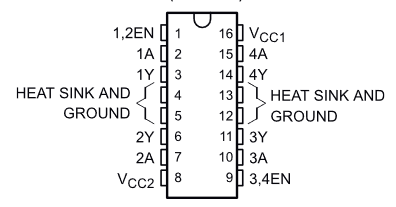
**EE 3420 Lab Guide: Motor Drivers w/ NIOS II**

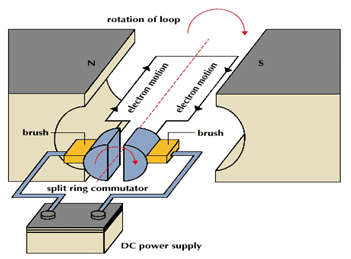
*Written by: Grant Seligman, Gabe Garves, and James Starks*

**SN754410 Quadruple H-Bridge**



**Figure 1** [[1]](https://www.zotero.org/google-docs/?D3zkG5)

**Part 1: DC Motor Driver**



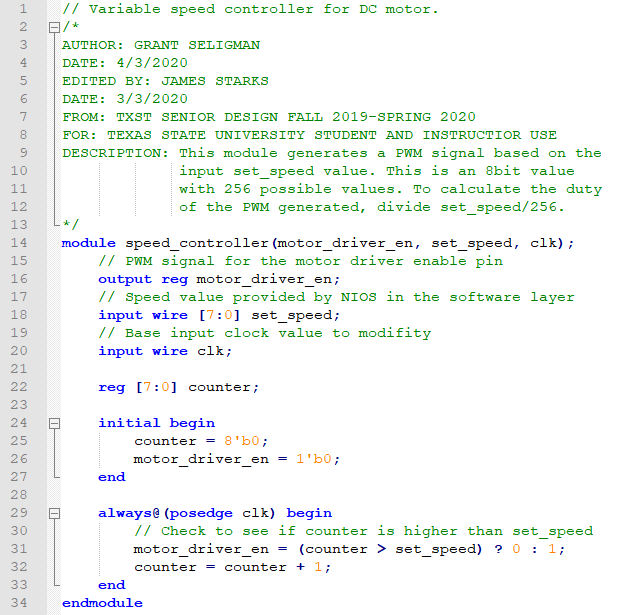
**Figure 2** [[2]](https://www.zotero.org/google-docs/?8c07R4)

**Example Overview:**

Implement and design a system for the Arrow Max1000 board controlling the SN754410 Half-H to drive a DC motor. NIOS will be used as the controller for all the all the Verilog modules. We will look at both motors projects separately in this guide.

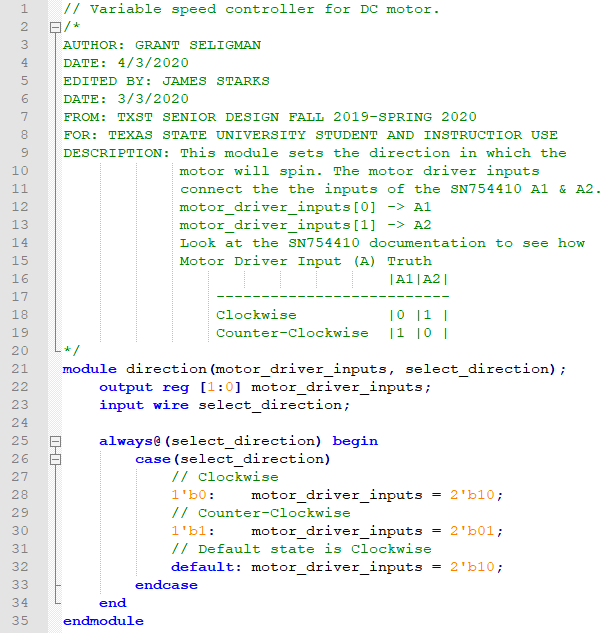
**Verilog Breakdown:**

Looking at **Figure 3** you can see the Verilog code that is used to control the speed of a DC motor. The speed controller is basically a PWM signal that, in this case, allows an external hardware module to control the duty cycle. The code works by iterating through a counter, and when the counter is greater than the set\_speed, the state switches low. Due to the counter being 8-bits wide, and the input clock frequency being 256kHz, this drops the PWM signal frequency to 1kHz. This is because for one cycle to complete, count must iterate 256 times.



**Figure 3**

**Figure 4**, which is the direction controller, is a simple 1-bit state machine. It checks for a 1 or 0, and depending on the student selection, the state outputs of 10 or 01 will be sent to the input pins on the SN754410. This flips the direction of through the DC motor causing it to switch directions.

