DIGITAL LOGIC CIRCUITS

1. AIMS & OBJECTIVES

The aim of this laboratory is to examine the behavior of simple logic circuits and to become familiar with the design flow for a Field Programmable Gate Array (FPGA) using current Electronic Design Automation (EDA) tools.

A binary combination lock will be designed, simulated and implemented on a Xilinx FPGA. Xilinx is the name of the company which supplies the FPGAs that you will use. This will involve the following steps:

i. Entering your design with a schematic editor.

ii. Verifying the functionality of your design with a simulator.

iii. Implementing your design for a Xilinx FPGA.

iv. Downloading and testing the design using a demonstration board.

All laboratory work must be detailed in your logbook. Make sure that you fully understand all of the steps in the design process. Do not just press the buttons then move onto the next step!

ASSESSMENT IS BY A FORMAL TECHNICAL REPORT

Details of this are included. Check your laboratory timetable for the submission deadline.

2. FPGAs

An FPGA is an electronic device that can be programmed or reprogrammed by the user to implement a required logic function. Both combinational and sequential circuits can be implemented. Each device contains an array of identical Configurable Logic Blocks (CLB) as shown in Fig 2-1. Metal lines run horizontally and vertically in the channels between the logic blocks. These can be used to make the required connections between the logic blocks or to connect to the Input and Output blocks of the circuit. These I/O blocks connect to the physical pins of the device package.

An FPGA is customized by programming it with configuration data. In Xilinx devices, the configuration data is loaded into memory cells which control switches that configure the logic blocks and connect selected wires between the blocks.

A big advantage of FPGAs is the ability to produce a large prototype logic design on your desktop in a relatively short time. The FPGA used in this lab is a Xilinx Spartan3 device mounted on a demonstration board.

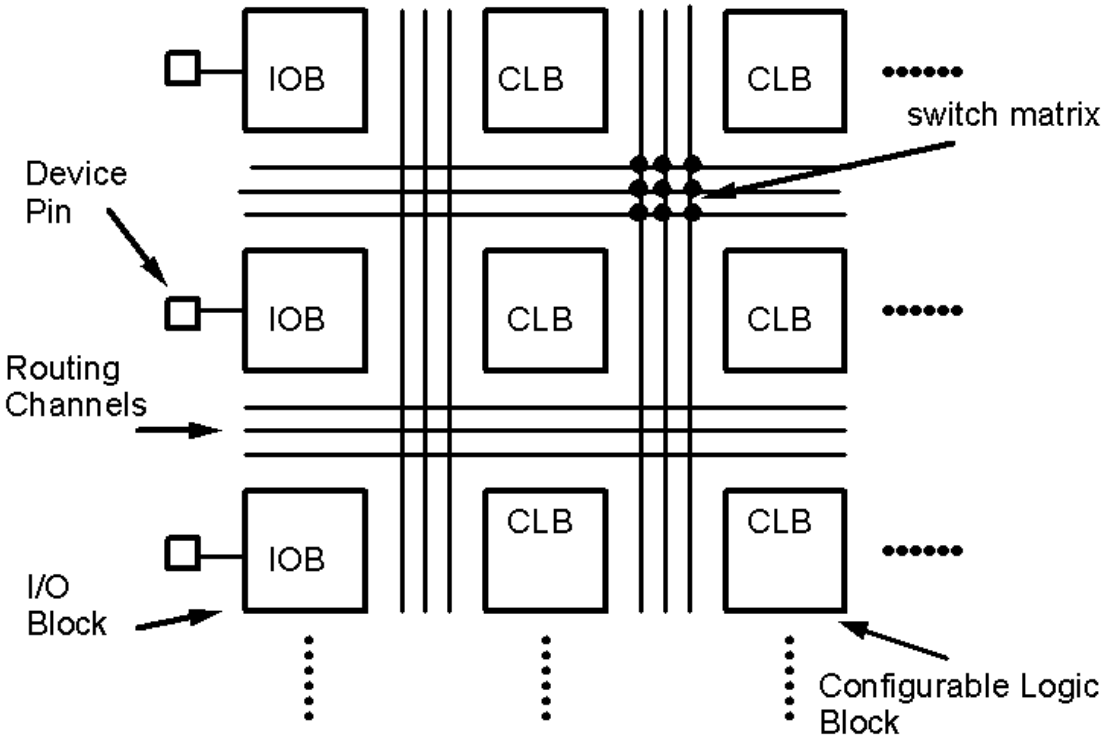


Figure 2-1. FPGA logic and routing resources.

