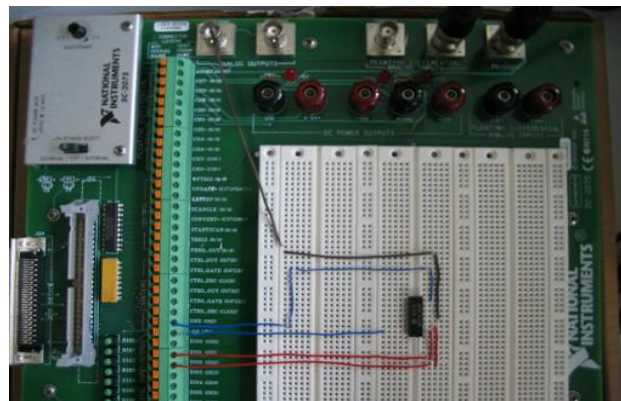




BOĞAZIÇI UNIVERSITY

DEPARTMENT OF ELECTRICAL  
& ELECTRONICS ENGINEERING

# EE 240 Digital System Design



## *Laboratory Manual*

By: Şenol Mutlu and H. Işıl Bozma

2015 İstanbul

Version 7.0

Copyright © Dept. of Electric Electronic Engineering, Boğaziçi University



## TABLE OF CONTENTS

<b>1</b>	<b>ACKNOWLEDGEMENTS .....</b>	<b>3</b>
<b>2</b>	<b>PREFACE .....</b>	<b>4</b>
<b>3</b>	<b>GETTING STARTED .....</b>	<b>4</b>
3.1	CIRCUIT CONSTRUCTION GUIDELINES .....	5
<b>4</b>	<b>LABORATORY 1 – INTRODUCTION TO SC-2075 &amp; LABVIEW .....</b>	<b>7</b>
4.1	DATA ACQUISITION.....	8
4.2	VCC – GND INPUTS .....	8
4.3	DIGITAL I/O.....	9
4.4	USING BREADBOARD.....	9
4.5	BUILDING A CIRCUIT ON BREADBOARD.....	10
4.6	WIRE CONNECTIONS .....	10
4.7	VIRTUAL INSTRUMENTATION WITH LABVIEW .....	11
4.7.1	LabVIEW .....	11
4.7.2	Opening LabVIEW and a VI Script.....	11
4.8	USING A VI SCRIPT.....	12
4.9	HOW TO RUN AND WORK WITH A VI ? .....	13
<b>5</b>	<b>VI SCRIPTS &amp; SC-2075 .....</b>	<b>15</b>
5.1	“ NAME OF THIS VI “ .....	15
5.2	COMPONENTS.....	15
5.2.1	Wave Graph.....	15
5.2.2	Voltage Source.....	16
5.2.3	OUTPUT PORTS.....	16
5.2.4	INPUT PORTS.....	17
<b>6</b>	<b>LABORATORY 1 CONTINUES– USING SC-2075 &amp; LABVIEW TO UNDERSTAND LOGIC BEHAVIOR OF IC GATES.....</b>	<b>19</b>
6.1	LAB MATERIALS:.....	19
6.2	CONCEPT:.....	19
6.3	PREPARATION.....	19
6.4	IN LAB.....	20
<b>7</b>	<b>LABORATORY 2 – COMBINATIONAL CIRCUITS.....</b>	<b>26</b>
7.1	LAB MATERIAL:.....	26
7.2	CONCEPT:.....	26
7.3	PREPARATION.....	26
7.4	IN LAB:.....	27
<b>8</b>	<b>INTRODUCTION TO XILINX FPGA &amp; PROJECT NAVIGATOR .....</b>	<b>32</b>
8.1	HARDWARE.....	32
8.1.1	On the NEXYS3 board .....	33
8.2	THE SOFTWARE.....	34
8.2.1	Design Entry.....	34
8.2.2	Functional Simulation.....	34
8.2.3	Implementation.....	35
8.2.4	Programming.....	35
8.2.5	Debugging .....	36
<b>9</b>	<b>LABORATORY 3 – SEQUENTIAL CIRCUITS.....</b>	<b>37</b>
9.1	LAB MATERIALS:.....	37
9.2	CONCEPT:.....	37
9.3	PREPARATION.....	37
9.4	IN LAB .....	38
9.4.1	With IC Components.....	38
9.4.2	Using FPGA Board.....	38
<b>10</b>	<b>LABORATORY 4 – COUNTERS.....</b>	<b>48</b>
10.1	LAB MATERIALS.....	48
10.2	CONCEPT .....	48

10.3	PREPARATION.....	48
10.4	IN LAB.....	48
10.4.1	Count-Up Ripple Counter.....	49
10.4.2	Decade Ripple Counter.....	51
10.4.3	Two-Stage Counter.....	53
10.4.4	Construct and test the following circuit.....	54
10.4.5	SUMMARY.....	55
10.4.6	SELF-TEST QUESTIONS.....	55
10.4.7	FPGA Programming.....	60
<b>11</b>	<b>LABORATORY 5 – CONVERSION OF 6-BIT FLOATING POINT NUMBERS INTO SIGN-MAGNITUDE REPRESENTATION AND ITS FPGA IMPLEMENTATION .....</b>	<b>64</b>
11.1	DESCRIPTION.....	64
11.2	FPGA IMPLEMENTATION OF THE DESIGN.....	66
11.2.1	Reading the Slide Switches .....	66
11.2.2	Displaying the sign-magnitude number on the four-digit-seven-segment LED Displays Sequentially .....	66
11.2.3	Displaying Digits sequentially on the four-digit-seven-segment LED Displays .....	67
11.2.4	Converting 6-bit binary number into two-digit BCD.....	68
11.2.5	Overall Design.....	69
11.2.6	Pin Assignment .....	71
11.3	PREPARATION (PRELAB) .....	72
11.4	IN LAB .....	72
<b>12</b>	<b>LABORATORY 6 – UP/DOWN BCD COUNTING THE NUMBER OF PUSHES ON BUTTONS AND ITS FPGA IMPLEMENTATION.....</b>	<b>73</b>
12.1	DESCRIPTION.....	73
12.2	FPGA IMPLEMENTATION OF THE UP/DOWN COUNTER.....	73
12.2.1	Reading the Switch .....	73
12.2.2	Up/Down BCD Counter.....	75
12.2.3	Writing the Counter Content to the LED Display.....	75
12.2.4	Overall Design.....	75
12.2.5	Pin Assignment .....	77
12.3	PREPARATION (PRELAB) .....	78
12.4	IN LAB .....	78
<b>13</b>	<b>LABORATORY 7 – SIMPLE VGA DRIVER TO DISPLAY 256 DIFFERENT COLORS...79</b>	
13.1	DESCRIPTION.....	79
13.2	VGA DRIVER .....	79
13.2.1	VGA Signal Timing.....	80
13.2.2	Overall Design.....	82
13.2.3	Pin Assignment .....	84
13.3	PREPARATION (PRELAB) .....	84
13.4	IN LAB .....	84
<b>14</b>	<b>LABORATORY 8 – STUDENT DESIGN PROJECT .....</b>	<b>85</b>
14.1	LAB MATERIALS.....	85
14.2	CONCEPT.....	85
<b>15</b>	<b>FINAL PROJECT LAB REPORT .....</b>	<b>86</b>
1	LAB MATERIAL .....	88
2	THEORY AND METHOD.....	88
3	EXPERIMENTAL DATA.....	88
4	ANALYSIS AND DISCUSSION .....	88
5	ANSWERS TO THE QUESTIONS .....	88
<b>16</b>	<b>REFERENCES.....</b>	<b>89</b>

## 1 ACKNOWLEDGEMENTS

---

The digital systems and circuits laboratory has been re-tooled to use

- Advanced computer integrated virtual instrumentation hardware and software from National Instruments,
- Advanced field-programmable logic chips and boards along with associated PC-based design software from the Xilinx corporations.

We gratefully acknowledge that this renovation has been accomplished through a grant funded by the Boğaziçi University President's Office for the improvement of undergraduate laboratories. Full support to the project has been provided by the Department chair Prof. Kadri Özçaldıran and vice-chair Prof. Dr. Günhan Dünder. Leyla Başürün has given secretarial support to the project. Erdiñ Atman has done all the computer setups.

The lab experiments are redesigned to take maximum advantage of the new hardware-software environment and to enrich the students' digital design experience. The Labview scripts and part of the documentation are designed and developed by a team of BUEE graduate and undergraduate students. We kindly acknowledge the support provided by the E3TAM team.

Previously supplied FPGA boards (Xilinx Spartan II Evaluation Boards from Xess Corporation) have been replaced with Xilinx Spartan VI Evaluation Boards from Digilent in 2013. These boards are donated to our department by the Xilinx Corporation through the Xilinx University Program with the recommendation of Mr. Robert Owen. We kindly acknowledge the support provided by Xilinx University Program and Mr. Robert Owen. Assoc. Prof. Şenol Mutlu applied for the program and solved shipping and handling problems. Mr. Yalçın Turgay and Erdiñ Atman purchased new computers using the department's budget and updated the software for the new hardware.

The first half of the manual, up to Lab 5, is prepared and guided by Prof. Dr. H. Işıl Bozma. FPGA experiments (Lab 5, 6 and 7) are designed and written by Assoc. Prof. Dr. Şenol Mutlu.

## **2 PREFACE**

---

This Lab Manual accompanies the EE 240 Digital System Design course offered in the Electrical Electronic Engineering Department at Boğaziçi University. The labs were developed over the last couple of years – as part of the required course curriculum.

Digital systems have been in an increasing trend of proliferation in the last ten years or so and have become an essential component in almost every gadget we use or depend on in a modern society. Computers are now at the centre of commerce, communication, automation, education, entertainment and military applications. However, computers are not the only digital systems at use – although they certainly are one of the most “visible”. Embedded digital circuits are used everywhere. Although they are not “visible”, they control many diverse aspects of our lives – automobiles, telephones, elevators, televisions, kitchen appliances are just a few examples. We are in midst of a digital revolution that is transforming our lives in numerous profound ways that were very hard to imagine only a small time ago.

EE240 introduces you to the exciting world of digital logic design. This course provides you with a basic understanding of digital systems including the basic components, how they operate and how to design and develop digital circuits. It forms the foundation necessary for more advanced hardware and software that is involved in designing a real system. The laboratory is an integral part of the course since it allows you to have a hands-on experience in regards to taking the theoretical coverage of the lectures of this course and EE142 (Digital Systems course) and learning their application in practice to construct real systems.

---

## **3 GETTING STARTED**

---

EE240 laboratory is located in Kare Blok, KB122. You will be working in groups of two for each laboratory. Each group will be assigned a workbench equipped with a computer connected to a “logic board” that houses the actual digital system being designed (more on this later.) Each group will be given the necessary electronic components at the beginning of each lab.

The student’s grading of lab is determined by three factors:

- Preparation for the lab,
- Performance during the lab and
- The lab report if required. Labs do not have this requirement but the final hardware project does.

Briefly, to pass a lab, the student has to design, build and demonstrate that s/he can implement all the circuits so that each meets all the lab specifications.

Normally, one lab partner is involved.

The student has to come prepared to the lab, having studied the circuit to be built as well as its components. If the lab requires work prior to coming to the lab, each group

should get together before and prepare this material. You will be asked to show your prepared work to your TAs at the beginning of your lab session. If you try to do this work during the lab, not only you will lose points from not preparing ahead, but also you will run the risk of either not being able to complete your lab in time or doing some wrong connections that may damage the devices you are using. Do not take any of these risks. Prepare ahead of time!

Labs 4 – 7 are progressively more challenging – which means that with increasing lab numbers, you have to use your lab time much more efficiently in order to be able to complete your lab in time.

### 3.1 CIRCUIT CONSTRUCTION GUIDELINES

You will be building your circuits on the NI sc-2075 breadboard. You will be given one introductory session at the beginning of the first lab on how to use a breadboard – that is how to connect a power supply to the breadboard, how to put components on the breadboard and how to view the outputs of your circuit. The following steps will help you with the labs and ensure that you will have minimal frustration:

1. Come prepared to the lab:
  - a. It is a good idea to begin your work with a design on paper or using a CAD tool.
  - b. Look at the datasheets of the components to be used. The most important about each chip is pinout – which pin is assigned to which operation. Some pins are input, some pins are output and two are for power and ground. Along with the datasheet's pinout, be sure to look at the truth table to understand the inner workings of the chip.
  - c. On your design, be sure to specify the pin numbers for each wiring connection. This will speed the circuit setup time during your lab.
2. Getting Organized: Arrange the parts required so that you do not waste time looking through all your components when you need a particular chip.
3. Assembling your circuit:
  - a. Do it neatly, with wires that are not so long that they rise up above the breadboard with one confused mass, but not so short that they loop directly over the tops of the chips and prevent you from popping out individual chips to test for defective chips (which may well happen !!). **Do not pull out a chip out of breadboard with your fingers!!!** The pins are liable to bend. Insert something like a bobby pin under the chip and pry the chip out gently or with a pair of pliers. If you need to cut a wire, doing it with an angle is better since the ends will have points that will more easily slip into the breadboard holes.
  - b. Use a consistent color scheme for your wires --- all white or black for ground; red or green for +5V; etc.
  - c. If you are building a complex circuit, do it modularly. Build the display first, the input, etc. That is, break the problem into smaller sections and do one section at a time. Test each section before you go on to the next.

4. **Logic Probes:** If you are provided with logic probes, which are attached by two cables, you can use it as your first weapon in troubleshooting the circuit. With it you can tell quickly the logic levels in your circuit. The LO LED (red) is on for voltages less than about 0.8 volt and HI LED (green) lights for voltages greater than about 2.2 volts. If the voltage is between 0.8 and 2.2, neither of the LEDs will light. If you are not provided with logic probes, you can use the LEDs on your breadboards as a logic probe.
5. **Power Supplies:** Each unit has a power supply. Un-plug your power supply while working on your circuit. Wire in the power only after you have double-checked your circuit and have made sure that there are no short-circuits.
6. **Breadboard:** The holes on your breadboard are connected in a special topology – as is explained in Section 4 as well as at the beginning of lab 1.
7. **Design, Troubleshooting & Frustration:** However well you organize your time and are faithful to your schedule, you will be occasionally frustrated by design or troubleshooting problems. Do not have an anxiety crisis.
  - a. Make sure that you understand the theoretical material that is covered by the lab.
  - b. Make sure that you understand the lab specifications.
  - c. During troubleshooting, calmly check your wiring. Are the power supply connections made properly? Are all the inputs to each gate accounted for? Do all the outputs go somewhere? Are you getting the correct voltage? Does one chip seem like the culprit? Take it out and test it by itself. If you have a defective chip, you should exchange it for a good one.
  - d. If you are still stuck, you may ask the TA. Please try to pin down the problem as much as possible so that s/he may help as efficiently as possible.



#### 4 LABORATORY 1 – INTRODUCTION TO SC-2075 & LABVIEW

In EE240 laboratory, SC 2075 Breadboard of National Instruments is used for circuit design. The SC-2075 connector accessory is shown in Figure 1. It includes a prototyping breadboard area for the construction and testing of circuits. The different parts on these devices are as identified in Figure 1. The SC-2075 is connected to the data acquisition (DAQ) board in the computer via a shielded cable [1]. This device allows a designer to prototype his/her design in the breadboard area. One can use both real and virtual instrumentation with the circuit implemented on the breadboard. To be able to use virtual instrumentation, we use a program called LabVIEW. In this case, the input signals can be sent to the circuit via the use of program coded within the LabVIEW program. The output signals can also be observed from a code within LabVIEW front panel or can be measured directly from the breadboard itself.

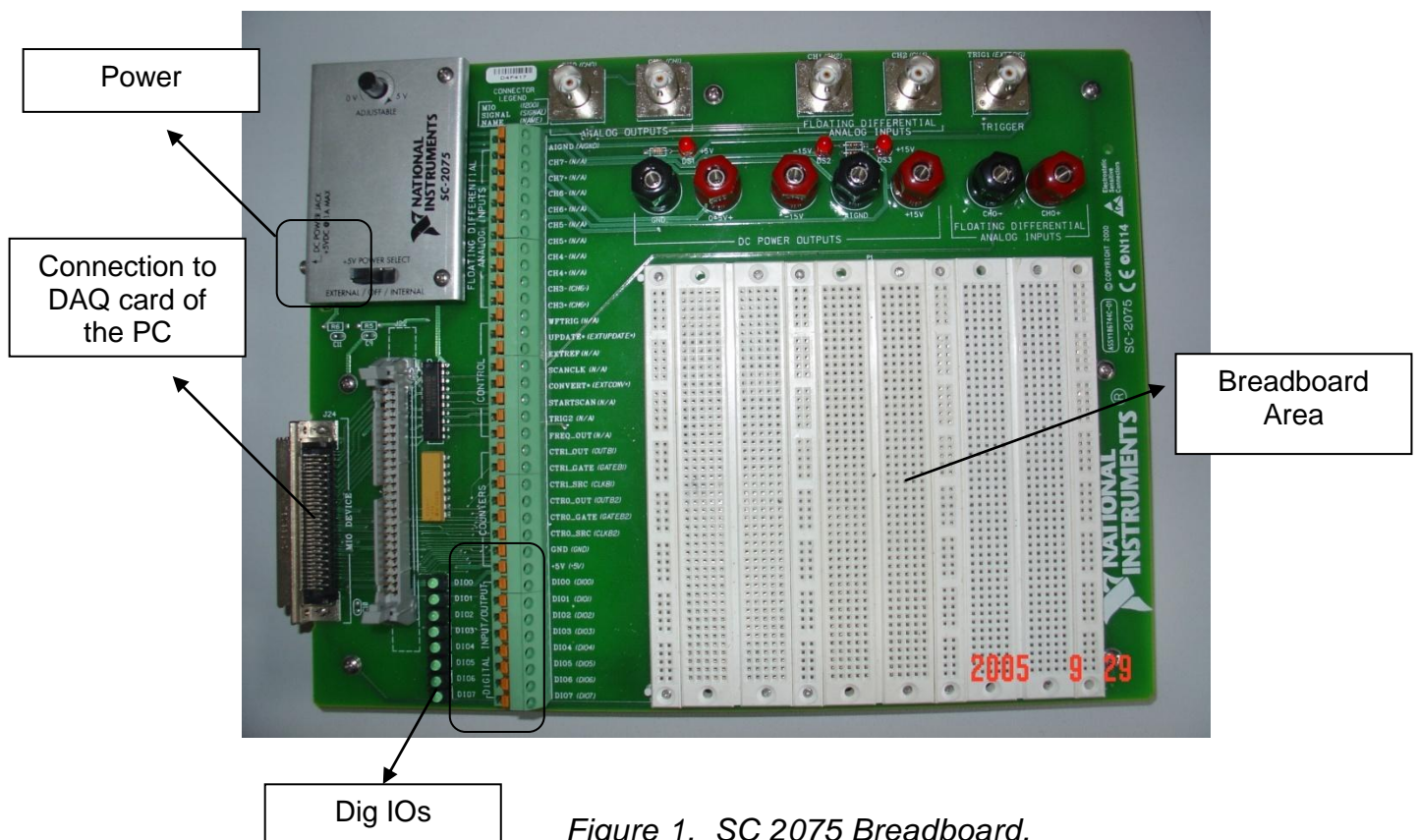


Figure 1. SC 2075 Breadboard.

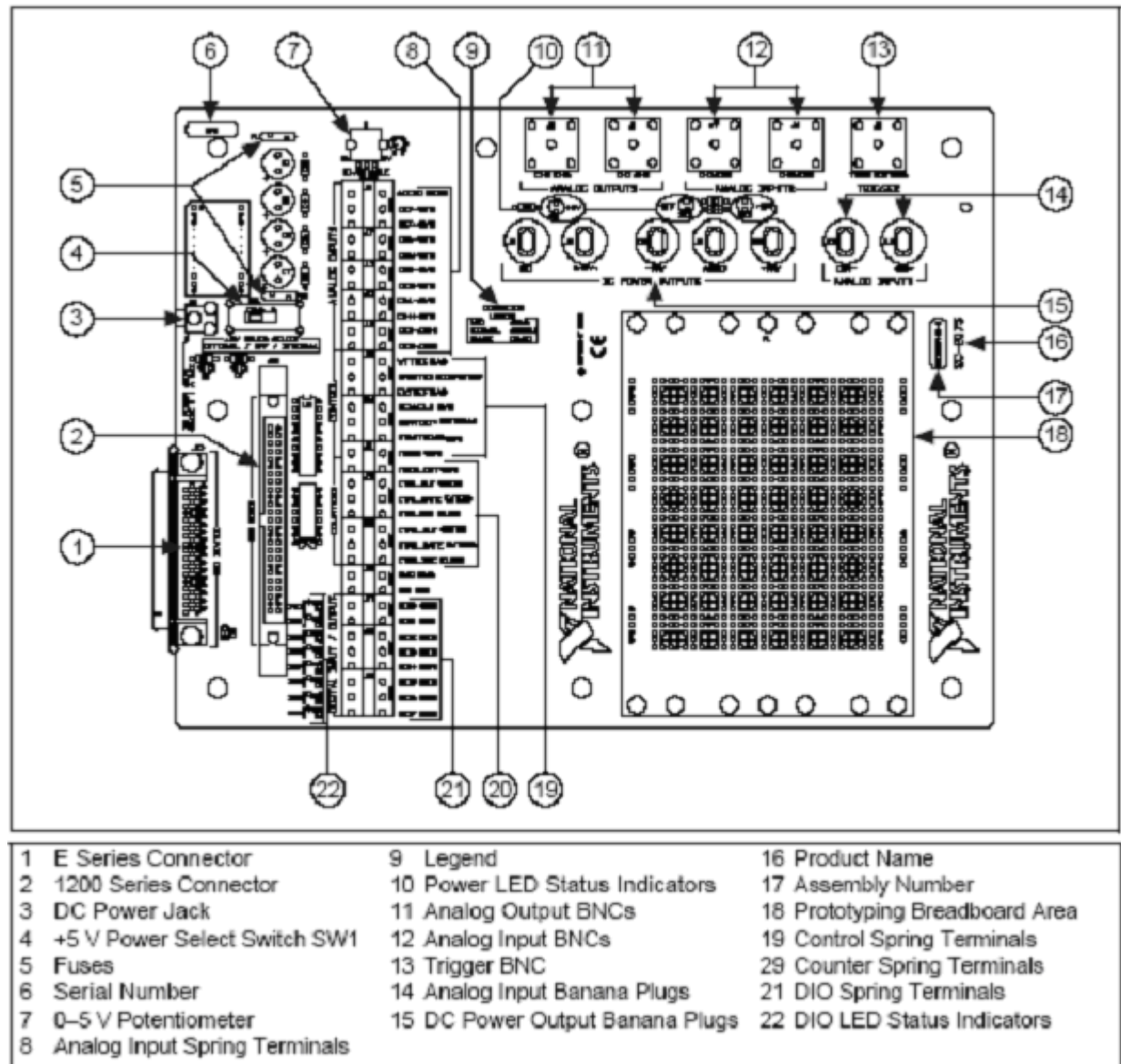


Figure 2: SC-2075 Breadboard connections [1].

#### 4.1 DATA ACQUISITION

The purpose of a data acquisition system is to capture and analyze some physical phenomenon from the real world. A typical DAQ board has analog inputs for making measurements of electric signals (thereby physical quantities). It has analog outputs for generating a stimulus [2].

#### 4.2 VCC – GND INPUTS

These can be taken from the green sockets with a wire. For ground and 5V there are GND and 5V input (Power Supply) on the socket, respectively. The connection will be accomplished with the serial port cable. For the other outputs that will be taken from the program the channels are used.

### 4.3 DIGITAL I/O

The SC-2075 contains spring terminal access to 8 digital input-output (DIO) channels as shown in area 21 in Figure 2. Each channel can operate either in read or write mode.

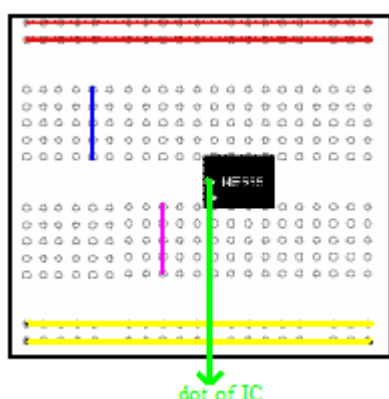
In the read mode, the channel logic level is set to that of the point which it is connected to. Using an appropriate vi code, it is possible to read this value in the Labview framework. If this channel is connected to an output pin of a component, it is then possible to determine the output logic level at an output point in the circuit.

In the write mode, the channel logic level is set to a value as determined by the vi code. Using an appropriate vi code, it is possible to write either a logic level "0" or "1" to this channel. If this channel is connected to an input pin of a component, it is then possible to send this input to the circuit.

Eight LEDs correlating to the eight DIO lines indicate the state of each digital channel. If the LED is lit, the channel is pulled high (high > 2.6 V). If the LED is off, the channel is pulled low (low < 0.4 V). A ground is available at the spring terminal labeled GND and is the reference for the DIO lines.

A very important final note: It is possible to access these channels through vi script.

### 4.4 USING BREADBOARD



We use the breadboard area in order to implement temporary circuits to try a circuit design idea. No soldering is required so it is easy to change connections and replace components. ICs and the cables are re-usable on the breadboard.

Breadboards have many tiny sockets called 'holes'. These holes have a special topology and connectivity as indicated in Figure 3 [3]. The properties of this topology need to be known prior to using the breadboard.

*Figure 3: Breadboard topology. [3]* The holes in the top and bottom rows are arranged in a horizontal structure as shown by the **red** and **yellow** lines on the diagram in the same figure. All the holes in one row are short circuited which means that electrically they are at the same potential.

The other holes are in a **vertical arrangement** in blocks of 5 as shown by the **blue** and **pink** lines on the same figure. All holes in one 5-tuple is short circuited. There is a groove separating each 5-tuple vertically.

A component is placed on this breadboard by making sure that each pin goes to a different 5-tuple hole block. Consequently, ICs are inserted across the central gap with their dot to the left. Wire connections are made with single-core plastic-coated wire. Notice how there are separate blocks of connections to each pin of ICs.

## 4.5 BUILDING A CIRCUIT ON BREADBOARD

When putting parts on breadboard, you must concentrate on their **connections** rather than their actual positions on the circuit diagram. The IC (chip) is a good starting point so place it in the centre of the breadboard. A good strategy is to work around it pin by pin, putting in all the connections and components for each pin in turn. IC pins are numbered anti-clockwise around the IC starting near the dot of the IC. Figure 4 shows the numbering for 14-pin IC. There are different types of ICs with 8, 16, etc... pins but the principle of numbering is the same for all sizes.



Figure 4: 7400 IC

## 4.6 WIRE CONNECTIONS

After placing the IC on the breadboard we make connections between them and with Ground/ VCC. To do these connections we use wires. An example is shown in Figure 5. The 8th pin is connected to one of the holes of the purple row (all holes of the purple row have the same value). Moreover, the 9th pin is connected by red wire to the green row. Figure 6 shows how to connect one end of a wire to the breadboard explicitly.

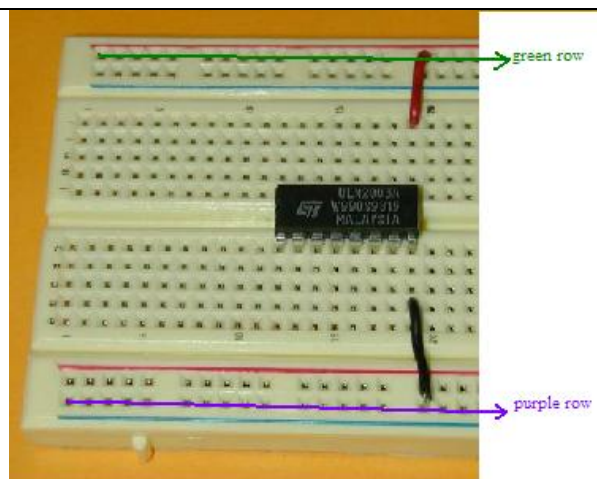


Figure 5: Wire connections [4].

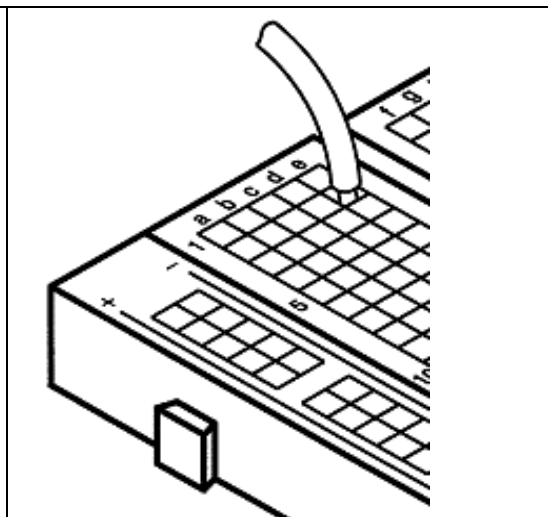


Figure 6: Inserting a wire to one hole [5].

Note that the wiring must be done such that each wire lies close to the breadboard and follow straight paths as much as possible. An example of bad wiring is as shown in Figure 7 . The wires are sloppily placed and dangled through each other. An example



of good wiring is shown in Figure 8. It is very easy to follow the connections of each wire in this case.

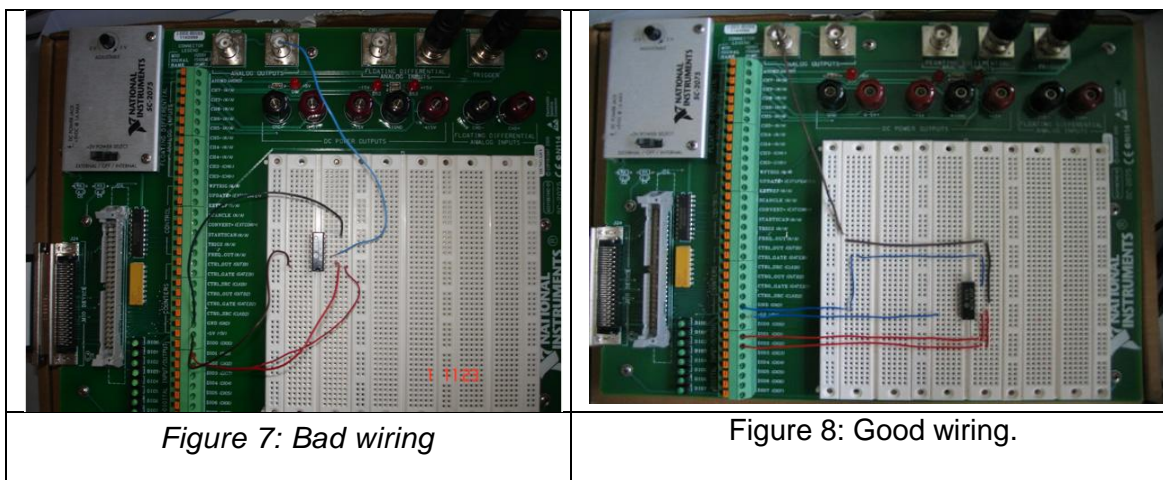


Figure 7: Bad wiring

Figure 8: Good wiring.

## 4.7 VIRTUAL INSTRUMENTATION WITH LABVIEW

The outputs can be taken from LabVIEW with Front Panel components and the outputs of the circuit can be seen too. The VI's Front Panel is used to control the program formed in the Block Diagram. The buttons, switches, knobs and indicators help us to form the Front Panel.

### 4.7.1 LABVIEW

With National Instruments LabVIEW, one can build a graphical program called a virtual instrument (VI) instead of writing a text-based program. In this environment, we can quickly create a front panel user interface that gives the user interactive control of the circuit implemented on the SC-2075 breadboard.

The user interface is implemented by assembling block diagrams. The resulting scripts are called VI scripts and have .vi suffixes. In the first four labs, you will not be required to develop VI scripts. Instead you will be using scripts previously developed for you.

### 4.7.2 OPENING LABVIEW AND A VI SCRIPT

If the VI that is related to your laboratory is prepared for you, please click related program icon on desktop. Otherwise if you are asked to open a new VI you should follow these steps:

**Start → Programs → National Instruments → LabVIEW 6 → LabVIEW**

Without filling the blanks on the first window that will be opened click "OK" twice. From "Open VI" button you can open the loaded VIs. For preparing a new VI, select "New VI".

