**EE 2420 Lab Guide 5: 3-Bit Counter State Machine**

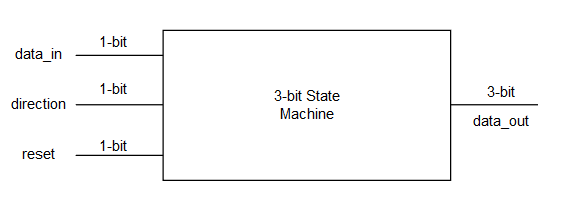
*Written By: Grant Seligman, Gabe Garves, James Starks*

**Example Overview:**

Design a state machine that follows this sequence:

000 → 100 → 110 → 111 → 101 → 001 → 011 → 010 → 000 ... {Gray code}

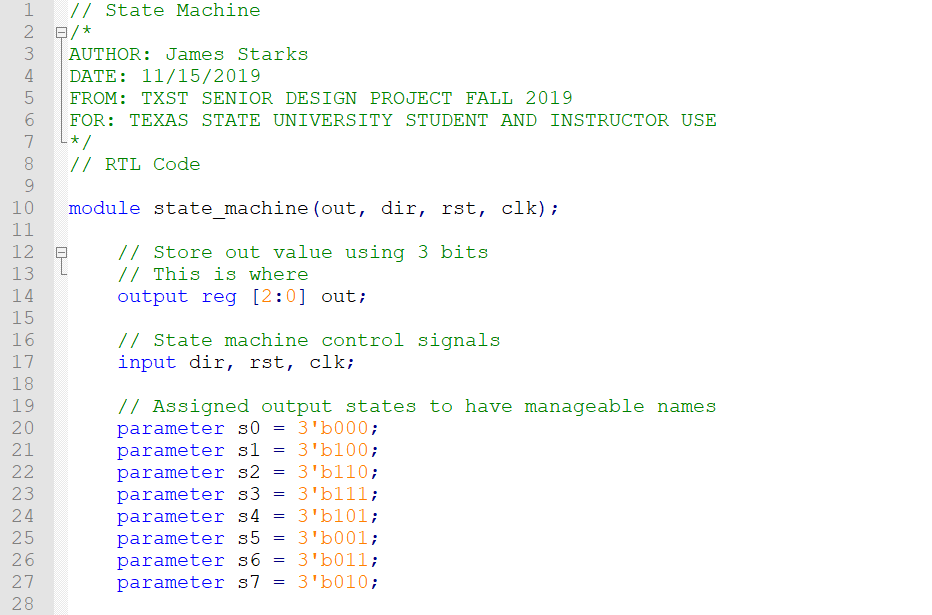
The data bit can be interpreted as a clock signal. If the data bit is high, we proceed to the next or previous step depending on the direction bit. If the data bit is low, output doesn’t change. When the direction bit is set to 0, the sequence should follow the order above. When the direction bit is set to 1, the sequence should go in the reverse order. There must also be a reset bit that resets the state machine. When reset goes low the state machine should be forced into the 000 state, otherwise normal operation will continue. Figure 1 is the example 3-bit state machine we will be working with.

