Introduction to digital logic design in Simulink

Nasibeh Teimouri
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA

February 19, 2015

Contents

List of Figures

List of Acronyms

AMBA Advanced Microcontroller Bus Architecture
AMBA AHB AMBA High Speed Bus
Atomic Block A block **Block** whose content is computed one time
Block Parameters and states in Simulink
EDK Embedded Development Kit
FPGA Field Programmable Gate Array
GUI Graphical User Interface
HDL Hardware Description Language
ISE Integrated Software Environment
JTAG Joint Test Action Group
Line Signals passed among blocks
PL Programmable logic
PS Processing System
RTL Register Transfer Level
SoC System on Chip
UART Universal Asynchronous Receiver and Transmitter

Abstract

All electronic equipment, including computers, use digital electronics where the quantities (i.e., voltages) are described by two states; on and off or in most physical systems are represented by the voltages +5V (+2.9V, +2.5V or some other standardized value) and 0V. Due to the importance and prominence of digital logic design, it is necessary to have a tools to capture and simulate the behavior of the intended design before moving a real implementation. In this tutorial, we will study Simulink (Simulation + Link) developed by the Mathworks, which is tightly integrated with MatLab.

Simulink provides a Graphical User Interface (GUI) to help users capture their designs as they would do with pen and paper, but captures it digitally. The user can then simulate the designs to verify design correctness before mapping it to a FPGA (Field Programmable Gate Aray). For this, Simulink provides the capability to generate HDL code of the design, which is used to program the FPGA, in your case, the FPGA on the Zedboard.

1 Introduction to Simulink for digital logic design

This section introduces Simulink and demonstrates the different features available for constructing designs. In the first part we present a general introduction to Simulink. This is followed by a discussion of some of the features of Simulink – going through the design process for a simple half adder with two boolean inputs and two boolean outputs (sum and carry out).

1.1 What is Simulink?

Simulink (Simulation and Link) is developed by the MathWorks [1] and is tightly integrated with MatLab (Simulink is one of Matlab's add-on products). Simulink provides a rich GUI for modeling, simulating and analyzing dynamic systems. Simulink enables rapid construction of virtual prototypes to explore design concepts at any level of detail with minimal effort.

The zeroth step for designing with Simulink (as with other tools) is to develop a design specification; fully specifying the system and its requirements. The specification covers what the input ranges and types are, what is expected as output and how the system is supposed to work. After you specify these high-level details of a system, the next step is modeling, where Simulink helps you visualize and debug your design.

Modeling is a process that enables faster and more cost-effective development of dynamic systems. This is especially helpful for systems with nondeterministic behavior. To build and understand a system before mapping the design to a real implementation, we would like to simulate the system in software. When we model our design in Simulink, we can model and test the behavior of our design with test inputs. After completing design verification, the design can be iteratively refined to finally produce the desired behavior.

All of these steps, which include modeling, testing, verification and refinement, can be done in Simulink. In this tutorial we will go through these steps as we build our knowledge of designing systems with digital logic.

Modeling:

To develop a model in Simulink, the user draws the designs as he/she would do with pencil and paper. For the purposes of drawing, the user can use the blocks from a comprehensive library of predefined blocks to construct graphical models of the system using drag-and-drop mouse operations.

In addition to this straightforward modeling approach, Simulink helps to produce an up-and-running model that would otherwise require hours to build in a laboratory environment. The only thing that you need to do is to break up any complex/large system of components into a number of

smaller sub-systems, each comprising a different functionality. You will find the necessary blocks are available in the Simulink libraries, which provide the functionality of the elements that are present in your design. As you develop your design in Simulink, you need to ask yourself the following questions about each subsystem:

- How many input signals does the subsystem have?
- How many output signals does the subsystem have?
- How many states (variables) does the subsystem have?
- What are the parameters (constants) in the subsystem?
- What are the intermediate (internal) signals in the subsystem?

where parameter, state and signal are 3 types of Simulink components defined as follows:
1. Parameters: System values that remain constant, unless you change them.
2. States: Variables in the system that change over time.
3. Signals: Input and output values that change dynamically during a simulation.

Note1- In Simulink, parameters and states are represented by blocks, while signals are represented by the lines that connect blocks.

For each subsystem that you have identified, once you have answered those questions, you should have a comprehensive list of system components, and then you are ready to begin modeling the system in Simulink.

Test, verification and refinement:

After you have modeled each subsystem, you can simulate them individually to make sure that each works correctly. Since you are developing your design in software, you can always change the design and re-simulate it to correct/improved functionality. Afterward, you can integrate the subsystems together to form your model of the complete system. You can then perform simulation and analysis of the whole system. After you verify system correctness, you can apply further refinements to make your system run more efficiently.

Note2- Simulink allows you to interactively define system inputs, simulate the model, and observe changes in behavior. This design loop provides you with the ability to perform quick evaluation of your model. You can change and refine your model to produce the design that you expect.

Code generation:

After modeling is completed, when the software and hardware implementation requirements are included and the correctness of the design is verified, Simulink can automatically generate code for your embedded system implementation.

The Simulink Coder (in our case, the Simulink HDL coder) generates and executes Hardware Description Language (HDL) code from your Simulink schematics (schematics is a term to describe your design implementation). Simulink generated source code can be used in a wide range of applications, including both realtime and more general applications. You can tune/monitor the generated code using Simulink, or you can run/interact with the code outside of MATLAB and Simulink [2].

Direction- We will focus our attention next on the subset of Simulink that is in the Embedded Design class. This tutorial will cover the topics to support learning in the course; this tutorial will continue with the library logic and bit operations that will enable us to design digital logic circuits.

1.2 Digital logic designing with Simulink

In this section, we will go through a modeling exercise, implementing a simple half adder step-by-step, in order to explore the different features in Simulink required for your lab and your digital design assignments.

The digital logic of a standard half adder is shown in Fig. 1. As discussed in Section 1.1, we start with the zeroth step which is the specification. To implement the half adder, there are two boolean inputs and also two boolean outputs; one output generates the sum (S), and the other output generates the carry out (C). The required functionality involves XORs of the two inputs to generate S and ANDs of the two inputs to generate C. You now have a complete specification of a half adder.

Note3- Since an XOR block is already provided in the library, you do not need to break it down further into subsystems. Otherwise, you would need to combine some AND and OR gates to implement a XOR gate.

The next step is to turn your specification in an executable. Simulink makes it easy for you to compile your design.

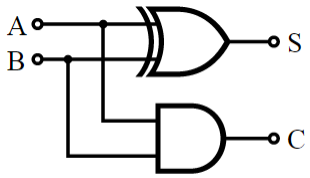

Figure 1: logic design of half-adder

Start Simulink:

In the Simulink Editor, you can build and simulate models of your system. You need the Simulink Library Browser open before you can create a new Simulink model. To start Simulink, you need to have MATLAB running. We can start by opening MatLab, then there are two ways to start running Simulink, as shown in Fig. 2.

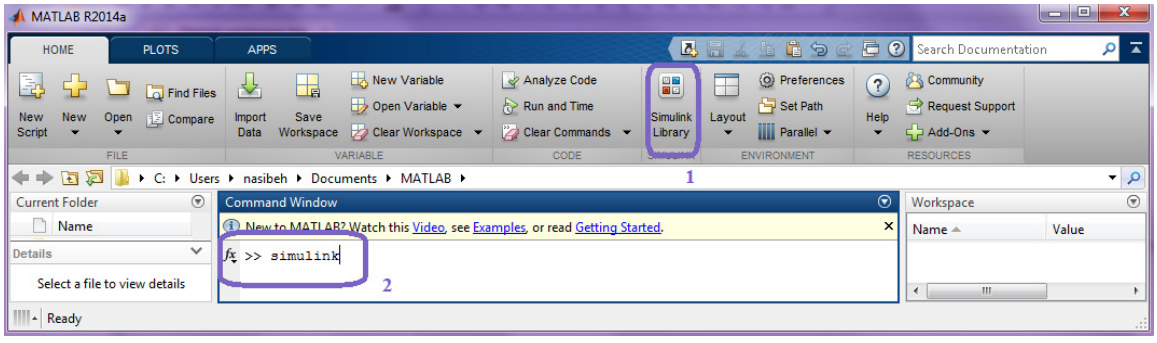

Figure 2: The open Simulink library through the toolbar.

1. Locate the Simulink Library button in the toolbar.
2. In the MATLAB Command Window, enter Simulink.

Note4- A short delay may occur before the Simulink Library Browser opens – you need to be patient.

After opening Simulink, you should make sure the Library Browser appears above all other windows on your desktop. To do this, in the Library Browser, select View>Stay on Top, as shown inFig. 3.

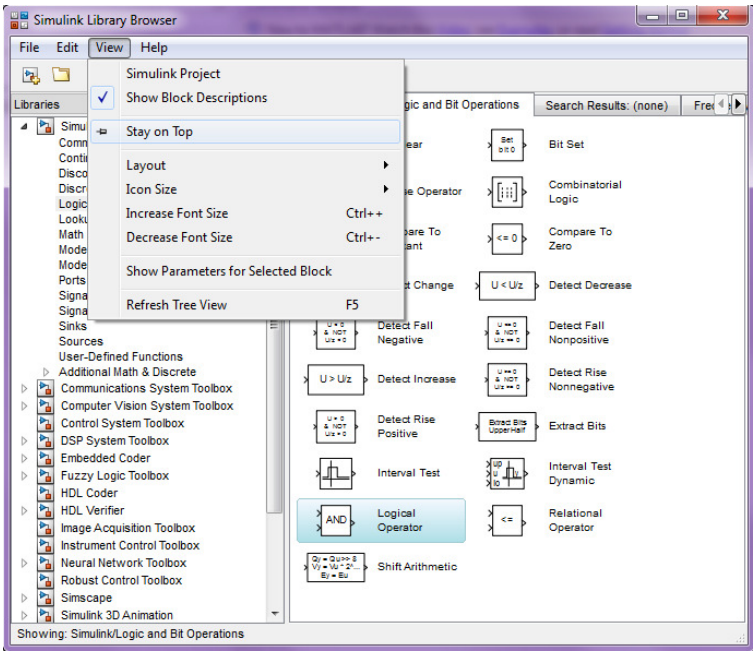

Figure 3: Allowing the Simulink library browser to appear as the top window.

