

## EE 2420 Lab Guide 2: 8-Bit Ripple Carry Adder

### Example Overview:

Design a circuit that takes two numbers, 8 bits long, and adds them. Now that you are familiar with creating an adder, it's important to know how to create one for any number of bits. Hardware changes fast in industry and thus hardware designs need to as well.

### Verilog Code Breakdown:

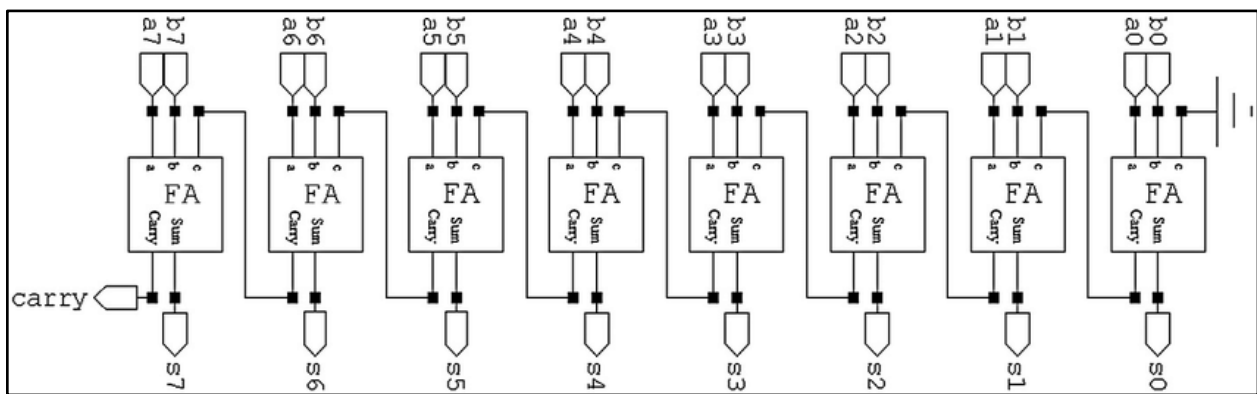


Figure 1

As you can see in Figure 1, you can create any size adder by cascading full adders together to  $n$ -bit lengths.

Start by creating a half adder module. This will be called on by other modules.

```
1 //Half Adder in Verilog
2 /*
3  AUTHOR: Gabe Garves
4  DATE: 11/8/2019
5  FROM: TXST SENIOR DESIGN PROJECT FALL 2019
6  FOR: TEXAS STATE UNIVERSITY STUDENT AND INSTRUCTOR USE
7  */
8  //Structural Code
9
10 module HA(
11     input wire a, b,          //2 inputs
12     output wire sum, cout     //2 outputs
13 );
14
15     //gate var_name(output, input1, input2);
16
17     xor xor1(sum, a, b);      //Logic for half adder
18     and and1(cout, a, b);
19
20 endmodule
```

Figure 2

Next a full adder module needs to be designed. It will use the half adder module from above.

```
1 //Full Adder in Verilog
2 /*
3  AUTHOR: Gabe Garves
4  DATE: 11/8/2019
5  FROM: TXST SENIOR DESIGN PROJECT FALL 2019
6  FOR: TEXAS STATE UNIVERSITY STUDENT AND INSTRUCTOR USE
7  */
8  //Structural Code
9
10 module FA(
11     input wire x, y, cin,     //3 inputs
12     output wire Sum, cout     //2 outputs
13 );
14
15     wire w1, w2, w3;         //connecting wires
16
17     //module var_name(output1, output2, input1, input2);
18
19     HA ha1(w1, w2, x, y);      //uses the HA module to add x and y
20     HA ha2(Sum, w3, cin, w1);  //add cin and carry from x and y
21     or o1(cout, w3, w2);       //calculating carry
22
23 endmodule
```

Figure 3

This is what the 8-bit ripple carry adder looks like. Just like before we will use the full adder module to add each of the bits.

```
1 //8-Bit Ripple Carry in Verilog
2 /*
3  AUTHOR: Gabe Garves
4  DATE: 11/8/2019
5  FROM: TXST SENIOR DESIGN PROJECT FALL 2019
6  FOR: TEXAS STATE UNIVERSITY STUDENT AND INSTRUCTOR USE
7  */
8  //Structural Code
9
10 module RCA8(
11     input wire [7:0] a, b, //2 8-Bit inputs
12     input wire cin, //carry in
13     output wire [7:0] sum, //8-Bit output
14     output wire cout //carry out
15 );
16     wire w[6:0]; //connecting wires
17
18
19     //module var_name(output1, output2, input1, input2, input3);
20
21     FA fa0(sum[0], w[0], a[0], b[0], cin); //Calling on the FA
22     FA fa1(sum[1], w[1], a[1], b[1], w[0]); //module 8 times for
23     FA fa2(sum[2], w[2], a[2], b[2], w[1]); //each of the inputs
24     FA fa3(sum[3], w[3], a[3], b[3], w[2]);
25     FA fa4(sum[4], w[4], a[4], b[4], w[3]);
26     FA fa5(sum[5], w[5], a[5], b[5], w[4]);
27     FA fa6(sum[6], w[6], a[6], b[6], w[5]);
28     FA fa7(sum[7], cout, a[7], b[7], w[6]);
29
30 endmodule
```

Figure 4

## FPGA Implementation:

The process for programming the FPGA with this ripple carry adder is the same as in Lab Guide 1. But all instructions are provided again for ease of use.

### Creating a Quartus Project:

Before opening a Quartus make sure to **create a new folder for each project**. This will keep the files Quartus creates organized and is recommended for ease of use. Then open up Quartus and you should see a window similar to **Figure 5**.

