

## ***Digital System Design using VHDL and FPGAs***

### **Project 1**

### **Simple Combinational Circuit Design**

#### **VHDL**

VHDL is an acronym for the VHSIC Hardware Description Language. VHSIC is in turn an acronym for the Very High Speed Integrated Circuits program of the US Department of Defense [1].

VHDL was originally developed in 1981 by the US Department of Defense to describe the structure and function of hardware. Today VHDL is the industry standard language for describing digital circuits, largely because it is an official IEEE standard. The original standard for VHDL was adopted in 1987 and was called IEEE 1076. The IEEE has updated the standard several times since.

The language was first envisioned for documentation but was quickly adopted for simulation and synthesis [2]. Today VHDL is popular for use in design entry in CAD systems, such as Xilinx ISE Design Suite. Simulation and synthesis use only a subset of VHDL's features and this subset may vary among different CAD systems.

#### **Xilinx ISE Design Suite**

Xilinx ISE Design Suite is a CAD system that accepts VHDL descriptions as design entry. Using Xilinx ISE Design Suite tools, the digital circuit designer can :

- Synthesize the VHDL description. Synthesis tools translate a digital circuit design to gates – specific Xilinx device primitives, such as Slices, RAM blocks, DSP blocks and other device specific blocks.
- Simulate the behavior of a specific digital circuit (described in VHDL) before and after synthesis.
- Implement the result of synthesis on a specific Xilinx device. Implementation consists of design translation, map and place-and-route. The first step in implementation is the design translation. Translation merges the synthesis result (also called a netlist) and the design constraints (e.g. FPGA pins used by the specific design and the desired clock period of the design) into a single design file. The next step is the design map which fits the design into the available resources on the target device, and optionally, places the design. The last step in implementation is place and route which places and routes the design to the timing constraints [3].
- Simulate the behavior of the implemented design.
- Generate a programming (configuration) file for FPGA configuration.
- Download the configuration file to the FPGA.

## Simple Combinational Circuit Design

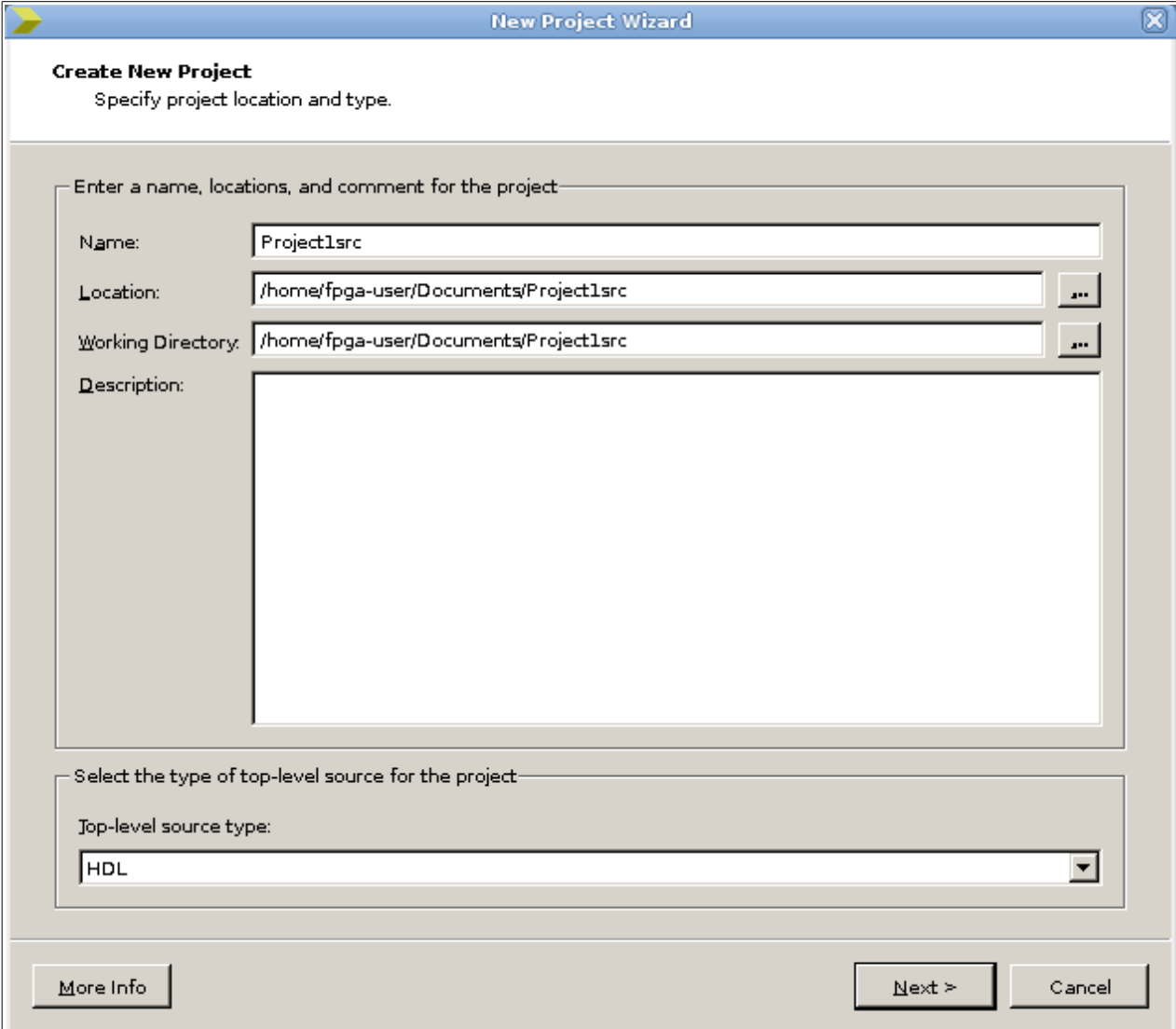
The following pages will demonstrate how to implement and verify a simple combinational circuit design using Xilinx ISE 13.3, VHDL as design entry and Xilinx Evaluation boards.

### 1. Open Xilinx ISE :

Double click the ISE desktop icon. Alternatively open a new terminal and give the command `./bin/ise` (without the quotes). For remote users, open a new terminal and give the following commands `export DISPLAY=:1` and `./remote/ise` (without the quotes).

### 2. Create a new Xilinx ISE Project :

In the ISE Project Navigator window select `File->New Project...` In the New Project Wizard window, type `Project1src` in the **Name** field and `/home/fpga-user/Documents/Project1src` in the **Location** field and click **Next** (see **Figure 1**).



The screenshot shows the 'New Project Wizard' dialog box in Xilinx ISE. The title bar is 'New Project Wizard'. The main heading is 'Create New Project' with the instruction 'Specify project location and type.' Below this, there are two sections. The first section is 'Enter a name, locations, and comment for the project' and contains four fields: 'Name' with the value 'Project1src', 'Location' with the value '/home/fpga-user/Documents/Project1src', 'Working Directory' with the value '/home/fpga-user/Documents/Project1src', and a 'Description' text area. The second section is 'Select the type of top-level source for the project' and contains a 'Top-level source type' dropdown menu with 'HDL' selected. At the bottom, there are three buttons: 'More Info', 'Next >', and 'Cancel'.

**Figure 1**

In the next New Project Wizard window, select the target Xilinx FPGA Evaluation board (e.g. the Spartan-6 SP605 Evaluation Platform) in the Evaluation Development Board field, select VHDL in the Preferred Language field and click Next. The last New Project Wizard window must be similar to the one shown in **Figure 2**. Click Finish to create the project `Project1src` or navigate to the previous windows using Back to make corrections.

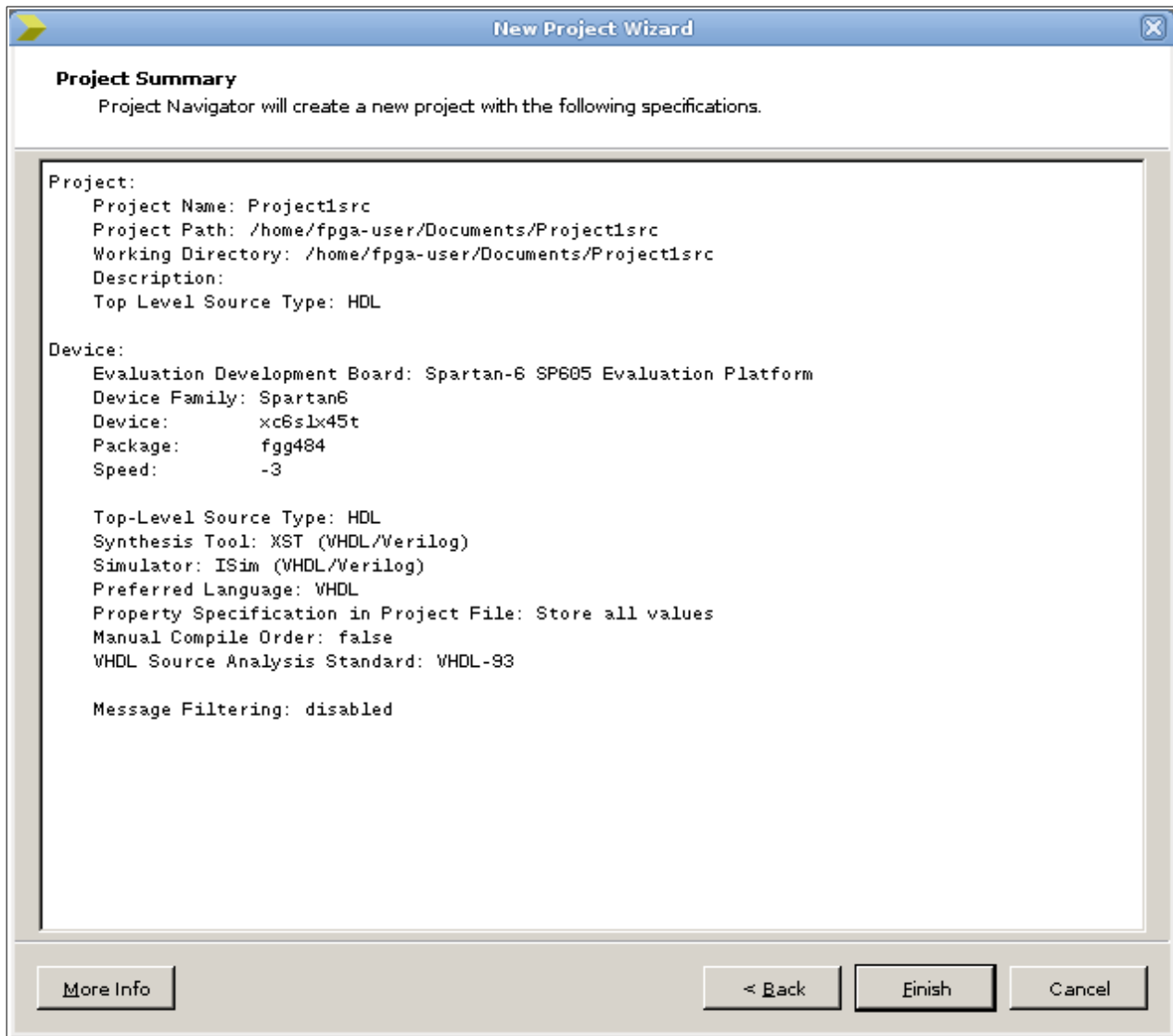


Figure 2

### 3. Add New Source in the Project1src project :

In the ISE Project Navigator window, select Project->New Source.... In the New Source Wizard window, type `scc` in the File Name field, select VHDL Module and click Next (see Figure 3).