Browse or Search for Specific block:

There are two major classes of items in Simulink: blocks and lines. Blocks are used to generate, modify, combine, output and display signals. Lines are used to transfer signals from one block to another. During the modeling process, you need to select blocks and combine them using lines to implement your design.

The Simulink online Help supplies a comprehensive online help describing features, blocks, and functions with detailed procedures. You are encouraged to use it. There is also offline Help embedded in the tool, and you can invoke as shown in Fig. 4 shows.

1. Click on Help on the toolbar to obtain access to the Simulink help services.
2. Type the name of the block that you trying find information about.
3. Right click on the block to find the specific help for that.

Creating a New Simulink Model:

Before starting the modeling process, you will need to create a new Simulink model in the Simulink Library Browser. To create a new model, from the Simulink Library Browser Toolstrip, click the New Model button, as shown in Fig. 5, or from the menu bar, select File >New >Model. These steps create an empty model and open it in the Simulink Editor. Save it with a name of your choice. Name the file to help you find it easier in the future.

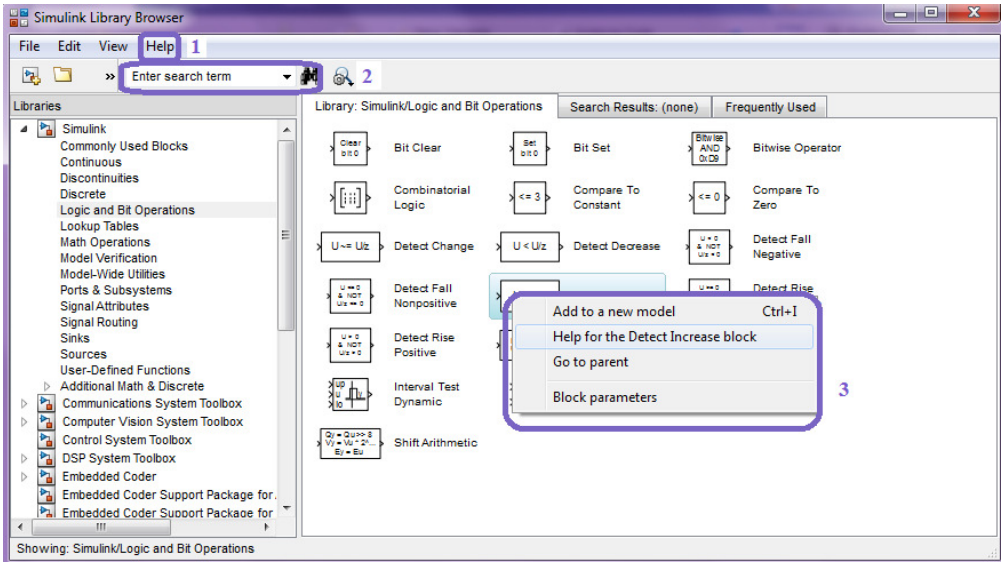

Figure 4: The open Simulink help.

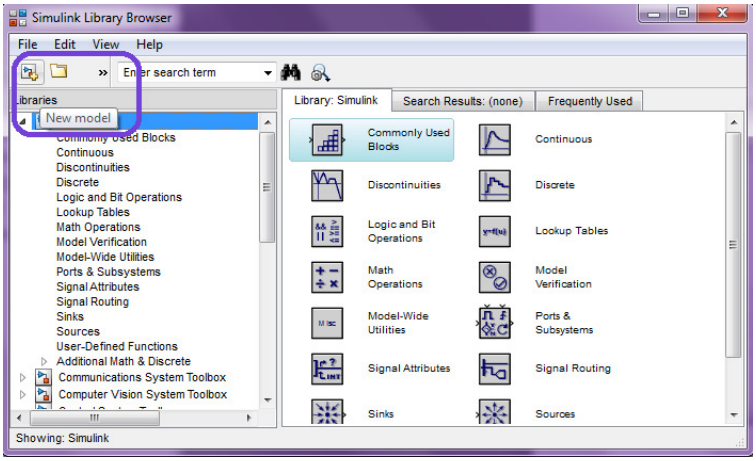

Figure 5: Creating a new model.

From the Fig. 6 you will see that there are a lot of libraries in Simulink, though for this course, the most relevant libraries will be logic and bit operations, ports and subsystems, and sinks and sources (see Fig. 6). You can also use userdefined functions, which allows you to create your own block, though this topic is a bit beyond the scope of this course.

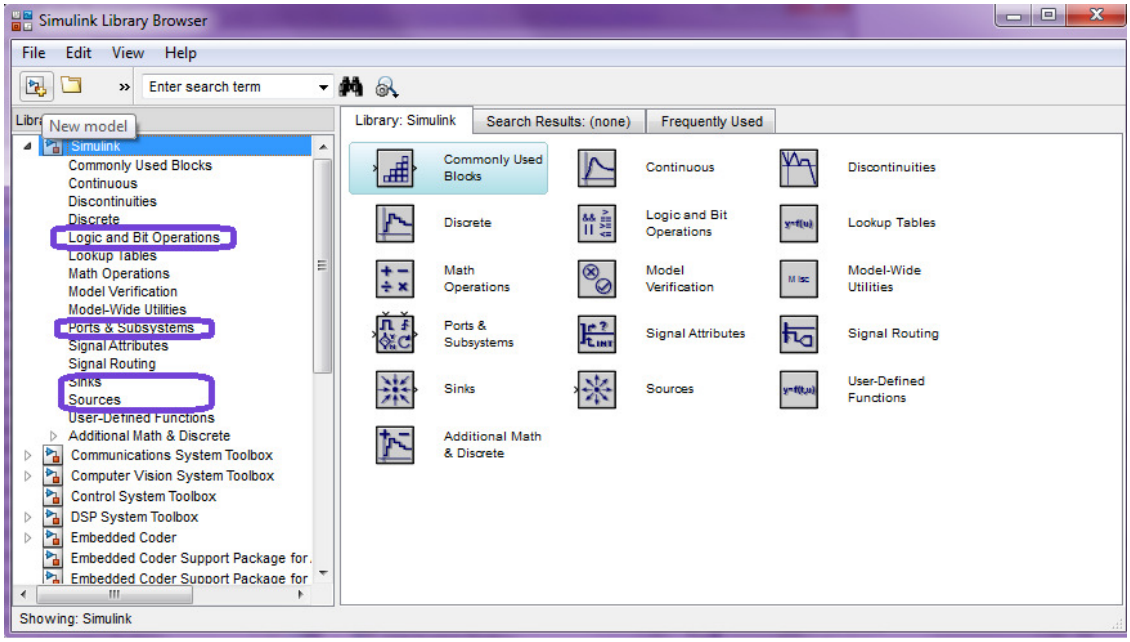

Figure 6: Required libraries

Focusing on the library logic and bit operations, the following components are provided:

- Logical Operator : Perform a specified logical operation on inputs.
- Relational Operator: Perform a specified relational operation on inputs.
- Bit Clear: Set a specified bit of stored integer to zero.
- Bit Set: Set a specified bit of a stored integer to one.
- Bitwise Operator: Perform the specified bitwise operation on the inputs.
- Combinatorial Logic: Implement a truth table.
- Compare To Constant: Determine how a signal compares to specified constant.
- Compare To Zero: Determine how a signal compares to zero.
- Detect Change: Detect a change in a signal value.
- Detect Decrease: Detect a decrease in a signal value.
- Detect Fall Negative: Detect the falling edge when a signal value decreases to a strictly negative value, and its previous value was nonnegative.

- Detect Fall Nonpositive: Detect the falling edge when a signal value decreases to a nonpositive value, and its previous value was positive.
- Detect Increase: Detect an increase in the signal value.
- Detect Rise Nonnegative: Detect a rising edge when a signal value increases to a nonnegative value, and its previous value was negative.
- Detect Rise Positive Detect a rising edge when a signal value increases to a positive value, and its previous value was negative.
- Extract Bits: Output the selection of contiguous bits from an input signal.
- Interval Test: Determine if a signal is in specified interval.
- Interval Test Dynamic: Determine if a signal is in specified interval.
- Shift Arithmetic: Shift bits or binary point of a signal.

Save your model with a meaningful name to make it easier to find the next time you will edit it. To explore the working directories, open your existing model as shown in Fig. 7.

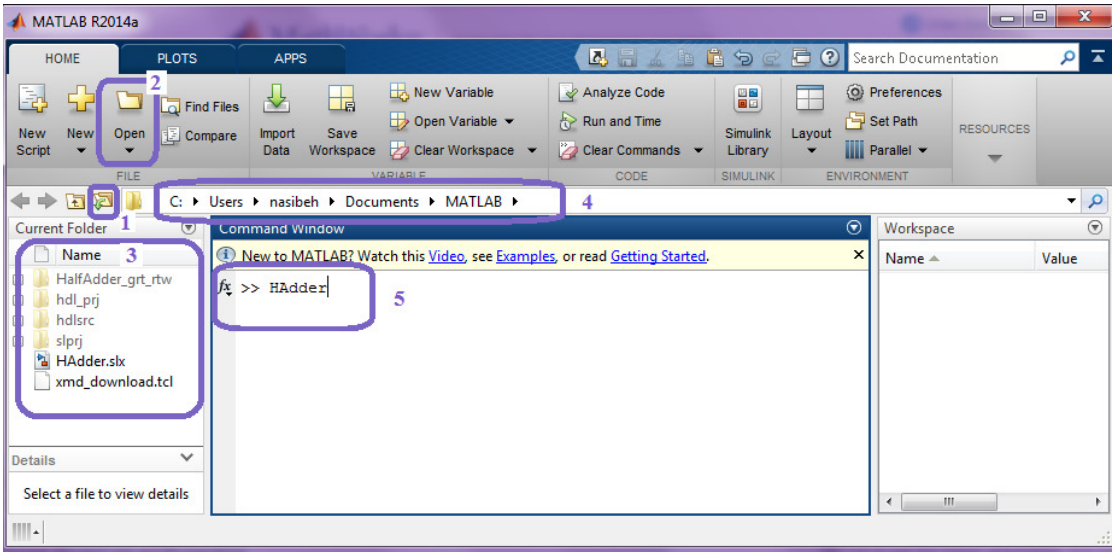

Figure 7: Exploring the directories and opening models

1. Look around in a few of the folders/directories.
2. Take a look at the different models in each folder.
3. View your current folder and existing models and sub-folders with hierarchy.
4. View your current/working folder.
5. Open your existing model using the MatLab command line, provided that the model exists in your current folder

Note5- In the Simulink library browser, you can either click on the icon to open a file, or access it from the toolbar. To use the toolbar, you can select File >Open and in the Open dialog box, select the model that you want to open, then click Open.

