National University of Singapore
Department of Electrical and Computer Engineering

CG3207: COMPUTER ARCHITECTURE

Brief Introduction to VHDL Simulation Using Xilinx ISE WebPACK 13.2 and FPGA Implementation on Xilinx Spartan-3A Evaluation Kit FPGA Board

VHDL Design and Simulation Using Xilinx ISE WebPACK 13.2:
This manual illustrates step-by-step instructions on how to use Xilinx's ISE WebPACK 13.2 software for design, synthesis and simulation of digital circuits in VHDL. FPGA implementation of the VHDL codes is performed on a Xilinx Spartan-3A Evaluation Kit FPGA Board. Instructions on how to obtain the Xilinx software are detailed in the ***Xilinx 13.2 Installation*** manual, and the programming utilities to upload the developped VHDL codes onto the FPGA board are detailed in the ***AvProg 4.05 Installation*** manual.

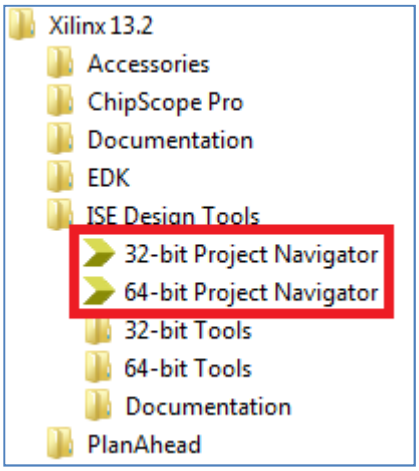
This manual is divided into eight main sections for ease of understanding. They are:

1. Creating a new VHDL project using Xilinx 13.2 ISE WebPACK.
2. Using the VHDL Editor for programming and syntax checking.
3. Simulating the design using VHDL Test Bench and ISE Simulator for combinational circuits.
4. Simulating the design using VHDL Test Bench and ISE Simulator for clocked circuits.
5. FPGA implementation of a VHDL design on the FPGA board.
6. Alternate method of FPGA implementation using PlanAhead 13.2.
7. Importing the VHDL code templates for the CG3207 project.
8. FPGA pin mappings and additional example.

Steps 1, 2, 3, 5 and 6 describe the VHDL implementation of a 1-bit half adder circuit.
Step 4 describes the VHDL implementation of a D Flipflop.

After going through this manual, you should be able to start implementing the VHDL codes for the sequencer of the 80C51 processor for the CG3207 project. This manual acts as a refresher of the EE2020 (Digital Fundamentals) module, where the basics of VHDL programming are covered.

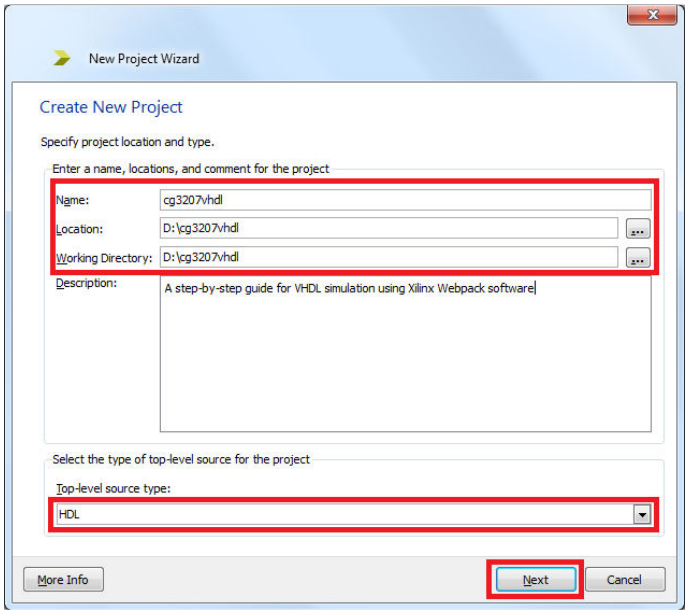
1. Xilinx Project Manager: Creating a New VHDL Project



1. To launch the Xilinx ISE 13.2 software package, either:

   (i) Double click the ***Xilinx 13.2*** icon on your desktop
   or
   (ii) Select ***Start Menu → All Programs → Xilinx ISE Design Suite 13.2 → ISE Design Tools → 32-bit Project Navigator***.

   By default, 64-bit operating systems will use the 64-bit Project Navigator when the desktop icon is double clicked. Either of the 32-bit Project Navigator or 64-bit Project Navigator can be used depending on your preference.

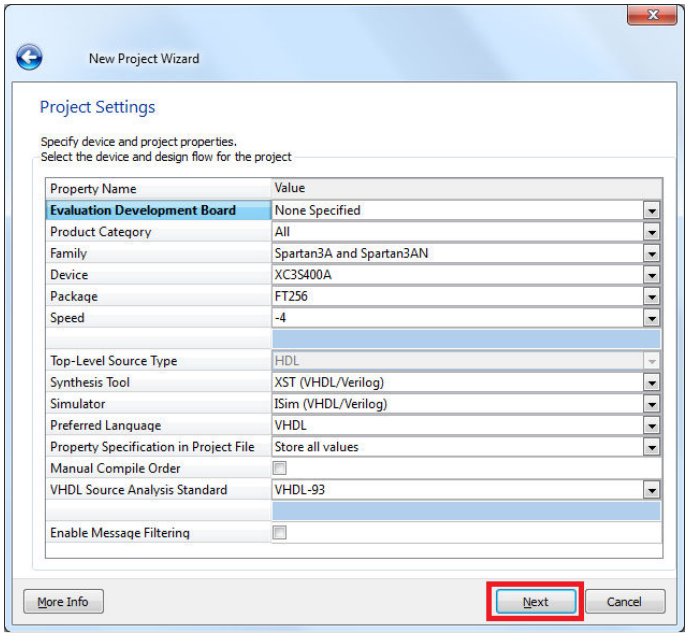


2. After launching the ***Project Navigator***, select ***File → New Project***.

   The ***Create New Project*** window will appear.

   (i) Specify a ***Name*** for the project.
   Example: ***cg3207vhdl***
   (ii) Specify a ***Location*** for the project.
   Example: ***D:\cg3207vhdl***
   (iii) Specify the ***Working Directory*** of the project.
   Example: ***D:\cg3207vhdl***
   (iv) Select ***HDL*** as the Top-level source type.

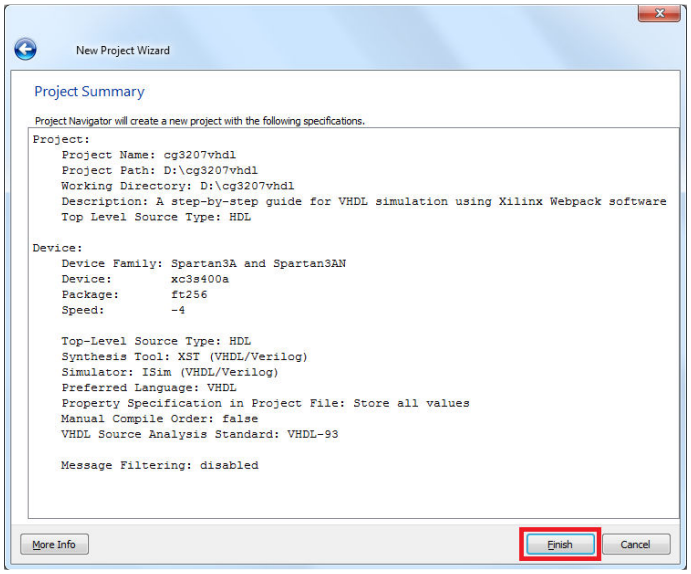
   Click ***Next*** to continue.



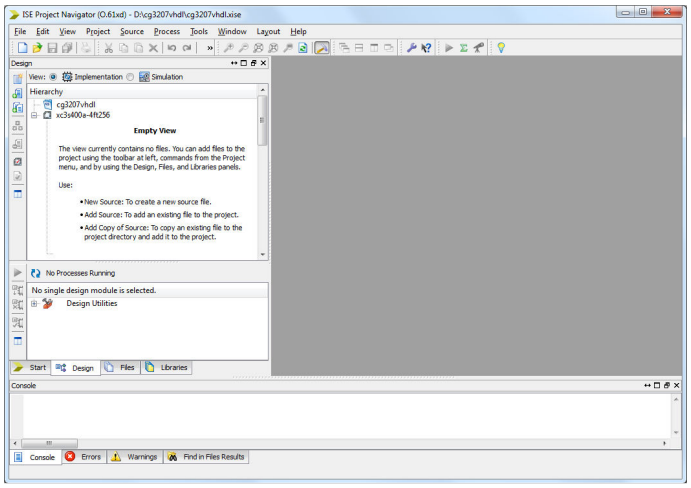
3. The ***Project Settings*** window will appear. Set the following:

(i) Evaluation Development Board: ***None Specified***
(ii) Product Category: ***All***
(iii) Family: ***Spartan3A and Spartan 3AN***
(iv) Device: ***XC3S400A***
(v) Package: ***FT256***
(vi) Speed: ***-4***
(vii) Synthesis Tool: ***XST (VHDL/Verilog)***
(viii) Simulator: ***ISim (VHDL/Verilog)***
(ix) Preferred Language: ***VHDL***
(x) Property Specification in Project File: ***Store all values***
(xi) Manual Compile Order: ***Unchecked***
(xii) VHDL Source Analysis Standard: ***VHDL-93***
(xiii) Enable Message Filtering: ***Unchecked***

Click on ***Next***.



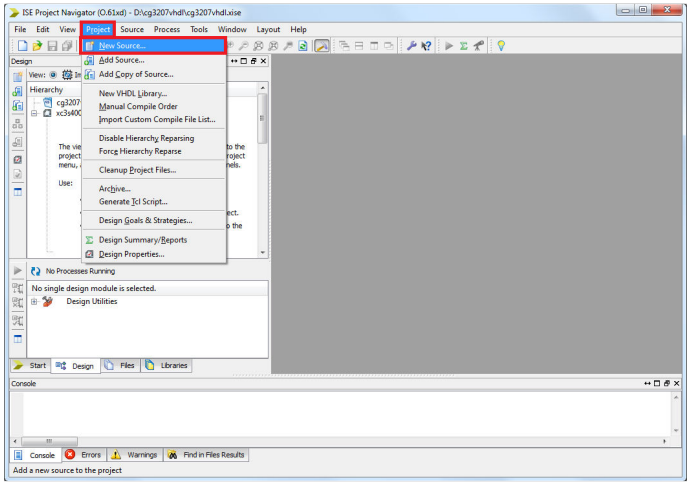
4. Click on ***Finish*** to continue to the ISE Workspace.



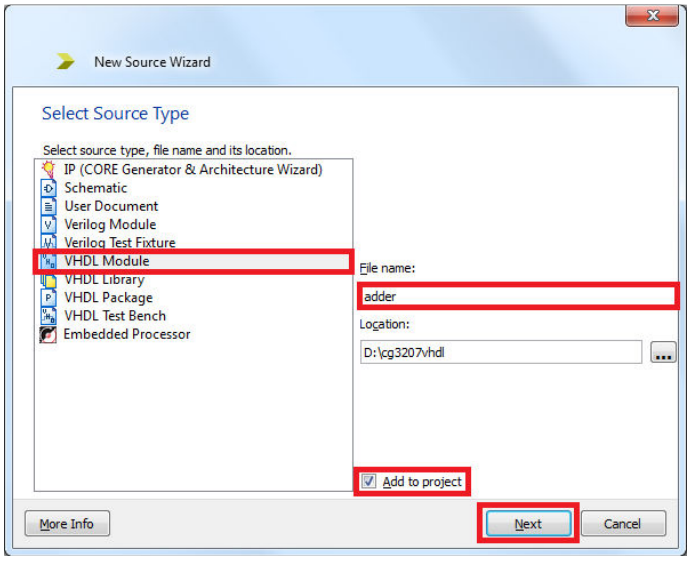
5. The ***ISE Workspace*** window should appear after the new project is created. It is the central place from where all aspects of the design/simulation process are controlled. The ***ISE Workspace*** main window is divided into four sub-windows:

(i) (Upper left) The ***Design*** window displays the files included in the project.
(ii) (Middle left) The ***Processes for Current Source*** window displays available processes.
(iii) (Bottom) The ***Console*** window displays status messages, errors, and warnings and is updated during all project actions.
(iv) (Grey window on the right) The ***VHDL Editor*** window enables editing of source files and access to the VHDL language templates catalogue which provides basic sample codes. The **Language Templates** can be accessed by selecting ***Edit → Language Templates...***

2. Xilinx VHDL Editor: Programming and Syntax Checking



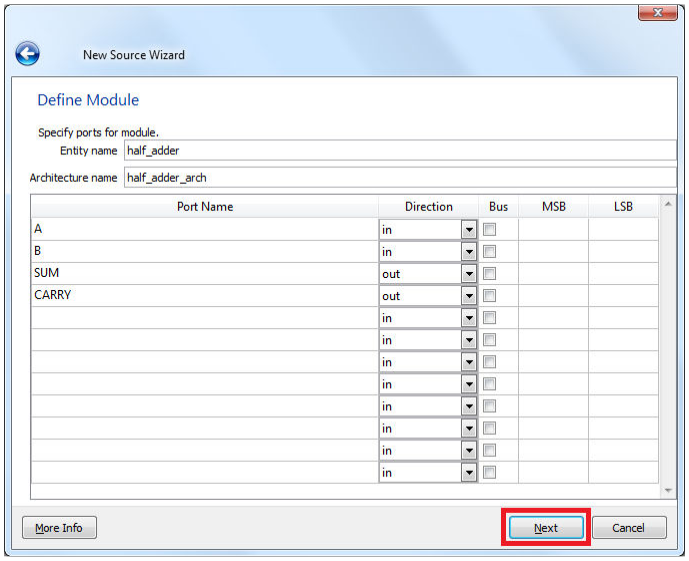
1. To create a new VHDL program, in ISE Project Navigator, select ***Project → New Source***.