The simple combinational circuit will implement the following two simple logic functions :

$$F = AB + \overline{A}\overline{C}$$

$$G = (C + B)(\overline{A} + C)$$

, where A, B, C are 1-bit inputs and F, G are 1-bit outputs.

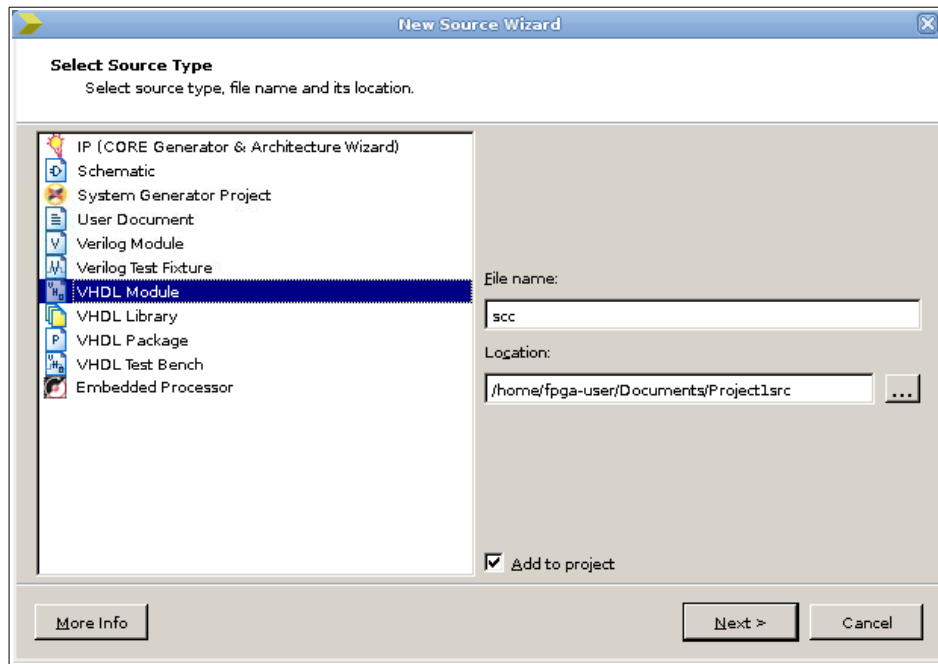


Figure 3

The next New Source Wizard window must be completed as shown in Figure 4.

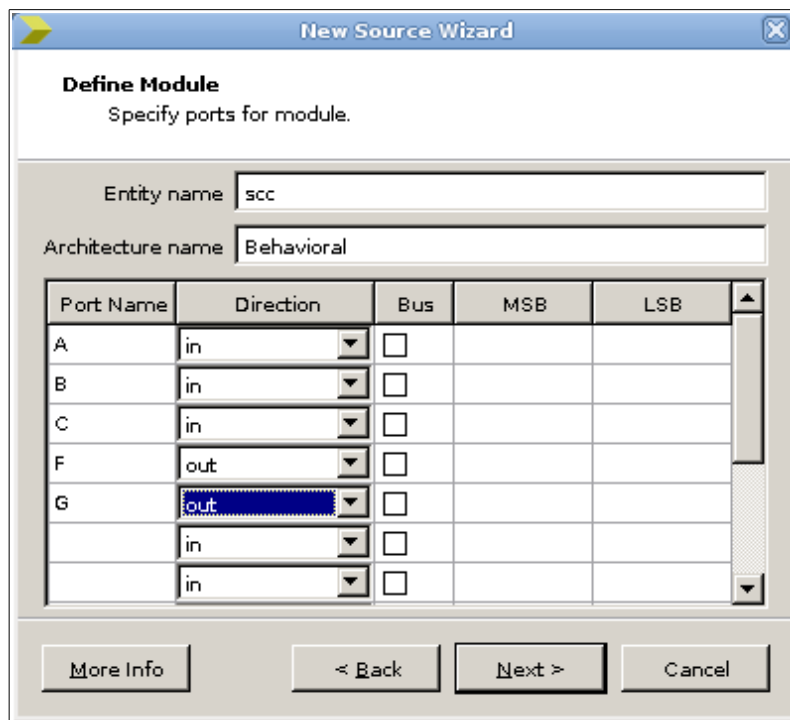


Figure 4

Clicking Next and Finish, the file `scc.vhd` is created.

#### 4. Define the behavior of the `scc` VHDL module :

The `scc.vhd` file is a text file that contains the VHDL description (also called VHDL source code) of the `scc` module, which will implement the `F` and `G` logic functions.

Every VHDL file consists of three parts. The first part declares the libraries and their packages that will be used in `scc`'s source code. The second part, called entity, is an external description of the `scc` module – it declares the inputs/outputs of the module. The third part, called architecture, describes the behavior or structure of the `scc` module (internal description).

A library is a collection of packages and every package consists of one or more VHDL source files. Line 20 of the `scc.vhd` file (**Figure 6**) states that packages from the library `IEEE` will be used and line 21 states that all the parts of the `STD_LOGIC_1164` package will be used in `scc`'s source code. The `STD_LOGIC_1164` package defines the `STD_LOGIC` type. The following values are legal for a `STD_LOGIC` data object : 0, 1, Z, –, L, H, U, X, and W. Only the first four are useful for synthesis of logic circuits. The value Z represents high impedance, and – stands for “don’t care.” The value L stands for “weak 0,” H means “weak 1,” U means “uninitialized,” X means “unknown,” and W means “weak unknown” [1].

Lines 32-38 describe the entity declaration. Module `scc` has three inputs and two outputs of `STD_LOGIC` type.

In order to implement the logic functions `F` and `G` the `scc.vhd` file must be edited. Until this point, only an external description of the `scc` module has been made. Adding the lines :

```
F <= (A and B) or (A and (not C));
G <= (C or B) and ((not A) or C);
```

after the word `begin` (edit lines 43 and 44 – see **Figure 6**) the `scc` architecture is completed and so is the `scc` module. The “<=” is called simple signal assignment and it is used for a logic or an arithmetic expression. A simple signal assignment is a concurrent assignment statement. Concurrent assignment statements describe the behavior of combinational logic circuits. The lines that describe the `F` and `G` logic functions use parentheses to ensure the correct interpretation of the expressions [1]. **Figure 5** shows the VHDL operators grouped into classes. Operators in a given class have the same precedence.

	Operator Class	Operator
Highest precedence	Miscellaneous	<b>**</b> , <b>ABS</b> , <b>NOT</b>
	Multiplying	<b>*</b> , <b>/</b> , <b>MOD</b> , <b>REM</b>
	Sign	<b>+</b> , <b>–</b>
	Adding	<b>+</b> , <b>–</b> , <b>&amp;</b>
	Relational	<b>=</b> , <b>/=</b> , <b>&lt;</b> , <b>&lt;=</b> , <b>&gt;</b> , <b>&gt;=</b>
Lowest precedence	Logical	<b>AND</b> , <b>OR</b> , <b>NAND</b> , <b>NOR</b> , <b>XOR</b> , <b>XNOR</b>

**Figure 5**

**Figure 6** shows the complete VHDL code of the `scc` module. Every line that starts with two dashes “--” is considered a comment in VHDL (lines highlighted with green in **Figure 6**).