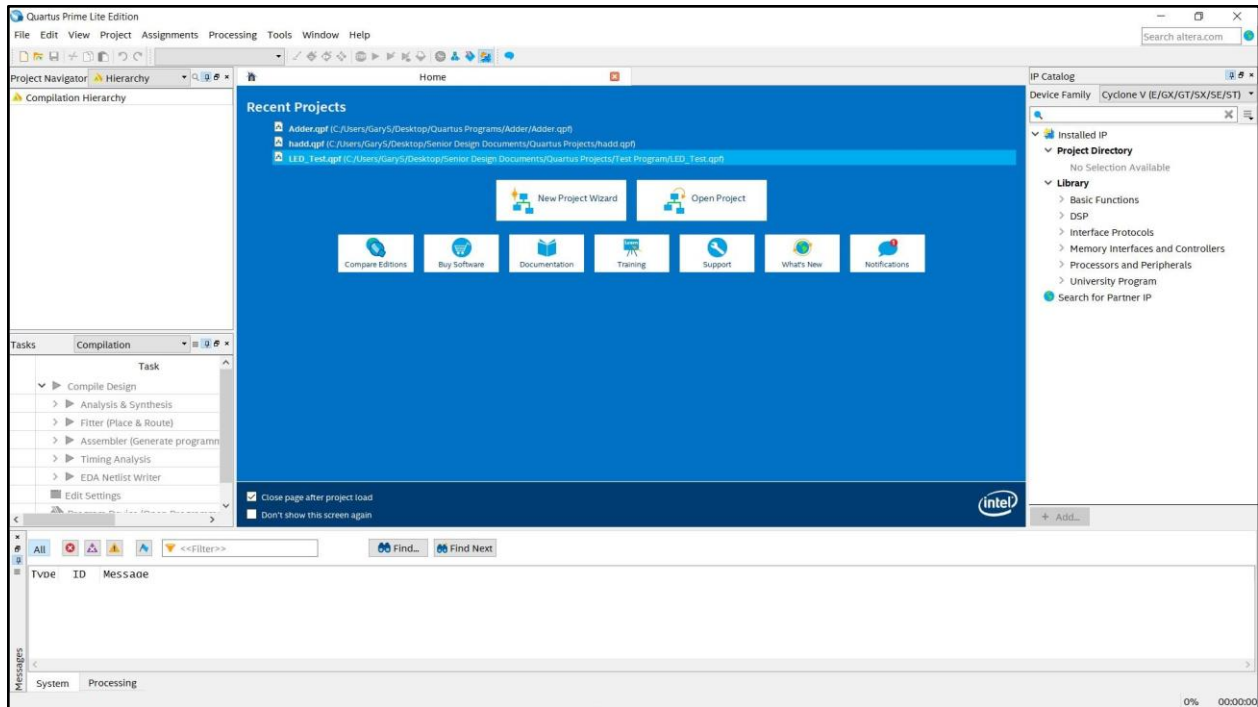
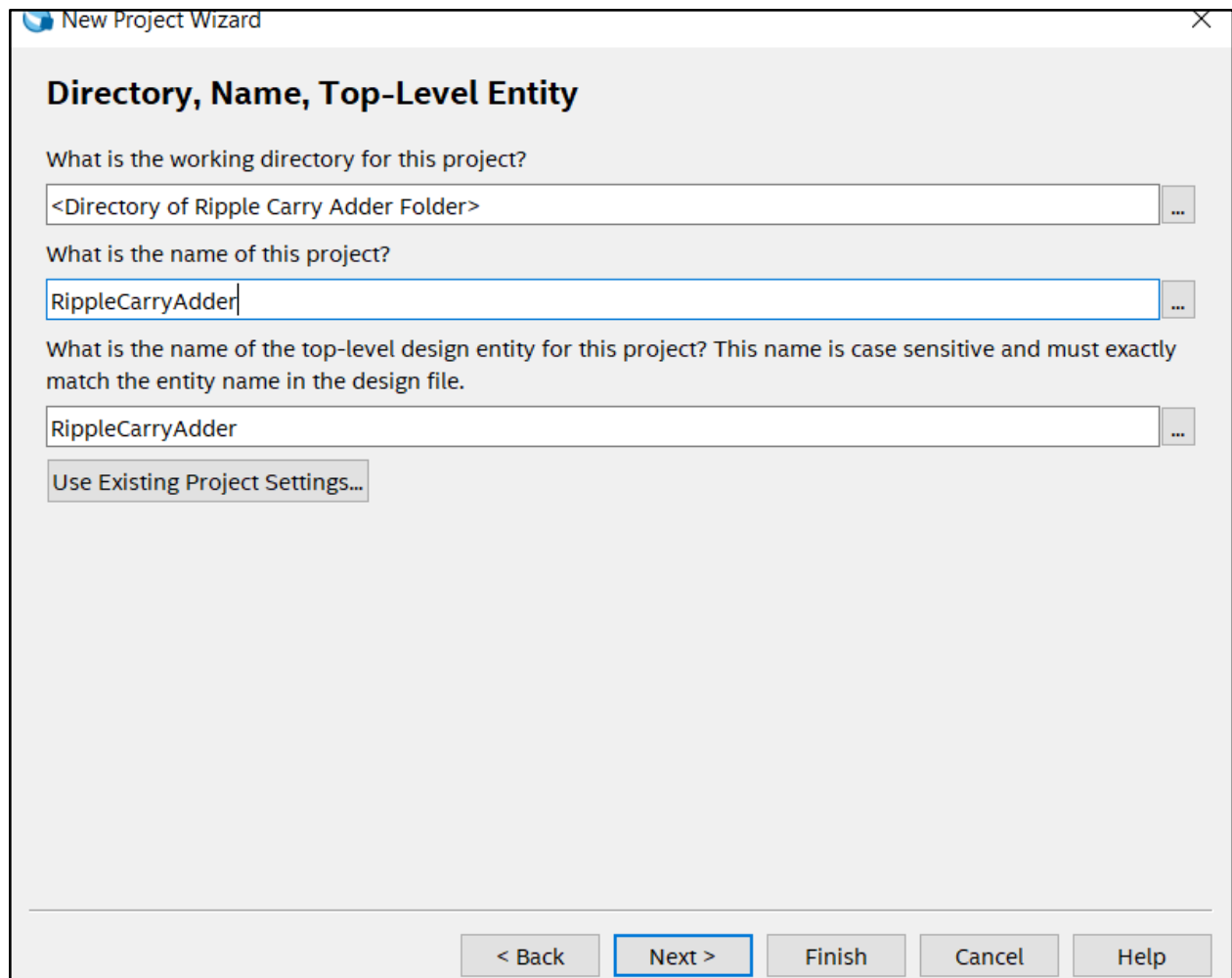


Figure 5

Next, click **New Project Wizard**, click next until you get to **Directory, Name, Top-Level Entity** window (Figure 5). Name the project **RippleCarryAdder** and then select the directory folder you created earlier. Click **Next** until you reach **Family, Device & Board Settings** window. Look at Figure 6, first in the **Name filter** box type in “10M08SAU169C8G.” Second click on the first entry in the table below, then click next until finish.



The image shows a 'New Project Wizard' dialog box with a title bar containing a blue icon and the text 'New Project Wizard'. The main title is 'Directory, Name, Top-Level Entity'. It contains three text input fields with labels: 'What is the working directory for this project?' (containing '<Directory of Ripple Carry Adder Folder>'), 'What is the name of this project?' (containing 'RippleCarryAdder'), and 'What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.' (containing 'RippleCarryAdder'). Below the third field is a button labeled 'Use Existing Project Settings...'. At the bottom are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'. The 'Next >' button is highlighted with a blue border.

**Directory, Name, Top-Level Entity**

What is the working directory for this project?

<Directory of Ripple Carry Adder Folder> ...

What is the name of this project?

RippleCarryAdder ...

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

RippleCarryAdder ...

Use Existing Project Settings...

< Back   Next >   Finish   Cancel   Help

**Figure 6**

*Here you are setting up the file path for your project. For each lab, make sure to make a new project so you don't risk working in a previous project or having directory path location issues later.*

New Project Wizard

## Family, Device & Board Settings

Device Board

Select the family and device you want to target for compilation.  
You can install additional device support with the Install Devices command on the Tools menu.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the [Device Support List](#) webpage.