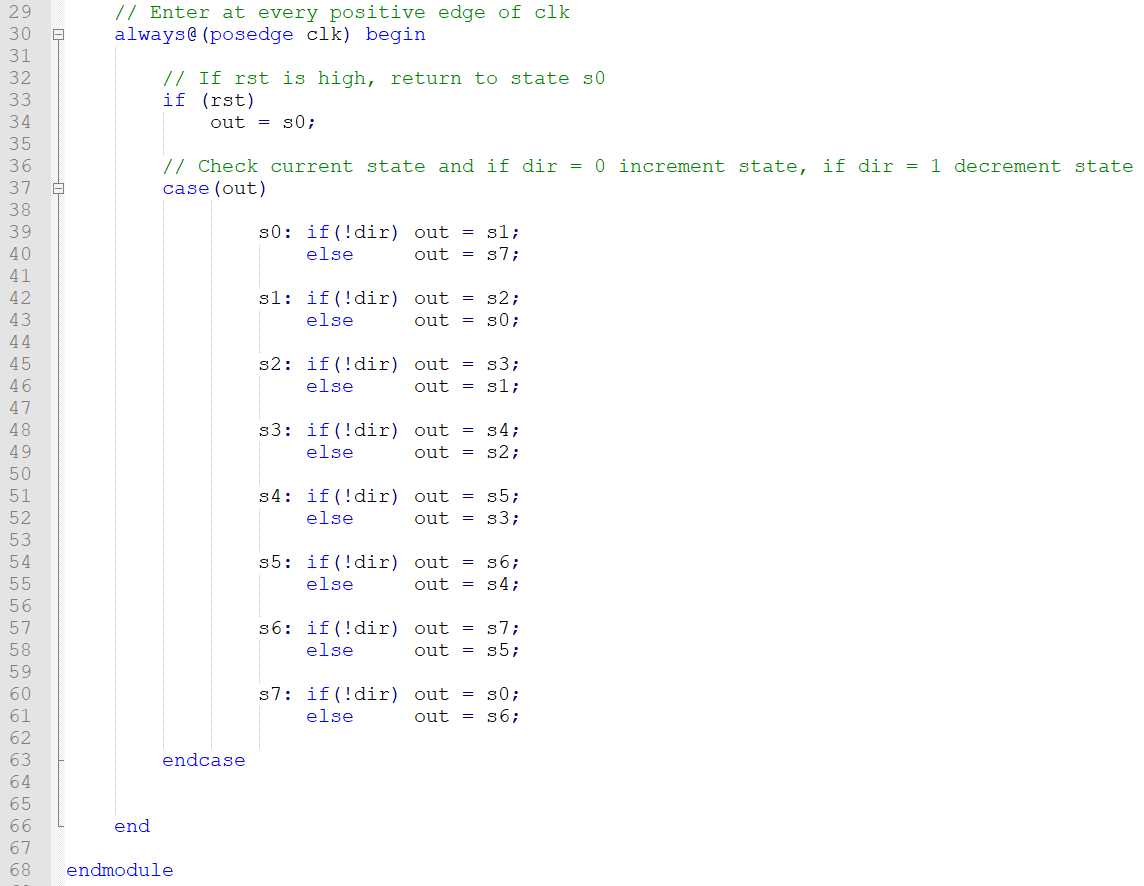


**Figure 1**

**Verilog Code Breakdown:**

For each state, a value is being assigned to the output. Since we are assigning values to the output, we have to define the output as a register to hold these values. We must also create/declare a variable/register named state to keep track of the current state. Whenever the direction or data bit goes high, (posedge) or the reset goes low, (negedge) the always block executes. If reset is ever low, the state is reset and the rest of the always block is ignored. The direction bit determines which section of the always block executes and will increment the state value accordingly. The current state and data bit will determine the next state. The case statement at the end executes every time the always block does, and sets the output to the next state. Not shown in **Figure 1,** is the clock input signal needed to drive the states. One can use the Clock Converter module from the counter lab to create a slow enough clock to see each state.



