Practical Workbook

Digital Logic Design

Name	:
Year	:
Batch	:
Roll No	:
Department:	

Dept. of Computer & Information Systems Engineering

NED University of Engineering & Technology,

Karachi – 75270, Pakistan

INTRODUCTION

The digital logic design area covers the digital building blocks, tools, and techniques in the design of computers and other digital systems. Digital logic design is one of the topic areas that differentiate computer engineers from electrical engineers and computer scientists. It covers a variety of basic topics, including switching theory, combinational and sequential logic circuits, and memory elements.

In this Practical Workbook, laboratory sessions based on both combinational and sequential logic are covered. The lab sessions fall into three categories:

1. Hardware implementation and IC testing. It covers basic circuit building on a bread board and testing of various MSI ICs including registers and different types of counters.
2. Logic circuit simulation on a CAD software – Electronics Workbench (EWB). EWB is excellent simulation software, where circuits can be designed and tested before physical implementation. Various laboratory sessions of this workbook provide activities and exercises on EWB.
3. Introduction to Verilog and emulation of Verilog code on Xilinx's ISE simulator. Verilog is a hardware description language (HDL) most commonly used in the design, verification, and implementation of digital logic circuits. Hardware description languages, differ from software programming languages because they include ways of describing the propagation of time and signal dependencies. This work book includes lab sessions that cover how to write Verilog codes for fundamental logic circuits and their emulation on Xilinx's ISE simulator.

All laboratory sessions of this workbook incorporate brief theoretical backgrounds, as details may be covered in the respective theory classes. Exercises / activities are included with almost all the sessions for the students to practice.

Three appendices are also included in this workbook. The first one provides pin diagrams for all the ICs required for the laboratory work provided in this workbook. It will help the students in preparing the pin diagrams for the circuits. Second appendix covers hardware equipment /components other than ICs that are commonly required in building circuits / mini projects. Third appendix discusses generation of square wave via 555 timer IC and a hardware debouncing circuit for mechanical switches as such switches are extensively used for input purpose in logic circuits.

CONTENTS

Lab Session 01

OBJECT

Working with Electronics Workbench

ELECTRONICS WORKBENCH - EWB

Electronics Workbench is a computer aided design tool that provides you with all the components and instruments necessary to create board-level designs. It has complete mixed analog and digital simulation and graphical waveform analysis, allowing you to design your circuit and then analyze it using different simulated instruments and analysis options. It is fully integrated and interactive, thus you can change your circuits quickly, allowing fast and repeated what-if analysis.

Electronics Workbench provides the following kind of components:

- Sources parts bin *(AC voltage source, Vcc source, ground, battery, etc)*
- Basic parts bin *(resistors, capacitors, transformers, switches, etc)*
- Diodes parts bin
- Transistors parts bin
- Analog ICs parts bin *(op-amps, etc)*
- Mixed ICs parts bin *(ADCs, DACs, 555 timers, etc)*
- Digital ICs parts bin *(AND, OR, adders, multiplexers, etc)*
- Indicators parts bin *(voltmeter, ammeter, probe, displays, etc)*
- Controls parts bin *(voltage differentiator, multiplier, etc)*
- Instruments parts bin *(multimeter, oscilloscope, function generator, etc)*
- Miscellaneous parts bin *(write data, textbox, etc)*
    - *Write data:* This component allows you to save simulation results as an ASCII file.
    - *Text Box:* Use this to add descriptive text anywhere in a circuit.

DESIGNING LOGIC CIRCUIT FOR A GIVEN LOGIC EXPRESSION

Given Logic Expression: $F = \overline{A}.B + A.\overline{B}.C$

Procedure

1. From *Logic Gates Parts Bin,* Drag and drop the required logic gates on the design area. Use *Component Properties* dialog box to customize these gates.
2. Connect the terminal of these gates according to the given expression. Use additional connectors form the *Basic Parts Bin* if more than one wire needs to be connected at a single node.
3. Drag and drop a *probe* from *Indicators Parts Bin.* Use *Component Properties* dialog box to customize the color and other properties of the probe. Connect this probe at the output terminal of the circuit to indicate results.
4. Select four *switches* form *Basic Parts Bin*. Specify the key that controls the switch by typing its name in the *Value* tab of the *Component Properties* dialog box. For example, if you want the switch to close or open when digit '1' is pressed, type 1 in the *Value* tab, then click *OK*. Assign different keys to all the switches.
5. Connect the output terminals of the switches to each of the circuit inputs A, B, and C.
6. Drag and drop *Vcc* and *Ground* form the *Sources Parts Bin*.

7. Connect *Vcc* terminal to one end and *Ground* terminal to the other end of all the switches.
8. Label the circuit properly using *text boxes* found in the *miscellaneous parts bin*.
9. Run the circuit using the *Activate Simulation* switch. Use the keys you have assigned to the switches to toggle them between *Vcc* and Ground connections, thus providing 1 or 0 respectively to the inputs. Record the results as indicated by the probe for all possible combinations of 1s and 0s at the inputs.

EWB Circuit

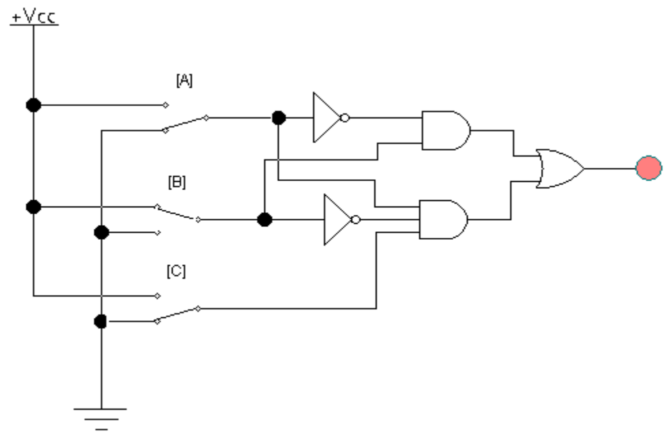


Figure 1.1: *Designed circuit on EWB*

Observations

A	*B*	*C*	*Expected Output*	*Observed Output*
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		
1	1	1		
1	1	1		

USING LOGIC CONVERTER

Logic converter can be used to derive truth table and logic expression for a given logic circuit or vice versa, i.e. if a logic circuit is expressed in any one of the three ways, other two can be directly obtained using the logic converter.

From the *Instruments Parts Bin*, drag and drop *Logic Converter* on the design area. Double click the *Logic Converter* to reveal *Logic Converter* dialog box. This dialog box shows various conversion options between truth table, logic expression and logic circuit.

Finding truth table and logic expression for a given logic circuit:

1. Create any arbitrary logic circuit on the design area.
2. Attach the input terminals of the logic converter to the input points in the circuit.

3. Connect the single output of the circuit to the output terminal on the logic converter icon.
4. Click the Circuit to Truth Table button. The truth table appears in the logic converter's display.
5. To convert this truth table to a Boolean expression, click the *Truth Table to Boolean Expression* button. The Boolean expression will be displayed at the bottom of the logic converter.

Creating logic circuit and truth table for a given logic expression:

1. Enter the given logic expression in the edit box found at the end of the Logic converter dialog box. Use **'** to represent invert of a variable. For example, $\overline{A}$ is written as **A'**.
2. Click the *Boolean Expression to Truth Table* button. The truth table appears in the logic converter's display.
3. Now click the *Boolean Expression to Circuit* button. This creates the logic circuit for the given expression in the design area. Label the diagram if needed.

Creating logic circuit and finding logic expression for a given truth table:

1. Click desired number of input channels from A to H, across the top of the logic converter. The display area below the terminals fills up with the necessary combinations of ones and zeros to fulfill the input conditions. The values in the output column on the right are initially set to 0.
2. Edit the output column to specify the desired output for each input condition. To change an output value, select it and type a new value: 1, 0 or x. An x indicates a don't care condition.
3. To convert this truth table to a Boolean expression, click the *Truth Table to Boolean Expression* button. The Boolean expression will be displayed at the bottom of the logic converter.
4. Simplify the expression by clicking the *Simplify* button.
5. Now click the *Boolean Expression to Circuit* button. This creates the logic circuit for the given expression in the design area.

EXERCISES

1. Create a neat pin diagram (using TTL ICs) for the logic expression $F = \overline{A}.B + A.\overline{B}.C$. Test the output and attach hard copy of the diagram here.

Lab Session 02

OBJECT

Getting Familiar with digital design using Verilog HDL

THEORY

Digital systems are highly complex. At their most detailed level, they may consist of millions of elements, such as a collection of logic gates or transistors. Hardware description languages have evolved to aid in the design of systems with this large number of elements and wide range of electronic and logical abstractions. The creative process of digital system design begins with a conceptual idea of a logical system to be built, a set of constraints that the final implementation must meet, and a set of primitive components from which to build the system. The design is typically divided into many smaller subparts (following the well-known divide-and conquer engineering approach) and each subpart is further divided, until the whole design is specified in terms of known primitive components.

There are two standard HDLs that are supported by IEEE: VHDL and Verilog HDL. The HDL used in our lab will be Verilog.

SOFTWARE USED IN LAB

- ISE-Xilinx (Integrated Synthesis Environment)
- ModelSIM

INTRODUCTION TO VERILOG HDL

Verilog is a Hardware Description Language (HDL) which is used to model electronic systems. It provides the designer entry into the world of large, complex digital systems design. The Verilog language provides the digital system designer with a means of describing a digital system at a wide range of levels of abstraction, and, at the same time, provides access to computer-aided design tools to aid in the design process at these levels. It also fulfils the need for verifying the design for functionality and timing constraints like propagation delay setup and hold times.

The components of the target design can be described at different levels with the help of constructs in Verilog.

1. *Circuit Level:* MOS switch is the basic element which can be used to build basic circuits like inverters, logic gates, 1-bit dynamic and static memories.
2. *Gate Level or structural level:* Design is carried out in terms of basic gates. All basic gates are available as ready modules called "primitives". Primitives can be incorporated into design descriptions directly.
3. *Data Flow Level:* All possible operations on signals and variables are represented in terms of assignments.
4. *Behavioral Level:* This level describes a system by concurrent algorithms and the design description looks like a C program. Compactness and the comprehensive nature of the design description make the development process fast and efficient. Functions, Tasks and Always blocks are the main elements.

There are two types of code in most HDLs:

Structural, this is a verbal wiring diagram without storage.
assign p= q & r | s;
assign s = t & (~r);
In structural code, the order of the statements does not matter. Changing t will change p.

Procedural, which is used for circuits with storage, or as a convenient way to write conditional logic.
always @ (posedge clk) // Execute the next statement on every rising clock edge. **count < = count+1;**

LANGUAGE CONSTRUCTS AND CONVENTIONS IN VERILOG

Verilog has its own constructs and conventions. Any source file in Verilog is made up of number of ASCII characters. These characters are grouped into sets - referred to as "lexical tokens". Verilog has 7 types of lexical tokens - operators, keywords, identifiers, white spaces, comments, numbers and strings. Verilog is case sensitive, so all keywords are in lower case.

White Space Characters

Blanks (\b), tabs (\t), newlines (\n) and formfeed form the white space characters in Verilog. In any design description the white space characters are included to improve readability. White space characters have significance only when they appear in string.

Comments

Comments are incorporated in two ways.

// begins a single line comment, terminated by a *newline*.
/* begins a multi-line block comment, terminated by a */.

For Example;

Module D_FF (Q, DP, CLK); //This is the design of a D flip-flop.	} Single Line Comment
/* Here Q is the output. DP is the input and CLK is the clock.*/	} Multi Line Comment

Keywords

The keywords define the language constructs. All keywords in Verilog are in small letters and require to be used as such.
Some of the examples are:
`module` **-** signifies the beginning of a module definition.
`endmodule` **-** signifies the end of module definition.
`begin` **-** signifies the beginning of block of statements.
`end` **-** signifies the end of a block of statements.
`if` **-** signifies a conditional activity to be checked.
`assign` **-** assigns a value or an expression to a net or variable.

Identifiers

Identifiers are user-defined words for variables, function names, module names, block names and instance names. Identifiers begin with a letter or underscore and can include any number of letters, digits and underscores. Identifiers in Verilog are case-sensitive. Verilog HDL allows any character to be used in an identifier by **escaping** the identifier. Escaped identifiers provide a means of including any of the printable ASCII characters in an identifier. Escaped identifiers start with backslash (\) character.
Examples: clock, enable, adder, gate_1, \reset.

Numbers

The numbers can be of integer type or real type.

Integer Numbers

Number storage is defined as a number of bits, but values can be specified in binary, octal, decimal or hexadecimal. The representation has three tokens with an optional sign preceding it. Numbers may be sized or unsized. Unsized integers default to at least 32 bits.
Syntax: size' base value
Examples:

Integer	Stored As	Description
1	00000000000000000000000000000001	Unsized 32 bits
8'h AA	10101010	Sized hexadecimal
3'b001	001	3 bit number
9'o123	001 010 011	9 bit octal number
9'h?A	zzzzz1010	5 bit Hex number.

