

TUTORIAL On USING XILINX ISE DESIGN SUITE 14.6: Mixing VHDL and Schematics “Design of 4-bit Ripple Carry Adder” Using the Spartan-6 (NEXYS3 Board)

Shawki Areibi

August 13, 2023

1 Introduction

The objective of this tutorial is to show you how VHDL can be integrated with Schematic Capture to implement complex designs using Xilinx ISE tools. At this point, the student should have read and understood the following documents:

1. Tutorial on Using Xilinx ISE Design Suite 14.6 (Design Entry using Schematic Capture “Half Adder”).
2. Tutorial on Using Xilinx ISE Design Suite 14.6 (Behavioral Simulation of a “Half Adder” Circuit).
3. Tutorial on Using Xilinx ISE Design Suite 14.6 (Design Entry using VHDL “Full Adder”).

In this tutorial you will learn the following topics:

1. Creating a top level schematic project.
2. Designing a Half Adder using VHDL.
3. Designing a Full Adder by using the Half Adder as a component.
4. Transforming the Full Adder design into a Macro Symbol.
5. Using the Full Adder Macro Symbols to design a 4-bit adder based on Schematic Capture.
6. Connecting the 4-bit Adder to I/O of the FPGA board.
7. Synthesizing, Implementing and reconfiguring the FPGA chip.
8. Testing the design.

2 Design of a 4-bit Adder Circuit

Before designing a schematic, you must derive Boolean equations for your circuit. Once you have derived these equation, you may verify and implement them using the Xilinx software.

We will use hierarchy to design the 4-bit adder. In other words, we will first design a Half Adder Circuit. Then we will use two Half Adder Components to design a Full Adder. Finally, we will use four components of the Full Adder to implement the 4-bit Ripple Carry Adder.

2.1 Design of a Half Adder Circuit

Specifications

The half adder will take in two binary digits in order to produce a sum bit and carry bit.

Define inputs and outputs

Inputs: two bits (A and B).

Outputs: a sum bit (S) and a carry bit (C).

Create a truth table

Inputs		Outputs	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Derive Boolean equations

As we can see from the truth table, the carry bit is only set when both A and B are set. This is an AND operation. The sum is set when only one of the two inputs are set. This is an Exclusive-OR operation.

Therefore,

$$C = AB$$

$$S = \overline{A}B + A\overline{B} = A \oplus B$$

2.2 Design of a Full Adder Circuit

Specifications

The full adder will take two binary digits and a carry-in bit and produce a sum bit and a carry-out bit.

Define inputs and outputs

Inputs: two binary digits (A and B) and a carry-in bit (Ci).

Outputs: a sum bit (S) and a carry-out bit (Co).

Create a truth table

Inputs			Outputs	
A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Derive Boolean equations

As can be seen in the truth table, the carry bit is only set when both A and B are set or when one of these and the carry-in bit is set. The sum bit is set when there is an odd number of 1s across the inputs. This function can be implemented by cascading two 2 input Exclusive-OR function gates.

$$C_o = AB + AC_i + BC_i = AB + C_i(A \oplus B)$$

$$S = \bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + ABC_i = (A \oplus B) \oplus C_i$$

This circuit is constructed from two half adders and an OR gate.

2.3 Design of a 4-bit ‘Ripple Carry’ Adder Circuit

A parallel binary adder is a digital circuit that produces the arithmetic sum of two binary numbers using only combinational logic. The parallel adder uses n full adders in parallel, with all input bits applied simultaneously to produce the sum as shown in Figure 1. The full adders are connected in cascade, with the carry output from one full adder connected to the carry input of the next full adder. Since a ‘1’ carry may appear near the least significant bit of the adder and yet propagate through many full adders to the most significant bit, just as a wave ripples outwards from a pebble dropped in a pond, the parallel adder is referred to as ripple carry adder.