3. DESIGN ENTRY

3.1. Schematic Capture

A design is entered into the EDA system using a schematic editor. This is a drawing package that creates an electrical drawing of the components and their connections. The components used are represented by graphical symbols consisting of a body and pins as shown in Fig. 3-1. The symbol body may be used to represent the logical function of the component. The symbol pins are the connection points used to connect components together.

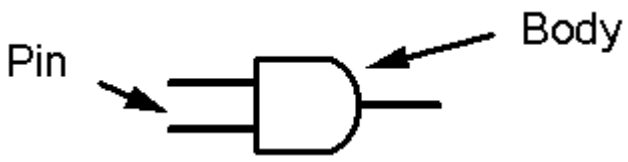


Figure 3-1. Graphical representation of component.

Schematic capture lets us enter digital circuits in a form that we are familiar with and can easily understand.

3.2. Physical and Logical Connections

There are two ways to connect symbols on a schematic.

(i)A physical connection is a wire drawn on the schematic between two or more pins.

(ii)A logical connection is made without drawing the connecting wire. The connection is made by assigning identical names to two or more wires connected to pins.

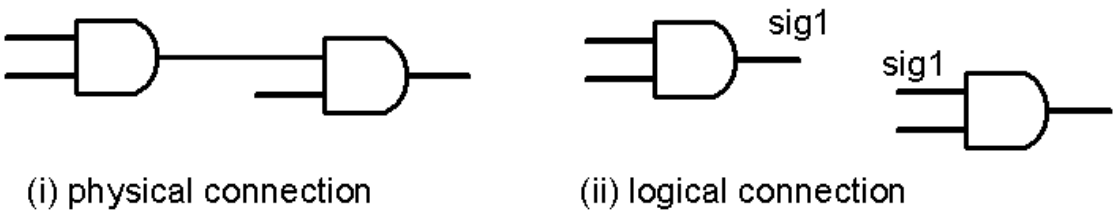


Figure 3-2. Component connections

3.3. Large Designs

Designs may require several schematic pages. Connections across drawing pages must be of the logical type. However, for large designs it is generally better to use a hierarchical structure.

Hierarchical designs are based on dividing a design into smaller sections (macros) that are easier to manage. These macros are single page schematics that can be converted into symbols and saved in a design library. See Fig. 3-3. These macro symbols can then be used in other schematics and connected to other symbols. The same schematic macro symbol can be used several times. Each occurrence of the symbol is known as an 'instance'. The higher level schematic takes less space and is usually clearer to the user because it uses higher level blocks to build complex designs.

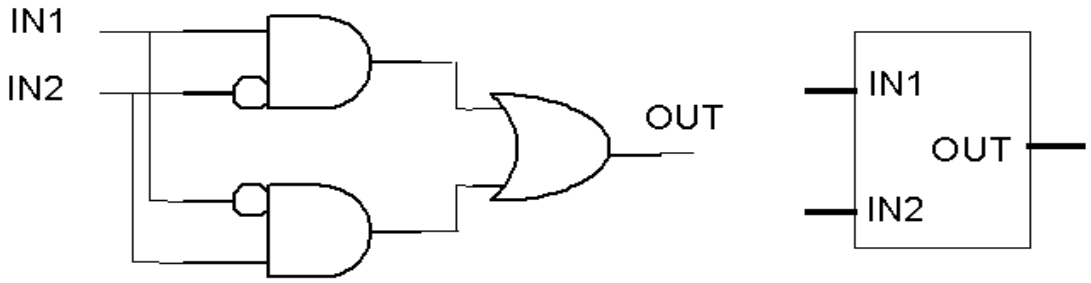


Figure 3-3. Symbolic representation.

4. DESIGN VERIFICATION

4.1. Simulation

Simulation is one of the most important steps in the design process. This is true in many areas of engineering. Design simulation is used to verify that your design is correct before committing it to a physical realization. Software models are used to describe the behavior of the components used in your design.

Test patterns are written and applied to the design inputs. The inputs and outputs at various points in the schematic can then be observed with a waveform viewer. If any mistakes are found, the design must be corrected and the process repeated until the design is correct.

4.2. Functional Simulation

Functional simulation tests the logic in your design to determine if it works properly. It is important to thoroughly simulate each macro in your design before you create a schematic symbol for it and commit it to a library for use elsewhere.

At this stage, the design has not been implemented, i.e. the logic blocks have not been configured and the wires in the routing channels have not been defined. This means that there is no timing information available and the simulator tests the logic in your design with 'unit' delays.

4.3. Timing Simulation

When the design has been implemented and the circuit propagation delays are known, a timing simulation is performed. This is used to verify that your design runs at the required speed.