2. A New Source Wizard window appears. In this ***Select Source Type*** window:

   (i) Select ***VHDL Module***.
   (ii) In the ***File name*** field, type the VHDL file name.
   Example: ***adder***
   (iii) Ensure ***Add to project*** is checked.

   Click ***Next*** to continue.



3. The ports are defined in this ***Define Module*** window

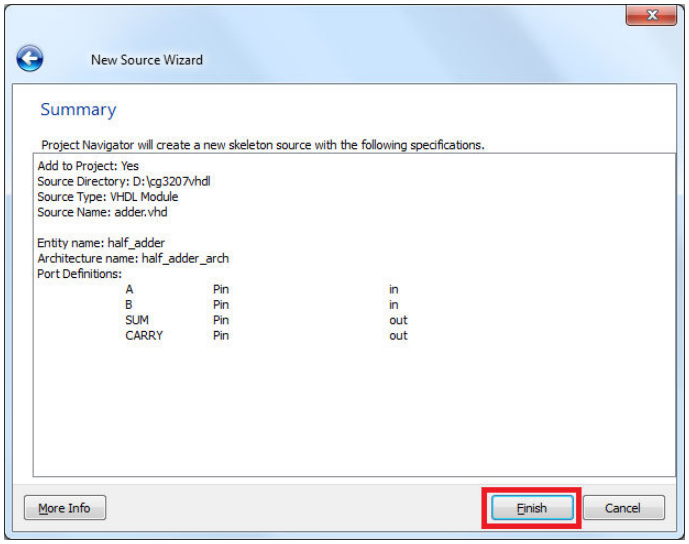
   (i) Enter an ***Entity name***.
   Example: ***half_adder***
   (ii) Enter an ***Architecture name***.
   Example: ***half_adder_arch***

   Click in the ***Port Name*** field to enter a value and select the corresponding direction in the ***Direction*** drop down menu. Enter the following values:

   (iv) Port Name: ***A***, Direction: ***in***.
   (v) Port Name: ***B***, Direction: ***in***.
   (vi) Port Name: ***SUM***, Direction: ***out***.
   (vii) Port Name: ***CARRY***, Direction: ***out***.

   The half adder has two 1-bit inputs and two 1-bit outputs, as defined above.

   Click on ***Next*** to proceed.



4. A description of the module summary is shown. Click ***Finish*** to complete the New Source Wizard.

```
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity half_adder is
33      Port ( A : in  STD_LOGIC;
34             B : in  STD_LOGIC;
35             SUM : out  STD_LOGIC;
36             CARRY : out  STD_LOGIC);
37  end half_adder;
38
39  architecture half_adder_arch of half_adder is
40
41  begin
42
43
44  end half_adder_arch;
```

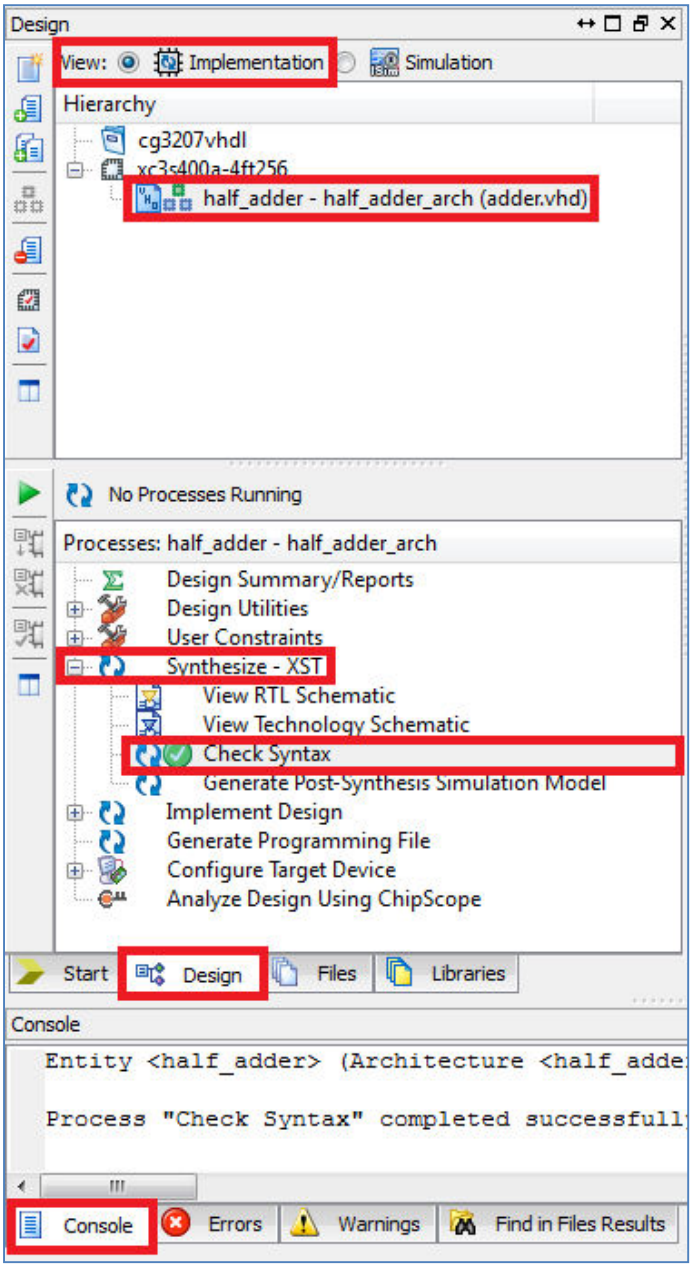
5. A VHDL file template is created in the ***VHDL Editor*** window.

```
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity half_adder is
33      Port ( A : in  STD_LOGIC;
34             B : in  STD_LOGIC;
35             SUM : out  STD_LOGIC;
36             CARRY : out  STD_LOGIC);
37  end half_adder;
38
39  architecture half_adder_arch of half_adder is
40
41  begin
42
43  SUM <= A xor B;
44  CARRY <= A and B;
45
46  end half_adder_arch;
```

6. Add the following lines within the architecture:

   SUM <= A xor B;
   CARRY <= A and B;

   The VHDL code is now complete and functional.



7. Save the project by selecting ***File → Save***.

Ensure the following:

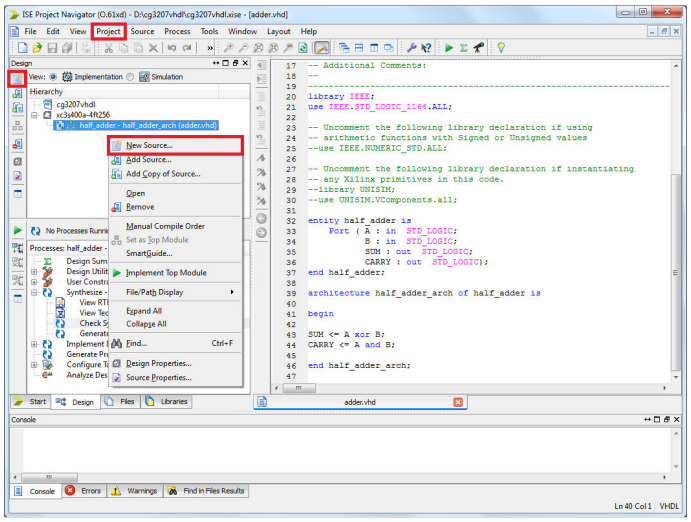
(i) The ***Design*** tab is selected.
(ii) ***View*** is set to ***Implementation***.
(iii) ***half_adder - half_adder_arch (adder.vhd)*** is selected.
(iv) ***Synthesize - XST*** is expanded.

Double click ***Check Syntax*** to verify the VHDL code syntax.

A tick in a green circle next to ***Check Syntax*** indicates that there are no syntax errors and that the VDHL program is ready for functional simulation.

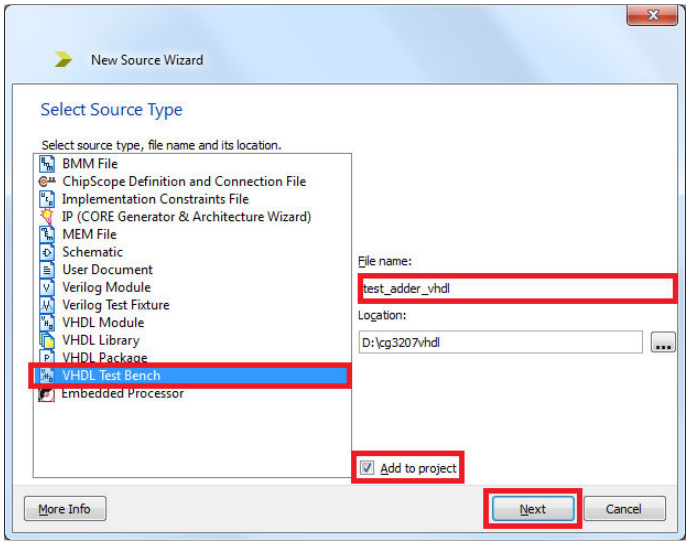
3. Behavioural Simulation: Using VHDL Test Bench for Combinational Circuits

Behavioural simulation is performed before actual design implementation on hardware to verify that the logic created is correct.



1. Add a ***New Source*** to the project. This can be done by:

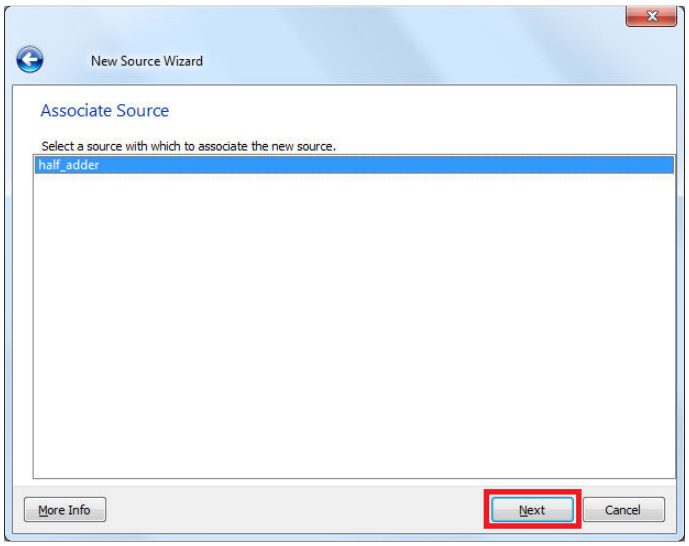
   (i) Right clicking in the design window and selecting ***New Source*** or
   (ii) Selecting ***Project → New Source*** or
   (iii) Clicking on the ***New Source*** shortcut icon.