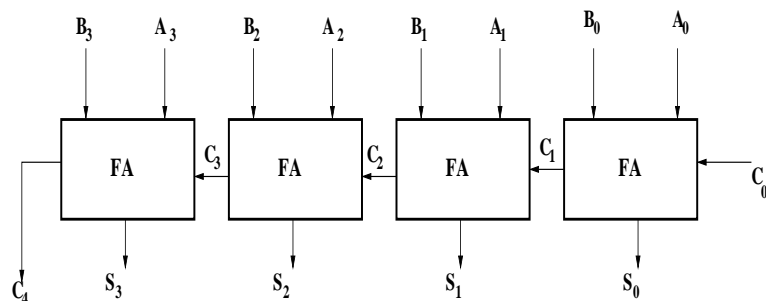

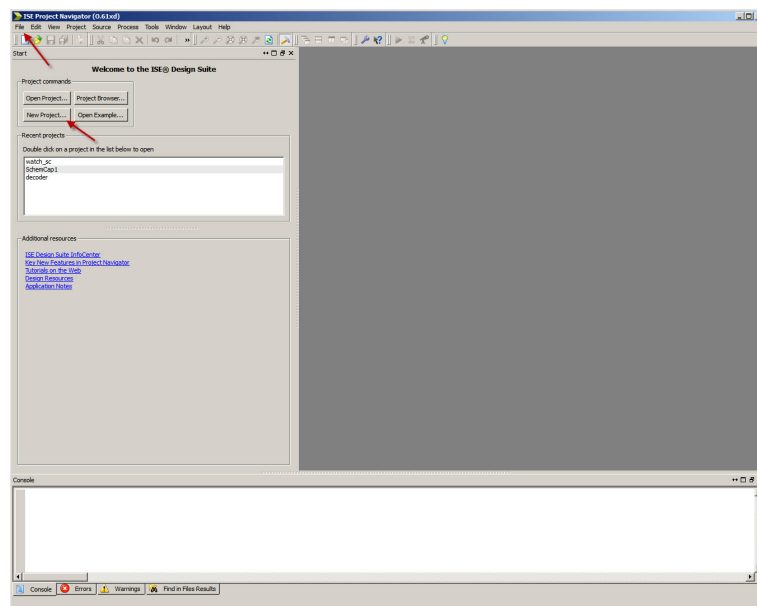


Figure 1: 4-Bit Adder

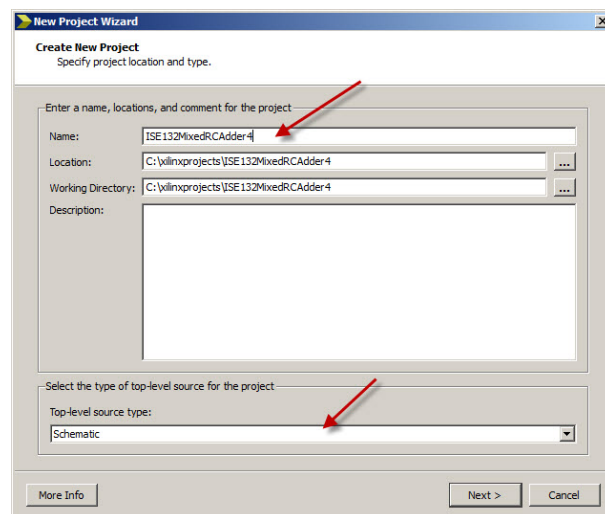
3 Starting a new project

Since we wish to combine VHDL with schematic capture, we will create a new project with schematic as the Top-Level Module Type.

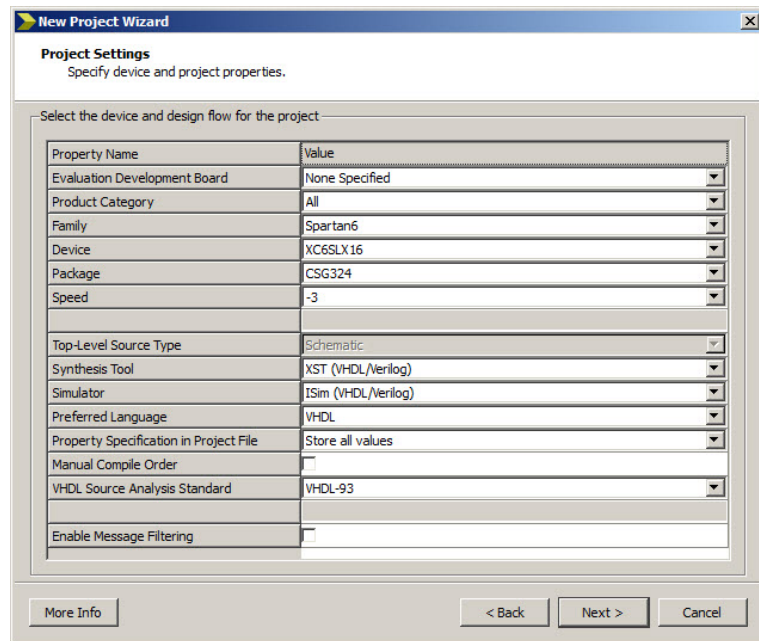
1. Load the Project Navigator from the  → **All Programs** → **Xilinx ISE Design Suite 14.6** → **ISE Design Tools** → **Project Navigator**.
2. The Project Navigator window will appear.



3. Select either **File** → **New Project** or click on the **New Project** tab.
4. The New Project Wizard dialog box will appear. Specify the directory in which you want to store the project in and name the project “ISE132MixedRCAdder4”. In the **Top-Level Source Type** section select **Schematic** and click **Next**.



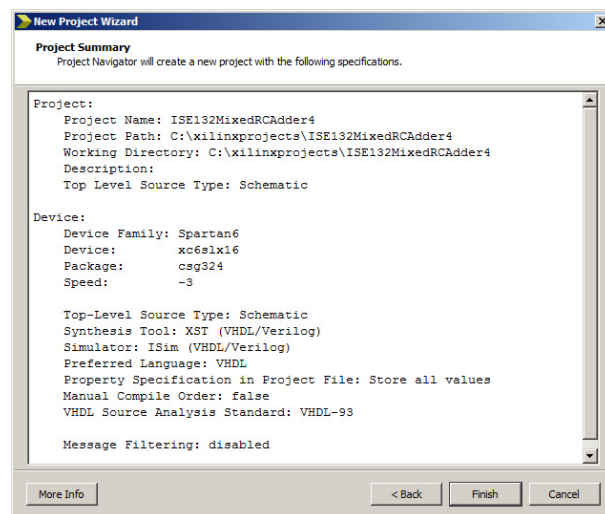
5. Another New Project Wizard dialog box will appear prompting you for device, synthesis and simulation settings for the project.



