Making Adder Circuits in the Vivado IPI

An FPGA/Computer architecture Tutorial by Matthew Harrison
Siena College department of Computer Science

Based on circuit designs found at: https://www.electronicshub.org/half-adder-and-full-adder-circuits/

Create the Project

- Launch Vivado
- Create a new project by selecting File->Project->New… and hit *Next*
- Give the project a name and location, ensuring there are no spaces in either
- Leave the box for *Create project subdirectory* checked and hit *Next*
- Choose RTL project and check *Do not specify sources at this time,* hit *Next*
- Navigate to the *boards* tab, and select *Basys3,* then hit *next* and *finish*

Block Design

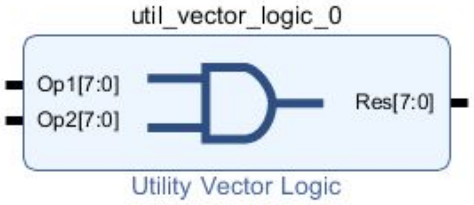
- From the *Flow Navigator* on the left, click *Create Block Design*
- Name the design “adder”, click *OK* and wait for the *Diagram* window to open

Intellectual Property (IP)

- IP refers to intangible creations that have resulted from the creativity and intellect of an individual or group, like copyright, designs, code, or patents
- Vivado uses an IP Integrator (IPI) to drag and drop blocks resembling IP into a diagram together to create an FPGA design
- Each IP block translates to hardware descriptive code to function as a component in the design such as a gate, circuit, or processor

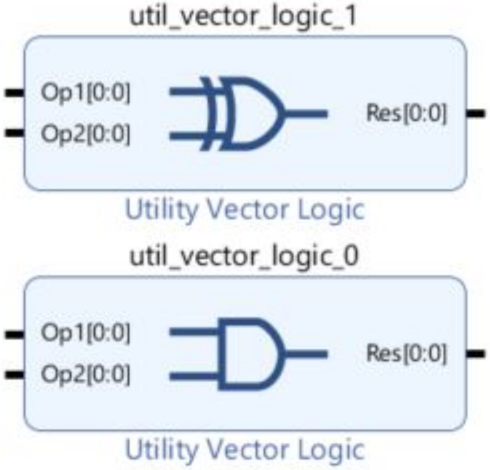
Adding Intellectual Property

- Right click inside the diagram and select *Add IP…* or click the plus sign along the top toolbar of the Diagram window
- Search for the “Utility Vector Logic” IP and drag it into the diagram. This IP block is how you will implement logic gates



Customize the IP

- Double click the IP block you just added to customize it
  - Set C_SIZE to 1 so the gate has 1-bit inputs and outputs
  - Leave C_OPERATION as “and”, then press OK
- The half adder requires two gates. Hold the ctrl key while clicking and dragging the first gate to duplicate it
- Double click the new block and change C_OPERATION to xor



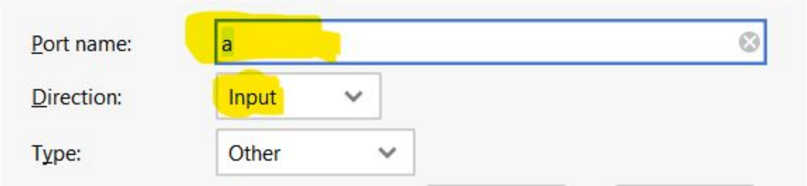
Logic

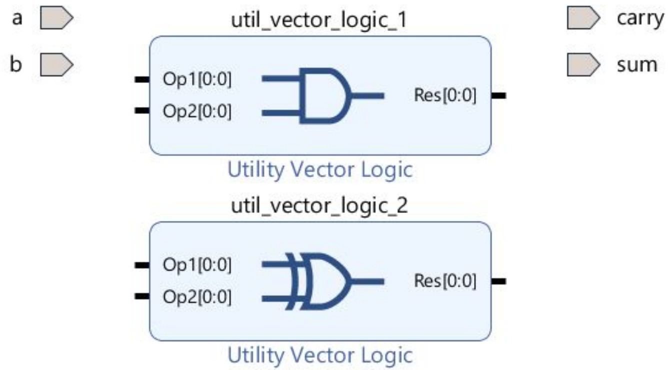
- Sum=1 when only one input is 1, and 0 otherwise. The XOR gate will give us the sum value
- When both inputs are one, sum is 0 but there is a carry out bit. The AND gate will give us the value of the carry out bit.