****

**Figure 4**

**FPGA Implementation:**

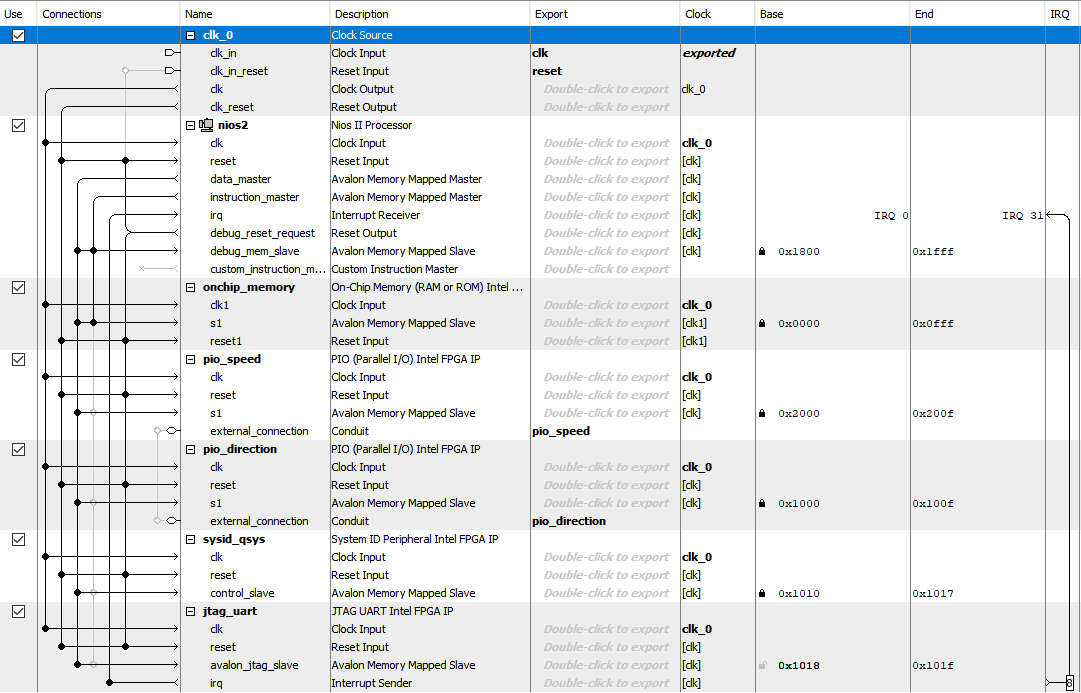
First you will need to create a new Quartus project and include these files…

* **speed\_controller.v**
* **direction.v**

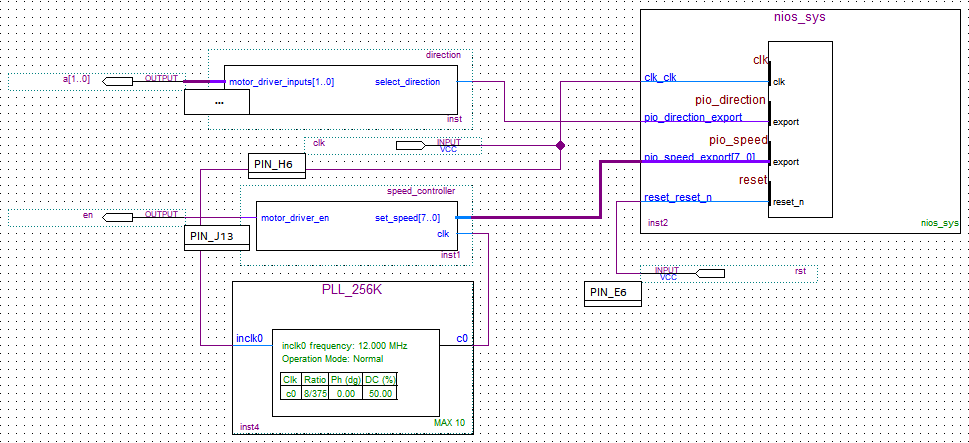
Create **Block Diagram/Schematic File** and create **Symbol Files** of all the included Verilog files.

Generate a 256kHz **ALTPLL** for the speed controller.