5. DESIGN IMPLEMENTATION

The schematic design is first flattened into a netlist. This is a text file containing the components and their connections. This stage basically expands all the internal details of the hierarchy. The software then places the various logic blocks and i/o blocks that make up your design in optimal positions inside the device, so a minimum amount of routing is necessary to connect them together. The design is then routed to make the electrical connections between logic blocks. The routing algorithms are designed to determine the interconnect paths that lead to the minimum delays. A bit stream is then generated. This is the configuration information needed to program the FPGA and it is generated from the placed and routed design and can be used to program the device.

Design1 - Two Bit Adder

The following exercise is intended to give you a guided introduction to the EDA tools. Before you attempt it, make sure that you are familiar with binary adders and registers from your EEE119 Digital Systems notes. Ensure that you understand each step that you take and the reason for doing it. Do not just blindly follow the guide.

The first design is a 2-bit synchronous adder that can take two 2-bit inputs A1,A0 and B1,B0 and add them together. The 2-bit sum S1, S0 and the carry will be stored in a register by the application of a single clock pulse.

Example1:

A = 10 (2) (A1 = 1, A0 = 0)
B = 01 (1) (B1 = 0, B0 = 1)

A + B = 11 (3) (carry = 0, S1 = 1, S0 = 1)

Example2:

A = 11 (3)
B = 10 (2)

A + B = 101 (5) (carry = 1, S1 = 0, S0 = 1)

A top-down, hierarchical design methodology will be used. The first step is to produce a top-level diagram as shown in Figure 1. This requires a 2-bit adder and a register to store the result.

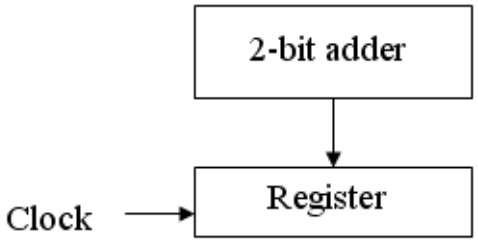


Figure 1. A 2-bit synchronous adder

Going further down the hierarchy, the adder itself consists of two full adders connected together with the carry-out of the first feeding the carry-in of the second.

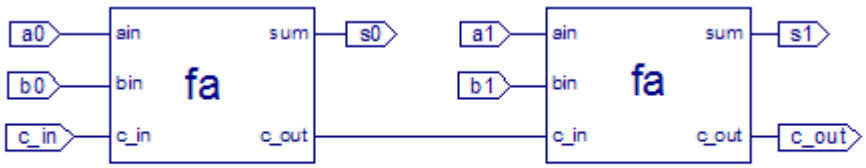


Figure 2. Full Adders

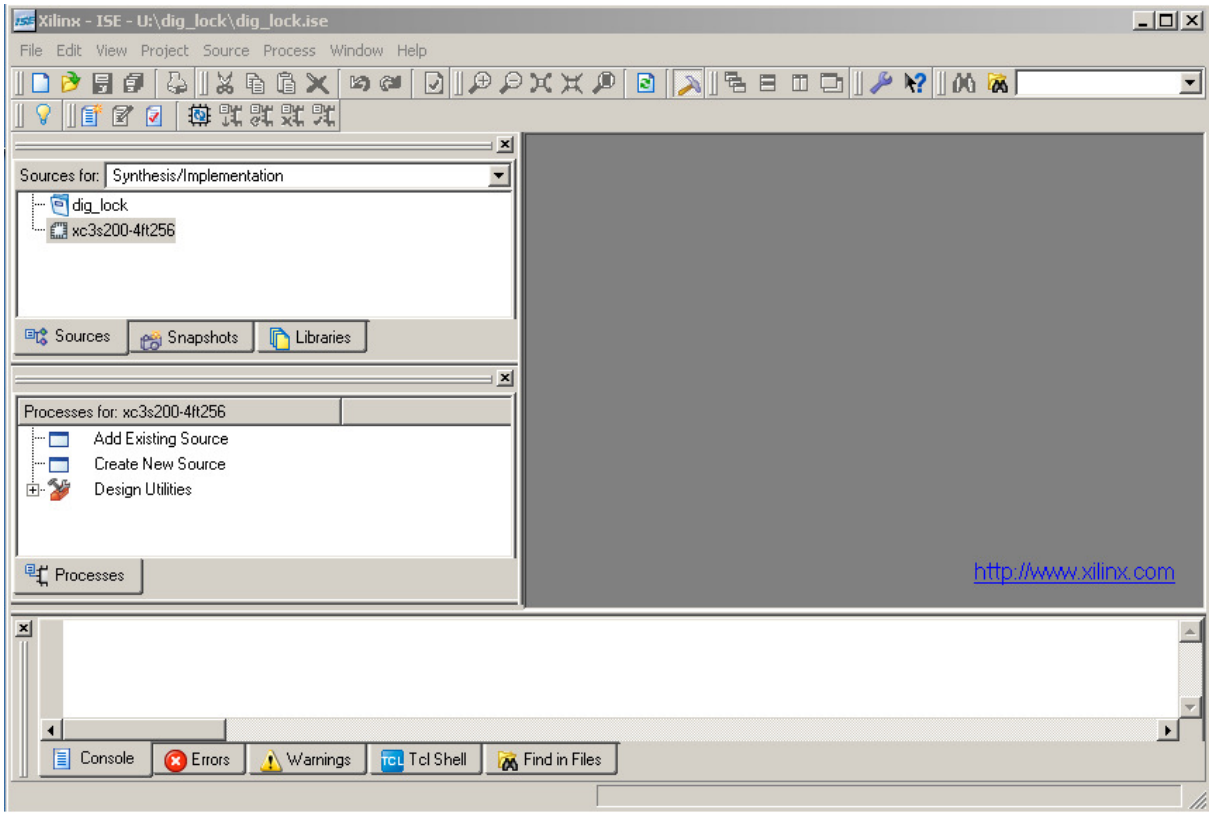
Each full adder is built from basic logic gates. This is the bottom of the hierarchy and the starting point for schematic capture. The register consists of D-Type flip-flops, and this is the bottom of the hierarchy for the register.