Observations:

- The? is another way of representing the **Z** logic value.
- An underscore is ignored (used to enhance readability). The underscore cannot be used as the first character of the value.
- Values are expanded from right to left (lsb to msb).
- When size is fewer bits than value, the upper bits are truncated.
- When size is more bits than value, and the left-most bit of value is **0** or **1**, zeros are left-extended to fill the size.
- When size is more bits than value, and the left-most bit of value is **Z** or **X**, the **Z** or **X** is left-extended to fill the size.
- Signed numbers are interpreted as 2's complement values.

Real Numbers

Real numbers may be specified in either decimal or scientific notation Real Numbers can not contain 'Z' and 'X'.
Syntax: *<Value>.<value>*
<Mantissa>E<exponent>
Examples:

Real Number	Decimal Notation
0.5	0.5
3e4	$3 * 10^4$(30000)
5.8 E -3	$5.8 * 10^{-3}$ (0.0058)

Strings

A string is a sequence of characters enclosed by double quotes. It must be contained on a single line. Special characters may be specified in a string using the "\" escape character as follows:
\n new line character. Typically the **return key.**
\t tab character. Equivalent to typing the tab key.
\\ is the \ character.

\" is the “character.
Verilog treats a string as a sequence of ASCII characters.

Logical Values

Verilog uses a 4 value logic system for modelling.

Logical Value	Description
0	Zero , low or false
1	One, high or true
Z or z	High impedance (tri-state or floating)
X or x	Unknown or uninitialized (don't care)

Logic Strengths

Logic values can have 8 strength levels: 4 driving, 3 capacitive, and high impedance (no strength).

Strength level	Strength name	Specification keyword	Display mnemonics
7	Supply drive	supply0 supply1	Su0 Su1
6	Strong drive	strong0 strong1	St0 St1
5	Pull drive	pull0 pull1	Pu0 Pu1
4	Large capacitor	large0 large1	La0 La1
3	Weak drive	weak0 weak1	We0 We1
2	Medium capacitor	medium0 medium1	Me0 Me1
1	Small capacitor	small0 small1	Sm0 Sm1
0	High impedance	highz0 highz1	Hi0 Hi1

Operators

They are of three types.

- Unary: the unary operator is associated with a single operand. E.g. b = ~a
- Binary: the binary operator is associated with two operands. E.g. c = a & b
- Ternary: the ternary operator is associated with three operands. E.g. a = b? c : d

Operator Precedence

The operator precedence is shown below. The top of the table is the highest precedence, and the bottom is the lowest. Operators listed on the same line have the same precedence. All operators associate left to right in an expression. Ternary operator is exception to this; it associates from right to left. Parentheses can be used to change the precedence or clarify the situation.

Operators	Description	Precedence
~ , ! , & , + , -	Unary	Highest precedence
* , / , %	MUL, DIV, MODULO	Binary
+ , -	Plus minus	
<< , >>	Logical shift left/ right	
<<< , >>>	Arithmetic shift left / right	
< , <= , > , >=	Relative comparisons	
= = , !=	Equality comparisons	
& , ~&	AND , NAND	
^ , ~^	XOR , XNOR	
| , ~|	OR , NOR	
? , :	Ternary Operators	Lowest precedence

Data Types

Verilog has two major data type classes:
1) Net Data type
2) Variable data type.

Nets

Net data types are used to make connections between parts of a design. Nets reflect the value and strength level of the drivers of the net or the capacitance of the net, and do not have a value of their own. A net can be specified in different ways.

Wire

A **wire** (or net) represents a physical wire in a circuit and is used to connect gates or modules. The value of a wire can be read, but not assigned to, in a function or block. A wire does not store its value but must be driven by a continuous assignment statement or by connecting it to the output of a gate or module.
Syntax wire [msb: lsb] wire_variable_list;
Example
wire c; //declare a wire c

Variable data types

Variable data types are used as temporary storage of programming data. Variables can only be assigned a value from within an initial procedure, an always procedure, a task or a function. Variables can only store logic values; they cannot store logic strength. Variables are un-initialized at the start of simulation, and will contain logic X until a value is assigned. Variables can be declared trough a keyword **reg.**

Reg

A **reg** (register) is a data object that holds its value from one procedural assignment to the next. They are used only in functions and procedural blocks. A reg is a Verilog variable type and does not necessarily imply a physical register.
Syntax reg [msb: lsb] reg_variable_list;
Example

```
reg a; // single 1-bit register variable
reg [7:0] r_vector; // an 8-bit vector; a bank of 8 registers.
```

Scalars and Vectors

Entities representing single bits are called “scalars”. Often multiple lines carry signals in a cluster like data bus or address bus. The group of **regs** is treated as “vector”.

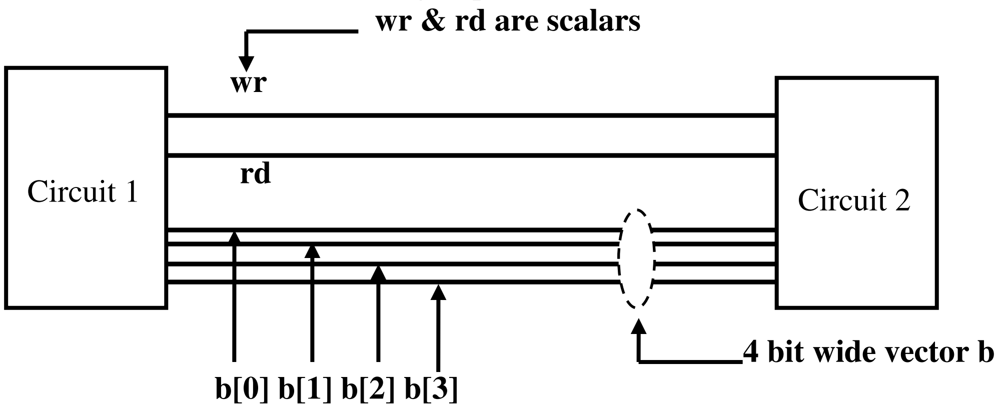


Figure 2.1: *Illustration of scalars and vectors*

Examples:

- **wire** [3:0] a ; /* a is a four bit vector of net type; the bits are designated as a[3] , a[2], a[1] , a[0].*/
- **reg** [2:0] b; /* b is a vector of **reg** type ; the bits are designated as b[2] , b[1] ,
  b [0] */
- **wire** [-2:2] d ; /* d is a 5 bit vector with individual bits designated as d[-2] ,d [-1], d[0] , d[1] , d[2].*/

Whenever a range is not specified for a net or a **reg**, the same is treated as a scalar.

Modules

Verilog HDL models are represented as modules. A module is the principal design entity in Verilog. The first line of a module declaration specifies the name and port list (arguments). The next few lines specify the input/output type (**input**, **output** or **inout)** and width of each port. The default port width is 1 bit. Then the port variables must be declared **wir**e or **reg**. The default is wire. Typically inputs are wire since their data is latched outside the module. Outputs are type **reg** if their signals were stored inside an **always** or **initial** block.

Syntax

```
module module_name (port_list);
input [msb: lsb] input_port_list;  //input ports
output [msb: lsb] output_port_list;  //output ports
inout [msb: lsb] inout_port_list;  // bidirectional ports
... statements ...
endmodule
```

Example*:*

```
module add_sub(add, in1, in2, sum_12);
input add; // defaults to wire
input [7:0] in1, in2; //wire by default
output [7:0] sum_12;
reg sum;
... statements ...
```

```
endmodule
```

Continuous Assignment

The continuous assignment is used to assign a value onto a wire in a module. It is the normal assignment outside of always or initial blocks. Continuous assignment is done with an **assign** statement or by assigning a value to a wire during its declaration. The order of assign statements does not matter. A change in any of the right-hand-side inputs will immediately reflect on the left-hand-side output.

Syntax

```
wire wire_variable = value;
assign wire_variable = expression;
```

```
Example:
wire [1:0] a = 2'b01; // assigned on declaration
assign b = c & d; // using assign statement
assign d = x | y; // The order of the assign statements does
not matter.
```

Behavioral Modeling

Behavioral or procedural statements in Verilog are used to model a design at a higher level of abstraction than the other levels. Procedural assignments are used within Verilog procedures (**always** and **initial** blocks). Only **reg** variables and **integers** can be placed left of the "=" in procedures. The assignment may be Blocking or non-Blocking.

- *variable = expression*;

Blocking procedural assignment. Expression is evaluated and assigned when the statement is encountered. In a *begin—end* sequential statement group, execution of the next statement is blocked until the assignment is complete. In the sequence ;

```
begin
m=n;
n=m;
end
```

Note: The first assignment changes m before the second assignment reads m.

- *variable* **<=** *expression*;

Non-blocking procedural assignment. Expression is evaluated when the statement is encountered, and assignment is postponed until the end of the simulation time-step. In a *begin—end* sequential statement group, execution of the next statement is not blocked; and will be evaluated before the assignment is complete. In the sequence ;

```
begin
m<=n;
n<=m;
end
```

Note: Both assignments will be evaluated before m or n changes.

Procedures: Always and Initial Blocks

The always block is the primary construct in RTL modeling. Like the continuous assignment, it is a concurrent statement that is continuously executed during simulation. This also means that all always blocks in a module execute simultaneously. The always block is triggered to execute by the level, positive edge or negative edge of one or more signals (separate signals by the keyword **or**). A double-edge trigger is implied if you include a signal in the event list of the always statement. The single edge-triggers are specified by **posedge** and **negedge** keywords.

Example:

```
always @(a or b); // level-triggered; if a or b changes levels
always @(posedge clk); // edge-triggered: on +ve edge of clk
```

The initial block is like the always block except that it is executed only once at the beginning of the simulation. It is typically used to initialize variables and specify signal waveforms during simulation. Initial blocks are not supported for synthesis.

Example:

```
inital
begin
clr = 0; // variables initialized at
clk = 1; // beginning of the simulation
end

inital // specify simulation waveforms
begin
a = 2'b00; // at time = 0, a = 00
#50 a = 2'b01; // at time = 50, a = 01
#50 a = 2'b10; // at time = 100, a = 10
end
```

Lab Session 03

OBJECT

Getting familiar with the environment of Xilinx ISE

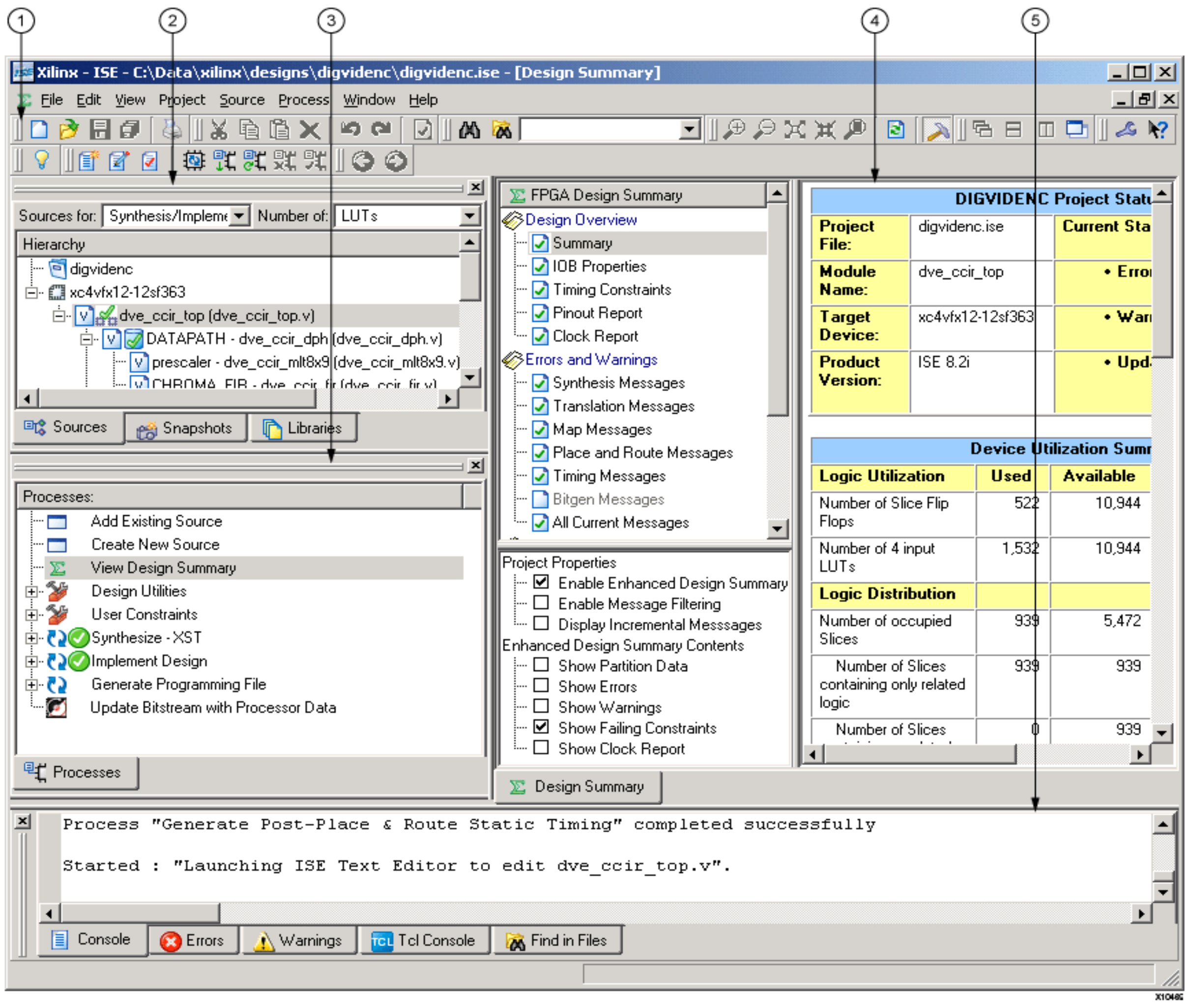
OVERVIEW

The Integrated Software Environment (ISE™) is the Xilinx® design software suite that allows you to take your design from design entry through Xilinx device programming.