Adding Blocks to your Model:

Now that you have some familiarity with the components available in the library logic and bit operations, to design a simple half adder, we need to use a logical operator which provides 7 operations, where the default is an AND. You can change the operation by right clicking on the block parameters (logic) to have different operations, including OR, NAND, NOR, XOR, NXOR, and NOT, as shown in Fig. 8. From the same wizard, you can change the number of inputs as well.

Note6- To refresh your memory, let's have a look of what these logic operators do:

- NOT : The output is FALSE if the input is TRUE, and vice versa
- AND : The output is TRUE if all inputs are TRUE
- OR : The output is FALSE if all inputs are FALSE
- NAND : Performs an AND of the inputs, then inverts the output
- NOR : Performs an OR of the inputs, then inverts the output
- XOR: Performs an OR of the inputs to produce the output, but produces FALSE if all inputs are TRUE
- NXOR: Performs an XOR, then inverts for the output

Note7- After inserting your components, if you double click on the name under the component instance in your model, you can change the name.

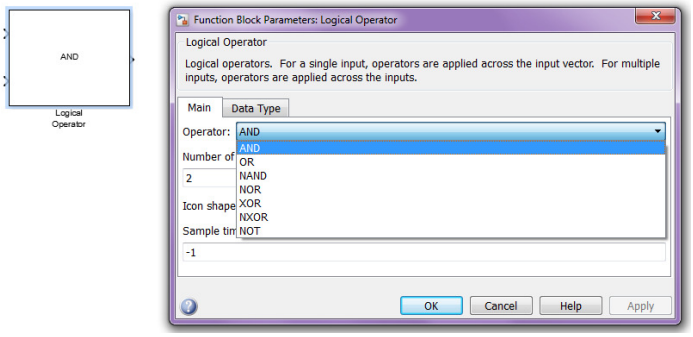

Figure 8: Logical operations

To instantiate input and output, we use the libraries sources and sinks, respectively. For the sake of simplicity, constants can be used for inputs and a display for outputs (but keep in your mind these two cannot be synthesized and embedded in the Zedboard). Again, you can double click on these instances to set the block parameters.

Up to now we have instantiated an XOR and AND, as shown in Fig. 9 - two of the main logic operations required for the design of your half adder.

Note8- Before you connect the blocks in your model, arrange them logically to make the signal connections/lines (you will learn about them next) as straightforward as possible or resize the components as you like.

Connecting the components:

After you have added blocks to your model (for our half-adder example, one XOR and one AND), you need to interconnect them. Simulink connects the blocks with a line and an arrow, indicating the direction of signal flow. The connecting lines represent the signals within your model. Most blocks have angle brackets on one or both sides. The pointer changes to a cross hair (+) while your cursor is over the port. These angle brackets represent input and output ports, as

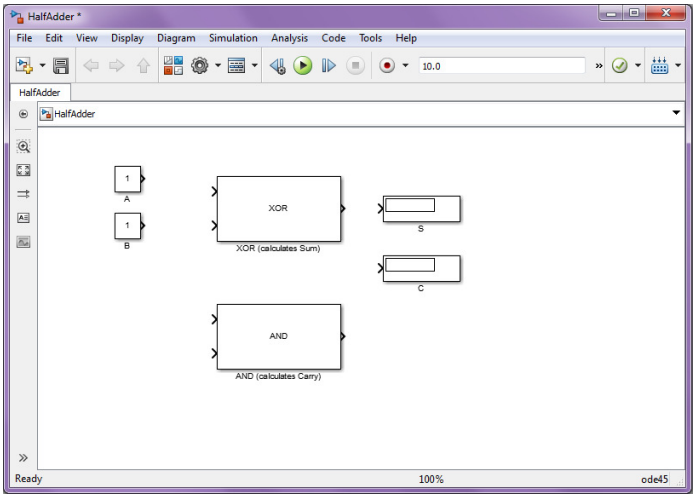

Figure 9: Required logic for a half adder

shown in Fig. 10. Position the cursor over the output port on the right side of the component or over the input port on the left side. The pointer will change to a crosshair (+) while over the port, (as shown in Fig. 11). Connect the components to the appropriate input and output ports.

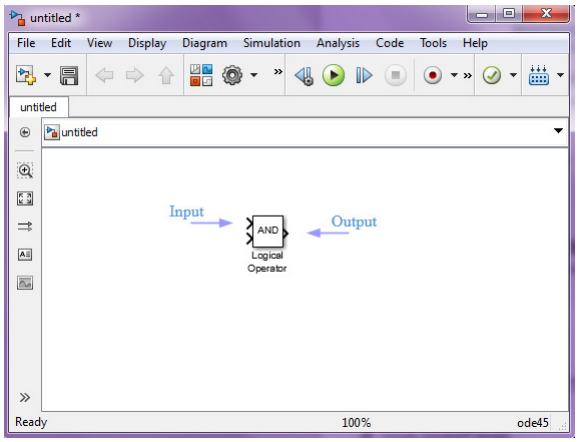

Figure 10: Input and output port type

You can also branch the input lines. When one input is being used by more than one block, we can branch an input line. In our half-adder design, we have A and B, which are being used as inputs to both the XOR and the AND gates. To branch a line, press the line and hold Ctrl. Then start dragging (refer to Fig. 12) and release the mouse button after your are done with branching.

Finally, you will have entered the design shown in Fig. 13.

Note9- you can add an annotation to any unoccupied area of your block diagram, where annotations add textual information, such as your name and information about your model. To create a model annotation, double click in an unoccupied area, then a small rectangle should appear and the pointer changes to a vertical insertion bar. Start typing the annotation content. To start a new line, simply hit the Enter key. Each line is centered within the rectangular box that surrounds the annotation. To close the annotation, click with the mouse any blank area in the window.

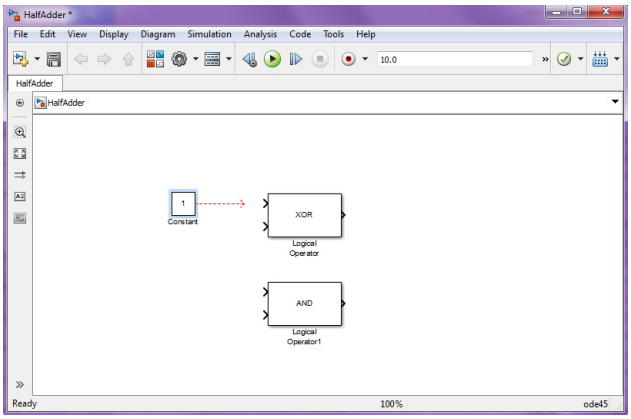

Figure 11: Draging the line

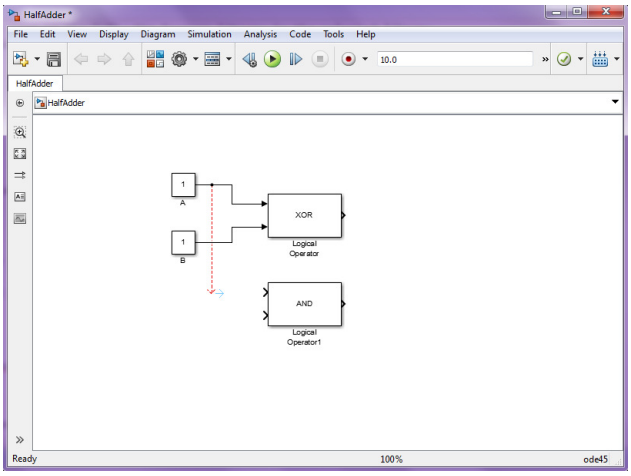

Figure 12: Branch the line by clicking and dragging.

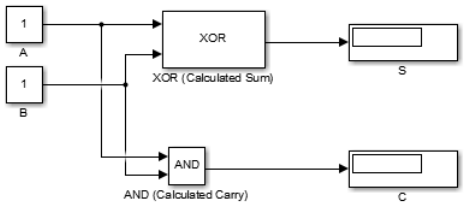

Figure 13: The logic implementation of a half adder.

Note10- To change the font size of an annotation, select it and with the mouse, select Format and then Font. In the pop-up window, select the desired font size and then click OK. The font size will change when the annotation is deselected.