However, for you there is a folder named EE240 in which there is a vi script. You can reach it in this way:

**e:/ee240/**

#### 4.8 USING A VI SCRIPT

Once you have opened up your script, you will see two panels open up: **Front Panel** and **Block Diagram**.

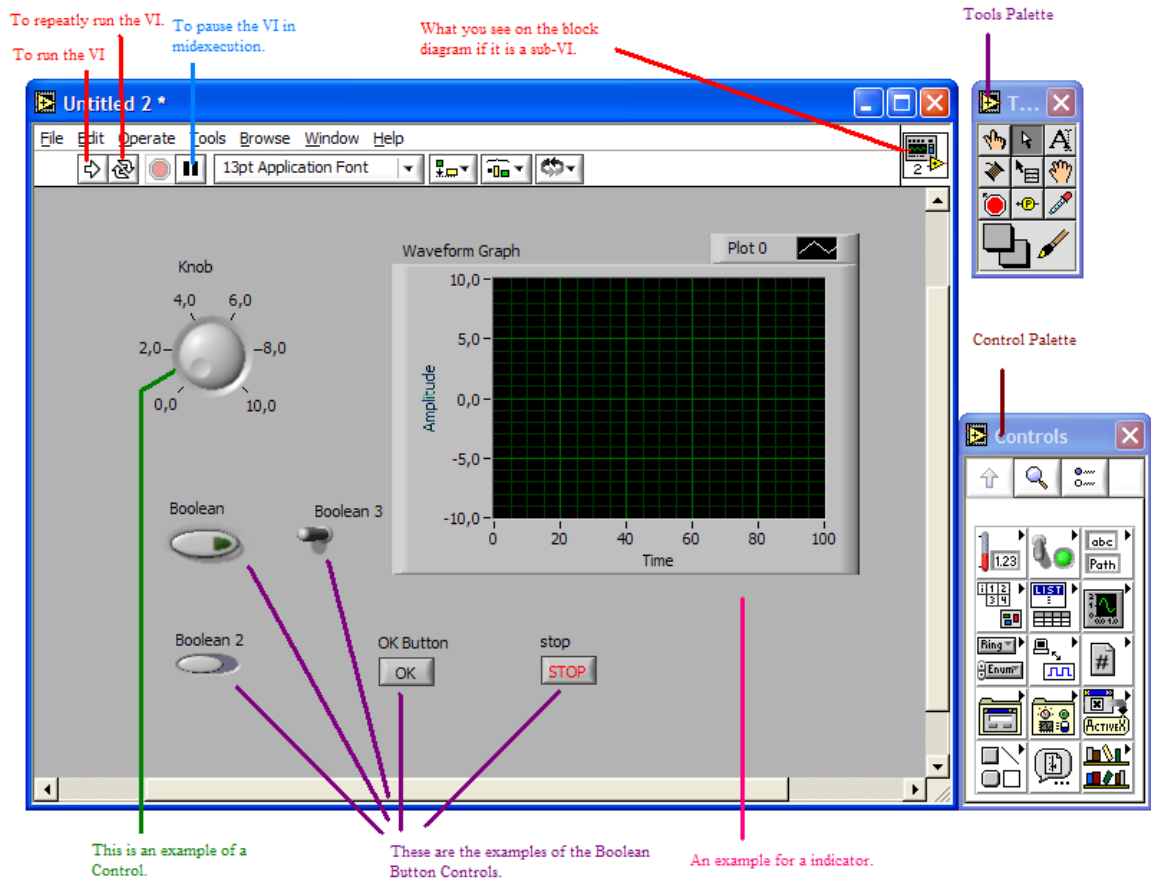


Figure9: An example of Front Panel.

The front panel contains the user interface of your vi code. An example front panel is shown in Figure 9. The block diagram contains the graphical code for your VI. However, you will not work with the programming of the vi scripts since they are designed and developed for you. You will be using already designed vi codes.

In order to use a vi code, you need to understand the “Front Panel” of your vi code. In order to send a signal to your circuit or read a line value from the circuit, you need to run “appropriately built” vi code. The user interface with the vi code is achieved through the front panel using components such as a slide bar, zoom-in graphs or text areas.

For those who may be interested, this panel is constructed using components from the Controls palette, such as numeric displays, meters, gauges, thermometers, tanks, LEDs, charts, and graphs. The vi language is visual-based using block diagrams. In this manner, you are isolated from worrying about the syntactical details of text-based programming languages. A code is constructed via selecting objects (icons) from the

Functions palette as shown in Figure 10 and connecting them together with wires to transfer data among block diagram objects [6].






Figure 10: Functions palette.

The Control Palette is used to place controls and indicators on the Front Panel. Each palette icon represents a subpalette. A control is formed for the interaction with the vi. Simple examples of controls are buttons, slides and knobs. An indicator is a front panel object that displays data to the user. Examples of indicators are graphs and thermometers. When you place a control or indicator on the front panel, a corresponding terminal is placed on the block diagram.

The code is built via constructing a block diagram using the terminals from the front panel controls and indicators and the VIs, functions, and structures from the Functions palette. However, you will not deal with the Block Diagram and Functions Palette.

#### 4.9 HOW TO RUN AND WORK WITH A VI ?

The circuits can be run in either one-shot or repeated excitation modes. In order to excite the circuit with the inputs only once, use the Run Button  on the Front Panel which is shown with a right sided arrow in the Diagram and Functions Palette. If you want to run the VI continuously, use the  button. In other words when you run a VI continuously you give the input more than once and repeatedly. To stop running, please click on the  button. In order to control the inputs by the Control and Boolean icons, you can click on them. You can change the inputs by sliding, clicking and moving the buttons. You can examine the outputs of the circuit by the indicators such as graphs.

A sample front panel is as shown in Figure 11. This is an example of Digital Port Control which is mostly used in our LabVIEW programs on the front panel. The code is available from [7]. With the help of this VI we can send information to the breadboard or take from it for outputs. In this code, the digital IO channels of the sc-2075 device are connected to the channels that are shown in Figure 11 at the bottom.

- First you decide which channels you have to use in your circuit. For example, if you want to use channel 1 as **input** to the breadboard (if you want to write the

information given from the Front Diagram by Digital Port Control to an input of a component on the breadboard), you need to configure this channel to the “write” mode. You should select the **Line 0**’s option as **Write**.

- If you want to use channel 6 as **output**, you should select Line 5’s position as Read. Don’t forget to select these positions; otherwise the program will not work.
- If you complete these steps, now you can set the inputs. “Write Values” is prepared for this purpose.
- On “Write Values” part, by positioning the button to “On” level which means Logic 1 and to the “Off” level which means Logic 0, you can assign the values of inputs of each line of the breadboard.
- By the way, you can follow the outputs of breadboard from the “Read Values” part. Dark green symbolizes Logic 0 and light green symbolizes Logic 1. For this VI example, please click on the red “STOP” button to finish running.

If we look at this VI, it is easy to say Line 0,1,2,3 are prepared for “Write” and this means they are inputs. Line 5,6,7,8 are on the “Read” option and it shows that they are outputs. On the “Write Values”, we can see Line 0 and Line 2 are Logic 1 because they are positioned to the “On” level. Line 1 and Line 3 are Logic 0, because they are positioned to the “Off” level. On the “Read Values”, it is obvious that Line 4, 5 and 7 are Logic 1 and Line 6 is Logic 0.

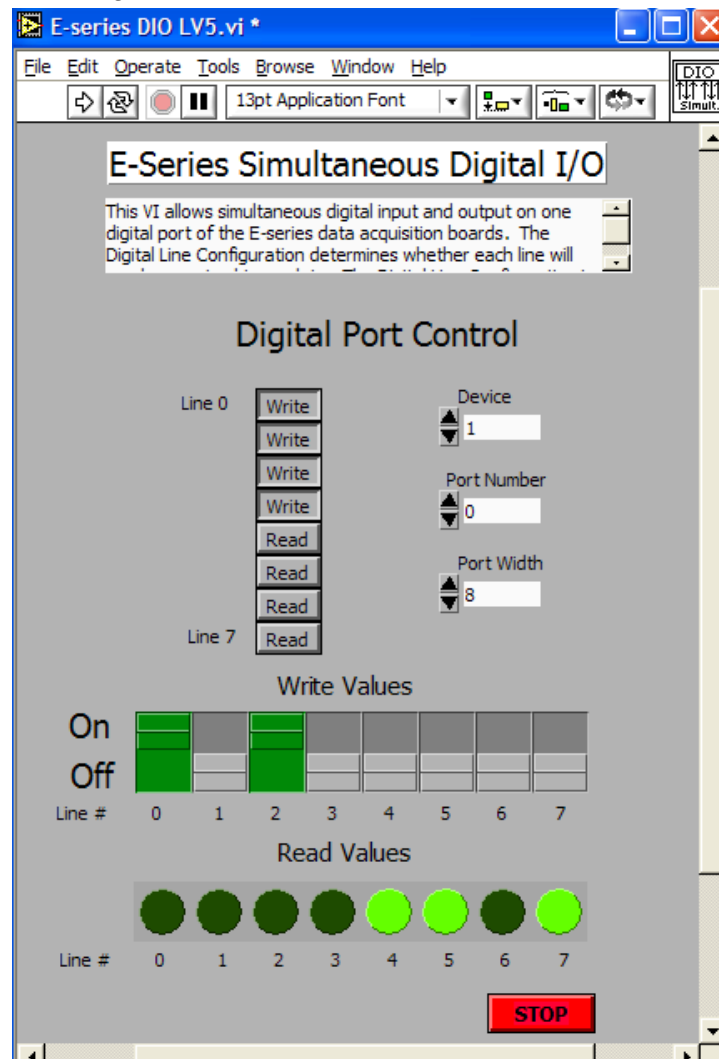


Figure11: Digital Port Control on the Front Panel.



## 5 VI SCRIPTS & SC-2075

In this course, you will be using previously developed vi scripts. This section describes each in detail.

### 5.1 “ NAME OF THIS VI “

The first vi is for lab1. It is accessible from e:\ee240\. Once opened using the directions as specified in previous section, it has a front panel as shown in Figure 12. This vi can be used to send pulse or Boolean signals to five alternative DIO output ports and read Boolean signals from three input DIO.

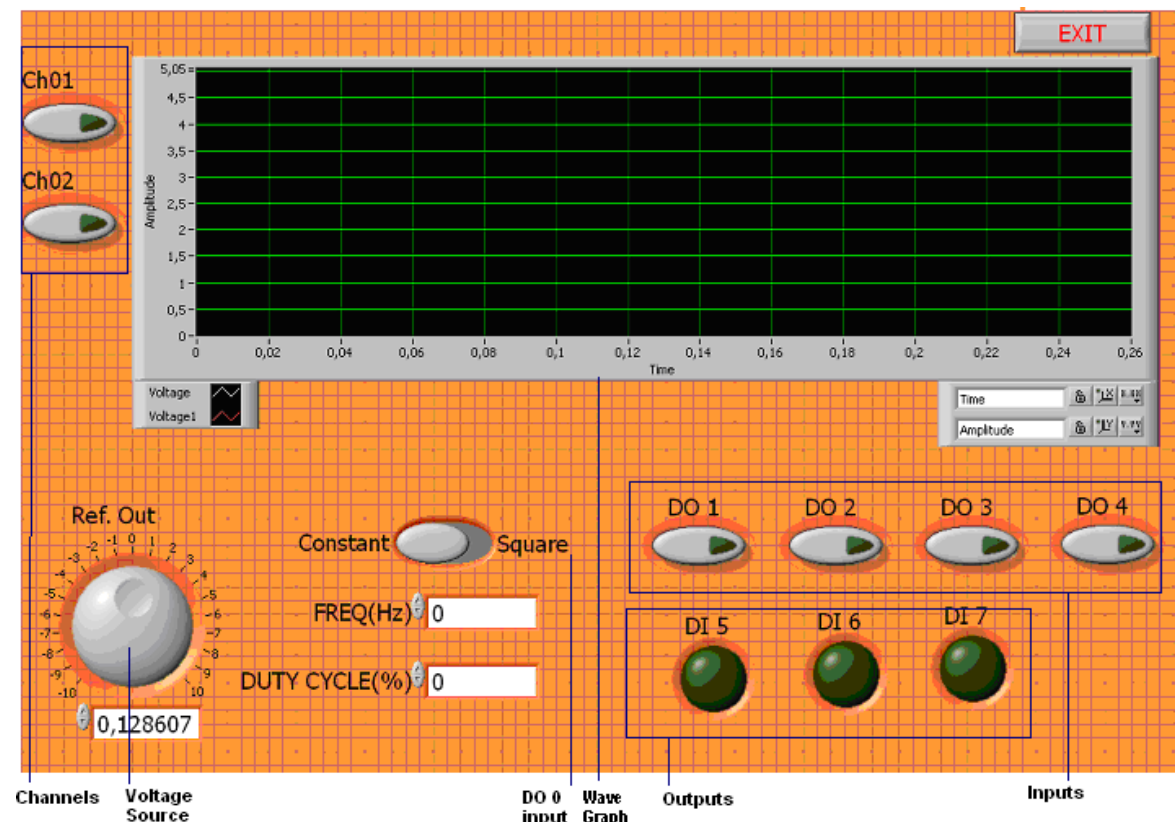


Figure12: Front panel of lab1 vi script.

### 5.2 COMPONENTS

This vi script contains several components:

1. Wave Graph
2. Voltage Source
3. Square Wave Generator
4. DIO Output Ports
5. DIO Input Ports

#### 5.2.1 WAVE GRAPH

One of the main components of the VI is **Wave Graph**. The Wave Graph is a virtual digital oscillator connected to the two analog BNC inputs of the sc-2075 device. The

analog BNC inputs are shown in (12) in Figure 2. They are referred to as Ch01 and Ch02.

- The Wave Graph can show only one of the channels or both depending on the user's choice.
- Each channel can be turned on or off by clicking on the channel button on the vi front panel.
- In order to associate a channel with a specific signal, you must put a physical wire between that point and the respective analog BNC input.

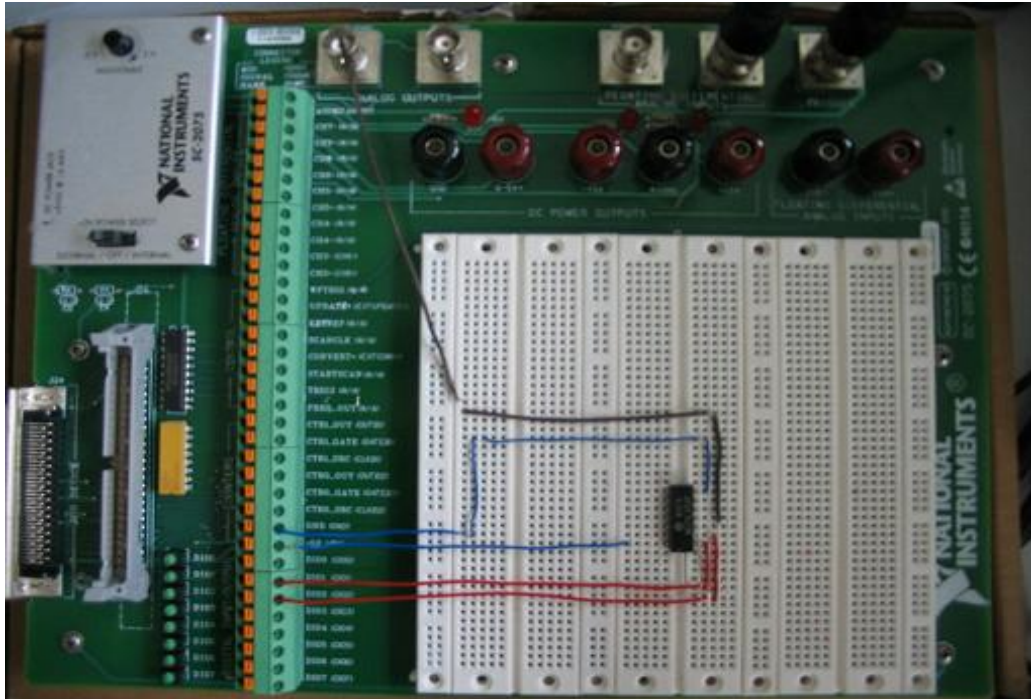


Figure 13. Connection to the Wave Graph

You can observe the usage of the wave graph by the connections of a simple circuit in Figure 13. The output of the IC (blue wire) is connected to second channel of the wave graph.

### 5.2.2 VOLTAGE SOURCE

“**Voltage Source**” supplies the breadboard an input other than Logic 1 or GND ranging between -10 and 10 Volts. However, we will not use this source much in our circuits.

### 5.2.3 OUTPUT PORTS

The first five channels of the breadboard from DO0 to DO4 are all used as inputs for the circuit on the breadboard.

**DO0** is different from the others as it can both become a **Clock Pulse (CK, CP)** or **constant input**. The mode is selected by toggling the respective button. If selected to operate in the CP mode, the frequency and the duty cycle of the generated pulse can be arranged via **FREQ** and **DUTY CYCLE** text areas. Each value is specified by clicking on the respective white area and simply writing in the desired values.

The remaining four buttons **DO1**, **DO2**, **DO3** and **DO4** operate in a switch-like manner. Each DOx is connected to DIO x on the sc-2075. Each can be independently controlled to be either in “off” or “on” state. If in “off” mode, then it is outputting “Logic 0” level which means +0V on the respective DIO port. Conversely, if in “on” mode, then it is outputting “Logic 1” level which means that there is +5V on the respective DIO port. The state of each port is signaled by an associated green LED. A green LED that is off indicates that there is “Logic 0” at that port. Conversely, a green LED that is on shows that there is “Logic 1” at that port. For example, if you click DO3 so that the associated green LED is lit, there is +5V on this port. If you then connect this port to an input pin of an IC, you will input the associated logic level at that pin.

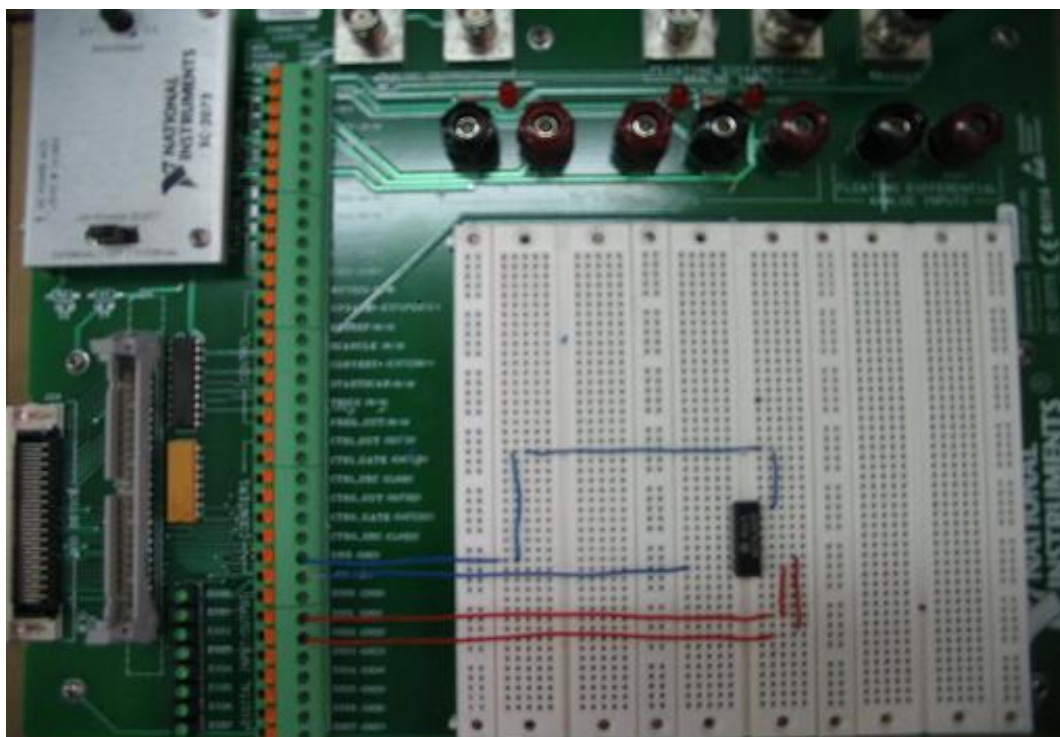


Figure 14. Usage of the Output Ports

You can observe the usage of the Output Ports by the connections of a simple circuit in Figure 14. The inputs of the circuit are taken from the output ports **DO1** and **DO2**. Looking at the data sheet of the 74LS00, the first and second pins of the IC are inputs of one NAND Gate. The third pin is output. We configure the ports according to this information.

#### 5.2.4 INPUT PORTS

There are 3 DIO ports ranging from DIO05 to DIO07 as input ports: **DI5**, **DI6**, **DI7**. These ports are configured so that you can examine only constant values from them. If you need to view changing signals, you need to use the Wave Graph.

- First, each output point of the circuit under design must be connected to one of these DIO ports.
- Each input port is associated with a green LED in the vi script.
- Each green LED lights up or not depending on the logic level in the associated DIO port.

- If there is “Logic 0” (and hence 0V) at that port, the LED does not light up. If there is “Logic 1” ( and hence +5V), the LED lights up.

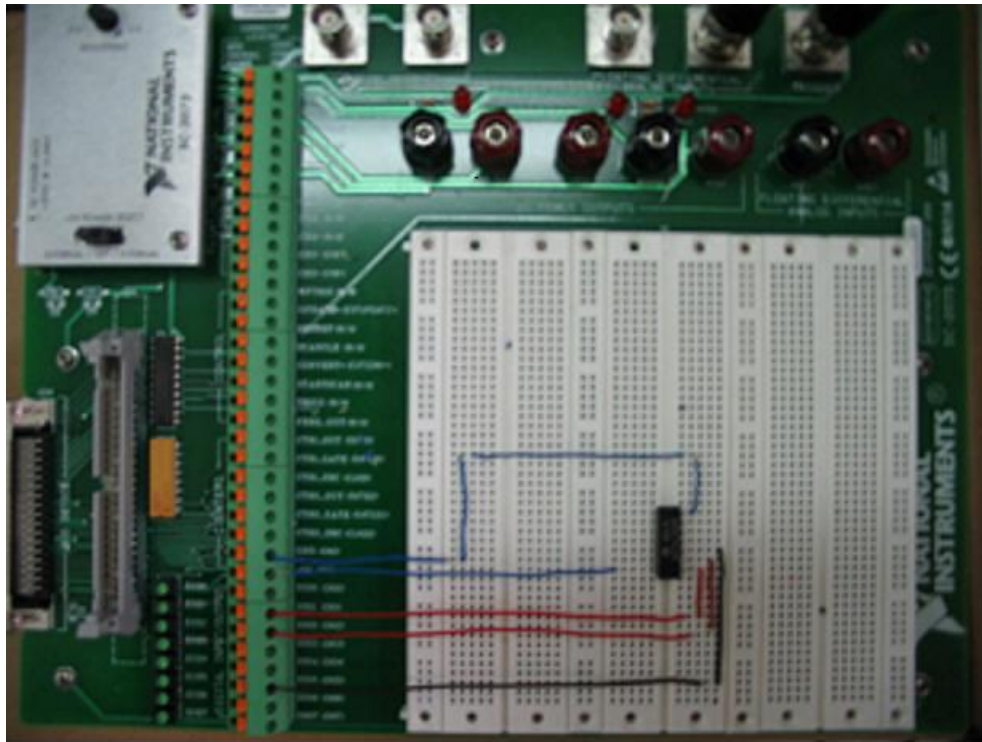


Figure 15. Usage of the Input Ports

You can observe the usage of one Input Port by the connections of a simple circuit in Figure 15. The output of the circuit (blue wire) is connected to **DO5** port.

---

## 6 LABORATORY 1 CONTINUES– USING SC-2075 & LABVIEW TO UNDERSTAND LOGIC BEHAVIOR OF IC GATES

---

### 6.1 LAB MATERIALS:

NI SC 2075

ICs: 74HC00, 74HC02, 74HC04, 74HC86

Lots of wires, pliers

### 6.2 CONCEPT:

In the first experiment, the goal is to familiarize you with the lab setup and to expose you to the usage of IC gates using NI SC 2075 breadboard interface and Labview software. Make sure that you read the first three sections carefully. In this lab, you will

- Learn NI SC 2075 breadboard
- Implement a simple digital circuit
- Learn using a Labview vi script to send values to and read from this circuit
- Investigate the logic behavior of various IC gates:

74HC00 or 74HCT00	Quadruple 2 input NAND gates
74HC02 or 74HCT02	Quadruple 2 input NOR gates
74HC04	Hex inverters
74HC86	Quadruple 2 input XOR gates

### 6.3 PREPARATION

This being your first lab, the actual **design** work required is quite minimal. However, you will need to prepare wiring connection diagrams. A wiring connection diagram is a drawing that shows your TTL circuit - including pin connections, ground and power connections.

1. You should get the datasheets of each IC from either the web or a databook.
2. For each IC, draw on paper the wiring connections as required by part 1 of Section 6.4. As you will be investigating four different IC's, you should prepare four such drawings. Use the space provided in Figure 16 through Figure 19.
3. Prepare the theoretical truth tables for part 1 of Section 6.4 by filling in the respective parts of the tables as given in Table 1.
4. Prepare the wiring connection diagram for part 2 in Section 6.4 on paper. There should be four of such drawings. Use the space provided in Figure 20 through Figure 23.
5. Prepare a wiring connection diagram for part 3 in Section 6.4 on paper. (Figure 24)
6. Construct the theoretical truth table for part 3.2 in Section 6.4 by filling in the truth table of Table 3 up to and including "theoretical" part.

## 6.4 IN LAB

1.) Use one gate from each IC and obtain the truth table of the gate. The truth table is obtained by connecting the inputs of the gate and the output to the DIO's on the sc-2075 board. Compare your results with the theoretical results by filling in the appropriate space provided in Table 1.

2.) Using a single 7400 IC, connect a circuit that produces:

2.1. Inverter

2.2 A 2-input AND

2.3 A 2-input OR

2.4 A 2-input XOR

Verify the truth table via sending binary inputs from 0 to 3 inclusively and reading out the output on Table 2.

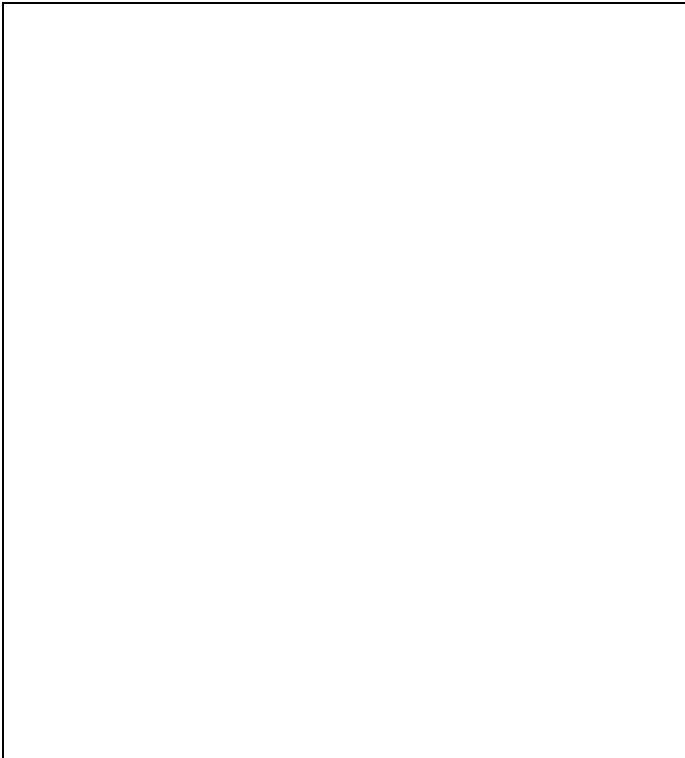
3.) Using a single 7400 IC, connect a circuit that implements the Boolean function

$$F = AB + CD$$

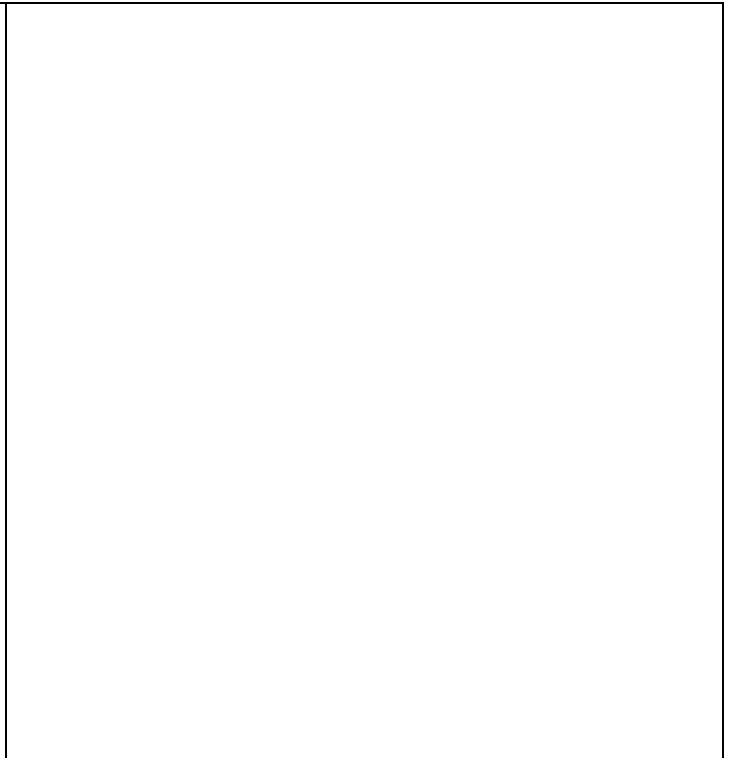
3.1 Draw the circuit diagram.

3.2 Obtain the truth table for F.

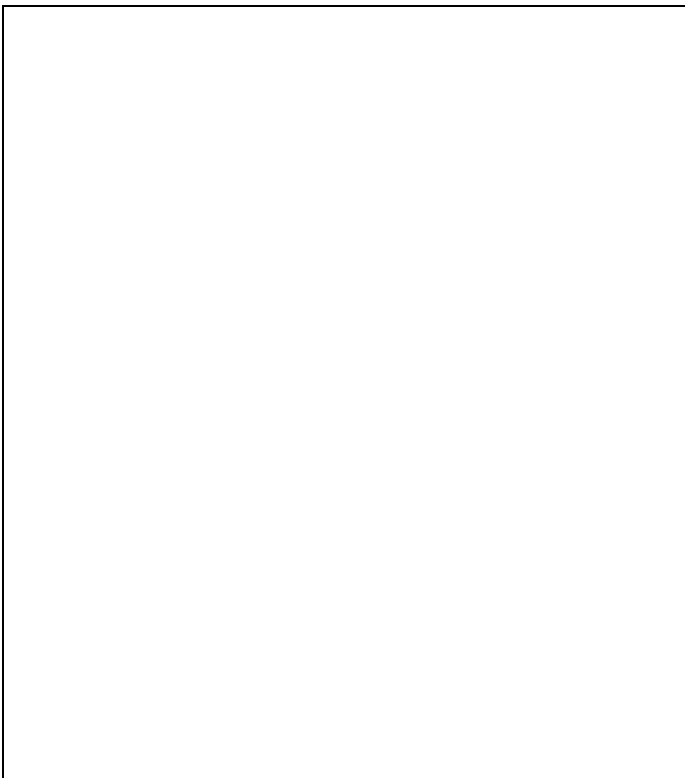
3.3 Connect the circuit and verify the truth table as inputs A, B, C and D go from binary 0 to binary 15 by filling in Table 3.