Device family

Family: MAX 10 (DA/DF/DC/SA/SC)

Device: All

Target device

☐ Auto device selected by the Fitter

☒ Specific device selected in 'Available devices' list

☐ Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Core speed grade: Any

Name filter: 1. 10M08SAU169C8G

☐ Show advanced devices

Available devices:

Name 2.	Core Voltage	LEs	Total I/Os	GPIOs	Memory Bits	Embedded multiplier 9-b
10M08SAU169C8G	3.3V	8064	130	130	387072	48
10M08SAU169C8GES	3.3V	8064	130	130	387072	48

< >

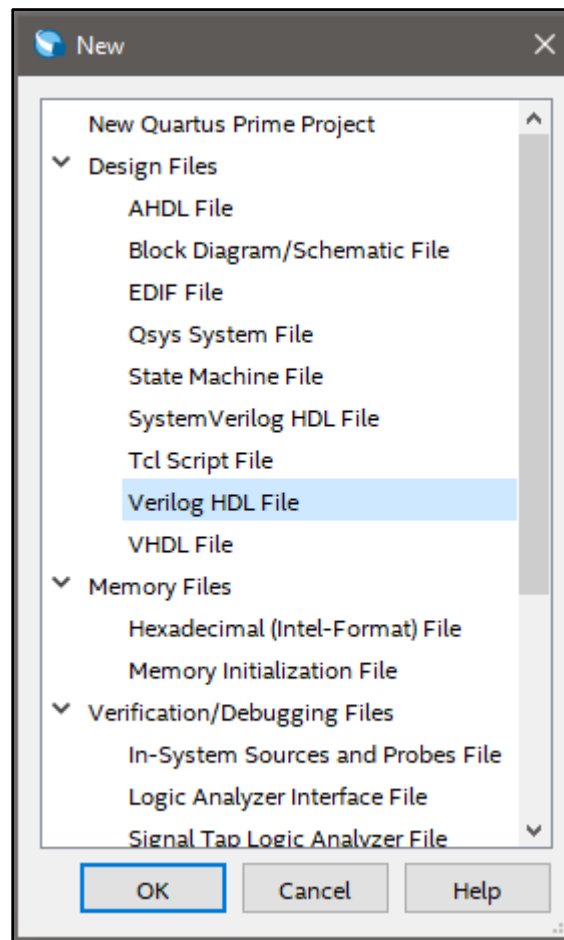
< Back Next > Finish Cancel Help

Figure 7

*This window is where you tell Quartus what hardware you will be working with. If you installed Quartus Lite correctly, The Max 10 device family should be present within Quartus' internal directory files.*

## Importing and Checking the Verilog File

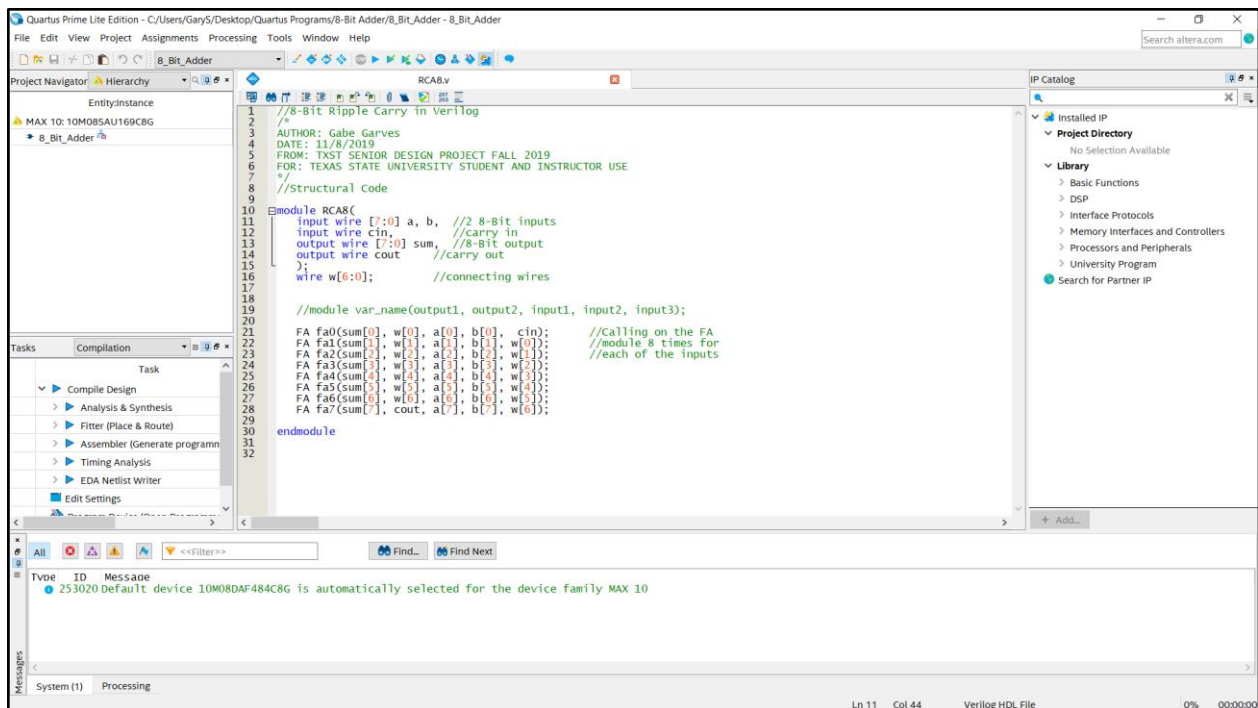
To create a new project: **File>>New** then a window will open (**Figure 8**) which will allow you to create a variety of programs that are linked with Quartus. For now, select the **Verilog HDL File**. This will create a new (.v) file within Quartus or the Ripple Carry Adder code file can be imported by **File>>Open>><Directory of (.v) files>**



**Figure 8**

*Quartus is basically a set of programs that can do a wide range of design analysis. You can potentially create all your programs, compile, simulate, and export hardware programs all within Quartus.*

Once you either have imported or copied over the 3-bit adder file the window should look similar to **Figure 8**.



**Figure 8**

## Creating a Symbol File

A symbol file is the hardware block representation of the verilog code. Similar to if you were to create a new model of a circuit or component in either MultiSim or LTSpice. While Adder.v is open click **File>>Create / Update>>Create Symbol Files from Current File**. Message window will notify you if the process was successful (Alt+3 brings up a message window if not already showing). Now a **.qsf** file should appear in your file directory.

When a **Symbol File** is created, Quartus checks for errors in the verilog file in question. So, if there are any errors that are present they will show up in the **Messages** tab. This is usually located at the bottom of the screen like in **Figure 8**.



## Creating and Wiring the Schematic File

**File>>New>>Block Diagram/Schematic File.** This will generate a **.bdf** file which is where you can drop custom symbol files. **Right click** on the Block Diagram window and click **Insert>>Symbol**, inside the Libraries window click **Project>>RippleCarryAdder**. Press okay and place the Ripple Carry Adder symbol on the schematic. As you can see the Ripple Carry Adder symbol you created from the verilog file has the inputs and outputs defined from the code. Inputs will be on the left and outputs on the right. **Note:** When creating a verilog file, make sure to label your I/O with relevant names. (**Figure 9**) This will make it easier for you to connect all the I/O.