2. A New Source Wizard window appears. In this ***Select Source Type*** window

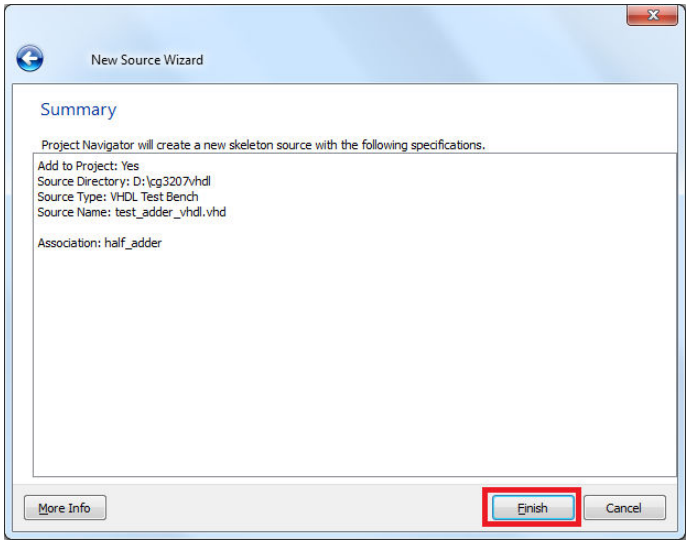
   (i) Select ***VHDL Test Bench***.
   (ii) In the ***File name*** field, type the file name.
   Example: ***test_adder_vhdl***
   (iii) Ensure ***Add to project*** is checked.

   Click ***Next*** to continue.

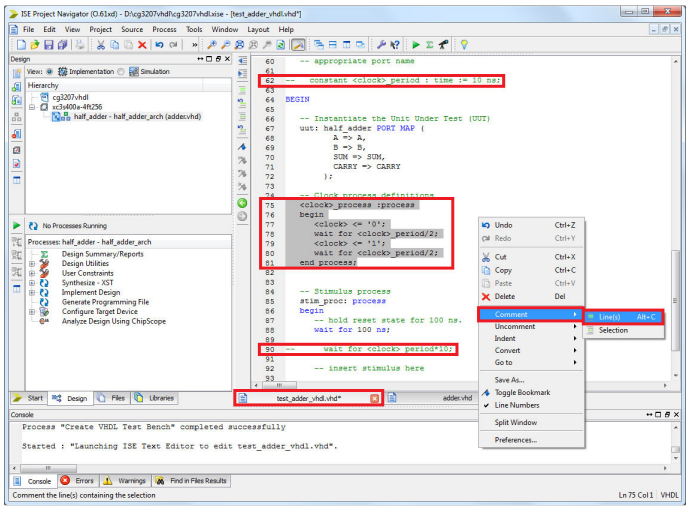


3. Make sure that it is associated with the source that needs to be tested. In this case, ***half_adder*** as defined in step 3 of section 2, is to be tested.

   Click ***Next*** to proceed.



4. Click ***Finish*** after reviewing the Summary window.



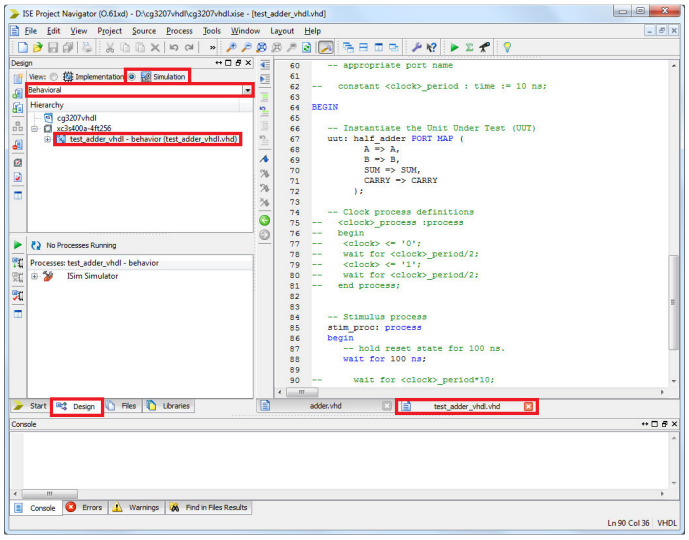
5. The test bench source opens up in a new tab. If a ***combinational circuit*** is being tested (like half_adder, which does not have a clock), comment out all lines which mention about the clock. Otherwise, syntax errors will occur.

   For this case, the lines to be commented are:

```
constant <clock>_period : time := 10 ns;

<clock>_process :process
begin
  <clock> <= '0';
  wait for <clock>_period/2;
  <clock> <= '1';
  wait for <clock>_period/2;
end process;

wait for <clock>_period*10;
```



6. This step is an additional note to the previous step. If the test bench source has been created but closed, perform the following to open the test bench source again:

   (i) Click on the ***Design*** tab.
   (ii) In View, select ***Simulation***.
   (iii) Double click on ***test_adder_vhdl - behaviour (test_adder_vhdl.vhd)***.
   (iv) Select the ***test_adder_vhdl.vhd*** tab.

```
84      -- Stimulus process
85      stim_proc: process
86      begin
87          -- hold reset state for 100 ns.
88          wait for 100 ns;
89
90  --        wait for <clock>_period*10;
91
92          -- insert stimulus here
93          A <= '0';
94          B <= '0';
95
96          wait for 100ns;
97          A <= '0';
98          B <= '1';
99
100         wait for 100ns;
101         A <= '1';
102         B <= '0';
103
104         wait for 100ns;
105         A <= '1';
106         B <= '1';
107
108         wait for 100ns;
109         A <= '0';
110         B <= '0';
111
112         wait;
113     end process;
114
115 END;
```

7. The Stimulus process indicates the input to the design. The process should begin with a ***wait for 100ns***, end with a ***wait***, and having appropriate stimuli in between.

   As shown in the figure, add the following stimuli to the VHDL test bench code:

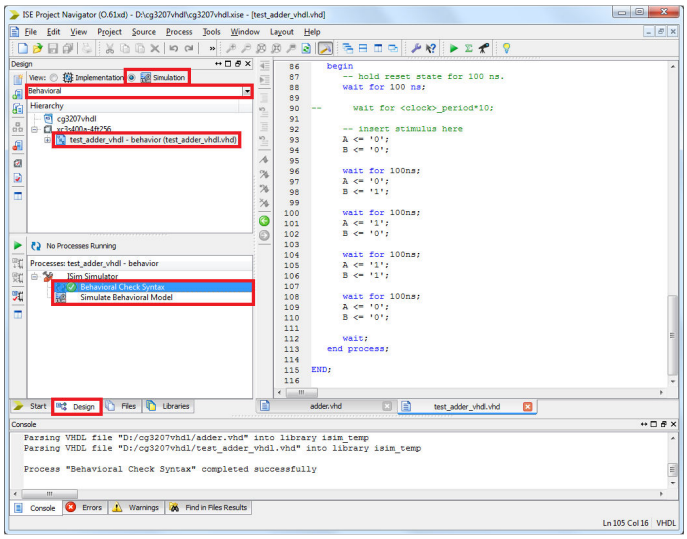
   A <= '0';
   B <= '0';

   wait for 100ns;
   A <= '0';
   B <= '1';

   wait for 100ns;
   A <= '1';
   B <= '0';

   wait for 100ns;
   A <= '1';
   B <= '1';

   wait for 100ns;
   A <= '0';
   B <= '0';



8. Save the project by selecting ***File → Save***.

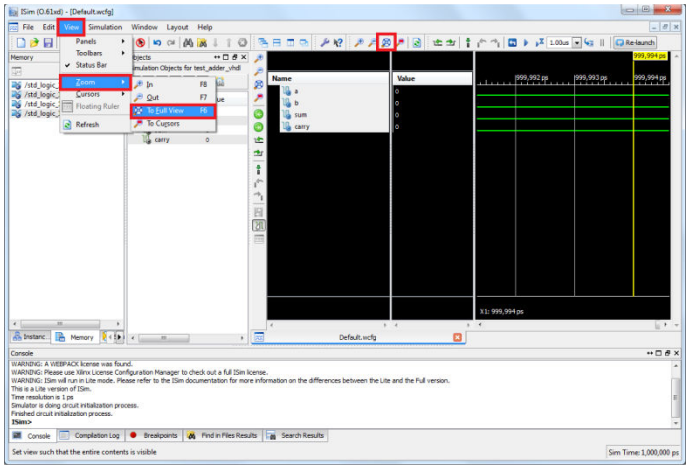
   Ensure the following:

   (i) The ***Design*** tab is selected.
   (ii) ***View*** is set to ***Simulation***.
   (iii) ***test_adder_vhdl - behaviour (test_adder_vhdl.vhd)*** is selected.
   (iv) ***ISim Simulator*** is expanded.

   Double click ***Behavioral Check Syntax*** to verify the VHDL code syntax.

   A tick in a green circle next to ***Check Syntax*** indicates that there are no syntax errors and that the VDHL program is ready for functional simulation.

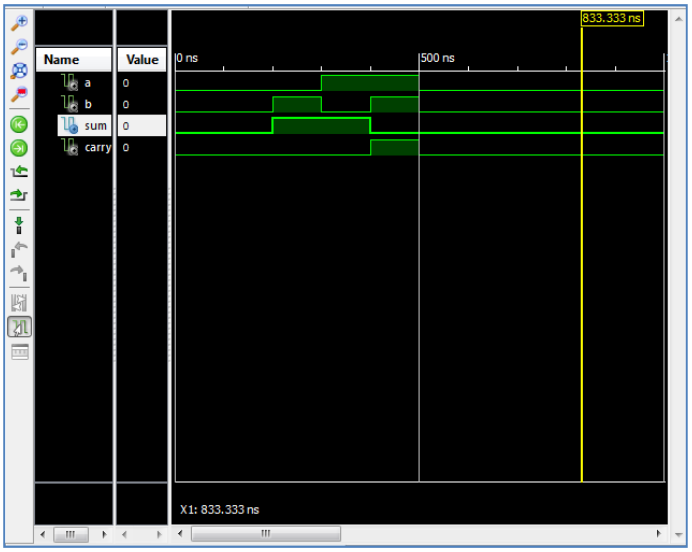
   Double click S***imulate Behavioral Model*** if the Behavioral Check Syntax is correct, else verify the VHDL codes if not.



9. The simulation window will open up. The time resolution might be too high. In this case, the resolution is in ***ps***.

   To adjust the view ***To Full View***, do one of the following:
   (i) Select ***View → Zoom → To Full View*** or
   (ii) Click on the ***Zoom to Full View*** icon or
   (iii) Press the keyboard shortcut button ***F6***.

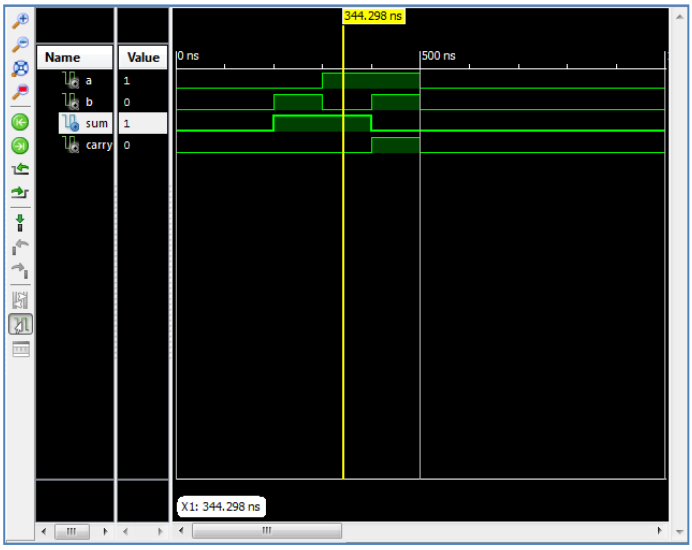
   In the event the ***Zoom to Full View*** option is not available, click anywhere in the black portion of the timescale window and try again.



10. In ***Full View***, the simulation window shows the full view of the simulation.

    a and ***b*** are the stimuli value, in other words, the input.
    sum and ***carry*** are the results of the stimuli, in other words, the output.

    All these four names and their behaviours were previously defined in steps 3 to 6 of section 2.



11. The yellow vertical line can be clicked on and moved around to get the ***Value*** of the ***Names*** at a particular time instant.