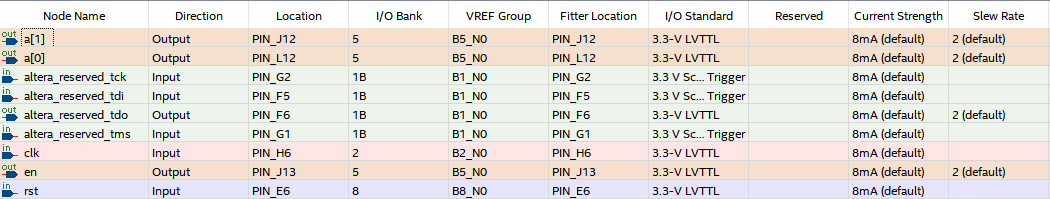
Design a **NIOS System** using the **Platform Design Tool** with **2** **PIOs** to control the **Speed** and **Direction** modules (**Figure 5**).

**Figure 5**

Look at **Figure 6** to see how everything is wired up and then **Start Compilation**.

**Figure 6**

After everything compiles with no errors, open the **Pin Planner** and use **Figure 7** as a guide and set the **Pin Locations**.



**Figure 7**

**NIOS II Setup:**

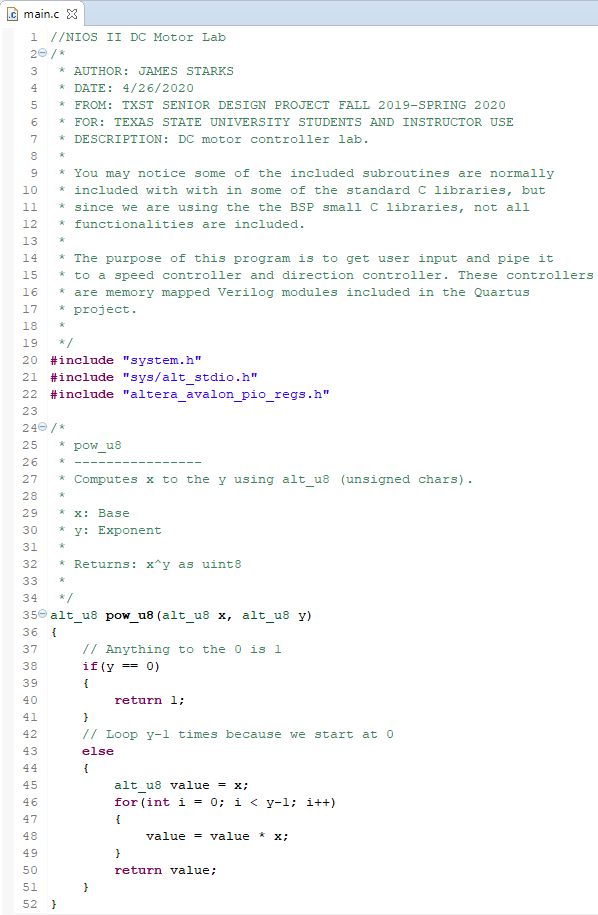
Like in the previous lab guides, start to create a **New NIOS II Application and BSP from template** and select the **Small Hello World** template. Next generate the **BSP** project and you can start editing the application project.

Look at **figures 8 through** **11** to get an idea on how to create this application.

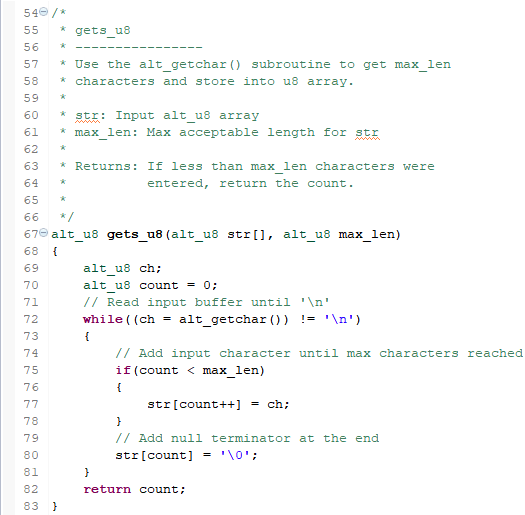
You may think the functions in this application are redundant because they are typically included with the standard C I/O and Math libs, but since this project has been created using the Altera small C library those libraries are not available.