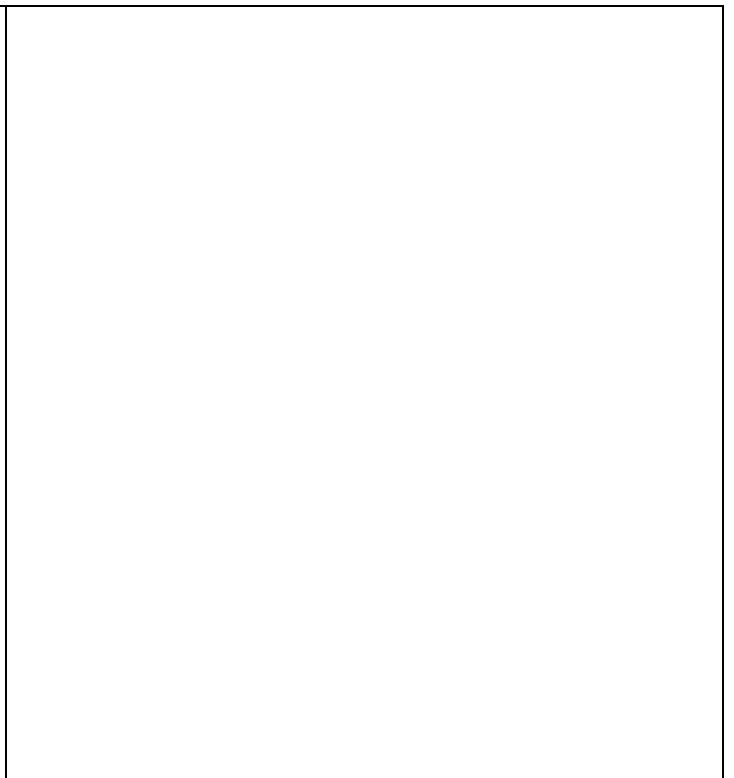
*Figure 16: IC Wiring connection for Section 6.3 part 2.*



*Figure 17: IC Wiring connection for Section 6.3 part 2.*



*Figure 18: IC Wiring connection for Section 6.3 part 2.*



*Figure 19: IC Wiring connection for Section 6.3 part 2.*

Table 1: Theoretical and Measured Truth Tables.

Gate: \_\_\_\_\_ Gate: \_\_\_\_\_

X2	X1	Theoretical	Measured

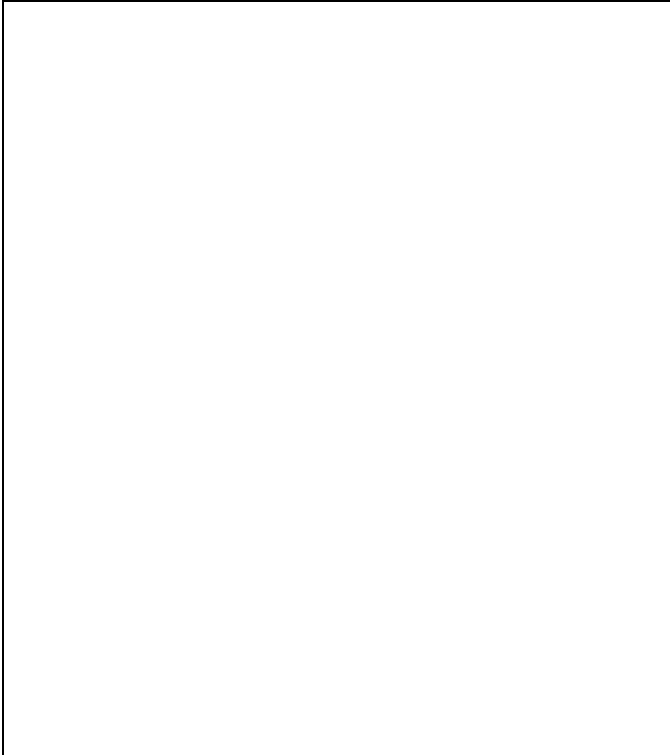
X2	X1	Theoretical	Measured

Gate: \_\_\_\_\_ Gate: \_\_\_\_\_

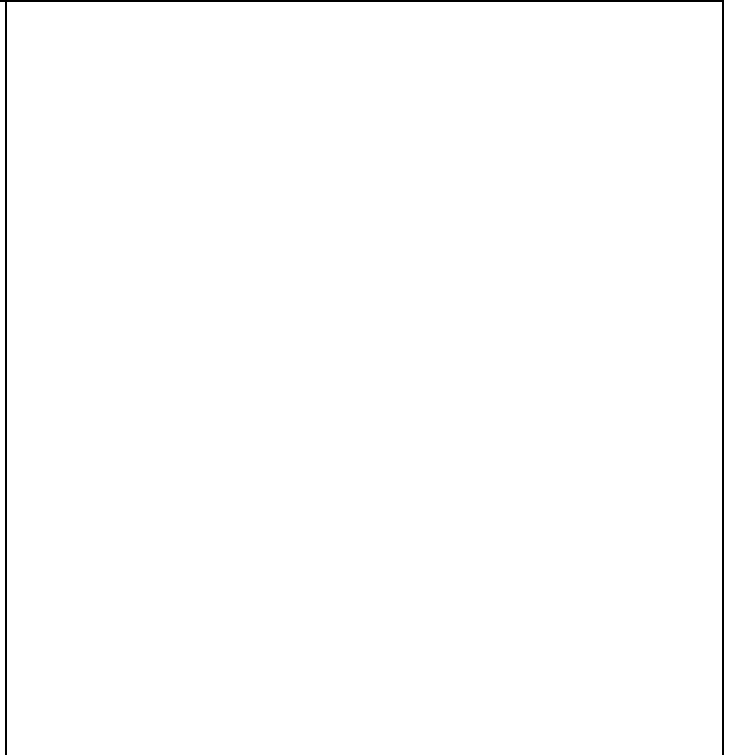
X2	X1	Theoretical	Measured

X2	X1	Theoretical	Measured

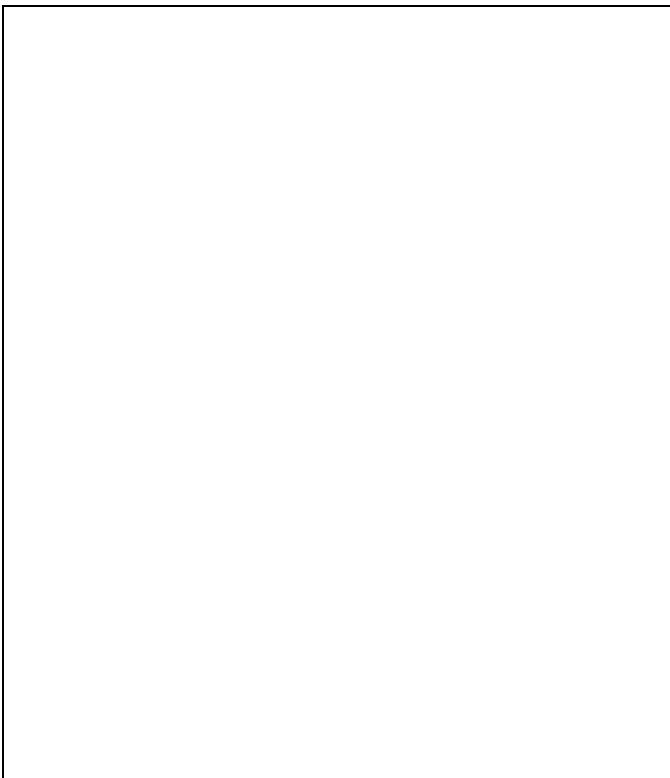




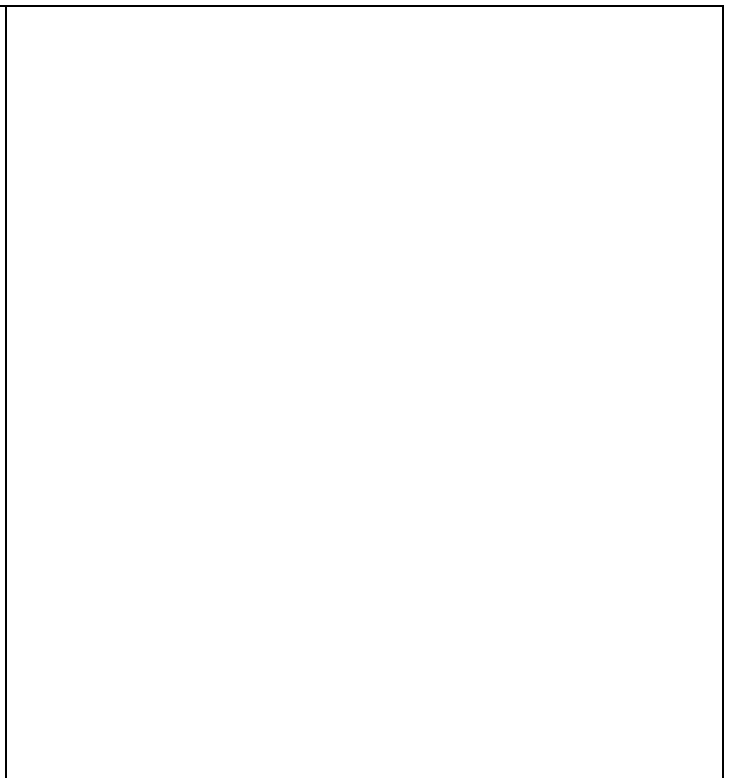
*Figure 20: IC Wiring connection for Section 6.4 part 2.1*



*Figure 24: IC Wiring connection for Section 6.4 part 2.2*



*Figure 25: IC Wiring connection for Section 6.4 part 2.3*



*Figure 26: IC Wiring connection for Section 6.4 part 2.4*

Table 2: Theoretical and Measured Truth Tables (Section 6.4 part 2)

Implementing  
Gate: .....

X2	X1	Theoretical	Measured

Implementing  
Gate: .....

X2	X1	Theoretical	Measured

Implementing  
Gate: .....

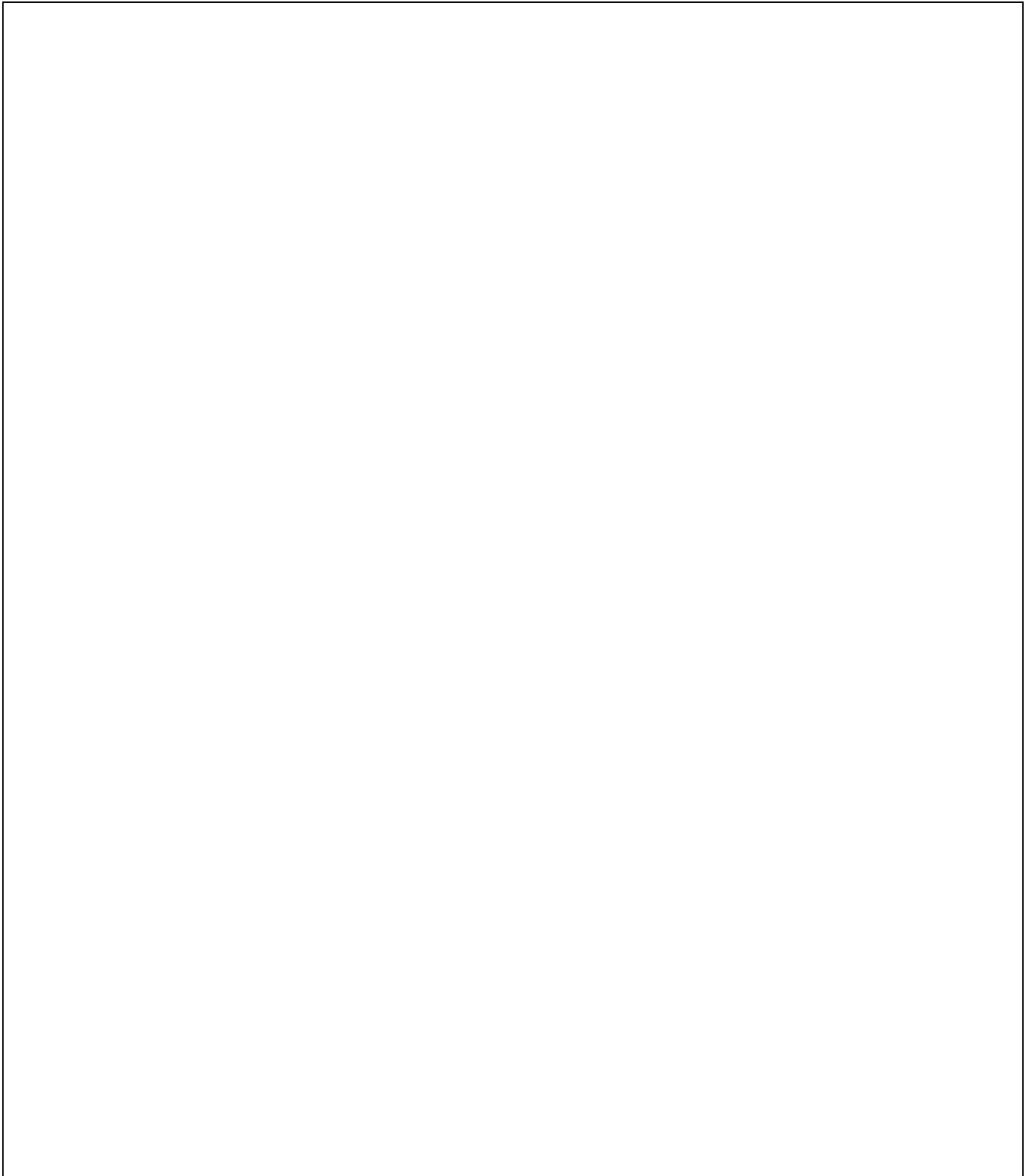
X2	X1	Theoretical	Measured

Implementing  
Gate: .....

X2	X1	Theoretical	Measured

Table 3: Lab 1 – Section 6.4. Part 3.3

A	B	C	D	AB + CD Theoretical (Fill in this part before coming to lab.)	AB + CD (Measured in Lab)



*Figure 24: Wiring connection Section 6.4 part 3.*

---

## 7 LABORATORY 2 – COMBINATIONAL CIRCUITS

---

### 7.1 LAB MATERIAL:

NI SC 2075

ICs: 74HC00 or 74HCT00, 74HC155, 74HC151 Mux

Lots of wires, pliers

### 7.2 CONCEPT:

In the second laboratory, you will experiment with various combinational circuit designs. You will be using the VI script used in lab 1 to send signals and receive signals from your circuits. In this lab, you will

- Design and realize a circuit from beginning to end. (Majority voter circuit)
- Design a circuit with only NAND gates.
- Investigate the logic behavior of various IC's.
  - 74HC151 Multiplexer
  - 74HC155 Decoder

### 7.3 PREPARATION

1. You should get the datasheets of each IC from either the web or a databook.
2. In all your circuits, you should show the whole circuit including pin, ground and  $V_{cc}$  connections, etc. You may either draw your circuit by hand or use a printout obtained from your Multisim design. If you use Multisim, you should simply attach the printout obtained.
3. *Majority Logic:* A majority logic circuit is a digital circuit whose output is equal to 1 if the majority of inputs are 1's. The output is 0 otherwise. Design a 3-input majority circuit using NAND gates with minimum number of IC's. Use the space provided in Figure 25 to draw your circuit. Fill in the theoretical part of Table 4.
4. *Decoder Implementation:* Consider the following three input combinational circuit with inputs  $x_1$ ,  $x_2$  and  $x_3$ . The circuit has two outputs  $f_1$  and  $f_2$  defined as follows:

$$f_1 = x_1x_3 + x_1'x_2'x_3'$$

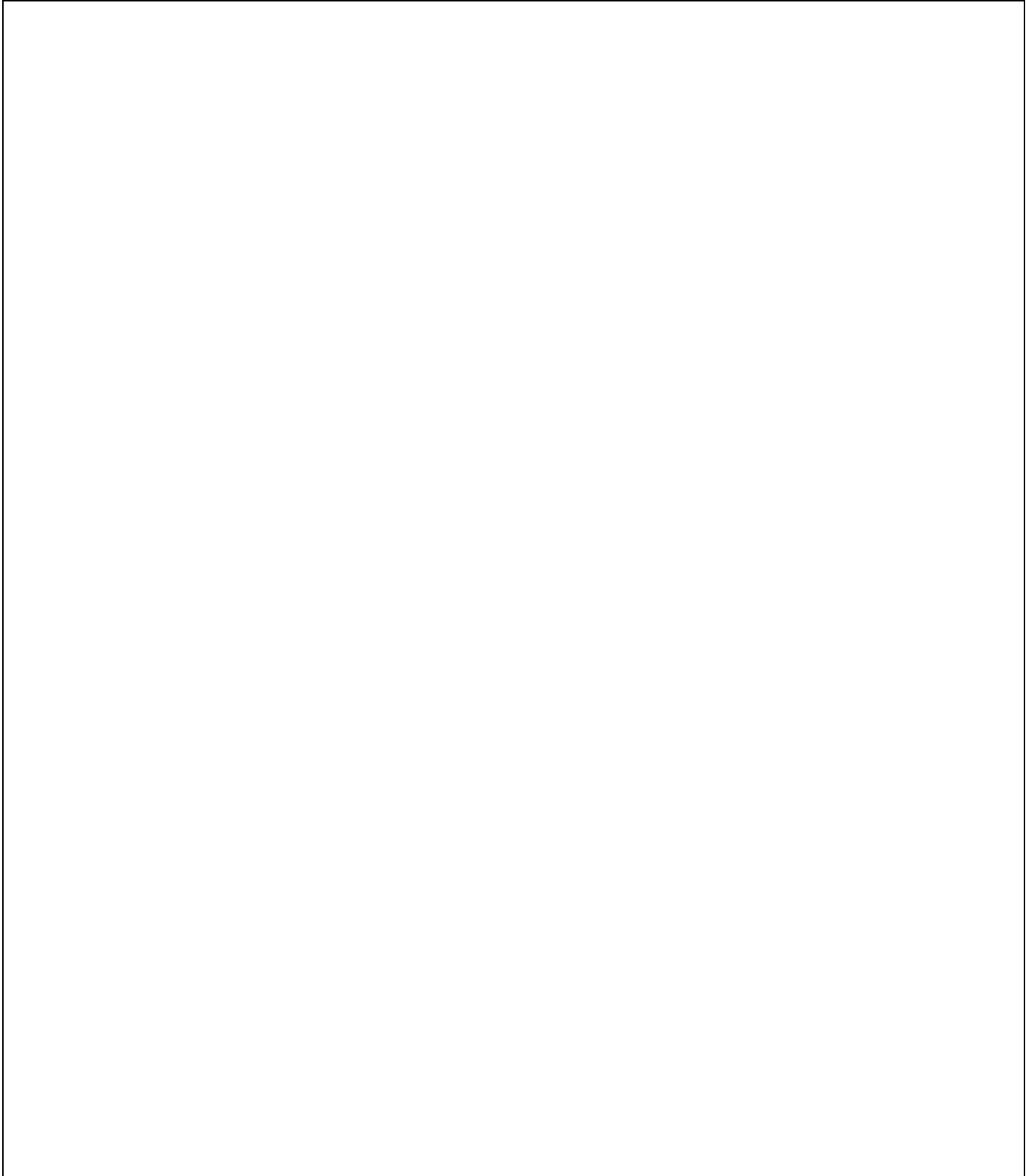
$$f_2 = x_1'x_2 + x_1x_2'x_3'$$

Implement using a 74HC155 decoder IC and external NAND gates. Use the space provided in Figure 26 to draw your circuit. Fill in the theoretical part in Table 5 and Table 6.

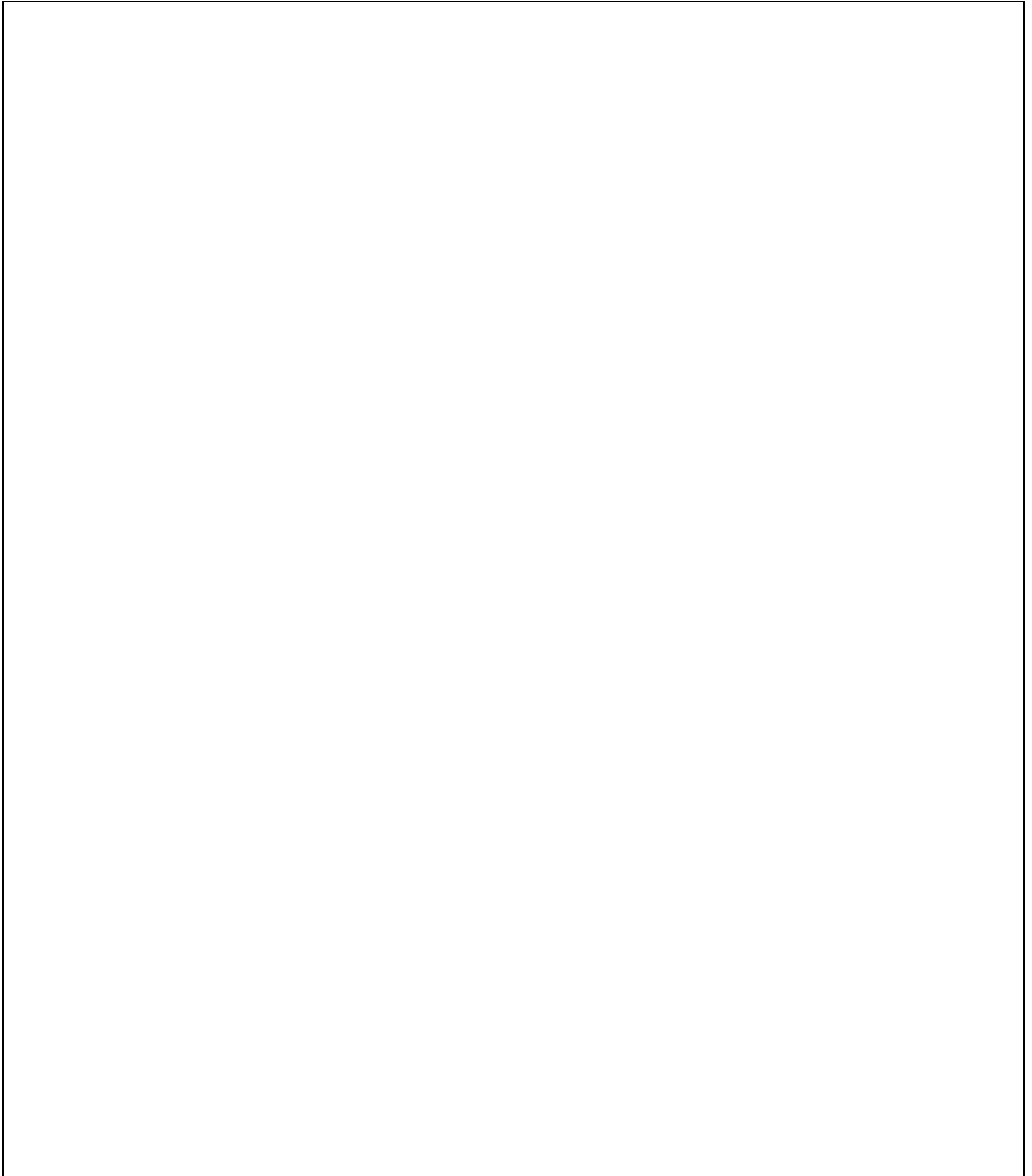
5. *Multiplexer Implementation:* In this part, you will design the same circuit with a multiplexer –74HC151 IC. Design the above given  $f_1$  and  $f_2$  functions using the multiplexer IC. Use the space provided in Figure 27 to draw your circuit.

#### **7.4 IN LAB:**

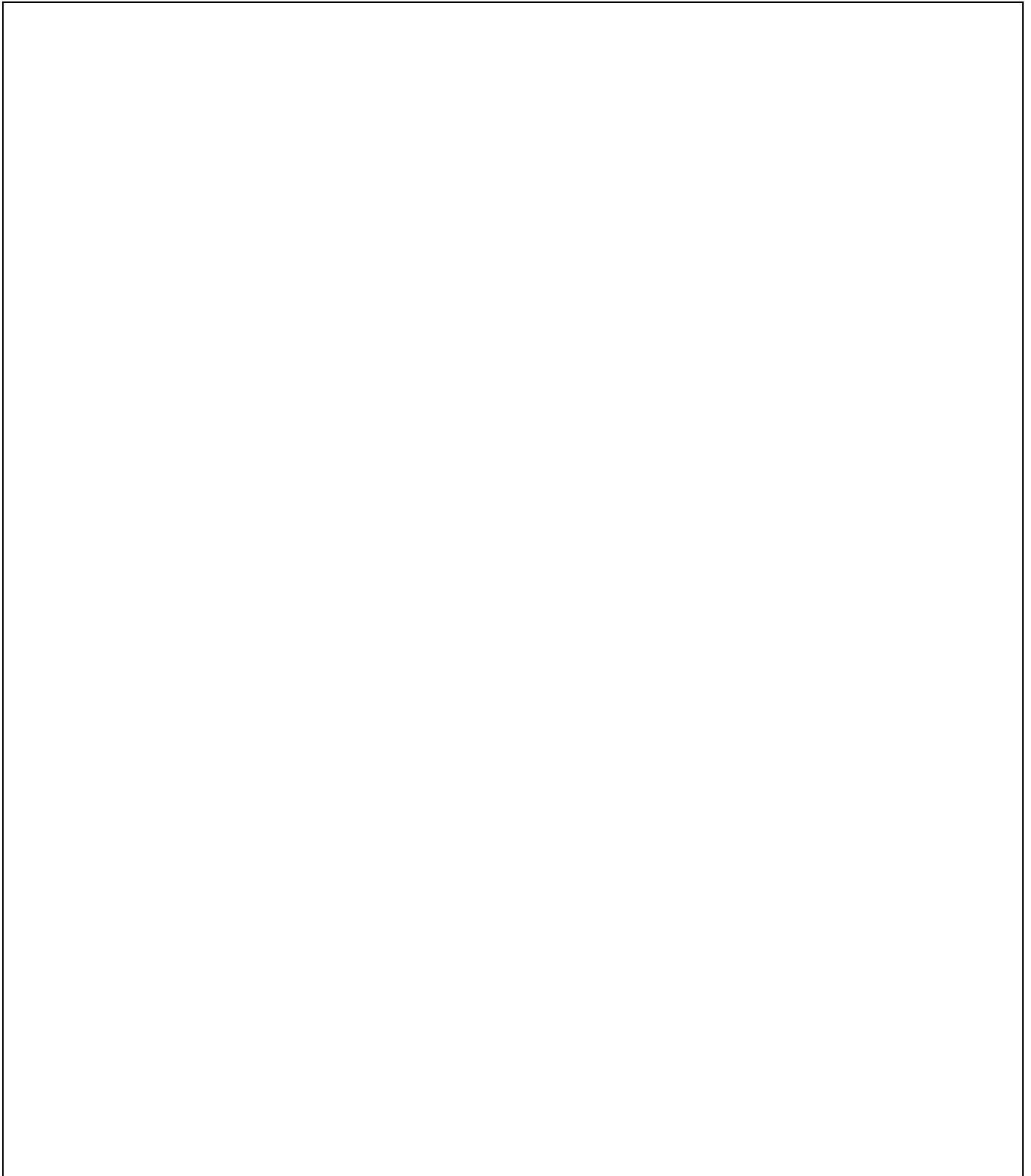
1. You should use the VI script to input signals to your circuit and to get output signals from your circuit.
4. Exhaustively test the 3-input majority circuit as designed. Fill in the measured part in Table 4.
5. Exhaustively test your circuit implemented to realize the  $f_1$  and  $f_2$  functions with a decoder. Fill in Table 5.
6. Exhaustively test the circuit you have implemented to realize  $f_1$  and  $f_2$  functions with a multiplexer. Fill in Table 6.



*Figure 25: Majority voter circuit using NAND gates.*



*Figure 26: Implementation of functions using a decoder*



*Figure 27: Design of  $f_1$  and  $f_2$  function using a multiplexer.*



Table 4: Testing “Majority Voter” circuit.

<b>I<sub>2</sub></b>	<b>I<sub>1</sub></b>	<b>I<sub>0</sub></b>	<b>Majority Voter Theoretical (Fill in this part before coming to lab.)</b>	<b>Majority Voter (Measured in Lab)</b>

Table 5: Testing “Boolean functions with decoder” circuits.

<b>x<sub>2</sub></b>	<b>x<sub>1</sub></b>	<b>x<sub>0</sub></b>	<b>f<sub>1</sub> (Fill in this part before coming to lab.)</b>	<b>f<sub>1</sub> (Measured in Lab)</b>	<b>f<sub>2</sub> (Fill in this part before coming to lab.)</b>	<b>f<sub>2</sub> (Measured in Lab)</b>

Table 6: Testing “Boolean functions with mux” circuits.

<b>x<sub>2</sub></b>	<b>x<sub>1</sub></b>	<b>x<sub>0</sub></b>	<b>f<sub>1</sub> (Fill in this part before coming to lab.)</b>	<b>f<sub>1</sub> (Measured in Lab)</b>	<b>f<sub>2</sub> (Fill in this part before coming to lab.)</b>	<b>f<sub>2</sub> (Measured in Lab)</b>

## 8 INTRODUCTION TO XILINX FPGA & PROJECT NAVIGATOR

Each lab workbench has a programmable logic board. The logic board is made by the Digilent corporation as shown in Figure 28: This board holds the Xilinx Spartan VI FPGA chip among other additional resources such as switches, indicators, etc.

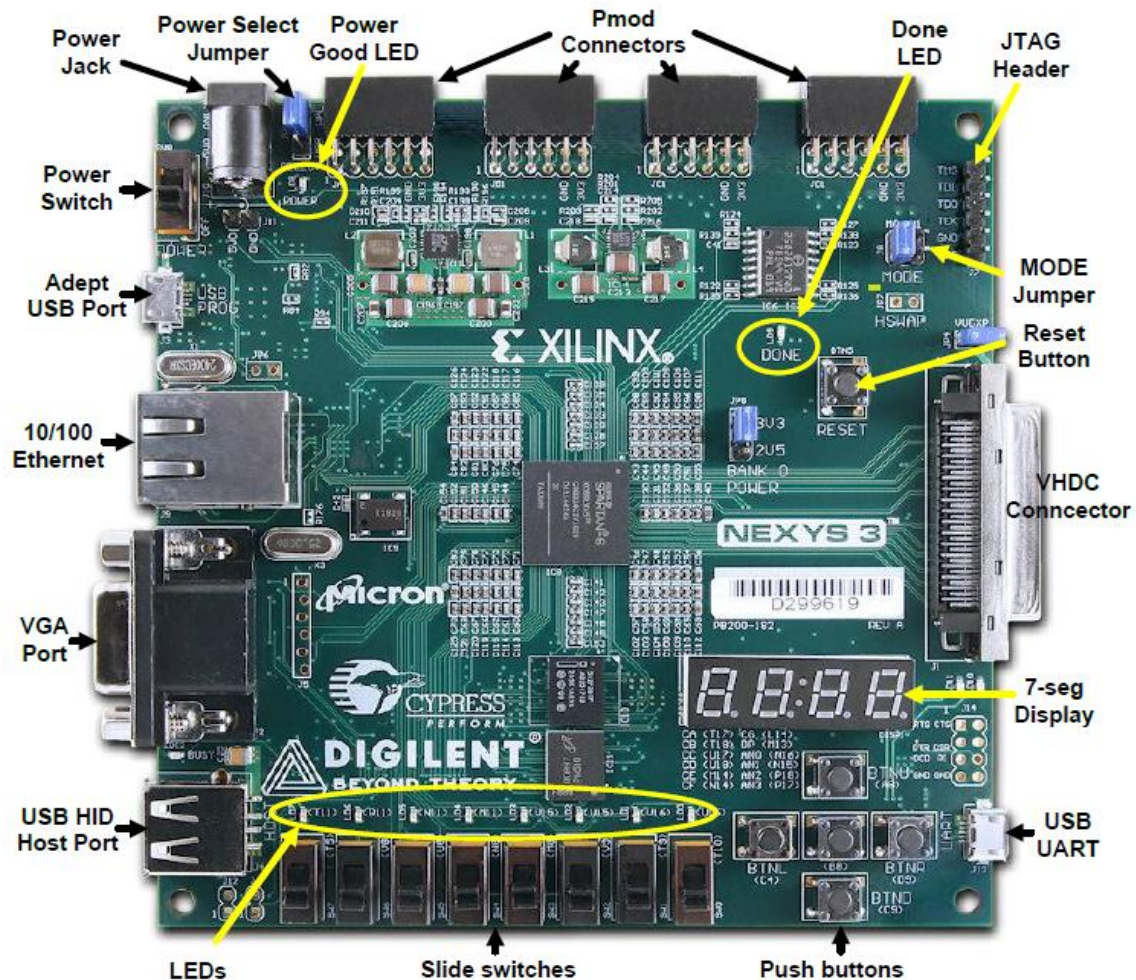


Figure 28: Digilent Nexys3 Board holding Xilinx Spartan VI FPGA Chip [8].

### 8.1 HARDWARE

FPGA board provides a rich environment for prototyping digital circuits of substantial complexity. A wealth of information is accessible from the Digilent corporation web site: (<http://www.digilentinc.com/Products/Detail.cfm?NavPath=2.400.897&Prod=NEXYS3>). For purposes of EE240 lab, we highlight here only those features that are directly relevant to the design experiments you will be conducting.