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    20:02:58 12/29/2011
6  -- Design Name:
7  -- Module Name:    scc - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
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 scc is
33     Port ( A : in  STD_LOGIC;
34           B : in  STD_LOGIC;
35           C : in  STD_LOGIC;
36           F : out STD_LOGIC;
37           G : out STD_LOGIC);
38 end scc;
39
40 architecture Behavioral of scc is
41
42 begin
43     F <= (A and B) or (A and (not C));
44     G <= (C or B) and ((not A) or C);
45 end Behavioral;

```

**Figure 6**

VHDL is case-insensitive. Changing the case of any letter in **Figure 6** will have no effect in code correctness. Capital letters are usually used for inputs/outputs – better VHDL code readability. For example try changing port A in line 33 to a.

**Figure 7** shows the possible modes for signals that are entity ports [1].

Mode	Purpose
IN	Used for a signal that is an input to an entity.
OUT	Used for a signal that is an output from an entity. The value of the signal can not be used inside the entity. This means that in an assignment statement, the signal can appear only to the left of the <code>&lt;=</code> operator.
INOUT	Used for a signal that is both an input to an entity and an output from the entity.
BUFFER	Used for a signal that is an output from an entity. The value of the signal can be used inside the entity, which means that in an assignment statement, the signal can appear both on the left and right sides of the <code>&lt;=</code> operator.

**Figure 7**

In the ISE Project Navigator window, expand Synthesize – XST and double-click on Check Syntax to ensure `scc` module's syntax correctness (see **Figure 8**).



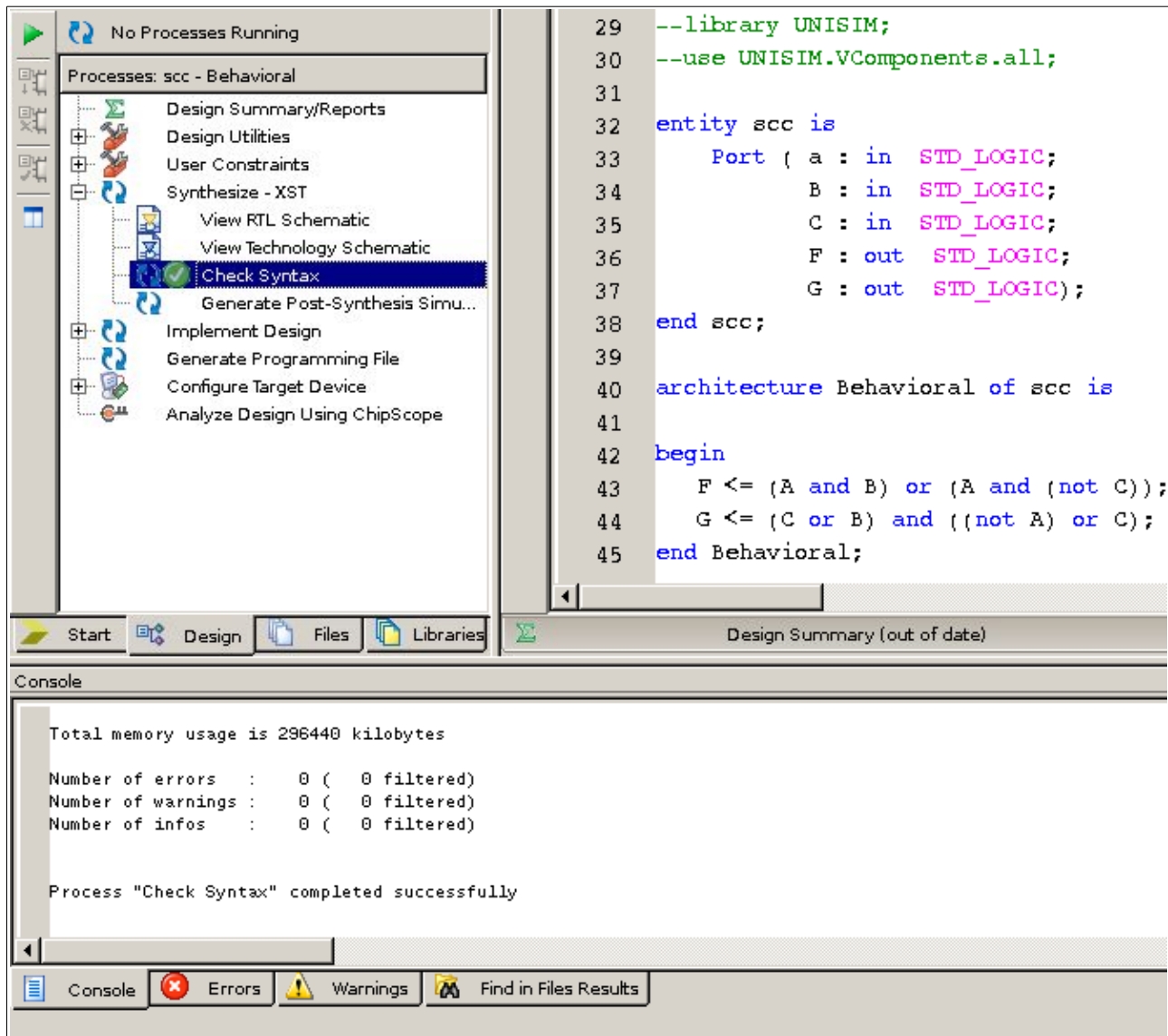


Figure 8

## 5. Behavioral simulation of the scc module :

To simulate the behavior of a VHDL module another VHDL file is needed. This file is called a VHDL testbench and it is used only for simulation. A testbench file instructs a simulation program to apply stimulus (specific values) to VHDL module's inputs and check the generated output values. In simulation, the VHDL module under test is called Design Under Test (DUT) or Unit Under Test (UUT).

In the ISE Project Navigator window select Project->New Source.... In the New Source Wizard window type `scc_tb` in the File Name field, select VHDL Test Bench and click Next (see Figure 9).

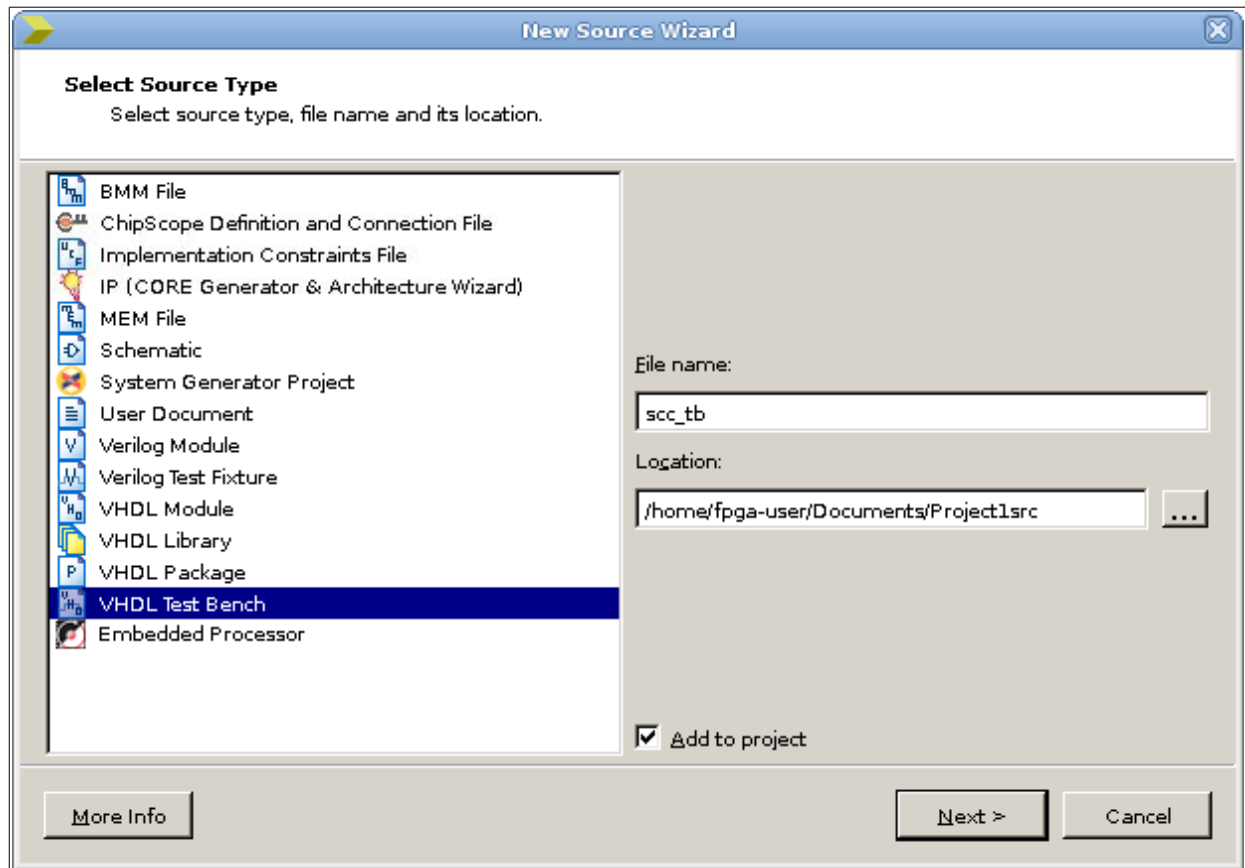


Figure 9

In the next New Source Wizard window associate the `scc_tb` file with the `scc` module, click Next and then click Finish to create the `scc_tb.vhd` testbench file.

Modify the `scc_tb.vhd` testbench file to look like the one in **Figure 10**.