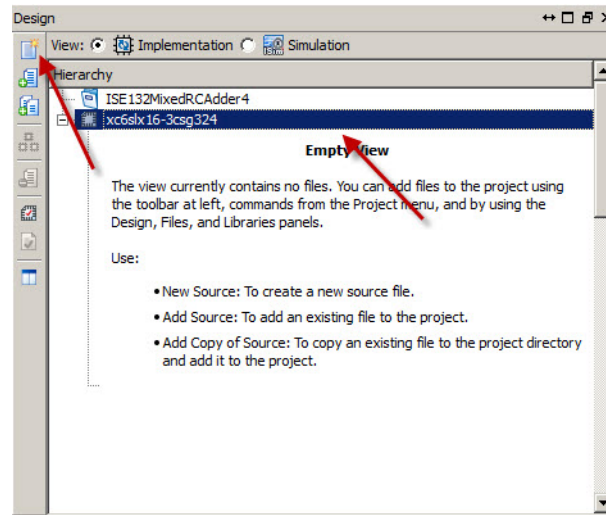
In this dialog box verify the following settings:

- **Family** → **Spartan6**.
- **Device** → **XC6SLX16**.
- **Package** → **CSG324**.
- **Speed Grade** → **-3**.
- **Synthesis Tool** → **XST (VHDL/Verilog)**.
- **Simulator** → **ISim (VHDL/Verilog)**.
- **Generated Simulation Language** → **VHDL**.

6. If the information is correct click **Next**. The following window will appear and click **Finish**.

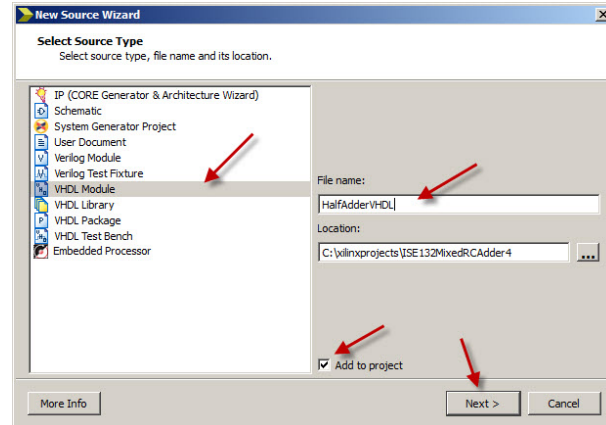


7. The next dialog box will allow you to create a new source file and add it to the project.



Click on the  **New Source** button.

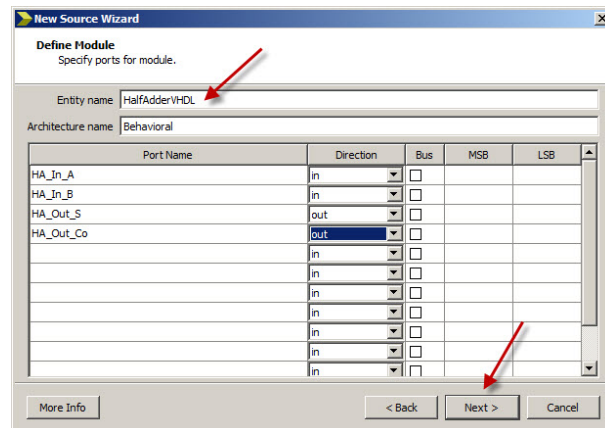
8. In the new dialog box that appears, select **VHDL Module** from the list of file types and enter “HalfAdderVHDL” as the file name.



The default location is the current project directory and can be left as is. Ensure the **Add to Project** box is selected and click the **Next** button.

Verify the information in the next dialog box and click **Finish**.

9. If you wish to add an existing source file to the project the **Add Existing Sources** dialog allows you to do this. Since we are only using new source files in this project we will not worry about this now.
10. In the **Define VHDL Source** dialog box that appears, add ports **HA_In_A**, **HA_In_B** as inputs and **HA_Out_S** and **HA_Out_Co** as outputs. Click **Next** when this is done.