A	B	Sum (A XOR B)	Carry (A AND B)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Making Connections- Ports

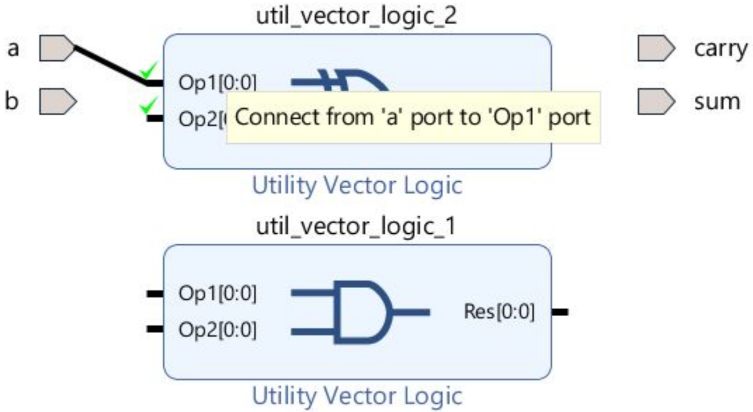
- Press ctrl+k to add a port
- Make each of the following ports with the proper direction and exact name:
  - Input, a
  - Input, b
  - Output, carryOut
  - Output, sum

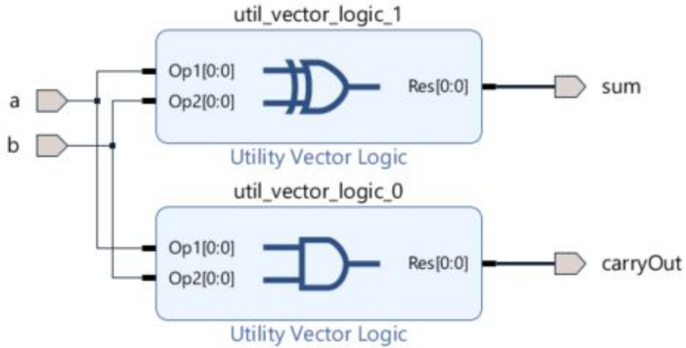




Making Connections- Wiring

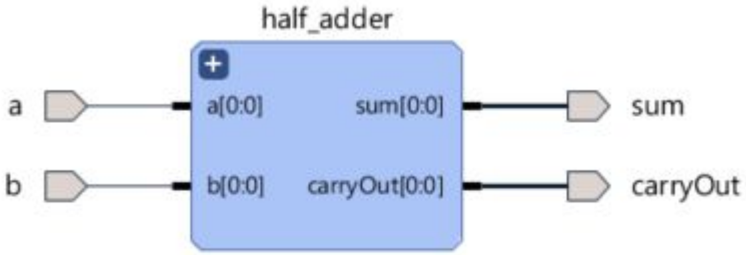
- After creating your ports, it’s time to connect them to the gates.
  - First move the XOR above the AND gate
  - Click and drag from the connectors on the gates to the desired port.
  - Connect the inputs of the gates to a and b
  - Connect the output of the AND gate to carryOut
  - Connect the output of the XOR gate to sum





