** to “**3.3V**.”

Now, run the Quartus compiler to generate input/output nodes in the **Assignments>>Pin Planner** so you can now set the **pin locations**. Use **Figure 11 and 12** as a guide to set your pins. It may prove useful to look back at the Keypad to Seven Segment Lab and review how the keypad is connected. Note: the 12MHz Clock is located on PIN\_H6. Below in **Figure 12** is an example of a completely planned out pin assignment.



**Figure 11**

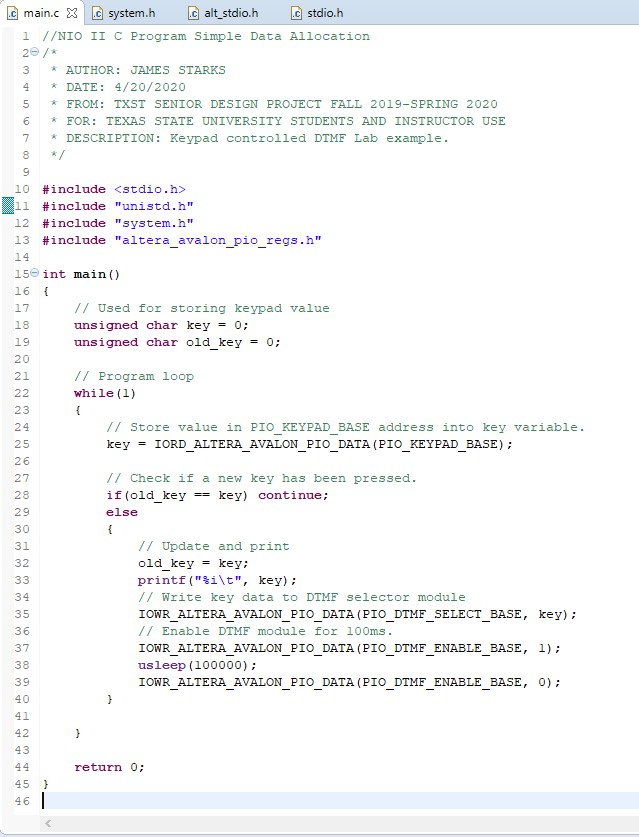
**Figure 12**

After setting pins, recompile the project and a **<project\_name>\_time\_limited.sof** file will generate in the **/output\_files** directory. This is because the current Quartus license requires the FPGA to remain connected to the computer when NIOS IPs are running on the FPGA.

**NIOS II Setup with Eclipse:**

You will find the software side of the project is very similar to the Keypad to Seven Segment Display Lab. Like before the program will read data from the keypad decoder module, store it, then pass it on to the selector module. But now you will need to switch the enable high for the tone to play. In the sample code below, **Figure 13,** the selector was enabled for 100ms and then turned off, so the tone isn’t played continuously.

Like before, start to create a **New NIOS II Application and BSP from template** and select the **Small Hello World** template. Next generate the **BSP** project and you can start editing the application project. **Would recommend going back and following the steps in the Keypad to Seven Segment if you are having trouble with the Eclipse setup.**



**Figure 13**

Finally, **build** your application project and then run as **NIOS** **II Hardware**.