A brief overview of the code below…

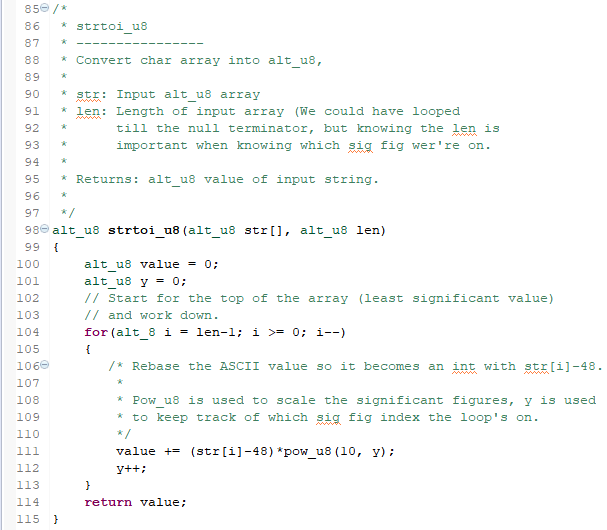
* **Figure 8** shows the included libraries and the custom **pow** function from the math lib used for calculating and returns an unsigned 8-bit value.
* **Figure 9** is a custom **gets** used to read in ***N*** ASCII encoded values, characters, or until ‘***\n***’ value is entered and returns the count of characters retrieved.
* **Figure 10** is a crude custom **strtol** function that assumes the ASCII value array passed is decimal formatted and converts and rebases the values to be read as standard integer values of 8-bits in length.
* **Figure 11** is the main function of the program that asks the user to input the a value from 0-255, duty cycle of the PWM enable will be speed/256\*100. User is then Prompted what direction they want the motor to spin. The desired settings are then written to the hardware modules and the console prints the applied settings.