### 8.1.1 ON THE NEXYS3 BOARD

The main component on the NEXYS3 board is a Xilinx field-programmable gate array (FPGA) integrated circuit (IC) chip Xilinx XC6LX16-CS324. This chip is built using 45 nm process technology. It is optimized for low-cost, high performance and low power consumption. It has special features such as digital signal processing systems for video, wireless and many other applications, embedded processing, integrated memory blocks and controllers, ...etc. This IC chip consists of thousands (147,000 exactly) of logic building blocks that can be “wired” according to the pattern of your specific design. This is achieved by appropriately opening and closing tiny electronic switches. Fortunately, the interfacing software hides all of these details. All you have to do is to download your design, which then leads to all the programmable switches to be turned on or off accordingly. Hence, seconds after you issue the command to download your design, the FPGA becomes the hardware embodiment of that design. Always “play” with your design. The logic board contains a variety of switches (for input) and indicators (for output) to help you apply stimuli to and observe responses from the FPGA. You can even control the FPGA from the attached PC and display its responses on the PC screen. Basic input and outputs are shown in Figure 29. These are:

- Four push-button switches.
- Eight slide switches.
- Eight individual Light-Emitting Diode (LED)
- Four seven-segment LED displays.

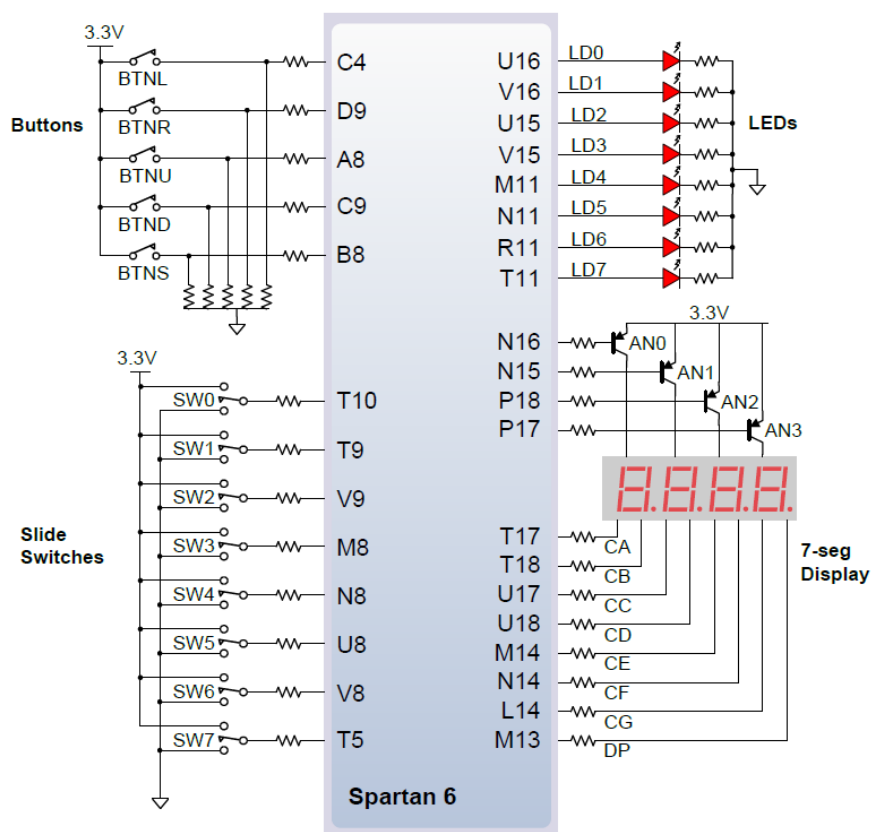


Figure 29: NEXYS3 basic input/output configuration [8].

The manual of the NEXYS3 board is available on the web from the address:  
[http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3\\_rm.pdf](http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf)

## 8.2 THE SOFTWARE

As mentioned earlier, you will be using the Xilinx ISE (Integrated Synthesis Environment) WebPACK CAD tools to enter, simulate, and implement your designs. We do not expect you to become an instant expert in using these tools. Rather, your skills are bound to improve over the course of the semester from repeated use and increased familiarity. The ISE WebPACK tools have extensive online documentation accessible from the Help menu. Keeping in mind the availability of such documentation, it is still helpful to elaborate the design steps outlined earlier to make the whole process more concrete and less intimidating. The design flow consists of five main stages.

### 8.2.1 DESIGN ENTRY

There are basically two methods to capture a design:

- (a) by drawing a schematic diagram showing the design's components and how they are interconnected; and
- (b) by writing a text document that describes the structure and behavior of the design in some appropriate notation.

We will use both methods of design entry in this lab.

The *Project Navigator*, which is the primary user interface of the ISE WebPACK software, allows one to create a design by instantiating components from one or more libraries and by wiring them together. It is similar to the Multisim simulation tool. The types of components we will be using ranges from simple logic gates to more complex building blocks (such as arithmetic units and data steering logic.)

The Project Navigator also allows design entry using HDLs, where HDL stands for Hardware Description Language. HDLs are analogous to the programming languages used to write general-purpose software, but are usually augmented with features geared specifically for describing hardware. The Project Navigator supports three of such languages: Verilog, VHDL, and ABEL. The first two have powerful constructs that are essential when you are designing complex hardware; they are the primary HDLs used by logic designers in the semiconductor and computer industries. On the other hand, ABEL, which stands for *Advanced Boolean Equation Language*, is a fairly simple language suitable for small designs; it will be more than adequate for our purposes. Whichever method of design entry you choose (schematic or HDL), you will need to specify your design's inputs and outputs and how you would like them to be mapped to the switches and indicators on the logic board.

### 8.2.2 FUNCTIONAL SIMULATION

One of the advantages of computer-aided design is that the correctness of a design can be studied before implementing it in hardware. Basically, the software model of the design (the schematic or HDL description) is run by applying various inputs to it and observing the resulting outputs. This process is referred to as *simulating* your design. The simulators can be used to check many aspects of a design. For digital design, the two most common classes of simulators are a) *functional* simulators for checking the

logical correctness of the design, and b) *timing* simulators for verifying that the design works at the required speed. We focus almost exclusively on verifying functionality, and only briefly explore timing simulation to get some appreciation of the temporal behavior of logic circuits.

The ISE WebPACK tools provide a functional logic simulator that can be invoked after a design has been entered. The simulator enables one to define *stimulators* that you can apply to the inputs of your design, and to observe the resulting logic waveforms on the outputs. Any anomalies detected in this process should be diagnosed to determine their cause, and the design should be modified to eliminate them (by switching back to either the Schematic or HDL Editors). When the designed circuit behaves according to its specification, one is ready for the implementation stage.

### 8.2.3 IMPLEMENTATION

Implementing a design in hardware takes different routes depending on the nature of that hardware. The choices range from off-the-shelf IC chips that must be physically wired together, to custom-designed ICs that are manufactured in a semiconductor fabrication facility, to a whole range of programmable ICs with varying degrees of flexibility. You have already gained some experience with the first two approaches in the first two labs. In the remaining labs, you will also get familiar with using programmable hardware. For this, the primary implementation medium is the Xilinx XC6LX16-CS324 FPGA IC chip.

An FPGA can be programmed to implement a particular design by downloading to it a “configuration” file that tells it how to set the state (on or off) of tiny electronic switches (transistors). Hence the structure and function of the chip is made to match those of the given design. The process of generating this configuration file for a given design is quite involved, but that need not concern us here. The Project Navigator tools conveniently take care of this. What you do need to know is that the resulting configuration file is a binary file with a “.bit” extension; it is also referred to as the *bitstream* file.

You can find helpful tutorials about the usage of Xilinx ISE WebPACK tools for design entry and simulation on the links given below:

- (a) [http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx14\\_4/ise\\_tutorial\\_ug695.pdf](http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/ise_tutorial_ug695.pdf)
- (b) <http://www.digilentinc.com/Data/Documents/Tutorials/Xilinx%20ISE%20WebPACK%20VHDL%20Tutorial.pdf>

And you can refer to the address:

[http://www.xilinx.com/support/documentation/data\\_sheets/ds162.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds162.pdf) for the datasheet of the Xilinx’s Spartan VI FPGA family that the XC6LX16-CS324 FPGA chip belongs to.

### 8.2.4 PROGRAMMING

The actual programming of the FPGA involves downloading the bitstream file from the PC to the logic board. This is accomplished by invoking a utility program from the Digilent Corp. called Adept as shown in Figure 30. This should take only a few seconds.

To program the Nexys3 board using Adept, first set up the board and initialize the software as follows [8]:

- Plug in the USB cable to the PC and to the USB port on the board (this should be done by the TA or the lab manager)
- Start the Adept software



- Turn on Nexys3's power switch
- Wait for the FPGA to be recognized.

Use browse on the Adept user interface to find the desired .bit file, and click on the Program button. This sends the configuration file to the FPGA. The user interface tells whether programming was successful or not. The configuration-done LED on the FPGA board turns on if the FPGA is configured successfully [8].

Adept program makes sure that the selected configuration file contains the correct FPGA ID code before starting the programming. This prevents incorrect .bit files from being sent to the FPGA [8].

The Config interface of the Adept program also provides an “Initialize Chain button”, console window, and status bar. If USB communication with the board is interrupted, the Initialize Chain button becomes useful. The console window shows current status, and the status bar displays real-time progress when downloading a configuration file [8].

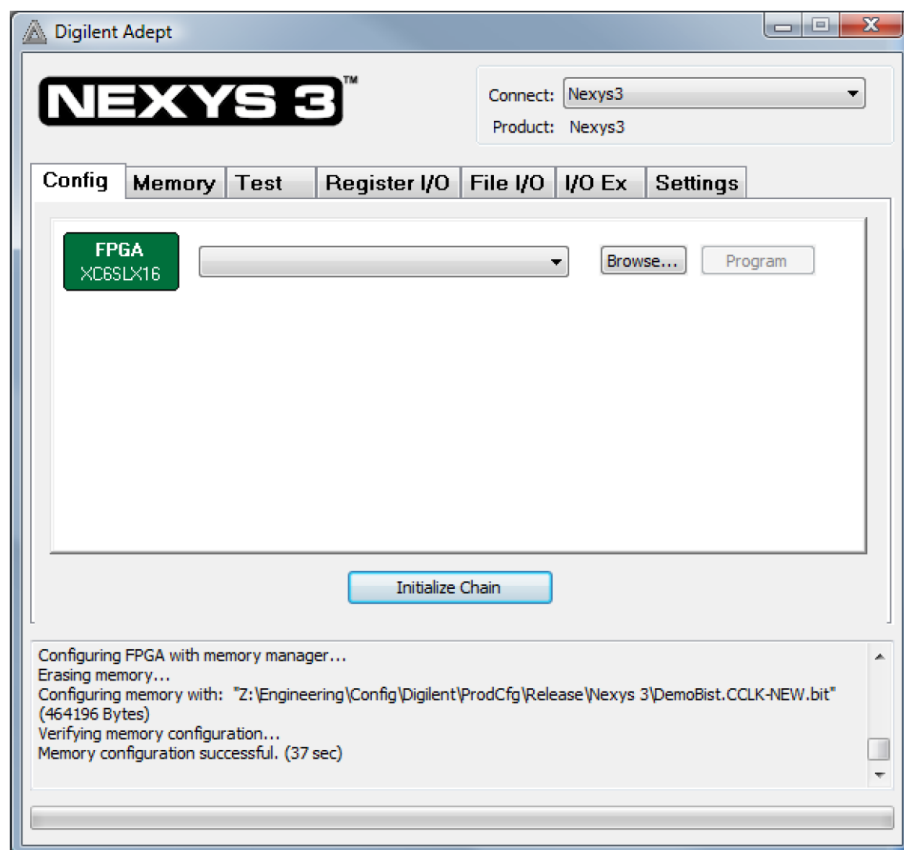


Figure 30: Digilent's Adept Program to program the FPGA chip [8].

### 8.2.5 DEBUGGING

At this point you have a “live” design on the logic board and you are ready to check it by applying inputs using the slide and push button switches and observing outputs on various LEDs. You can also apply inputs to your design from various ports.

You can find a manual for the usage of utilities from the Digilent Corp. on the link given below: [http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3\\_rm.pdf](http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf)

## 9 LABORATORY 3 – SEQUENTIAL CIRCUITS

### 9.1 LAB MATERIALS:

NI SC-2075,  
ICs: 74HC00, 74HC74  
Lots of wires

### 9.2 CONCEPT:

In the third laboratory, you will experiment with D flip-flops. In this lab, you will

- Design and realize a D latch circuit using NAND gates.
- Design and realize a 2-bit shift register using 74HC74 D-flip-flop IC.
- Repeat an 8-bit shift register implementation in FPGA that is already prepared.

The lab will consist of two parts:

1. In the first part, you will design circuits using off-the-shelf IC's.
2. In the second, you will use an FPGA board and experiment with the same circuits but now implemented by FPGA programming.

### 9.3 PREPARATION

1. You should get the datasheets of each IC from either the web or a databook.
2. In all your circuits, you should show the whole circuit including pin, ground and Vcc connections, etc. You may either draw your circuit by hand or use a printout obtained from your Multisim design. If you use Multisim, you should simply attach the printout obtained.
3. Design a clocked D latch with four NAND gates. Construct your circuit using 74HC00 ICs and verify its characteristic table. Use the space provided in **Figure 40** to draw your circuit. Fill in the theoretical part of Table 8.
4. IC type 74HC74 consists of two D positive-edge-triggered flip-flops with preset and clear. Obtain a copy of its datasheet before you come to the laboratory - including pin assignment and function table. Fill in the theoretical part of Table 9.

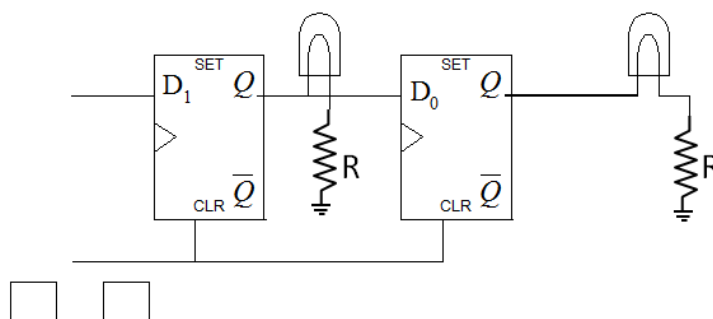


Figure 31: 2-bit shift register.

5. Now design a two-bit shift register as shown in Figure 31. Put your design in space provided in Figure 41. Fill in the theoretical part of Table 10.

6. Please think how you can obtain the results in Part 12a, 12b and 12c of Section 9.4.2 and fill in Table 7. On this table:

- *Input* corresponds to your data that you will enter in the shift register in the next clock cycle (The value in the Input column is written to the Q0 column in the next clock cycle).
- Q0 to Q7 correspond to the internal state of the 8 bit shift register (e.g. Q4 is the output of the 5th D flip-flop of the shift register), with Q7 the most significant bit corresponding to LD7 on Nexys3 Board.
- Each part in this experiment starts with an initial state which is the final state obtained in the previous part. So for example after filling the upper parts of the table as required by part (a), you will use the result of part (a) as the initial state for part (b). And so in order to show the beginning of part (b) you will put a 0 on the clock cycle index column for part (b) corresponding to its initial state and restart your count accordingly.

## 9.4 IN LAB

### 9.4.1 WITH IC COMPONENTS

1. Implement the clocked D latch circuit as designed in Section 9.9.3 and verify its characteristic table by filling in the in-lab part of Table 8.
2. Verify the characteristic table of 74HC74 by testing it exhaustively. Fill in the in-lab part of Table 9.
3. Implement the 2-bit shift register as designed in Section 9.3.5 . Initialize  $D_1$  to 1 and  $D_0$  to 0. Verify that your circuit functions as a shift register. Fill in the in-lab part of Table 10.
4. Now, remove the input line to  $D_1$  and connect the output of  $D_0$  flip-flop to  $D_1$ . What does your circuit do?

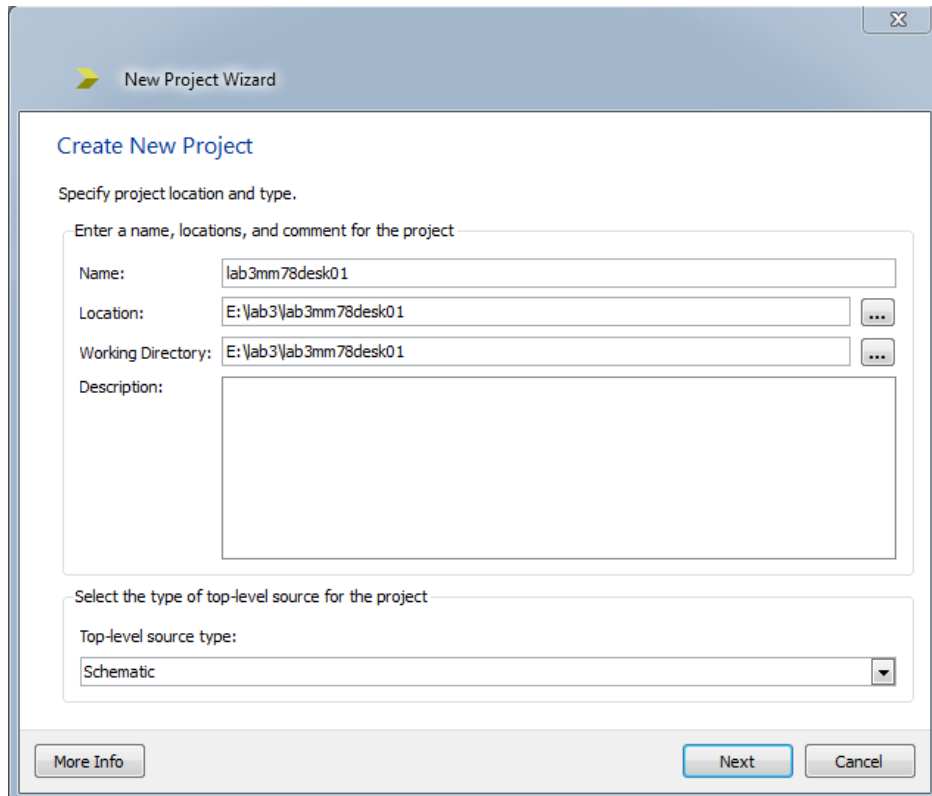
### 9.4.2 USING FPGA BOARD

In this part of the lab, you will be using an FPGA card to implement a shift register. In order to accomplish this, one must have the software code that implements the circuit. In this lab, you are given with a pre-designed and written code that realizes this. You must follow the following steps in order to download this code onto your FPGA card.

- 1.) The first step is to open the software code that implements the shift register. In order to do this, you must start a new project as follows:
  - a. Click on the shortcut of Project Navigator on the desktop.
  - b. From File → New Project, first choose your Project Location by clicking on '...' and selecting your file as 'E:\EE240\lab3'.
  - c. Enter your project name as 'lab3<your\_section><your\_desk>' such as 'lab3mm78desk01' and select ' Top-Level Model Type ' as 'Schematic' from the box as shown in Figure 32. Click next.
  - d. Make the appropriate changes from the new dialog box as shown in Figure 33 (Make sure you have selected every value correctly!). Click next.



- e. Click next without doing any changes on the next 2 dialog boxes that will open consecutively. Finally the dialog box in Figure 34 will appear. Click finish.



**New Project Wizard**

### Create New Project

Specify project location and type.

Enter a name, locations, and comment for the project

Name:

Location:  ...

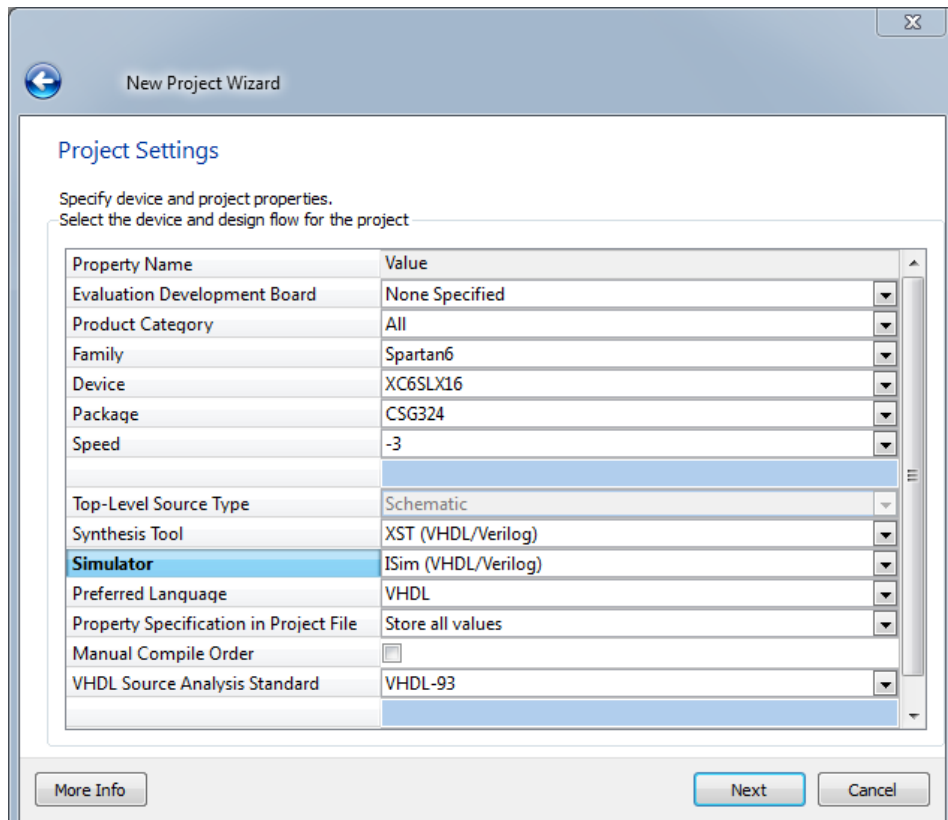
Working Directory:  ...

Description:

Select the type of top-level source for the project

Top-level source type:

Figure 32: Starting a new project.



**New Project Wizard**

### Project Settings

Specify device and project properties.

Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan6
Device	XC6SLX16
Package	CSG324
Speed	-3
Top-Level Source Type	Schematic
Synthesis Tool	XST (VHDL/Verilog)
<b>Simulator</b>	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93

Figure 33: Setting up project environment.

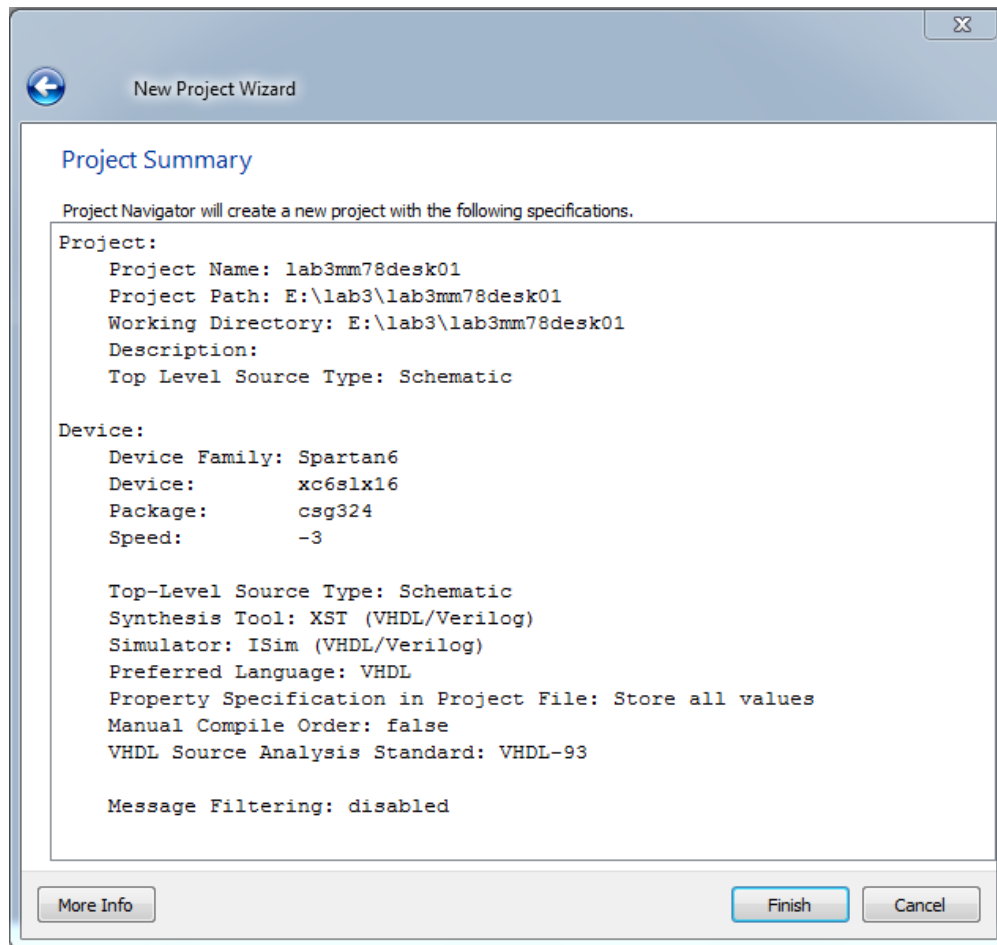


Figure 34: Finalizing the project environment setup.

- 2.) You have now created a new project. Next you should copy the two already prepared files to your Project. To do this, click Project → Add Copy of Source. Select the files 'lab3sr8.sch' and 'lab3sr8.ucf' at once from the directory 'E:\EE240\lab3'. Then click open. The files should be seen at top left side of the screen as shown in Figure 35.

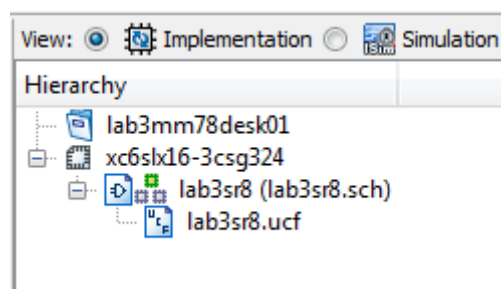


Figure 35: Project Navigator File Manager.

You can see the circuit you will use by double clicking on the 'lab3sr8.sch' file shown in the window in Figure 35. A new window will open and the circuit will look like that of Figure 36. It is an 8-bit shift register constructed by D flip-flops (activated with the rising edge of the input clock signal). This is a circuit similar to that you have implemented in Section 9.3.5, but contains 8 D flip-flops instead of 2. Do not try to modify the circuit! You have now the schematic for the circuit you are to program. Note that there are 2 inputs and 8 outputs of the circuit.

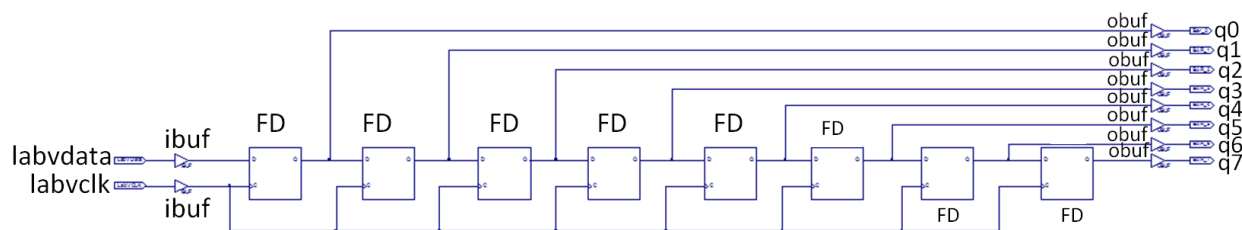


Figure 36: 8-bit Shift Register.

- 3.) Next, you should transform your schematic to a format which can be downloaded to your FPGA card. This format is known as bitstream format. This is achieved through a series of steps as explained below. Make sure that you run each step carefully. You should run synthesis, implement design and generate programming file processes.
- 4.) First, you need to “run your synthesis”. To do so, choose “Synthesize - XST” menu from the Processes for Source menu, right click to have a pop-up menu and click ‘Run’ as shown in Figure 37.

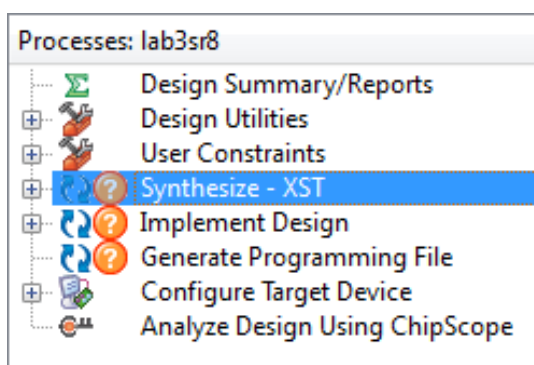


Figure 37: Running the synthesis.

- 5.) Next, you need to “Implement design” in a similar manner.
- 6.) Finally, you need to “Generate programming file” in a similar manner.
- 7.) For each of these steps, it will take some time for the computer to do the necessary operations and after all a green tick or a yellow exclamation mark should be seen near the names of each process as shown in Figure 38.

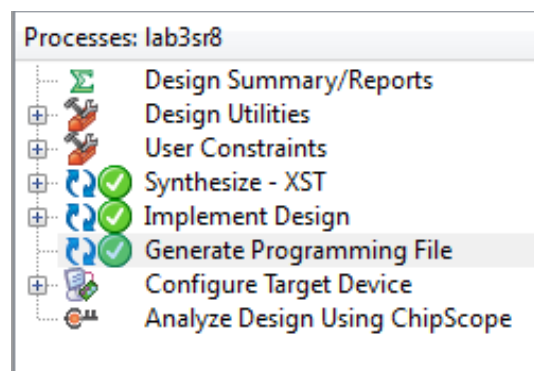


Figure 38: Generating the programming file.

- 8.) You have now successfully completed the schematic transformation stage and have now a bitstream code that is ready to be downloaded to the FPGA card. This new code is located in a new directory at 'E:\EE240\lab3\lab3mm78desk01'. You have generated a bit file in that directory.
- 9.) Now you can start working on the FPGA Card to realize the 8 bit shift register design. First plug in the USB cable to the PC and to the USB port on the board. Start the Adept software by clicking on the shortcut of Digilent Adept utility on the desktop. Turn ON Nexys3's power switch. Wait for the FPGA to be recognized. Check the Adept software window if everything is all right with your Nexys3 board. In our case USB port is used for both programming and application of power to your card (TAs must make sure that the Jumper JP1 (power select jumper near the power jack) on your board selects USB port). For applications that require more power than what USB port can supply, a separate power supply must be used.

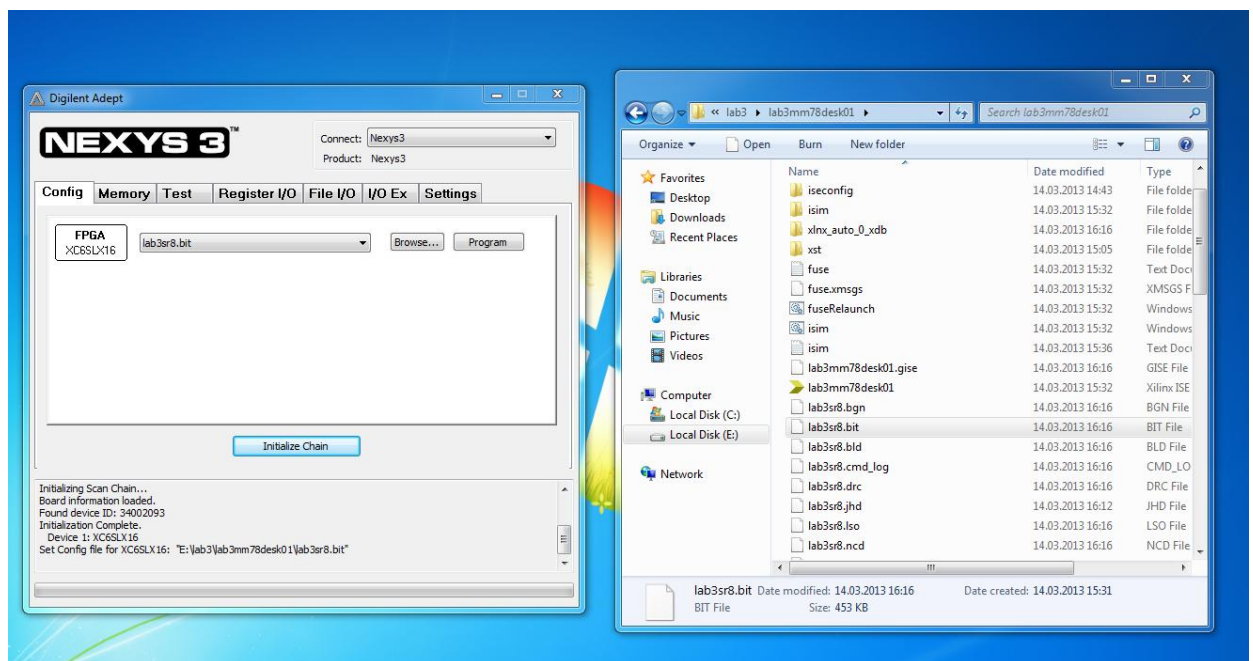


Figure 39: Using the Digilent Adept utility.