1. Starting the Xilinx ISE software.

Log on to a P.C. (**Log on to: VIE**) with your Micro-Unit user name and password. The drive **U:** will then be assigned as your user directory. Your work will be stored on the Micro-Unit server and backed up each night.

KEEP YOUR PASSWORD SAFE – you will need it for both afternoons!

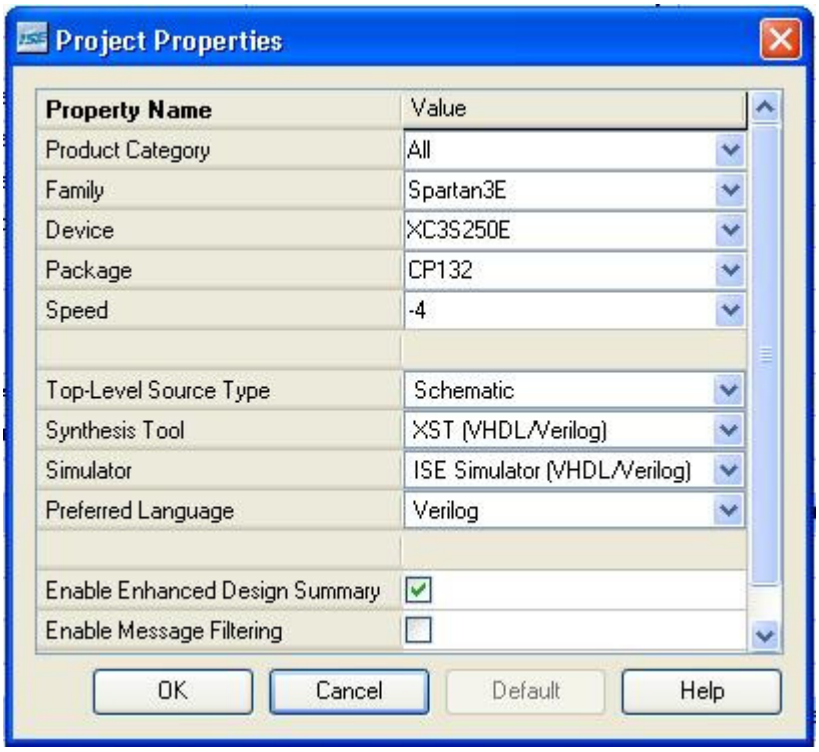
Xilinx ISE consists of many tools in a single framework. The user interface to these tools and the source files that they manipulate is called the **Project Navigator**. Locate the **Project Navigator** icon on the desktop and double click on it to start ISE. Alternatively use the Start Menus to locate the Xilinx ISE 9.2i software.



Files that you create will appear in the **Sources** window. The files appearing will depend upon the setting of **Sources for:** at the top of the window. In the diagram it is set to **Synthesis/Implementation** and only files relating to these activities will appear. When it is set to **Behavioral Simulation**, files related to this activity will appear.