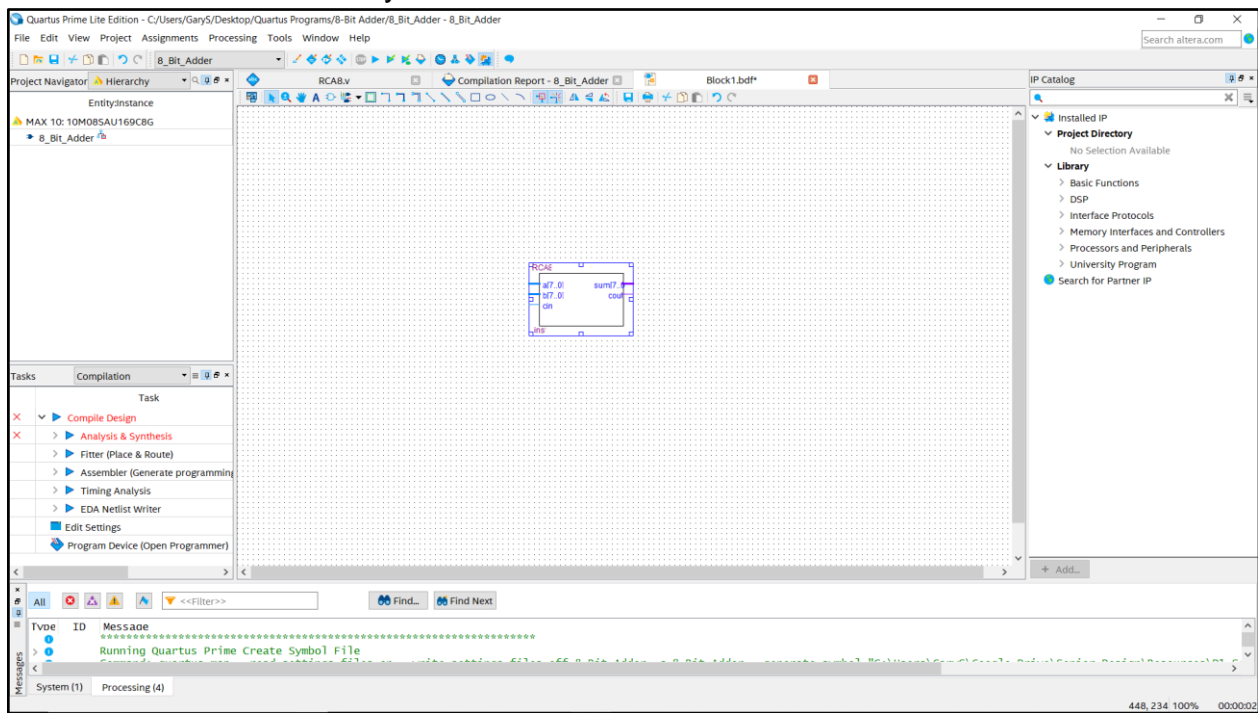


Figure 9

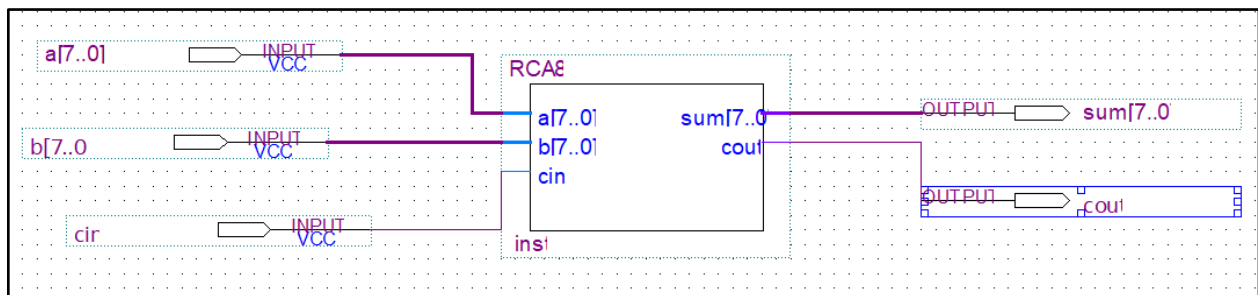
The **.bdf** file will allow you to configure all your I/O for your symbols. Quartus also has built in symbols for basic functions, clocks, etc. To add them into your grid, just view the drop-down menus on the **Library** section to the right in **Figure 9**.

On the toolbar click on the drop-down arrow next to the **Pin Tool (Figure 10, Blue Box)**. Place two inputs and two outputs on their respective sides. You can rename the I/O pins by **right clicking** the **Pin>>Properties** then typing in the desired name into the **Pin name(s)** text box. Press **Ok** to save changes.



Figure 10


Now you need to connect I/O pins to the I/O terminals on the Ripple Carry Adder symbol. Input A, B, and output Sum will all need the **Bus Tool** (Figure 10, Green Box) because they are composed of multiple wires. Since Cout is composed of only one wire, you will use the **Orthogonal Node Tool** (Figure 10, Red Box). You can use the Diagonal version of the Bus and Node tool if you wish. After all has been wired, the schematic file should look similar to the one in **Figure 11**.



**Figure 11**

## Compiling the Project

Once you have created your circuit, you need to compile the project. Compiling will check all the files for final errors and then set the initial pin I/O for the **Pin Planner** which we will get to in the next section. To compile: Go to the top menu bar and click:

**Processing >> Start Compilation.** Or you can press  button. If this initial compilation is successful you will see something similar to **Figure 12**.