    Here, the window shows that at time instant 344.298 ns, the values of the ***Names*** are:

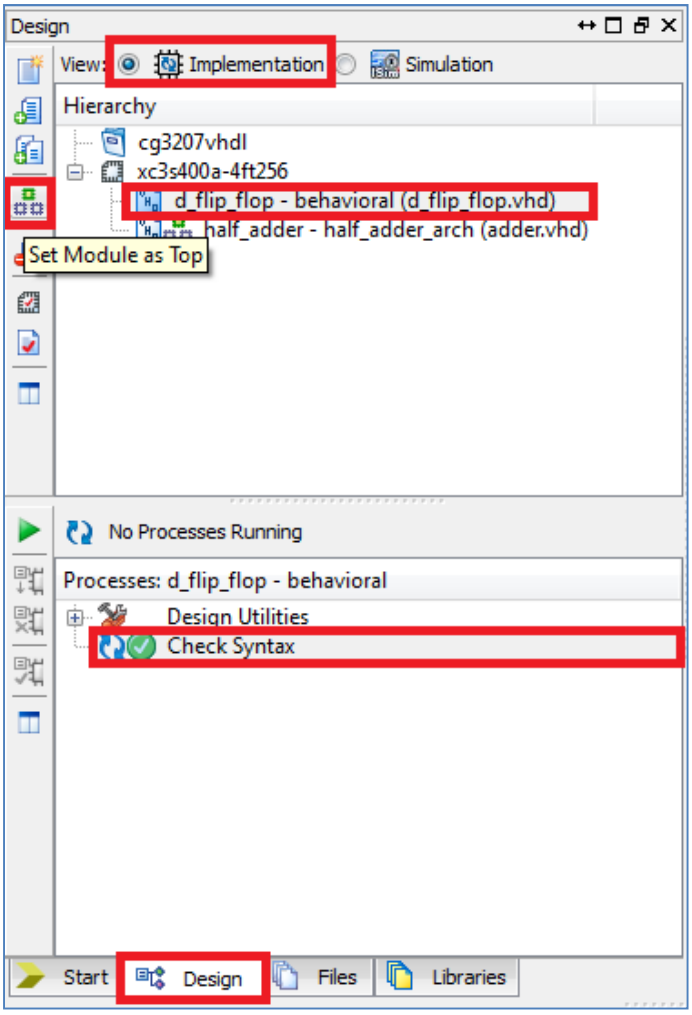
    a = 1
    b = 0
    sum = 1
    carry = 0

    Exit the application, or save if required. If the results are not as expected, modify the stimuli process and perform the simulation again.

4. Behavioural Simulation: Using VHDL Test Bench for Clocked Circuits

This section will show how to write a VHDL Test Bench for a clocked circuit. It involves a positive edge triggered D-flip-flop.

```
32  entity d_flip_flop is
33      Port ( clk : in  STD_LOGIC;
34             D : in  STD_LOGIC;
35             Q : out  STD_LOGIC);
36  end d_flip_flop;
37
38  architecture behavioral of d_flip_flop is
39
40  begin
41     process(clk)
42     begin
43        if clk'event and clk = '1' then
44           Q <= D;
45        end if;
46     end process;
47  end behavioral;
```



1. Similar to step 1 to 6 of section 2, create the VHDL code for a D-flip-flop. The VHDL codes will look like:

```
entity d_flip_flop is
   Port ( clk : in  STD_LOGIC;
          D : in  STD_LOGIC;
          Q : out  STD_LOGIC);
end d_flip_flop;

architecture behavioral of d_flip_flop is

begin
  process(clk)
  begin
    if clk'event and clk = '1' then
      Q <= D;
    end if;
  end process;
end behavioral;
```

2. Do not forget to save and then ***Check for Syntax*** while the ***d_flip_flop - behavioral (d_flip_flop.vhd)*** is selected.

   Notice that the ***Processes for Current Source*** window is different from that of step 7 of section 2. This is still ok because syntax can still be checked.

   If there is a need to implement the d_flip_flop design on an FPGA board, a ***Processes for Current Source*** window similar to that of step 7 in section 2 will be required. To have such a window, select the ***d_flip_flop - behavioral (d_flip_flop.vhd)*** and click on the ***Set Module as Top*** icon. Accept the reset when prompted. This ***Set Module as Top*** is not required if FPGA implementation is not needed.

```
58    -- Clock period definitions
59    constant clk_period : time := 10 ns;
60
61  BEGIN
62
63    -- Instantiate the Unit Under Test (UUT)
64    uut: d_flip_flop PORT MAP (
65           clk => clk,
66           D => D,
67           Q => Q
68         );
69
70    -- Clock process definitions
71    clk_process :process
72    begin
73       clk <= '0';
74       wait for clk_period/2;
75       clk <= '1';
76       wait for clk_period/2;
77    end process;
78
79
80    -- Stimulus process
81    stim_proc: process
82    begin
83       -- hold reset state for 100 ns.
84       wait for 100 ns;
85
86       wait for clk_period*10;
87
88       -- insert stimulus here
89
90       wait;
91    end process;
92
93  END;
```

3. After the syntax has been determined to be correct, create a VHDL Test Bench for this d_flip_flop. The steps are similar to steps 1 to 4 of section 3.

   Xilinx ISE WebPACK 13.2 usually does a good job of identifying the clock and creating a good skeleton code for the test bench accordingly. The VHDL codes used in the VHDL module help Xilinx ISE WebPACK 13.2 in determining the clock. For example:

   clk : in STD_LOGIC

   if clk'event and clk = '1' then

   The above codes indicated that ***clk*** is most likely the clock signal.

   In this figure, Xilinx ISE WebPACK 13.2 has properly identified ***clk*** as being the clock, and the correct skeleton code for the test bench has been created.

```
58    -- Clock period definitions
59    constant <clock>_period : time := 10 ns;
60
61  BEGIN
62
63    -- Instantiate the Unit Under Test (UUT)
64    uut: d_flip_flop PORT MAP (
65           clk => clk,
66           D => D,
67           Q => Q
68         );
69
70    -- Clock process definitions
71    <clock> process :process
72    begin
73       <clock> <= '0';
74       wait for <clock>_period/2;
75       <clock> <= '1';
76       wait for <clock>_period/2;
77    end process;
78
79
80    -- Stimulus process
81    stim_proc: process
82    begin
83       -- hold reset state for 100 ns.
84       wait for 100 ns;
85
86       wait for <clock>_period*10;
87
88       -- insert stimulus here
89
90       wait;
91    end process;
92
93  END;
```

4. If clock is not identified properly, this figure might appear. Replace <clock> with the name of the port being used for clock. For example, "***clk***".

   It is recommended that the VHDL codes of the VHDL module be revised logically when Xilinx ISE WebPACK 13.2 is not able to determine the clock automatically. VHDL codes with correct clock syntax should enable Xilinx ISE WebPACK 13.2 to automatically determine the clock signal, as indicated in step 3 of this section.

```
26
27  use IEEE.STD_LOGIC_ARITH.ALL
28  use IEEE.STD_LOGIC_UNSIGNED.ALL;
29
```

5. If mathematical operations are being used for giving stimuli in an automated fashion, the following two libraries might have to be included as well:

use IEEE.STD_LOGIC_ARITH.ALL
use IEEE.STD_LOGIC_UNSIGNED.ALL;

In the example described in this section, the stimuli given does not require the use of the above two libraries.

```
57
58      -- Clock period definitions
59      constant clk_period : time := 100 ns;
60
```

6. Continuing from step 3 of this section, change the ***clk_period*** to ***100 ns*** instead of the default ***10 ns***

```
80      -- Stimulus process
81      stim_proc: process
82      begin
83         -- hold reset state for 100 ns.
84         wait for 100 ns;
85
86         -- wait for clk_period*10;
87
88         -- insert stimulus here
89         wait for 25 ns;
90         -- use 75ns here for a negative
91         -- edge triggered circuit
92         D <= '1';
93
94         wait for 100 ns;
95         D <= '0';
96
97         wait for 100 ns;
98         D <= '0';
99
100        wait for 100 ns;
101        D <= '1';
102
103        wait for 100 ns;
104        D <= '1';
105
106        wait;
107     end process;
108
109 END;
```

7. Comment out the following line:

wait for clk_period*10;

Insert the following stimuli in the VHDL test bench:

wait for 25 ns;
D <= '1';

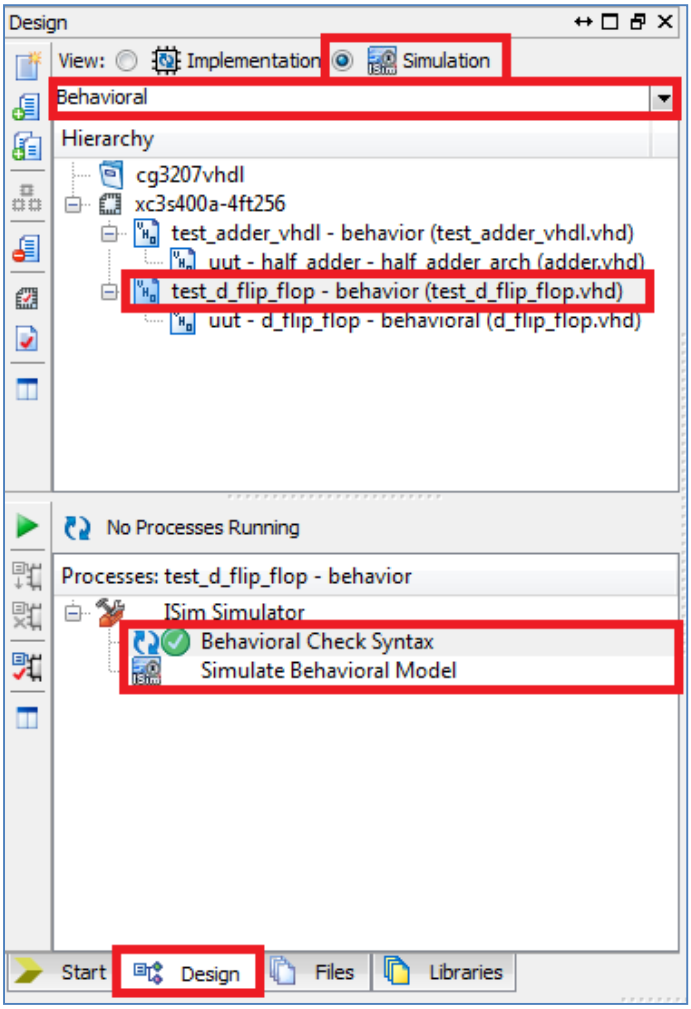
wait for 100 ns;
D <= '0';

wait for 100 ns;
D <= '0';

wait for 100 ns;
D <= '1';

wait for 100 ns;
D <= '1';

Note that the ***wait for 25 ns*** should preferably be ***wait for 75 ns*** if negative edge triggered circuits are used. If there is more than one input, these inputs can be changed simultaneously in between the stimuli. In this example, D is the only input that is being changed in between stimuli.



8. Save the project by selecting ***File → Save***

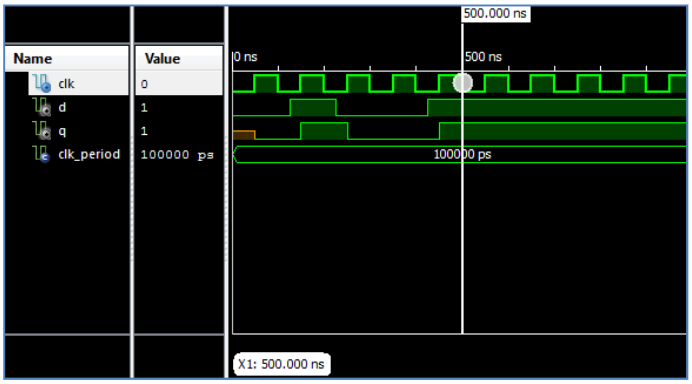
Ensure the following:

(i) The ***Design*** tab is selected.
(ii) View is set to ***Simulation***.
(iii) ***test_d_flip_flop - behaviour (test_d_flip_flop.vhd)*** is selected.
(iv) ***ISim Simulator*** is expanded.

Double click ***Behavioral Check Syntax*** to verify the VHDL code syntax.

A tick in a green circle next to ***Check Syntax*** indicates that there are no syntax errors and that the VDHL program is ready for functional simulation.

Double click S***imulate Behavioral Model*** if the Behavioral Check Syntax is correct, else verify the VHDL codes if not.