When a file is selected in the **Sources** window, all of the processes that could be applied to that file will be displayed in the **Processes** window. The **Console** area at the bottom shows your interactions with the system and the area above and right is the workspace for the available design tools.

2. Project Management.
Create a new project using **File > New Project**. Check that the project location is set to **U:** and then type **myadder** (all one word) in the project name box. The top level source type is **schematic**. Fill in the device/flow form as shown below. (Carefully check the details or you will have problems later)



Click next until you can click finish.

3. Schematic Capture.
The first module required for your design is the full adder shown in Figure 3. Open a source file for this schematic by: **Project > New Source**. From the list of source type options, select **Schematic** and name the file **my_fa**. Accept Next and Finish to open the schematic editor.

The toolbar shows the editor functions available. Let the cursor hover over the icons to see their function. They are selected with a mouse click, which will put you into the 'mode' of that function. To return to select mode, press escape.

Components can be selected from the symbol menu on the left hand side. Place the required gate symbols on your schematic and use the add wire symbol from the toolbar to connect them together as shown. To draw wires, click on **Add Wire** in the tool bar. Wires can be drawn by selecting a start point then holding down the mouse button and dragging the wire to the desired location. Adding **I/O Markers** (from the tool bar) will enable a symbol to be created for this part. The markers are dropped onto the wire ends and should automatically connect. The inputs are named ain, bin, c_in with corresponding outputs sum, c_out. These are the signal names that will appear on the macro symbol. Double click on the default name supplied to change to the required names.

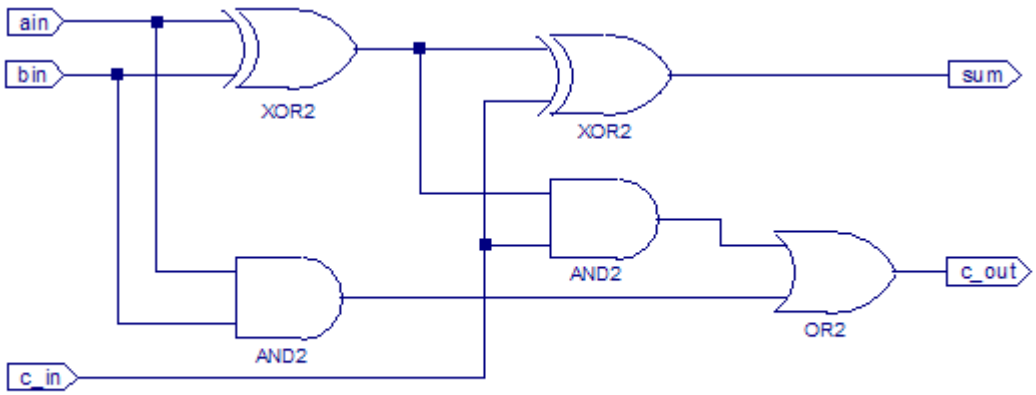


Figure 3. Full Adder Logic Gates

It is now necessary to simulate the design to ensure correct functionality before it is used as a component at a higher level. Save the file and close it.

4. Simulation.

In order to simulate the model of the full adder, it is first necessary to create a file that can drive the inputs to different values. This is known as a testbench. Open a source file for this stimulus file by: **Project > New Source**. From the list of source entry options, select **Verilog Test Fixture** and name the file **fa_tb**. Click Next, then associate the file with **my_fa**.

A Verilog test bench template will appear. Remove the line 'ifdef auto_init and change 'endif to only say end. Add the stimulus as follows:

```
initial begin
        ain = 0; bin = 0; c_in = 0;
  #10   ain = 0; bin = 0; c_in = 1;
  #10   ain = 0; bin = 1; c_in = 0;
  #10   ain = 0; bin = 1; c_in = 1;
  #10   ain = 1; bin = 0; c_in = 0;
  #10   ain = 1; bin = 0; c_in = 1;
  #10   ain = 1; bin = 1; c_in = 0;
  #10   ain = 1; bin = 1; c_in = 1;
  end
endmodule
```

This defines each input to be '0' initially. Signal changes are then given at the times they should occur, each 10 time units after the previous one. The patterns shown cover all possible input combinations ($2^3 = 8$). This is an exhaustive simulation and will thoroughly check the functionality of the full adder.

Save this file, and then close the window. Set the **Sources for:** window to display sources for **Behavioral Simulation**. Select the testbench file then in the '**Processes**' window, expand the **Xilinx ISE Simulator** toolbox. Double click on Check Syntax and when this is successful

double click on **Simulate Behavioural Model** to run the simulator. Use the magnifying glass in the toolbar to zoom out and view the full waveform, then check the results.