****

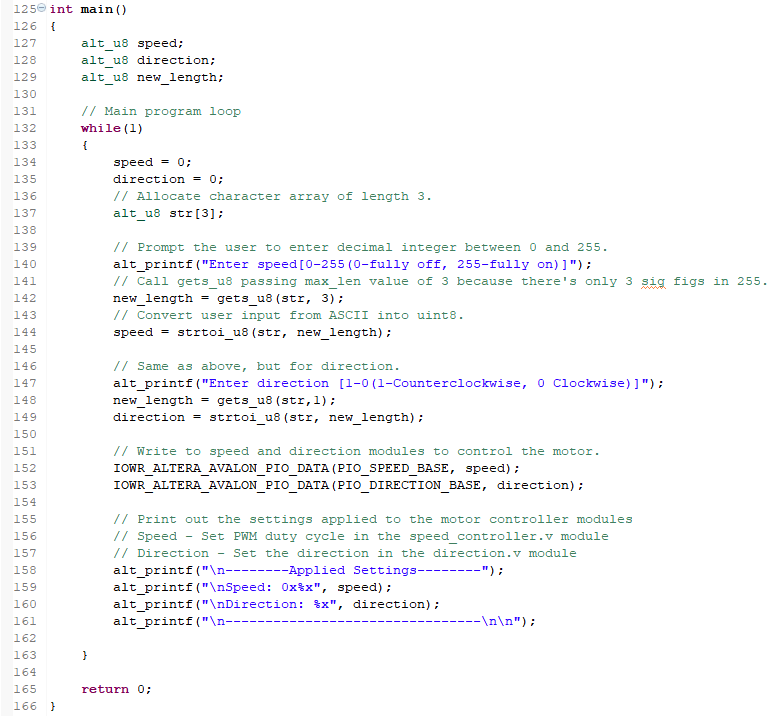
**Figure 8**

****

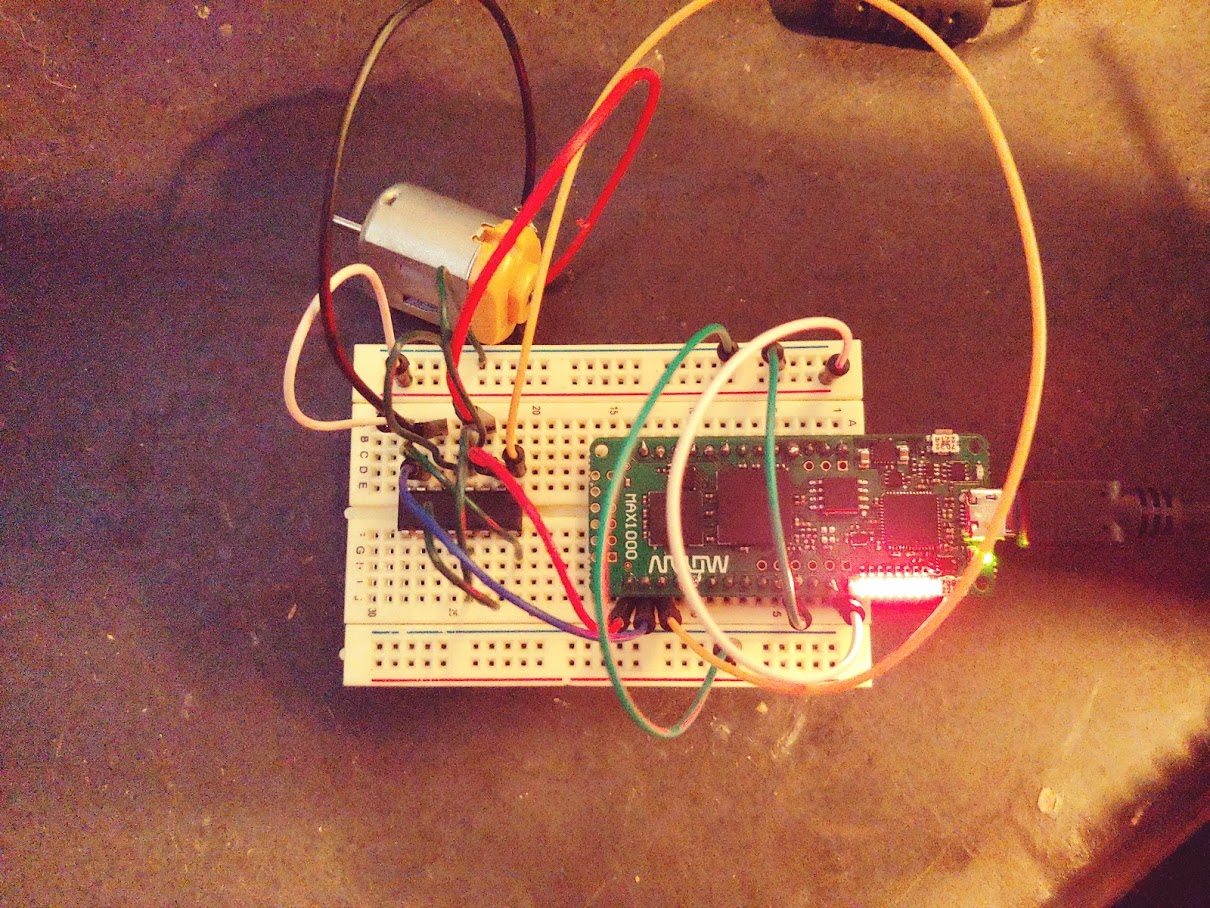
**Figure 9**

****

**Figure 10**

**Figure 11**

**Figure 12** is what the final project should look like on a breadboard.

****

**Figure 12**

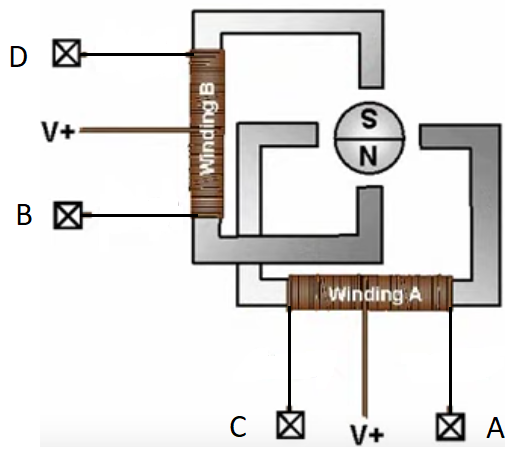
**Part 2: Stepper Motor**

**Example Overview:**

Implement and design a system for the Arrow Max1000 board controlling the SN754410 Half-H to drive a stepper motor.