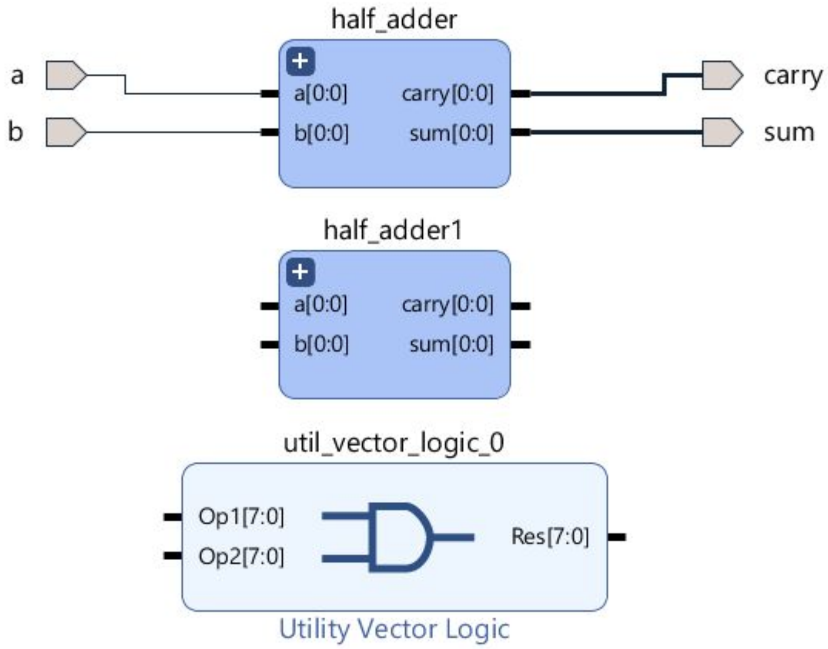
Creating a Hierarchy- Abstraction

- Now that the half adder is complete, we can create a hierarchy to hide its internal functionality.
- Select both gates by holding ctrl while clicking each of them or pressing ctrl+a
- Right click and select *Create Hierarchy*, name it half_adder
- Save your design so you do not lose progress should something go wrong



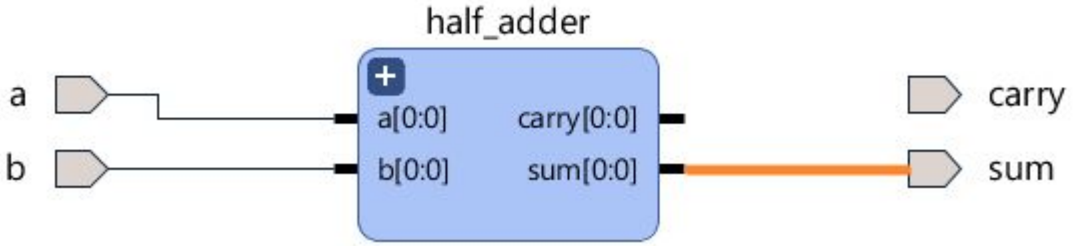
Full Adder

- A full adder requires two half adders and an OR gate.
- Copy and paste the hierarchy block you created
- Add another “Utility Vector Logic” block from the IP Catalog, and customize it to be an OR gate of size=1



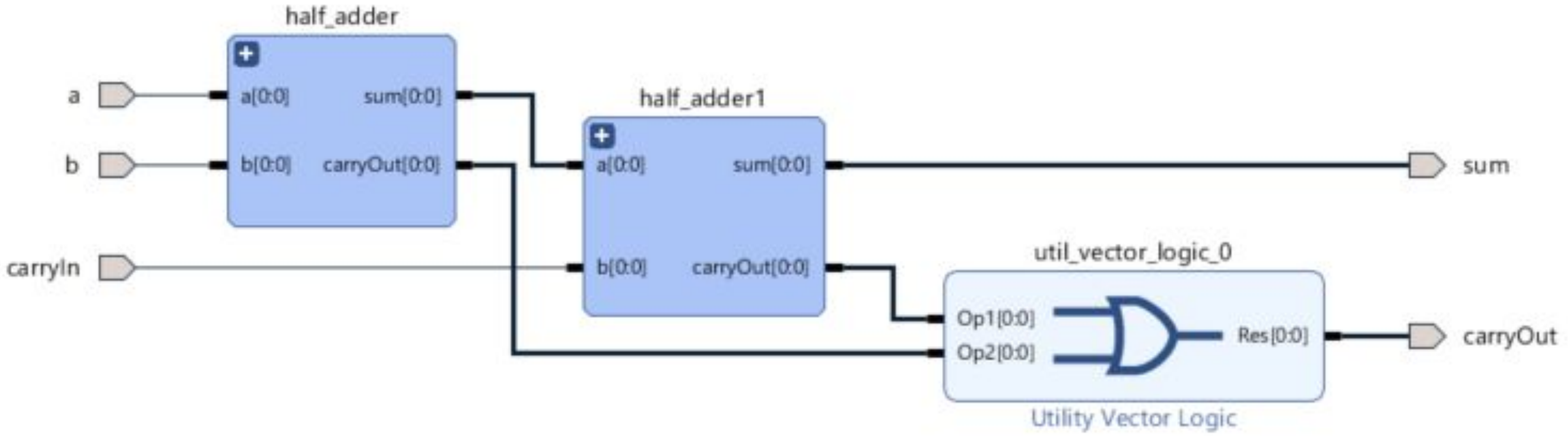
Connections

- Since we need to connect this half adder to another, we must disconnect its outputs while leaving the ports in the diagram
- It was necessary to have these outputs connected when making the hierarchy so that the resulting half adder block would have the proper inputs and outputs
- Select and delete the wires connecting the original hierarchy to its output ports. When a wire is highlighted orange like in the image below, press the delete key