GETTING STARTED

To start ISE, double-click the desktop icon, Xilinx ISE 9.1i

Or start ISE from the Start menu by selecting:
Start → All Programs → Xilinx ISE 9.1i → Project Navigator



DIGVIDENC Project Statu

Project File:	digvidenc.ise	Current Sta
Module Name:	dve_ccir_top	• Erro
Target Device:	xc4vfx12-12sf363	• Warı
Product Version:	ISE 8.2i	• Upd

Device Utilization Sumr

Logic Utilization	Used	Available
Number of Slice Flip Flops	522	10,944
Number of 4 input LUTs	1,532	10,944
Logic Distribution		
Number of occupied Slices	939	5,472
Number of Slices containing only related logic	939	939
Number of Slices	0	939

```
Process "Generate Post-Place & Route Static Timing" completed successfully

Started : "Launching ISE Text Editor to edit dve_ccir_top.v".
```

Figure 3.1: *Main ISE Window*

1. Toolbar : Toolbars provide convenient access to frequently used commands
2. Sources window : The Sources tab in the Sources window shows the source files you create and add to your project
3. Processes window: The Process tab shows the available processes in a hierarchical view.

4. Workspace: When you open a project source file, open the Language Templates, or run certain processes, such as viewing reports or logs, the corresponding file or view appears in the Workspace.
5. Transcript window: The Console tab of the Transcript window shows output messages from the processes you run.

ACCESSING HELP

At any time during the tutorial, you can access online help for additional information about the ISE software and related tools. To open Help, do either of the following:

- Press **F1** to view Help for the specific tool or function that you have selected or highlighted.
- Launch the **ISE Help Contents** from the Help menu. It contains information about creating and maintaining your complete design flow in ISE.

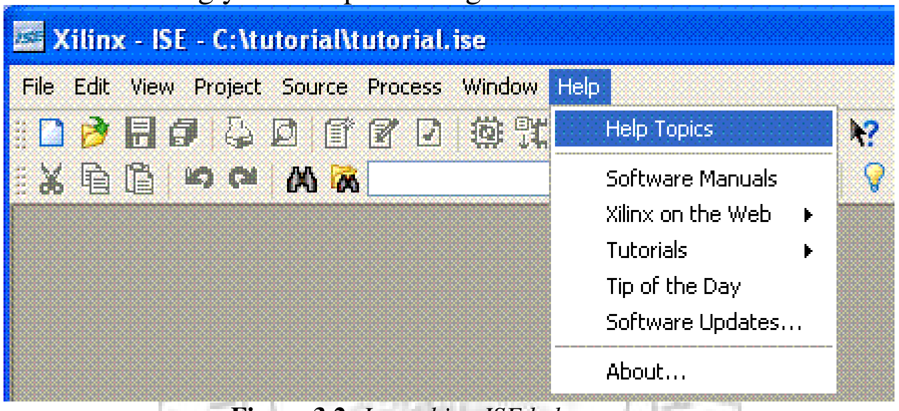
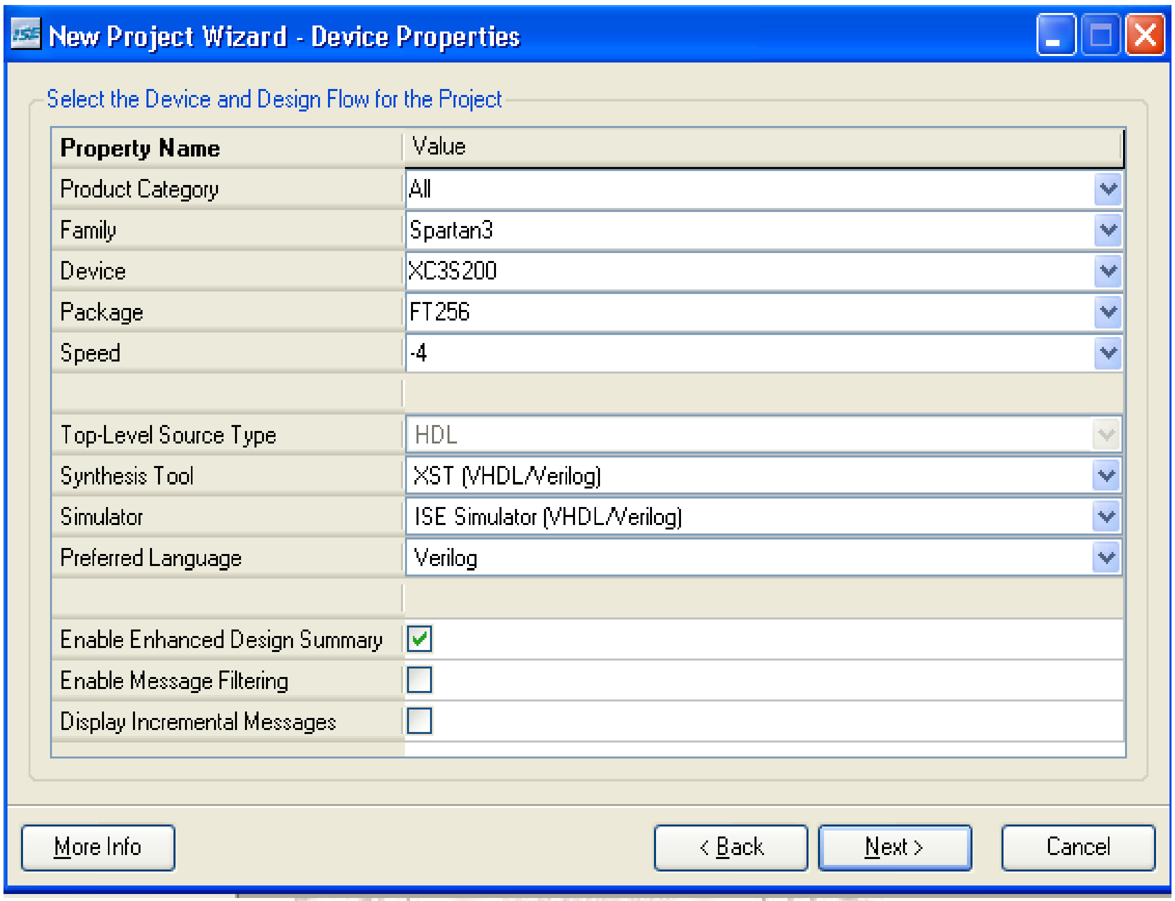


Figure 3.2: *Launching ISE help contents*

CREATE A NEW PROJECT

To create a new project:
1. Select **File** > **New Project...** The New Project Wizard appears.
2. Type **<project_name>** in the Project Name field.
3. Enter or browse to a location (directory path) for the new project. A subdirectory is created automatically.
4. Verify that **HDL** is selected from the Top-Level Source Type list.
5. Click **Next** to move to the device properties page.
6. Fill in the properties in the table as shown below:
   - Product Category: All
   - Family: Spartan3
   - Device: XC3S200
   - Package: FT256
   - Speed Grade: -4
   - Top-Level Source Type: HDL
   - Synthesis Tool: XST (VHDL/Verilog)
   - Simulator: ISE Simulator (VHDL/Verilog)
   - Preferred Language: Verilog
   - Verify that Enable Enhanced Design Summary is selected.

Leave the default values in the remaining fields.
When the table is complete, your project properties will look like the following:


New Project Wizard - Device Properties
Select the Device and Design Flow for the Project
More Info
< Back
Next >
Cancel

Property Name	Value
Product Category	All
Family	Spartan3
Device	XC3S200
Package	FT256
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISE Simulator (VHDL/Verilog)
Preferred Language	Verilog
Enable Enhanced Design Summary	✓
Enable Message Filtering	☐
Display Incremental Messages	☐

Figure 3.3: *Dialog box for project properties*

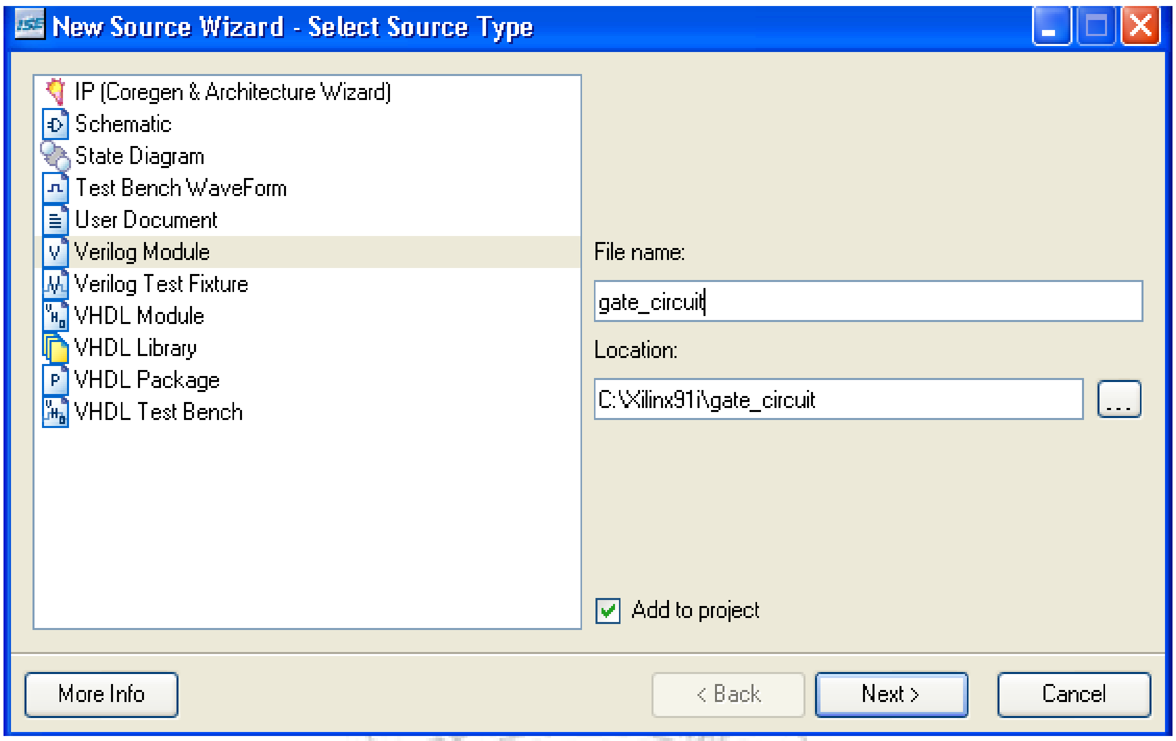
7. Click **Next** to proceed to the Create New Source window in the New Project wizard.

CREATING A VERILOG SOURCE

Create the top-level Verilog source file for the project as follows:

1. Click **New Source** in the New Project dialog box.
2. Select **Verilog Module** as the source type in the New Source dialog box.
3. Type in the file name **<file_name>**.
4. Verify that the **Add to Project** checkbox is selected.
5. Click **Next**.