```

1  -- scc's testbench file.
2
3  LIBRARY ieee;
4  USE ieee.std_logic_1164.ALL;
5
6  ENTITY scc_tb IS
7  END scc_tb;
8
9  ARCHITECTURE behavior OF scc_tb IS
10     -- Component Declaration for
11     -- the Unit Under Test (UUT)
12     COMPONENT scc
13     PORT(
14         a : IN  std_logic;
15         B : IN  std_logic;
16         C : IN  std_logic;
17         F : OUT std_logic;
18         G : OUT std_logic
19     );
20     END COMPONENT;
21     --Inputs
22     signal A : std_logic := '0';
23     signal B : std_logic := '0';
24     signal C : std_logic := '0';
25     --Outputs
26     signal F : std_logic;
27     signal G : std_logic;
28 BEGIN
29     -- Instantiate the UUT
30     uut: scc PORT MAP (
31         a => A,
32         C => C,
33         B => B,
34         F => F,
35         G => G
36     );
37
38     -- Stimulus process
39     stim_proc: process
40     begin
41         -- hold reset state for 100 ns.
42         wait for 100 ns;
43         A <= '0';
44         B <= '0';
45         C <= '1';
46         wait for 100 ns;
47         A <= '0';
48         B <= '1';
49         C <= '0';
50         wait for 100 ns;
51         A <= '0';
52         B <= '1';
53         C <= '1';
54         wait for 100 ns;
55         A <= '1';
56         B <= '0';
57         C <= '0';
58         wait for 100 ns;
59         A <= '1';
60         B <= '0';
61         C <= '1';
62         wait for 100 ns;
63         A <= '1';
64         B <= '1';
65         C <= '0';
66         wait for 100 ns;
67         A <= '1';
68         B <= '1';
69         C <= '1';
70         wait;
71     end process;
72 END;

```

Figure 10

In **Figure 10**, lines 6 and 7 describe the entity of the `scc_tb` testbench file. The entity has no ports. This is because a testbench file is used only for simulation – a testbench file doesn't describe a synthesizable logic circuit.

In the architecture declaration and before the word `begin`, lines 10 to 27 describe the unit under test (component declaration) and the interconnection signals that will be used by the simulation program to apply stimulus to the `scc` module (UUT) and check its outputs. Signals could have different names than `scc` module's ports. The rules for specifying port/signal names are simple: any alphanumeric character may be used in the name, as well as the '\_' underscore character. There are four caveats. A name cannot be a VHDL keyword, it must begin with a letter, it cannot end with an '\_' underscore, and it cannot have two successive '\_' underscores [1]. In lines 22-24 every signal that will be used for applying stimulus to the `scc` component is initialized with a logic zero (`:= '0'`).

Lines 30-36 describe the `scc` component's instantiation. Component instantiation begins with a

label, an instance name (`uut`) followed by “:”. Instance naming follows the signal naming rules. In `port map` expression component ports associate with the signals defined in lines 22-27. This is called the named association. In named association the signals listed after the `port map` keywords do not have to be in the same order as the ports in the corresponding component declaration (see **Figure 10**).