Connections

- Create input port carryIn
- Connect as shown below, save your work when completed

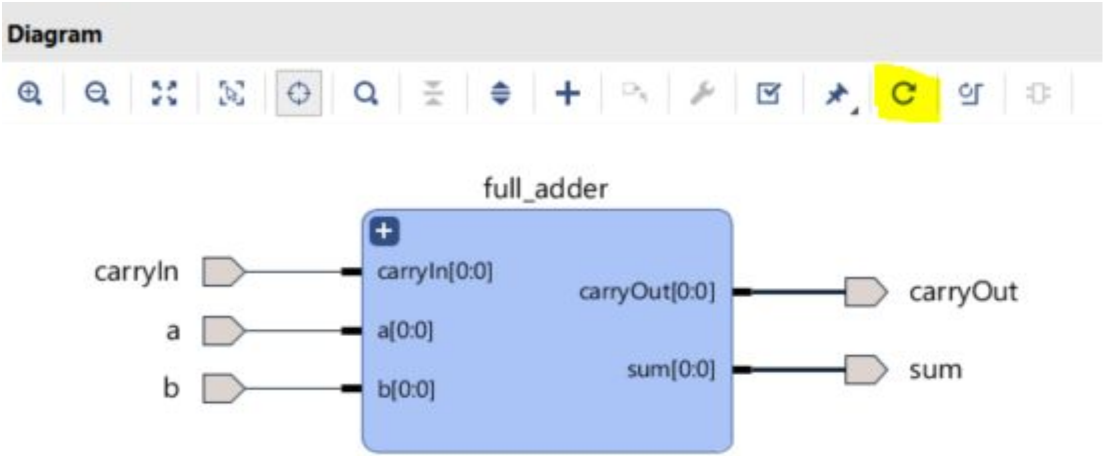


Full Adder Truth Table

a	b	carryIn	sum	carryOut
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

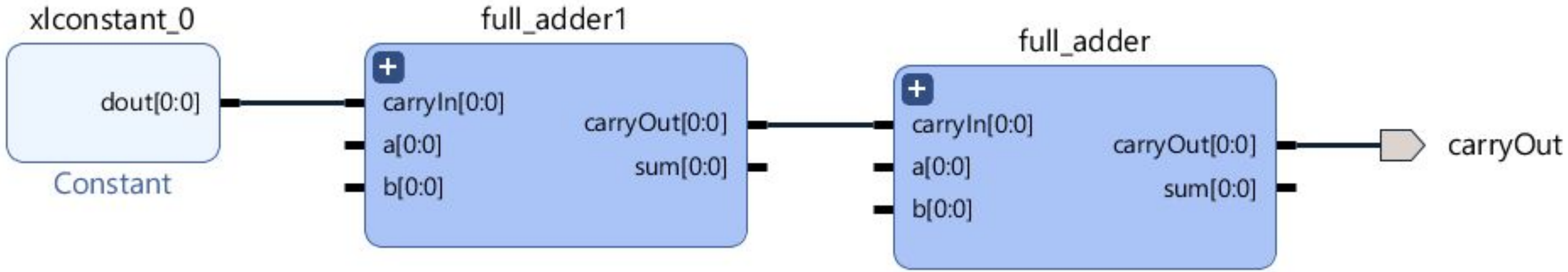
Full Adder- Abstraction

- Just like with the half adder, select all the components in the full adder diagram and make a hierarchy named full_adder
- Click the *Regenerate Layout* button (highlighted below) to make the diagram more readable (Only do this when instructed, it can negatively impact readability in more complex circuits)