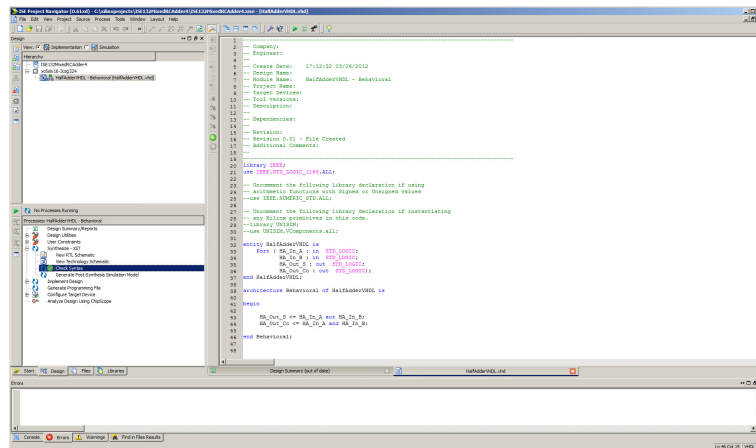
11. The next dialog box will allow you to confirm the previous choices. Click **Finish** if they are correct.
12. At the completion of the last step the VHDL editor window should now have the template code with the variables that you entered. Modify the code to match the following:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity HalfAdderVHDL is
    Port ( HA_In_A : in  STD_LOGIC;
          HA_In_B : in  STD_LOGIC;
          HA_Out_S : out STD_LOGIC;
          HA_Out_Co : out STD_LOGIC);
end HalfAdderVHDL;

architecture Behavioral of HalfAdderVHDL is
begin
    HA_Out_S <= HA_In_A xor HA_In_B;
    HA_Out_Co <= HA_In_A and HA_In_B;
end Behavioral;
```

13. We must now check that the syntax of the VHDL code is correct. In the *Project Navigator* window highlight the VHDL source in the **Sources in Project** pane. Expand **Synthesize - XST** if it is not already and double click **Check Syntax**. A process will be spawned to verify the VHDL code for syntax errors.
 - If there are none, a green check mark will appear next to the **Check Syntax** process as seen in the Figure below.



- If errors were found, A red cross will appear next to the **Check Syntax** process. The errors will be listed in the **Errors** tab of the **Transcript** window located at the bottom of the *Project Navigator* window. In it you will see a listing of the errors and the lines on which they occur. If you double click on an error message a red dot will appear in the VHDL editor pane next to (or close to) the line where the error was found.

14. Once you have eliminated all of the errors, save your work.

15. We will now create a new VHDL module (Full Adder) that will utilize the Half Adder design again using the VHDL Editor.

16. The following is code that you can utilize for the Full Adder which incorporates the Half Adder Design as a component. This listing can be in an already saved file in a certain directory and can be added as a copy to the current project.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FullAdderVHDL is
    Port ( FA_In_A : in  STD_LOGIC;
          FA_In_B : in  STD_LOGIC;
          FA_In_Ci : in  STD_LOGIC;
          FA_Out_S : out STD_LOGIC;
          FA_Out_Co : out STD_LOGIC);
end FullAdderVHDL;

architecture struc_fulladder of FullAdderVHDL is

    component HalfAdderVHDL
        port (HA_In_A, HA_In_B: in std_logic;
             HA_Out_S, HA_Out_Co: out std_logic);
    end component;

    signal sig_sum_1, sig_co_1, sig_co_2: std_logic;

begin
    HA1: HalfAdderVHDL
```



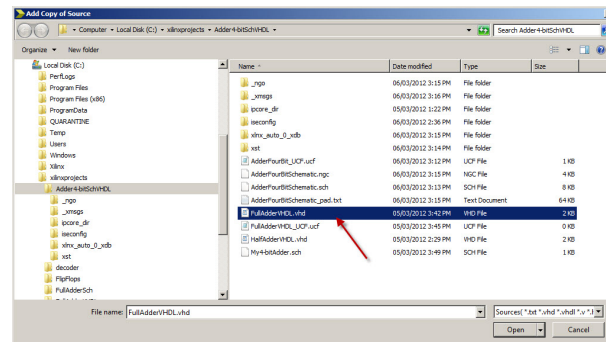
```

        port map (FA_In_A, FA_In_B, sig_sum_1, sig_co_1);
    HA2: HalfAdderVHDL
        port map (sig_sum_1, FA_In_Ci, FA_Out_S, sig_co_2);
    FA_Out_Co <= sig_Co_2 or sig_co_1;
end struc_fulladder;



```

17. Click on the  **Add Copy of Source** button.

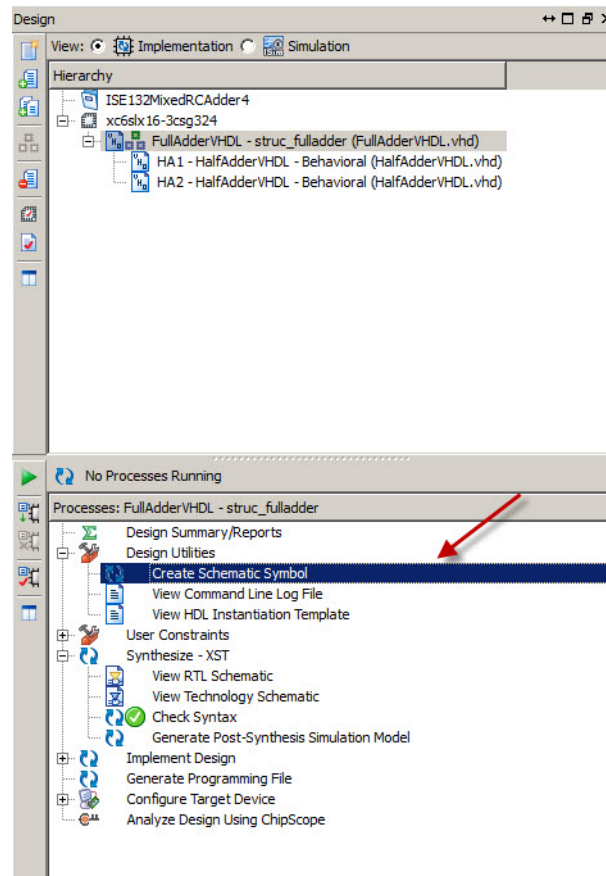
18. The system will show the following window. Assuming that you named the copy of the Full Addder VHDL code (FullAdderVHDL.vhd) then highlight it and add it to the project.