Lines 38-70 describe the stimulus process. Stimulus process defines the UUT stimulus at different simulation time intervals. In the first 100 ns of the simulation the UUT inputs have the initial value (line 41, see also lines 22-24 for the initial value). The next 100 ns (line 45) the UUT inputs take the values defined by the lines 42-44 and so on. The `wait` statement in line 69 means that UUT inputs will take the values defined by the lines 66-68 (at simulation time 700 ns) and keep these values forever. Process statement begins with a process name (`stim_proc`) followed by “:”. Process naming follows the signal naming rules.

In the ISE Project Navigator window select Simulation, select `scc_tb` testbench file, expand ISim Simulator and double-click Behavioral Check Syntax to check `scc_tb` testbench's syntax (see **Figure 11**).





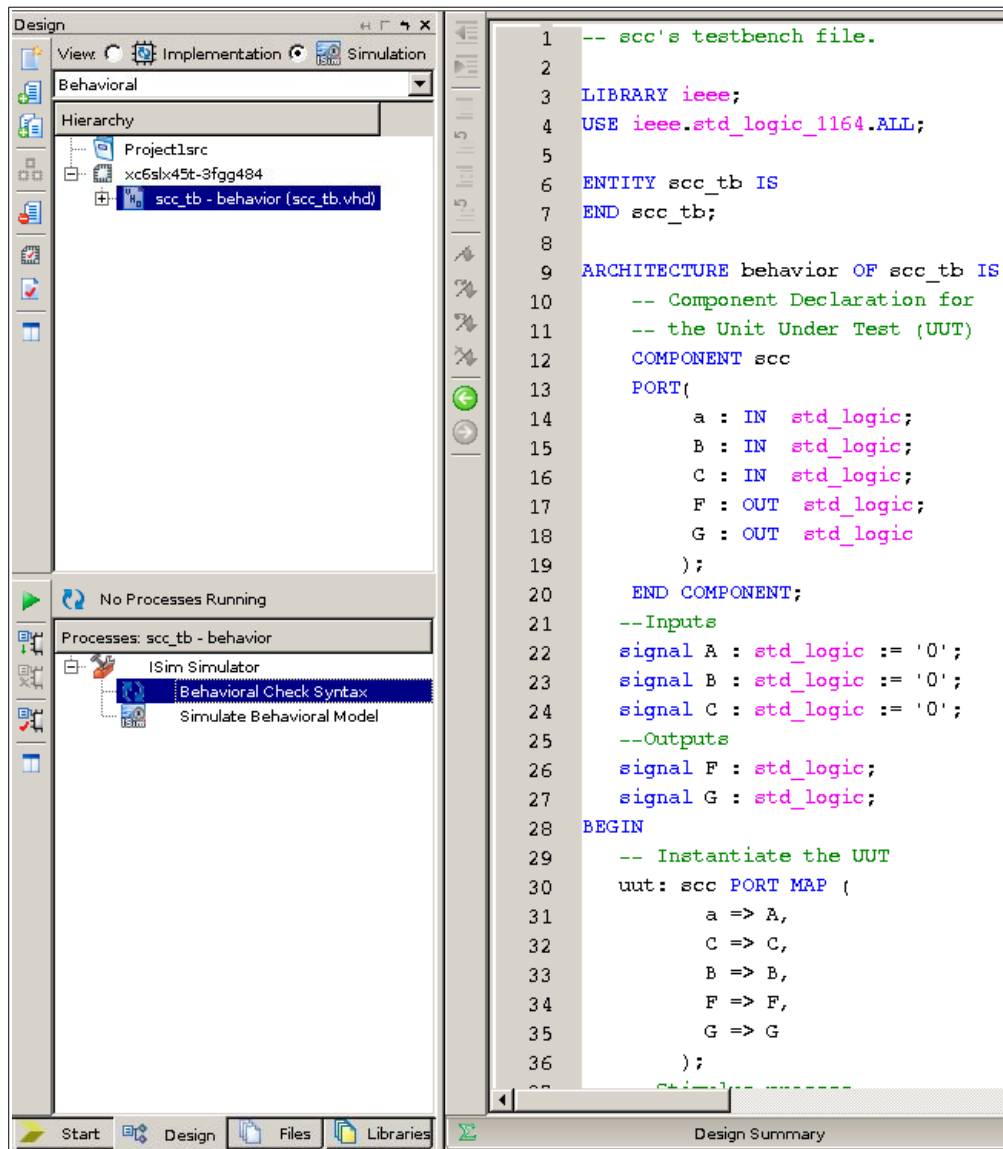


Figure 11

ISim is the name of the simulation tool that is integrated in Xilinx ISE. Right-click Simulate Behavioral Model and select Process Properties. Change Simulation Run Time to 900 ns and click OK. Now double-click Simulate Behavioral Model to run the simulation as specified by the `scc_tb` testbench file and the ISim Simulation Run Time parameter. Select View → Zoom → To Full View in ISim simulator window and compare the resulted waveform with the truth table shown in **Figure 12**.

A	B	C	=>	F	G
0	0	0		0	0
0	0	1		0	1
0	1	0		0	1
0	1	1		0	1
1	0	0		1	0
1	0	1		0	1
1	1	0		1	0
1	1	1		1	1

**Figure 12**

Close the ISim simulator window.

## **6. Synthesis of the `scc` module :**

Synthesis follows behavioral simulation. XST (Xilinx Synthesis Technology) is the name of Xilinx's synthesis tool. In the ISE Project Navigator window select Implementation, select the `scc` module and double-click Synthesize – XST (see **Figure 13**).





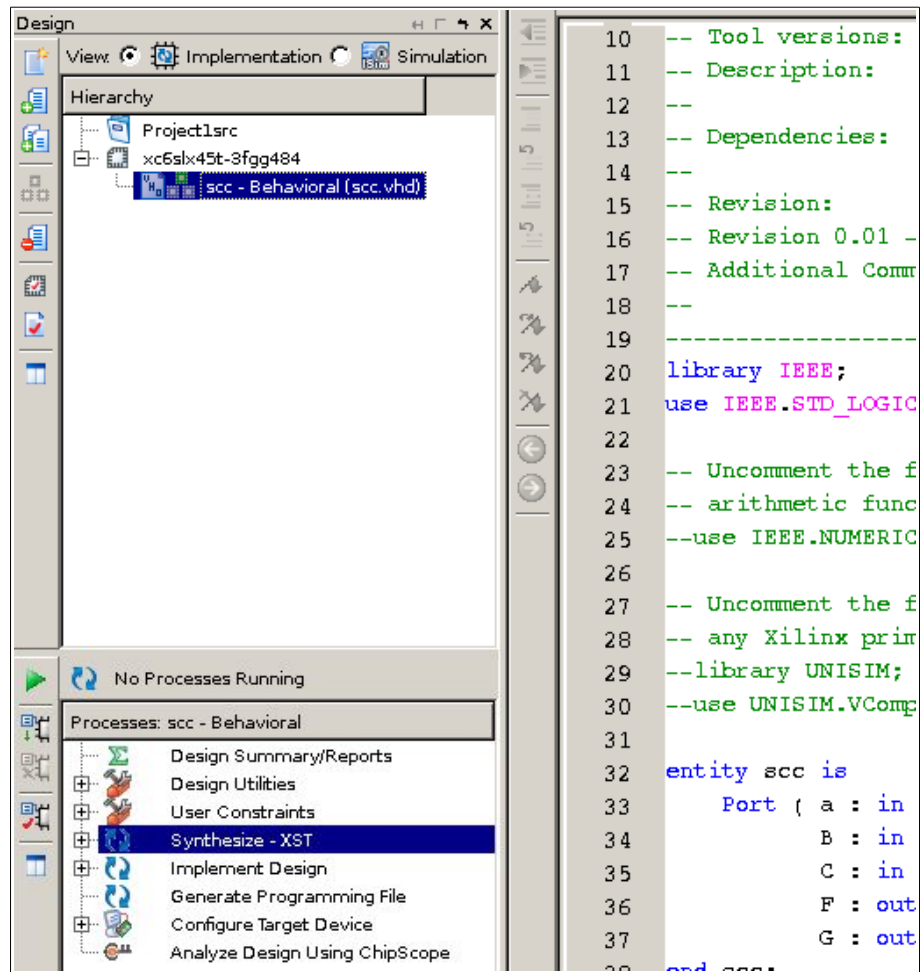


Figure 13

After completing the Synthesize – XST process successfully (see **Figure 14**), if it is not already open, double-click Design Summary/Reports.

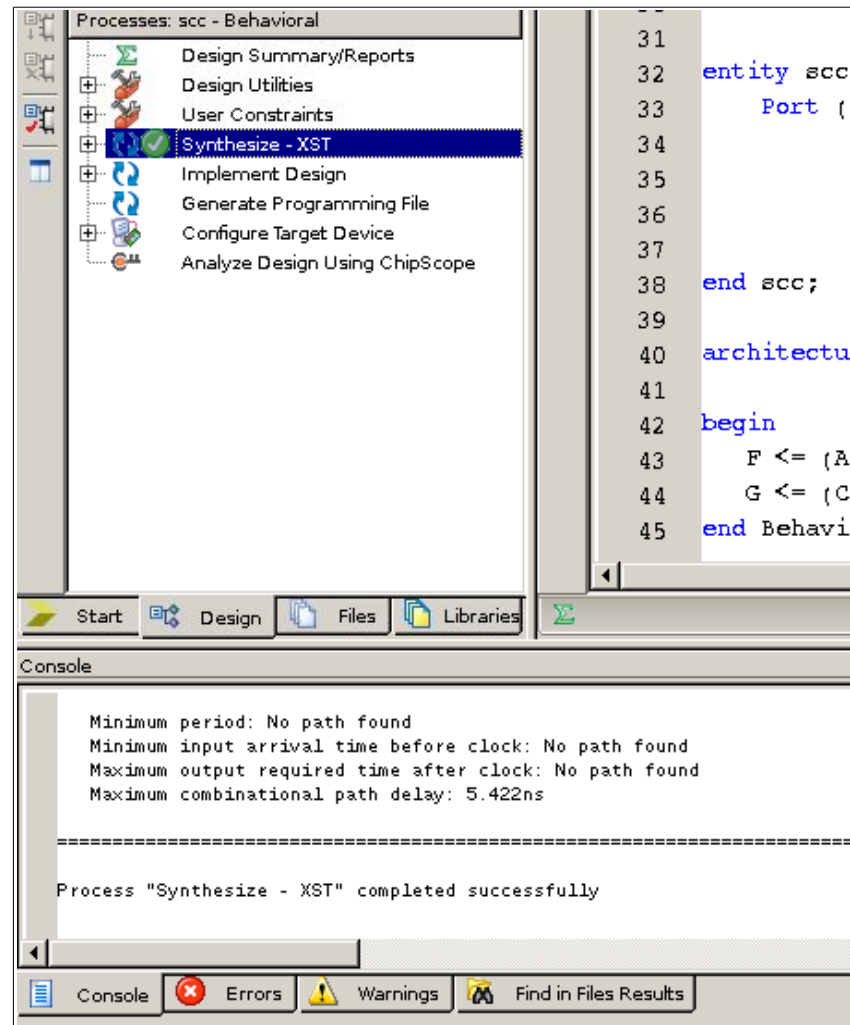


Figure 14

Design Summary provides important information in each step of the design process. **Figure 15** shows the estimated Device Utilization Summary after scc module's synthesis for the specific FPGA device (selected at project creation – Spartan-6 xc6slx45t-3fpg484). Values in **Figure 15** are an estimation because the design isn't implemented yet.

Device Utilization Summary (estimated values)				<a href="#">[1]</a>
Logic Utilization	Used	Available	Utilization	
Number of Slice LUTs	2	27288	0%	
Number of fully used LUT-FF pairs	0	2	0%	
Number of bonded IOBs	5	296	1%	

Figure 15

Expand Synthesize - XST and double-click View RTL Schematic. In the Set RTL/Tech Viewer Startup Mode select Start with a Schematic of the top-level block and click OK. Double-click on the synthesized scc module (black-box-like) graphic to view the gate schematic shown in **Figure 16**.