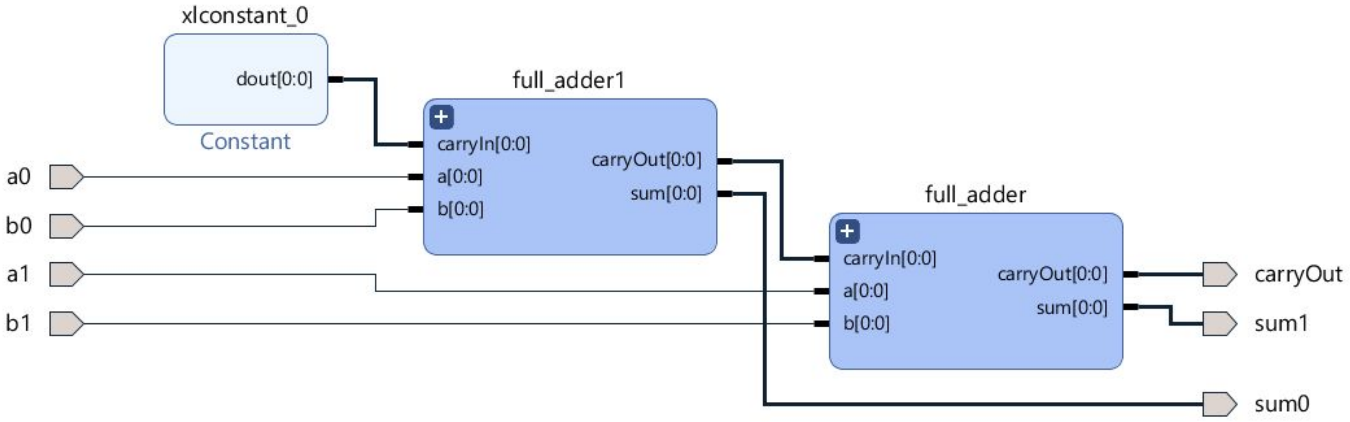
2-bit Adder

- Duplicate the full_adder block (ctrl+drag or copy/paste)
- There is no carry in bit to the first adder, so search for “constant” in the IP catalog and drag one into the design.
- Double click the block and set its value to zero
- Delete ports a, b, carryIn, and sum. These will be replaced later
- Connect the constant to carryIn of the new full_adder
- Connect carryOut of that adder to carryIn of the second



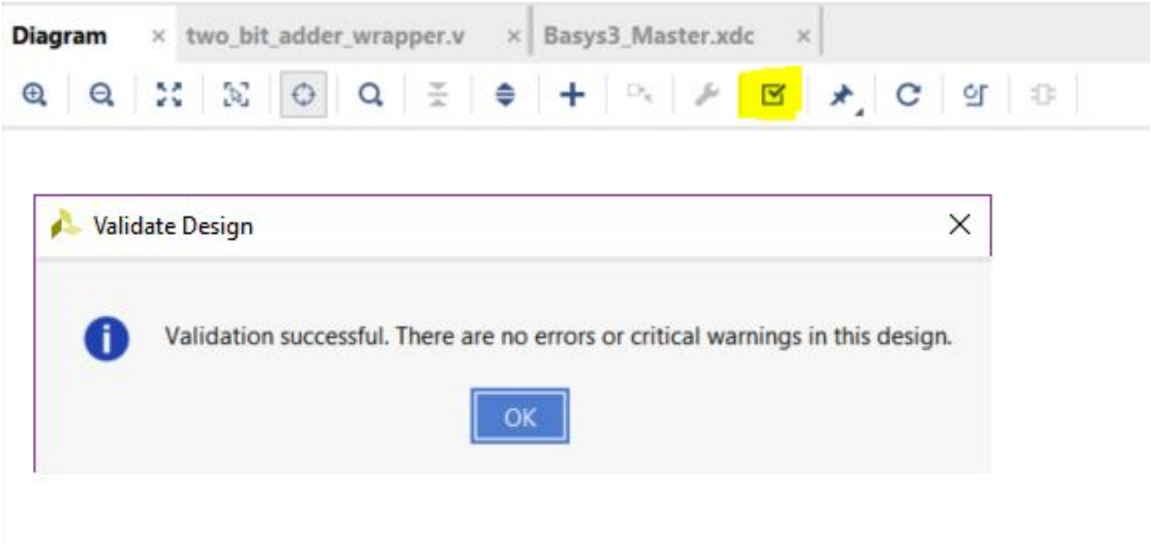
Top Level Ports

- The first adder needs input ports a0 and b0, the second needs a1 and b1
- Make an output port named sum0 for the first adder, and sum1 and carryOut for the second.
- **Port names must be named exactly as shown**



Validate Design

- Click the validate design button on the diagram window toolbar(highlighted below)
- If no mistakes were made, your design will validate successfully