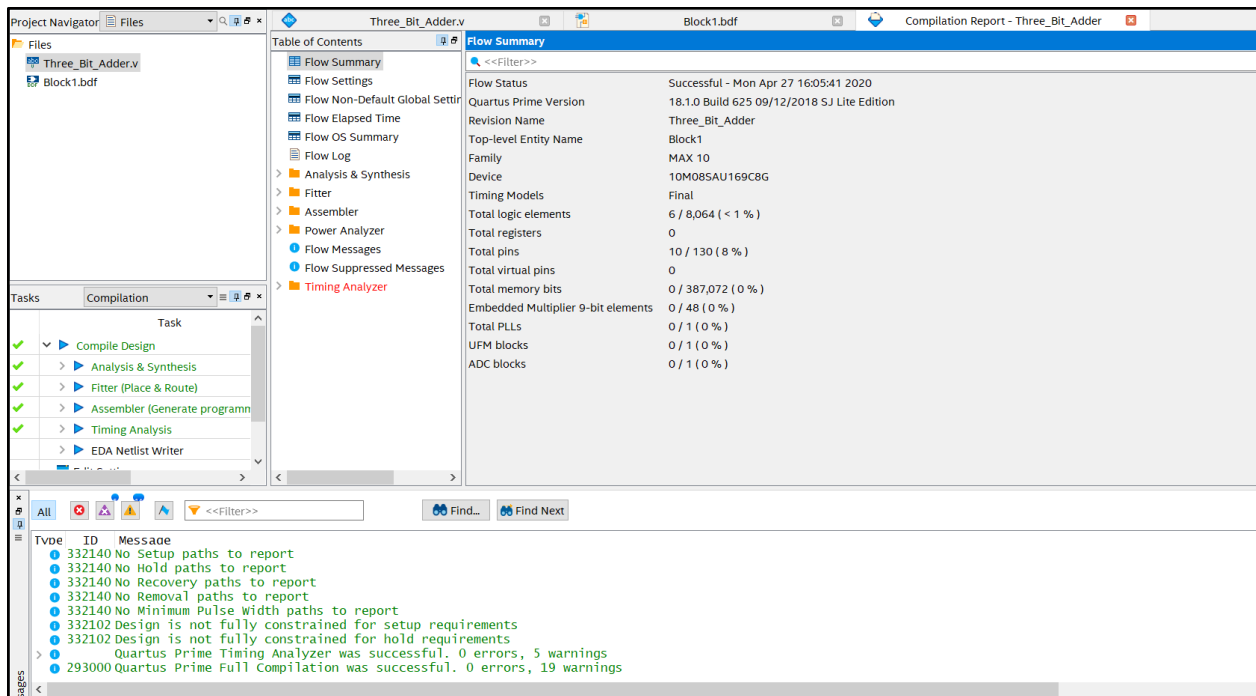


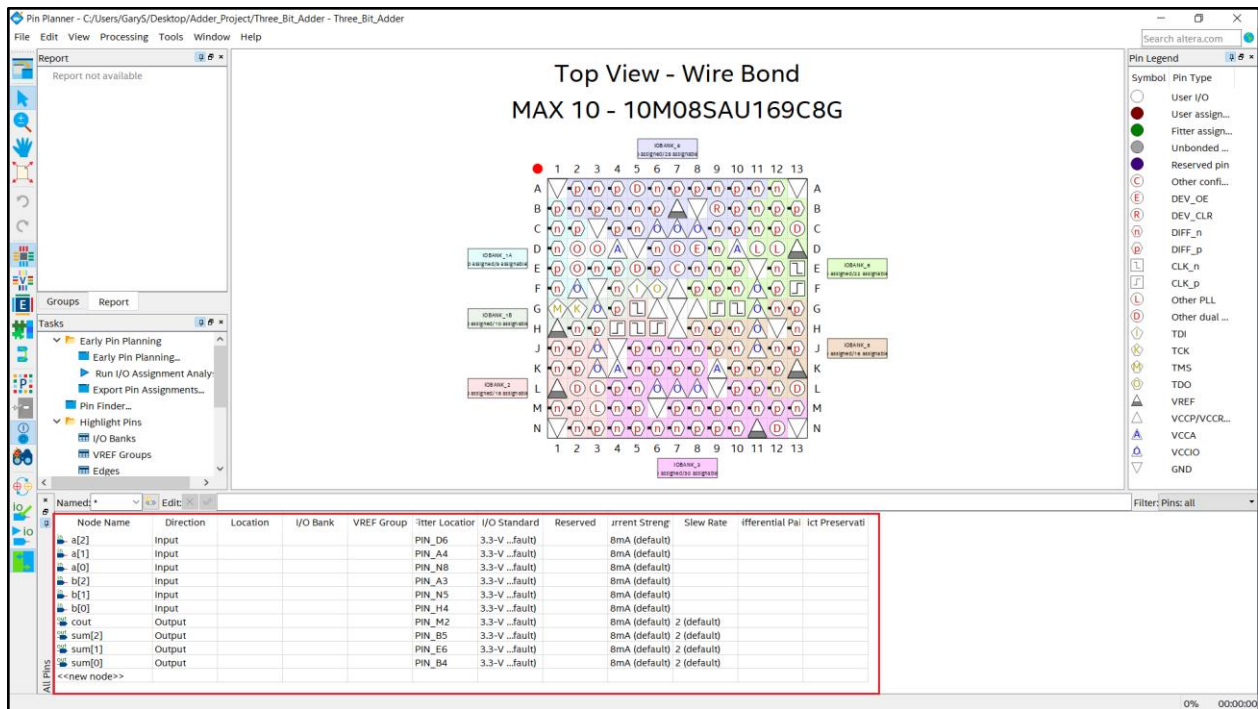
Figure 12

*If you run into the **Top-Level Entity** error again, make sure to set your **.bdf** file as the **Top-Level Entity** and then recompile the project. Refer back to the 3-Bit Adder Guide if you need to refresh yourself on this.*

## Pin Allocation with the Pin Planner

Compile the program again, then launch the Quartus Pin Planner by clicking **Assignments** on the menu bar then **Pin Planner (Figure 13)**. In the **All Pins List** window you can see what Nodes can be assigned a pin on the Max1000.

You can either click and drag the nodes to the desired pin location on the **Top View** or type the pin location in the **Location** field. The **Top View** grid uses letters for rows and numbers for columns. Use **Figure 14** as a guide to set the pins. Any of the orange pins can be used as inputs and the LEDs will be the outputs. **The orange pins names do not associate with the Pin Planner layout. The actual pin names are the smaller letters in Figure 18.** Insert the pin name in the **Locations** column in the **All Pins** window. After all pins have been set, exit out of the pin planner, all data saves automatically.



**Figure 13**

There will be more pins than what is shown in **Figure 13** given you will have 16 inputs and 8 outputs.