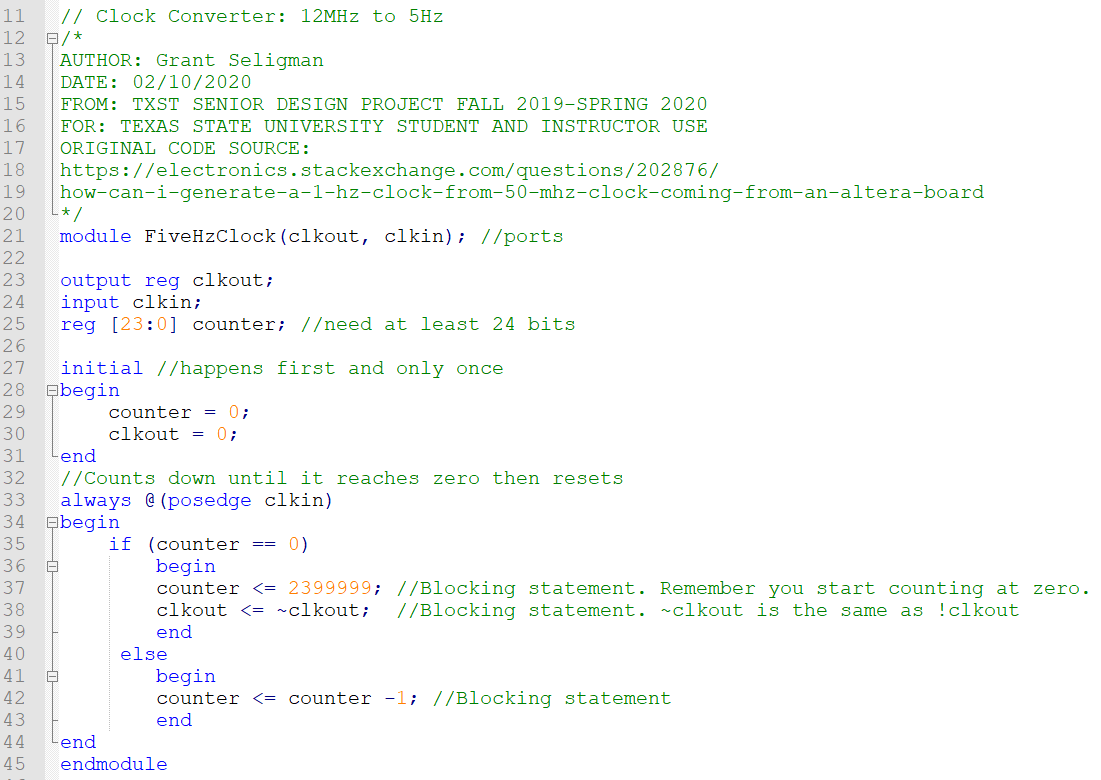


**Figure 2**

The direction bit is set to active low and will reverse the states when switched high. What is nice about this state machine is that it is almost identical to the counter except now one can control the direction of states.

**FPGA Implementation:**

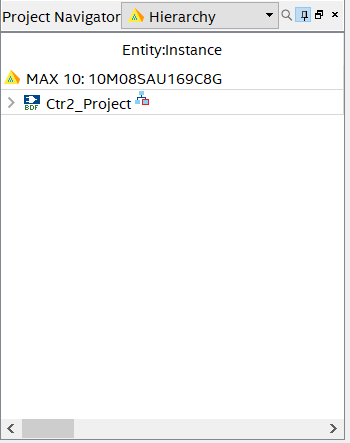
Using your knowledge from the previous labs, implement your verilog code into a **symbol** for each module. For this project, along with the module in **Figure 2**, we will just use the 12Mhz clock and drop it down to a 5Hz clock to see the changes in state easier. All that changes is the number of counts per cycle:



**Figure 3**

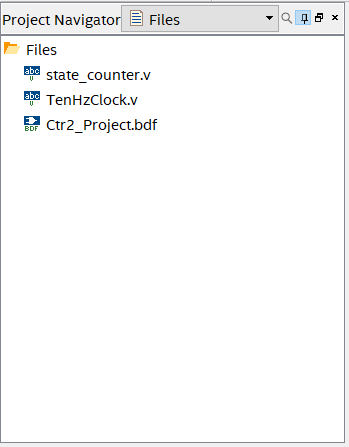
*The original code for the counter was sourced from a stackexchange webpage.* [1] *The module was modified by Grant Seligman.*

**Make sure to set the finished .bdf file as your Top Level Entity before compiling and then setting the pins. This sets the entire block chain as one entity and allows Quartus to set up the pin planner correctly. Follow Figures 5 - 7 to set the Top-Level Entity.**

****

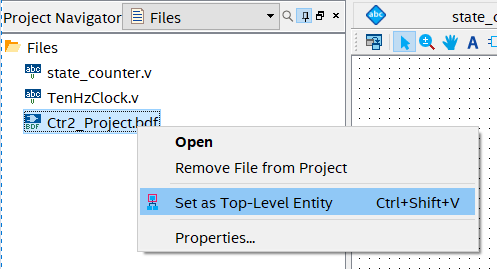
**Figure 5**

Go to the **Project Navigator** and use the drop down menu and select **Files.** This will show you all the current files within your Quartus Project.