Declare the ports for the counter design by filling in the port information as shown below:

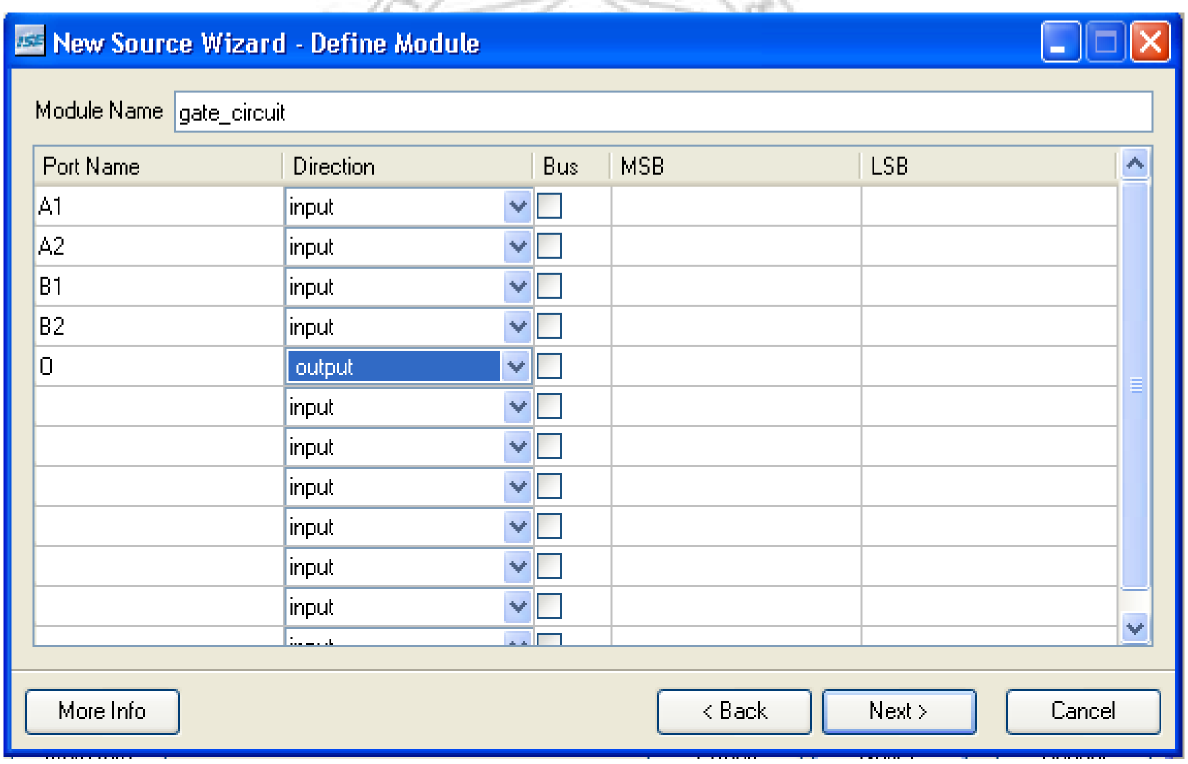


File name: gate_circuit

Location: C:\Xilinx91i\gate_circuit

✓ Add to project

Figure 3.4: *Dialog box for selecting source type*

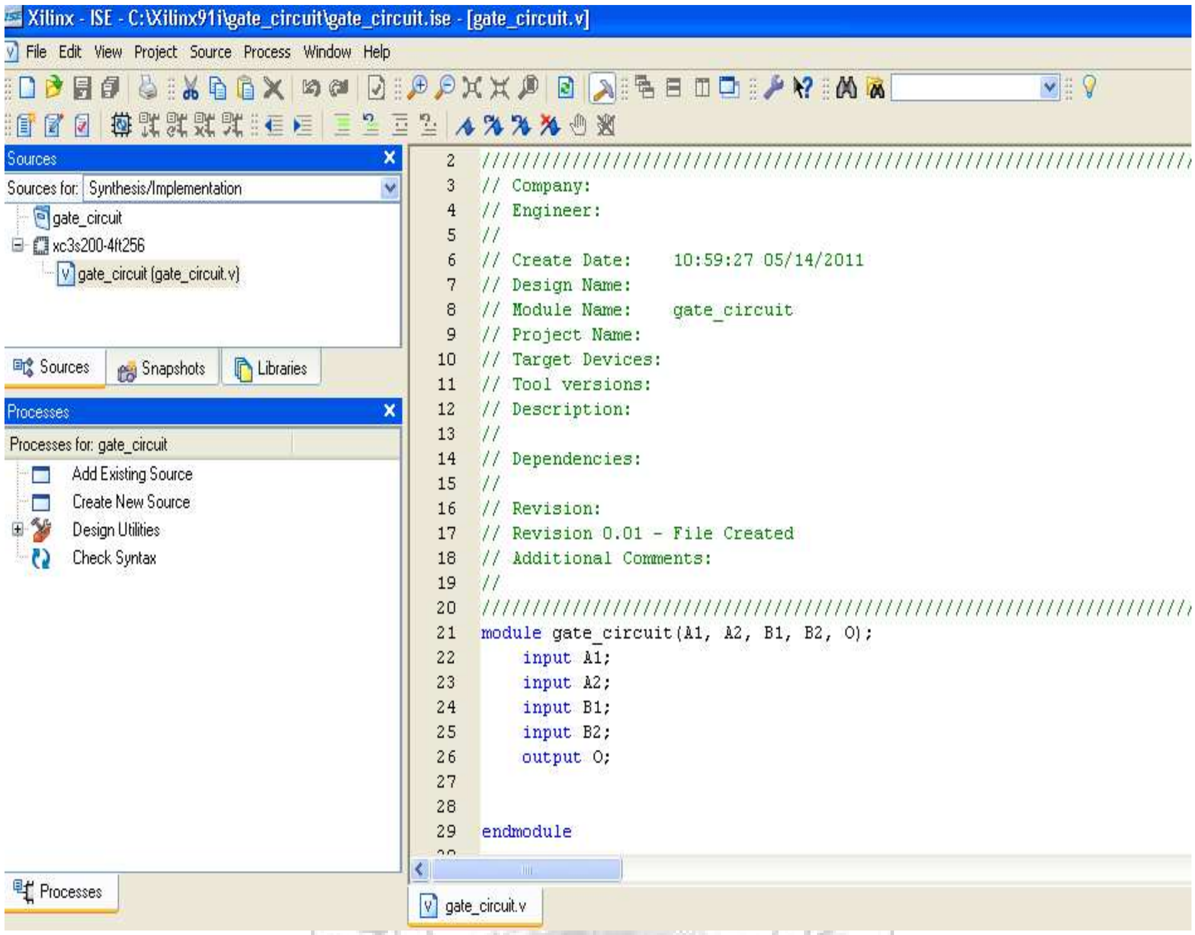


Module Name: gate_circuit

Port Name	Direction	Bus	MSB	LSB
A1	input	☐		
A2	input	☐		
B1	input	☐		
B2	input	☐		
O	output	☐		
	input	☐		
	input	☐		
	input	☐		
	input	☐		
	input	☐		
	input	☐		

Figure 3.5: *dialog box for new source wizard*

The sources file containing the <file_name> module displays in the Workspace, and the <file_name> displays in the Sources tab, as shown below:


Xilinx - ISE - C:\Xilinx91i\gate_circuit\gate_circuit.ise - [gate_circuit.v]
File Edit View Project Source Process Window Help
Sources
Sources for: Synthesis/Implementation
gate_circuit
xc3s200-4ft256
gate_circuit (gate_circuit.v)
Sources
Snapshots
Libraries
Processes
Processes for: gate_circuit
Add Existing Source
Create New Source
Design Utilities
Check Syntax
Processes
gate_circuit.v

```
////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    10:59:27 05/14/2011
// Design Name:
// Module Name:    gate_circuit
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////
module gate_circuit(A1, A2, B1, B2, O);
    input A1;
    input A2;
    input B1;
    input B2;
    output O;


endmodule
```

Figure 3.6: *Coding window*

Enter the following code in workspace.

```
module gate_circuit(A1, A2, B1, B2, O);
    input A1;              /* A1, A2, B1, B2 form the input port
    input A2;              list of the module*/
    input B1;
    input B2;
    output O;    // O is the single output port of the module

wire O1,O2;   /*O1  and  O2  are  intermediate  signals  within  the
              module */
and G1(O1,A1,A2);    // AND gate primitive has two instantiation with
and G2(O2,B1,B2);    // assigned names G1, G2
nor G3(O,O1,O2);
                     // NOR gate primitive has one instantiation with
endmodule
                     // assigned name G3
```

CHECKING THE SYNTAX OF THE NEW MODULE

When the source files are complete the next step is to check the syntax of the design to find errors.

1. Verify that **Synthesis/Implementation** is selected from the drop-down list in the Sources window.
2. Select the **<file_name> design** source in the Sources window to display the related processes in the Processes window.

3. Click the "+" next to the Synthesize-XST process to expand the process group.
4. Double-click the **Check Syntax** process.

Note: You must correct any errors found in your source files. You can check for errors in the Console tab of the Transcript window. If you continue without valid syntax, you will not be able to simulate or synthesize your design.

5. Double-click the **View RTL Schematic** process. The RTL schematic of the circuit looks like this.

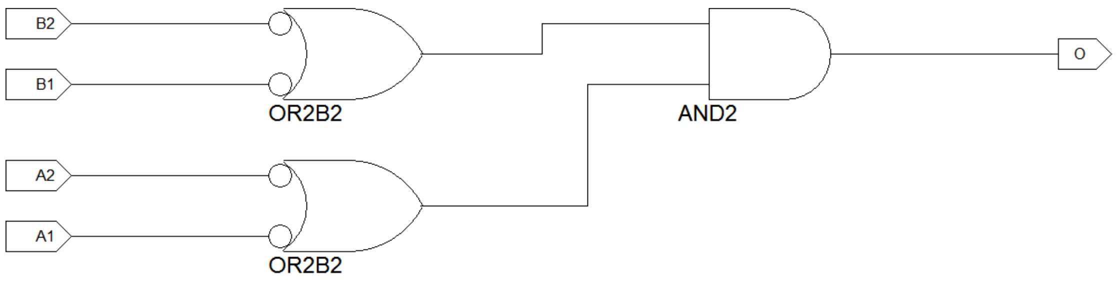


Figure 3.7: *RTL Schematic*

DESIGN SIMULATION

Verifying functionality using behavioral simulation

Create a test bench containing input stimulus you can use to verify the functionality of the module.

1. Create the test bench as follows:
2. Select the **<file_name>** HDL file in the Sources window.
3. Create a new test bench source by selecting **Project → New Source** OR by selecting **Create New Source** from the processes window.
4. In the New Source Wizard, select **Verilog Test Fixture** as the source type, and type **<testbench_name>** in the File Name field. Click **Next**.
5. The Associated Source page shows that you are associating the test bench with the source file (already created) .Click **Next**.
6. The Summary page shows that the source will be added to the project, and it displays the source directory, type and name. Click **Finish**.

Test Bench for the created module is as follows:

```
module gate_citcuit_test_v;
     // Inputs    //  specific  values  will  be  assigned
     reg A1;      //  to  A1,  A2,  B1,  and  B2  hence  these
     reg A2;      // are declared as reg
     reg B1;
     reg B2;
// Outputs
     wire O;

     // Instantiate the Unit Under Test (UUT)
     gate_circuit uut (
          .A1(A1),
          .A2(A2),
          .B1(B1),
          .B2(B2),
          .O(O)
     );
     initial begin
          // Initialize Inputs
          A1 = 0;
          A2 = 0;
          B1 = 0;
          B2 = 0;
          // Add stimulus here
          #3 A1 = 1;
          #3 A2 = 1;
          #3 B1 = 1;
          #3 B2 = 0;
          #3 A1 = 1;
          #3 A2 = 0;
          #3 B1 = 0;

     end
     initial $monitor ($time, "O = %b, A1 = %b, A2 = %b,  B1 =
%b, B2=%b", O, A1 ,A2 , B1 , B2);
endmodule
```

7. Save the test bench file and check simulation.
8. Verify that **Behavioral Simulation** and **<testbench_name> are** selected in the Sources window.
9. In the **Processes** tab, click the "**+**" to expand the Xilinx ISE Simulator process and double-click the **Simulate Behavioral Model** process.
10. The ISE Simulator opens and runs the simulation to the end of the test bench. To view your simulation results, select the **Simulation** tab and zoom in on the transitions.
11. The result produced is as follows:

```
Finished circuit initialization process.
                     0 O = 1, A1 = 0, A2 = 0, B1 = 0, B2=0
                     3 O = 1, A1 = 1, A2 = 0, B1 = 0, B2=0
                     6 O = 0, A1 = 1, A2 = 1, B1 = 0, B2=0
                     9 O = 0, A1 = 1, A2 = 1, B1 = 1, B2=0
                    18 O = 1, A1 = 1, A2 = 0, B1 = 1, B2=0
                    21 O = 1, A1 = 1, A2 = 0, B1 = 0, B2=0
```

EXERCISES

1. Write a code for the ALU module which performs four basic functions of addition, subtraction, multiplication and division. The ALU must be realized using `case` construct.

____
____
____
____
____
____
____

2. Write the test bench for the ALU module. Attach the printout of the simulation and RTL schematic.

Lab Session 04

OBJECT

Simplifying, implementing and testing the given logic expression in hardware and realizing its NAND equivalent.