# TEI0001-02 MAX1000

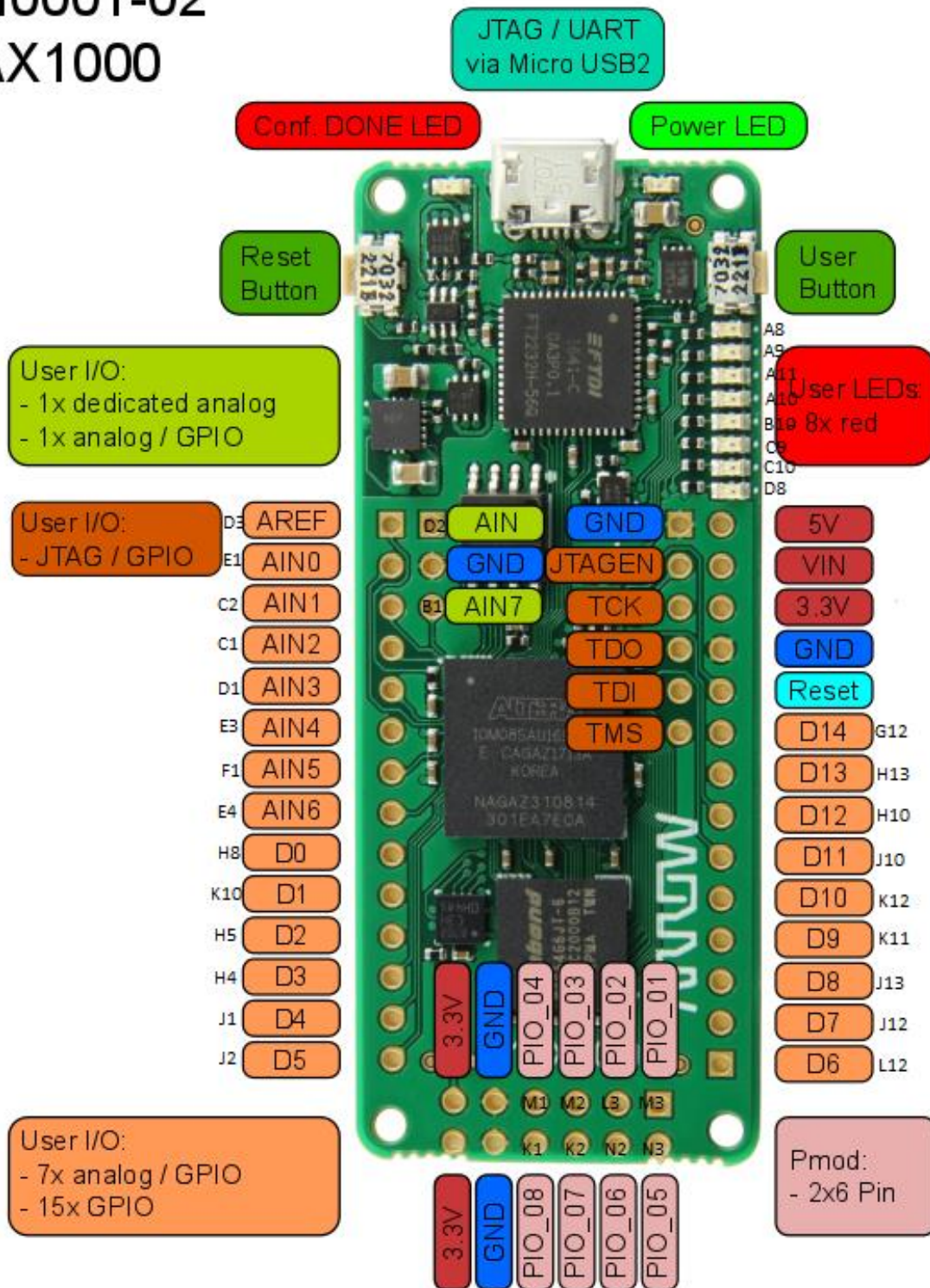


Figure 14



Lastly make sure operating conditions of the device are set correctly. Go to **Assignments>> Settings>>Operating Settings and Conditions>>Voltage** and set **VCCA** to **3.3V** then click Apply and Ok (Figure 15).

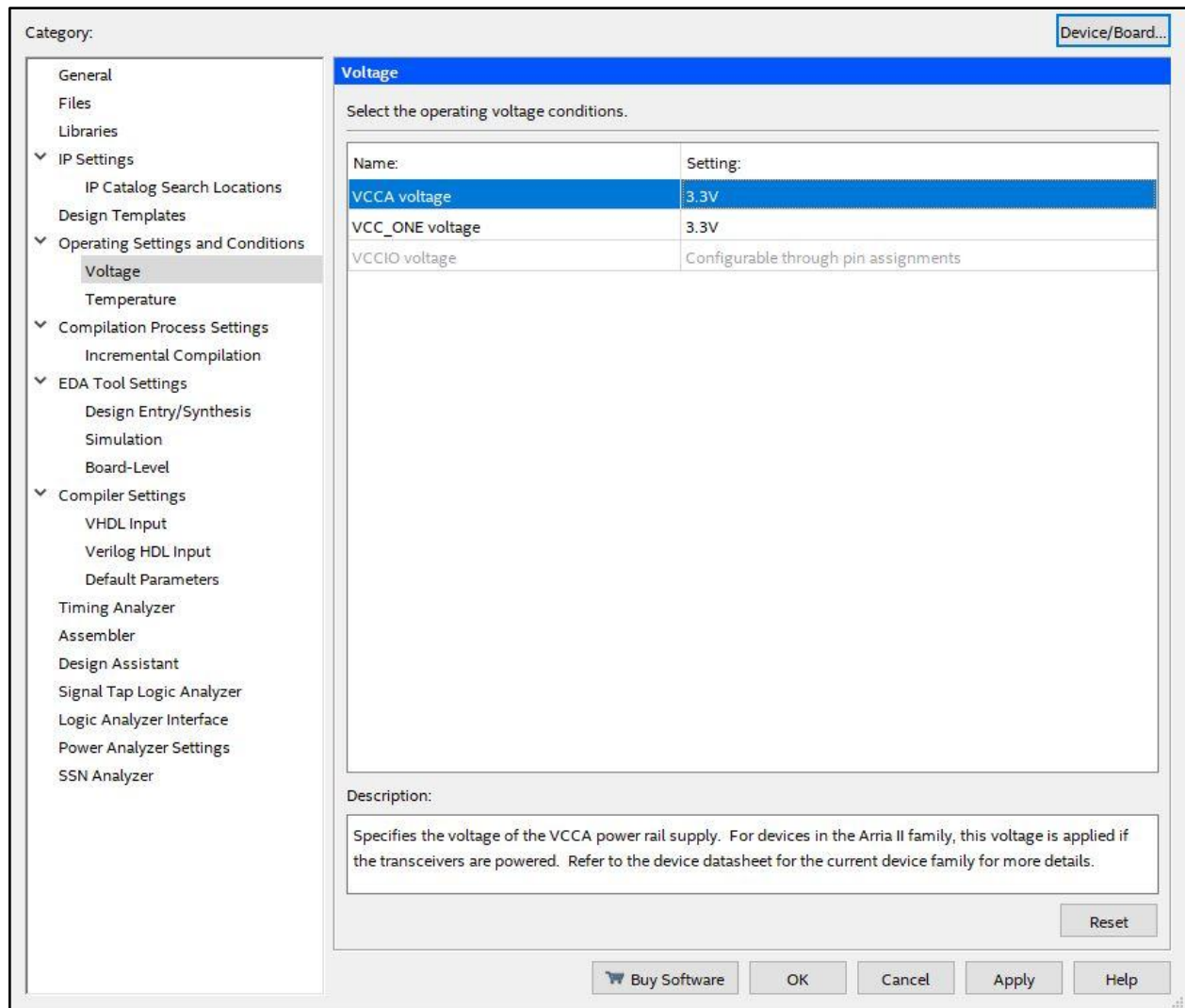


Figure 15

Now go to **Assignments>>Device>>Device and pin Options>>Voltage** and set **Default I/O standard** to **3.3-V LVTTTL** (Figure 16 and 17).