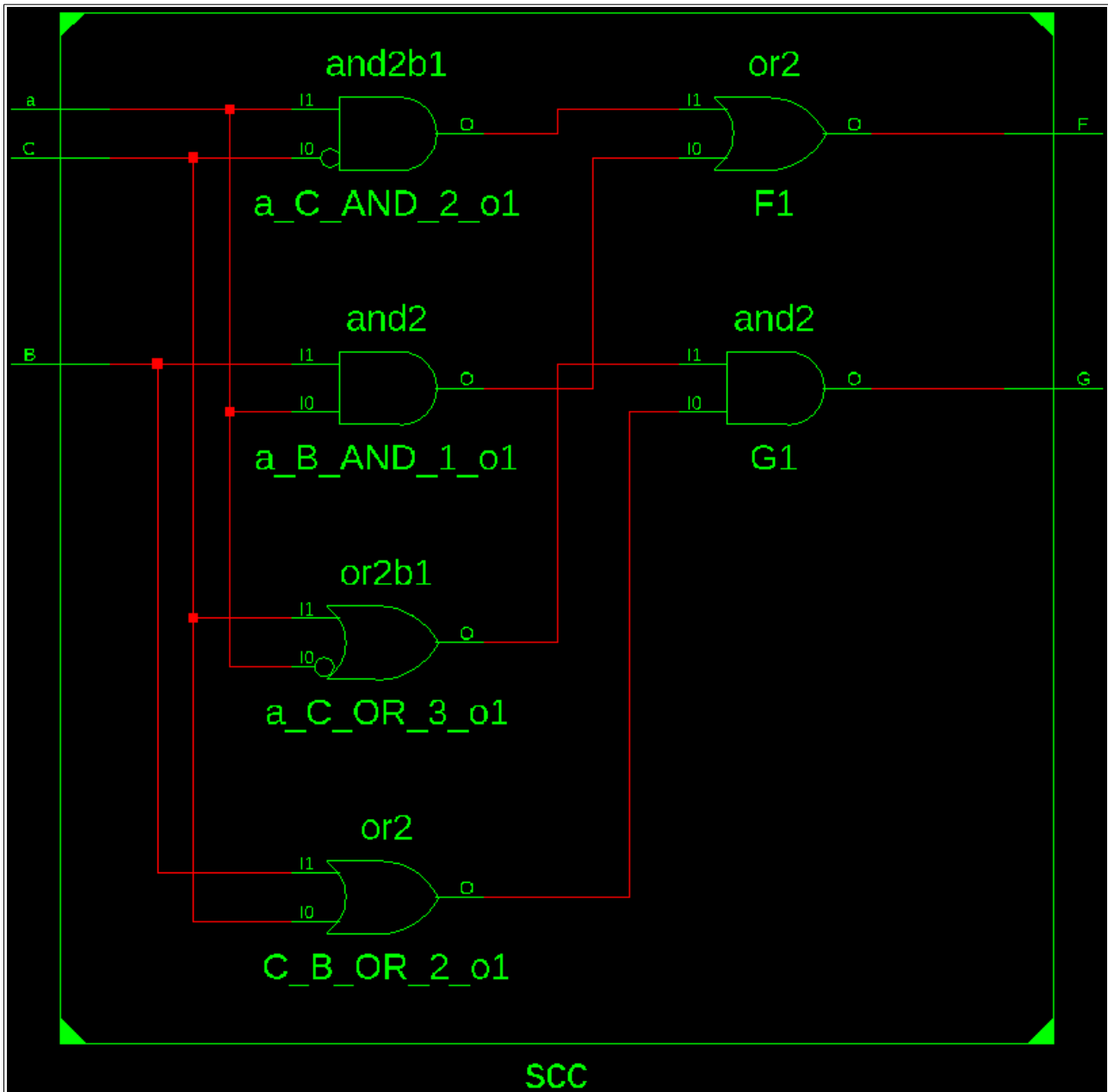


Figure 16

Using the RTL Viewer, the designer can view a schematic representation of the pre-optimized design in terms of generic symbols that are independent of the targeted Xilinx device, for example, in terms of adders, multipliers, counters, AND gates and OR gates [3].

Double-click View Technology Schematic, select Start with a Schematic of the top-level block, in the Set RTL/Tech Viewer Startup Mode, and click OK. Double-click on the synthesized scc module (black-box-like) graphic to view the technology schematic shown in **Figure 17**. Using the Technology Viewer the designer can view a schematic representation of the design in terms of logic elements optimized to the target Xilinx device or "technology", for example, in terms of LUTs, carry logic, I/O buffers, and other technology-specific components [3].

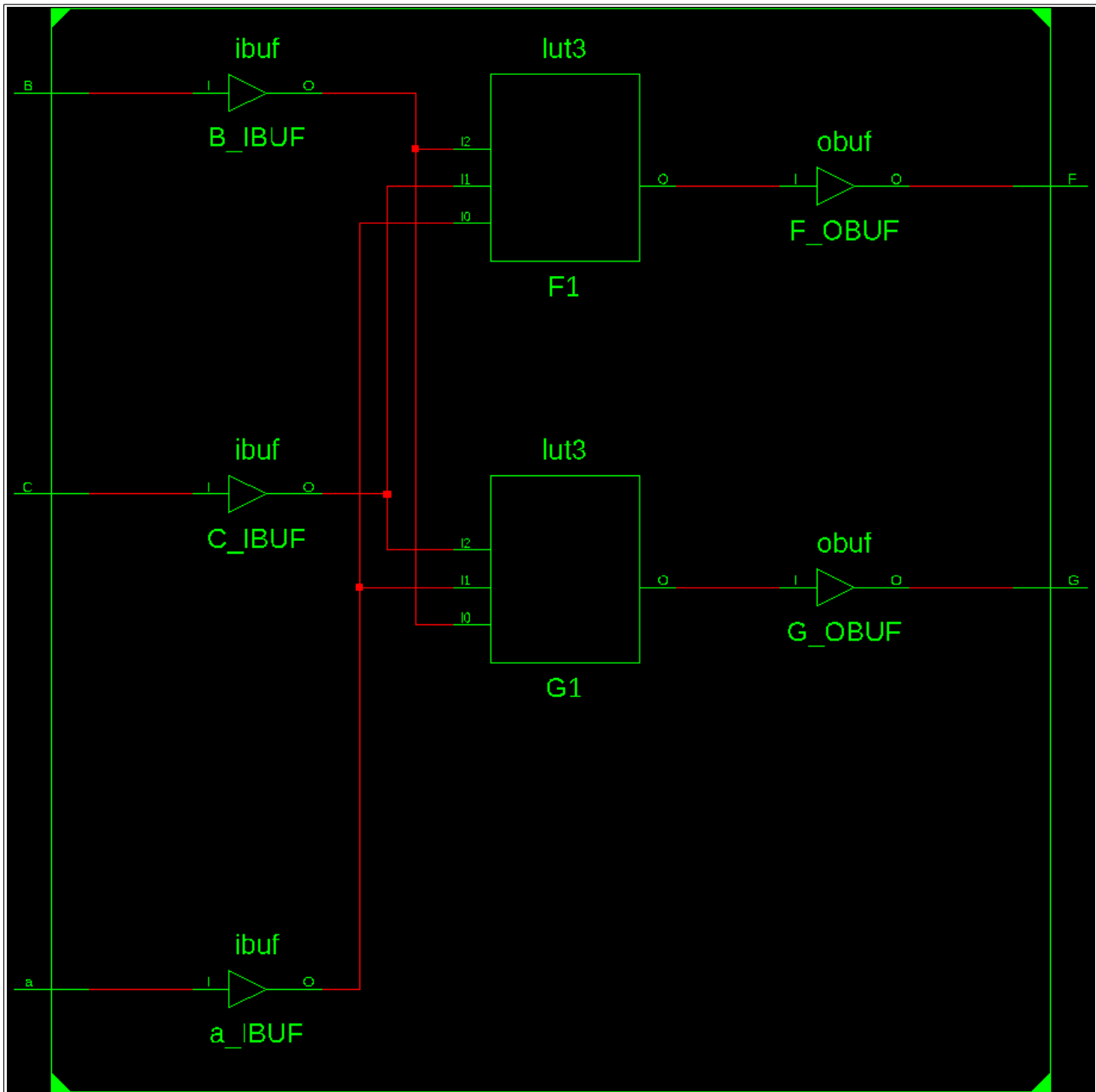


Figure 17

Double-clicking lut3 F1 or lut3 G1 the designer can view the schematic, logic equation, truth table and Karnaugh map representations of the logic expression implemented by the specific LUT. Viewing 3-input LUTs in **Figure 17** does not mean that the specific device (Spartan-6 xc6slx45t-3fgg484) has 3-input LUTs. Spartan-6 device family has only 6-input LUTs [4]. Technology Viewer uses 3-input LUTs in the specific design for presentation purposes.

## 7. Constraints :

Since there are no flip-flops/registers in the design and therefore no clock port, there is no need entering timing constraints. Nevertheless, to implement and test the scc module on a FPGA board, FPGA I/O ports are needed. Ports of the scc module must be associated with FPGA I/O ports to implement and test the design in real time. Components such as switches and leds are connected

to the FPGA's pins on a FPGA board. Specific FPGA I/O ports must be selected before design implementation, because implementation tools need to “know” where to place and how to route the design.

In the ISE Project Navigator window select Implementation, select the `scc` module, expand User Constraints and double-click I/O Pin Planning (PlanAhead) - Post-Synthesis (see **Figure 18**).

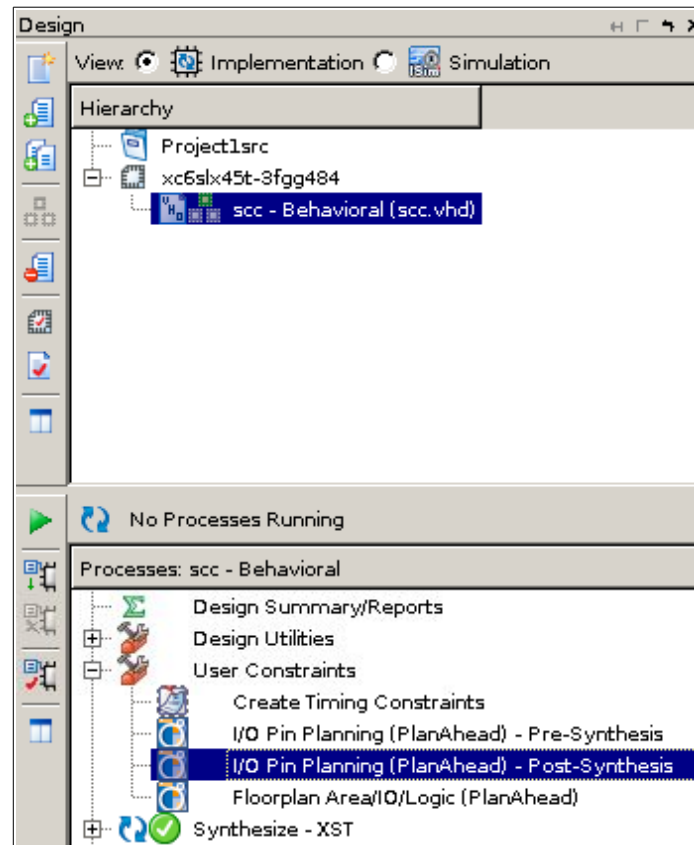


Figure 18

Answer YES to let the Project Navigator to automatically create a UCF (Implementation Constraint File) and add it to the project and wait for PlanAhead 13.3 software tool to open. **Figure 19** shows the PlanAhead interface. Expand Scalar ports in I/O Ports as in **Figure 19**. The Package graphic window shows the Spartan-6 xc6slx45t-3fgg484 FPGA package pins grouped in I/O banks. Using the mouse, drag and drop a, B, C, F and G ports from Scalar ports to the package pins (Package window) as shown in **Figure 20** (pin names are listed under the Site field).



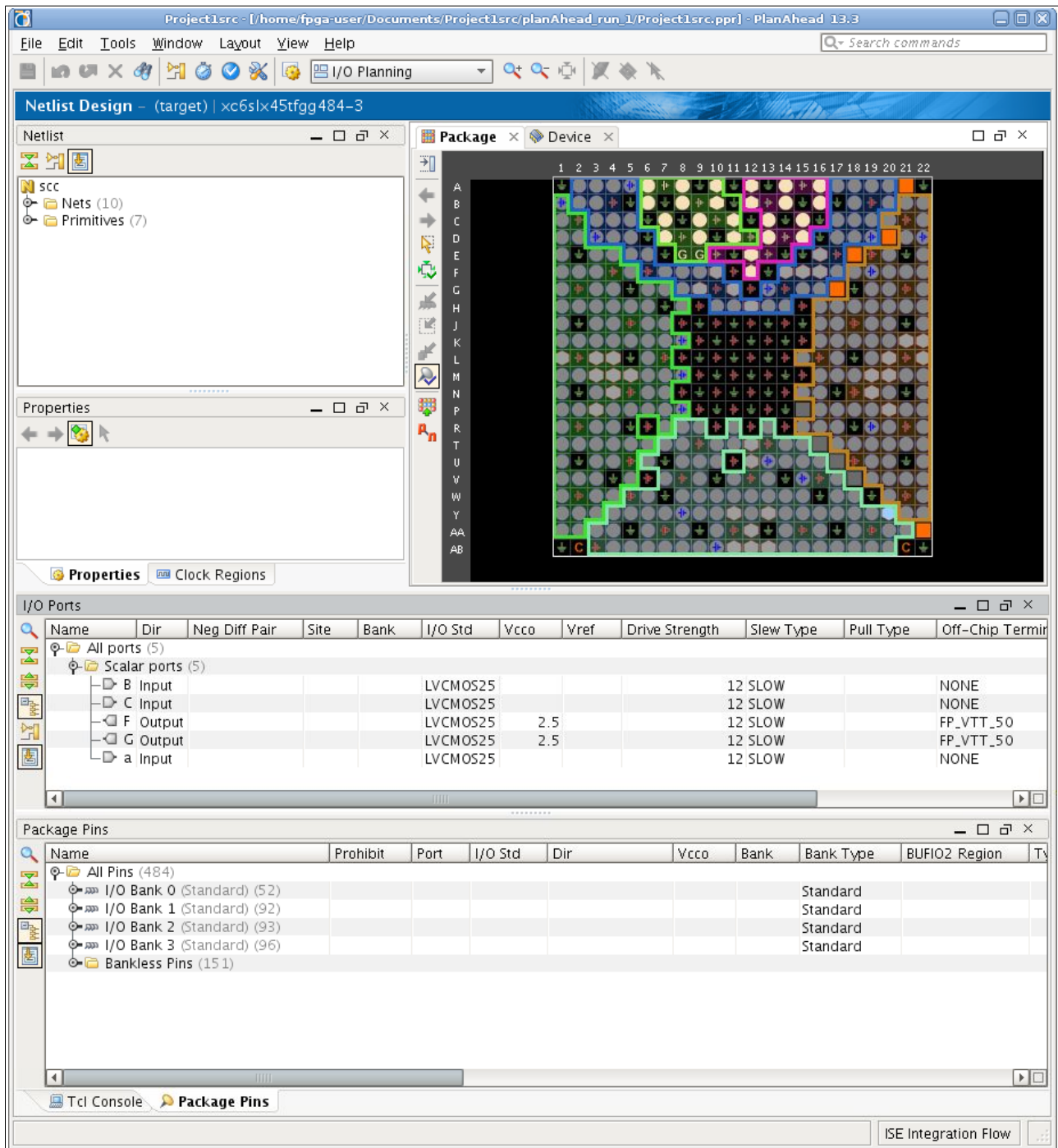


Figure 19

Name	Dir	Neg Diff Pair	Site	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termin