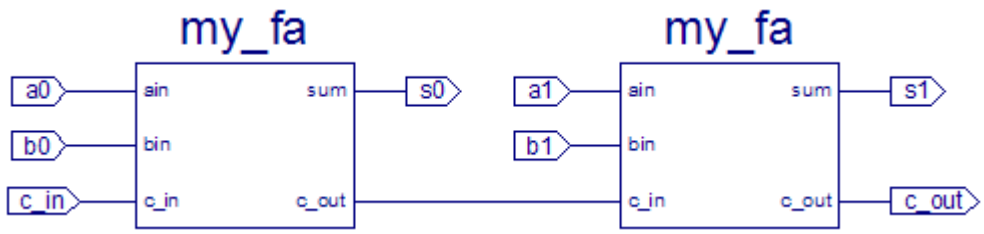
5. Symbol Creation

When a design has been satisfactorily simulated, and you have a high level of confidence that it will function as desired, a symbol can be created for it so that it can be used in higher levels of the design hierarchy.

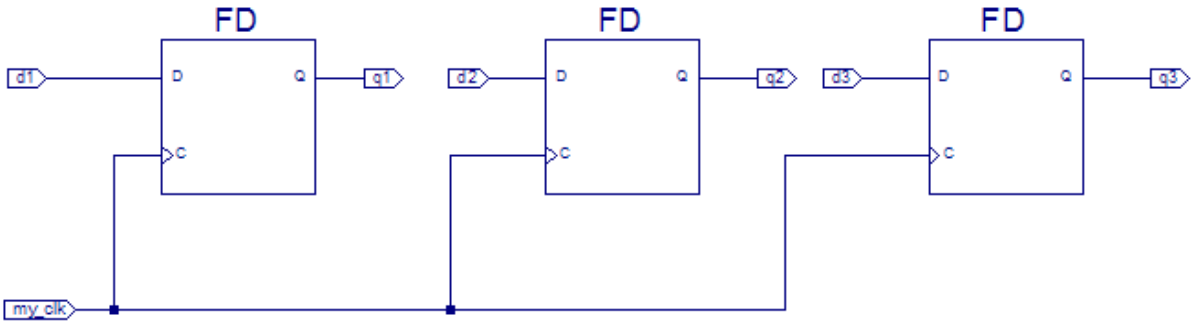
Return to **Sources for: Synthesis/Implementation** and select the source for my_fa. Open the Design Utilities toolbox and double click on **Create Schematic Symbol**. The symbol created will be stored in your own library which will appear in the **Categories** section of the **Symbols** window.

6. Hierarchical Design

Two of these full adder units can be used to form the required two bit adder. Open a new schematic source page called my_fa2 and using your full adder macros, complete the 2 bit adder as shown. Save this and create a symbol for it.



As this is a synchronous adder, a register is required to store the three output bits. This can be created by using three D-Type flip flops (symbol FD). Open a new schematic source page called my_reg and enter the schematic as shown. The flip-flop symbol is FD. Inputs are d1, d2, d3, my_clk and outputs are q1, q2, q3.



Simulate this register to verify that it functions correctly. Check that for all combinations of the d inputs, the output does not change when no clock is applied. Clock three different three bit values into the register.

Hint: (i) cut and paste the previous test data into your new testbench and change the signal names appropriately. Run the simulation for my_clk = 0, then my_clk = 1. (ii) Add stimulus values for my_clk which give a rising edge at the data values you want to store i.e. if the clock is currently 0, #5 my_clk = 1; #5 my_clk = 0)

Create a symbol for the register.

The top-level schematic can now be entered. Call the source file sync_adder. The initial carry input can be set to logic ‘0’ by wiring it to a ground (gnd) symbol. The inputs should be named a0, a1, b0, b1, my_clk and the outputs s0, s1, c_out.

As you have already thoroughly simulated the basic components of this design, your top level simulation is basically to confirm that you have connected the parts together correctly and that they function as a whole. To verify this, simulate the top-level with enough test-patterns to ensure that all connections in the top-level circuit are toggled (change from ‘0’ to ‘1’ or ‘1’ to ‘0’).

7. Synthesis.
Set the sources window to display sources for **Synthesis/Implementation**. Select you top level schematic (sync_adder). In the processes window, double click **Synthesize**. If it is successful you can move onto the next stage otherwise you will have to fix any errors until the process can complete.