- 10.) Now you can download the bitstream file that you have generated in part 8 to the FPGA of your Nexys3 board. To do this:
- On the Digilent Adept program menu, on the config tab, click Browse and select your bitstream file as shown in Figure 39.
  - Click on Program to program Nexys3 board.
- 11.) One slide-switch and one push-button on the Nexys3 board are configured via universal constraints file (lab3sr8.ucf) so that you can send clock and data signals to the 8-bit shift register in the FPGA.

Clock signal : PushButton1  
Data signal : SlideSwitch1

Please note that the output of every D flip-flop in the design is connected to an individual LED on the Nexys3 board. (Please refer to Figure 30 for identifying the LEDs on the board.) This is accomplished via lab3sr8.ucf file given to you. You do not have to worry about this at the moment. You will be able to observe the internal state of the 8-bit shift register directly from 8 individual LED. The output of the first D flip-flop is connected to the LED named LED<0> (the least significant bit) and the output of the shift register (the output of the 8th D flip-flop) is connected to the LED named LED<7> (the most significant bit). Each LED emits light for the logic high value in the corresponding D flip-flop output.

12.) To give input to your 8 bit shift register, you will use the slide-switch for the data signal and the push-button for a manual clock pulses.

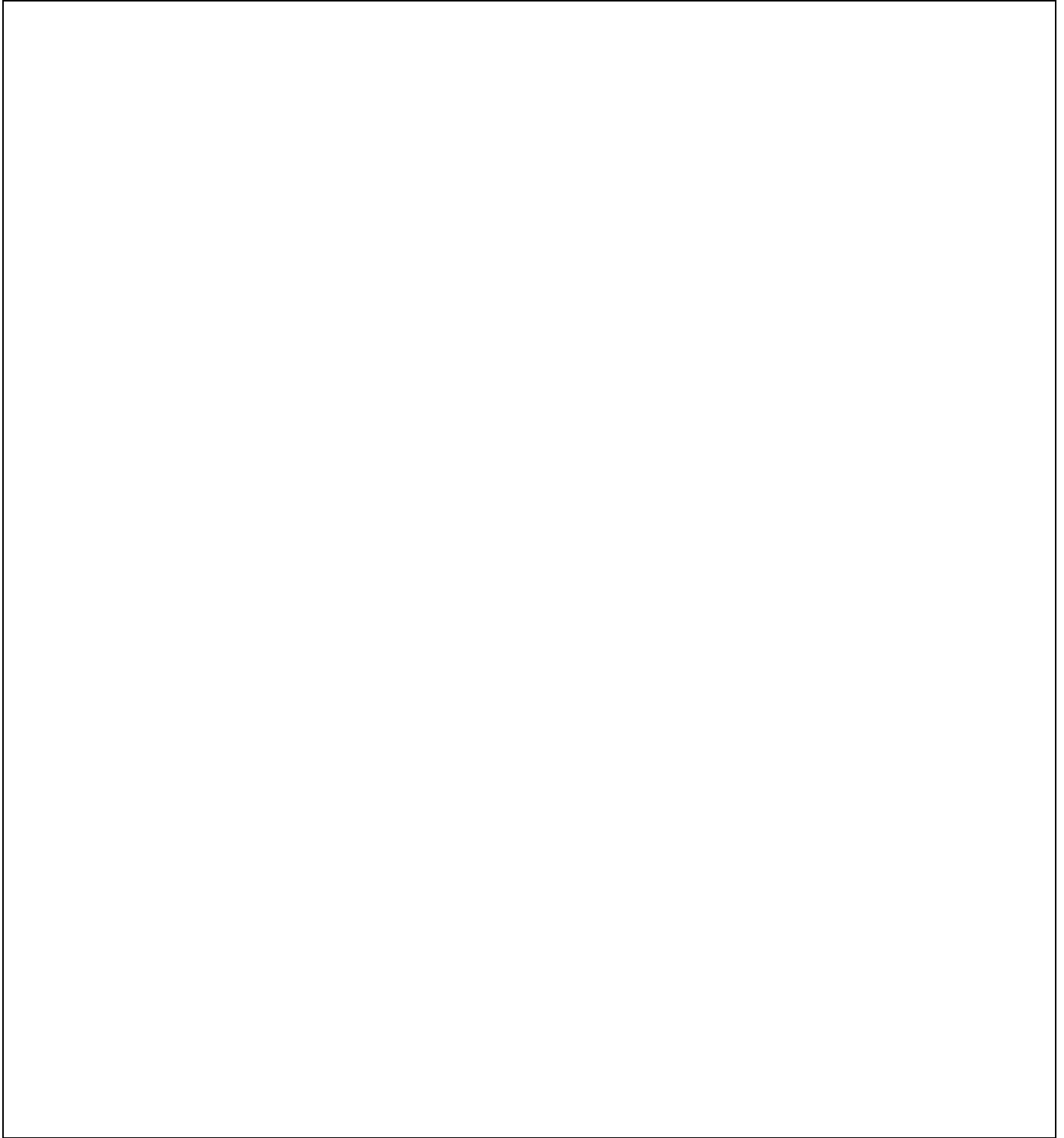
(a) After becoming familiar with the experiment environment; initialize the output of all D flip-flops to logic 0. Then, enter the 4-bit binary equivalent of your *lab group number* in the first 4 flip-flops of the 8-bit shift register using manual clock pulses (be careful about the order of the least and most significant digits).

(b) Obtain the binary equivalent of the number on the 8 LEDs corresponding to the product of your lab group number and 2 with minimum number of manual clock cycles.

(c) Multiply the number resulted in the previous step, by 8 and add 5 to it. Again obtain the binary equivalent of the resulting number on the 8 LEDs with minimum manual clock pulses.

Table 7: Expected Results For The Experiment Of 8-Bit Shift Register

<b>a</b>	<b>b</b>	<b>C</b>	<b>8-bit Shift Register</b>								
<i>Clock-Cycle Index</i>			<i>Clock</i>	<i>Input</i>	Q0	Q1	Q2	Q4	Q5	Q6	Q7
0			1		0	0	0	0	0	0	0
			0								
1			1								
			0								
2			1								
			0								
3			1								
			0								
4			1								
			0								
5			1								
			0								
6			1								
			0								
			1								
			0								
			1								
			0								
			1								
			0								
			1								
			0								
			1								
			0								
			1								
			0								
			1								
			0								
			1								
			0								
			1								
			0								
			1								
			0								
			1								
			0								



*Figure 40: A Clocked D Latch Using NAND gates.*

Table 8: Truth table for D Latch constructed from NAND gates.

CP	Q(t)	I	Q(t+1) (Fill in this part before coming to lab.)	Q(t+1) (Tested in Lab)

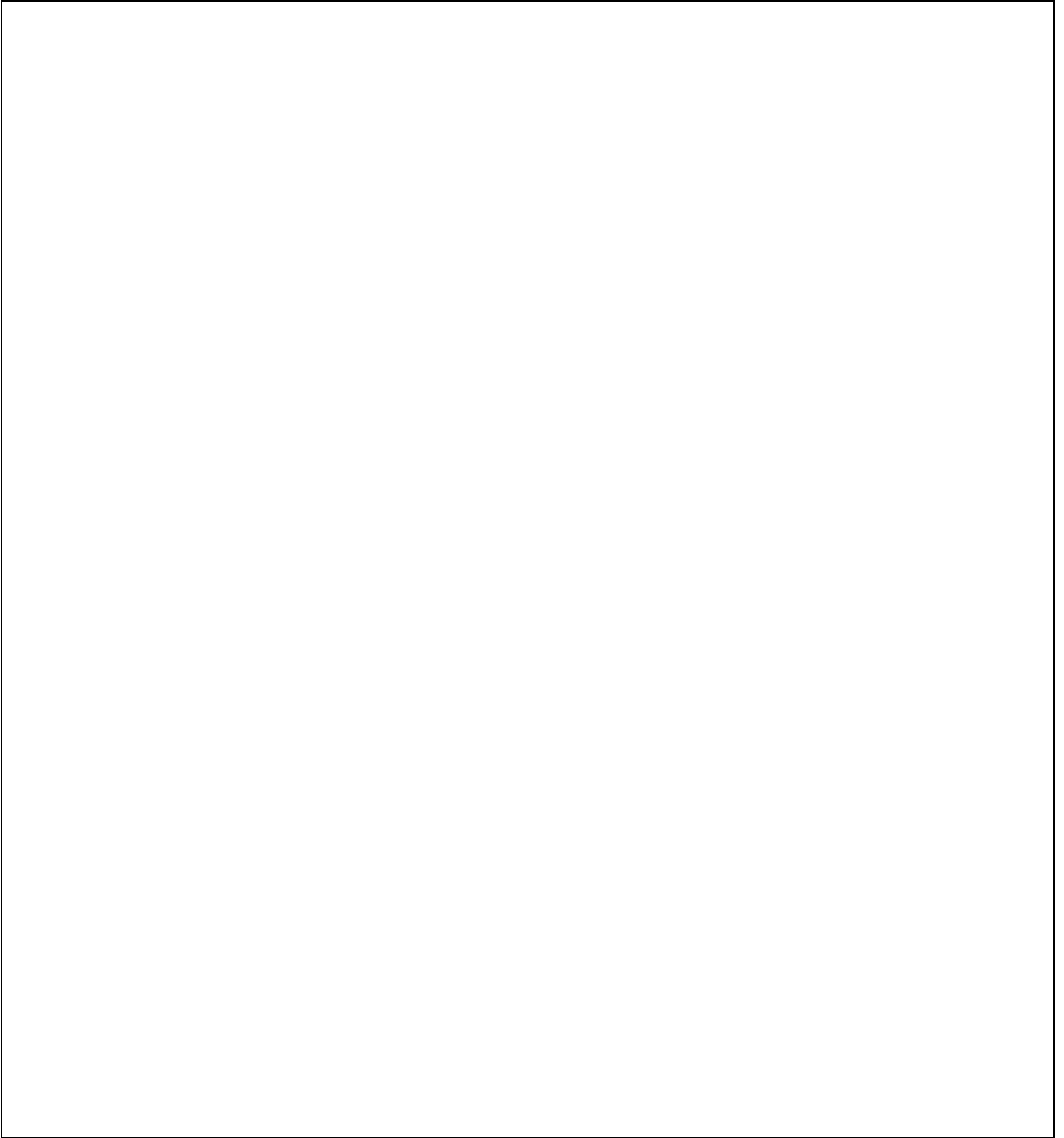
Table 9: Truth table for D Flip-Flop

CP	Q(t)	I	Q(t+1) (Fill in this part before coming to lab.)	Q(t+1) (Tested in Lab)

Table10: Truth table for 2-bit shift register.

CPI	Q <sub>0</sub> (t)	Q <sub>1</sub> (t)	I	Q <sub>0</sub> (t+1) (Fill in this part before coming to lab.)	Q <sub>0</sub> (t+1) (Tested in Lab)	Q <sub>1</sub> (t+1) (Fill in this part before coming to lab.)	Q <sub>1</sub> (t+1) (Tested in Lab)





*Figure 41: 2-bit shift register.*

---

## 10 LABORATORY 4 – COUNTERS

---

### 10.1 LAB MATERIALS

Breadboard  
Lots of wires  
74HC73 dual JK Flip-Flops  
74HC00  
74HC4511  
74HC4510  
Resistances

### 10.2 CONCEPT

In the fourth laboratory, you will experiment with counters. In this lab, you will

- Design and realize various counters using JK IC.
- Design a light sensitive counter using counter IC

The lab will consist of two parts:

- In the first part, you will design circuits using off-the-shelf IC's.
- In the second part, you will use an FPGA board and enter a 4-bit synchronous binary counter design using structural style of VHDL programming. You have entered your design to the FPGA board schematically before in Lab 3 for the 8-bit shift register. Now, you start to use hardware description language to enter your design.

### 10.3 PREPARATION

1. You should prepare the connections for the count-up ripple counter as shown in **Figure 42**. Use area provided in **Figure 46**.
2. You should prepare the connections for the decade ripple counter as shown in **Figure 43**. Use area provided in **Figure 47**.
3. You should prepare the connections for the two-stage counter as shown in **Figure 44**. Use area provided in **Figure 48**.
4. You should prepare the connections for the two-stage counter that is light sensitive as shown in **Figure 45**. Use area provided in **Figure 49**.
5. You will need to write short VHDL code for Section 10.4.7. Read this section carefully. Write the VHDL codes as required. Make sure they work by running them on the Project Navigator. Bring copies of them to the lab.

### 10.4 IN LAB

An asynchronous (ripple) counter is comprised of a series of flip-flops configured with the output of the one connected to the clock input of the next. A signal introduced at the clock input of the first flip-flop will cause the output to change state when the correct edge is detected. This output then triggers the next clock input on the correct edge. In this way a signal at the input will ripple (trigger the next input) from one flip-flop to the

next until it reaches the final flip-flop. A ripple counter has the ability to count from 0 to  $2^n - 1$  (where  $n$  equals to number of flip-flops in the series). The modulus of a counter is the number of different states it is allowed to have. Counter modulus is normally  $2^n$  ( $n$  = the number of flip-flops in the series) unless controlled by a feedback circuits, which limits the number of possible states. In this experiment, basic type of ripple counters will be examined and the definition of modulus will be introduced.

#### 10.4.1 COUNT-UP RIPPLE COUNTER

- A.** Install two 74HC73 dual JK Flip-Flops in the Logic Lab breadboard.
- B.** Construct the circuit as shown. Do not forget to connect series resistances to the LEDs marked as L1, L2 and L3. Do not wire the L4 connection since we do not have space for it on our board. Also connect the J-K inputs of the flip-flops to appropriate logic levels. Otherwise, the flip-flops may give false results or be damaged.

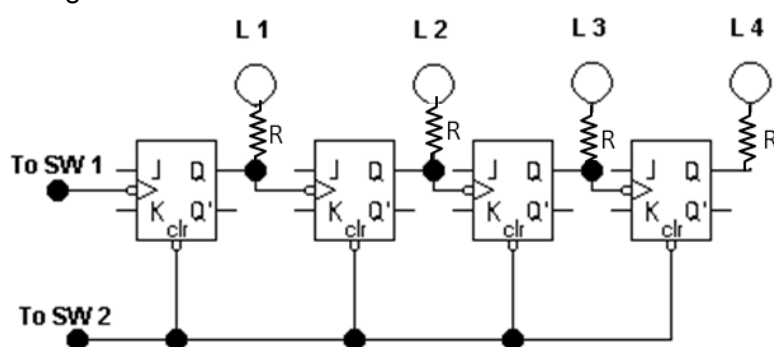
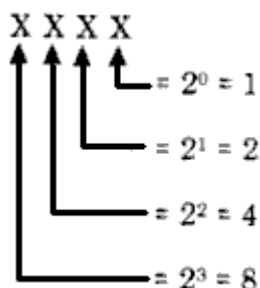


Figure 42: Count-up ripple counter.

- C.** Set data switch SW1 = HIGH (initializes clock settings). Set data switch SW2 LOW to HIGH (clears all flip-flops).
- D.** Set data switch SW1  $\downarrow$ . The 74HC73 Dual JK flip-flop is a negative edge triggered device ( $\downarrow$  = HIGH to LOW). At each transition, record the output indications of L1-L3 and compute the decimal equivalent of each binary number in Table 11.

Note: Data output is an 8421-weighted code.



If a bit is a 1, that column weight is summed with the other column weights in the group. If a bit is 0, that column weight is not included in the sum.

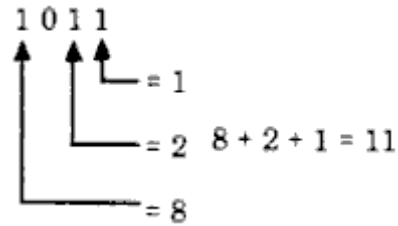
**Example:**

Table 11: Count-up ripple counter.

Count-Up Ripple Counter Output Table					
INPUT	OUTPUTS				
Number of clk ↓	L4 = 8	L3 = 4	L2 = 2	L1 = 1	Decimal equivalent
1	0	0	0	0	0
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					
28					
29					
30					
31					
32					

**Observe:** At transition from 8 = HIGH, 4 = HIGH, 2 = HIGH, 1 = HIGH to 8 = LOW, 4 = LOW, 2 = LOW, 1 = LOW, all flip-flops must change state. This type of condition limits the maximum frequency capability of the asynchronous counter. The input must ripple through all flip-flops in the series thus the next input cannot occur before this has taken place. Note: The counter will count from 0-15.

$$(2^n - 1) \rightarrow (2^4 - 1) = 15$$

The sequence repeats every  $2^n$  transitions.

**Observe:** When you apply a clock signal manually by pressing a switch, can you count one by one properly? Why do you think sometimes the result is increased multiple times even though you press switch once?

#### 10.4.2 DECADE RIPPLE COUNTER

- A.** Install one 74HC00 Quad 2-Input NAND Gate and the Numeric Displays (with socket) in the Logic Lab breadboard. Do not forget to connect series resistances to the LEDs. Do not wire the L4 connection since we do not have space for it on our board. Do not forget to connect the J-K inputs of the flip-flops to appropriate logic levels. Otherwise, the flip-flops may give false results or be damaged.
- B.** Reconfigure the circuit as shown.

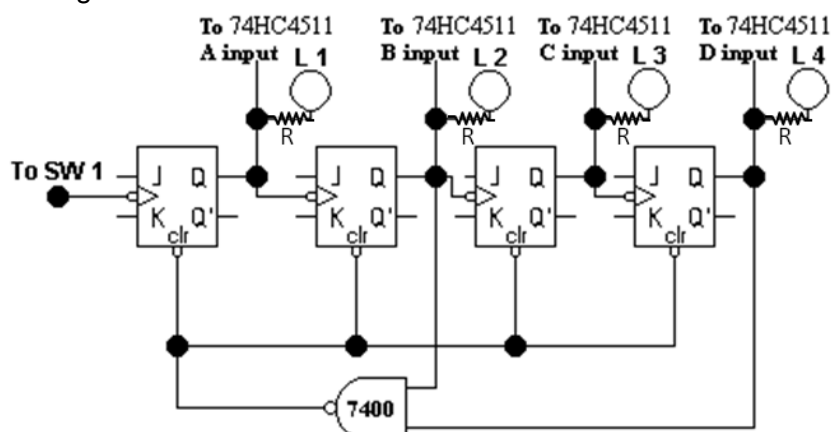
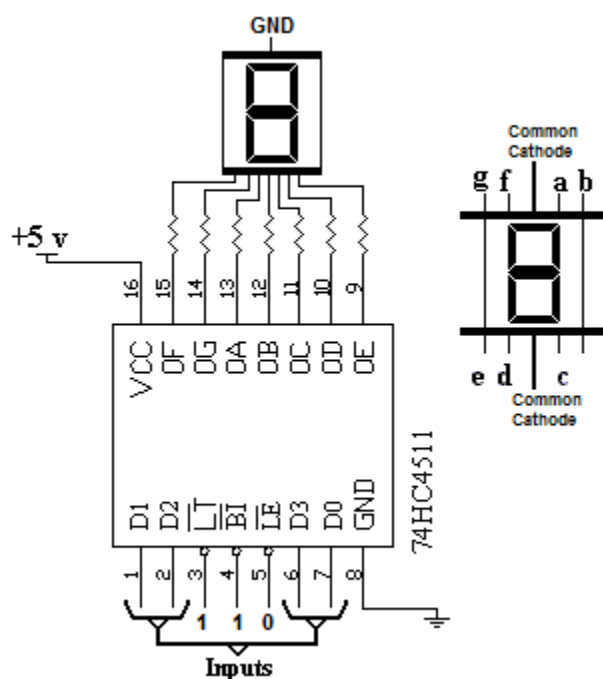


Figure 43: Decade ripple counter.



- C.** Set Data switch SW1 LOW to HIGH (clears all flip-flops).  
**D.** Set data switch SW2 ↓ as shown in the Decade counter output table. Record the output indications of the numeric display and L1-L3 in Table 12.

Table 12: Decade counter.

Decade Counter Output Table					
INPUT	OUTPUTS				
Number of clk ↓	L4 = 8	L3 = 4	L2 = 2	L1 = 1	Decimal equivalent
1	0	0	0	0	0
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					
28					
29					
30					
31					
32					

**Observe:** When the state 8 = HIGH, 4 = LOW, 2 = HIGH, 1 = LOW is reached, it will remain in that state until the NAND gate resets the flip-flops to the 0000 state.

Modulus is defined as the number of different states a counter is allowed to have. The decade counter limits the number of possible states to 10 by controlled feedback. Thus the decade counter has a modulus of 10.

### 10.4.3 TWO-STAGE COUNTER

- A. Insert two 74HC4510 Decade counters and the Numeric Displays (with socket) in the Logic Lab breadboard.
- B. Construct the circuit as shown. Do not forget to connect every input of 74HC4510 ICs to appropriate logic levels. You should find what to connect to “?” marks in Figure 44. Do not leave any input unconnected. Otherwise, unpredictable results may be generated or ICs can be damaged.

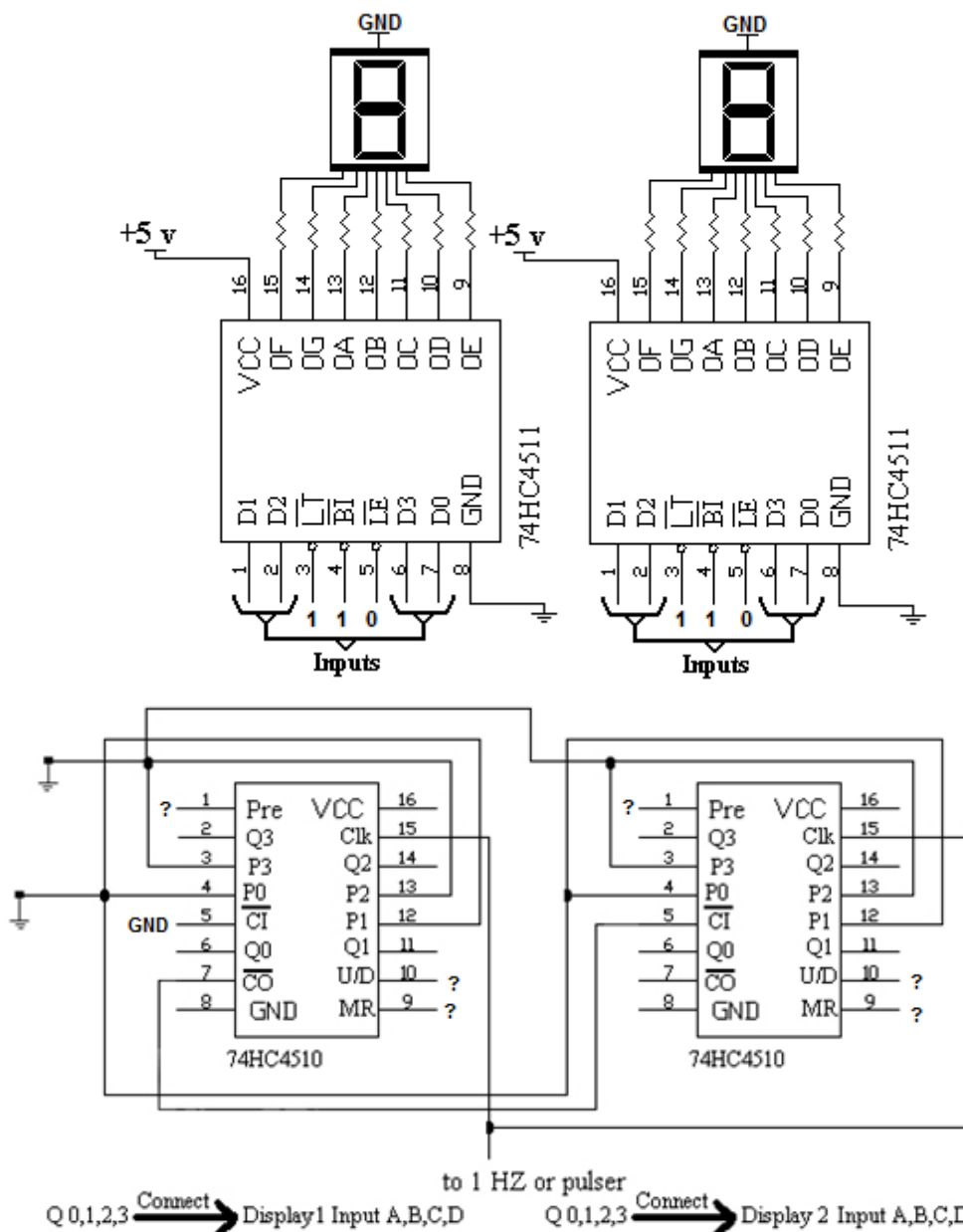


Figure 44: Two stage counter.

- C. Do not worry about resetting 74HC4510 ICs so that you start counting from 00. Your circuit can start counting from any arbitrary number. **Observe:** The two-stage decade counter has 100 possible states and will count from 00 to 99. The most significant bit of the first counter (/CO, terminal count output) is used as the count enable input to the second counter.

- D.** Because the second uses the most significant bit of the first as an input, total number of states in a multistage counter may be determined by multiplying the moduli ( $m$ ) of all stages together.

**Example:**

$(m_1 \times m_2 \times m_n) = \text{number of states}$  (where  $m$  is the modulus of each stage).

$[(10 \times 10)] = \text{number of states}$

100 = number of states

Note the range of a counter is always one less than the number of states. A counter with 100 possible states will count from 0 to 99.

**10.4.4 CONSTRUCT AND TEST THE FOLLOWING CIRCUIT.**

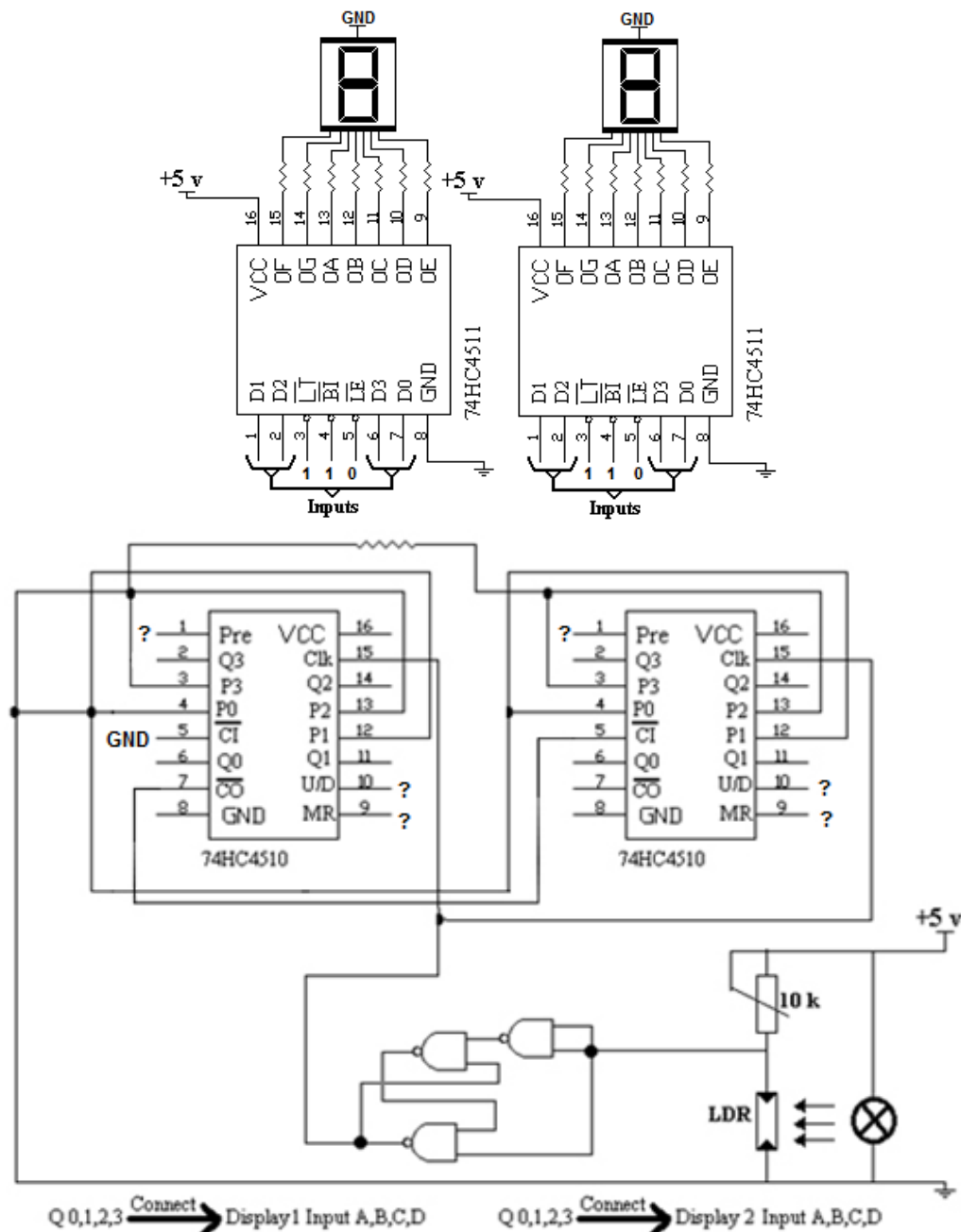


Figure 45: Light-sensitive circuit.