All ports (5)											
Scalar ports (5)											
B Input			Y6		2 LVCMOS25			12 SLOW			NONE
C Input			W6		2 LVCMOS25			12 SLOW			NONE
F Output			D17		0 LVCMOS25	2.5		12 SLOW			FP_VTT_50
G Output			AB4		2 LVCMOS25	2.5		12 SLOW			FP_VTT_50
a Input			C18		0 LVCMOS25			12 SLOW			NONE

Figure 20

The FPGA pins associated with scc's ports are connected to board DIP switches and leds. **Figure 21** shows the Spartan-6 (xc6slx45t-3fgg484) FPGA pin connections to the DIP switches and leds on the SP605 Evaluation Board [5].

U1 FPGA Pin	DIP Switch Pin
C18	S2.1
Y6	S2.2
W6	S2.3
E4	S2.4

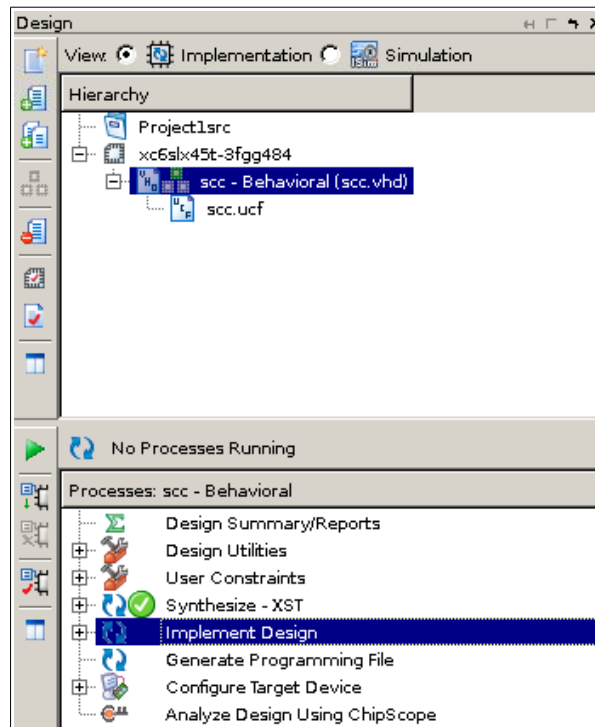
U1 FPGA Pin	Controlled LED
D17	DS3
AB4	DS4
D21	DS5
W15	DS6

Figure 21

In the PlanAhead window select **File** → **Save Design**, close the PlanAhead window and return to the ISE Project Navigator window.

## 8. Design Implementation :

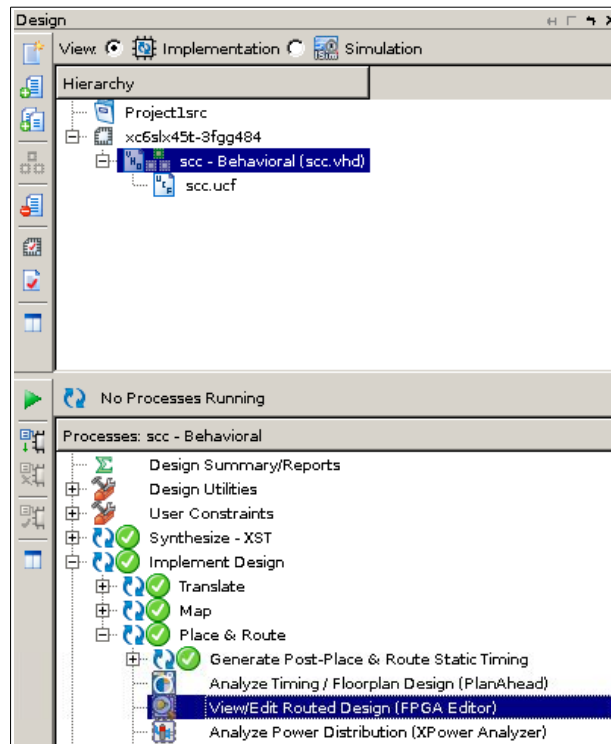
In the ISE Project Navigator window select **Implementation**, select the `scc` module and double-click **Implement Design** (see **Figure 22**).



**Figure 22**

After completing the Implement Design process successfully, Design Summary provides information regarding the exact FPGA utilization and FPGA pinout. Module `scc` uses 1 of 27288 Spartan-6 (xc6slx45t-3fgg484) Slice LUTs and 5 of 296 Spartan-6 (xc6slx45t-3fgg484) IOBs (I/O blocks).

In the ISE Project Navigator window select Implementation, select the `scc` module, expand Implement Design, expand Place & Route, double-click View/Edit Routed Design (FPGA Editor) (see **Figure 23**) and wait for the Xilinx FPGA Editor window to open.



**Figure 23**

Maximize the Xilinx FPGA Editor window and select Window → Arrange Windows → Default Layout. With the FPGA Editor the designer can view/edit the design implemented on the FPGA. Double-click G\_OBUF in the List1 subwindow to view the SLICEX Slice [4] that the implementation tools used for scc module implementation (see **Figure 24**). Double-clicking the red colored Slice the designer can view the LUTs inside the specific slice.

Use zoom-in and zoom-out tools to view design routing.

Close the Xilinx FPGA Editor window and return to the ISE Project Navigator window.

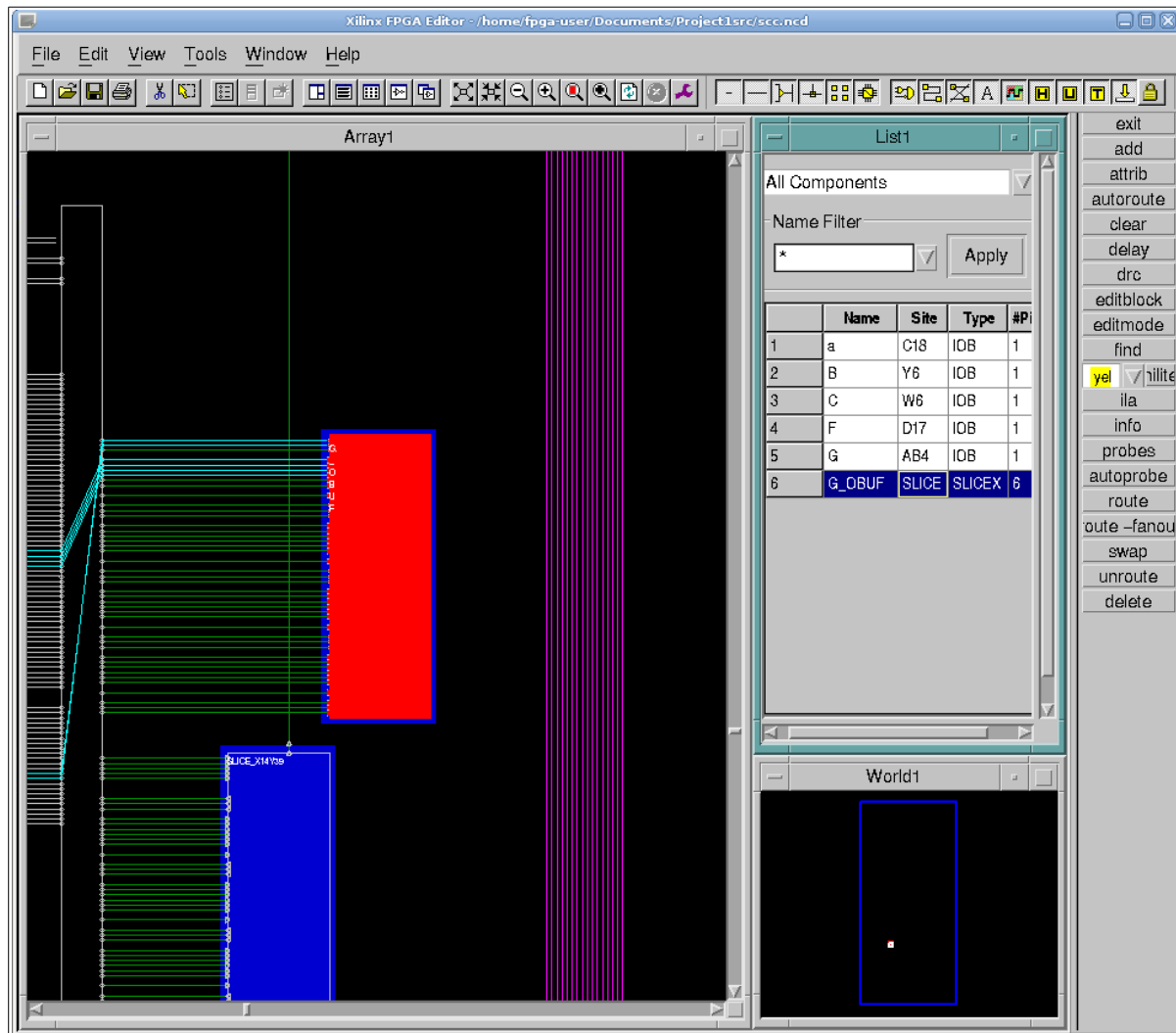


Figure 24

## 9. Post-Place & Route Simulation :

In the ISE Project Navigator window select Implementation, select the `scc` module, expand Implement Design, expand Place & Route and double-click Generate Post-Place & Route Simulation Model (see Figure 25). The Generate Post-Place & Route Simulation Model creates a model that contains true timing delay information of the `scc` design implementation.

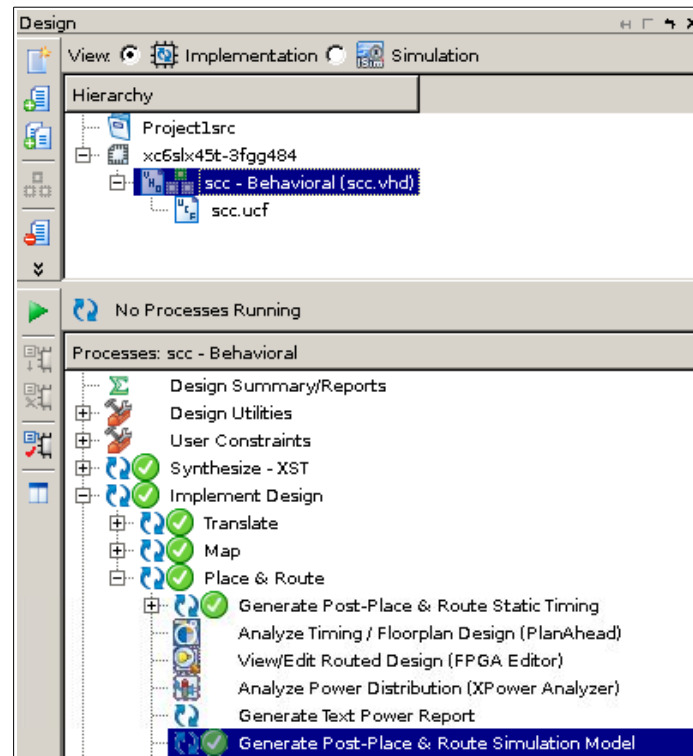


Figure 25

In the ISE Project Navigator window select Simulation, select Post-Route, select the `scc_tb`, expand ISim Simulator (see Figure 26), right-click Simulate Post-Place & Route Model and select Process Properties. In the Process Properties – ISim Properties window set Simulation Run Time to 900 ns and click OK. Double-click Simulate Post-Place & Route Model to run the simulation.