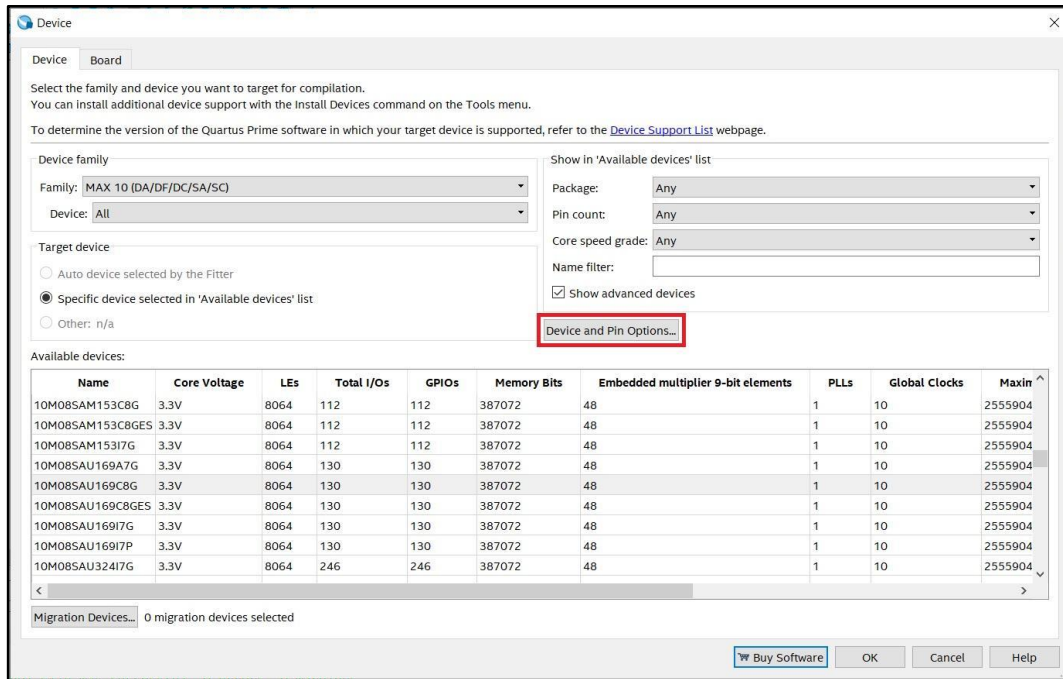


Figure 16

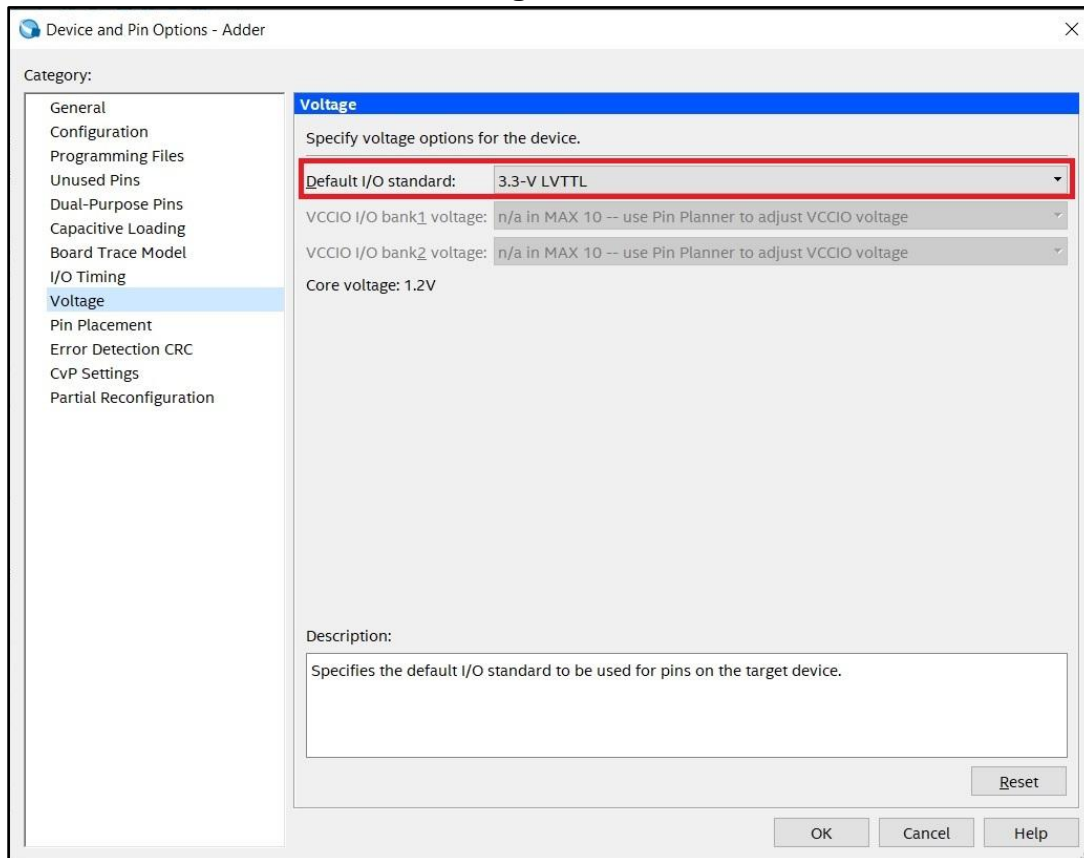
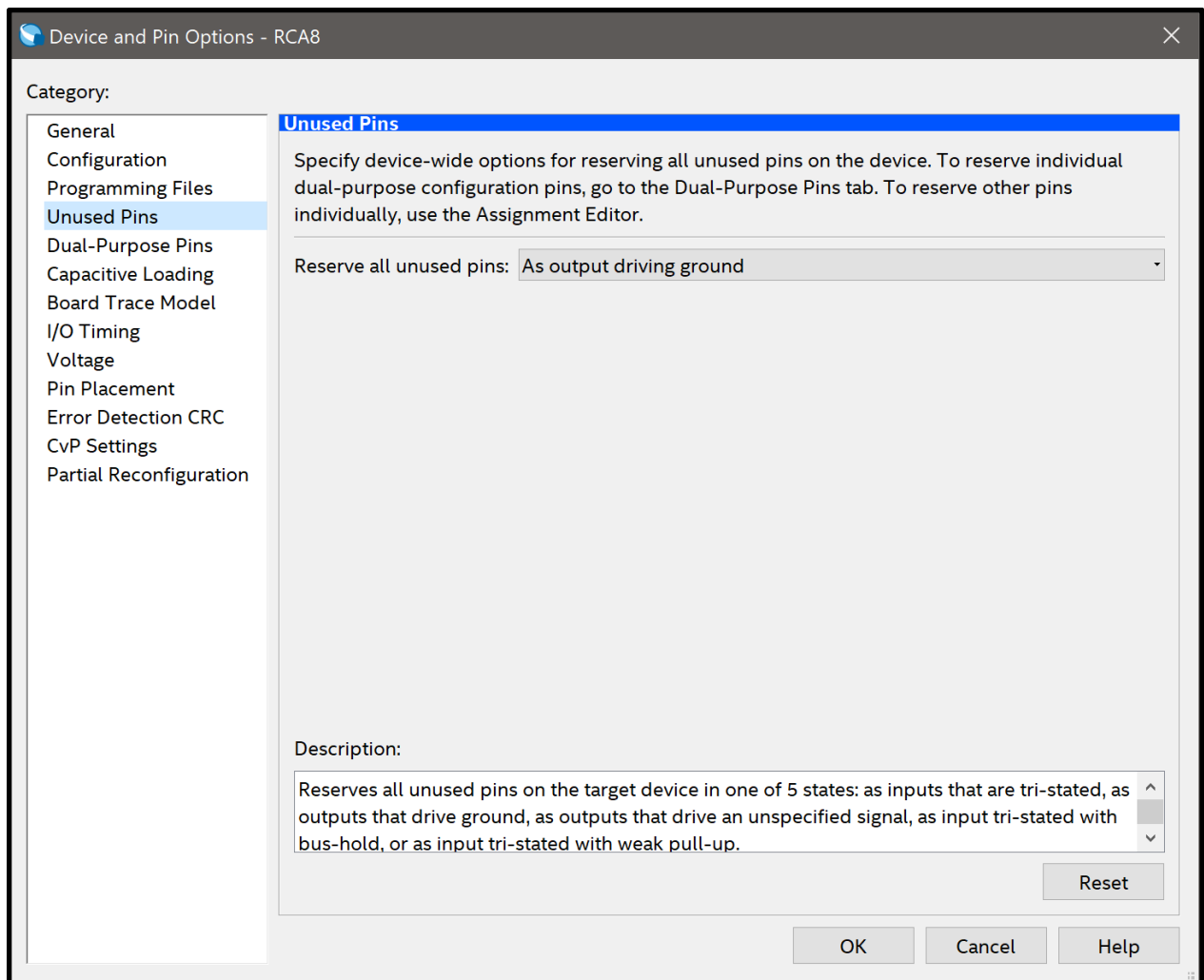