Simulation:

After your model is complete, you can change the value of inputs (change the block parameters by double clicking on the input block) then either hit the Run button in toolbar (see Fig. 14) or select Run from the drop-down menu of Simulation (Simulatation >Run)

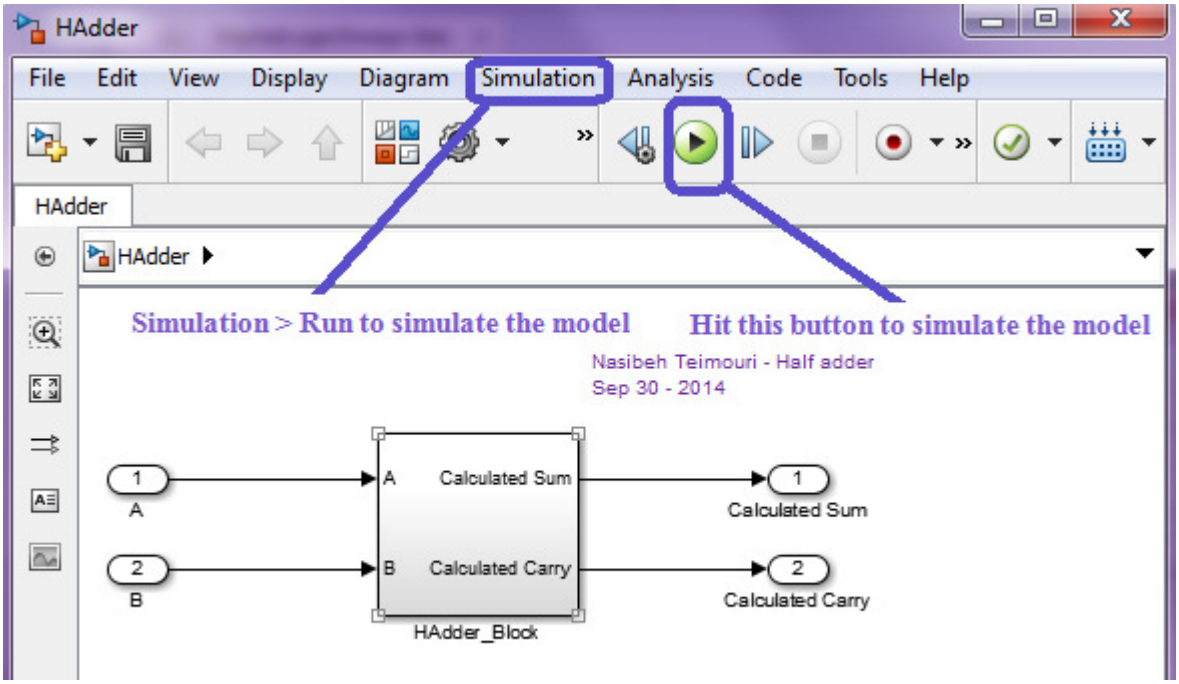

Figure 14: Simulate the model.

To simulate your design, right click on the inputs and set the constant values via block parameters (constants). Then simulate the design for different cases to see how correctly your half adder is working. Fig. 15, Fig. 16, Fig. 17, Fig. 18 show the simulation results reflected on the outputs respectively for 4 different cases: A=0 and B=0, A=0 and B=1, A=1 and B=0, A=1 and B=1 respectively.

Synthesizing your design:

If you want to synthesize your design and download it to the FPGA on the Zynq, since both Constants (the inputs we selected) and Displays (the output we selected) cannot be synthesized, we need to use In and Out elements, as shown in Fig. 19 and Fig. 20.

Note11- When you create an atomic block of your design -as you will see in the following sections- you should not include inputs and outputs, let's call them tester and monitor respectively. You will change the inputs to test your design and you will look at the outputs to monitor how correctly your design works. It means that after creating an atomic block of your design, you will insert the block 'input' from Simulink library for testers and 'output' for monitors. Don't use 'constant' or 'display' because they cannot be synthesized.

To set the properties of the inputs, right click on it and change the data type and inheritance as Fig. 21 shows. For half-adder simply select Boolean.

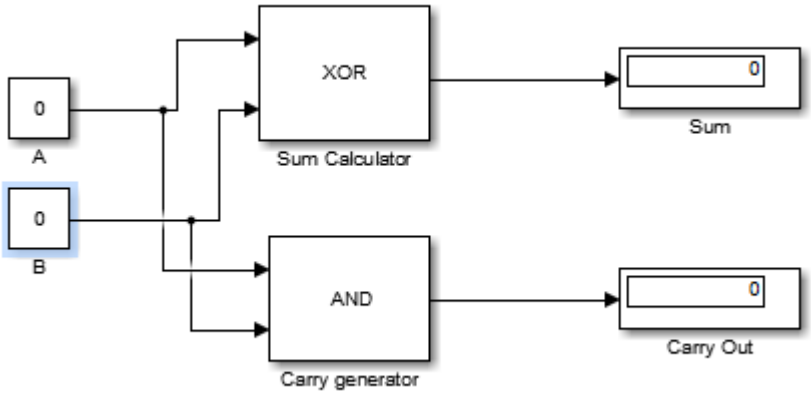

Figure 15: Simulate the model with A=0 and B=0

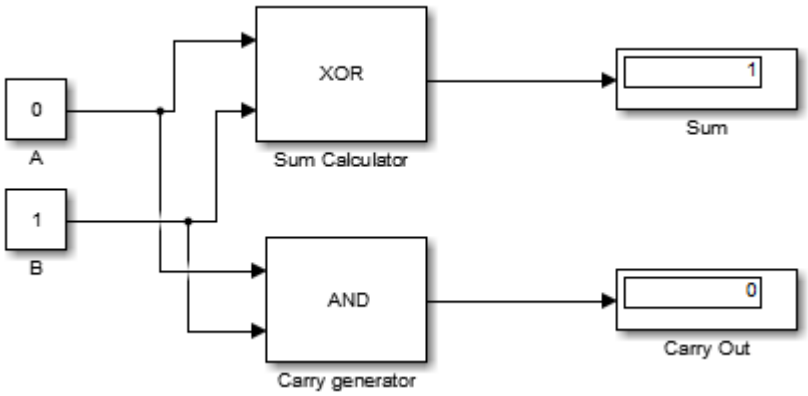

Figure 16: Simulate the model with A=0 and B=1

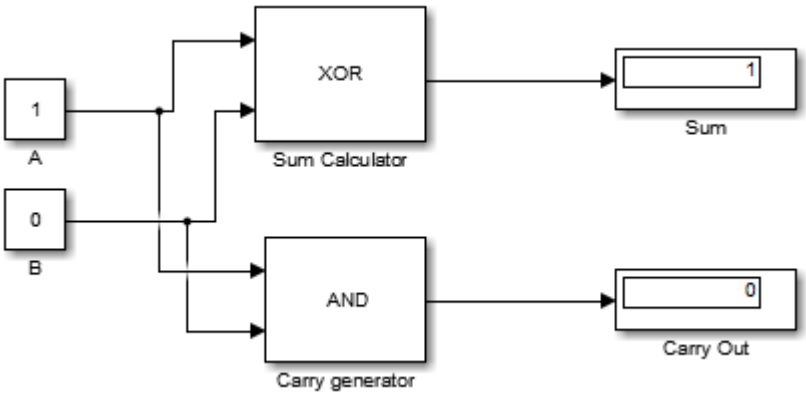

Figure 17: Simulate the model with A=1 and B=0

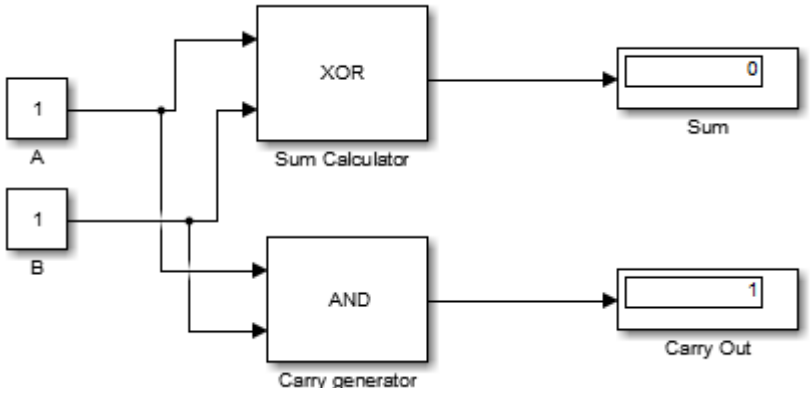

Figure 18: Simulate the model with A=1 and B=1

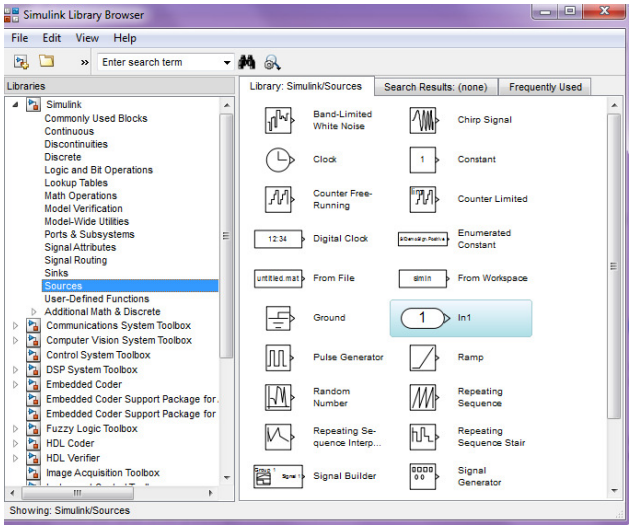

Figure 19: Synthesizable input

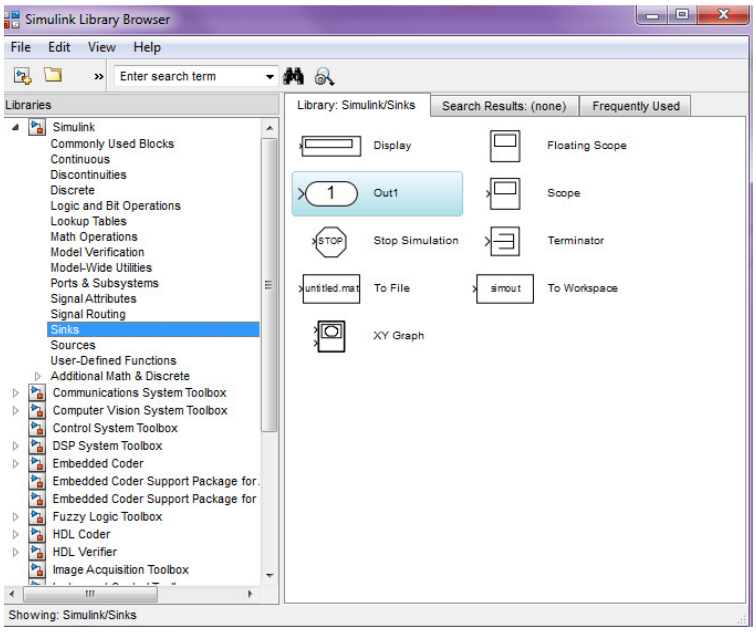

Figure 20: Synthesizable output

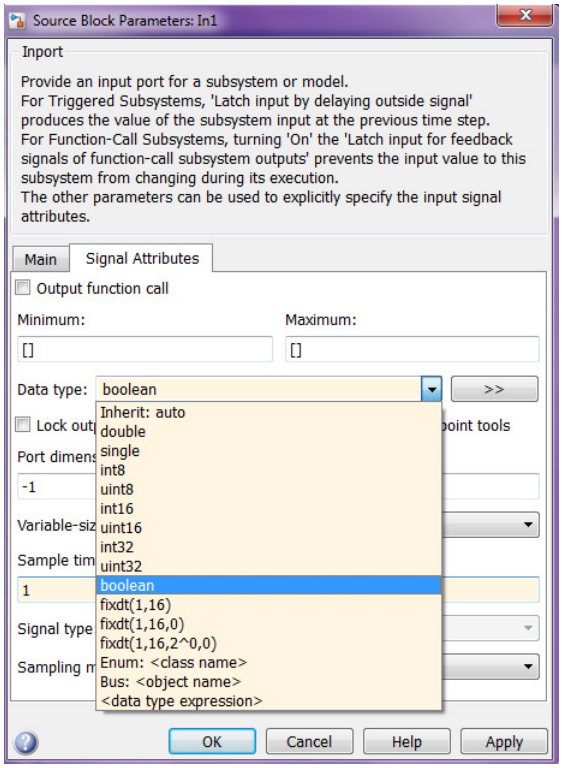

Figure 21: Input data type

2 Digital Design with the Xilinx Zynq using Simulink

This section focuses on using the Zynq-7000 SoC, with your digital design developed and synthesized with Simulink to design your half-adder design. At the end of this chapter, you will be able to run your Simulink examples on the Zynq.