COMPONENTS REQUIRED

1. Digital logic trainer board or the following components:
   Bread board, 5 V - Power Supply, Multimeter, LEDs with Resistors, Switches
2. Logic probe
3. Connecting wires
4. Following ICs and their datasheets or pin configurations:
   - 7400 quad 2 input NAND gate
   - 7404 hex inverter
   - 7408 quad 2 input AND gate
   - 7432 quad 2 input OR gate

GIVEN LOGIC EXPRESSION

F (A, B, C, D) = $\sum$ (0, 2, 3, 7, 8, 10, 11, 14, 15)

PROCEDURE

1. Use Karnaugh map to reduce the given function.
2. Draw the circuit diagram for the obtained reduced function.
3. Implement the reduced circuit using digital ICs on a bread board (refer to appendix A for IC pin configurations) and record the observations.
4. Find NAND realization for the simplified circuit.
5. Implement the all NAND circuit using digital ICs on a bread board (refer to appendix A for the pin diagram) and verify the observations taken in step 3.

REDUCTION OF LOGIC EXPRESSION

Reduced logic expression comes out to be: ______________________________________

LOGIC DIAGRAM (REDUCED FORM)

NAND REALIZATION OF THE REDUCED LOGIC EXPRESSION

OBSERVATIONS

A	*B*	*C*	*D*	*Expected Output*	*Observed Output*	
					For simplified expression	*For NAND realization*
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
1	0	0	0			
1	0	0	1			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1			

Lab Session 05

OBJECT

Designing Half and Full Adder Circuits in hardware and simulating the same using Verilod HDL.

COMPONENTS AND APPARATUS REQUIRED

1. Digital logic trainer board or the following components:
   Bread board, 5 V - Power Supply, Multimeter, LEDs with Resistors, Switches
2. Logic probe
3. Connecting wires
4. Following ICs and their datasheets or pin configurations:
   - 7408 Quad 2-input AND Gate
   - 7432 Quad 2-input OR Gate
   - 7486 Quad 2-input XOR Gate

THEORY

Half Adder

A combination circuit that performs the addition of two bits without accounting for the previous carry is called half adder. It needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits. The output variables produce the sum and carry. The simplified sum of product expressions for a half adder are:

$$S = \bar{x}.y + x.\bar{y} = x \oplus y$$
$$C = x.y$$

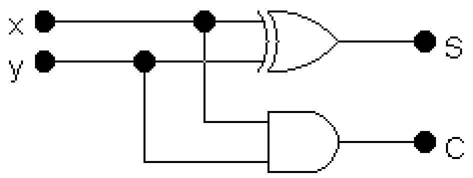


Figure 5.1: *Logic circuit for Half Adder*

Full Adder

A combinational circuit that performs the addition of three input bits. It consist of three inputs and two outputs. Two of the input variables, represent the two significant bits to be added. The third input, represents the carry from the previous lower significant position. The output variables produce the sum and carry. The simplified sum of product expressions for a half adder are:

$$S = \bar{x}.\bar{y}.z + \bar{x}.y.\bar{z} + x.\bar{y}.\bar{z} + x.y.z = x \oplus y \oplus z$$
$$C = y.x + x.z + y.z = (x \oplus y).z + x.y$$

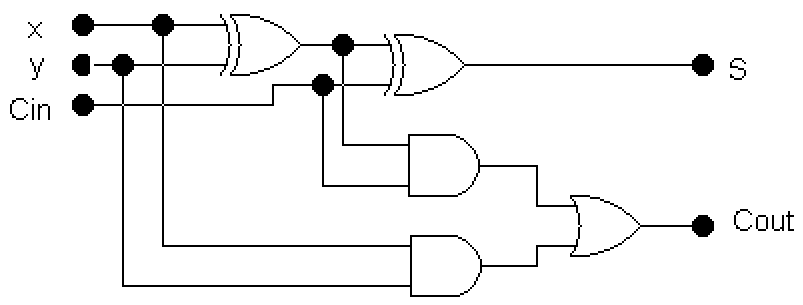


Figure 5.2: *Circuit diagram for Full Adder*

HARDWARE IMPLEMENTATION AND OBSERVATIONS

Implement the half adder and full adder circuits in hardware (refer to appendix A for IC pin configurations) and record the observations in the following tables:

Half Adder

Inputs		*Outputs*	
x	*Y*	*Carry*	*Sum*
0	0		
0	1		
1	0		
1	1		

Full Adder

Inputs			*Outputs*	
X	*y*	*z*	*Carry*	*Sum*
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

SIMULATION USING VERILOG HDL.

Design Module & Test Bench for Half Adder

```
// module definition
module half_adder(a, b, sum, ca);
    input a;
    input b;
    output sum;
    output ca;
     xor(sum,a,b);
     and (ca,a,b);
endmodule

// test bench
module half_adder_test_v;
     // Inputs
     reg a;
     reg b;

     // Outputs
```

```
	wire sum;
	wire ca;

	// Instantiate the Unit Under Test (UUT)
	half_adder uut (
		.a(a),
		.b(b),
		.sum(sum),
		.ca(ca)
	);

	initial begin
		// Initialize Inputs
		a = 0;
		b = 0;
	end
always
begin
#2 a = 1; b = 0;
#2 a = 0; b = 1;
#2 a = 1; b = 1;
#2 a = 0; b = 0;
end
initial $monitor($time , " a = %b ,b = %b , outcarry = %b ,
outsum = %b " , a,b,ca,sum);
initial #24 $stop;
endmodule
```

Results

1. Write the results as produced by the simulation.

2. Attach the printout of RTL schematic and simulation waveform.

EXERCISES

1. Write a code for full adder module by instantiating half adder as created previously.

2. Write a test bench for the following values of inputs with a time delay of 2. The simulation must stop at 30 ns.

A	B	Cin
1	1	0
1	0	1
1	1	1
1	0	0
0	0	0
0	1	0
0	0	1
0	1	1
1	0	0
0	1	0
1	1	1

3. Design a full subtractor circuit and simulate it using any simulation tool. Compare the results of simulation with that of your designed circuit. Attach hardcopy of the simulated circuit/output here.

Lab Session 06

OBJECT

Designing a 4-bit Parallel Adder on Electronics Workbench

THEORY

4-BIT PARALLEL ADDER

A 4-bit parallel adder can add two 4-bit values. One way of constructing such an adder is to simply cascade four full adders as shown in figure 6.1. Here $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are two 4-bit values to be added. FA3, FA2, FA1 and FA0 are the four full adders that are cascaded in parallel to provide the desired result. $S_3S_2S_1S_0$ is the final 4-bit sum. C_0 is the external carry input (if any) to the circuit and C_4 is the final carry generated by the 4-bit addition.

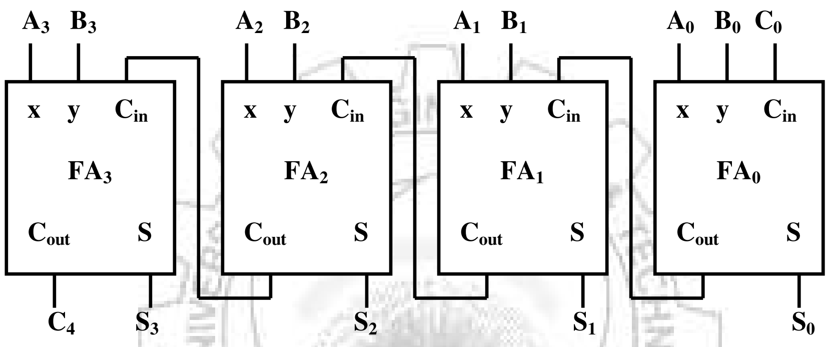


Figure 6.1: *4-bit Parallel Adder*

CREATING SUBCIRCUITS IN ELECTRONIC WORKBENCH

Creating a subcircuit allows you to reuse the circuit multiple times in a design and in future designs. Subcircuits may contain basic circuit elements or other subcircuit definitions.

Creating a subcircuit

1. First you have to design the circuit that you want to implement. Let's say you want to create a half-adder, so you simply draw a circuit with a XOR and AND gate (XOR is SUM and AND is Carry Out).
2. Select all components of the design that needs to be included in the subcircuit.
3. From the menu *Circuit*, select *Create Subsircuit.* Type a name and choose appropriate options. This will open a subwindow on the main design area containing the initially selected components.
4. Drag the input and output lines towards the edges to make input and output ports/terminals in the subcircuit.
5. Now click anywhere in the outer design area, this will close the internal structure of the subcircuit. The subcircuit will appear as a block on the main design area with assigned input and output lines. Now this block can be replicated (copy-pasted) where ever required or can be saved for future reuse.

PROCEDURE

1. Create a subcircuit for a full adder unit. Refer to lab session 5.a for the gate level circuit of a full adder. This subcircuit should have three inputs namely x, y, C_{in} and two outputs namely C_{out} and S.
2. Replicate the subcircuit to make a total of four full adder subcircuits namely FA3, FA2, FA1 and FA0.
3. Make connections among these full adders as shown in figure 6.1.
4. Again create a subcircuit containing all these four full adders to make a single unit of a 4-bit parallel adder. This new subcircuit should have nine inputs (C_0, A3, A_2, A_1, A_0, B_3, B_2, B_1, B_0) and five outputs (C_4, S_3, S_2, S_1, S_0).
5. Connect the inputs to switches and outputs to indicators. Apply various combinations of 1s and 0s at inputs and check the binary values at the outputs.

Lab Session 07

OBJECT

Experimenting with decoder and multiplexer ICs.

COMPONENTS AND APPARATUS REQUIRED

1. Digital logic trainer board or the following components:
   Bread board, 5 V - Power Supply, Multimeter, LEDs with Resistors, Switches
2. Logic probe
3. Connecting wires
4. Following ICs and their datasheets or pin configurations:
   - 74138 - 3 x 8 Line Decoder / Demultiplexer
   - 74150 - 16 x 1 Mutiplexer

THEORY

Decoder

A Decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. In practical applications, decoders are often used for selecting one of several devices.

Demultiplexer

A decoder with an enable input can function as a Demultiplexer (DMUX). A Demultiplexer (DMUX) is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines. The selection of a specific output line is controlled by the bit values of n selection lines.

74138 - 3 x 8 Line Decoder / Demultiplexer

The 74138 IC has three inputs and eight output lines. It has three enable inputs and for the IC to function all three inputs need to be enabled. Refer to appendix A for IC pin configuration. Function of various pins of this IC is described below:

- ***Y0 through Y7:*** Active low data outputs
- ***A, B, C:*** Active high input / select lines with C being the MSB
- ***G1:*** Active high enable Input
- ***G2A' and G2B':*** Active low enable Inputs
- ***VCC and GND:*** Supply connections lines

Multiplexers

A digital data Multiplexer (MUX) is a combinational circuit having several data inputs and a single output. A set of *data-select* inputs is used to control which of the data inputs is routed to the single output. A multiplexer is also called a *data selector* because of this ability to select which data input is connected o the output. Normally there are 2^n input lines and n selection lines whose bit combination determine which input is selected.

74150 - 16 x 1 Multiplexer

The 74150 IC has sixteen data inputs and four data-selection lines. It also contains an active low enable and an active low output line. Refer to appendix A for IC pin configuration. Function of various pins of this IC is described below:

- ***E0 through E15:*** Active high data input lines
- ***A, B, C, D:*** Active high data select lines with D being the MSB
- ***W:*** Active low output line
- ***G':*** Active low enable line
- ***VCC and GND:*** Supply connections lines

TESTING PROCEDURE AND OBSERVATIONS

74138 - 3 x 8 Line Decoder / Demultiplexer

- Make Vcc and Gnd supply connections.
- Connect the three inputs A, B and C to switches.
- Set appropriate values at enable inputs to activate the IC.
- Apply different combinations of 1s and 0s at data inputs.
- Observe the output and record your observations in the following table.

C	*B*	*A*	*Y0*	*Y1*	*Y2*	*Y3*	*Y4*	*Y5*	*Y6*	*Y7*
0	0	0								
0	0	1								
0	1	0								
0	1	1								
1	0	0								
1	0	1								
1	1	0								
1	1	1								

74150 - 16 x 1 Multiplexer

- Make Vcc and Gnd supply connections.
- Connect the data select inputs A, B, C and D to switches.
- Set appropriate values at enable inputs to activate the IC.
- Connect the data inputs E0 through E15 to switches. For simplicity just two or three data inputs can be connected to switches at a time. Remaining inputs will draw 1 by default (float high characteristic of TTL ICs).
- Now select any data input with the help of data selectors A, B, C, and D. Apply different data (1 or 0) at this selected data inputs.

- Observe the output. The invert of the value applied at the selected data input will appear at the output.
- Record your observations for the input configurations given in the following table. Here only E0, E4 and E15 data inputs are considered.