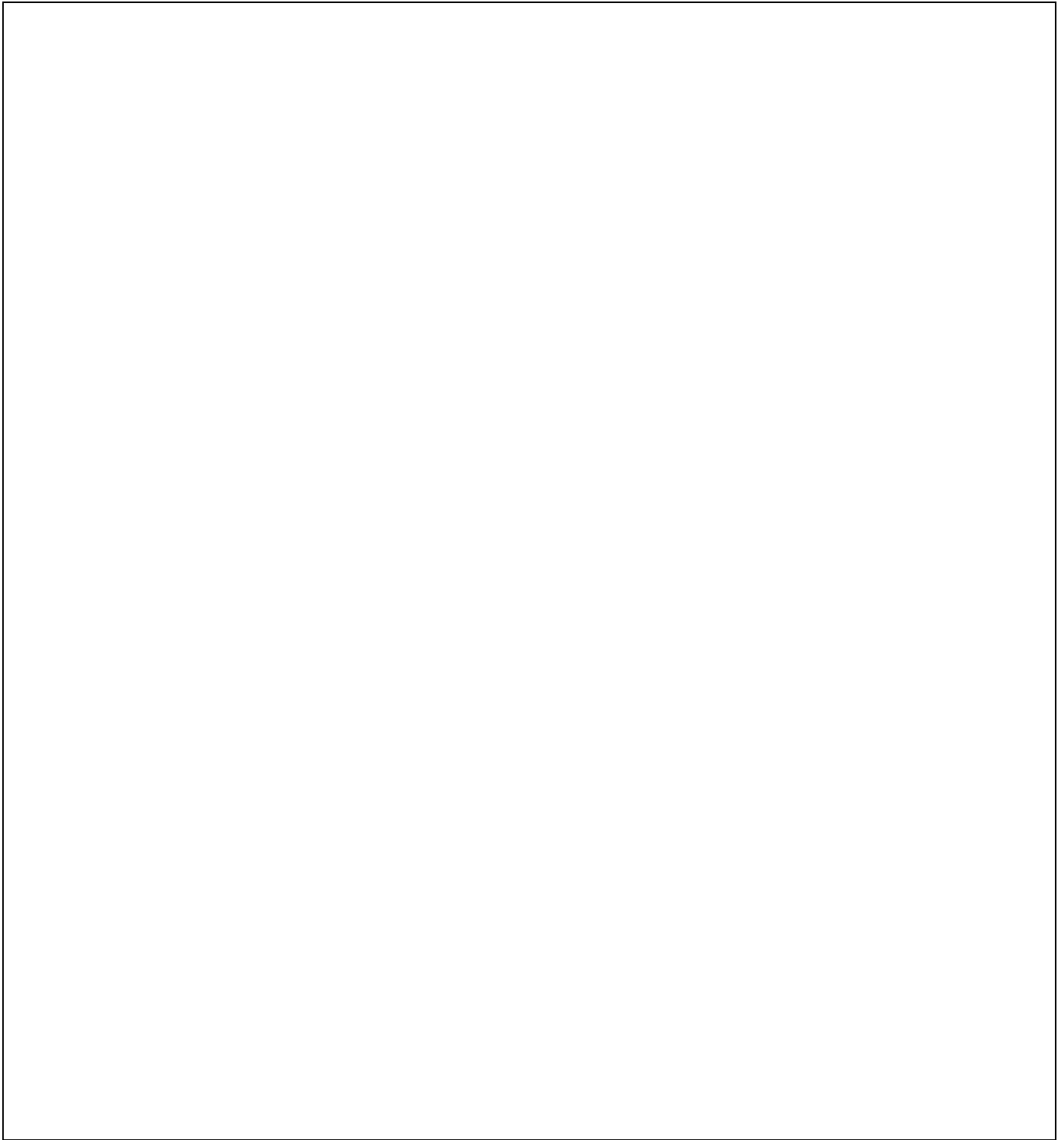


### 10.4.5 SUMMARY

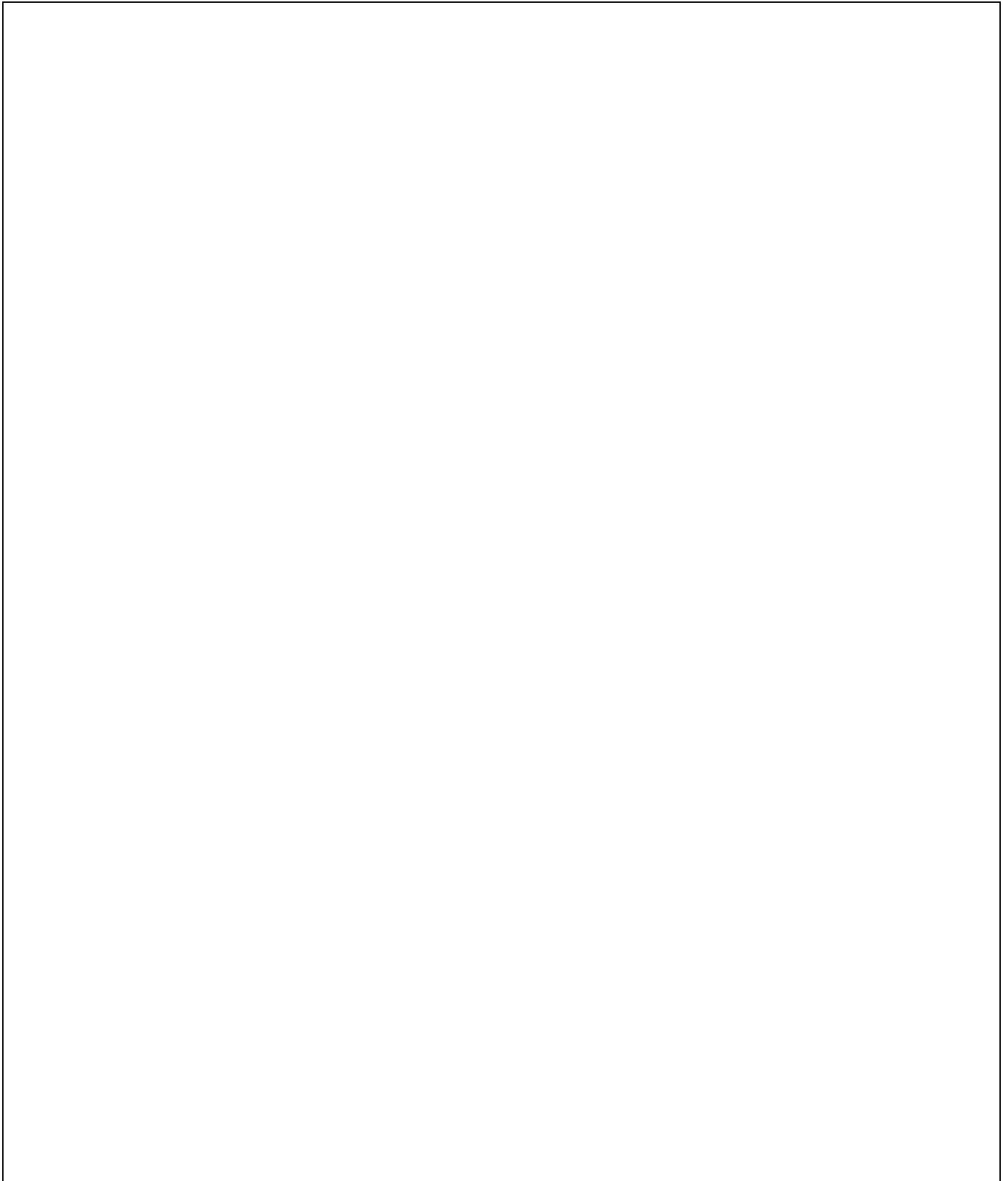
An asynchronous (ripple) counter is comprised of a series of flip-flops with the output of one connected to the clock input of the next. A signal introduced at the input will ripple (propagate) from one flip-flop to another until it reaches the end of the series. A count-up counter (up-counter) connects the Q output to the next input. A count-down counter (down-counter) connects the Q' output to the next input. The maximum frequency capability of a ripple counter is limited because some conditions require that the input must ripple through all flip-flops in the series before the next can occur. The modulus of a counter is defined as the number of states that can exist before the sequence repeats. The number of possible states of a multistage counter is determined by  $(m_1 \times m_2 \times m_n)$  where m equals to modulus of each counter.

### 10.4.6 SELF-TEST QUESTIONS

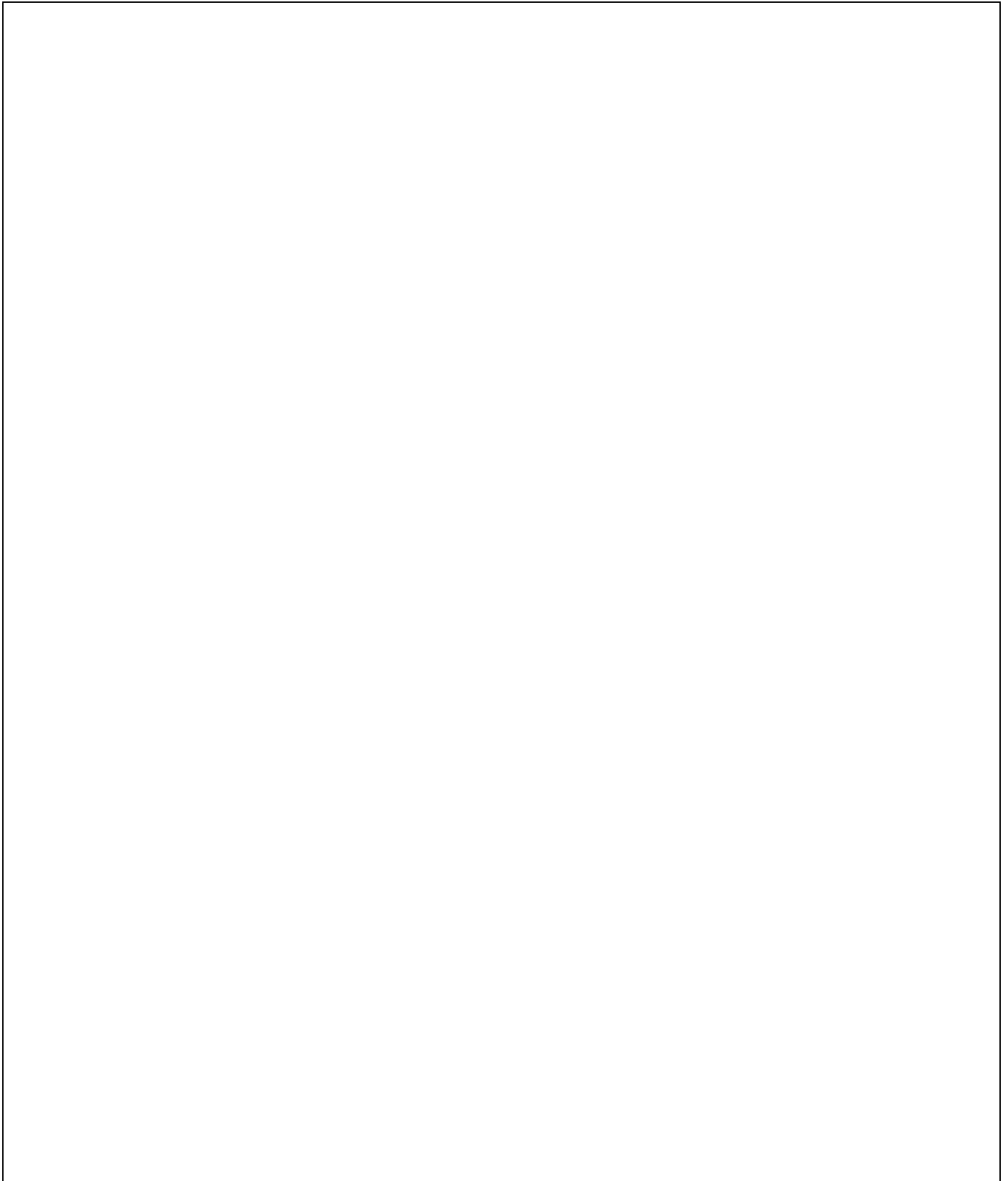
1. An asynchronous count-up counter
  - a) has a modulus of 16
  - b) has a modulus of 12
  - c) has its **Q** output connected to the next input
  - d) has its **Q'** output connected to the next input
  - e) a and c
  - f) a and d
2. The modulus of a counter can be determined by
  - a)  $2^n - 1$
  - b)  $m_1 \times m_2 \times m_n$
  - c) the number of states that exists before the sequence repeats
  - d) all of the above
3. A three stage counter with modulus of 12, 16, 10 has a range of 0 to \_\_\_\_\_.
  - a) 38
  - b) 1919
  - c) 3779
  - d) 1990
4. A binary counter which consists of 6 flip-flops will count from 0 to \_\_\_\_\_.
  - a) 6
  - b) 32
  - c) 64
  - d) 63



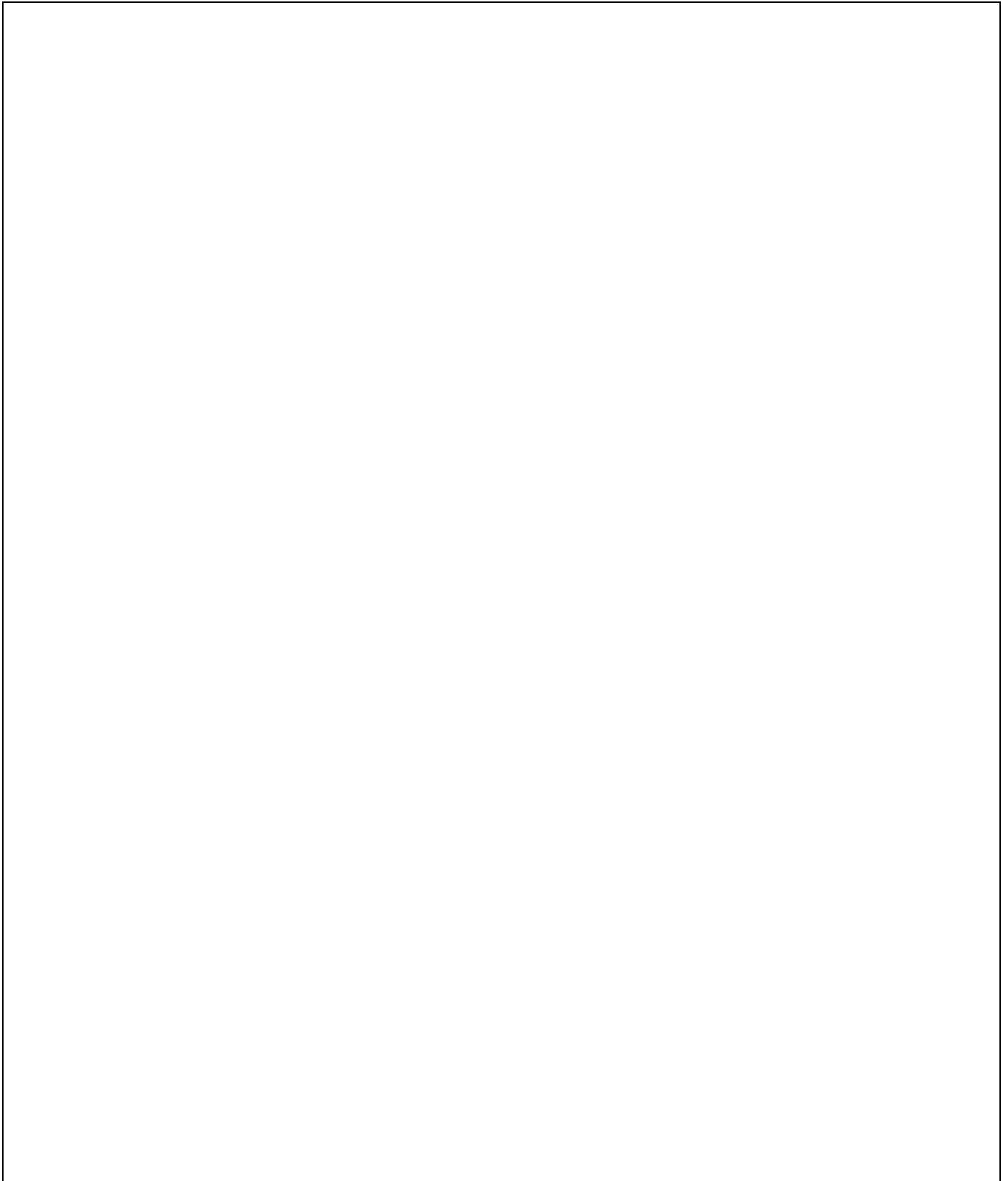
*Figure 46: Work area for count-up ripple counter.*



*Figure 47: Work area for decade counter.*



*Figure 48: Work area for two-stage counter circuit*



*Figure 49: Light sensitive counting circuit.*

### 10.4.7 FPGA PROGRAMMING

In this step you will design a 4-bit synchronous binary up-counter with VHDL and run it on the FPGA board. The entity declaration of the 4-bit counter is given below. You will write the architecture part of this VHDL code. You must write a structural and / or dataflow architecture for the counter. For structural and / or dataflow architecture, you can use the given instances of D flip-flop and your AND and XOR components to build your counter. The schematic for the counter is given in Figure 50.

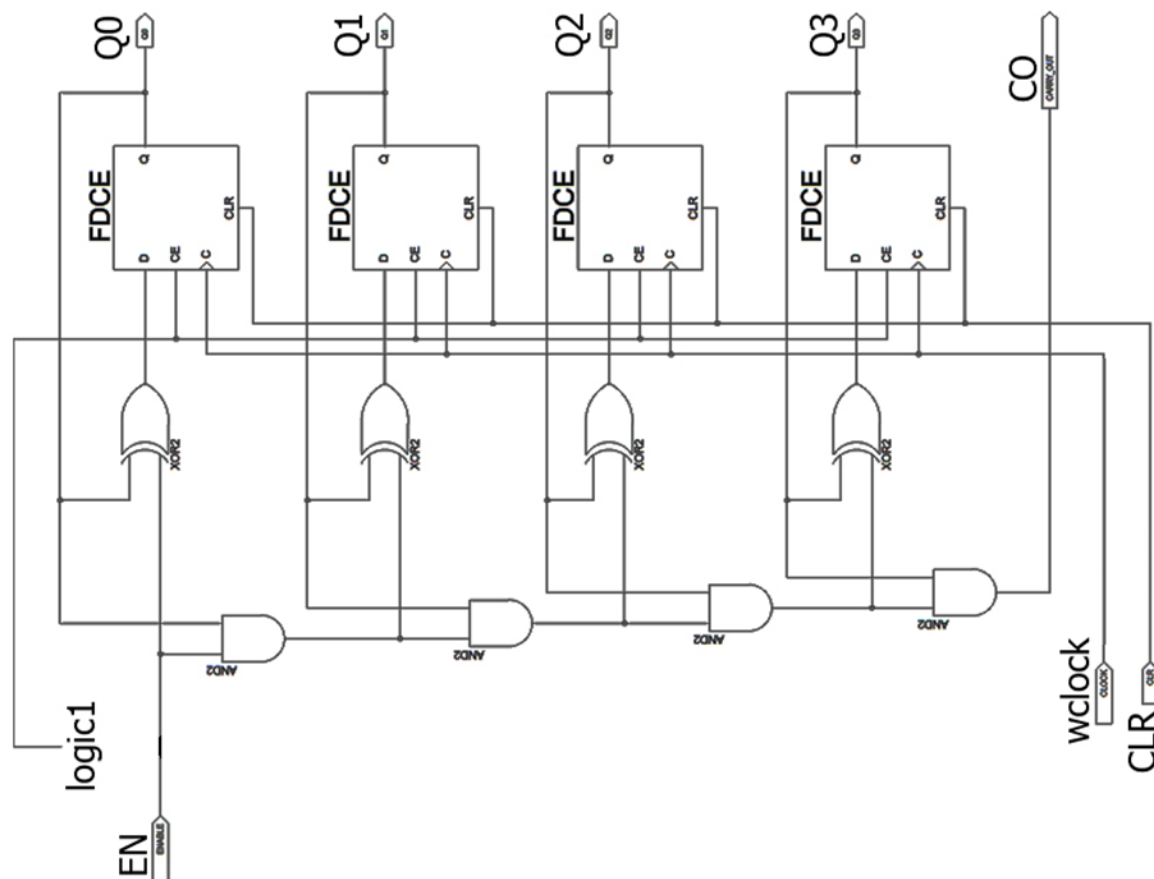


Figure 50: D Flip-flop instances used in the SpartanVI FPGA and 4-bit synchronous binary counter design built using AND and XOR components

In your structural and/or dataflow architecture of `my_counter4`, you will use FDCE instances as components. FDCE is the name of your D-flip flop. FDCE is a built-in device inside your FPGA. You can use them as components by properly calling them from the Xilinx library. This part is already done for you. All you need to do is to use them as components to build the design in Figure 50.

You also need to use AND and XOR components. For these components, you have to create entities and architectures for them. Be careful about your naming convention. Use creative names for your components. For example if you use the name "AND" for your AND component, Xilinx ISE may confuse this with the built-in AND components of the FPGA. "myand" or "my\_and" can be better naming alternatives. After you create your own AND and XOR components, you can use them inside the architecture of the `my_count4` with the FDCE components. Alternatively, if you do not want to use AND and XOR components, you may use "AND" and "XOR" operators in signal assignments to implement AND and XOR gates, respectively, in Figure 50.

Here is your 4-bit counter's library and entity declarations (and some comments).

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.vcomponents.all;

entity my_counter4 is
    Port ( wclock : in  STD_LOGIC;
          EN : in  STD_LOGIC;
          CLR: in  STD_LOGIC;
          Q : out  STD_LOGIC_VECTOR (3 downto 0);
          CO : out  STD_LOGIC);
end my_counter4 ;

architecture Structural of my_counter4 is

--FDCE instance is your D flip-flop. With this declaration,
--now you can use FDCE as component in your architecture.
--You can do this because FDCE is defined in Xilinx library.
--Do not edit this
component FDCE
    port (Q : out STD_LOGIC;
          C : in STD_LOGIC;
          CE : STD_LOGIC;
          CLR : in STD_LOGIC;
          D :in STD_LOGIC );
end component;
-----

--Declare your AND and XOR components and intermediate signals here
--If you do not want to use AND and XOR components, you can use
--predefined AND and XOR operators in your signal assignments.

begin

--Fill the architecture part here. Use the FDCE components
--as well as your own AND and XOR components. You should use
--intermediate signals.
--FDCE components have clock enable input, named CE. It is an
--active-high signal. Always connect them to logic1.
--Warning: Output port of an entity cannot be used as input inside.
--In order to do that you need an intermediate signal.

end Structural;

```

Design a 4-bit up counter with your VHDL file (my\_counter4.vhd) as your top-level design file and run it on the FPGA. You will use this VHDL file and the "lab4count4.ucf" file as your source files in the Project Navigator. And you will do the same operations as you did in Lab3, to get your design running on the FPGA. You will give your inputs again using the switches on your Nexys3 board and observe the outputs on the 5 individual LEDs on this board.

You now need to set pin connections of your input and outputs to the FPGA chip according to your port declarations given in the entity declaration. They are given in the lab4count4.ucf file. When you use this ucf file with your design file and implement the design, you will get a bit file of the design which has "wclock" input connected to the

BTNS push-button, and "EN", and "CLR" inputs connected to the SW0 and SW1 sliding switches, respectively, on the Nexys3 board and output connections, Q0, Q1, Q2, Q3 and CO to individual LEDs, LD0, LD1, LD2, LD3 and LD4 respectively on the board. In this part, you use only five LEDs of the FPGA board.

Note: You may find the "lab4count4.ucf" file in " E:\EE240\lab4" directory, which is also copied below.

#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments

NET "Q<0>" LOC = "U16" ;

NET "Q<1>" LOC = "V16" ;

NET "Q<2>" LOC = "U15" ;

NET "Q<3>" LOC = "V15" ;

NET "CO" LOC = "M11" ;

NET "wclock" LOC = "B8" ;

NET "wclock" CLOCK\_DEDICATED\_ROUTE=FALSE;

NET "CLR" LOC = "T9" ;

NET "EN" LOC = "T10" ;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE

You may bring your VHDL files for all parts with you, so that you will be ready to implement your designs in Lab. And before coming to lab, write or get a hardcopy of your VHDL code on an A4 sheet as your preparation for this part of the lab.

#### 10.4.7.1 Simulating Designs with ISim

You can verify design functionality early in the design flow by simulating your schematics or VHDL designs. Testing your design decisions before the design is implemented on a FPGA board allows you to make any necessary changes early on. It is very important part of whole design flow so simulation phase cannot be taken lightly for any kind of digital design project. Before you use Nexys3 boards to implement your design in reality, you have to simulate them using a simulator and be sure that your designs work as intended for all cases. We will use ISim simulator to simulate designs.

For simulating your designs with ISim simulator, you need to create a testbench file in ISE Project Navigator. In order to this:

1-) Select "New Source" from the Project Menu and choose "VHDL Test Bench" type from the pop-up menu and give a name such as "lab3mm78desk01\_tb" to your testbench file. Then click "Next".

2-) In the new window, choose the name of the design file that you want to simulate in ISim. This way, you will associate your testbench waveform file with that file. Click "Next" and then "Finish" in the new window. An empty testbench file will be created according to your design.

3-) You should modify this testbench file to give inputs for your design simulation. As an example, input test sequences for en, clr and clock signals are given below.



These input sequences define a simple user case of clearing the counter such that clear signal, `clr`, is initially logic '0', will be logic '1' after 500ns and will be logic '0' again after 1000 ns. Clock signal is also defined in a similar manner. Enable signal, `en`, is activated all the time by setting it to logic '1'. We expect that the outputs of the counter will be "0000" when `clr` signal goes from '0' to '1'. The counter outputs will be incremented every time `clk` signal makes a transition from logic '0' to logic '1' (rising edge) after the `clr` signal is deactivated to logic '0' again. You may freely change this pattern to test different conditions and scenarios especially the roll-over case to make sure that your design is working for every condition.

```
tb : PROCESS
BEGIN
    en <= '1',
    clr <= '0',
        '1' after 500 ns,
        '0' after 1000 ns;

    wclock <= '0',
        '1' after 150 ns,
        '0' after 200 ns,
        '1' after 250 ns,
        '0' after 300 ns,
        '1' after 350 ns,
        '0' after 400 ns,
        '1' after 450 ns,
        '0' after 500 ns,
        '1' after 550 ns,
        '0' after 600 ns,
        '1' after 650 ns,
        '0' after 700 ns;
    WAIT; -- will wait forever
END PROCESS;
```

4-) Now, you have a testbench file to test your design with ISim. Select your new testbench file in the "Simulation" view of "Design" window and then select "Simulate Behavioral Model" in the "Processes" window.

5-) ISim simulator is launched if your testbench's syntax is correct. Otherwise, go back and fix the errors in your testbench according to error messages given by the tool. If everything is OK, it loads your design files and simulates it. The results of the simulation for the input signals you have entered while creating the testbench waveform file are given in the "wave" window of ISim. By using zoom buttons you can observe the outputs of your circuit. Please spend some time to explore ISim interface.

6-) After finishing your simulation, select "Quit" from the file menu to close ISim.

Note: You may simulate your design after various steps in the implementation with Project Navigator. So for example, after the place and route step, by selecting "Simulate Post-Place&Route Model" in the "Processes Window" with your testbench waveform file chosen in "Sources Window", you can get a more realistic simulation of your design with gate delays (inside the FPGA) incorporated.

## 11 LABORATORY 5 – CONVERSION OF 6-BIT FLOATING POINT NUMBERS INTO SIGN-MAGNITUDE REPRESENTATION AND ITS FPGA IMPLEMENTATION

### 11.1 DESCRIPTION

The purpose of this lab is to design a decoder that converts a companded 6-bit floating point representation into sign-magnitude representation and then show the result in BCD format using 7-segment LED displays. This decoder is implemented on the FPGA board. Overall schematic for the system is shown in Figure 51.

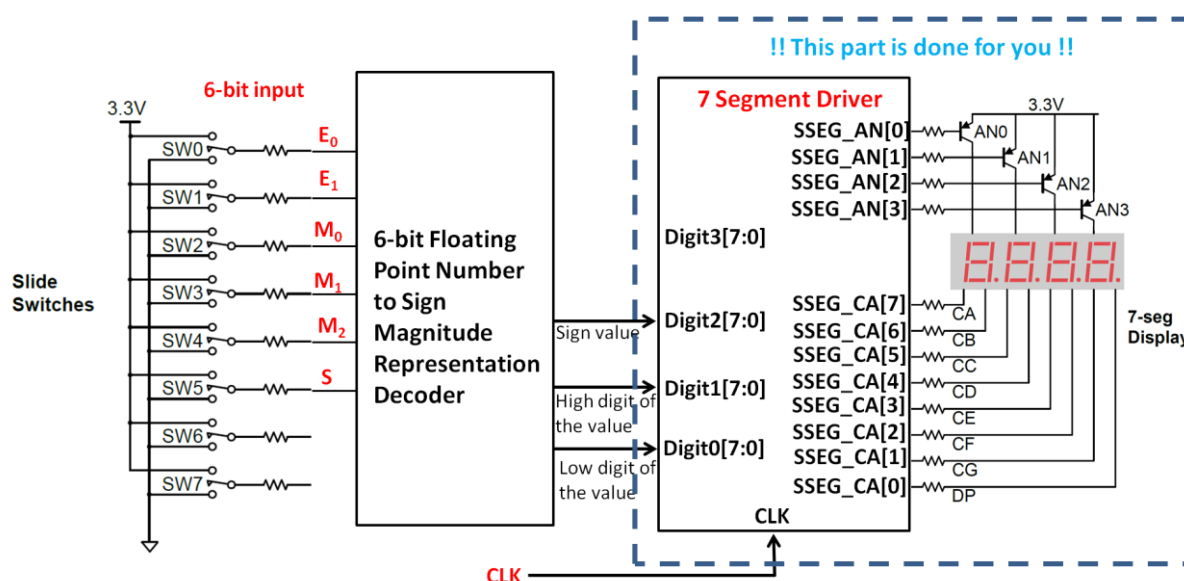


Figure 51. System overview of Lab 5

A linear encoding using 8 bits can represent signed numbers ranging from -128 to +127 using two's-complement representation. This dynamic range supplied by 8-bit linear encoding may not be sufficient for certain applications or the same range may be desired to be covered using less bits. Therefore, nonlinear encodings are used in most practical systems. These encodings represent signals by numbers that approximate the logarithms of their values. Two standard systems,  $\mu$ -law PCM and A-law PCM, are typically used.

For this laboratory assignment, we will use a simplified floating-point representation consisting of one sign bit, a 2-bit *exponent*, and a 3-bit *significand*. The value represented by a 6-bit floating number,  $SM_2M_1M_0E_1E_0$ , in this format is:

$$\text{Value} = (1-2S) M 2^E$$

where  $S$  is the sign bit,  $M$  is the significand and  $E$  is the exponent. The 3-bit significand,  $M$ , ranges from  $(000)_2$  to  $(111)_2$  representing 0 and 7, respectively and the exponent ranges from  $(00)_2$  to  $(11)_2$  representing 0 and 3, respectively. Here are some examples for the number representation in floating point and sign-magnitude methods.

Floating Point Representation	Corresponding Value	Sign Magnitude Representation
0-111-11	+56	0111000
1-111-11	-56	1111000
0-100-00	+4	0000100
1-110-01	-12	1001100

Some numbers have multiple representations in floating points. The preferred representation is the one in which the most significant bit of the significand is 1; this representation is said to be *normalized*. It is quite straightforward to produce the linear encoding corresponding to a floating-point representation; this operation is called *expansion*. This laboratory assignment is about building this expansion decoder. The combinational circuit that does the inverse operation is called *compression* and it is a little harder to implement. A device that performs both expansion and compression is called a *comparer*. The compression half of a comparer is more challenging because there are more input bits than output bits and therefore many different linear encodings must be mapped to same floating-point representation. Values that do not have floating point representations should be mapped to the closest floating point encoding. This process is called *rounding*. In this laboratory work, you do not need to worry about compression operation.

An overall block diagram for the floating-point conversion circuit is shown in Figure 52. A 6-bit floating number representation is first converted into sign-magnitude representation by doing multiplications. Then, the resulting sign-magnitude representation is decoded again (for example, binary to BCD and then BCD to 7-segment decoding) and displayed using 7-segment LEDs. BCD is binary coded decimal.

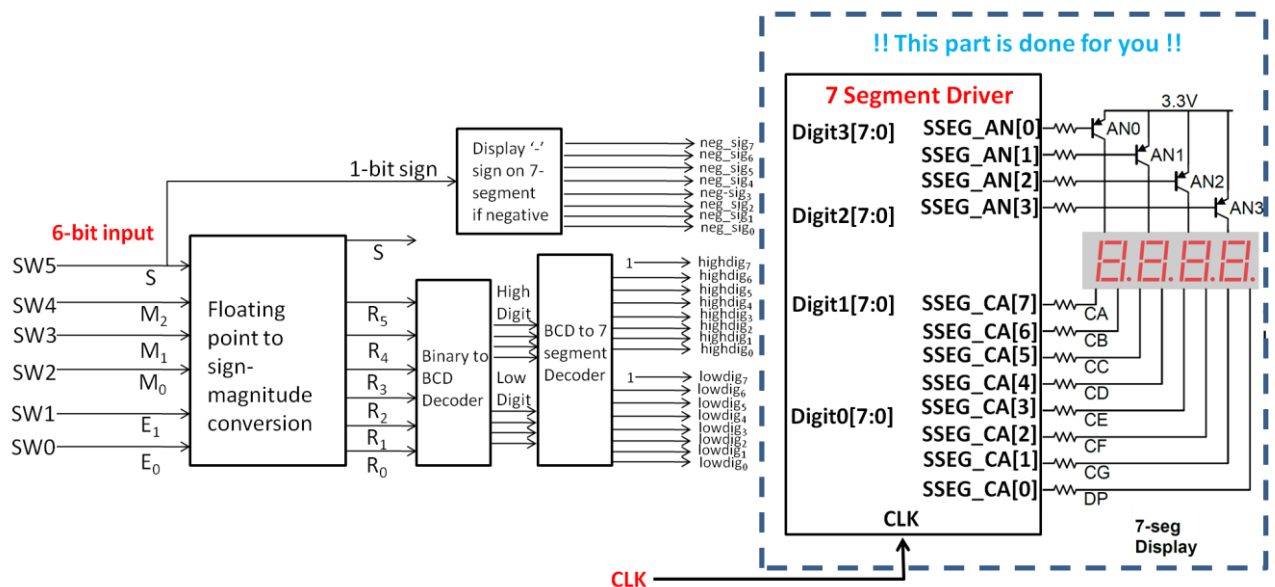


Figure 52. Overall block diagram for the floating-point conversion circuit

## 11.2 FPGA IMPLEMENTATION OF THE DESIGN

The purpose of this part of the laboratory assignment is to synthesize your decoder you designed and to actually implement it on an FPGA. You are going to use 6 slide switches available on the FPGA kit to enter a 6 bit floating point data.

### 11.2.1 READING THE SLIDE SWITCHES

Eight slide switches are available on the FPGA board as shown in Figure 28 and 29. When closed (ON), each switch pulls the connected pin of the FPGA to ground through a resistor. The pin is pulled high through a resistor when the switch is open (OFF). This is shown in Figure 53. You will use 6 of these switches to enter 6 bit floating point number to your design. The entity of your design will have these 6 bits as input ports.

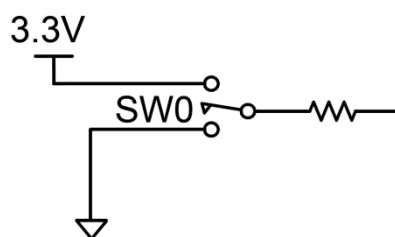


Figure 53. Structure of the slide switch.

### 11.2.2 DISPLAYING THE SIGN-MAGNITUDE NUMBER ON THE FOUR-DIGIT-SEVEN-SEGMENT LED DISPLAYS SEQUENTIALLY

There is a four-digit-seven-segment display on the FPGA board to display four digits (Digit0, Digit1, Digit2 and Digit3). We want to display the number in the BCD format with an additional displaying of sign value. In order to do that, binary-to-BCD and BCD-to-7-segment decoding are necessary. To show each digit, 7 outputs are required (Figure 54). The VHDL code below shows how a single digit BCD code can be decoded to show its value on a 7-segment display for a common anode configuration (the outputs must be complemented if a common cathode seven-segment display is used.). You can modify this code to accommodate two digits.

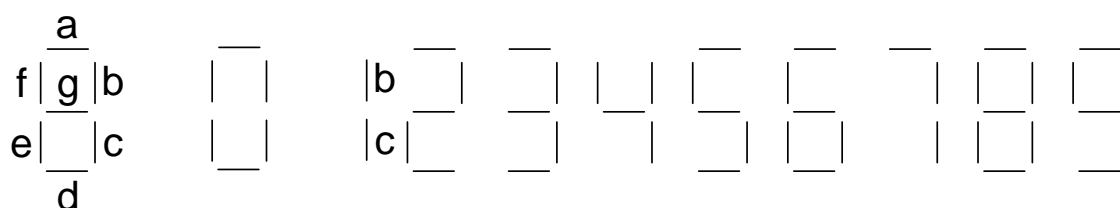


Figure 54. Segments of the seven-segment LED display.