19. At this point you have a complete design of a Full Addder using purely VHDL code.


20. We must now check that the syntax of the VHDL code is correct. In the *Project Navigator* window highlight the Full Addder VHDL source in the **Sources in Project** pane. Expand  **Synthesize - XST** if it is not already and double click  **Check Syntax**. A process will be spawned to verify the VHDL code for syntax errors. Again, if there are none, a green check mark will appear next to the the **Check Syntax** process.

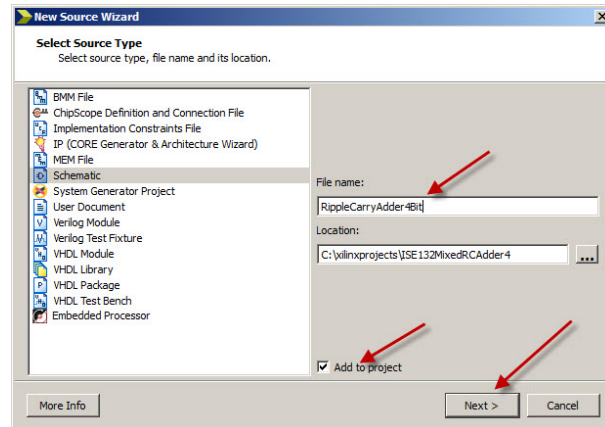
21. The next step we will create a “Schematic Symbol” of the Full Addder so that we can use it in our Schematic Editor to build the 4-bit Ripple Carry Addder.



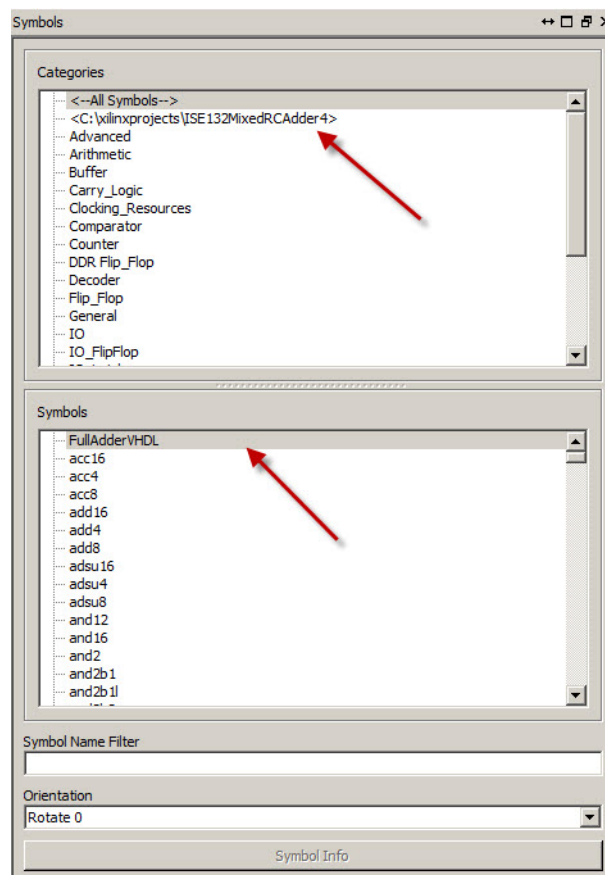
Expand the  **Design Utilities** if it is not already and double click  **Create Schematic Symbol**. If the process is successful a green check will appear next to the process name. Output from the process is shown in the **Transcript** window at the bottom of the Project Navigator window. We are now ready to use the symbol in a schematic.

4 Designing the 4-bit Ripple Carry Adder using the Macro

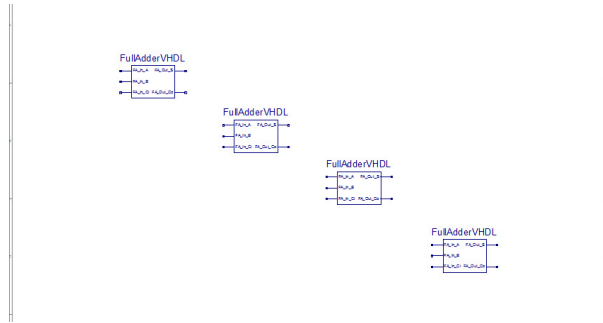
1. Now return to the Project Navigator and add a schematic source to the project. Make sure that **xc6lx16-3cg324** is highlighted and click on the  **New Source** button. A new menu will appear again as seen below:



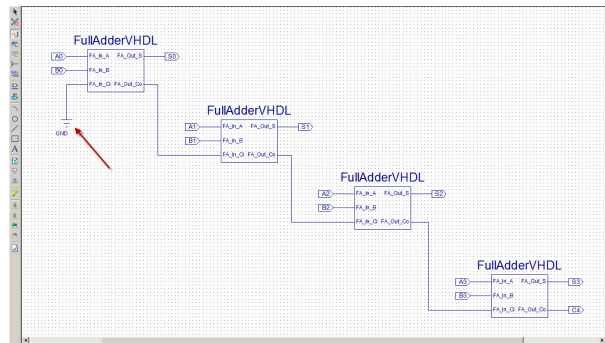
2. In the **Symbols** tab of the **Options** toolbar, the project directory will be listed in the **Categories** pane. Highlight this entry and the **FullAdderVHDL** symbol that you have created will be listed in the **Symbols** pane. You can use it in the same way you use the other components.



3. Instantiate four FullAdder Symbols and place them on the schematic.



4. Connect them together to create a 4-bit Ripple Carry Adder.
5. Add markers as you have learned from previous tutorials.
6. Change the names of the markers to suitable names as shown in the figure below.






7. Connect the Carry In C_{in} of the least significant Full Adder to Ground.
8. Now you are ready to synthesize, implement and configure your FPGA.

In the next section we will create the UCF file.

5 Creating a UCF File

A **User Constraint File** (UCF) is used to assign I/O pins in a design to the actual pins on the FPGA. Please refer to **Appendix B** for more information.

1. Click on the Design Tab  so that you can view the Hierarchy and Processes panes.
2. Highlight and right-click on the schematic source **RippleCarryAdder4Bit.sch** in the **Hierarchy** pane of the Project Navigator and select **New Source**  from the floating menu.
3. From the list of file types select **Implementation Constraints File**. Name the file “RippleCaryAd-der4Bit_UCF”. Ensure the **Add to Project** box is selected. Click **Next**.
4. The final dialog box is for confirming the information input in the previous dialog boxes. Click **Finish** if the information is correct.
5. In the **Hierarchy** pane ensure “RippleCarryAdder4Bit_UCF.ucf” is highlighted. In the **Processes** pane click on the “+” in  **User Constraints** to expand the section and double click **Edit Constraints (Text)** in the list.
 - The UCF file has the following format:

NET <pin name in Schematic design> LOC=P<pin number on FPGA>

NOTE: Make sure to use upper case letters for the pin names.




- Let us assign the inputs to the dip switches on the FPGA and the outputs to the LEDs. Please refer to the NEXYS 3 board schematic for more information about pin connections. We will connect ‘A0’ to Switch 0 (SW0), ‘B’ to Switch 1 (SW1), ‘S’ to LED 0 (LD0) and ‘C’ LED 1 (LD1). For further information check the link “NEXYS 3 Board Pins (FOR UCF FILE ASSIGNMENT)” on the web.
6. Enter the following statements in the **Workspace** panel:
NET A0 LOC=T10;
NET A1 LOC=T9;
NET A2 LOC=V9;
NET A3 LOC=M8;
NET B0 LOC=N8;
NET B1 LOC=U8;
NET B2 LOC=V8;
NET B3 LOC=T5;
NET S0 LOC=U16;
NET S1 LOC=V16;
NET S2 LOC=U15;
NET S3 LOC=V15;
NET C4 LOC=M11;

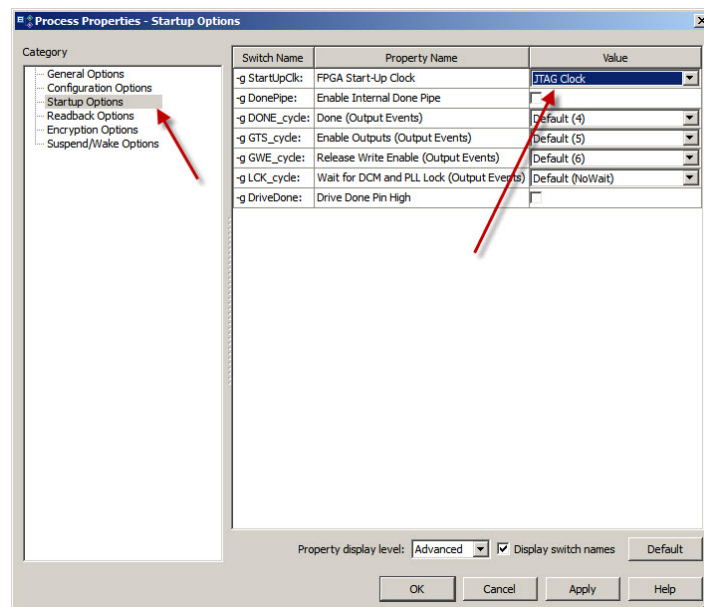
7. You can now save and close the file by pressing **File** → **Save**

You can get a better idea of our setup by looking at the Digilent NEXYS 3 Programmer’s model in **Appendix B**. We will look at this setup in more detail in the following sections.