The first part of this section briefly introduces the Zynq SoC and highlights the hardware and software required for programming the Zynq. We then explore how to setup the Zedboard to be used with Simulink. This will include the Zedboard configuration, jumper setup and the connections that will be used. Finally we will generate the HDL code for our half-adder design and download it to the FPGA on the zedboard. For this purpose, there are some videos located here[3] developed by The Mathworks and you can view them.

2.1 Introduction and Requirements

As Fig. 22 shows, the Zynq-7000 SoC is a single device system-on-chip that contains the following units:

1. A dual-core ARM Cortex-A9 based processing system (PS).
2. Xilinx programmable logic (PL).
3. A high-bandwidth interface between PS and PL based on the ARM AMBA AXI specification.

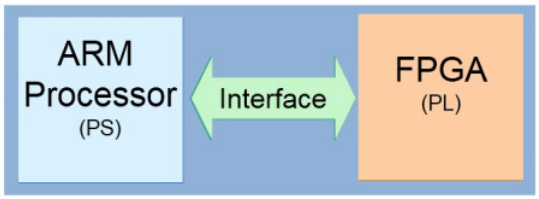

Figure 22: Zynq-7000 SoC

To run a full application on the Zynq, you need to consider the following issues:

- You need to perform algorithm and task partitioning between the PL and the PS, which can be challenging. You need to select what to develop in hardware and what to develop in software, based on a number of factors.
- You will need to decide how you will interface I/O connections on the Zedboard. You have a range of available interfaces, including buttons, LEDs, switches, and many I/O adapters.
- You will develop a program to run on the ARM.
- You will program the FPGA using your Simulink design. The tools will generate the bit stream used for programming the FPGA.

The work flow of an integrated hardware/software design is shown in Fig. 23. You should start with developing an clear understanding of the requirements. Then you will then model your design (input it into Simulink). You will then simulate and validate your design.

Once you complete your design, you will need to figure out which elements you will develop in software on the ARM. You will probably need to modify your design slightly, leaving out those

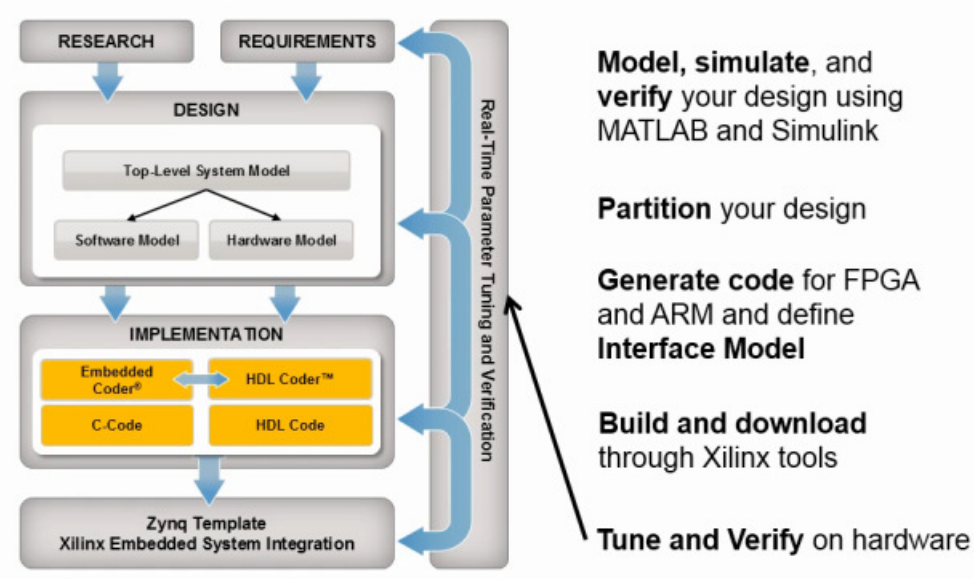

Figure 23: An integrated HW/SW design

elements you will implement in software. Finally, you will use the HDL coder to build the interface model between FPGA and ARM and generate code.

Now that you are familiar with the design steps, it is time to start your implementation. Before starting, you need the following required items:

- A ZedBoard
- The Mathworks Zynq Design Package, which includes MATLAB, the MATLAB coder, Simulink, the Simulink coder, and the HDL Coder. These are all installed on our lab PCs.
- The Xilinx ISE Design Suite, which is already installed on our lab PCs.
- Terminal emulation program (i.e., Mobaxterm) to verify connectivity to your board.

2.2 Zedboard Set Up

To setup the Zedboard for programming, check the jumpers on the Zedboard. The Zedboard has a number of jumpers to control system functions such as boot source and configuration. To setup the Zedboard properly, follow the diagram shown in Fig. 24, making sure that the values of MIO 2,3 and 6 are set to 0, while MIO 4 and 5 are set to 3.3v. MIO 1 should be set to 2.5 v.

As Fig. 25 shows, there are three connections between the Zedboard and computer: 1) UART port, 2) JTAG and 3) Ethernet port. The UART is for talking to the OS running on the Zedboard, the JTAG is for programming the FPGA, and the Ethernet is for connecting to the processor.

Note1- during this course, you will not use UART.

Next, we will use the JTAG to program the FGPA and the Ethernet to program the ARM (as you have been doing earlier in this class). Before moving on to the next part, in your MatLab window, open ISE Design Suite 14.4. You should find the shortcut on your Desktop or you can find it here: C:\Xilinx\14.4\ISE_DS\ISE\bin\nt64\ise.exe

Set the ISE synthesis tool path with the following command in the MatLab command line:

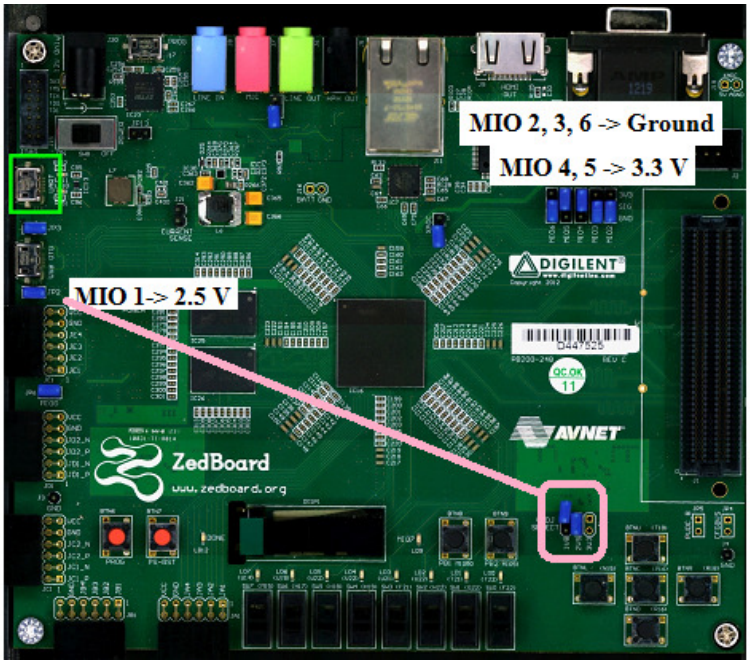

Figure 24: Jumper settings

hdlsetuptoolpath('ToolName', 'Xilinx ISE', 'ToolPath',
'C:\Xilinx\14.4\ISE_DS\ISE\bin\nt64\ise.exe');
Where the forth argument is the path to the tool ISE.

2.3 Code Generation and Deployment

The next step is to create an atomic block of your design, which is going to be executed on FPGA (PL). You will compile it to make sure that there are no syntax errors. You will use the Work Flow Advisor to carry out a number of steps. Finally you will successfully program the FPGA.

Note2-The contents of an Atomic Subsystem block are all computed in together, where the content of the block includes three element: 1- Compute the block's outputs, 2- Compute the block's states, 3- Computes the time for the next time step

Note3- Before creating your block of the the half adder shown in Fig. 26, right click on the inputs to change the sample time from -1 (to avoid inheritance from previous block because there is no block before) to 1 and set the data type as Fig. 21 shows.

Creating an atomic subsystem block:

Decide which parts of your design will be implemented on the programmable logic, and which parts will be run on the ARM processor. The portion of the design which is going to be mapped to the PL should be atomic in order for its contents to be computed as a unit (look at Note2 to refresh your memory about an atomic block).

Note4- The atomic subsystem is the boundary of your hardware/software partition. For our example -the half-adder design- we will map the XOR and the AND to the PL and the remaining blocks, including inputs and outputs, to the PS.

To map the XOR and the AND to the PL, select both of them by dragging a bounding box that encloses them. Then from toolbar, select diagram >Subsystems & Model Reference >Create Subsystem from selection. Fig. 27 shows the block you have created, including input and outputs.