8. Assigning Package Pins.
Select the top level schematic and then in the user constraints toolbox, double click on **Assign Package Pins**. Click **Yes** to add an implementation constraints file to the project. In the Loc column, assign the package pins as follows.

a0 > **P11**, a1 > **L3**, b0 > **K3**, b1 > **B4**, my_clk > **G12**,
s0 > **M5**, s1 > **M11**, c_out > **P7**

You can check the pin numbers as they appear against the resources that they connect to on the board. The four inputs are the right-most slide switches, the clock is operated by pushing button BTN0 and the outputs are the right-most LEDs.

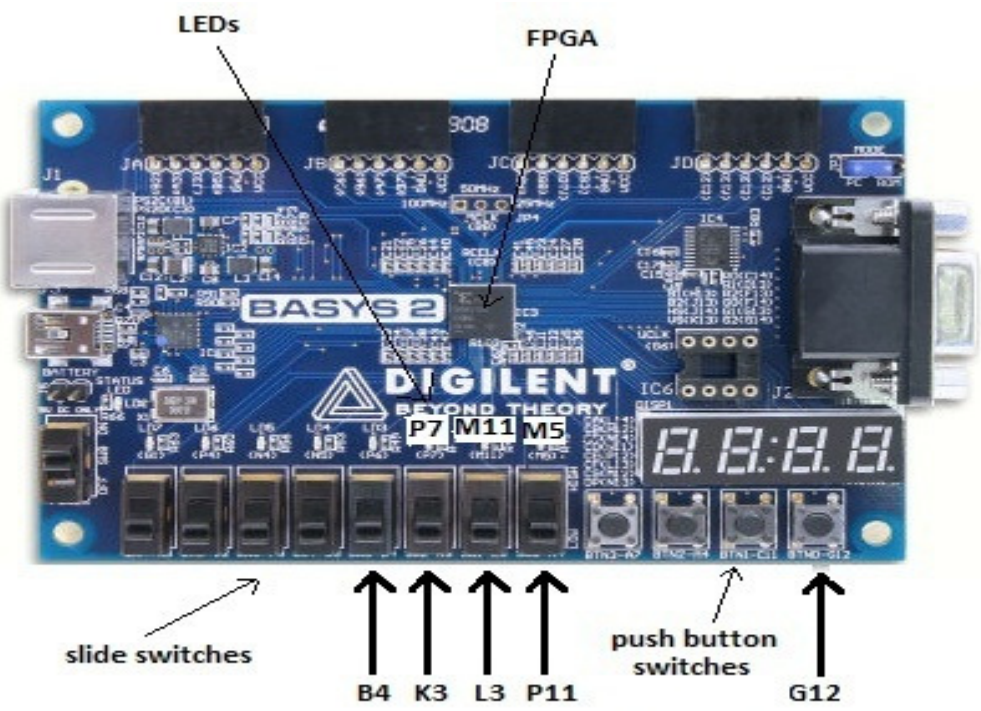


Figure 4. Spartan 3E Demonstration Board

Save (accept XST default) and exit. If you need to make changes to the pin assignment, double click on the ucf file in the sources window (expand sync_adder to find it).

9. Implementation
In the project navigator select the top level source file and double click **Implement Design**. This will translate your design into the required logic format for a Xilinx FPGA.

10. Generate Programming File
In the project navigator select the top level source file and double click **Generate Programming File**. This will produce the actual bitstream used to program the FPGA. It will have the same name as your top level design but with the extension .bit.

11. Device Programming.

Connect a BASYS2 board to the PC using the USB connector. Locate the ADEPT programming tool and start it up. (From the desktop or the pull-down menus) Use the Browser to locate and download your bitfile. You can ignore the clock warning message, just click yes.

12. Hardware Testing
Test you adder using the same sequences that you used in your top level simulation.

Design2 – Digital Lock

Design and implement a binary combinational lock. The lock must have a three bit input (*d1*, *d2*, *d3*) used to enter 3-bit codes and a single bit input (*sysclk*) used to program the lock with the desired access code. A single bit output (*valid*) should go 'high' for a valid input (i.e. an input that exactly matches the code programmed into the lock). The design will be demonstrated on the Spartan-3 Board.

OPERATION OF THE LOCK

The lock has two modes of operation, (i) programming and (ii) normal operation.

(i) The lock operator decides upon a 3-bit code that will allow users to gain access. This code is placed on the *d* inputs, say **101**. A clock pulse is applied to store this value. Only the operator has access to the clock button. The code **101** is now stored in the lock and the *d* inputs are set to a random value. **101** is the only value that will open the lock (send *valid* 'high').

(ii) In normal operation, users will attempt to gain access by trying out various codes on the *d* input switches. The *valid* output should go 'high' when a correct match is applied, in this case **101**. For all other *d* input combinations, *valid* must remain 'low'.