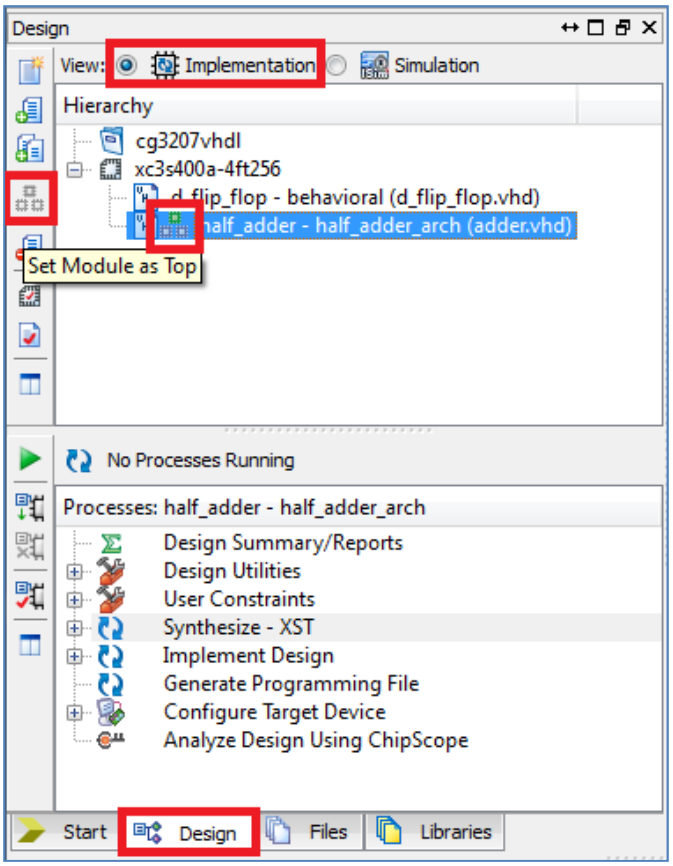


9. Similar to step 9 to 11 of section 3, the simulation results will be presented.

In this figure, the D-flip-flop is triggered at every positive clock cycle.

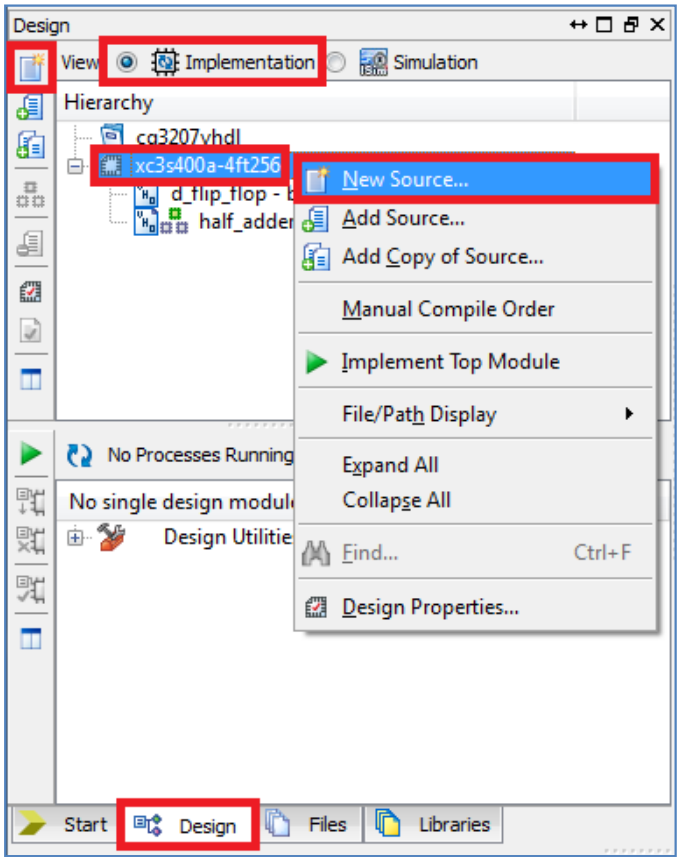
5. FPGA Implementation: 1-Bit Half-Adder

This section describes how to implement the design made in section 1 to 3 on an actual FPGA hardware.

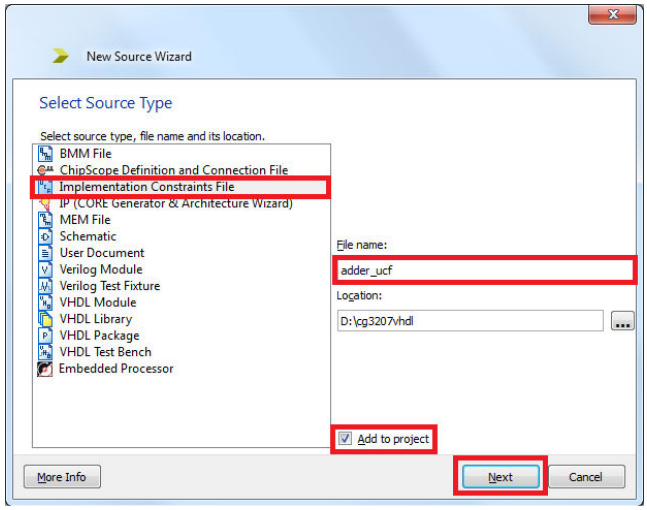


1. Set the ***half_adder - half_adder_arch (adder.vhd)*** as ***Top Module*** if not already a ***Top Module***. To this by selecting ***half_adder - half_adder_arch (adder.vhd)*** and clicking on the ***Set Module as Top*** icon

   If the file is already a ***Top Module***, an icon will appear to the left of the file name as shown in the figure, and the ***Set Module as Top*** icon will be disabled



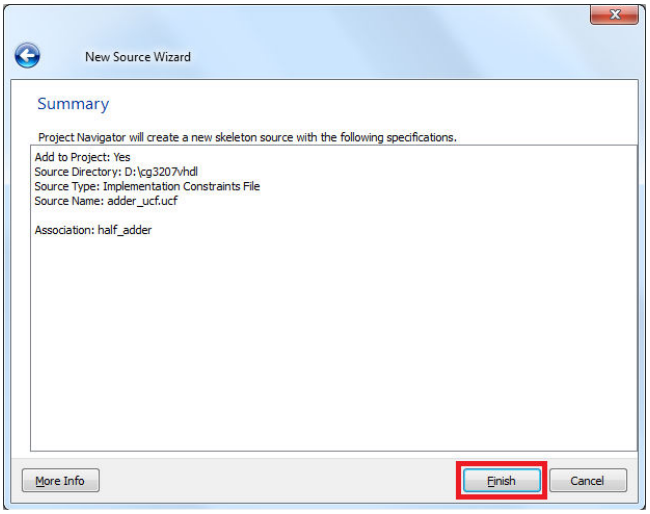
2. Add a ***New Source*** to the project.



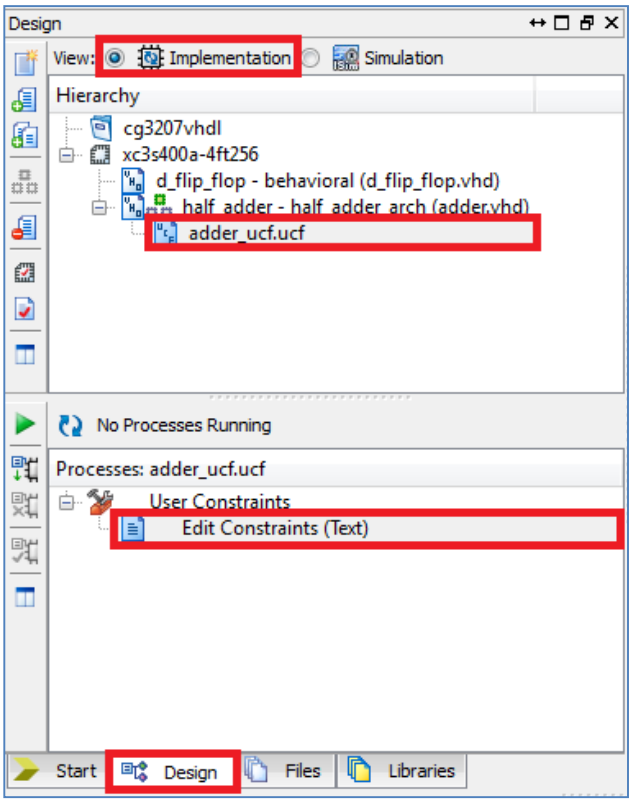
3. Select ***Implementation Constraints File*** in the ***Select Source Type*** window.

   Enter an appropriate ***File Name***.
   Example: ***adder_ucf***

   Check ***Add to Project*** and click on ***Next*** to continue.

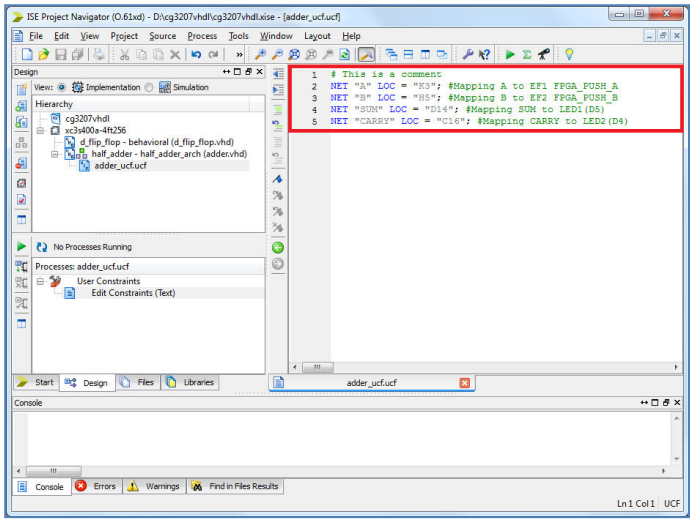


4. Click on ***Finish*** to close the New Source Wizard window.



5. The ***.ucf*** file needs to be edited. To do this,

   (i) Ensure the ***Design*** tab is selected.
   (ii) ***View*** is set to ***Implementation***.
   (iii) In the Design window, expand ***half_adder - half-adder_arch (adder.vhd)***.
   (iv) Click once on ***adder_ucf.ucf***.
   (v) In the Processes for Current Source window, expand ***User Constraints***.
   (vi) Double click on ***Edit Constraints (Text)***.



6. In the text file opening up in the ***Editor*** window, specify the mapping of inputs and outputs to FPGA pins.

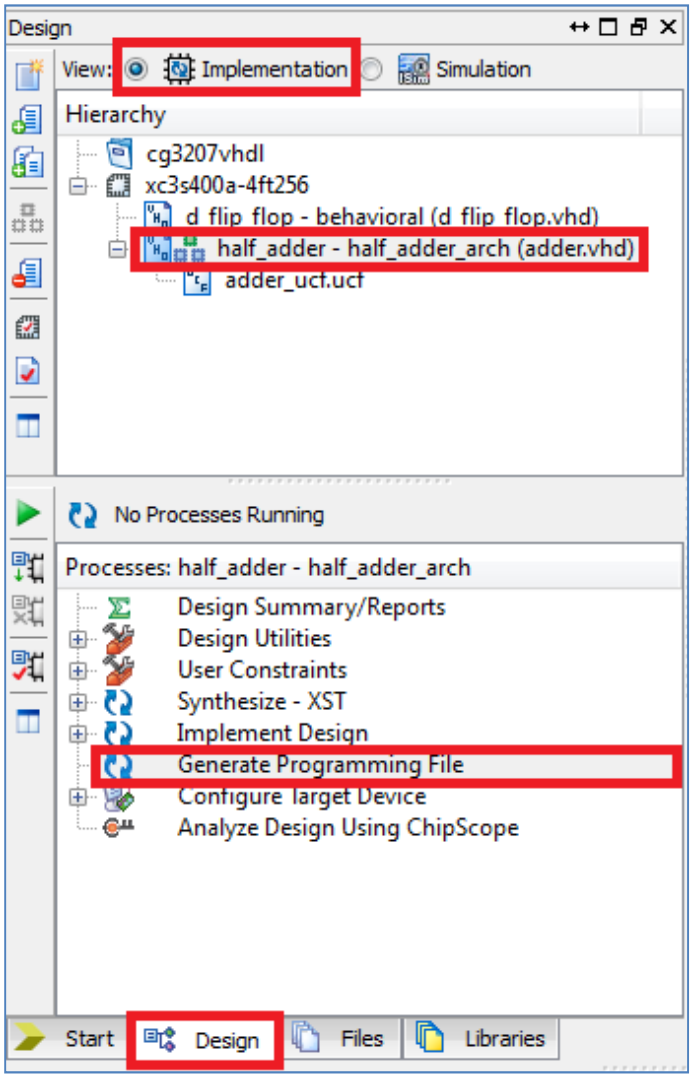
   Create the UCF file mapping input/output signals (in your code) with pins. Text following '#' is a comment.