G'	*D*	*C*	*B*	*A*	*Value at data input*	*W*	*G'*	*D*	*C*	*B*	*A*	*Value at data input*	*W*
0	0	0	0	0			0	1	0	0	1		
0	0	0	0	1			0	1	0	1	0		
0	0	0	1	0			0	1	0	1	1		
0	0	0	1	1			0	1	1	0	0		
0	0	1	0	0			0	1	1	0	1		
0	0	1	0	1			0	1	1	1	0		
0	0	1	1	0			0	1	1	1	1		
0	0	1	1	1			1	0	1	0	1		
0	1	0	0	0			1	1	0	1	0		

EXERCISES

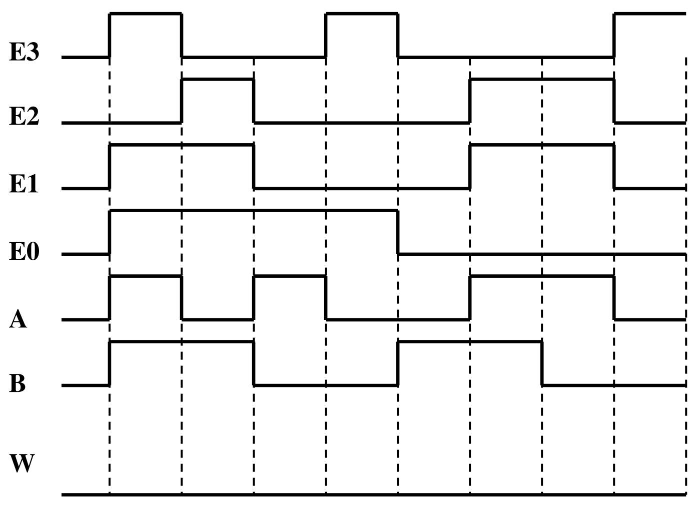
1. What will be the binary values at the outputs, Y0 through Y7, of 74138 if:
   - All three enable pins are connected to ground?

   ____
   ____
   ____
   ____

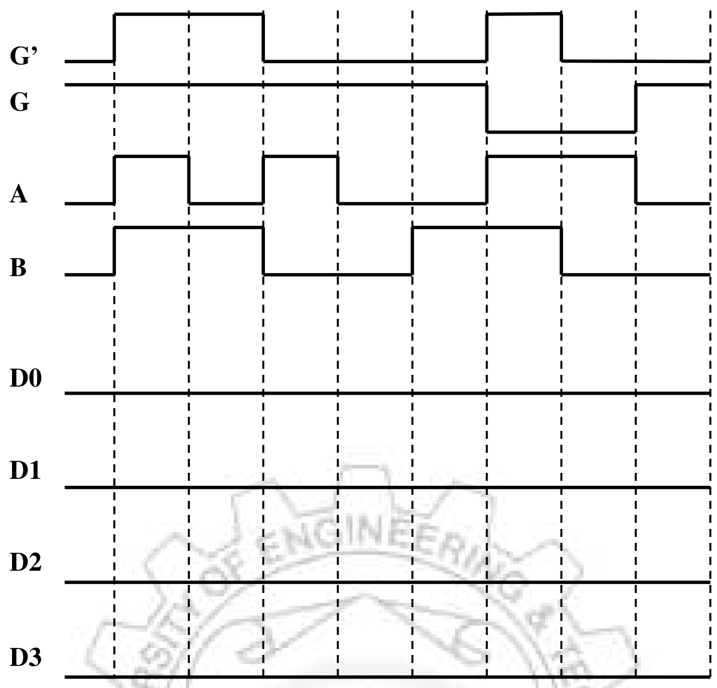
   - All three enable pins are connected to VCC?

   ____
   ____
   ____
   ____

2. Consider 4x1 Multiplexer. Draw the output wave-form for the following data inputs (E_0, E_1, E_2, E_3,) and select lines A, B (B being the MSB). Output is W.
3.



4. Consider a 2 x 4 Decoder with two enable inputs (one active high - G and one active low – G'). Draw the output wave-forms for D0, D1, D2 and D3, if the two select inputs are A and B (B being the MSB). All outputs are active low.



Lab Session 08

OBJECT

Simulating decoder and multiplexer Circuits using Verilog HDL

DESIGN MODULE FOR DECODER

```
// module definition
module  two_to_4decoder(in0,  in1,  enable,  out1,  out2,  out3,
out4);
    input in0;
    input in1;
    input enable;
    output out1;
    output out2;
    output out3;
    output out4;
      reg [3:0] out;

      always @ (enable or in0 or in1)
      begin
          if(~enable)
               out <= 4'b1111;
      else
          begin
          case({in1,in0})
               2'b00: out <= 4'b1110;
               2'b00: out <= 4'b1101;
               2'b00: out <= 4'b1011;
               2'b00: out <= 4'b0111;
          endcase
          end
      end

      assign {out4,out3,out2,out1} = out;
endmodule
```

1. Write the test bench for the decoder module for the following inputs.

In0	In1	Enable
0	0	0
0	0	1
0	1	0
0	1	1
1	0	1
1	1	1

2. Show the results as produced by the simulation. Also attach RTL schematic.

DESIGN MODULE & TEST BENCH FOR MULTIPLEXER

```
// module definition
module four_to_oneMUX(i0, i1, i2, i3, select, Y);
    input i0;
    input i1;
    input i2;
    input i3;
    input [1:0] select;
    output Y;
      reg Y;
      always @ (i0 or i1 or i2 or i3 or select)
      case (select)
      2'b00: Y = i0;
      2'b01: Y = i1;
      2'b10: Y = i2;
      2'b11: Y = i3;
      endcase
endmodule
```

1. Write the test bench for the following values of inputs. The time delay should be of 2 ns.

I0	I1	I2	I3	Select
0	0	0	0	00
0	0	0	1	00
0	0	1	0	01
0	1	0	0	10
1	0	0	0	11

2. Write the results as produced by the simulation.

Time delay	Inputs	Y

Lab Session 09

OBJECT

Experimenting with encoder and seven segment display driver ICs.

COMPONENTS AND APPARATUS REQUIRED

1. Digital logic trainer board or the following components:
   Bread board, 5 V - Power Supply, Multimeter, LEDs with Resistors, Switches
2. Logic probe
3. Connecting wires
4. Following ICs and their datasheets or pin configurations:
   - 74148 Octal to Binary Priority Encoder
   - 7447 BCD to Seven Segment Driver
   - 7404 hex inverter
   - Seven Segment Displays (Common Anode Type)

THEORY

Encoder

An Encoder is a digital function that produces a reverse operation from that of a decoder. An Encoder has 2^n (or less) input lines and n output lines. The output lines generate the binary code for the 2^n input variables.

Priority Encoder

A simple encoder may produce an erroneous output if more than one of its inputs is high. A Priority Encoder is one that responds to just one input among those that may be simultaneously high, in accordance with some priority system. The most common priority system is based on the relative magnitudes of the inputs: whichever decimal input is largest is the one that is encoded.

74148 8 x 3 Octal to Binary Priority Encoder

The 74148 is a priority encoder with active-low inputs for decimal digits. There are nine inputs lines (including an enable input) and five output lines, of which three represents the binary code for the octal digit. Refer to appendix A for IC pin configuration. Function of various pins of this IC is described below:

- ***0 through 7:*** Active low data inputs representing the octal digits
- ***A2, A1, A0:*** Active low output lines representing the binary code
- ***E1:*** Active low enable Input
- ***E0:*** Active low output indicating none of the inputs is high
- ***GS:*** Active low output indicating any of the inputs is high
- ***VCC and GND:*** Supply connections lines

7447 BCD to Seven Segment Driver

7447 IC is particularly used to drive common-anode Seven Segment displays. Its input is a BCD number and output drives a seven segment display. Refer to appendix A for IC pin configuration. Function of various pins of these ICs is described below:

- ***A, B, C, D:*** Active high inputs representing BCD digits (D being the MSB).
- ***OA through OG:*** Active low outputs to drive segments *a* through *g* of the display.
- ***RBI:*** Ripple Blanking Input. Turns off all the segments if kept low, provided that LT is kept high and all other inputs (A, B, C, D, BI) are kept low. Should be kept high otherwise.
- ***BI / RBO:*** Wire-AND logic serving as a Blanking Input and / or Ripple Blanking Output.
    - *BI:* Turns off all the segments if low.
    - *RBO:* Goes to a low level (response condition) along with other outputs, when RBI and inputs A, B, C, and D are low with LT input at high level.
- ***LT:*** Lamp Test input. Tests whether all segments are working or not. Illuminates all segments if kept low, provided that BI is kept high. Should be kept high otherwise.
- ***VCC and GND:*** Supply connections lines

Seven Segment Displays

A Seven Segment Display consists of seven light-emitting segments. The segments are designated by letters *a* through *g* (see figure 9.1). By illuminating various combinations of segments, the numerals 0 through 9 can be displayed. Seven Segment Displays are commonly constructed with light-emitting diodes (LEDs) and with liquid-crystal displays (LCDs). LEDs generally provide greater illumination levels but require much greater power than LCDs.

An LED display can be a common-anode type or common cathode type. In common anode type, a high voltage is applied at the *common* terminal of the display and low voltage is applied at a segment's terminal for illumination. In the common-cathode type, a low voltage is applied at the *common* terminal of the display and high voltage is applied at a segment's terminal for illumination.

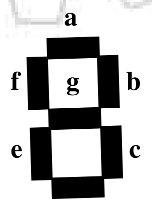


Figure 9.1: *Seven Segment Display*

IMPLEMENTATION AND OBSERVATIONS

You are required to display the outputs of a 74148 encoder IC on a seven segment display. The circuit is given in figure 9.2.

- Make connections as shown in figure 9.2. (refer to appendix A for IC pin configurations).
- Select any input from 74148 IC and observe the corresponding decimal code being displayed on the seven segment.

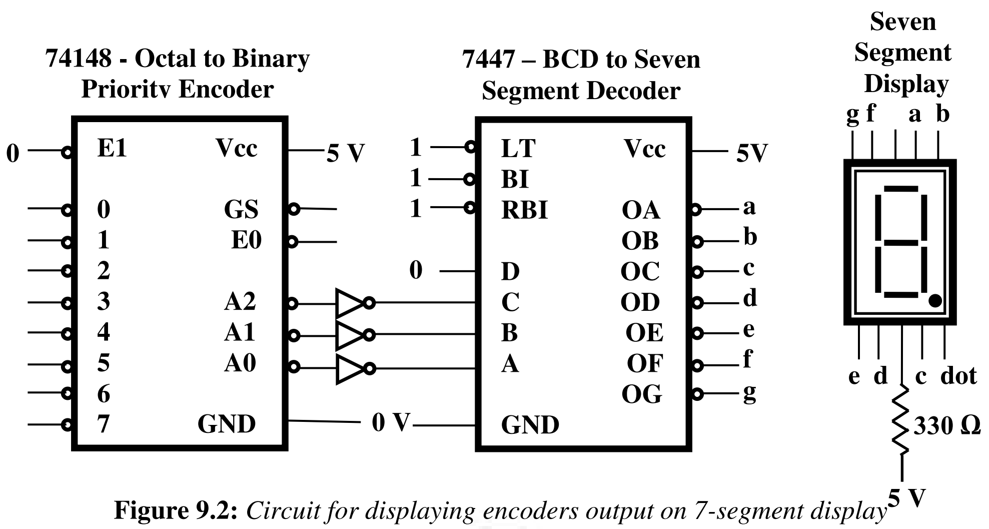


Figure 9.2: *Circuit for displaying encoders output on 7-segment display*

EXERCISES

1. Perform the Lamp Test for the designed circuit and write your observations.

2. How can you use 7447 IC to drive a common-cathode display?

Lab Session 10

OBJECT

Testing different modes of JK-FF. Also, designing a modulo-4 asynchronous up-counter using JK-FF

COMPONENTS AND APPARATUS REQUIRED

1. Digital logic trainer board or the following components:
   Bread board, 5 V - Power Supply, Multimeter, LEDs with Resistors, Switches
2. Logic probe
3. Connecting wires
4. Following ICs and their datasheets or pin configurations:
   - 7473 / 7476 JK Flip-Flop

THEORY

Flip-Flop

A flip-flop circuit can maintain a binary state indefinitely (as long as the power is delivered to the circuit) until directed by an input signal to switch states. The major differences among various types of flip-flops are in the number of inputs they possess and in the manner in which the inputs affect the binary state.

JK Flip-Flop

JK flip flop is an edge triggered device. A typical JK flip flop has three inputs: J, K and a clock input. The flip-flop can be either positive or negative edge triggered. The output Q is available in complemented form as well.

Besides the usual inputs and output, most of the flip-flop IC also possess two asynchronous inputs, namely *Preset* and *Clear*. These inputs are usually active low. If used, *Preset* and *Clear* inputs keep the flip-flop in set and reset state respectively, irrespective of the other inputs. Both of these inputs cannot be used simultaneously, otherwise they will bring the flip-flop in unstable state.