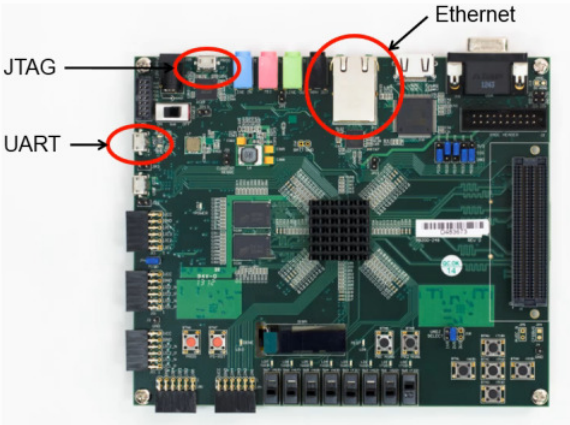

Figure 25: UART, JTAG and Ethernet Cable

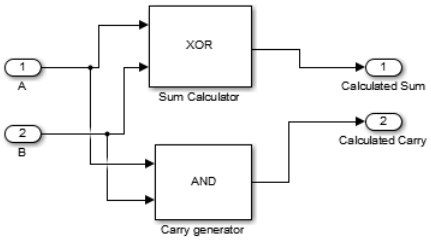

Figure 26: A half adder with a synthesizable input and output.

To make the subsystem/created block atomic, right click on the block, then select the block parameters (subsystems) dialog box. Check "treat as atomic" as shown in Fig. 28. Also, we need to set a sample time (by default, it value is inherited from previous block). Change the sample time from -1 to 1.

Note5- Here you could use the Ctrl and A keys to select the whole design, because you did not have any other operational blocks other than the XOR and the AND blocks to map them on the FPGA.

Note6- You can resize the block or re-name.

To make sure that HDL code generation does not find errors in your design, compile it before going ahead. To compile the design, From tool bar, select Code >HDL code >Generate HDL code as shown in Fig. 29.

In case of successful compilation, you will see the message, "HDL code generation complete" in your MatLab window.

*Note7-*In case you receive errors, the error messages should guide you to identify which part has a problem.

Now everything is ready for you to go through the ISE (Integrated Software Environment) steps to program the FPGA on the Zedboard. For this purpose, you will use the Work Flow Advisor in Simulink.

The rest of the steps from now, to the end of this tutorial, are done through invoking different steps of the Work Flow Advisor to build your hardware, software and integrate them together.

Note8- The Xilinx ISE Design Suite is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize/compile their designs, perform timing

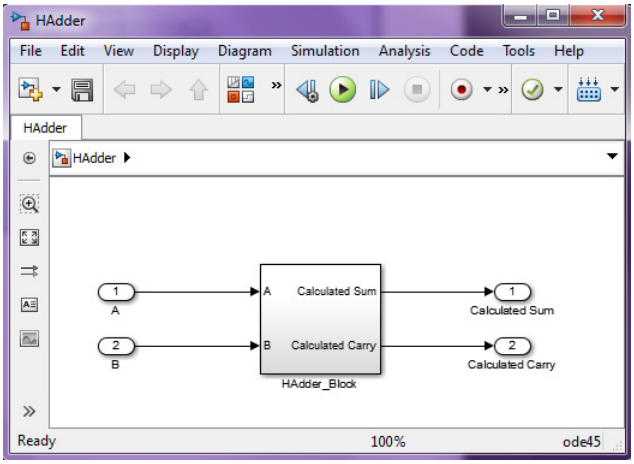

Figure 27: Generated block

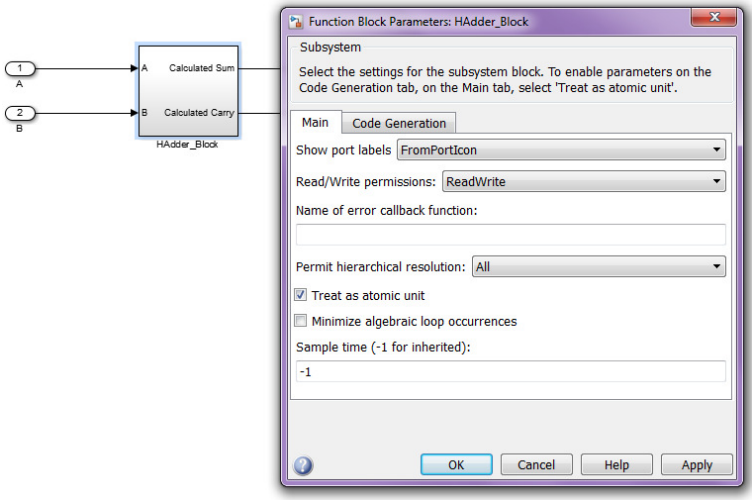

Figure 28: Making the block atomic and setting the sample time

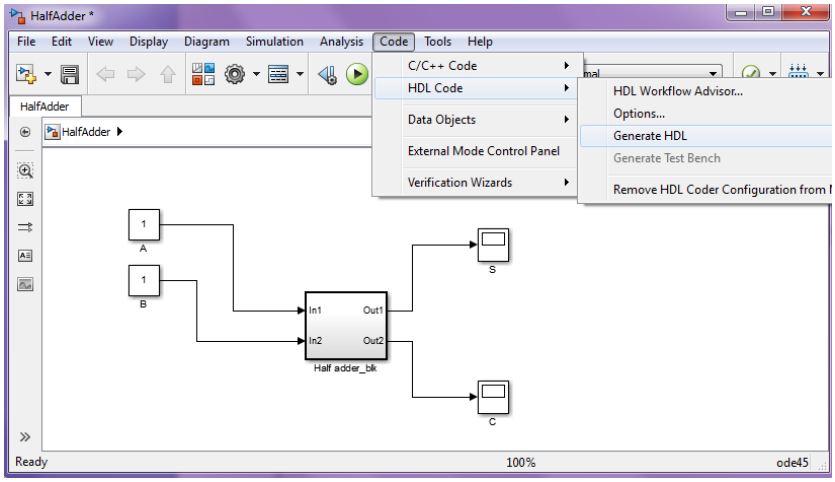

Figure 29: Compiling the HDL code

analysis, examine RTL (Register transfer level) diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

Calling Work Flow Advisor:

To start the Advisor, from the toolbar select Code >HDL code >HDL Work Flow Advisor, then you should see the hierarchy window, as shown in Fig. 30. Select the atomic block that you created and click on OK.

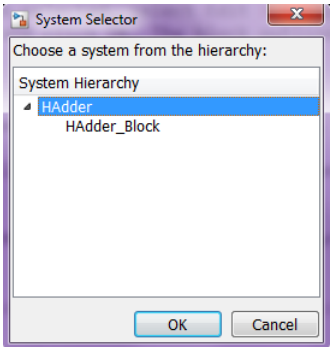

Figure 30: Setting the hierarchy

After that, you will see the Advisor as Fig. 31.

There are 4 steps – we will go through each one in detail.

1. ***Set Target:***

The first step is to run Set Target. Click on it. We need to set the Target Work Flow of the IP core generation and the target platform to be the Zedboard (1.1 Set Target Device and Synthesis tool), and then wait for the system to load the files, as shown in Fig. 32.

Note9- Using the IP Core Generation workflow in the HDL Workflow Advisor, you are able to automatically generate a sharable and reusable IP core module from a Simulink model. The generated IP core is designed to be connected to an embedded processor on an FPGA device. The

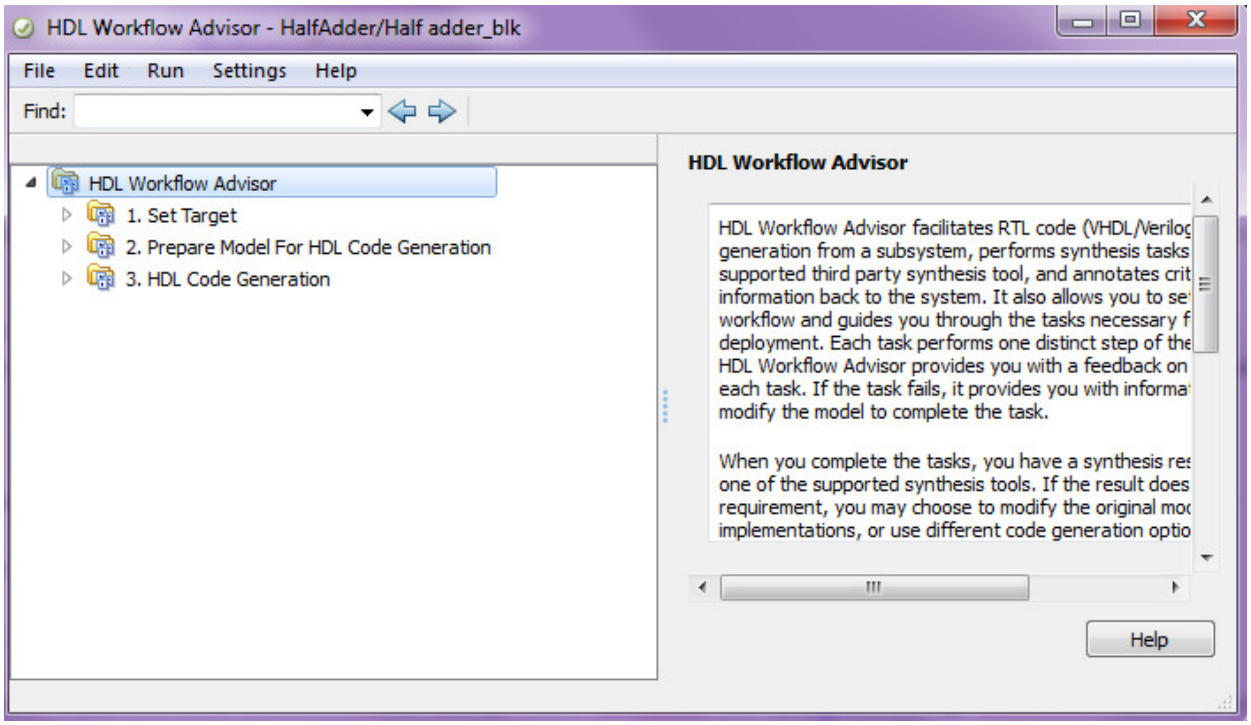

Figure 31: Work flow Advisor

HDL Coder generates HDL code from the Simulink blocks, and also generates HDL code for the AXI interface logic connecting the IP core to the embedded processor. The HDL Coder packages store all generated files in an IP core folder. You can then integrate the generated IP core with a larger FPGA embedded design in the Xilinx EDK environment.