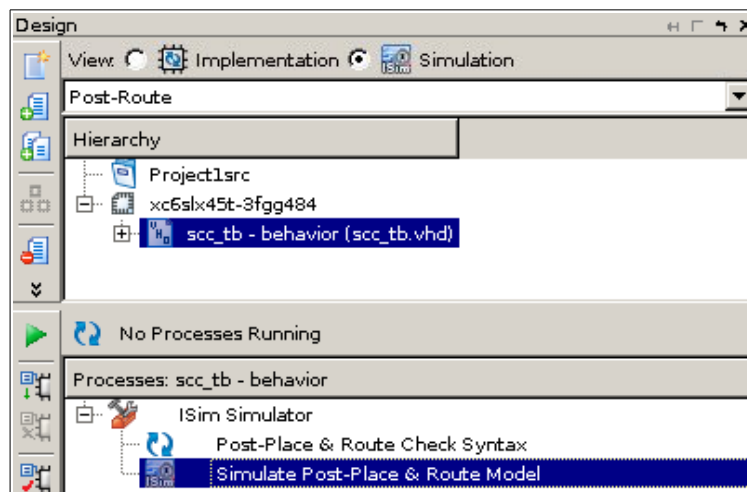


Figure 26

In the ISim window select View → Zoom → To Full View. Observe the propagation delay (F and G outputs) due to the timing delay information included in the Post-Place & Route Simulation Model.

Close the ISim window and return to the ISE Project Navigator window.

## 10. Programming (Configuration) File Generation :

In the ISE Project Navigator window select Implementation, select the `soc` module, right-click Generate Programming File and select Process Properties. In the Process Properties - Startup Options window select Startup Options category and change FPGA Start-Up Clock property to JTAG Clock, then select Readback Options category and check Create Readback Data Files and Create Mask File properties (see **Figure 27**). Click OK to close the Process Properties - Startup Options window. The above changes have been made in order to program the FPGA and verify FPGA programming (not the design) using JTAG.

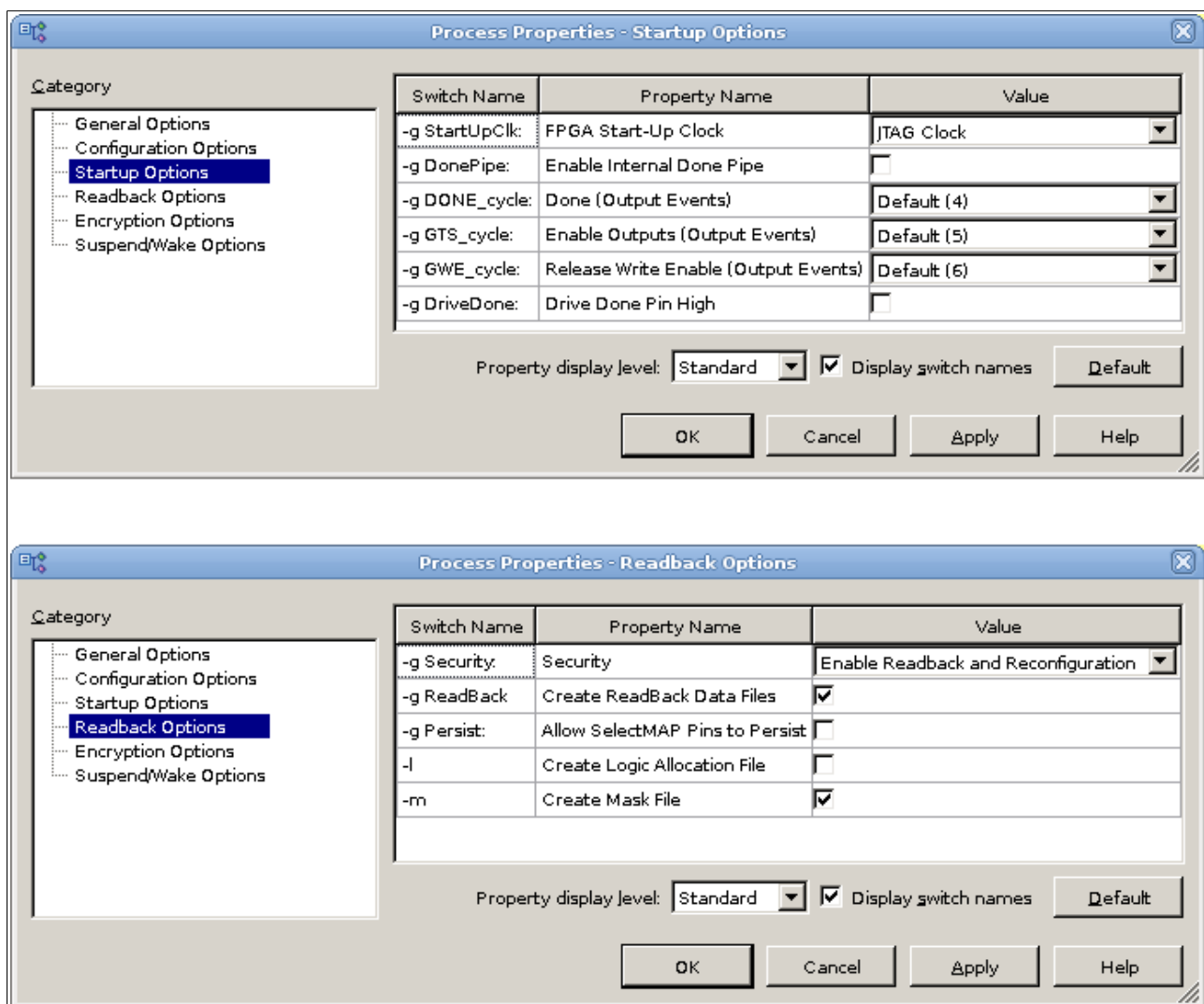
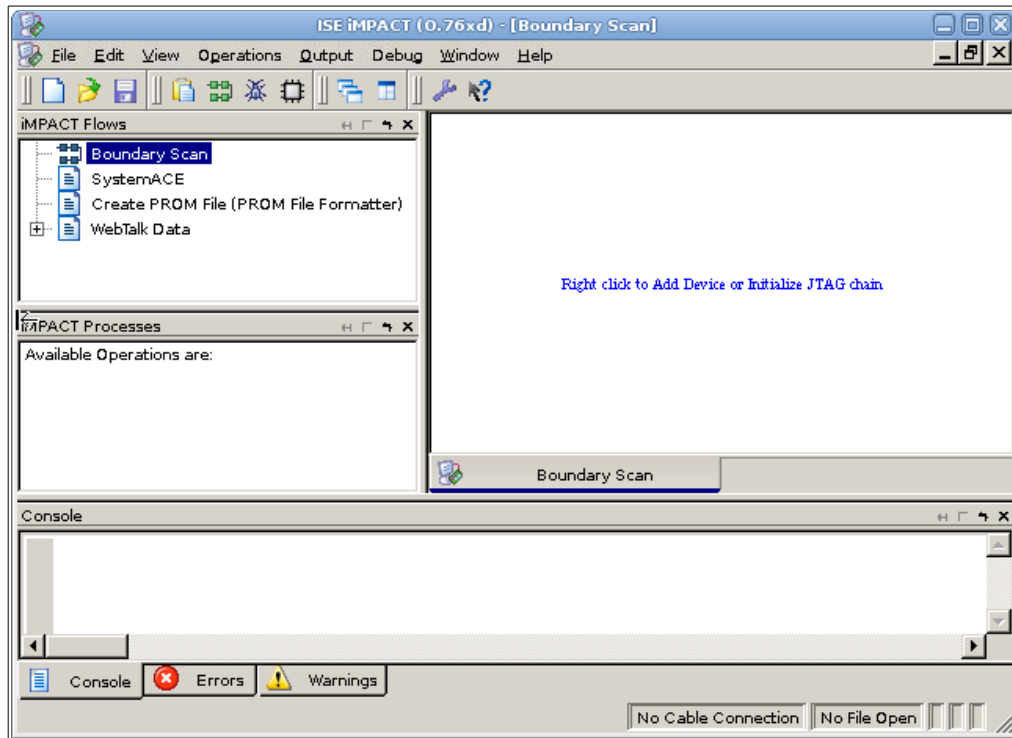


Figure 27

Double-click Generate Programming File to complete programming file generation.

## 10. FPGA Configuration :

In the ISE Project Navigator window select Implementation, select the `scc` module, double-click Configure Target Device and click OK to open Impact. Impact software tool is used for FPGA configuration. In the ISE iMPACT window double-click Boundary Scan (see **Figure 28**).



**Figure 28**

Right-click on Right click to Add Device or Initialize JTAG chain and select Initialize Chain. Click Yes to the Auto Assign Configuration Files Query Dialog, select Bypass for the `xc6slx45t` device, navigate to `/home/fpga-user/Documents/Project1src`, select `scc.bit` file and click Open for the `xc6slx45t` Spartan-6 FPGA. Click No to the Attach SPI or BPI PROM window. Select Device 2 in the Device Programming Properties window and check the Verify property. Right-click on the `xc6slx45t` Spartan-6 FPGA and select Program (see **Figure 29**).



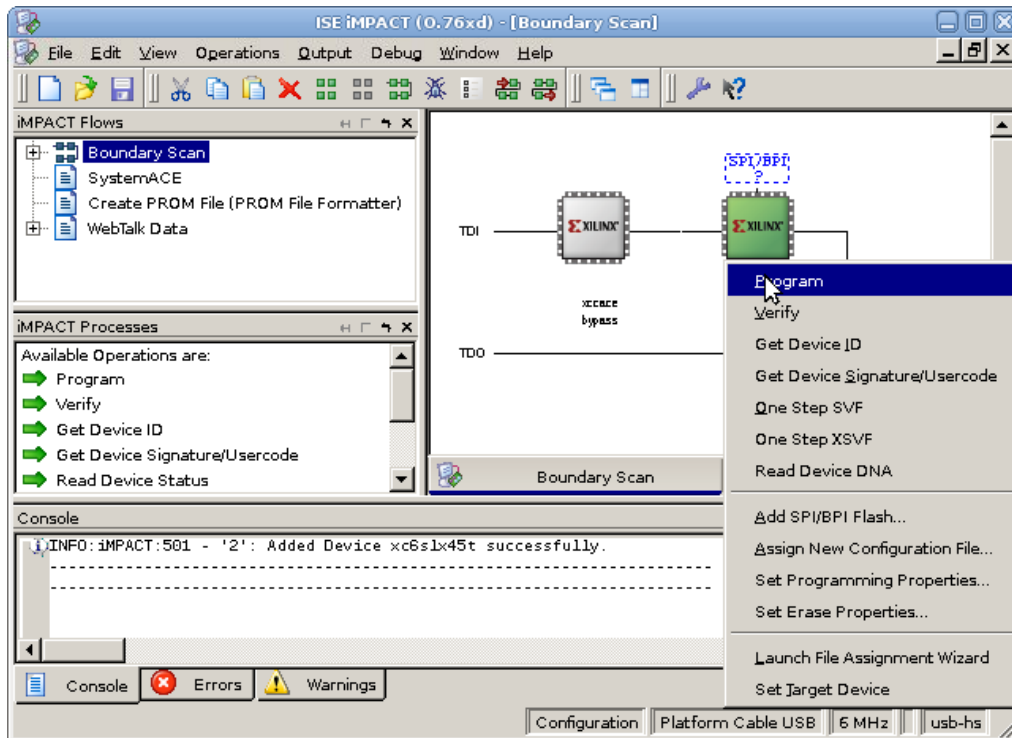


Figure 29

After the Program Succeeded message, the FPGA is configured and the FPGA board (SP605) is ready for testing the scc design. Use the board DIP switches and leds to test the scc design in real time (see Figure 30).

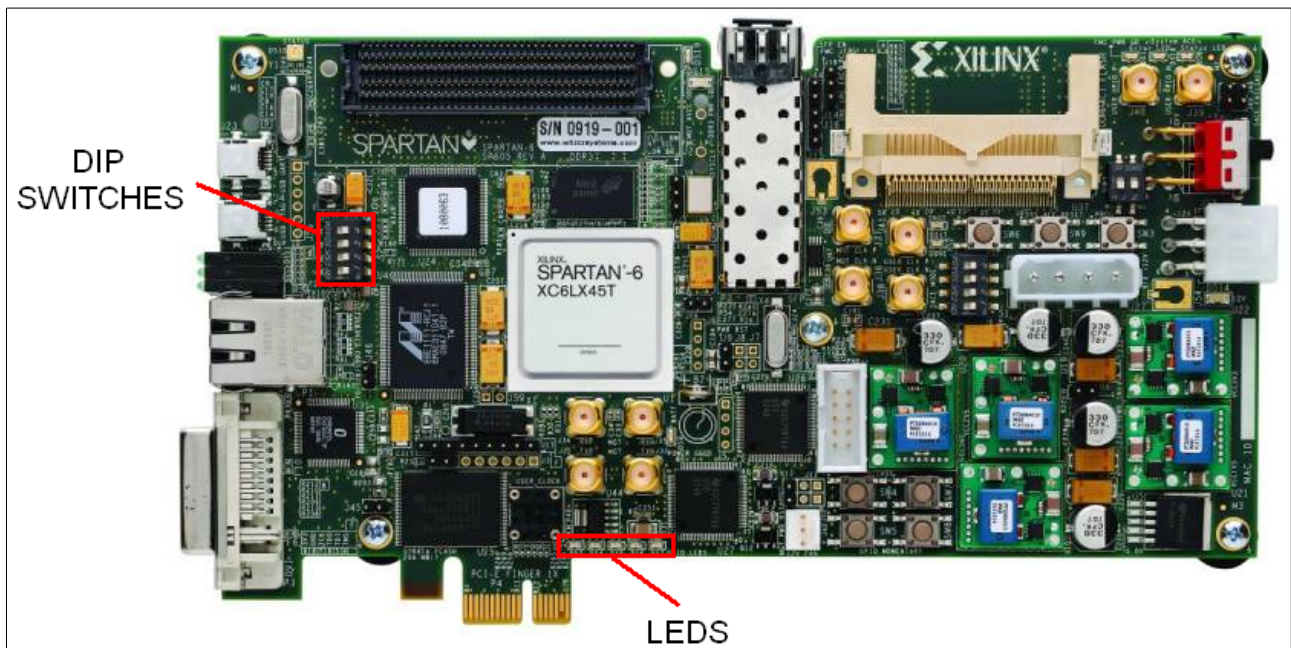


Figure 30

Close the iMPACT window and the ISE Project Navigator window.

## ***References***

- [1] Stephen Brown and Zvonko Vranesic, Fundamentals of Digital Logic with VHDL Design, 3<sup>rd</sup> Edition, McGraw-Hill, 2009
- [2] David Money Harris and Sarah L. Harris, Digital design and computer architecture, Morgan Kaufmann - Elsevier, 2007
- [3] Xilinx ISE 13.3 Help
- [4] Xilinx Corporation, Spartan-6 FPGA Configurable Logic Block User Guide, 2010, <http://www.xilinx.com>
- [5] Xilinx Corporation, SP605 Hardware User Guide, 2011, <http://www.xilinx.com>