HDL Wrapper

- FPGAs are programmed via a Hardware Descriptive Language, like VHDL or Verilog
- Navigate to the *Sources* window in Vivado. If you can not find this, click *Project Manager* from the Flow Navigator window on the left
- In the *Sources* tab under *Design Sources,* Right click *adder.bd* and select *Create HDL Wrapper…*
- Select *Let Vivado manage wrapper and auto-update* and hit *OK*
- The new file should be at the top of the source hierarchy(this will take a few seconds while the hierarchy updates)

Inside the Wrapper

- The generated wrapper *adder_wrapper.v* is now the "top module" of your project, similar to a main method in high level programming
- A top module instantiates other modules, but no module can instantiate it
- Explore the full source hierarchy on the next slide to see how the 2-bit adder is composed of two full adders, each composed of two half adders that are composed of two logic gates

```
module adder_wrapper
   (a0,
    a1,
    b0,
    b1,
    carryOut,
    sum0,
    sum1);
  input a0;
  input a1;
  input b0;
  input b1;
  output [0:0]carryOut;
  output [0:0]sum0;
  output [0:0]sum1;

  wire a0;
  wire a1;
  wire b0;
  wire b1;
  wire [0:0]carryOut;
  wire [0:0]sum0;
  wire [0:0]sum1;

  adder adder_i
       (.a0(a0),
        .a1(a1),
        .b0(b0),
        .b1(b1),
        .carryOut(carryOut),
        .sum0(sum0),
        .sum1(sum1));
endmodule
```

- Design Sources (1)
  - **adder_wrapper** (adder_wrapper.v) (1)
    - adder_i : adder (adder.bd) (1)
      - adder (adder.v) (3)
        - full_adder : full_adder_imp_A0IDF1 (adder.v) (3)
          - half_adder : half_adder_imp_2C7W20 (adder.v) (2)
            - util_vector_logic_1 : adder_util_vector_logic_1_0 (adder_util_vector_logic_1_0.xci)
            - util_vector_logic_2 : adder_util_vector_logic_1_1 (adder_util_vector_logic_1_1.xci)
          - half_adder2 : half_adder2_imp_A3DNBP (adder.v) (2)
            - util_vector_logic_1 : adder_util_vector_logic_1_4 (adder_util_vector_logic_1_4.xci)
            - util_vector_logic_2 : adder_util_vector_logic_2_2 (adder_util_vector_logic_2_2.xci)
          - util_vector_logic_0 : adder_util_vector_logic_0_1 (adder_util_vector_logic_0_1.xci)
        - full_adder1 : full_adder1_imp_1849P9E (adder.v) (3)
          - half_adder : half_adder_imp_CW98II (adder.v) (2)
            - util_vector_logic_1 : adder_util_vector_logic_1_8 (adder_util_vector_logic_1_8.xci)
            - util_vector_logic_2 : adder_util_vector_logic_2_6 (adder_util_vector_logic_2_6.xci)
          - half_adder2 : half_adder2_imp_18PILJW (adder.v) (2)
            - util_vector_logic_1 : adder_util_vector_logic_1_7 (adder_util_vector_logic_1_7.xci)
            - util_vector_logic_2 : adder_util_vector_logic_2_5 (adder_util_vector_logic_2_5.xci)
          - util_vector_logic_0 : adder_util_vector_logic_0_3 (adder_util_vector_logic_0_3.xci)
        - xlconstant_0 : adder_xlconstant_0_1 (adder_xlconstant_0_1.xci)

Constraints

- Vivado needs a constraints file so it knows what pins on the FPGA hardware the HDL code is referring to
- Download the basys3.xdc located here:
  - https://github.com/mfHarrison/CSIS220Tutorials/blob/master/adders/basys3.xdc
- Add Sources the same way as you created the verilog files at the beginning of the project, but this time select *Add or Create Constraints, Add Files,* then locate and open basys3.xdc

Modifying the Master Constraint File

- Find the section of the constraints file labeled *Switches*
    - Uncomment the lines with arguments *sw[0]* and replace it with our port name *b0*
    - Uncomment and replace lines for *sw[1]* with *b1*
    - Uncomment and replace lines for *sw[2]* with *a0*
    - Uncomment and replace lines for *sw[3]* with *a1*
- Find the LED section
    - Uncomment and replace lines for *LED[13]* with *sum0*
    - Uncomment and replace lines for *LED[14]* with *sum1*
    - Uncomment and replace lines for *LED[15]* with *carryOut*
- Any other lines in the constraint file should be commented