Note10- EDK (Embedded Development Kit) is a suite of tools and Instruction Processor (IP) that you can use to design a complete embedded processor system for implementation in a Xilinx FPGA device.

Run this task and wait for the tools to acknowledge success.

Note11- After you *Run* each task, you will observe the red progress bar as Fig. 33, be patient for it to be done.

Next, click on step 1.2 (Set Target Interface) to map each port in your half-adder block to one of the IP core target interfaces. You will see the inputs and outputs. For now, select the Push button for input and the LED general purpose for output, as shown inFig. 34.

Run task and wait for successful response.

Note12- There are several options for inputs and outputs, during the lab assignments you will get familiar with them. Fig. 35 shows the options.

2. Prepare Model for HDL Code generation:

This step has 4 parts. The first part is 2.1 - we need to check global settings. If you **Run** the task, you will typically see an error such as the one shown in Fig. 36. This errors means your model's inputs do not have the expected simulation setting.

To resolve this issue, click on Modify all. ISE will change/configure the model to use the recommended simulation settings. Again, **Run** the task, and it should pass successfully. The three remaining parts involve checking for algebraic loops, block compatibility and sample times. They should not encounter any problems. Run these one-by-one.

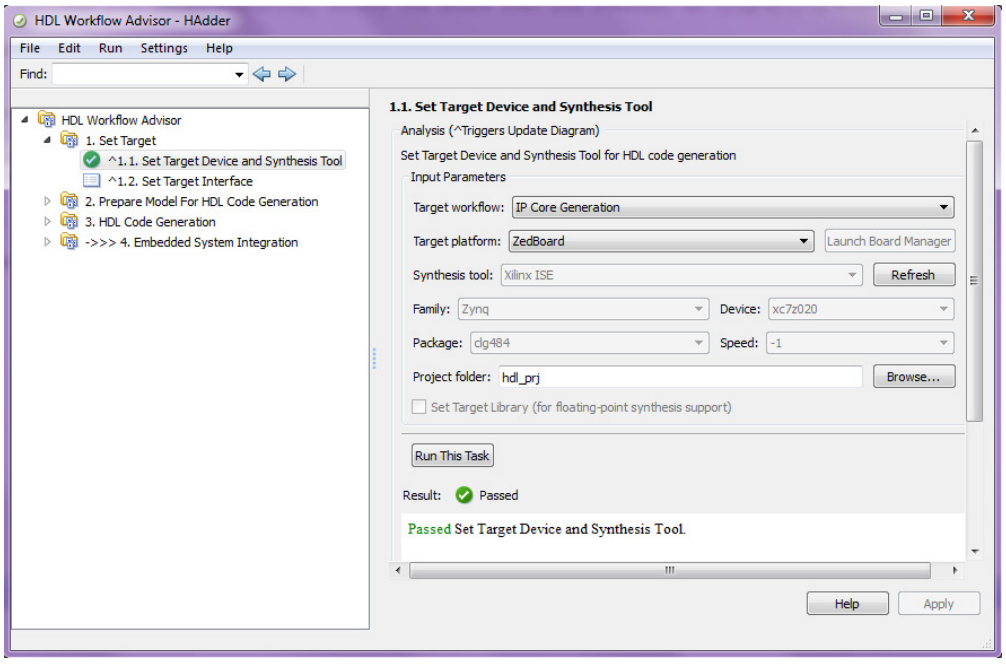

Figure 32: 1.1 Set Target device and synthesis tool

Figure 33: Progress bar during running each work flow task

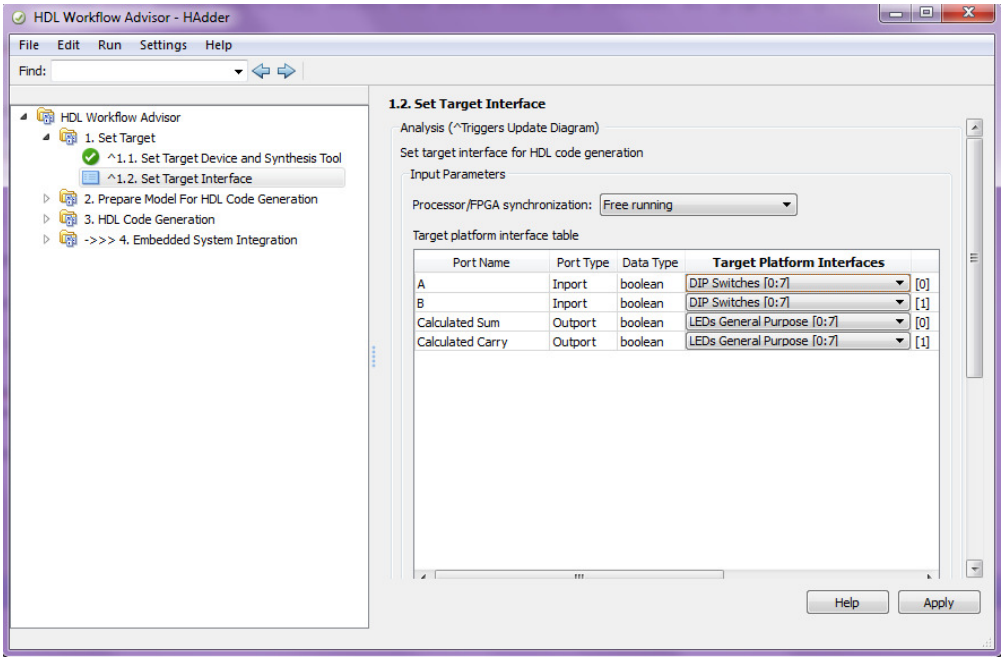

Figure 34: 1.2 Set Target Interface

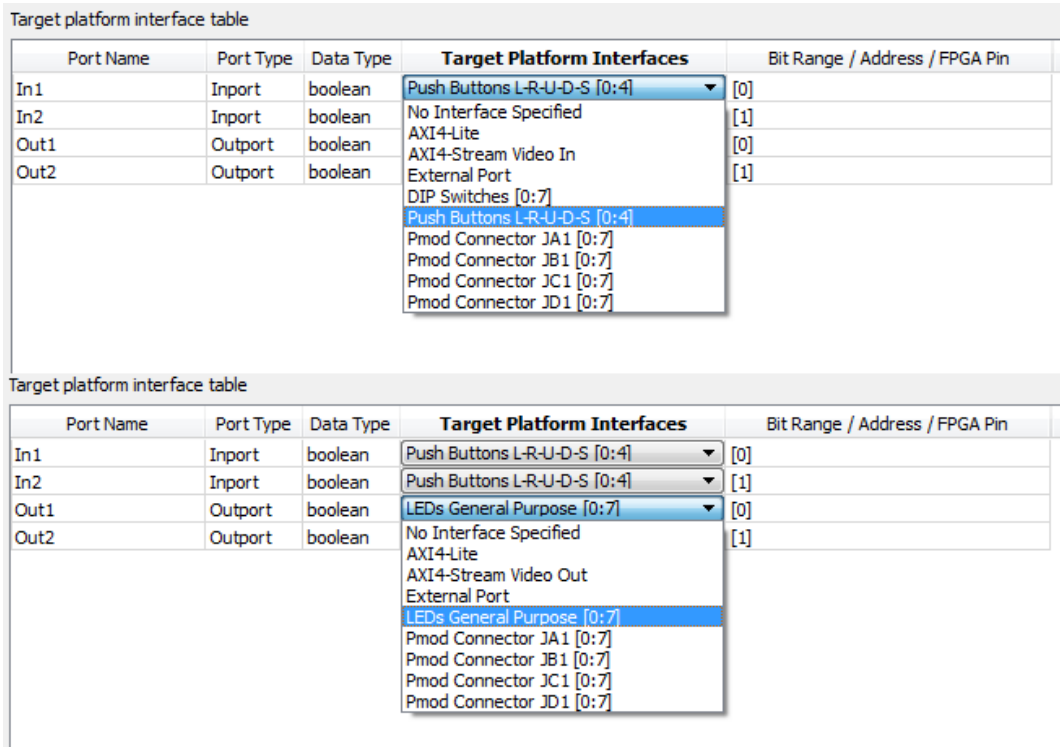

Figure 35: Input/output interface options

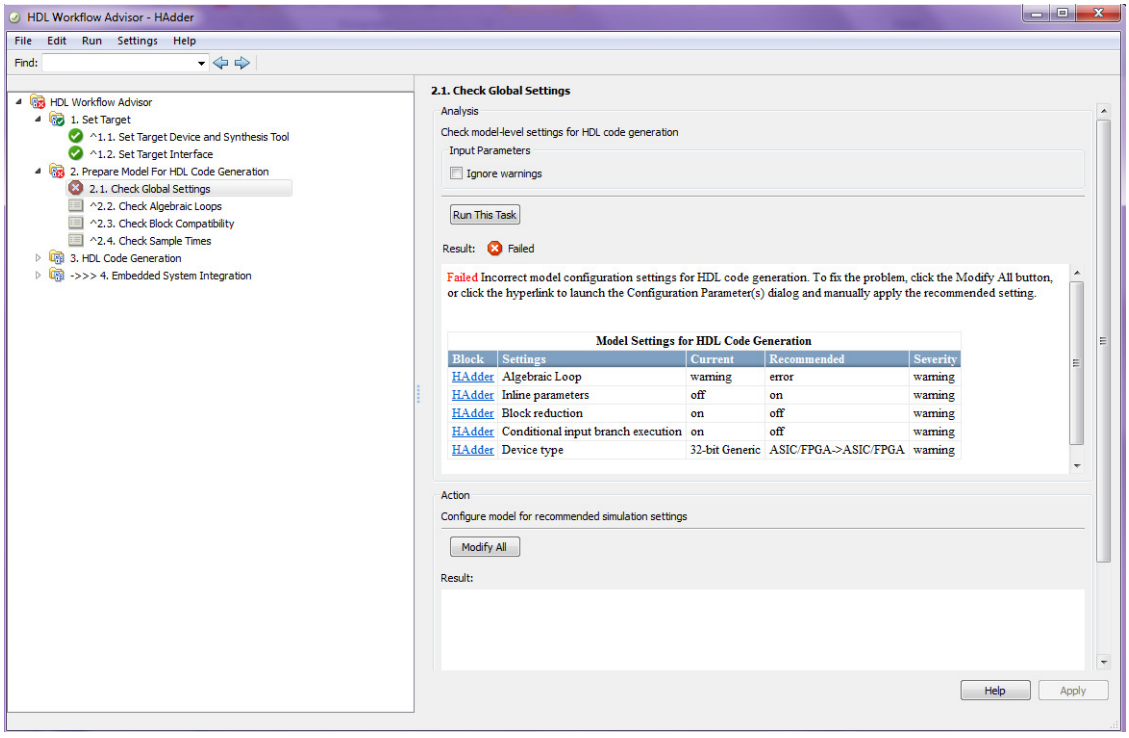

Figure 36: 2.1 error in checking global settings

Note13- Basically, algebraic loops occur when an input port that has a direct feed-through path is driven by the output of the same block, either directly, or by a feedback path through other blocks which have direct feed-through. Algebraic loops are difficult to solve mathematically. Simulink does have algebraic loop solvers that will attempt to resolve this situation iteratively and give the correct answer, which in a lot of cases it does. One instance of an algebraic loop is shown in Fig. 37. In this case, there is a pathway through the model where all of the blocks include direct feedthrough.