6 Generating a Bit Stream “Compiling the Design”

Hitherto, we have examined how to design a digital circuit using the Xilinx ISE Design Suite 14.6 Software. We will now look at how to compile (generate the bit-stream) and download the design to the Digilent NEXYS 3 board.

1. Now go back to the Project Navigator window. Highlight **RippleCarryAdder4Bit.sch** in the **Hierarchy** pane. Left double click  **Synthesize - XST** in the **Processes** pane to synthesize the design.
2. When the synthesis stage has completed, you will see the following message on the console panel (process “synthesis – xst” completed successfully).
3. Left double click  **Implement Design** to implement the design.
4. When the implement stage has completed successfully you will see the following message on the console panel (process “Generate Post-Place & Route static Timing” completed successfully).
5. Right click on  **Generate Programming File** and choose properties. A window will pop as seen below:






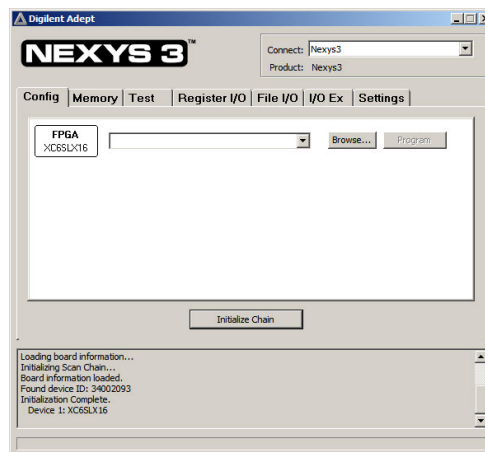
Highlight startup options and change the switch name “-g StartUpClk” value to JTAG clock. Press **Apply** then **OK**.

6. Finally, left double click  **Generate Programming File** to generate the programming file that will be downloaded to the FPGA. It is possible to go straight to generating the programming file, and the Project Navigator will determine which of the previous step need to be run to produce an up to date programming file.
7. When the generation of the bit-stream stage has completed successfully you will see the following message on the console panel (process “Generate Programming File” completed successfully).

7 Downloading the design to the FPGA

Downloading a bit-stream to the FPGA is accomplished via either the **iMPACT** tool within the ISE Project Navigator or the Digilent **Adept** tool for NEXYS 3 board.

1. Steps to download the design using the **iMPACT** tool within the ISE Project Navigator:
 - (a) Select “Configure Target Device”. **Double click** and a new window will appear stating that “No iMPACT project file exists”. You can ignore the message and **press** OK.
 - (b) A new window will then appear (ISE iMPACT).
 - (c) **Double click** on the Boundary Scan icon  Boundary Scan to start the ISE iMPACT tool.
 - (d) Click on the boundary scan box in the top menu “Initialize Chain”. Make sure that your FPGA is powered on and connected to the host. A small window will appear with the message “Do you want to continue and assign the configuration files(s)?”. **Press** yes.
 - (e) A new window will appear (Assign New Configuration File). Here you will zoom to the directory where your *.bit file exists.
 - (f) **Double click** on the file name.
 - (g) A Message window titled Attach SPI or BPI PROM will appear. **Press** Yes.
 - (h) A new window titled “Add Prom File” will appear. Just simply **close** the window.
 - (i) A message box will appear (Device Programming Properties). **Click** OK.
 - (j) Move the cursor over the device that appears in the **Boundary-Scan** tab and right click the mouse button. A menu will appear. **Press** the left mouse button and select **Program**.
 - (k) If the programming succeeds you will see the following message “Program Succeeded”.
2. Steps to download the design using the Digilent **Adept** tool:
 - (a) Load the Digilent Adept from the  **Start** → **All Programs** → **Digilent** → **Adept** → 
 - (b) The Digilent Adept window will appear as seen in the Figure below.
 - (c) Click the **Browse** icon. A new window will appear to choose your bit file.
 - (d) Zoom onto the directory where your bit file resides and double click it.
 - (e) Click the **Program** button.
 - (f) The system will start to program the device and at the bottom of the Adept Tool you will see some messages indicating that it has successfully programmed the device.



8 Testing the Design

Depending on the state of the inputs, you may or may not see some of the LEDs on the bar-graph display glowing. We have assigned our Sum (S) bits to **LEDs LD0, LD1, LD2, LD3**. The carry-out bit (C_4) is displayed on **LED LD4**.


We are using the slide switches for our A, B inputs. The A inputs is assigned to **Switch SW0-SW3**. The B input is assigned to **Switches SW4-SW7**. Moving a switch to the **ON** position puts a 1 on the input. Moving a switch to the **OFF** position puts a 0 on the input.