****

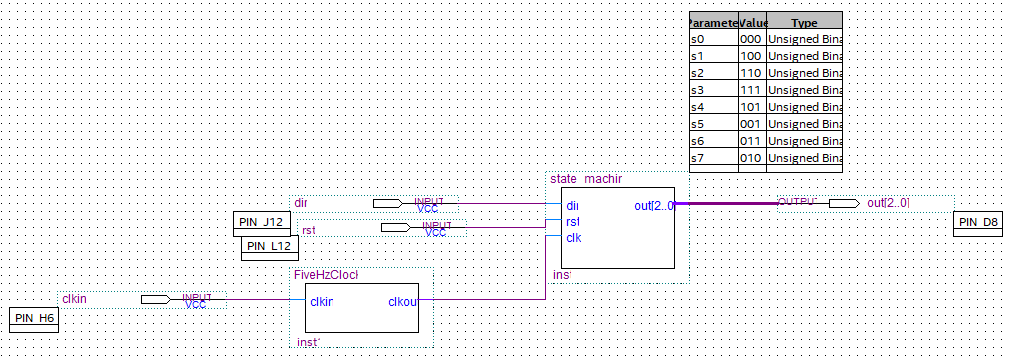
**Figure 6**

Right-click on the **.bdf** file of your project and select **Set as Top-Level Entity.**

****

**Figure 7**

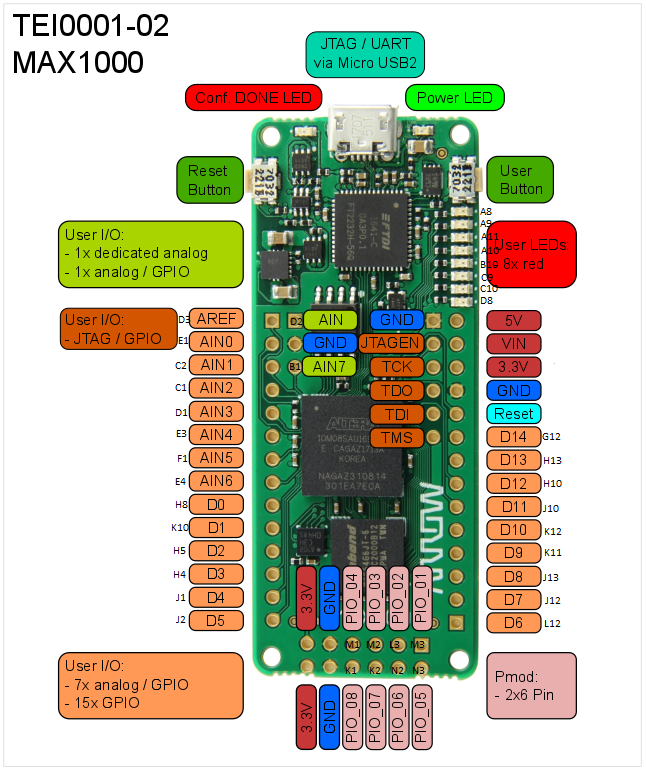
Your **.bdf** file should look similar to **Figure 8.** Then follow **Table 1** and **Figure 9** to set your pins accordingly. You can simplify the process if you use a function generator as a clock input, but you will have to set the **clk** input to an orange pin.



**Figure 8**

|  |  |  |  |
| --- | --- | --- | --- |
| Node Name | Direction | Location | I/O Standard |
| rst | Input | PIN\_L12 | 3.3-V LVTTL |
| clkin | Input | PIN\_H6 | 3.3-V LVTTL |
| dir | Input | PIN\_J12 | 3.3-V LVTTL |
| out[2] | Output | PIN\_C9 | 3.3-V LVTTL |
| out[1] | Output | PIN\_C10 | 3.3-V LVTTL |
| out[0] | Output | PIN\_D8 | 3.3-V LVTTL |