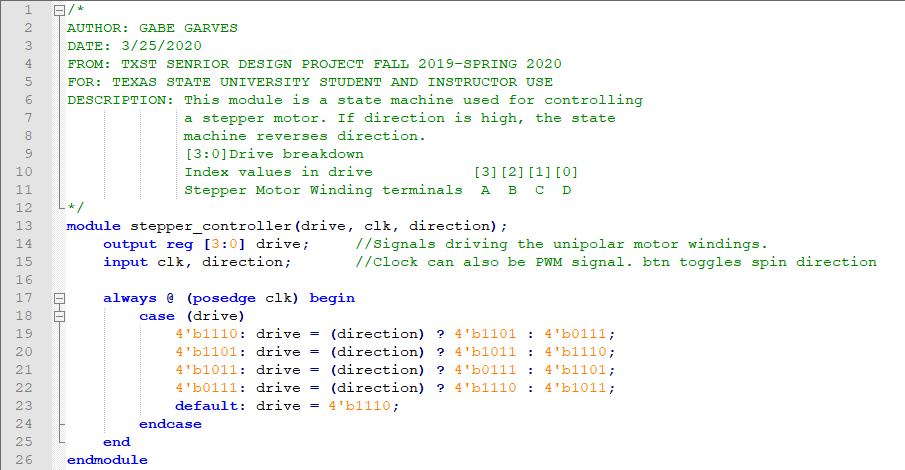
**Verilog Breakdown:**

Looking at **Figure 13,** we can see four pins and a 5 V source center tapped on both windings. When one of the pins is set low (while others remain high) you get current flowing through half of the winding that creates a magnetic field. The order at which you set the pins low can change the magnetic field to spin the motor clockwise or counter clockwise. The order for clockwise would go setting A to low, then B to low, then C, and finally D. For counter clockwise just reverse the order. Remember, only set one pin low at a time.



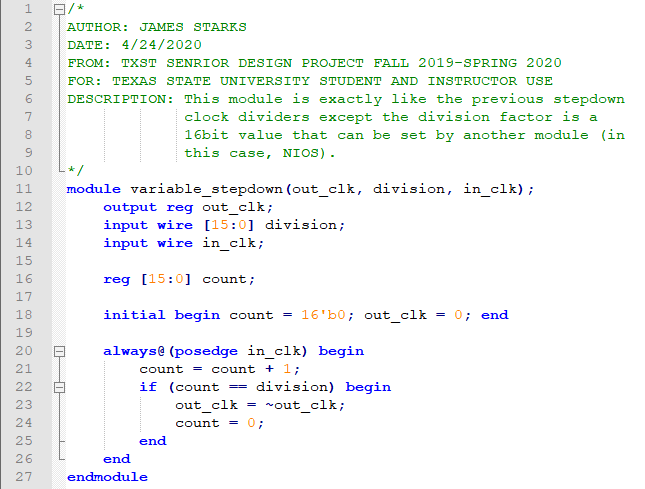
**Figure 13** [[2]](https://www.zotero.org/google-docs/?AwyMk0)

**Figure 14** is a simple bidirectional state machine used to control which coils in the stepper motor are energized. When a 1 is seen on the direction the motor spins counter-clockwise, and clockwise when 0 is on direction. The speed at which the motor spins is dependent on the clock frequency into this module.

**Figure 14**

*Lines 19-22 in* **Figure 14** *are tertiary case statements. They are basically a one-line if statement. In the first case statement, direction is checked and if it is high the next state will go to 4’b1101. If direction is low the next state will be 4’b0111. These case statements are more efficient at the hardware level than if/else-if/else statements.*

**Figure 15** is the module that controls the clock frequency into the stepper controller. The module essentially acts like a variable clock divider that uses a 1-6bit input to divide the clock down. The output frequency can be calculated with in\_clk/division/2 = out\_clk.



**Figure 15**

*You may be wondering why this code isn’t just dividing the clock input directly. Well division in gate logic is very costly to space and inefficient. It is better to have the NIOS C software layer to the division and just send the quotient to this hardware module.*

**FPGA Implementation:**

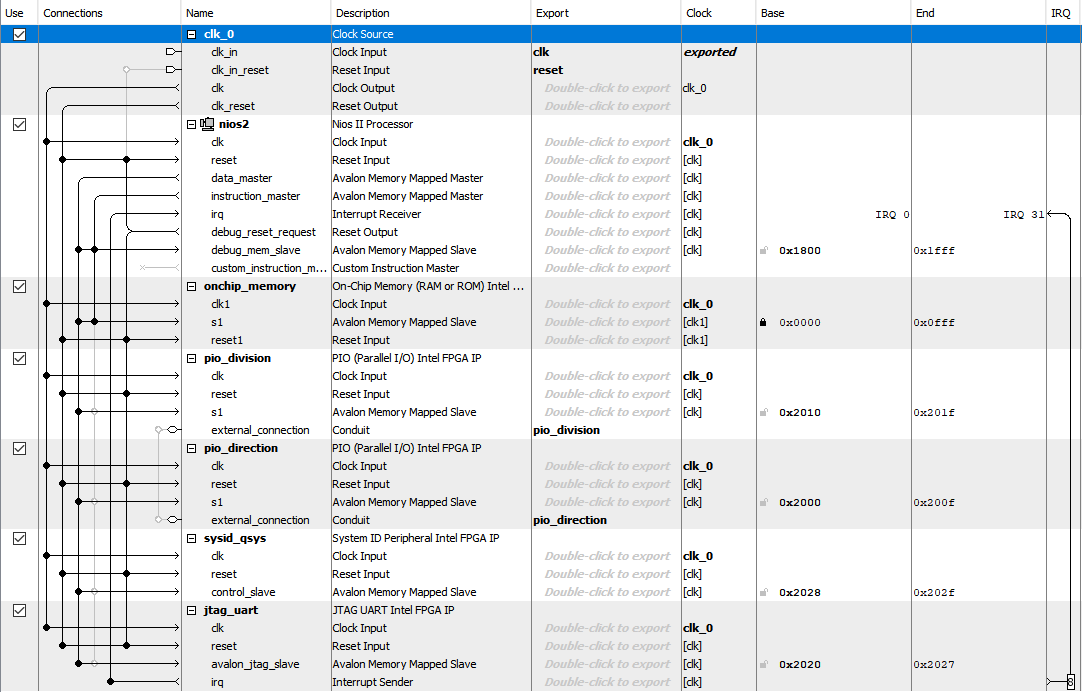
First you will need to create a new Quartus project and include these files…