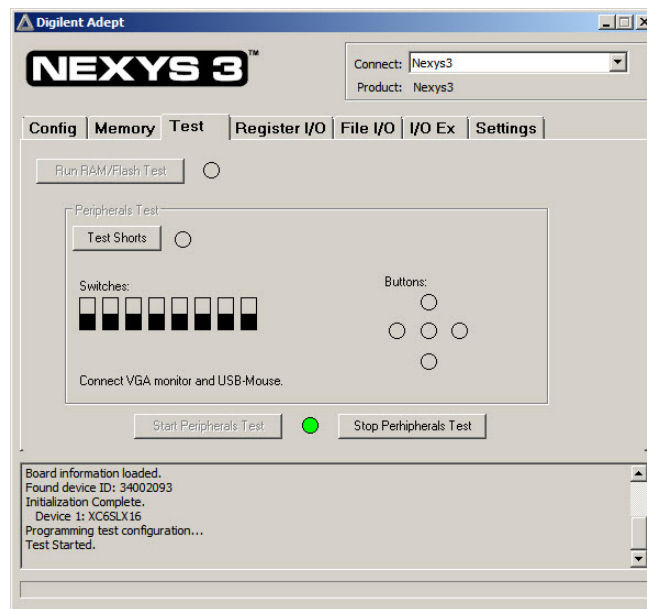
Try different combinations of inputs and verify that the circuit is working correctly.

9 Appendix A - Setting-up and Testing the NEXYS3 board

This is intended to allow the student to quickly set up the NEXYS 3 board for this tutorial. It does not attempt to explain the configuration and is in no way a substitute for the documentation provided with the board. It will allow you to use the slide switches as input and the LEDs as outputs.

1. Connect the USB cable to the NEXYS 3 board.
2. Connect the host computer to your USB cable.
3. When the power switch to the board is on a small yellow LED labeled *Done* should glow.

You can test if the Digilent NEXYS3 Board is operational by using the Digilent Adept Tool. Double click on the  icon and you will see the Digilent Adept GUI on your screen. Press the **Test** icon. A new menu will appear. Press the “Start Peripherals Test”.



The test will display different values on the 7-segment display. You can also test the switches and light emitting diodes by sliding the switches to the on-off position. Once a switch is turned on the corresponding LED will glow. You will also notice that the switches on the Digilent Adept tool will change value. You can also test the push buttons by pressing on them. You will see the color of the corresponding button on the Adept tool change from transparent to black. Once you are satisfied that the FPGA board is operational you can press the “Stop Peripherals Test”. By pressing the “Reset Button” on the FPGA you will reset the board to the factory setting where it tests all other modules on the PCB board. Power off the board using the slide switch found at the top left part of the board.

10 Appendix B - LEDs, 7-Segments and Switches

The following sections explain the connection and location of the DIP switches and LEDs of the Digilent NEXYS 3 Board.

10.1 LEDs

The Digilent NEXYS 3 Board provides a series of eight LEDs (LD0–LD7) for use. All of these LEDs are **Logic Active High** meaning that an LED segment will glow when a logic-high is applied to it. The following table show the connection from the NEXYS 3 Board to LEDs expressed as UCF constraints.

—Description	—Location
NET LD0	LOC=U16
NET LD1	LOC=V16
NET LD2	LOC=U15
NET LD3	LOC=V15
NET LD4	LOC=M11
NET LD5	LOC=N11
NET LD6	LOC=R11
NET LD7	LOC=T11

Table 1: NEXYS 3 (Light Emitting Diodes) LEDs

10.2 Seven Segment Displays

The Digilent NEXYS 3 Board provides four multiplexed 7-segment displays for use. The following tables show the connection from the NEXYS 3 Board to the 7-segment displays expressed as UCF constraints.

—Description	—Location
NET CA	LOC=T17;
NET CB	LOC=T18;
NET CC	LOC=U17;
NET CD	LOC=U18;
NET CE	LOC=M14;
NET CF	LOC=N14;
NET CG	LOC=L14;
NET DP	LOC=M13;
NET AN0	LOC=N16;
NET AN1	LOC=N15;
NET AN2	LOC=P19;
NET AN3	LOC=P17

Table 2: NEXYS 3 (7-Segment display)

10.3 Slide Switches

The Digilent NEXYS 3 board has a bank of eight slide switches which are accessible by the user.

When closed or ON, each DIP switch pulls the connected pin of the NEXYS 3 Board to ground. When the DIP switch is open or OFF, the pin is pulled high through a $10K\Omega$ resistor.

The table below shows the connections from the Digilent NEXYS 3 Board to the switches expressed as UCF constraints.

—Description	—Location
NET SW0	LOC=T10
NET SW1	LOC=T9
NET SW2	LOC=V9
NET SW3	LOC=M8
NET SW4	LOC=N8
NET SW5	LOC=U8
NET SW6	LOC=V8
NET SW7	LOC=T5

Table 3: NEXYS 3 (Slide Switches)

10.4 Push Buttons

The Digilent NEXYS 3 board has five pushbuttons (labeled BTNS through BTNR) which are accessible by the user.

When pressed, each pushbutton pulls the connected pin of the NEXYS 3 Board to ground. Otherwise, the pin is pulled high through a $10K\Omega$ resistor. The table below shows the connections from the the Digilent NEXYS 3 Board to the push buttons expressed as UCF constraints.

—Description	—Location
NET BTNS	LOC=B8
NET BTNU	LOC=A8
NET BTNL	LOC=C4
NET BTND	LOC=C9
NET BTNR	LOC=D9

Table 4: NEXYS 3 (Pushbuttons)