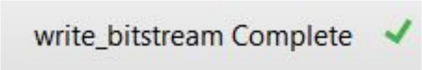
Generate a Bitstream

- Verify that the generated HDL wrapper has been set as the Top Module by checking the sources tab. There should be three squares next to the name, as highlighted in the image above
  - If it is not the top module, right click the name and choose *Set as top*
- As long as Vivado has a Top Module and constraints, it can generate the Bitstream needed to program the FPGA.
- Click on *Generate Bitstream* near the bottom of the Flow Manager on the left
- Click *OK* at the prompts, leaving all the options as the defaults.

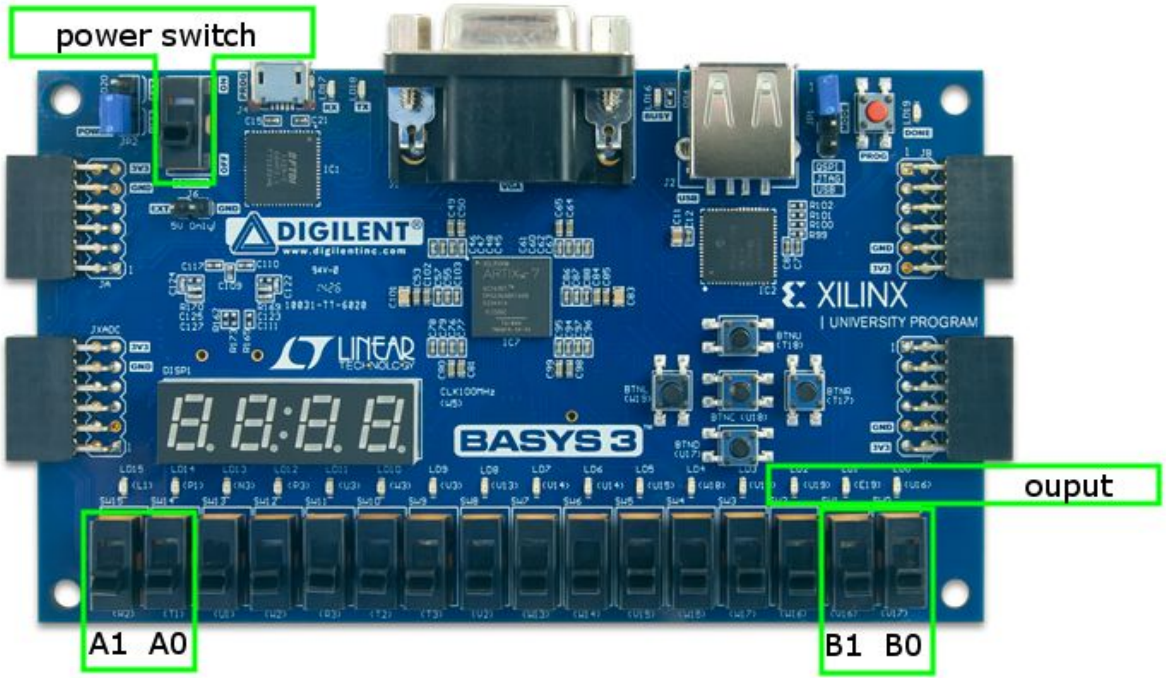
Generate a Bitstream

- A few progress bars will open and close after beginning this process
- In the top right corner of Vivado, there is a progress window that shows what task it is working on

- Vivado has to synthesize and implement the design before it can generate a bitstream for our device, so it may take a few minutes to complete
- While waiting for the bitstream:
    - Connect the Basys3 to the PC using a micro USB cable if you have not already
    - Construct a truth table for our 2-bit adder. Do not include a carry in bit, just show the 3-bit output of a1 a0 + b1 b0
    - Wait until the dialogue window or progress window inform you the bitstream is complete.



Important Labels



https://reference.digilentinc.com/_media/reference/programmable-logic/basys-3/basys-3-2.png

Programming & Testing

- Turn the power switch on
- Select “Open Hardware Manager” from the bottom of the Flow Navigator
- Select *Open Target> Autoconnect*
- Now select *Program Device*
- Click Program
- Refer to the previous slide for the input/output layout of the board
- Verify that your output matches your truth table