   Type:

   NET "A" LOC = "K3"; #Mapping A to EF1 FPGA_PUSH_A
   NET "B" LOC = "H5"; #Mapping B to EF2 FPGA_PUSH_B
   NET "SUM" LOC = "D14"; #Mapping SUM to LED1(D5)
   NET "CARRY" LOC = "C16"; #Mapping CARRY to LED2(D4)

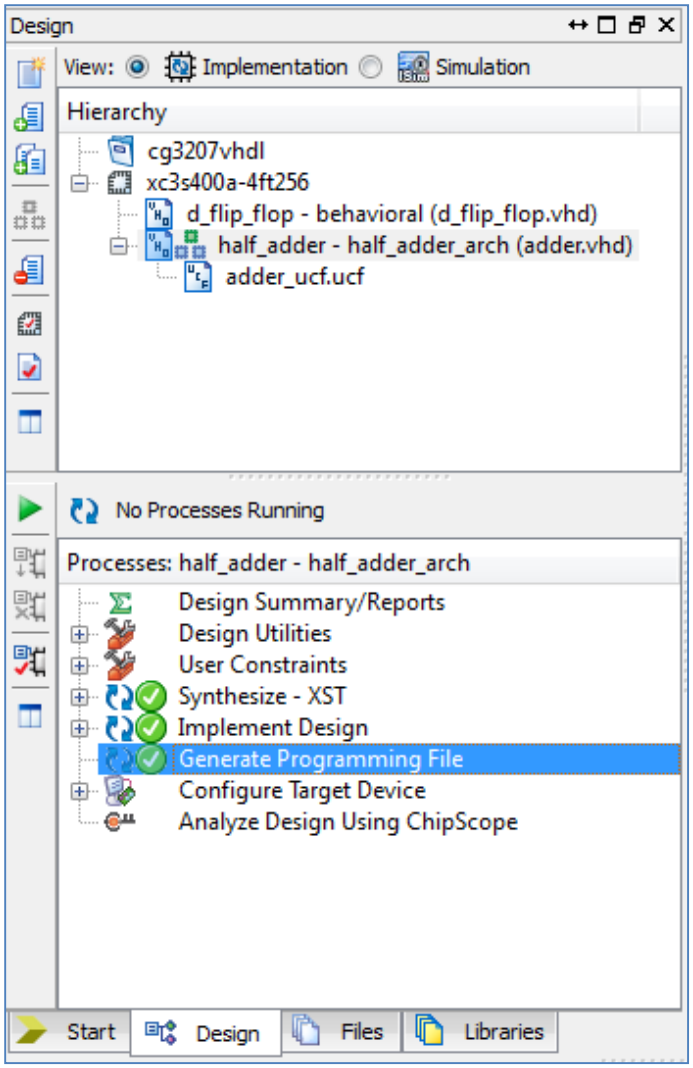
   Save the .ucf file by clicking on ***File → Save***.

   ** Refer to the end of the manual for additional mappings.*

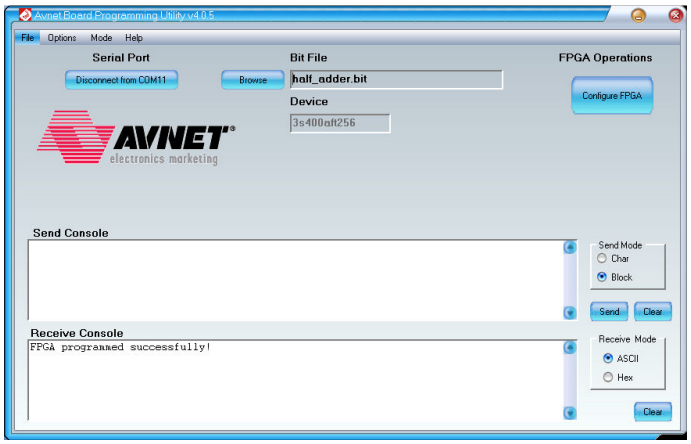


7. In the ***Design*** window, highlight the VHDL source ***half_adder - half_adder_arch (adder.vhd)***, and double click on ***Generate Programming File*** in the ***Process for Current Source*** window.

   This process will take some time.



8. Once the synthesis is complete and syntax is correct, tick marks in green circles will be shown to the left of ***Synthesize - XST***, ***Implement Design*** and ***Generate Programming File***.



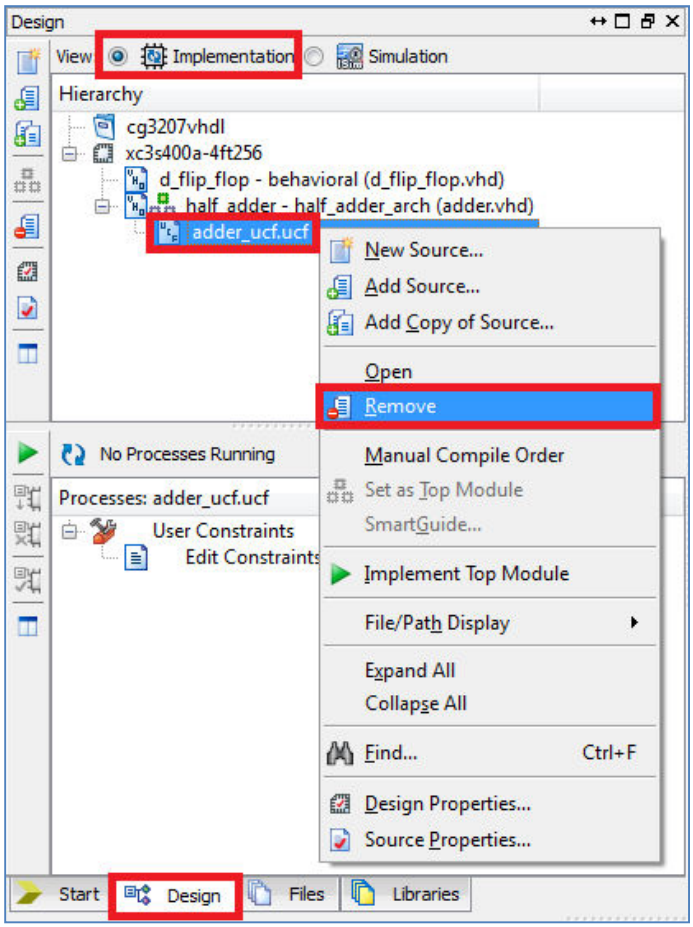
9. Open the ***Avnet Board Programming Utility v4.0.5***. Refer to the ***AvProg 4.05 Installation*** manual for instructions. Load the bit file, which is found in the folder where the Xilinx project is saved. The name of the bit file will be the same as the name of the VHDL entity, in this case being ***half_adder***.

   For this manual, the location of the bit file was:
   D:\cg3207vhdl

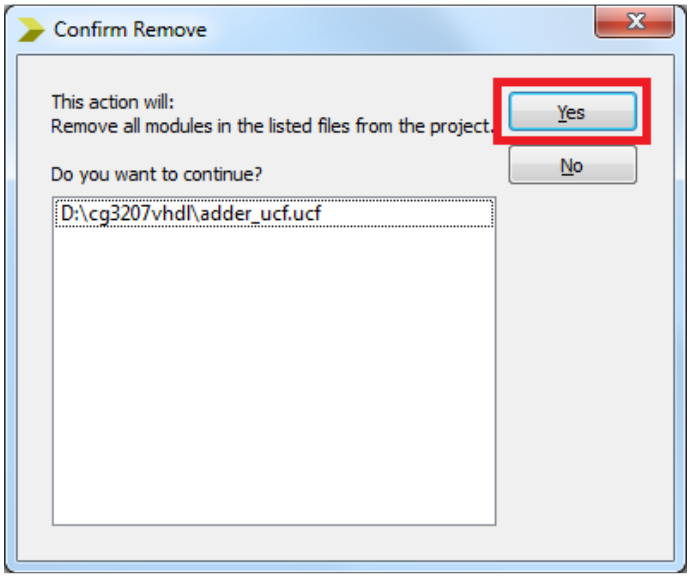
   Program the FPGA board. The FPGA board now performs as determined by the VHDL codes and the user constraints file from step 6 of this section. When the capacitive pads ***PUSH_A EF1*** and ***PUSH_B EF2*** are pressed, the appropriate LEDs ***D4*** and ***D5*** light up. Compare step 10 of section 3 with the results from the FPGA board. They are the same!

6. FPGA Implementation: Using PlanAhead 13.2

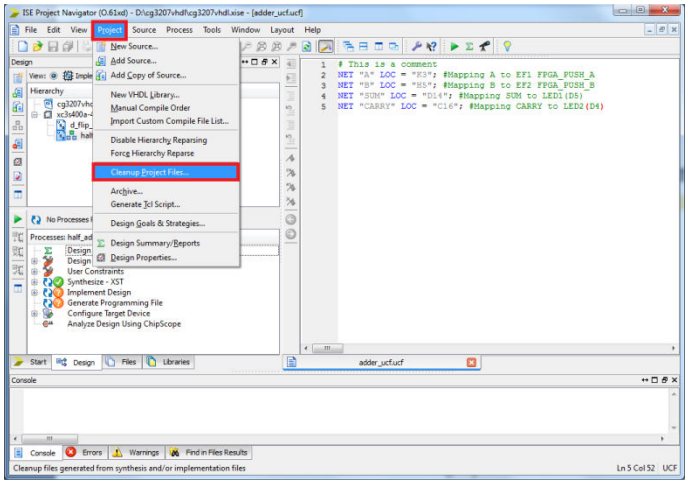
This section describes another method of FPGA pin mapping using PlanAhead 13.2



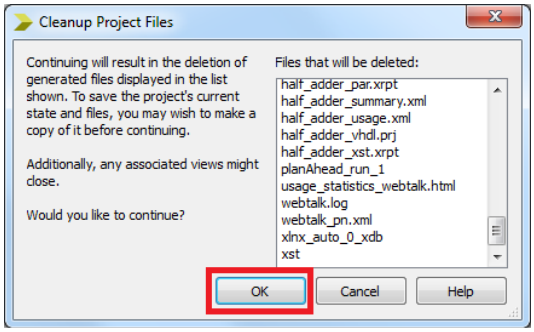
1. Right click on the user constraints file that was previously created (***adder_ucf.ucf***) and ***Remove*** it.



2. Click on ***Yes***.

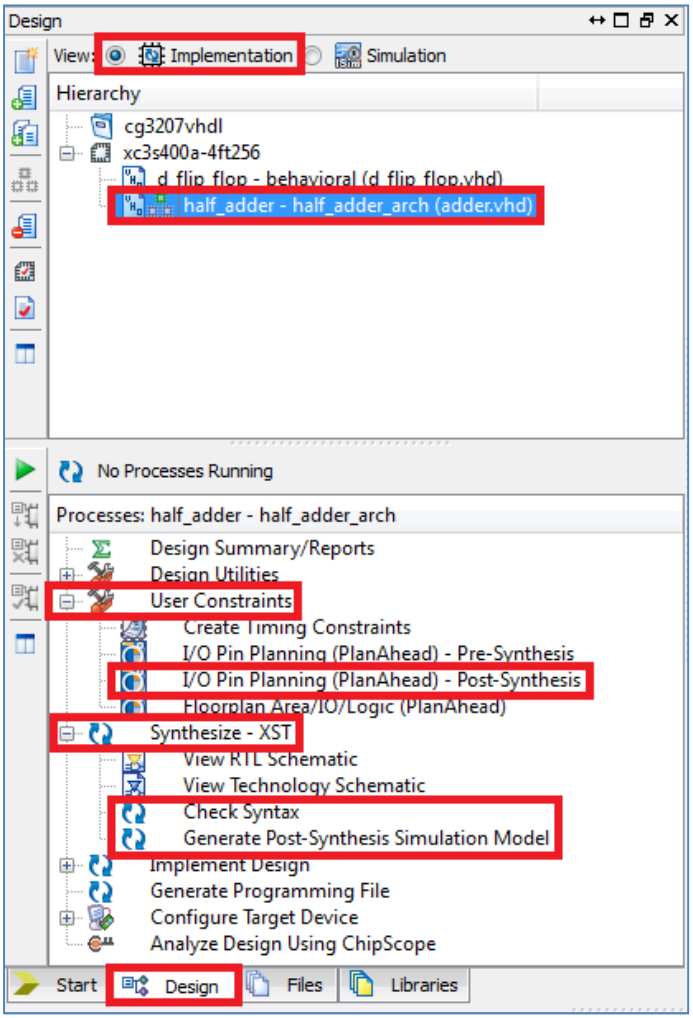


3. Click on ***Project → Cleanup Project Files***.



4. Confirm by clicking on ***OK***.

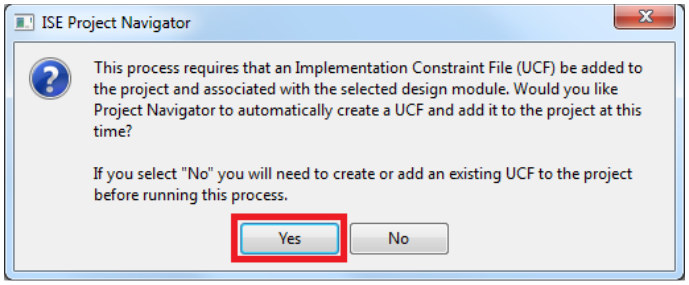
   After the cleanup, the ***adder_ucf.ucf*** tab in the ***VHDL Editor*** window can be closed. Save the project if required by clicking on ***File →Save***.