```
entity BCD_to_seven_segment is
port (
    d: in std_logic_vector (3 downto 0);
    s: out std_logic_vector ( 6 downto 0) );
end BCD_to_seven_segment;

architecture dataflow of BCD_to_seven_segment is
begin
```

```

with d select
  s <="1000000" when "0000",
    "1111001" when "0001",
    "0100100" when "0010",
    "0110000" when "0011",
    "0011001" when "0100",
    "0010010" when "0101",
    "0000010" when "0110",
    "1111000" when "0111",
    "0000000" when "1000",
    "0010000" when "1001",
    "1111111" when others;
end dataflow;

```

### 11.2.3 DISPLAYING DIGITS SEQUENTIALLY ON THE FOUR-DIGIT-SEVEN-SEGMENT LED DISPLAYS

A seven-segment LED display is composed of individual LEDs with common anode (or cathode) arranged in “figure 8” pattern. However, seven-segment name used for the display of our FPGA board should not mislead you since it actually has eight LEDs (one LED for the decimal point called DP) as shown in Figure 55 [8]. Each LED on the seven-segment display can be turned on individually. In this lab, you are generating 7-bit data but the board requires 8-bit input for each digit. Since we are not going to use this decimal point, you can assign logic 1 value to the DP LED to turn it off, which is the most significant bit. Therefore, Digit0[7] and Digit1[7] can be constant values of logic 1.

In a typical application, each digit of seven-segment display can be used individually at any time. This means, in order to show “1” in Digit0, CB and CC pins of Digit0 must be logic0 (0 V), CA, CD,CE,CF,CG and DP pins of Digit0 must be logic1 (3.3 V) and the common anode pin of Digit0 (AN0) must be logic1 (3.3 V). To use the other three digits, we need individual CA to DP pins. This means for four digits we need to insert 32 pins (for CA to DP pins) plus 4 (for common anodes) pins, which sums to 36 pins. Since this display must be connected to the FPGA chip, this method requires the dedication of 36 pins of the FPGA chip to the seven-segment display, which is wasteful and not desirable. Instead, the same function can be implemented by just using 12 pins with time multiplexing. In this method, CA to DP inputs are made common to all four digits, which requires 8 pins of the FPGA chip. Separate 4 anode pins (AN0, AN1, AN2 and AN3) for 4 digits are required to turn on desired digit. Inserting CA to DP inputs does not turn on LEDs as long as the corresponding common anode pin is pulled up to 3.3 V. Therefore, in order to display a digit in Digit0, CA to DP values are inserted; 3.3 V is applied to AN0 but 0 V to AN1, AN2 and AN3. This way, only one digit can be displayed at a time. Therefore, Digit0, Digit1, Digit2 and Digit3 must be displayed one at a time, one after the other. If this is done very fast (faster than 60 Hz), human eye will see the individual digits turned on at the same time. It will fail to detect this sequential operation of LEDs. The time multiplexing method is explained in Figure 55. Cathode values (CA to DP) must be inserted according to the corresponding digit, which is activated by its common anode value (AN0 to AN3). Note that, according to Figure 29, which shows how the common anode pins are connected to the FPGA chip, when the pin of the FPGA chip, which is N16 for AN0 is logic 0 (0 V), the anode of the digit0 goes to 3.3 V, thus turning it on.

We want to display three digits in this lab. You do not need to design and implement this sequential seven-segment driver operation. This is already done for you. All you need to do is to present your 3 digit data to the seven segment driver component, which is given in the *nexys3\_sseg\_driver.vhd* file. You can use structural vhdl to connect your

design and this seven segment driver design. Thus, your design should have 24 outputs. The first 8-bit output (least significant digit) is connected to Digit0. The second 8-bit output (most significant digit) is connected to Digit1. The sign bit of your input should be converted to “11111111” to display nothing if the input is positive or to “10111111” to display minus sign if the input is negative. This new 8-bit representation of the sign is connected to Digit2. The entity of your design should have three sets of 8-bit output ports. These outputs will be connected to seven-segment driver design. Seven segment driver uses a clock signal to turn on individual digits one at a time with a refresh rate of 100 Hz. Thus, the overall design also needs a clock signal for display purposes even though this lab is about combinational circuits.

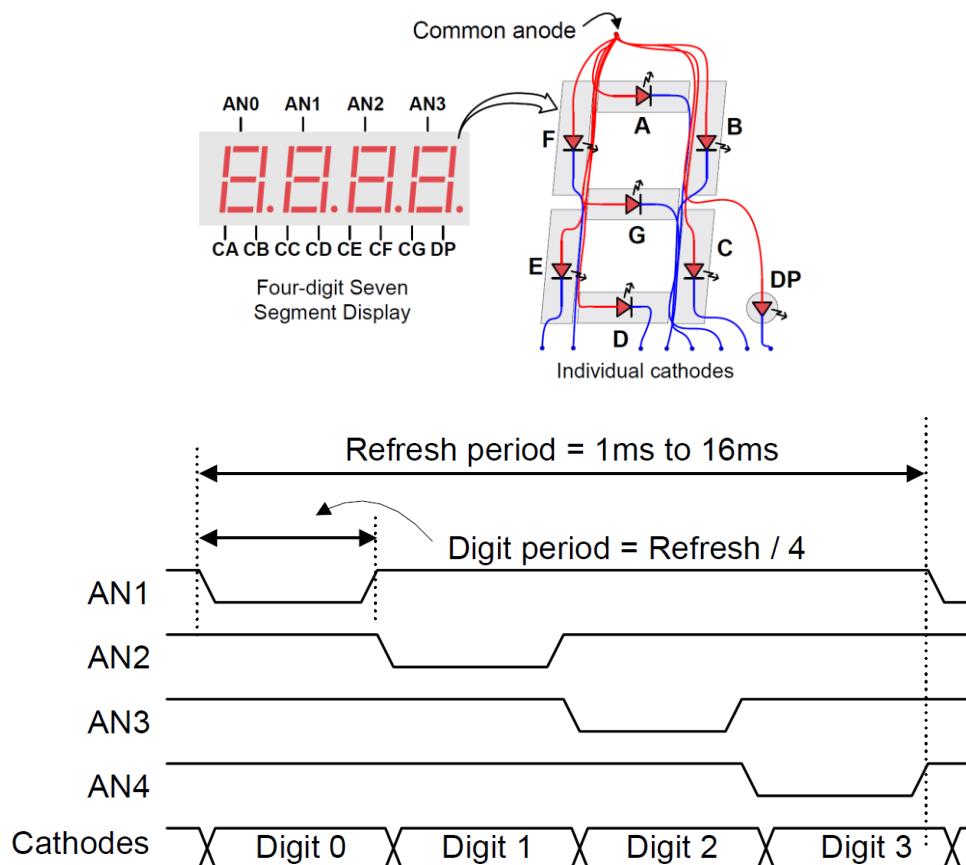


Figure 55. Displaying four digits on four-digit-seven-segment display sequentially. [8]

#### 11.2.4 CONVERTING 6-BIT BINARY NUMBER INTO TWO-DIGIT BCD

You can use “brute force” to implement this converter. This means you can write down all of the 64 cases. However, a better way is to use shift and add-3 algorithm. In this algorithm  $n$ -bit binary number is shifted left one at a time to a shift register that has enough 4-bit segments. The process ends when  $n$  shifts take place. The BCD number becomes ready in the shift register. During shifting, if the binary value in any of the BCD segment is 5 or greater, 3 is added to that value in that BCD segment. For a 6-bit binary number this algorithm can be implemented in structural style as shown in Figure 56. A combinational circuit labeled “Add-3(Conditional)” takes a 4-bit binary number and

passes it to the 4-bit output as it is if its magnitude is smaller than 5. It adds 3 to this number and it passes this incremented value to the output, otherwise. We do not care what the outputs are for binary input values greater than 9 since we do not insert these values to the circuit. By using three of these circuits as shown on the left of Figure 56, a 6-bit binary number can be converted to BCD. This structure can be described either using structural or dataflow VHDL.

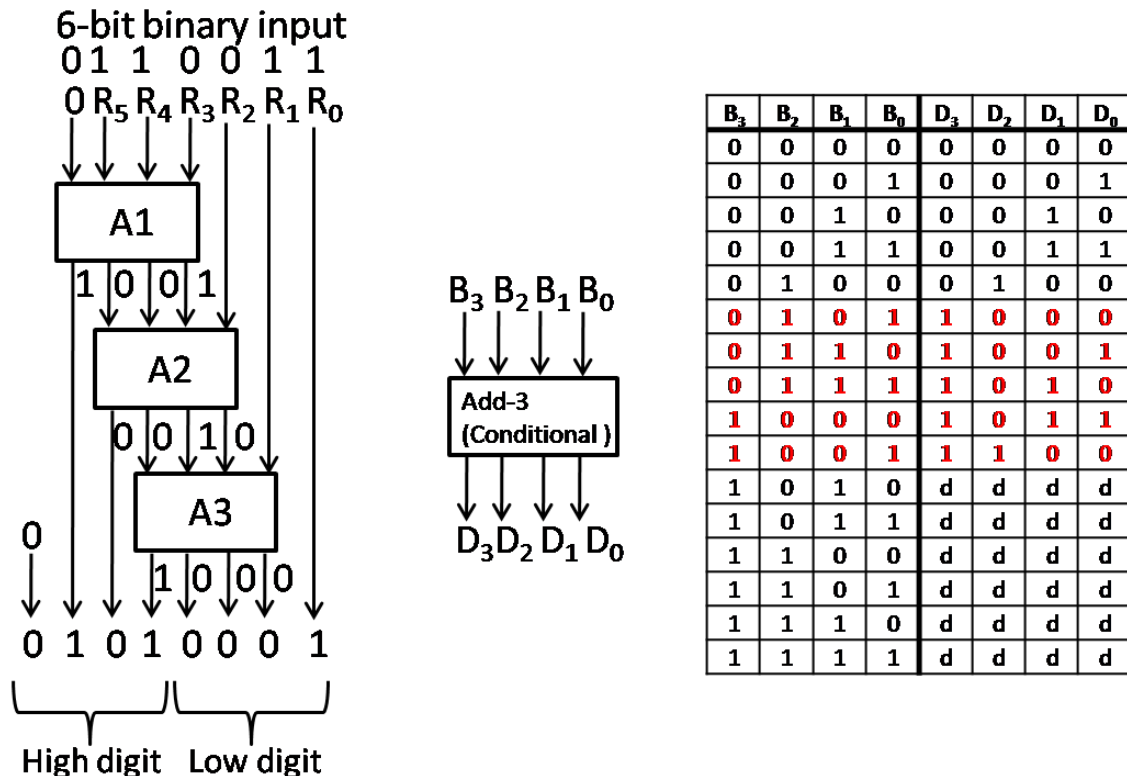


Figure 56. Conversion of a 6-bit binary number to BCD

### 11.2.5 OVERALL DESIGN

As explained above your design will accept 6-bit input, named FP, and a clock input, my\_clk, to display digits sequentially and generate 8-bit neg\_sig output, 8-bit lowdig output and 8-bit highdig output. Then, these outputs will be inputs to the design given in the *nexys3\_sseg\_driver.vhd* file using structural vhdl style. This design will generate the SSEG\_CA(7 downto 0) and SSEG\_AN(3 downto 0) outputs using the clock signal. Your VHDL code for this design should have the following structure. You should complete the architecture. Your VHDL code can use only structural and/or dataflow style VHDL. Behavioral style is not allowed in this lab.

-- Synthesizable Floating Point to Signed Magnitude Decoder, EE240 class Bogazici University  
-- Implemented on Xilinx Spartan VI FPGA chip

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;
USE ieee.std_logic_arith.all ;
```

```
entity FP_decoder is
  port (
    FP: in std_logic_vector(5 downto 0);
    MY_CLK: in std_logic;
```

```

        SSEG_CA: out std_logic_vector(7 downto 0);
        SSEG_AN: out std_logic_vector(3 downto 0));
end;

architecture arch_FP_decoder of FP_decoder is

--FP(5) is the sign bit coming from SW5
--FP(4) is M2 coming from SW4
--FP(3) is M1 coming from SW3
--FP(2) is M0 coming from SW2
--FP(1) is E1 coming from SW1
--FP(0) is E0 coming from SW0

--MY_CLK is coming from the board clock, connected to V10

--lowdig(0) is displayed on Digit0 Segment CA
--lowdig(1) is displayed on Digit0 Segment CB
--lowdig(2) is displayed on Digit0 Segment CC
--lowdig(3) is displayed on Digit0 Segment CD
--lowdig(4) is displayed on Digit0 Segment CE
--lowdig(5) is displayed on Digit0 Segment CF
--lowdig(6) is displayed on Digit0 Segment CG
--lowdig(7) is displayed on Digit0 Segment DP, which is always logic 1

--highdig(0) is displayed on Digit1 Segment CA
--highdig(1) is displayed on Digit1 Segment CB
--highdig(2) is displayed on Digit1 Segment CC
--highdig(3) is displayed on Digit1 Segment CD
--highdig(4) is displayed on Digit1 Segment CE
--highdig(5) is displayed on Digit1 Segment CF
--highdig(6) is displayed on Digit1 Segment CG
--highdig(7) is displayed on Digit1 Segment DP, which is always logic 1

-- neg_sig digit either shows minus sign or nothing
--neg_sig(0) is displayed on Digit2 Segment CA,
--neg_sig(1) is displayed on Digit2 Segment CB
--neg_sig(2) is displayed on Digit2 Segment CC
--neg_sig(3) is displayed on Digit2 Segment CD
--neg_sig(4) is displayed on Digit2 Segment CE
--neg_sig(5) is displayed on Digit2 Segment CF
--neg_sig(6) is displayed on Digit2 Segment CG
--neg_sig(7) is displayed on Digit2 Segment DP

-- lowdig(7 downto 0), highdig(7 downto 0) and neg_sig(7 down to 0) will be input to the design
-- given in the nexys3_sseg_driver.vhd file using structural vhd style. This design will generate
-- the SSEG_CA(7 downto 0) and SSEG_AN(3 downto 0) outputs using the clock signal.

end arch_FP_decoder;
```

You may need to use a multiplier in this design. AVHDL multiplier example is given below.

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;
USE ieee.std_logic_arith.all ;
entity mymultiply is
    port(
        a : in STD_LOGIC_VECTOR(15 downto 0);
```



```

        b : in STD_LOGIC_VECTOR(7 downto 0);
        mult : out STD_LOGIC_VECTOR(23 downto 0);
    );
end mymultiply;
architecture dataflow of mymultiply is
begin
    mult <= a * b;
end dataflow;

```

### 11.2.6 PIN ASSIGNMENT

After successfully completing the design, compilation, ISim simulations and synthesis, you should show that your design work on the FPGA board. FPGA board has certain wiring. Only specific pins of the FPGA are connected to the slide switches and seven-segment display. The input and output ports of your design should be connected to these pins of FPGA. You can specify which pin of the FPGA is connected to your input and output of your design before starting synthesis. You can achieve this pin assignment either manually by entering the corresponding pin numbers below using the graphical interface of the Xilinx ISE CAD tool or using the .ucf file below. By looking at the manual of the Nexys3 Board, we can find which FPGA pins are connected to which components. The following is a list of pin connection we want and their meanings.

```

FP(5) -> U8      Slide SW-5 -> U8
FP(4) -> N8      Slide SW-4 -> N8
FP(3) -> M8      Slide SW-3 -> M8
FP(2) -> V9      Slide SW-2 -> V9
FP(1) -> T9      Slide SW-1 -> T9
FP(0) -> T10     Slide SW-0 -> T10

```

```

SSEG_CA<0> -> T17
SSEG_CA<1> -> T18
SSEG_CA<2> -> U17
SSEG_CA<3> -> U18
SSEG_CA<4> -> M14
SSEG_CA<5> -> N14
SSEG_CA<6> -> L14
SSEG_CA<7> -> M13

```

```

SSEG_AN<0> -> N16
SSEG_AN<1> -> N15
SSEG_AN<2> -> P18
SSEG_AN<3> -> P17

```

```

MY_CLK ->V10

```

The .ucf file should be like this.

```

#
# Pin assignments for the Nexys3 Spartan VI Board.
#
Net FP<0> LOC=T10 | IOSTANDARD=LVCMOS33; # Slide switch0
Net FP<1> LOC=T9 | IOSTANDARD=LVCMOS33; # Slide switch1
Net FP<2> LOC=V9 | IOSTANDARD=LVCMOS33; # Slide switch2
Net FP<3> LOC=M8 | IOSTANDARD=LVCMOS33; # Slide switch3
Net FP<4> LOC=N8 | IOSTANDARD=LVCMOS33; # Slide switch4
Net FP<5> LOC=U8 | IOSTANDARD=LVCMOS33; # Slide switch5

Net SSEG_CA<0> LOC=T17 | IOSTANDARD=LVCMOS33; # Connected to CA

```

```
Net SSEG_CA<1> LOC=T18 | IOSTANDARD=LVCMOS33; # Connected to CB
Net SSEG_CA<2> LOC=U17 | IOSTANDARD=LVCMOS33; # Connected to CC
Net SSEG_CA<3> LOC=U18 | IOSTANDARD=LVCMOS33; # Connected to CD
Net SSEG_CA<4> LOC=M14 | IOSTANDARD=LVCMOS33; # Connected to CE
Net SSEG_CA<5> LOC=N14 | IOSTANDARD=LVCMOS33; # Connected to CF
Net SSEG_CA<6> LOC=L14 | IOSTANDARD=LVCMOS33; # Connected to CG
Net SSEG_CA<7> LOC=M13 | IOSTANDARD=LVCMOS33; # Connected to DP
```

```
Net SSEG_AN<0> LOC=N16 | IOSTANDARD=LVCMOS33; # Connected to AN0
Net SSEG_AN<1> LOC=N15 | IOSTANDARD=LVCMOS33; # Connected to AN1
Net SSEG_AN<2> LOC=P18 | IOSTANDARD=LVCMOS33; # Connected to AN2
Net SSEG_AN<3> LOC=P17 | IOSTANDARD=LVCMOS33; # Connected to AN3
```

```
Net MY_CLK LOC=V10 | IOSTANDARD=LVCMOS33; # Connected to board clock
```

### 11.3 PREPARATION (PRELAB)

#### For Floating Point to Sign-Magnitude Conversion

- Write a VHDL code for this component. Use only structural or dataflow style.
- Run ISim and verify that it functions as expected by running a few test sequences.

#### For the 6-bit binary to BCD Conversion

- Write a VHDL code for this component. Use only structural or dataflow style.
- Run ISim and verify that it functions as expected by running a few test sequences.

#### For the BCD-to7-Segment Conversion

- Use the binary-to\_seven\_segment code given above.

#### For the Overall Design

- Combine your design components and the design given to you in *nexys3\_sseg\_driver.vhd* under FP\_decoder architecture and run ISim on it and verify that it functions as expected by running a few test sequences.

### 11.4 IN LAB

- Use Xilinx ISE to synthesize the design based on Xilinx Spartan VI family. Use either the given ucf file or enter manually to make pin assignment.
- Generate the bit file and upload it to your FPGA.
- On the hardware show that your design works.
- See if the LEDs show what you expect every time you apply your data using slide switches.

## 12 LABORATORY 6 – UP/DOWN BCD COUNTING THE NUMBER OF PUSHES ON BUTTONS AND ITS FPGA IMPLEMENTATION

### 12.1 DESCRIPTION

The purpose of this lab is to design a parallel loadable, 2-digit up/down BCD counter. Counting is done by pressing and releasing a push-button switch. By pressing and releasing one of the three push-button switches, the counter can be parallel loaded, up-counted or down-counted. The content of the counter is displayed on two 7-segment LED displays. This counter is implemented on the FPGA board. Overall schematic for the system is shown in Figure 57.

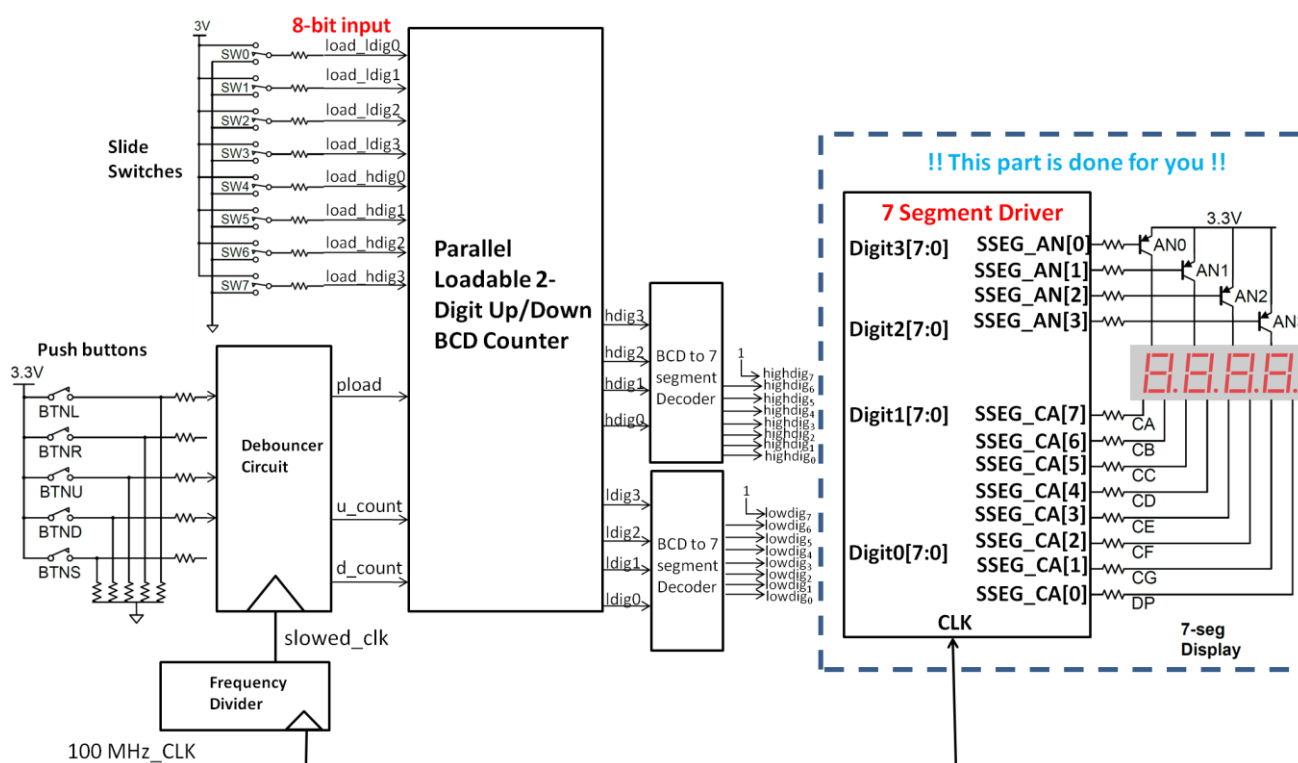


Figure 57. System overview of Lab 6

### 12.2 FPGA IMPLEMENTATION OF THE UP/DOWN COUNTER

The purpose of this laboratory assignment is to design, synthesize and implement a BCD counter. In this assignment you are supposed to show that you can do up-counting, down-counting and also parallel loading. You are going to use 8 slide switches available on the FPGA kit to enter an 8 bit data for parallel loading. You will use 3 push-button switches, to do parallel load and then stop, to count up and to count down. The content of your counter should be displayed on the two digits of the seven-segment display of the Nexys3 board. The details are given below.

#### 12.2.1 READING THE SWITCH

The FPGA is connected to five push buttons (BTNL, BTNR, BTNU, BTND and BTNS) on the Nexys3 board. Each push button applies a high level to its FPGA pin through a

resistor when pressed and a resistor pulls the pin to a low level when the pushbutton is released. You will use the outputs of three of these switches as parallel load, up-count and down-count commands. If you press and release BTNL, it should load the 8bit data presented by the slide switches to the counter. After loading, it should not count automatically. If you press and release BTNU once, your system should count up once. If you press and release BTND once, your system should count down once. If you keep pressing BTNU or BTND, your counter should not do any counting. Counting is executed once you release the button.

You should condition the output of the push-button switches before you use them as inputs to your system since pressing these buttons once actually generates many glitches (many negative and positive levels and edges). You can use a shift-register to detect if the button is pressed and then generate a clean input signal.

In order to use the push buttons as inputs, they must be debounced so that a single push of the button results in a clean signal. One way to debounce a switch signal is to shift the bouncing signal down to a shift register. Then, the outputs of all the flip-flops can be ANDed to generate a clean signal (Figure 58). Assume that the bounce interval of a switch has been experimentally determined to be about 10 ms. That is the output of the switch is stabilized 10 ms after acting on the switch. If we sample the switch value every 100 Hz, and record the last four or more switch values in a shift register and AND these four or more recent switch results, bouncing on the mechanical switch will be filtered out and the AND gate will give us a clean signal with one edge for every press of the switch. The fixed frequency of the global clock that the FPGA uses, which is generated by the Nexys3 board, is 100 MHz. We should design a clock divider circuit using an N-bit counter that will divide the 100 MHz clock into a clock rate suitable for switch debouncing (approximately to 100 Hz by dividing it by  $2^{20}$  times). Note that an n-bit counter divides the frequency of a clock signal by  $2^n$  times. Design a switch debouncing circuit in VHDL using a 4-bit shift register and an AND gate. The output of this circuit should control your counter.

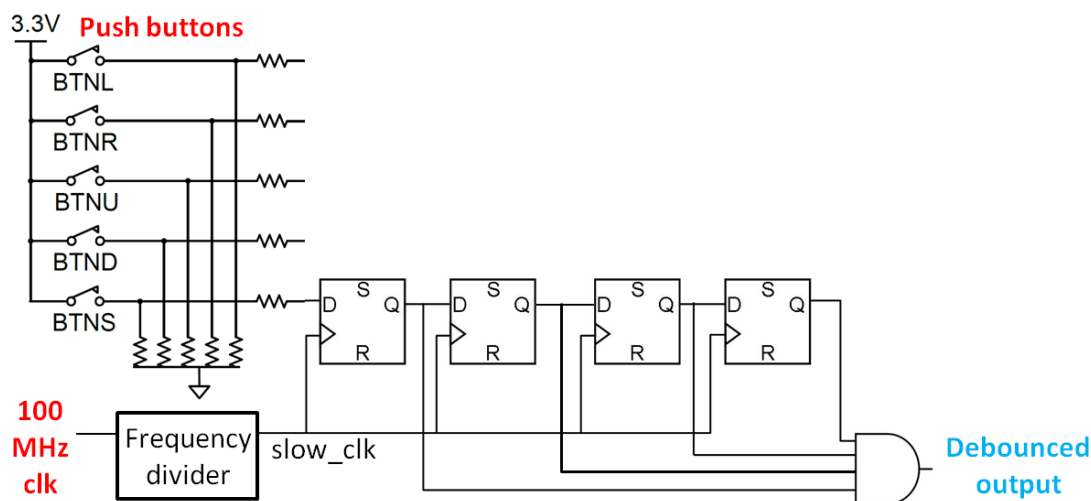


Figure 58. A switch debouncing circuit example

Also eight slide switches are available on the FPGA board. When closed (ON), each switch pulls the connected pin of the FPGA to ground through a resistor. The pin is pulled high through a resistor when the switch is open (OFF). You will use eight of these switches as the 8 bit input to be parallel loaded to your counter. The format of the 8-bit input should be in BCD. You do not need to implement debouncer circuits for them.

### 12.2.2 UP/DOWN BCD COUNTER

Your counter should count up and down in two-digit BCD format. It should count from 00 (0000-0000) to 99 (1001-1001) as a push-button is pressed and released. You can achieve this by implementing a BCD up/down counter. Alternatively, you can design a 7 or 8 bit binary up/down counter. Then, connect a binary to BCD converter to its outputs. In this case, you can enter input from slide switches in the binary format. Also, you should not worry about binary values exceeding 99. A behavioral VHDL code for an up-counting single digit BCD counter is given below. You can modify this code to realize the desired BCD counter in this lab.

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

entity upBCD is
    port( clock_enable: in std_logic;
          clock: in std_logic;
          reset: in std_logic;
          content: out std_logic_vector(3 downto 0));
end upBCD;

architecture behavioral of upBCD is
    signal temp: std_logic_vector(3 downto 0);
begin
    process(clock, reset)
    begin
        if reset='1' then
            temp <= "0000";
        elsif(clock'event and clock='1') then
            if clock_enable='0' then
                if temp="1001" then temp<="0000";
                else temp <= temp + "0001";
                end if;
            else temp <= temp;
            end if;
        end if;
    end process;
    content <= temp;
end Behavioral;
```

### 12.2.3 WRITING THE COUNTER CONTENT TO THE LED DISPLAY

As done before in Lab 5, the contents of the counter are displayed on the four-digit-seven-segment LED display on the FPGA board using Digit0 and Digit1. In order to do that, a BCD-to-7-segment decoding is necessary. After adding the decimal point bit, 8 bits are required to show each digit. We want to display two digits. Thus, your design should have 16 bit output. The first 8 bit (least significant digit) is connected to Digit0. The second 8 bit (most significant digit) is connected to Digit1. The entity of your design should have two sets of 8-bit output ports. These outputs are connected to the seven-segment display driver, *nexys3\_sseg\_driver.vhd*, given and explained in Lab 5.

### 12.2.4 OVERALL DESIGN

As explained above your design accepts 8-bit input, named pdata, one clock signal named board\_clk, three control signals named pload, upcount and downcount and

generate 8-bit lowdig output and 8-bit highdig output. Then, these outputs will be inputs to the design given in the *nexys3\_sseg\_driver.vhd* file using structural vhd style. This design will generate the SSEG\_CA(7 downto 0) and SSEG\_AN(3 downto 0) outputs using the clock signal. Your VHDL code for this design should have the following structure. You should complete the architecture. Your VHDL code can use structural and/or dataflow and/or behavioral style VHDL.