* **variable\_stepdown.v**
* **unipolar\_stepper.v**

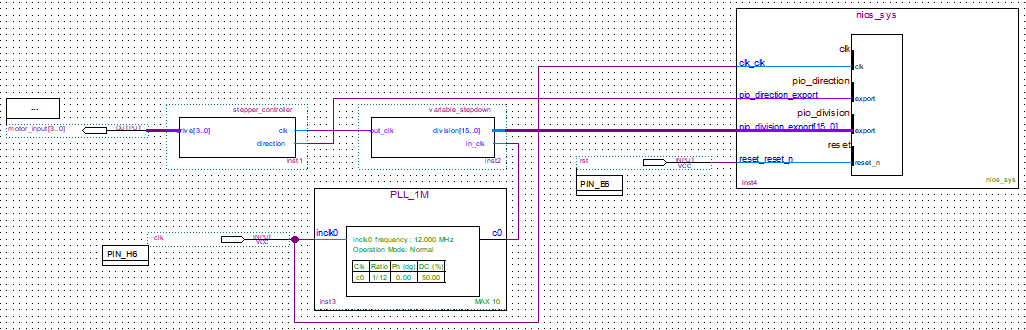
Create **Block Diagram/Schematic File** and create **Symbol Files** of all the included Verilog files.

Generate a 1MHz **ALTPLL** for the speed controller.

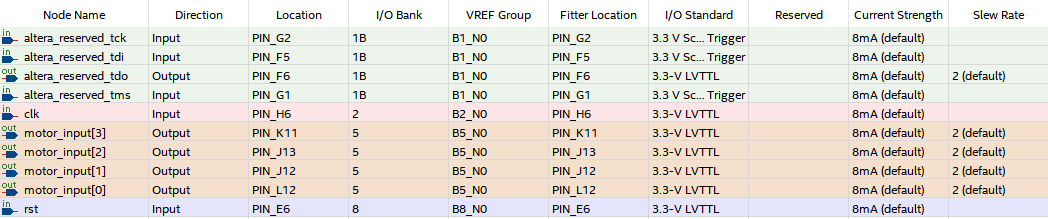
Design a **NIOS System** using the **Platform Design Tool** with **2** **PIOs** to control the **Speed** and **Direction** modules (**Figure 16**).

**Figure 16**

Look at **Figure 17** to see how everything is wired up and then **Start Compilation**.

**Figure 17**

After everything compiles with no errors, open the **Pin Planner** and use **figure 18** as a guide and set the **Pin Locations**.