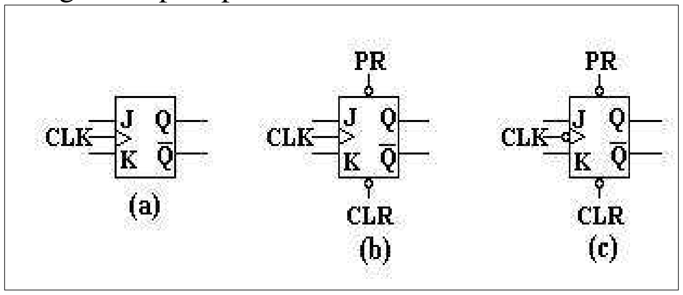


Figure 10.1: *Symbol for JK flip-flop*
a. positive-edge triggering
b. active low Preset (PR) and Clear (CLR) with positive-edge triggering
c. active low Preset (PR) and Clear (CLR) with negative-edge triggering

7473 / 7476 Dual JK Flip Flop

Both the ICs 7473 and 7476 are similar in functionality except for one difference. The flip-flops in 7473 have only one type of active low asynchronous input, which is the *Clear* input, whereas the flip-flops in 7476 have both *Preset* and *Clear* inputs. Both these ICs have negative edge triggered flip-flops. Refer to appendix A for IC pin configuration Function of various pins of this IC is described below:

- ***1CLK, 2CLK:*** Negative edge triggered clock inputs for FF1 and FF2 respectively.
- ***1PRE, 2PRE(for 7476 only):*** Active low preset inputs for FF1 and FF2 respectively.
- ***1CLR, 2CLR:*** Active low clear inputs for FF1 and FF2 respectively.
- ***1J, 2J:*** Active high J inputs for FF1 and FF2 respectively.
- ***1K, 2K:*** Active high K inputs for FF1 and FF2 respectively.
- ***1Q, 2Q:*** Active high outputs for FF1 and FF2 respectively.
- ***1Q`, 2Q`:*** Active low outputs for FF1 and FF2 respectively.
- ***VCC and GND:*** Supply connections lines

Digital Counters

A digital counter is a set of flip-flops whose states change in response to pulses applied at the input to the counter. Every counter resets after a certain number of clock pulses. Thus, as it name implies, a counter is used to count pulses. An *n* stage counter can count up to a maximum of 2^n states. *n* is equal to the number of flip-flops required for the construction of counter.

Modulus Counters

The number of input pulses that causes a counter to reset to its initial count is called the *modulus* of the counter. Thus, the modulus equals to total number of distinct *states* (counts), including zero that a counter can store. A binary counter with *n* stages is a *modulo-*2^n (or *MOD-*2^n) counter. The largest count a *mod-N* counter can achieve is *N-1*, i.e. a *mod-N* counter never reaches the binary number equal to its modulus. N is always equal to or less than 2^n.

Counters can be classified as:

- ***Synchronous Counters,*** which are clock driven. All the flip-flops are driven by a single clock.
- ***Asynchronous Counters,*** which are event driven. Clock input is given to the first flip-flop only. Rest of the flip-flops are driven by their preceding flip-flops.

MOD-4 ASYNCHRONOUS UP COUNTER

The number of flip-flops required to construct a mod-4 counter is 4. This counter will count from 0 to 3, a total of 4 distinct states.

TESTING PROCEDURE AND OBSERVATIONS

7473 / 7476 Dual JK Flip Flop

- Make Vcc and Gnd supply connections.
- Connect the CLK input to some clock source or a switch.
- Apply different combinations of 1s and 0s at inputs J, K, Preset and Clear.
- Observe the output and record your observations in the following table.

CLK	*PRE**	*CLR*	*J*	*K*	*Q*
↓	1	1	0	0	
↓	1	1	0	1	
↓	1	1	1	0	
↓	1	1	1	1	
↓	1	0	1	1	
↓	0	1	1	1	
1	1	1	1	1	

*ignore if 7473 IC is being used

MOD-4 Asynchronous Counter

- Make connections as shown in figure 10.2. (refer to appendix A for IC pin configurations).
- Observe the binary values at the outputs of the two flip flops with the incoming clock pulses.

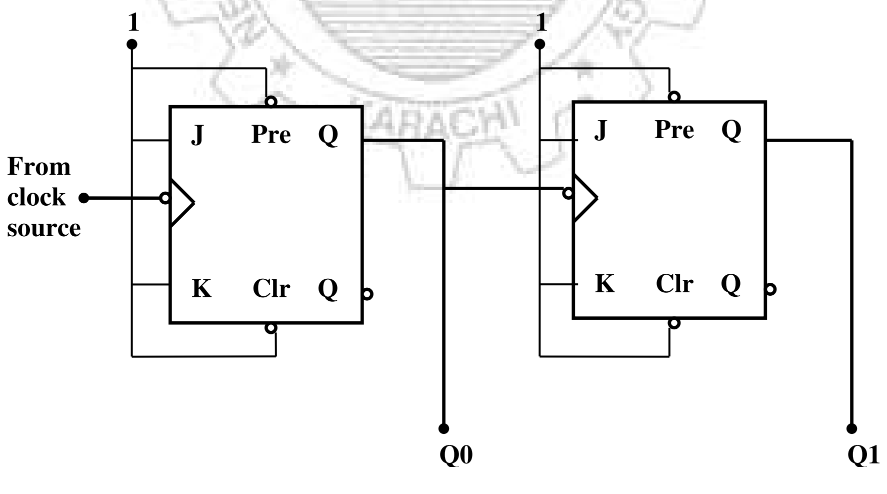


Figure 10.2: *MOD-4 Asynchronous Counter*

EXERCISES

1. Draw the timing diagram for MOD-4 counter designed in this laboratory session.

2. Simulate MOD-5 asynchronous counter using any simulation tool. The number of flip-flops required to construct a MOD-5 counter is 3. This counter will count from 0 to 4, a total of 5 distinct states. Since a 3-stage counter can count up to 8 states at maximum, a NAND gate is used to reset it after 5 clock pulses.
   Attach hardcopy of the simulated circuit/output here.

3. Draw the timing diagram for MOD-5 counter simulated in activity 2.

Lab Session 11

OBJECT

Experimenting with decade counter.

COMPONENTS AND APPARATUS REQUIRED

1. Digital logic trainer board or the following components:
   Bread board, 5 V - Power Supply, Multimeter, LEDs with Resistors, Switches
2. Logic probe
3. Connecting wires
4. Following ICs and their datasheets or pin configurations:
   - 7490 decade counter

THEORY

7490 Decade Counter

7490 decade counter can generate the following two sequences:
1. BCD sequence: In this case output QA is connected to input B, external clock is applied to input A
2. Bi-quinary sequence: In this case output QD is connected to input A, external clock is applied to input B

Refer to appendix A for IC pin configuration. Function of various pins of this IC is described below:
- ***Input A, Input B:*** Positive edge triggered clock inputs.
- ***QA, QB, QC and QD:*** Active high outputs.
- ***R91, R92, R01, R02:*** Active high reset inputs. These resets are activated as follows:

Reset Inputs				*Output*			
R01	***R02***	***R91***	***R92***	***QD***	***QC***	***QB***	***QA***
H	H	L	X	L	L	L	L
H	H	X	L	L	L	L	L
X	X	H	H	H	L	L	H
X	L	X	L	COUNT			
L	X	L	X	COUNT			
L	X	X	L	COUNT			
X	L	L	X	COUNT			

- ***VCC and GND:*** Supply connections lines.

TESTING PROCEDURE AND OBSERVATIONS

- Make Vcc and Gnd supply connections.
- Disable all RESET inputs.
- Connect input A and input B according to the connections indicated in the previous section to generate BCD count and then bi-quinary count.

- Observe the output in each case and record your observations given in the following table.

BCD Count				
Clock Pulse	***QD***	***QC***	***QB***	***QA***
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Bi-quinary Count				
Clock Pulse	***QD***	***QC***	***QB***	***QA***
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

EXERCISES

1. Enable the RESET inputs RO1 and RO2 (keeping R91 and R92 disabled) and write your observations below:

2. Enable the RESET inputs R91 and R92 (keeping R01 and R02 disabled) and write your observations below:

3. Enable all the RESET inputs R01, R02, R91 and R92 and write your observations below:

Lab Session 12

OBJECT

Designing seconds section of a digital clock on Electronics Workbench

THEORY

Digital Clock

A digital clock is a time keeping circuit that displays seconds, minutes and hours. For seconds' section of a digital clock, a MOD-60 counter needs to be designed. A simple way to do this is to design MOD-10 and MOD-6 counters separately and then cascade them.

Generating Clock (Square Wave) Signals for Sequential Circuits in Electronic Workbench

Most sequential circuits require trigger/pulse for the functioning of their memory elements. This trigger is merely a 0 to1 or 1 to 0 signal transition at the input where it needs to be applied. This can be achieved by simply connecting a switch which can toggle between 0 and 1.

For continuous pulse generation a proper clock source is needed. In EWB, this can be achieved via function generator (in *Instruments Parts Bin*) or more easily as follows:

1. From the *Sources Parts Bin*, drag and drop *Clock* on the design area. Double click the *Clock* to reveal the properties dialog box, where frequency, duty cycle and voltage of the signal can be chosen.
2. Connect the negative end of the component to the ground.
3. Positive end can be connected to the point where the clock signal needs to be applied. You can connect an indicator here to view the generated signal.

PROCEDURE

In this lab session we will use two 7490 ICs to design MOD-10 and MOD-6 counters.

1. For MOD-10 counter, connect a 7490 IC in BCD count mode (refer to lab session 9.a). Disable all RESET inputs and apply clock at input A as illustrated is the previous section. Set the frequency of the clock to 1 Hz.
2. Connect another 7490 IC in BCD mode. To convert it to MOD-6 counter, connect R01 to QB and R02 to QC. This will activate the resets of 7490 when it reaches the count of 6. Disable the other two RESET inputs R91 and R92. Now connect QD of the MOD-10 counter created in the last step to the clock input (input A) of this IC. QD is the MSB of the MOD-10 count and it goes from 1 to 0 only when the counter resets after 9, thus providing the negative edge to increment the MOD-6 counter. Here also a seven segment display can be connected to view the outputs. Now collectively this circuit works as a MOD-60 counter.
3. The output of this circuit can be viewed on seven segment displays. Two kinds of 7 segment displays are available in EWB. One takes four BCD inputs directly and the

other one takes seven inputs for the seven segments. The later one is connected via a BCD to seven segment converter. For our application the former one is simpler to use as it provides the BCD to seven segment conversion internally.

EXERCISES

1. Design the hours' section of a digital clock on EWB. The clock should be a 24 hour clock. Attach hardcopy of the simulated design here.

Lab Session 13

OBJECT

Experimenting with bidirectional universal shift register

COMPONENTS AND APPARATUS REQUIRED

1. Digital logic trainer board or the following components:
   Bread board, 5 V - Power Supply, Multimeter, LEDs with Resistors, Switches
2. Logic probe
3. Connecting wires
4. Following ICs and their datasheets or pin configurations:
   - 74194 4-bit bidirectional universal shift register

THEORY

Shift Registers

A Register is a set of flip-flops used to store binary data. A register, which is capable of shifting its binary information either to the right or left, is called a shift register. The logical configuration of a register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop connected to the input of next flip-flop. All flip-flops receive a common clock pulse, which causes the shift from one stage to the next.

Universal Bidirectional Shift Registers

A bi-directional shift register is one whose bits can be shifted from left to right or from right to left. A universal shift register is a bi-directional register whose input can be in either serial or parallel form and whose output can be in either serial or parallel form.

74194 4-Bit Bidirectional Universal Shift Register

The 74194 register provides parallel as well as serial loading in both directions. A, B, C, and D are inputs for parallel loading, whereas SR and SL are inputs for serial loading with right or left shifting respectively. S1 and S0 are used to select the loading mode. Refer to appendix A for IC pin configuration. Function of various pins of this IC is described below:

- ***A, B, C and D:*** Active high inputs for parallel loading.
- ***QA, QB, QC and QD:*** Active high outputs.
- ***S0 and S1:*** Active high mode control inputs. The following table shows combinations of S1 and S0 to enable various modes.

S1	*S0*	*Clock*	*Action*
0	0	x	No change
0	1	↑	Shift right
1	0	↑	Shift left
1	1	↑	Parallel load

- ***SR:*** Active high serial input for right shifting.
- ***SL:*** Active high serial input for left shifting.
- ***CLR:*** Active low clear input.
- ***CLK:*** Positive edger triggered cock input.
- ***VCC and GND:*** Supply connections lines.