- Synthesizable 2-Digit, parallel loadable up/down BCD Count by pressing push-button switches
- EE240 class Bogazici University
- Implemented on Xilinx Spartan VI FPGA chip

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;
USE ieee.std_logic_arith.all ;
USE ieee.std_logic_unsigned.all;

entity ee240_bcdcounter is
    port (
        pdata: in std_logic_vector(7 downto 0);
        pload: in std_logic;
        upcount: in std_logic;
        downcount: in std_logic;
        board_clk: in std_logic;
        SSEG_CA: out std_logic_vector(7 downto 0);
        SSEG_AN: out std_logic_vector(3 downto 0));
end;
```

architecture arch\_BCD\_counter of ee240\_bcdcounter is

```

-- pdata(7) comes from slide SW-7 (slide Switch)
-- pdata(6) comes from slide SW-6 (slide Switch)
-- pdata(5) comes from slide SW-5 (slide Switch)
-- pdata(4) comes from slide SW-4 (slide Switch)
-- pdata(3) comes from slide SW-3 (slide Switch)
-- pdata(2) comes from slide SW-2 (slide Switch)
-- pdata(1) comes from slide SW-1 (slide Switch)
-- pdata(0) comes from slide SW-0 (slide Switch)

-- pload comes from BTNL (Push-Button Switch)
-- upcount comes from BTNU (Push-Button Switch)
-- downcount comes from BTND (Push-Button Switch)

-- board_clk comes from the 100 MHz board clock connected to V10

--lowdig(0) is displayed on Digit0 Segment CA
--lowdig(1) is displayed on Digit0 Segment CB
--lowdig(2) is displayed on Digit0 Segment CC
--lowdig(3) is displayed on Digit0 Segment CD
--lowdig(4) is displayed on Digit0 Segment CE
--lowdig(5) is displayed on Digit0 Segment CF
--lowdig(6) is displayed on Digit0 Segment CG
--lowdig(7) is displayed on Digit0 Segment DP, which is always logic 1

--highdig(0) is displayed on Digit1 Segment CA
--highdig(1) is displayed on Digit1 Segment CB
--highdig(2) is displayed on Digit1 Segment CC
--highdig(3) is displayed on Digit1 Segment CD
```

--highdig(4) is displayed on Digit1 Segment CE  
 --highdig(5) is displayed on Digit1 Segment CF  
 --highdig(6) is displayed on Digit1 Segment CG  
 --highdig(7) is displayed on Digit1 Segment DP, which is always logic 1

-- lowdig(7 downto 0), highdig(7 downto 0) will be input to the design  
 -- given in the *nexys3\_sseg\_driver.vhd* file using structural vhd style. This design will generate  
 -- the SSEG\_CA(7 downto 0) and SSEG\_AN(3 downto 0) outputs using the clock signal.

end arch\_BCD\_counter;

### 12.2.5 PIN ASSIGNMENT

After successfully completing the design, compilation, ISim simulations and synthesis, you should show that your design work on the FPGA board. By looking at the manual of the Nexys3 Board, we can find which FPGA pins are connected to which components. The following is a list of pin connection we want for this lab and their meanings.

pdata(7) -> T5	Slide SW-7 (Slide Switch)-> T5
pdata(6) -> V8	Slide SW-6 (Slide Switch)-> V8
pdata(5) -> U8	Slide SW-5 (Slide Switch)-> U8
pdata(4) -> N8	Slide SW-4 (Slide Switch)-> N8
pdata(3) -> M8	Slide SW-3 (Slide Switch)-> M8
pdata(2) -> V9	Slide SW-2 (Slide Switch)-> V9
pdata(1) -> T9	Slide SW-1 (Slide Switch)-> T9
pdata(0) -> T10	Slide SW-0 (Slide Switch)-> T10
pload -> C4	BTNL (Push-Button Switch)-> C4
upcount -> A8	BTNU (Push-Button Switch)-> A8
downcount -> C9	BTND (Push-Button Switch)-> C9
board_clk -> V10	100 MHz board clock -> V10
SSEG_CA<0> -> T17	
SSEG_CA<1> -> T18	
SSEG_CA<2> -> U17	
SSEG_CA<3> -> U18	
SSEG_CA<4> -> M14	
SSEG_CA<5> -> N14	
SSEG_CA<6> -> L14	
SSEG_CA<7> -> M13	
SSEG_AN<0> -> N16	
SSEG_AN<1> -> N15	
SSEG_AN<2> -> P18	
SSEG_AN<3> -> P17	

The .ucf file should be like this.

```
#
# Pin assignments for the Nexys3 Board.
#
net pdata<7>      loc=T5; # Slide SW-7 (Slide Switch)
net pdata<6>      loc=V8; # Slide SW-6 (Slide Switch)
net pdata<5>      loc=U8; # Slide SW-5 (Slide Switch)
```

```

net pdata<4>      loc=N8; # Slide SW-4 (Slide Switch)
net pdata<3>      loc=M8; # Slide SW-3 (Slide Switch)
net pdata<2>      loc=V9; # Slide SW-2 (Slide Switch)
net pdata<1>      loc=T9; # Slide SW-1 (Slide Switch)
net pdata<0>      loc=T10; # Slide SW-0 (Slide Switch)

```

```

net pload          loc=C4; # BTNL (Push-Button Switch)
net upcount        loc=A8; # BTNU (Push-Button Switch)
net downcount      loc=C9; # BTND (Push-Button Switch)

```

```

net board_clk      loc=V10; # 100 MHz board clock

```

```

Net SSEG_CA<0> LOC=T17 | IOSTANDARD=LVCMOS33; # Connected to CA
Net SSEG_CA<1> LOC=T18 | IOSTANDARD=LVCMOS33; # Connected to CB
Net SSEG_CA<2> LOC=U17 | IOSTANDARD=LVCMOS33; # Connected to CC
Net SSEG_CA<3> LOC=U18 | IOSTANDARD=LVCMOS33; # Connected to CD
Net SSEG_CA<4> LOC=M14 | IOSTANDARD=LVCMOS33; # Connected to CE
Net SSEG_CA<5> LOC=N14 | IOSTANDARD=LVCMOS33; # Connected to CF
Net SSEG_CA<6> LOC=L14 | IOSTANDARD=LVCMOS33; # Connected to CG
Net SSEG_CA<7> LOC=M13 | IOSTANDARD=LVCMOS33; # Connected to DP

```

```

Net SSEG_AN<0> LOC=N16 | IOSTANDARD=LVCMOS33; # Connected to AN0
Net SSEG_AN<1> LOC=N15 | IOSTANDARD=LVCMOS33; # Connected to AN1
Net SSEG_AN<2> LOC=P18 | IOSTANDARD=LVCMOS33; # Connected to AN2
Net SSEG_AN<3> LOC=P17 | IOSTANDARD=LVCMOS33; # Connected to AN3

```

### 12.3 PREPARATION (PRELAB)

#### For Frequency Divider and Debouncer Circuit

- Write a VHDL code for these components.
- Run ISim and verify that they function as expected by running test sequences.

#### For parallel loadable 2-digit up/down BCD counter

- Write a VHDL code for this component.
- Run ISim and verify that it functions as expected by running a few test sequences.

#### For the BCD-to7-Segment Conversion

- Use the BCD-to\_seven\_segment code used before.

#### For the Overall Design

- Combine the design components under ee240\_bcdcounter architecture and run ISim on it and verify that it functions as expected by running a few test sequences.

### 12.4 IN LAB

- Use Xilinx ISE to synthesize the design based on Xilinx Spartan VI family. Use either the given ucf file or enter pins manually to make pin assignment.
  - Generate the bit file and upload it to your FPGA.
  - On the hardware show that your design works.
  - See if the LEDs show what you expect every time you apply your data using Slide and push-button switches.



## 13 LABORATORY 7 – SIMPLE VGA DRIVER TO DISPLAY 256 DIFFERENT COLORS

### 13.1 DESCRIPTION

The purpose of this lab is to design a VGA driver to display 256 different colors on a monitor. Two timing signals are generated in this system, vsync and hsync to synchronize the plotting of vertical and horizontal pixels, respectively. The color data of each pixel is generated using an 8 bit counter whose first 3-bits are for red, next 3-bits are for green and last 2-bits are for blue color data while the synchronization signals are generated. This way, as the counter counts, 256 different color combinations are displayed on the screen one after another. This counter is implemented on the FPGA board. Overall schematic for the system is shown in Figure 59.

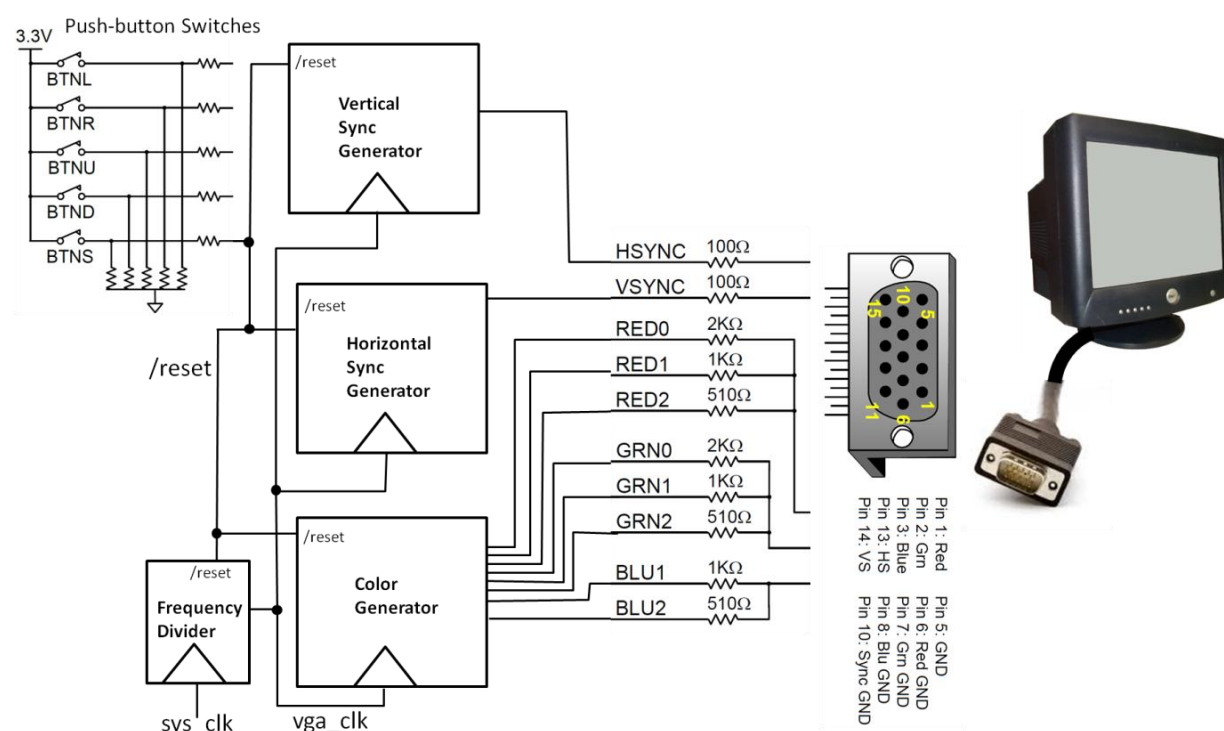


Figure 59. System overview of Lab 7

### 13.2 VGA DRIVER

FPGA can generate a video signal to display it on a VGA monitor. The FPGA board has three bits of red, three bits of green, and two bits of blue color information connected to a simple Digital to Analog Converter (DAC) of resistor-ladder type as shown in Figure 60. This provides a palette of  $2^3 \times 2^3 \times 2^2 = 256$  different colors. The three analog outputs (for red, green and blue color) of this simple DAC are sent to the RGB inputs of a VGA monitor. The FPGA must also generate horizontal and vertical synchronization pulses [9]. These five signals (including the two synchronization signals) typically drive an electron gun that emits electrons, which paint one primary color at a point on a Cathode Ray Tube (CRT) monitor. Signal levels between 0 (completely dark) and 0.7 V (maximum brightness) control the intensity of each color component. This makes the color of a dot (or pixel) on the monitor screen [10]. Even though, display operation of an

LCD monitor is different than the CRT one, same signals and methods are also used for LCD monitors that are driven through VGA ports.

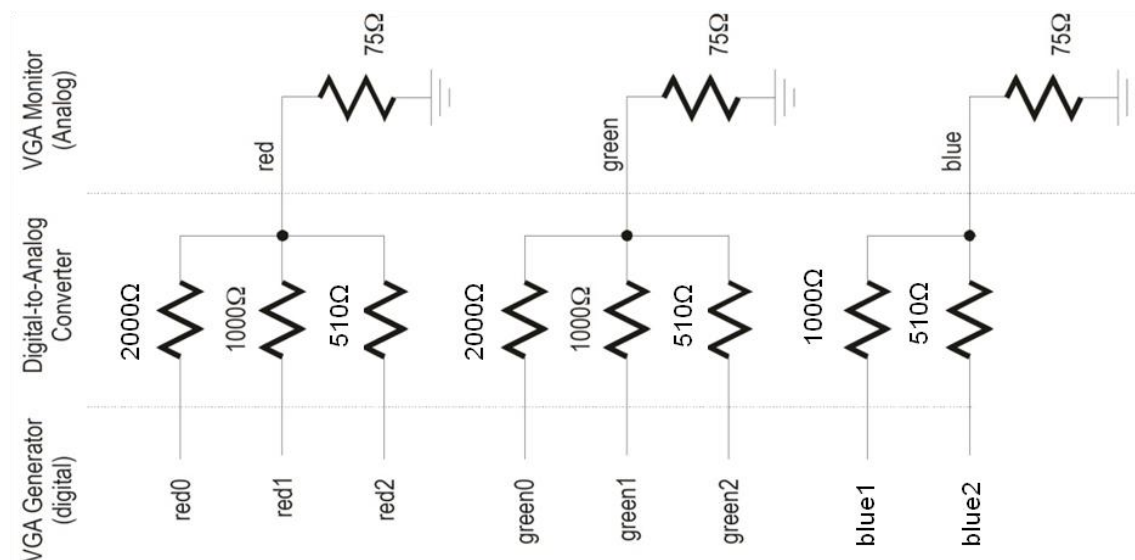


Figure 60. Digital to Analog Conversion done on the FPGA board and signals going to the VGA port of a monitor.

### 13.2.1 VGA SIGNAL TIMING

An image (or frame) on a monitor screen is composed of  $h$  lines. Each line has  $w$  pixels. VGA frame size is shown as  $w \times h$  with typical sizes of  $640 \times 480$ ,  $800 \times 600$ ,  $1024 \times 768$  and  $1280 \times 1024$ . In this lab, we aim for  $640 \times 480$  size. In order to paint a frame, there are deflection circuits in the CRT monitor that move the beams of electrons from left-to-right and top-to-bottom across the screen (totally three beams for three fundamental colors). These circuits require two synchronization signals in order to start and stop the beams at the right time so that a line of pixels is plotted across the monitor and the lines stack up from the top to the bottom to form an image. An example for the timing of the VGA synchronization signals is shown in Figure 61 [10].

Each synchronization signal is composed of two periodic negative pulses. Negative pulses on the horizontal synchronization signal mark the start and end of a line and ensure that the monitor displays the pixels between the left and right edges of the visible screen area. Only certain scanning angle range of the electron beam is visible to viewers. The start and end information of this visible scanning angle of the beam movement are given by the synchronization signals. The pixels are sent on the red, green and blue signal lines within a  $25.17 \mu\text{s}$  window. After this, an interval called front porch of  $0.94 \mu\text{s}$  is inserted before the horizontal synchronization signal pulses stays low for  $3.77 \mu\text{s}$ . After an interval called back porch of  $1.89 \mu\text{s}$ , the next line of pixels begins. Therefore, a single line of pixels occupies  $25.17 \mu\text{s}$  of a  $31.77 \mu\text{s}$  interval. The red, green and blue signals are blanked during the  $6.6 \mu\text{s}$  interval comprised of the front porch, synchronization pulse and back porch [10].

In a similar fashion, negative pulses on the vertical sync signal mark the start and end of a frame made up of video lines and ensure that the monitor displays the lines between the top and bottom edges of the visible monitor screen. The lines are displayed within a  $15.25 \text{ ms}$  window. After this, a front porch interval of  $0.45 \text{ ms}$  is inserted before the vertical sync signal pulses stay low for  $64 \mu\text{s}$ . After a back porch

interval of 1.02 ms, the next frame begins. Therefore, a single frame of pixels occupies 15.25 ms of a 16.784 ms interval (almost 60 Hz refresh rate). The red, green and blue signals are blanked during the 1.534 ms interval comprised of the front porch, synchronization pulse and back porch [10].

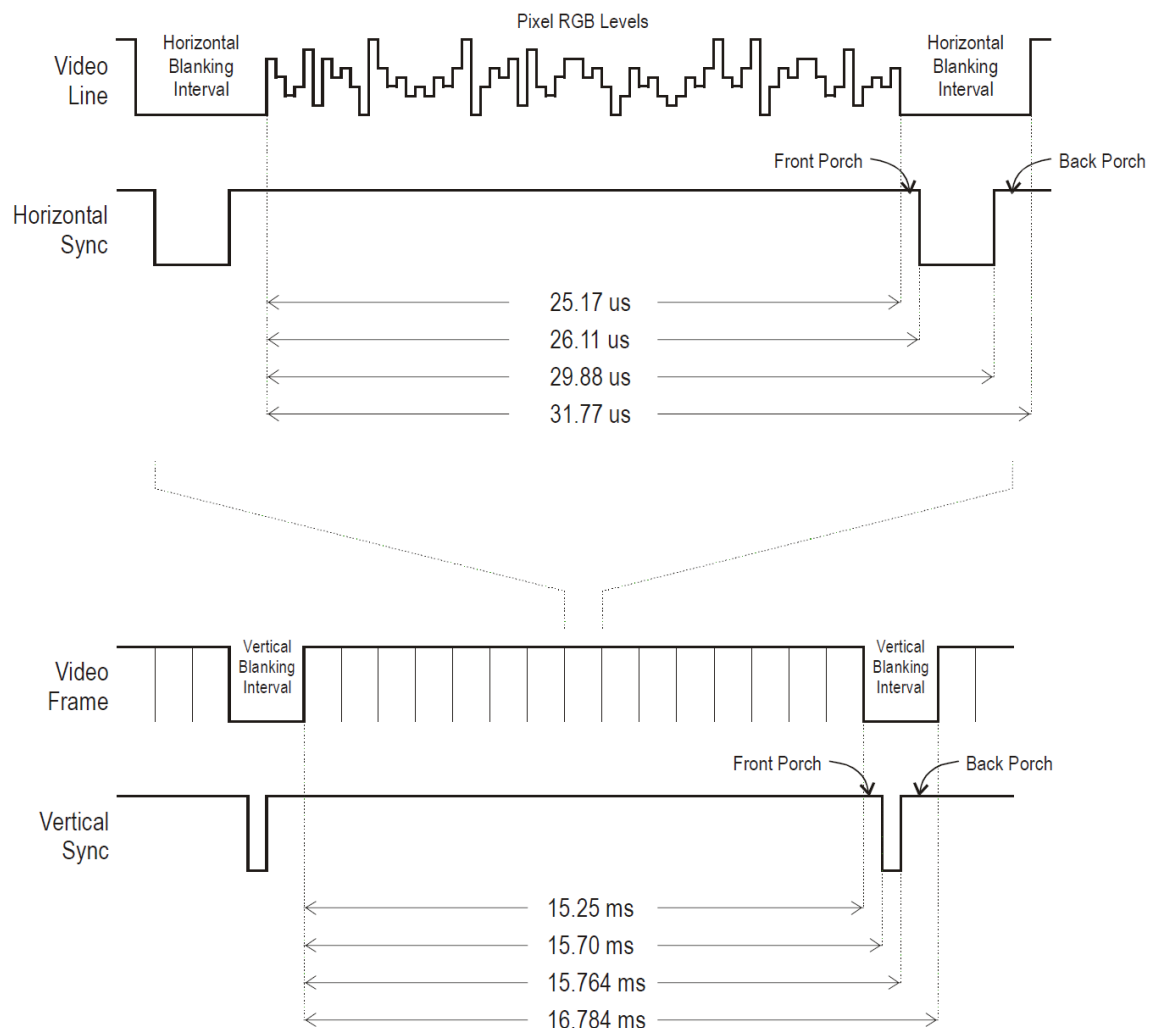


Figure 61. VGA Signal Timing [10].

The timing values given above are just part of one possible solution to drive a VGA monitor. Another possible timing summary is given in Table 13. Timing parameters used in this table is explained in Figure 62. The given timing summary is for a 25 MHz pixel clock and 60 Hz  $\pm 1$  refresh rate. These timing parameters are found by observing various VGA monitors. You are advised to use these parameters in your design. You may divide the 100 MHz system clock to generate the 25MHz pixel clock. Generally, a counter clocked by this pixel clock controls the horizontal timing. Content of this counter is used by logic gates to generate the horizontal synchronization signal. It tracks the current pixel display location on a given row. A separate counter with the same structure as the horizontal synchronization counter tracks the vertical timing. The vertical synchronization counter increments with each horizontal synchronization pulse. Logic gates are used to generate the vertical synchronization signal from the content of this counter. This counter tracks the current display row [11].

Table. 13. Timing summary for a 640x480 VGA, 25 MHz pixel clock [11]

25 MHz pixel clock and 60 Hz $\pm 1$ refresh						
Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
$T_S$	Sync pulse time	16.7ms	416,800	521	32 $\mu$ s	800
$T_{DISP}$	Display time	15.36ms	384,000	480	25.6 $\mu$ s	640
$T_{PW}$	Pulse width	64 $\mu$ s	1,600	2	3.84 $\mu$ s	96
$T_{FP}$	Front Porch	320 $\mu$ s	8,000	10	640 ns	16
$T_{BP}$	Back Porch	928 $\mu$ s	23,200	29	1.92 $\mu$ s	48

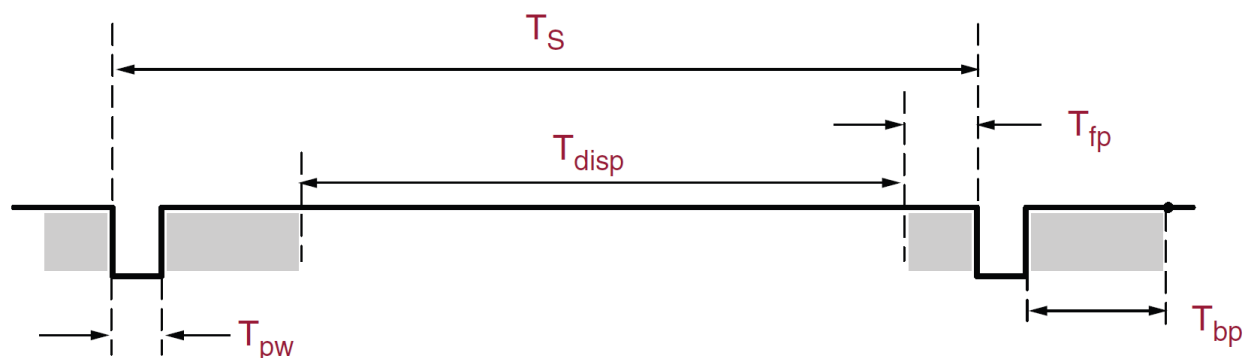


Figure 62. Explanations of timing parameters in Table 13 [11]

### 13.2.2 OVERALL DESIGN

Horizontal and vertical synchronization signals are simply signals with just two-negative-pulses. Vertical and horizontal synchronization generator circuits generate these two pulse signals. These circuits are identical to each other except for the values that determine the pulse timing. In your design you may want your horizontal synchronization generator to communicate with your vertical synchronization generator. For example, the horizontal sync generator may output a single-cycle signal coincident with the leading edge of the horizontal sync pulse to the vertical sync generator. This signal may be connected to the clock-enable of the vertical sync generator so it only updates its timing counter once per line of pixels. Furthermore, a single-cycle signal may be generated in the vertical sync generator to be used as an end-of-frame indicator to the color generator to reset its contents so the VGA generator starts from a completely cleared state on every frame [10].

Color generator is simply an 8-bit binary up-counter. The first three bits of the counter form the RED0, RED1 and RED2 signals. The next three bits form the GREEN0, GREEN1 and GREEN2 signals. Last two bits form BLUE0 and BLUE1 signals. The counter rolls back when it reaches the maximum count. This way, as the counter counts up, all possible color combinations are displayed on the screen. It is not important from which count the counter starts with respect to the synchronization signals. The counter simply creates data to plot on the monitor. Otherwise, to display a meaningful image on the monitor,  $640 \times 480 \times 3 \times 3 \times 2 = 5,529,600$  bits have to be presented for each frame (every 0,0167 second). This can be done by writing the image data to an SRAM and reading from it. However, since this would complicate the laboratory assignment, we form display information by using a continuously running counter.

The clock signal for the timing signals and counters can be generated from system clock (100MHz, pin V10) of the Nexys3 board.

Push button on the center, BTNS, of the Nexys3 board is used to reset the system, putting the circuit into a known state.

Your VHDL code for this design should have the following structure. You should complete the architecture. Your VHDL code can use structural and/or dataflow and/or behavioral style VHDL.

```
-- Synthesizable Simple VGA Driver to Display All Possible Colors
-- EE240 class Bogazici University
-- Implemented on Xilinx Spartan VI FPGA chip
```

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;
USE ieee.std_logic_arith.all ;
USE ieee.std_logic_unsigned.all;

entity ee240_vgadriv er is
    port (
        nreset: in std_logic;
        board_clk: in std_logic;
        vsync: out std_logic;
        hsync: out std_logic;
        red: out std_logic_vector(2 downto 0);
        green: out std_logic_vector(2 downto 0);
        blue: out std_logic_vector(1 downto 0));
end;
```

```
architecture arch_vga_driver of ee240_vgadriv er is
```

```
-- nreset comes from BTNS (Push-Button Switch)
```

```
-- board_clk comes from the 100 MHz board clock
```

```
--vsync is connected to VGA-VSYNC#
--hsync is connected to VGA-HSYNC#
--red(0) is connected to VGA-RED0
--red(1) is connected to VGA-RED1
--red(2) is connected to VGA-RED2
--green(0) is connected to VGA-GRN0
--green(1) is connected to VGA-GRN1
--green(2) is connected to VGA-GRN2
```

```
--blue(0) is connected to VGA-BLU1
--blue(1) is connected to VGA-BLU2
```

```
end arch_vga_driver;
```

### 13.2.3 PIN ASSIGNMENT

After successfully completing the design, compilation, ISim simulations and synthesis, you should show that your design work on the FPGA board. By looking at the manual of the Nexys3 Board, we can find which FPGA pins are connected to which components. The following is the ucf file for this lab including the explanations for the pin assignments.

```
#
# Pin assignments for the Nexys3 Board.
#
net nreset          loc=B8; # Push-Button BTNS

net board_clk       loc=V10; # 100 MHz board clock

net vsync           loc=P7; # VGA-VSYNC
net hsync           loc=N6; # VGA-HSYNC
net red<0>           loc=U7; # VGA-RED0
net red<1>           loc=V7; # VGA-RED1
net red<2>           loc=N7; # VGA-RED2
net green<0>         loc=P8; # VGA-GRN0
net green<1>         loc=T6; # VGA-GRN1
net green<2>         loc=V6; # VGA-GRN2
net blue<0>          loc=R7; # VGA-BLU1
net blue<1>          loc=T7; # VGA-BLU2
```

## 13.3 PREPARATION (PRELAB)

### For Frequency Divider

- Write a VHDL code for these components.
- Run ISim and verify that they function as expected by running test sequences.

### For Horizontal and Vertical Synchronization Signal Generators

- Write a VHDL code for this component.
- Run ISim and verify that it functions as expected by running a few test sequences.

### For the Color Generator

- Write a VHDL code for a resettable 9-bit up-counter.
- Run ISim and verify that it functions as expected by running a few test sequences.

### For the Overall Design

- Combine the design components under arch\_vga\_driver architecture and run ISim on it and verify that it functions as expected by running a few test sequences.

## 13.4 IN LAB

- Use Xilinx ISE to synthesize the design based on Xilinx Spartan VI family. Use either the given ucf file or enter pins manually to make pin assignment.
- Generate the bit file and upload it to your FPGA.
- On the hardware show that your design works.

---

## 14 LABORATORY 8 – STUDENT DESIGN PROJECT

---

### 14.1 LAB MATERIALS

Of your own design

### 14.2 CONCEPT

The aim of this lab is to let you design and implement your own circuit with desired functionality. However, there are certain restrictions on the circuit:

- The circuit should have some sort of an external input – a switch, a sensor, etc.
- The circuit should have some sort of display – LEDS, etc.
- The circuit should have its CAD design fully implemented.
- The circuit should be demonstrated.
- The circuit should have a full component cost analysis included.

---

## **15 FINAL PROJECT LAB REPORT**

---

The final project laboratory report is to be in the following format:

- It should be as concise as possible.
- It should contain your design and lab details.



**BOĞAZİÇİ UNIVERSITY****ELECTRICAL ELECTRONIC ENGINEERING  
DEPARTMENT**

---

**EE240-DIGITAL DESIGN FINAL LAB REPORT**

---

Number of The Experiment:

Name of The Experiment:

Name of The Student:

E-mail:

Name of The Partner:

Name of The Lab. Assistant:

Date of The Experiment:	
Dadline for The Submission of the Report	
Date of The Submission of the Report:	
Delay	

**Grading**

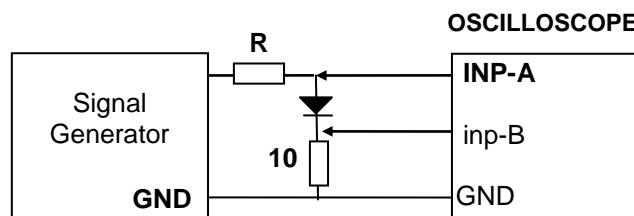
General		/20
Data		/30
Discussion		/20
Answers		/30
Total	0	/100
Delay	0	
SCORE		/100

## 1. LAB MATERIAL

- 1.1 Oscilloscope : Hameg 203A, 2x20MHz
- 1.2 Multimeter: HP973A, True RMS
- 1.3 .....
- ...

## 2. THEORY AND METHOD

- \* Do not write the same information which is written in the Lab. book
- \* Write the objective of the experiment in your words
- \* Draw the experimental setups indicating the instruments
- \* Calculations done before the experiment



**EXAMPLE OF EXPERIMENTAL SET-UP**

## 3. EXPERIMENTAL DATA

- \* All computed values in tabular form
- \* All measured values in tabular form
- \* Draw all curves with titles, units and scales

## 4. ANALYSIS AND DISCUSSION

- \* Discussion of the experimental results. (IF there are unexpected results, explain the possible causes)
- \* Compare the experimental and theoretical results
- \* Explain the probable causes of errors
- \* Personal opinions about the experiment

## 5. ANSWERS TO THE QUESTIONS

- \* Answer all of the questions.

---

## 16 REFERENCES

---

1. National Instruments, "SC-2075 SIGNAL CONDITIONING ACCESSORY- User Guide", p.5, 9, April 2000
2. <http://sine.ni.com/images/products/us/sc2075m.jpg>
3. Maravar, Ravi. Gretchen Edelman, Carl Nybro, Scott Duncan, Campbell Britton. "Combining the use of Multisim and LabVIEW for Circuit Design and Analysis in Undergraduate Electronics Curriculum", p.2, July 2003.
4. [www.kpsec.freeuk.com/ images/bblinks.gif](http://www.kpsec.freeuk.com/images/bblinks.gif)
5. <http://snapcircuits.net/elenco/Breadboard%20Wire.jpg>
6. <http://www.mathmachines.net/construction/Breadboard/BBA02.jpg>
7. [http://sine.ni.com/apps/we/niepd\\_web\\_display.display\\_epd4?p\\_guid=B45EACE3DDD956A4E034080020E74861&p\\_node=DZ53016&p\\_source=external](http://sine.ni.com/apps/we/niepd_web_display.display_epd4?p_guid=B45EACE3DDD956A4E034080020E74861&p_node=DZ53016&p_source=external)
8. NEXYS 3 Manual, Digilent Corporation, Revision January 7<sup>th</sup>, 2013, [http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3\\_rm.pdf](http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf)
9. XSA-200 Board V1.3 User Manual, XESS Corporation, 2005.
10. <http://www.xess.com/appnotes/an-101204-vgagen.pdf>
11. [http://www.xilinx.com/support/documentation/boards\\_and\\_kits/ug230.pdf](http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf) (pages 56-58)