**Figure 18**

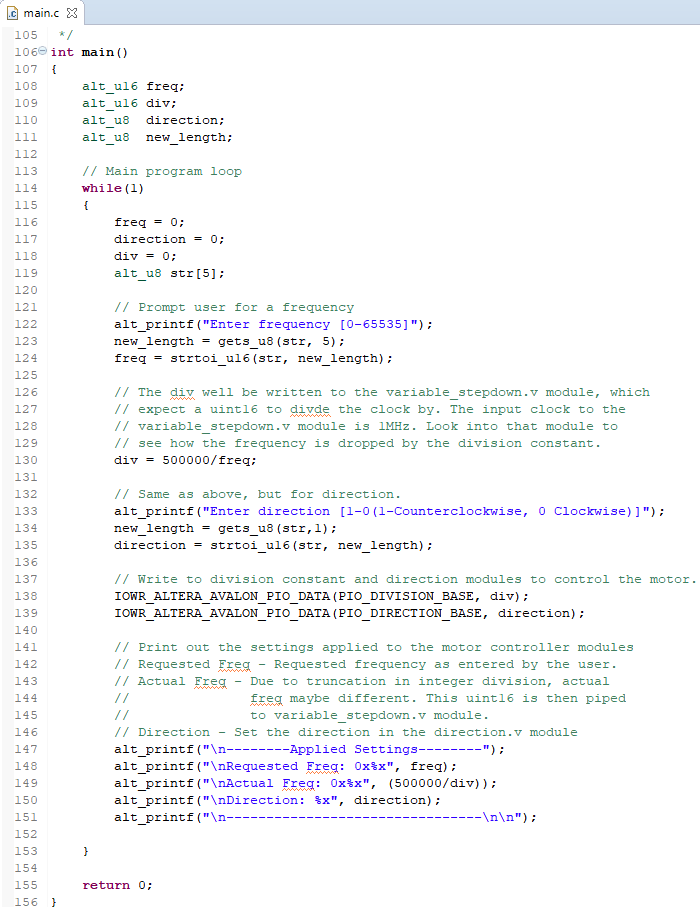
**NIOS II Setup:**

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.

Looking at **Figure 19** only the main function was included, this is because the functions above from the DC motor lab were pretty much lifted to this project, with one exception. Since 16-bits are used in the variable clock, **strtoi** must be adapted (**See line 135)**. Even though you will find the stepper stops functioning reliably around 700Hz, or a division factor of ~2857, having these extra bits means the module can more accurately divide down to the desired frequency.

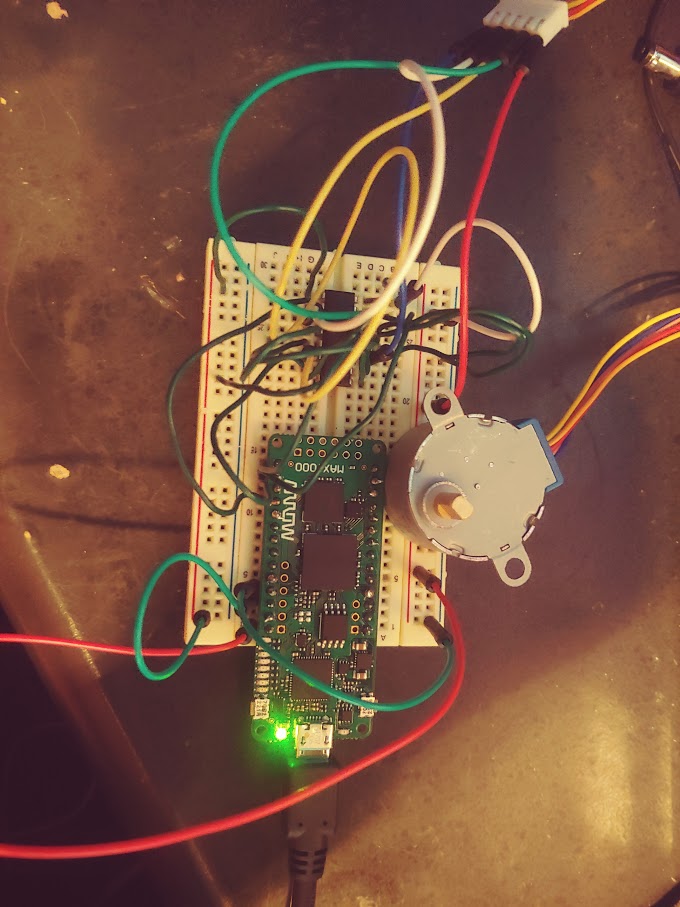
**Note**: On line 130 there is a bit of math, this converts the requested frequency into a division factor that the variable clock uses. Since the variable step down takes a 1MHz input clock, the formula for calculating the division factor from frequency is:

Lastly, like before the options that are applied to the variable step down and stepper controller are displayed to the user, then the program loops. You may notice on the options display, it also outputs the “actual frequency,” this is because most of the time user input frequency doesn’t divide nicely, and integer division truncates any decimal places. So, in the off chance the user enters a frequency that doesn’t divide nicely, the user can see the actual value.



**Figure 19**

**Figure 20** is what the final project should look like on a breadboard.

****

**Figure 20**

**References**

[[1] “sn754410.pdf.” Accessed: Apr. 29, 2020. [Online]. Available: http://www.ti.com/lit/ds/symlink/sn754410.pdf.](https://www.zotero.org/google-docs/?RYYXqL)

[[2] *Unipolar and Bipolar Stepper Motors*. .](https://www.zotero.org/google-docs/?RYYXqL)<https://www.youtube.com/watch?v=vxxnPJBxG3M>