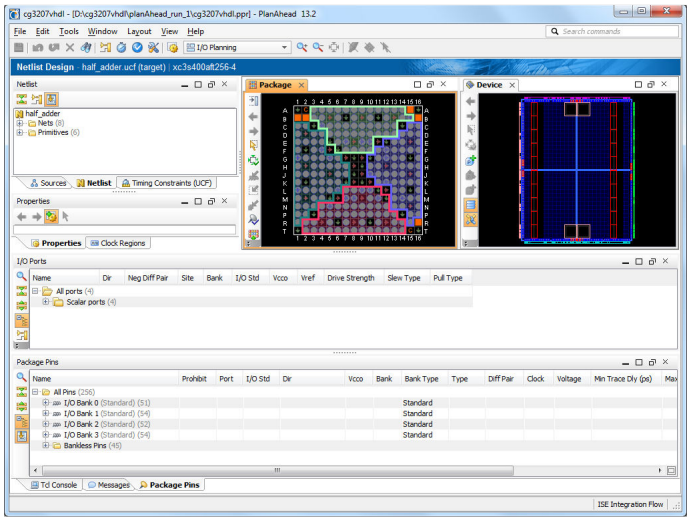
5. Notice that all previous summary, ticks, question marks, exclamation marks will disappear after cleanup, and your Xilinx project folder will become smaller in size.

   Perform the following in order:

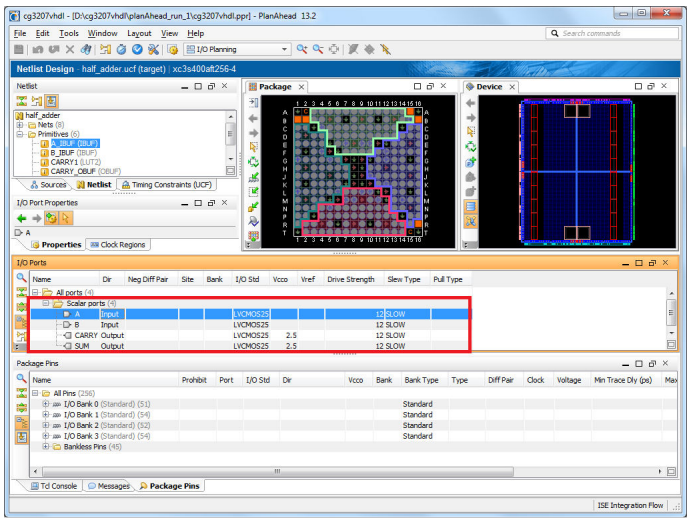
   (i) Select ***half_adder - half_adder_arch (adder.vhd)***.
   (ii) Expand ***User Constraints***.
   (iii) Expand ***Synthesize - XST***.
   (iv) Double click on ***Check Syntax***.
   (v) After Check Syntax succeeds, double click on ***Generate Post-Synthesis Simulation Model***. This process will take some time.
   (vi) After ***Post-Synthesis Simulation Model*** succeeds, double click on ***I/O Pin Planning (PlanAhead) - Post-Synthesis***



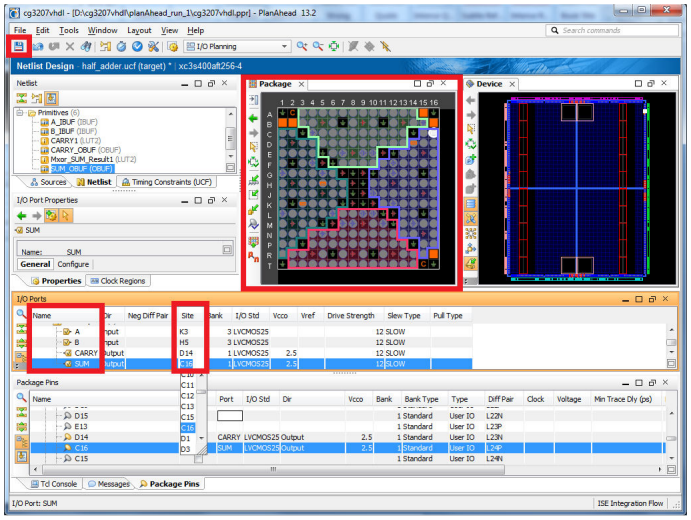
6. Click on Yes. PlanAhead 13.2 will launch and the loading will take some time.



7. The PlanAhead 13.2 program window will appear after some time.



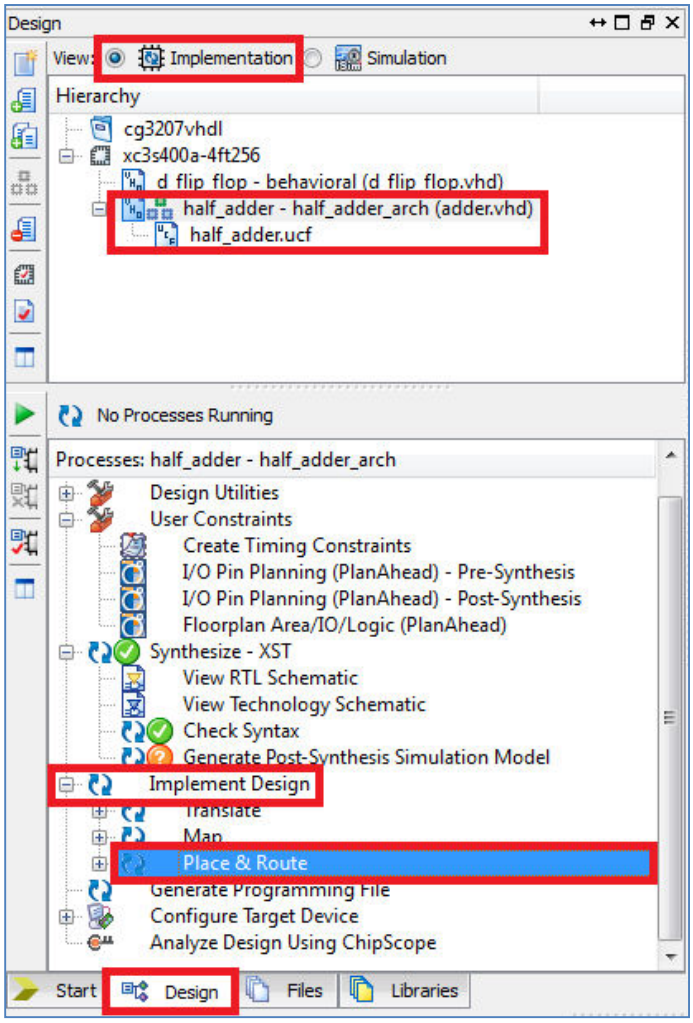
8. Expand ***Scalar ports (4)***.



9. The FPGA pin mapping can be done in several ways, including:
   (i) Click on ***Site*** and enter the mapping value.
   (ii) Click on ***Site*** and choose a value from the dropdown menu.
   (iii) Click on one of the ***Name*** and drag it onto the ***Package*** window at the correct tile location.

   ** Refer to the end of the manual for additional mappings.*

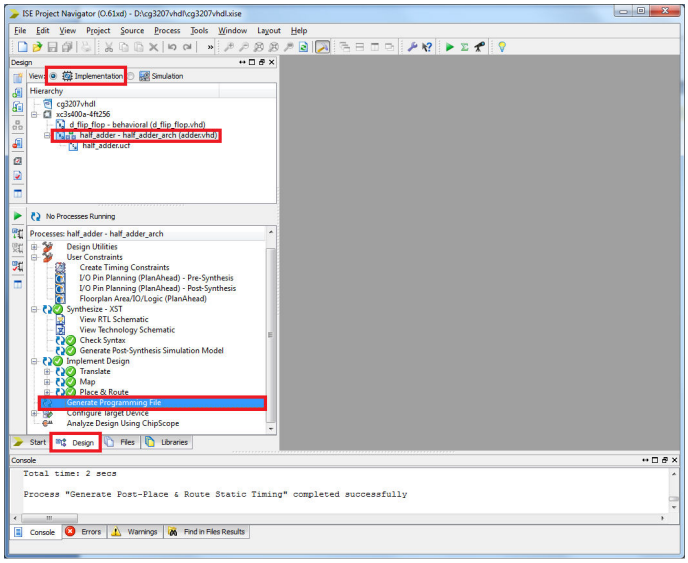
   Click on the ***Save Design*** icon and close this PlanAhead 13.2 window.



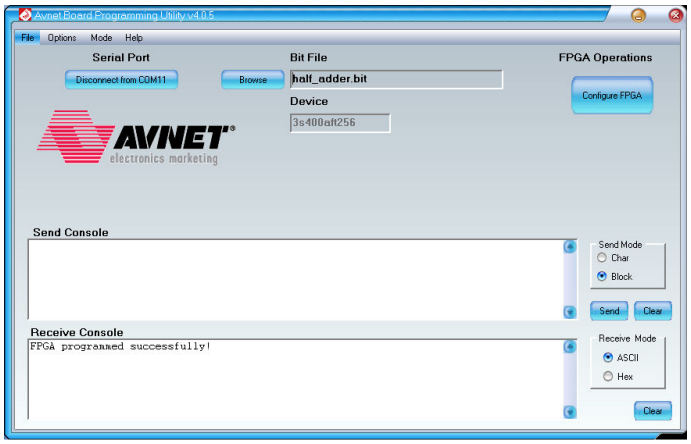
10. Back in Xilinx ISE WebPACK, the ***half_adder.ucf*** file for the half_adder will have been created.

    (i) Click once on ***half_adder - half_adder_arch (adder.vhd)***.
    (ii) Expand ***Implement Design***.
    (iii) Double click on ***Place & Route***.

    Wait for some time until completion.



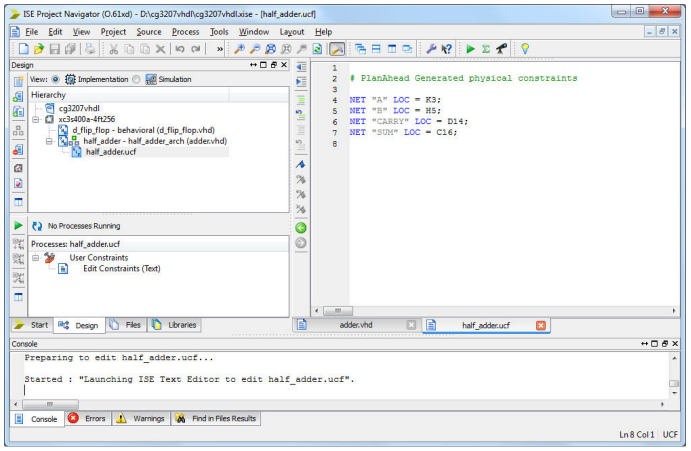
11. Success will be indicated by tick marks in green circles. Double click on ***Generate Programming File*** in the ***Process for Current Source*** window.



12. Open the ***Avnet Board Programming Utility v4.0.5***. Refer to the ***AvProg 4.05 Installation*** manual for instructions. Load the bit file, which is found in the folder where the Xilinx project is saved. The name of the bit file will be the same as the name of the VHDL entity, in this case being ***half_adder***.

    For this manual, the location of the bit file was:
    D:\cg3207vhdl

    Program the FPGA board. The FPGA board now performs as determined by the VHDL codes and the FPGA pin mappings from step 9 of this section. When the capacitive pads ***PUSH_A EF1*** and ***PUSH_B EF2*** are pressed, the appropriate LEDs ***D4*** and ***D5*** light up. Compare step 10 of section 3 with the results from the FPGA board. They are the same!

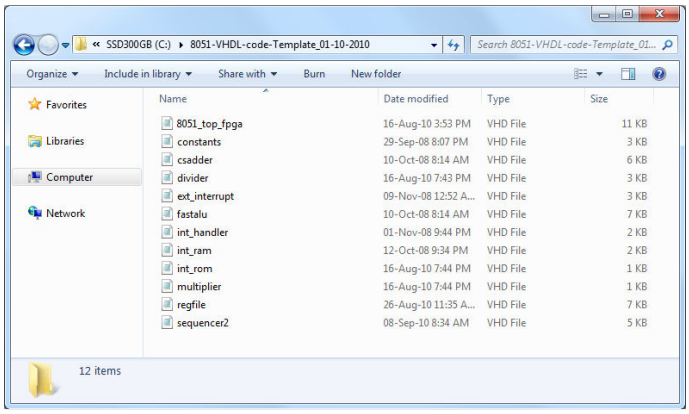


13. Comparing the User Constraints File (***.ucf*** file) with that from section 5, there is not much difference. Depending on the number of FPGA pins that needs to be mapped, or personal preference, you can choose either of the method of FPGA pin mappings.