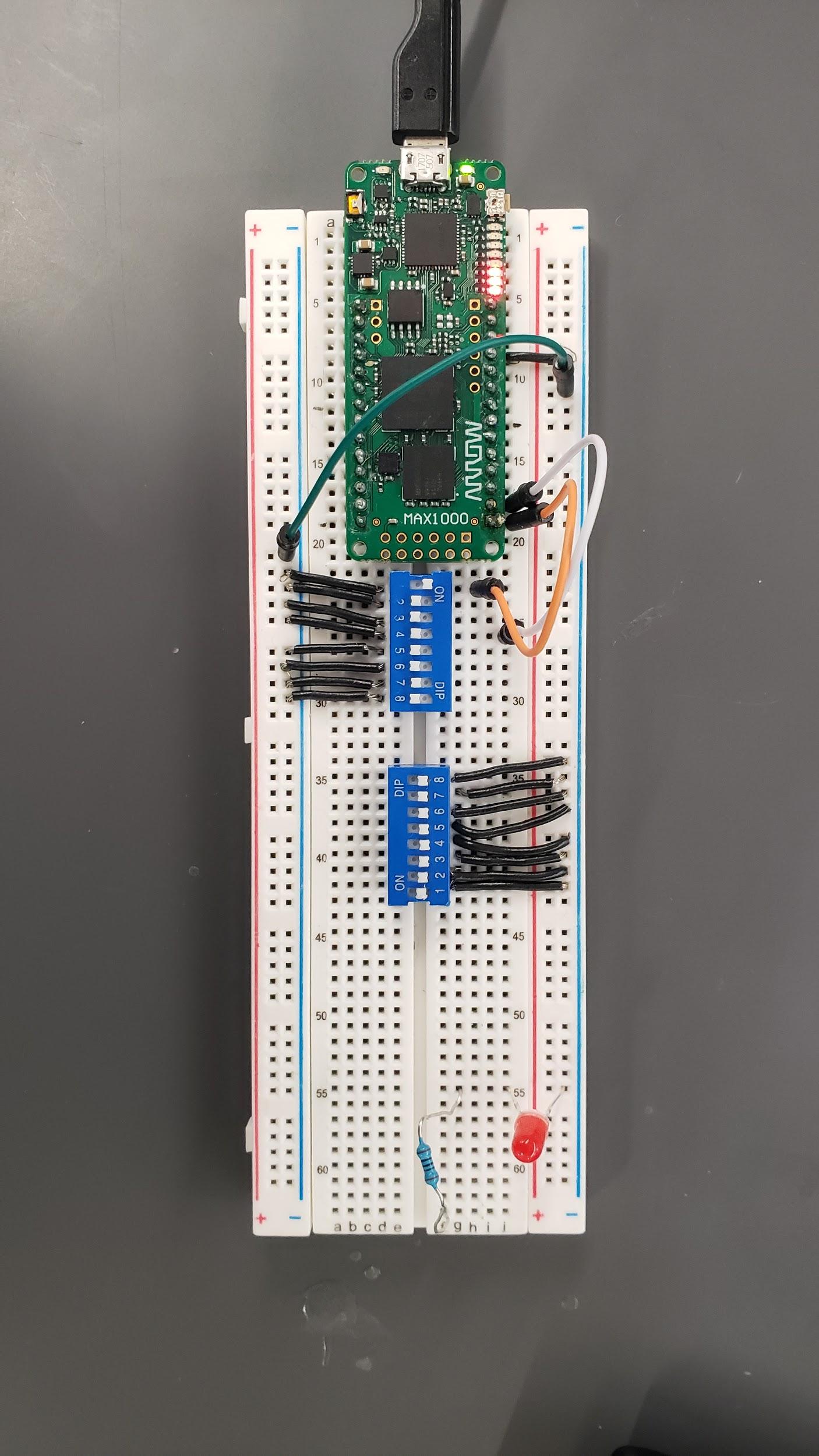
**Table 1**



**Figure 9**

**Working with the Peripheral Board**

For the board, all one needs is two dip switches to control reset and the direction. Depending on how you set your pins your board should look similar to **Figure 10.**

****

**Figure 10**

**REFERENCES SECTION**

[1] “verilog - How can I generate a 1 Hz clock from 50 MHz clock coming from an Altera board?,” *Electrical Engineering Stack Exchange*. [Online]. Available: https://electronics.stackexchange.com/questions/202876/how-can-i-generate-a-1-hz-clock-from-50-mhz-clock-coming-from-an-altera-board. [Accessed: 10-Feb-2020].