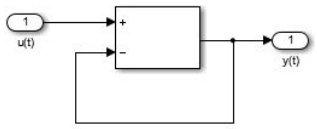

Figure 37: One instance of an algebric loop.

y(t) = u(t) - y(t)

As you can see, the block cannot simply resolve its output as it needs to compute it. Simulink detects instances such as these (i.e., algebraic loops) and resolves them using an iterative loop. Simulink will compute the correct result in the above example, which is:

y(t) = u(t) / 2

Part 2.2 checks for algebraic loops in your design to resolve it.

Note14- Some Simulink blocks impose restrictions on the data type of the signals they can handle. Part 2.3 checks for block compatibility to make sure that the connected blocks are compatible with each other in view of the signal types they passed to each other.

Note15- The sample time of a block is a parameter that indicates when, during simulation, the block produces outputs, and if appropriate, updates its internal state. Part 2.4 considers this sample time.

Note16- Instead of selectively hitting **Run** for each step, you can **Run all**. However whenever the advisor faces an error it will stop, and after correcting the problem, you need to re-run.

3. HDL Code generation: There are two parts for HDL Code generation: part 3.1 (Set Code Generation) and part 3.2 (Generate RTL Code and IP core). Run these parts and at the end of part 3.2, you will see the HTML custom report Fig. 38 displayed. Now the IP core files are in the ipcore folder within your project folder. The HTML custom IP core report is generated together with the custom IP core. The report describes the behavior and contents of the generated custom IP core.

Close the HTML custom report. You have done well to get to this point and now your hardware (IP core) is generated.

4. Embedded System integration:

In this step of the workflow, you will insert your generated IP core into a embedded system reference design, generate an FPGA bitstream, and download the bitstream to the Zynq hardware and finally you will be done.

Note17- Xilinx offers EDK (Embedded Development Kit) as a suite of tools and IP that you can be used to design a complete embedded processor system for implementation in a Xilinx FPGA device. It contains all the elements that the Xilinx software needs to deploy your design on the Zynq platform, except for the custom IP core that you have generated and the embedded software that you will generate.

In part 4.1 you will create the project and in part 4.2 you generate the software model. **Run** the first part, and after successfully project creation, **Run** part 4.2. You will have some errors, you

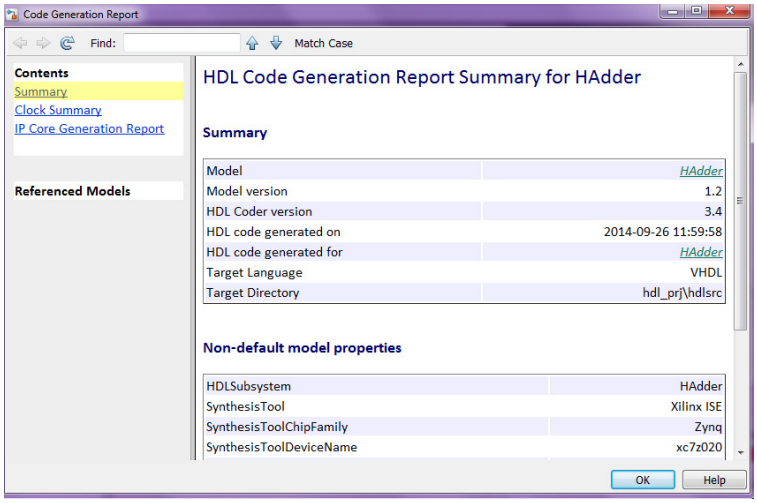

Figure 38: 3.2 Code generation report

do not need to resolve it, check skip this task and again hit **Run**, it will go on successfully.

In part 4.3, the FPGA bit stream is built which can take a long time (many minutes). When the bit stream is built, the terminal should look like Fig. 39.

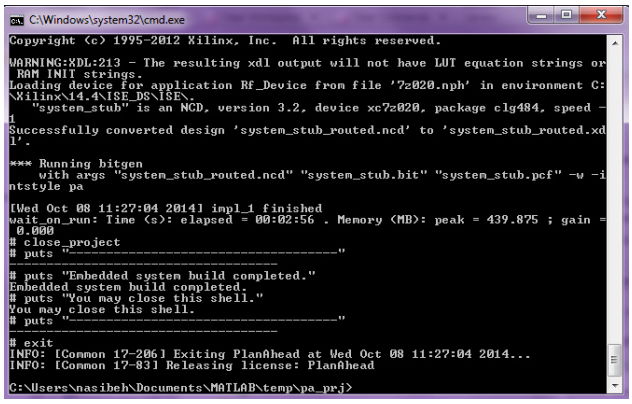

Figure 39: 4.3 FPGA bitstream generation

And after compilation of the bitstream, close the window, and go to the part 4.4 to program target device. After you hit **Run**, your FPGA is being programmed.

Test programmed zynq:

The FPGA is programmed to receive left and centered button as input A and input B and reflect Sum and Carry out on LED 1 and 2 respectively. Fig. 40 shows the case A is 1 (left button is hit) and B is 0 (right button is not hit).

Fig. 41 shows the case A is 0 (left button is not hit) and B is 1 (right button is hit).

Fig. 42 shows the case A is 1 (left button is hit) and B is 1 (right button is hit).

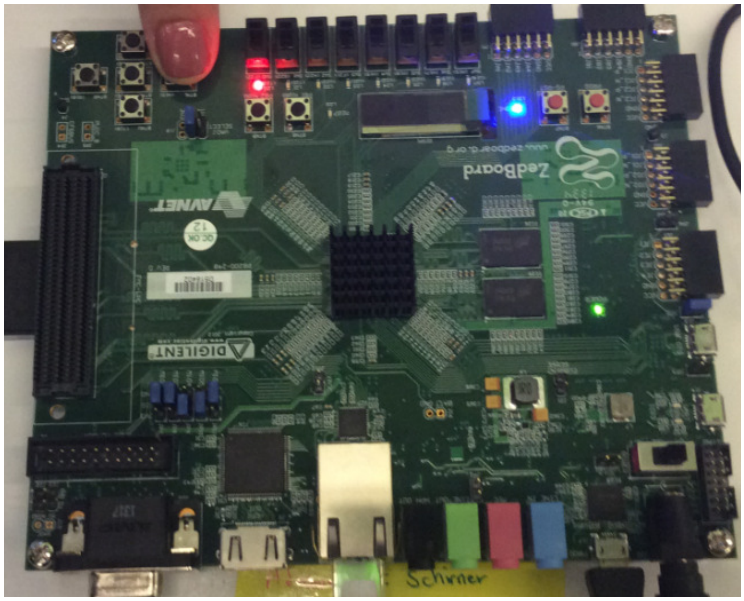

Figure 40: Testing programmed zynq with half adder and A=1 and B=0

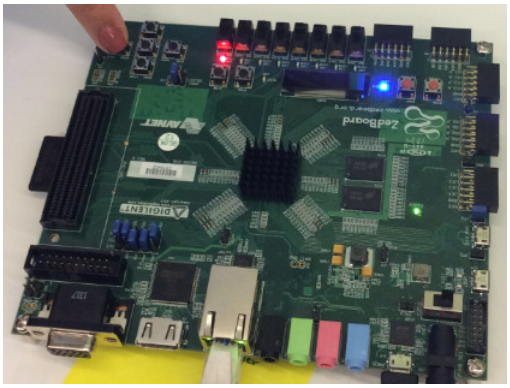

Figure 41: Testing programmed zynq with half adder and A=0 and B=1

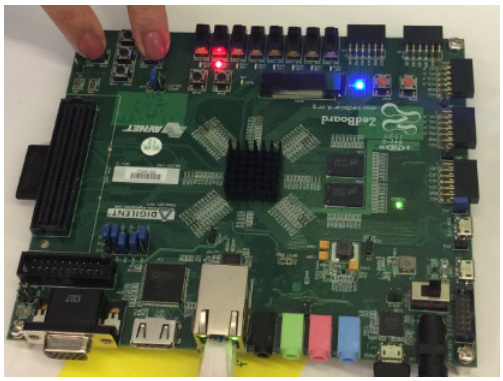

Figure 42: Testing programmed zynq with half adder and A=1 and B=1

References

[1] Mathworks. Mathworks company. `http://www.mathworks.com`.

[2] Mathworks. Simulinkgetting started guide. `http://www.mathworks.com/help/pdf_doc/simulink/sl_gs.pdf`.

[3] Mathworks. Zynq design with simulink. `http://www.mathworks.com/programs/xilinx-zynq-design-with-simulink.html`.

...Special thanks to ***Professor Kaeli*** *and* ***Professor Schirner*** *for their revising the manual and to* ***Amir Momeni*** *for his help programming ZYNQ.*