The slide switches SW0, SW1, SW2 will be shared to enter the initial access code and user inputs:

SW01 > *d1*, SW1 > *d2*, SW2 > *d3*

The push button BTN0 will be used to clock in the chosen access code:

BTN0 > *sysclk*

The LD0 will light up when the current input to the lock matches the desired access code.

LD0 > *valid*

Look on a demonstration board to find the required pin numbers to enter in the LOC column of the pin assignment tool.

Start by drawing a top level system diagram of your proposed solution. Explain your solution to a demonstrator. When satisfactory it can then be entered into the EDA system as a new project called digital_lock, implemented and tested on the demonstration board. To solve the problem it will be necessary to revise the operation of basic logic gates, especially the uses of XOR gates and registers. Remember to simulate all parts of your design!

N.B Do not use reserved words for signal names e.g. in, out, clock

Design Methodology

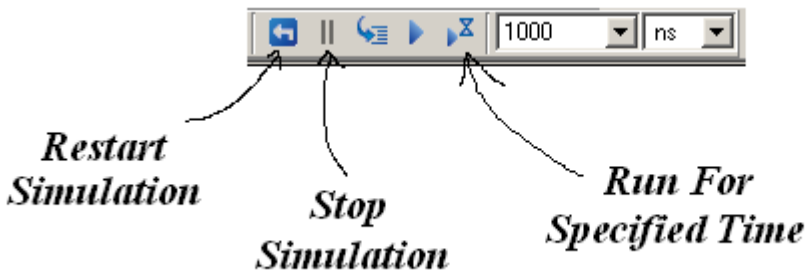
You should use the same design process as that in the tutorial example. Start at the top-level and identify the parts required. The register used to store the access code can be imported from the tutorial and has already been simulated. A comparator will be required that can compare two 3-bit values.

Create a new schematic source for this module called **lock_comp**. This is effectively comparing the value stored in the register with the value being input to try to open the lock. The output should be 'high' for a correct match. This module should be simulated exhaustively. How many patterns does this require? An easy way to achieve this is to set up repetitive waveforms as shown below.

```
initial
begin
  d1 = 0;
  forever #10 d1 = !d1;
end
```

The input to be driven, in this example d1 is initially set to 0. A loop is then set up which is used to invert the signal every 10 time units. The code in the loop will execute for as long as the simulation run time. Use one of these loops for each input of the comparator. They are placed in the user defined area of the test-bench. If you double the wait period for each subsequent loop, you will set up a binary count sequence that covers all of the possible input conditions. Sketch out the waveforms to confirm this.

The default simulation run is 1000 ns. This will not be enough to cover all of the changes for the given wave periods. The simulation run time can be specified in the simulation workspace as shown in the diagram below.



Set the required run time then restart the simulation and click on **Run For Specified Time**.

Check for the eight correct matches.

When the design is functionally correct, create a symbol for it. Complete and simulate the top level then implement and download the design.

Logic Circuits Report

This report should be a technical report of about 1000 words plus essential diagrams. It should detail the design flow for either the synchronous adder or the digital lock and highlight the importance of rapid prototyping with FPGA. You are trying to describe how an idea in your head ends up as a functioning piece of silicon on the desktop. The format should follow the guidelines that you have been given in your professional skills lecture.

The following sections should be included as a minimum.

1. Introduction: This should focus the reader's attention on the subject to be treated. (i.e. rapid prototyping with FPGA) In particular, what is the relevance of this laboratory to modern engineering practice. It should also outline the design problem.

2. Principles of Design Methodology: Here you can describe such things as design techniques for FPGAs (hierarchical design, simulation, synthesis) and detail the design flow from schematic to functioning silicon.

3. Design Section: This should be a brief description of your design. Start by describing how your design solves the problem with the aid of a top-level diagram. Then describe the individual macros.

4. Discussion: This section should give particular detail to the simulation of your design and your choice of test patterns. How exhaustive was simulation of individual modules? How did you choose the test patterns for your top level? Discuss your design. Could it be improved? What are the advantages of rapid prototyping with FPGA?

5. Conclusions: This should summarise the main points of the discussion and refer back to the original objectives in the introduction and state to what extent they were met.

Not all of the knowledge that you require will be readily available. You are expected to use the library for research and ask questions. Remember to reference your sources!

Important Point : Plagiarism.

CHECK THAT YOU KNOW WHAT IS MEANT BY PLAGIARISM and ensure that you do not fall foul of it. Do not use quoted material in this report, it is not necessary or allowed.