Figure 17

Now go to **Device and pin Options>>Unused Pins** and set **Reserve all unused pins** to **As output driving ground** then click **Ok (Figure 18)**.



**Figure 18**

*All unused pins are now grounded and will be off. If this is left undone, the LEDs on the Max1000 that are not in use will stay on. The pins can also be set to the tri-state option, which should ground all unused pins.*



## Programming FPGA

Once all is compiled successfully, you can view the report which is full of interesting information. Flow Summary will show how many logic elements were needed, along with pins, registers, etc. You can look through the table of contents for loads of information regarding the synthesis of your design.

Now open the **Quartus Prime Programmer** by selecting **Tools>>Programmer**. Connect your Max1000 board to the PCB Peripheral provided and then to your computer using a USB cable. A green LED on the Max1000 should light up indicating power, along with a brief flash of a red LED. Click **Hardware Setup** and you should see **Arrow USB Blaster** listed in the **Available Hardware Items** table. If the Arrow USB Blaster isn't shown in the table contact the lab instructor. Click Ok to close this window.

Click **Auto Detect** and make sure the **Mode** is set to **JTAG**. Select **10M08SA** then click Ok. You will see an entry in the file select table generate. Click on the entry then select **Add File**. Navigate to **<Project Dir>/output\_files/** and select **RippleCarryAdder.sof** or **RippleCarryAdder.pof**. SOF stands for SRAM Object Files, these are programmed directly to the FPGA fabric. They are volatile and will be erased when power is cycled. Select the checkbox under **Program/Configure**, then click **Start** to program the Max1000.

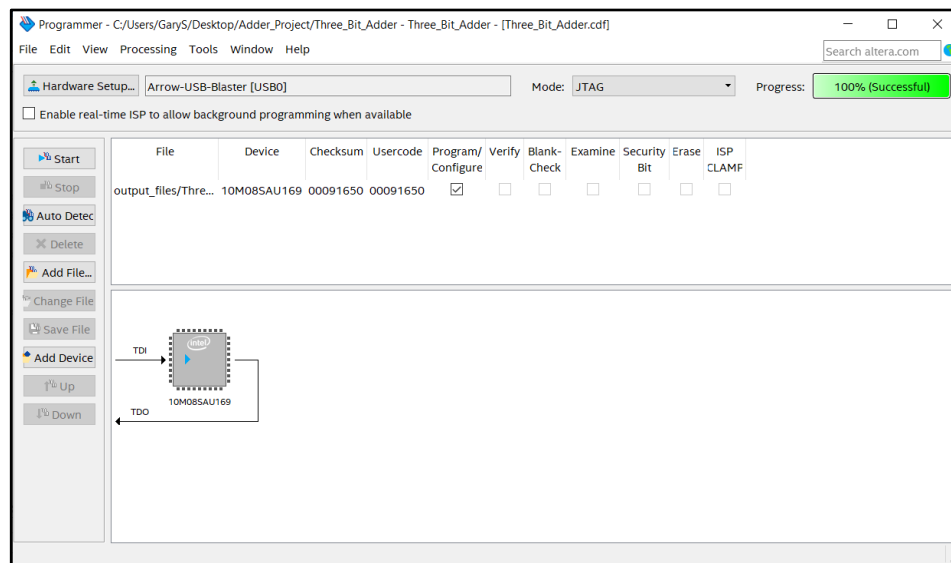


Figure 19

*POF stands for Programming Object Files, these are programmed to the Max1000's Flash memory unit. They are non-volatile and will remain on the device even after a power cycle.*

## Working with the Peripheral Board

If you set your pins correctly and upload the program the FPGA LEDs should be working. 8 of the dip switches will represent the A inputs and the 8 other switches will represent the B inputs. The sum of any combination of these switches should appear. The pin layout is set up so that the right-most LED on the FPGA is the LSB.

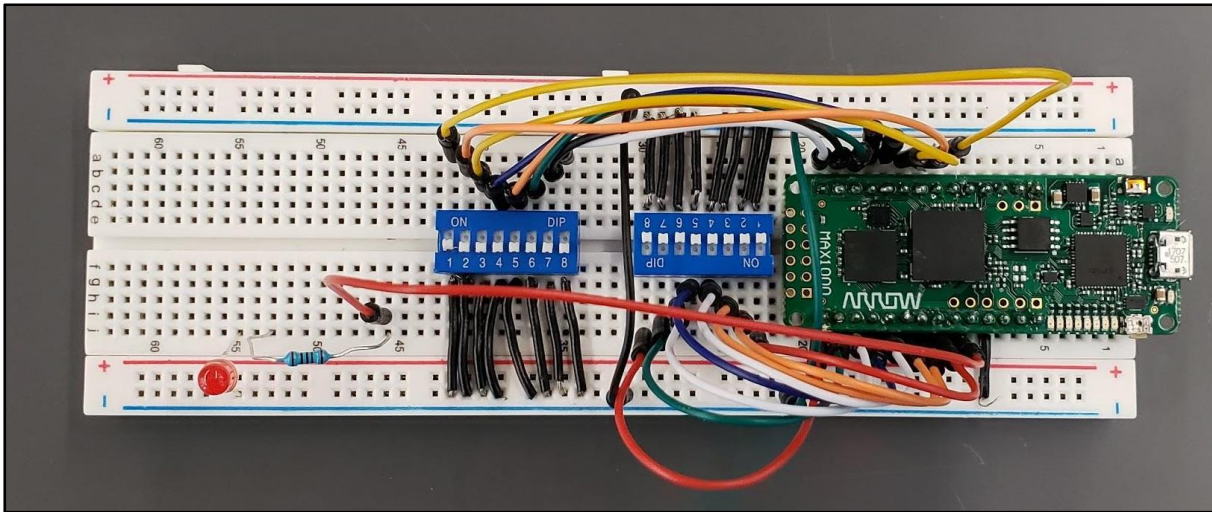


Figure 20