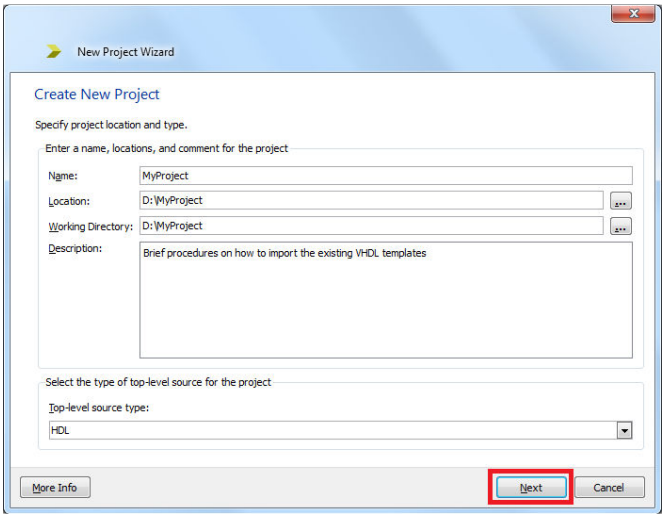
    Either way, pin mappings need to be referenced from the FPGA manual. In this manual, the most commonly used mappings are indicated at the end of this manual.

7. CG3207 Project: Importing the Existing Files

This section briefly describes how to import the existing VHDL templates to a Xilinx project.

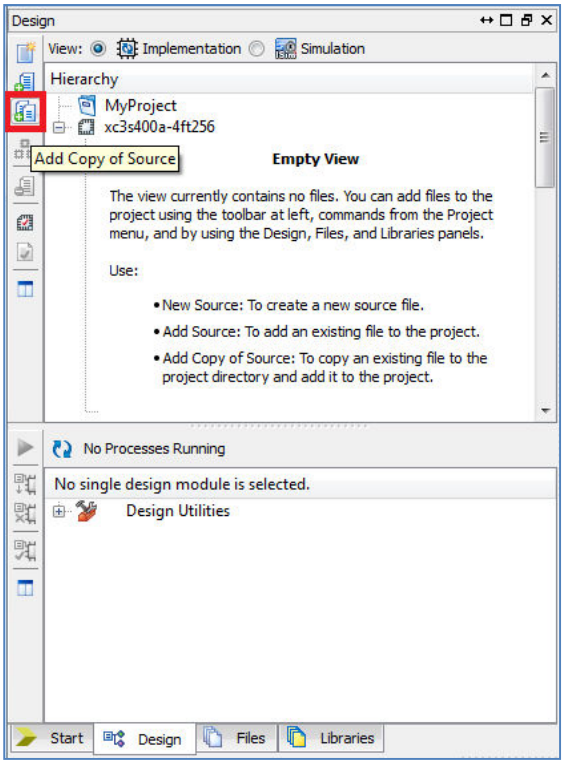


1. The files to import to Xilinx project.

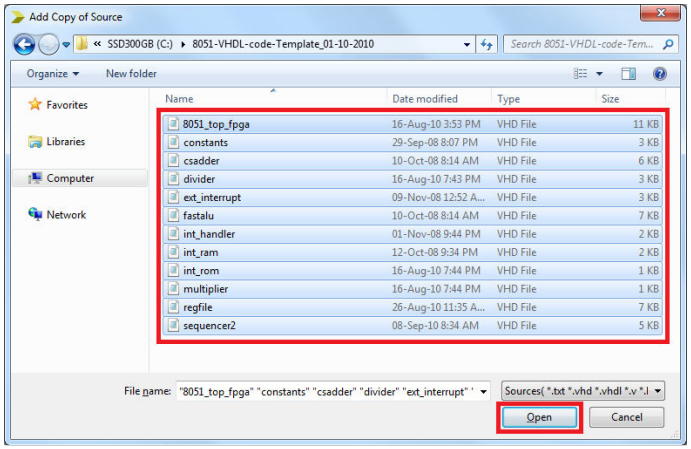


2. Create a new project (***File → New Project...***) and complete the wizard with settings as:

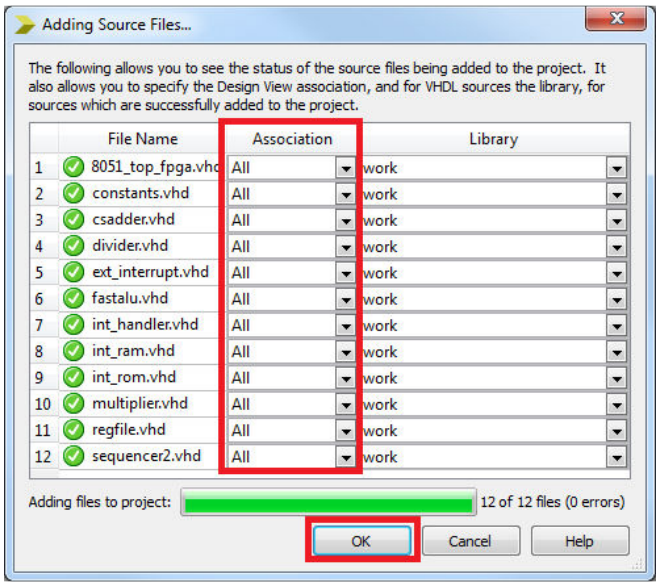
   (i) Evaluation Development Board: ***None Specified***
   (ii) Product Category: ***All***
   (iii) Family: ***Spartan3A and Spartan 3AN***
   (iv) Device: ***XC3S400A***
   (v) Package: ***FT256***
   (vi) Speed: ***-4***
   (vii) Synthesis Tool: ***XST (VHDL/Verilog)***
   (viii) Simulator: ***ISim (VHDL/Verilog)***
   (ix) Preferred Language: ***VHDL***
   (x) Property Specification in Project File: ***Store all values***
   (xi) Manual Compile Order: ***Unchecked***
   (xii) VHDL Source Analysis Standard: ***VHDL-93***
   (xiii) Enable Message Filtering: ***Unchecked***



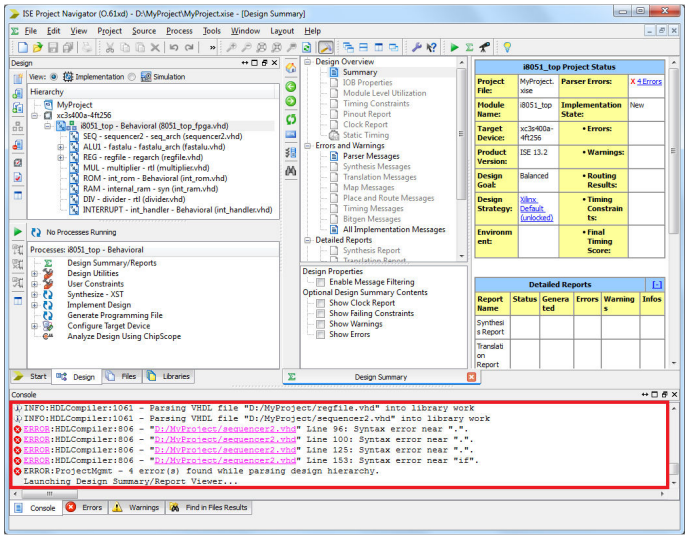
3. Click on ***Add Copy of Source***.



4. Select all the files and click on ***Open***.



5. Ensure ***Association*** is set to **All** for all the files and click on ***OK***.



6. After importing the files to the project, four syntax errors will be displayed in the ***Console*** box. This is normal - you will have to replace these error lines with several thousand lines of VHDL codes to emulate the sequencer of the 80C51 processor...

   The initial VHDL templates, as shown in step 1 of this section can be deleted as a copy has now been made to the Xilinx project folder (Example: ***D:\MyProject***).

   Do not forget to make regular backups of your Xilinx project folder. Additional VHDL tutorials are available at:

   http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html
   http://www.embeddedrelated.com/showarticle/31.php

   or by reviewing the EE2020 (Digital Fundamentals) lecture notes.

   Enjoy! :)

8. Miscellaneous: FPGA Pin Mappings and Additional Example

Xilinx Spartan-3A Evaluation Kit FPGA Board Pin Mappings

PSoC Cap Sense	FPGA "Pushbutton"	FPGA Pin#
EF1	FPGA_PUSH_A	K3
EF2	FPGA_PUSH_B	H5
EF3	FPGA_PUSH_C	L3
EF4	FPGA_RESET	H4

PSoC/FPGA Push Buttons

1. **Miscellaneous I/O**

Four user push button switches are provided via capacitive touch-pads connected to the Cypress PSoC device. A "touch" at any of these four pads is sensed by the PSoC and forwarded to the FPGA; these "push buttons" and their relationship to the FPGA are depicted in this figure.

Note that FPGA_RESET is a "soft" reset intended for FPGA code usage and does not perform any type of FPGA hardware reset.

Other user accessible FPGA pins are described in the FPGA datasheet (***Xilinx Spartan-3A Evaluation Kit User Guide***)

LEDs	FPGA Pin#
LED1 (D5)	D14
LED2 (D4)	C16
LED3 (D3)	C15
LED4 (D2)	B15

LED assignment

2. **LEDs**

Four LEDs are provided for signalling purposes and connected to the FPGA as shown in this figure. The corresponding FPGA pin must be driven high to light an LED.

Clocks	FPGA Pin#
16.0MHz	C10 (GCLK4)
12.0MHz	N9 (GCLK0)
32.0kHz	T7 (GCLK13)

Module Clocks

3. **Module Clocks**

Three clocks are provided to the FPGA; 16.0 MHz from a Maxim MAX7381 CMOS oscillator (U6), and 12.0 MHz and 32.0 kHz from the PSoC. This figure provides FPGA connection details.

Additional VHDL Example

To understand more about the Xilinx Spartan-3A Evaluation Kit FPGA Board clock, an additional VHDL example is provided:

(i) Download the ***Additional VHDL Example.zip*** from IVLE and unzip it.
(ii) Open the ***blink_led_vdhl*** folder.
(iii) Open the ***blink_led*** Xilinx ISE Project.
(iv) Analyse the VHDL code for the ***simple_count.vhd***.
(v) Either ***Generate Programming File***, or simply use the existing ***simple_count.bit*** file from the Additional VHDL Example folder.
(vi) Program the Xilinx Spartan-3A Evaluation Kit FPGA Board with the ***simple_count.bit*** file.

Notice that ***D2***,***D3***,***D4*** and ***D5*** are acting as counters on the Xilinx Spartan-3A Evaluation Kit FPGA Board. What the VHDL code is doing is that it is simply reading the last 4 MSB of a 26-bit string "**0000**00000000000000000000". The 4 LEDs represent these last 4 MSB of the string. The rest of the string is discarded because they change values too fast to be of use when perceived through the LEDs. The clock speed of the FPGA board is counted in the range of KHz or MHz. Such type of VHDL code can be used to emulate a slower clock that the human eye can perceive and make use of when represented through LEDs.

Also, note that if the ***PUSH_A EF1*** capacitive button is pressed on the Xilinx Spartan-3A Evaluation Kit FPGA Board, the counter is paused. This is indicated by the VHDL code and user constraints file:

```
41  process (clock)
42  begin
43     if clock='1' and clock'event then
44        if enable='0' then
45           count <= count + 1;
46        end if;
47     end if;
48  end process;
```

```
1  NET "clock" TNM_NET = clock;
2  TIMESPEC TS_clock = PERIOD "clock" 62.5 ns HIGH 50%;
3  NET "clock" LOC = C10;
4  NET "enable" LOC = K3;
5  NET "leds<0>" LOC = D14;
6  NET "leds<1>" LOC = C16;
7  NET "leds<2>" LOC = C15;
8  NET "leds<3>" LOC = B15;
```

The VHDL codes indicate that when the ***PUSH_A EF1*** (LOC = K3) capacitive button is pressed, ***enable*** = '1' and the ***count <= count + 1*** will not occur.

The manual ***s3aeval_fpga_intro_tutorial_v10.1.00.pdf***, which is included with the downloaded zip file, can be referred to for more details. However, take note that the included manual is based on an older version of Xilinx ISE WebPACK and some Xilinx functions have been removed or changed in the newer version Xilinx ISE WebPACK 13.2.