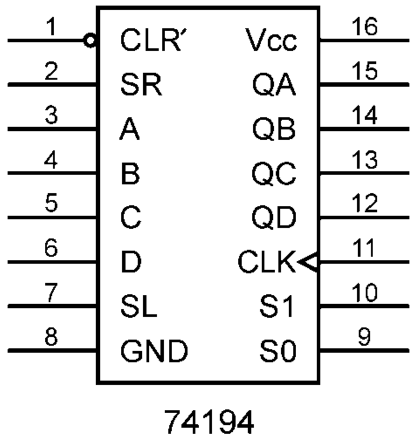
TESTING PROCEDURE AND OBSERVATIONS

- Make Vcc and Gnd supply connections.
- Connect the CLK input to some clock source or a switch.
- Disable CLR input.
- Apply different combinations of 1s and 0s at inputs J, K, Preset and Clear.
- Observe the output and record your observations given in the following table.

S1	*S0*	*SL*	*SR*	*A*	*B*	*C*	*D*	*QA*	*QB*	*QC*	*QD*
0	0	0	1	0	0	0	0				
0	1	1	0	0	0	0	1				
1	0	0	0	0	0	1	0				
1	1	0	0	0	0	1	1				
0	0	1	1	0	1	0	0				
0	1	1	1	0	1	0	1				
1	0	1	0	0	1	1	0				
1	1	1	0	0	1	1	1				
1	1	0	0	1	1	1	1				

EXERCISES

1. Show connections of 74194 to convert it into a ring counter with right shifting.



Lab Session 14

OBJECT

Designing sequential circuit for the given state diagram using D flip-flop.

GIVEN STATE DIAGRAM

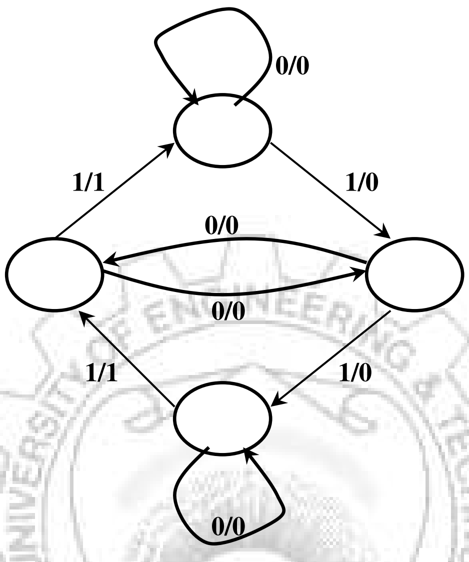


Figure 14.1: ***State Diagram***

COMPONENTS AND APPARATUS REQUIRED

1. Digital logic trainer board or the following components:
   Bread board, 5 V - Power Supply, Multimeter, LEDs with Resistors, Switches
2. Logic probe
3. Connecting wires
4. Following ICs and their datasheets or pin configurations:
   - 7474 D Flip-Flop
   - 7408 Quad 2-input AND Gate
   - 7486 Quad 2-input XOR Gate

THEORY

Sequential Circuits

In *Sequential Circuits,* the output not only depends on the present inputs, but also on previous states of the circuit. These circuits use memory elements (latches, flip-flops) and the binary information stored in the memory elements at any given time defines the state of the sequential circuit.

Analysis of a Sequential Circuit

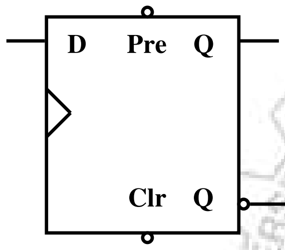
The analysis of a sequential circuit form a state diagram comprises of the following:

- Selecting a particular flip-flop to design the circuit
- Obtaining state table from the state diagram
- Finding input equations for the selected flip-flop
- Implementing the circuit using selected flip-flops and their input equations

ANALYZING THE GIVEN STATE DIAGRAM

D Flip-Flop

D flip-flop is also called transparent flip-flop as it simply transfers the input data to the output. For the design, the graphical symbol and excitation table of D flip-flop are required.



(a): *Graphical Symbol*

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

(b): *Excitation table of D Flip-Flop*

Figure 14.2: *D Flip Flop*

Characteristic equation of D flip-flop is: $Q_{(t+1)} = D$

State Table

The given state diagram has four states, so we will need two D flip-flops (FF-0, FF-1). Form state table form the given state diagram and record the results in the following table:

Present State		*Input*	*Next State*		*Output*	*Input to FF-0, D_0*	*Input to FF-1, D_1*
Q_{1t}	Q_{0t}	x	$Q_{1(t+1)}$	$Q_{0(t+1)}$	z		

Input Equations

Using Karnaugh map, the input equations, D_0 and D_1, for the two flip-flops can be found.

D_0

$Q_1 \backslash Q_0x$	$\overline{Q}_0\overline{x}$	$\overline{Q}_0x$	Q_0x	$\overline{Q}_0x$
$\overline{Q}_1$				
Q_1				

D_0 = ___________________

D_1

$Q_1 \backslash Q_0x$	$\overline{Q}_0\overline{x}$	$\overline{Q}_0x$	Q_0x	$\overline{Q}_0x$
$\overline{Q}_1$				
Q_1				

D_1 = ___________________

Circuit Diagram

Implementation Procedure and Observations

1. Implement the circuit using digital ICs on a bread board (refer to appendix A for IC pin configurations).
2. Make Vcc and Gnd supply connections.
3. Connect the CLK and the CLR inputs to input switches (preferably push buttons).
4. Apply 0 to the input *x*.
5. Enable CLR input momentarily to clear all the flip flops.
6. Apply CLK through the switch and record your observations in the following table.
7. Apply 1 to the input *x*.
8. Enable CLR input momentarily to clear all the flip flops.
9. Apply CLK through the switch and record your observations in the following table.

x = 0				***x = 1***			
Clock Pulse #	***Q_1***	***Q_0***	***Output z***	***Clock Pulse #***	***Q_1***	***Q_0***	***Output z***
Initial value	0	0		Initial value	0	0	
1				1			
2				2			
3				3			
4				4			
5				5			

Alternatively, connect the CLK input to a continuous clock source. Now output will change continuously. These outputs can be viewed using an oscilloscope.

Lab Session 15

OBJECT

Experimenting with octal bus transceivers using parallel port PC interfacing

COMPONENTS AND APPARATUS REQUIRED

1. Digital logic trainer board or the following components:
   Bread board, 5 V - Power Supply, Multimeter, LEDs with Resistors, Switches
2. Logic probe
3. Connecting wires
4. Following ICs and their datasheets or pin configurations:
   - 74245 Octal Bus Transceivers with 3-State Outputs
5. Parallel Port Connectors (DB-25) with Cable
6. Personal Computer TurboC Compiler Installed

THEORY

3-State Logic

Sometimes it is necessary to isolate one part of circuit from the other part. For this purpose some TTL ICs are designed with 3-state logic. Such devices have three possible output states: *high, low* and *high impedance*. The device can be put into high impedance state by a control signal applied to an appropriate pin. When such a device is in high impedance state, there is very high impedance at its output, effectively isolating the device from whatever circuitry the output normally drives.

74245 Octal Bus Transceivers

A bus is any conducting path or set of paths having electrical connections to one or more devices. 74245 octal bus transceivers are mainly used to control connectivity between two devices or circuits. It also provides direction control for signal flow. Refer to appendix A for IC pin configurations. Function of various pin of this IC is given below:

- ***A1 through A8:*** Bus A.
- ***B1 through B8:*** Bus B.
- ***G´:*** Active low enable input. When high, sets all the bus pins to high impedance state.
- ***DIR:*** Direction control for signal flow. When set to high logic level, transfers bus A data to B bus. When set to low logic level, transfers bus B data to A bus.
- ***VCC and GND:*** Supply connections lines.

Serial and Parallel Port Connectors

A Parallel Port can consist of only 25 pin port adapter called a DB-25 and a serial port can consist of either a 25 pin port adapter called a DB-25 or 9 pin adapter called a DB-9 port adapter. Whether the port is a 9 pin or 25 pin it can accomplish all of the same tasks that serial port communications have been designed for.

Each adapter can be a male type connector with pins or a female type adapter with tiny holes. Generally a printer port (called LPT1) on the back of a computer is female type adapter and we need to use male DB-25 pin cable on it.

Parallel Port Connector – DB-25

The DB-25 connector (named for its "B"-size "D"-shaped shell and 25 pins) is practically ubiquitous in the electronics industry. The DB-25 connector is used for a variety of purposes. One of the most common applications is the parallel printer interface on the IBM PC.

The parallel port sends data 8-bits or one byte at a time in parallel. The other lines available on the DB-25 connector are a combination of status lines, control lines, and ground lines. The status and control lines are used for handshaking, commands, and feedback when we are talking to a printer.

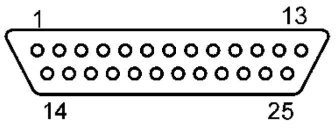


Figure 15.1: *DB-25 Male*

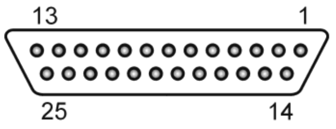


Figure 15.2: *DB-25 Female*

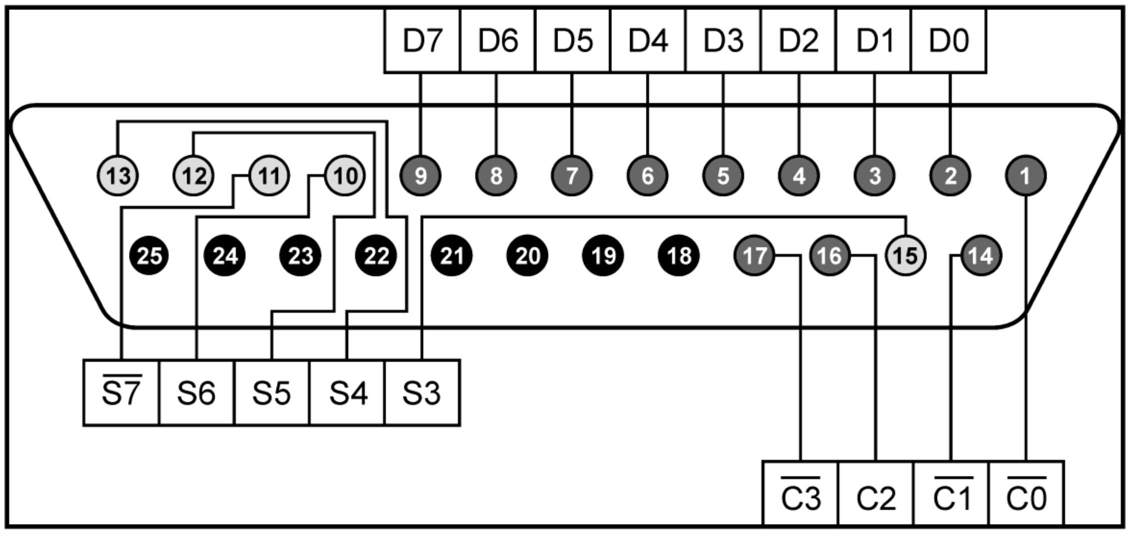


Figure 15.3: *Pin Layout for DB-25*

As can be seen form figure 15.3, there are four types of pins in this connector:

- ***Data pins (8 pins - D0 to D7):*** Active high bidirectional lines.
- ***Control pins (4 pins - C0 to C3):*** C0, C1 and C3 are active low, whereas C2 is active high. These lines also are bidirectional.
- ***Status pins (5 pins – S3 to S7):*** Only S7 is active low, rest are active high input lines.
- ***Ground pins (8 pins):*** Used as a reference signal for the low (below 0.5 volts) charge.

Parallel Port Addresses

A DB-25 connector at a parallel port has three address, one for each kind of pins / register: data, status and control. The first or base address generally refers to data register. Second address is for status register and the third one is for control register. In Windows OS, the base address is usually found to be 278H, 378H, or 3BCH. Now if 378H is the base address then it refers to the data register, whereas addresses for status and control register are respectively 379H and 37AH. In any version of Windows OS the base address can be found as follows:

1. On your Desktop, right-click on My Computer and select Properties.
2. Click on the Device Manager tab and find LPT1 under Ports.
3. After selecting LPT1, click the Properties button.
4. Next select the Resources tab and the address should then appear next to Input/Output Range.

Accessing parallel port through C Language Programming

There are multiple functions in C Language for accessing external ports. The most commonly used are listed below.

- `unsigned char inportb(int portid);`
  inportb reads a byte from a hardware port

- `int inport(int portid);`
  reads a word from a hardware port

- `void outportb(int portid, unsigned char value);`
  outputs a byte to a hardware port

- `void outport(int portid, int value);`
  outputs a word to a hardware port

`portid` refers to the port address, whereas `value` refers to the data to be sent

TESTING 74245 USING PARALLEL PORT PC INTERFACING

At one end the parallel port connector will be connected to the PC via parallel port cable and another connector (to be plugged into the PC parallel port slot). At the other end it will be connected to the buffer IC 74245. Data lines of the connector will be used to transfer 8-bit data between the PC and the buffer. Out of the 8-bits of 74245, 4-bits will be used to enter data, which will be displayed at the monitor through a C language program. The remaining 4-bits will receive data from PC as entered by the user via keyboard.

Circuit Diagram

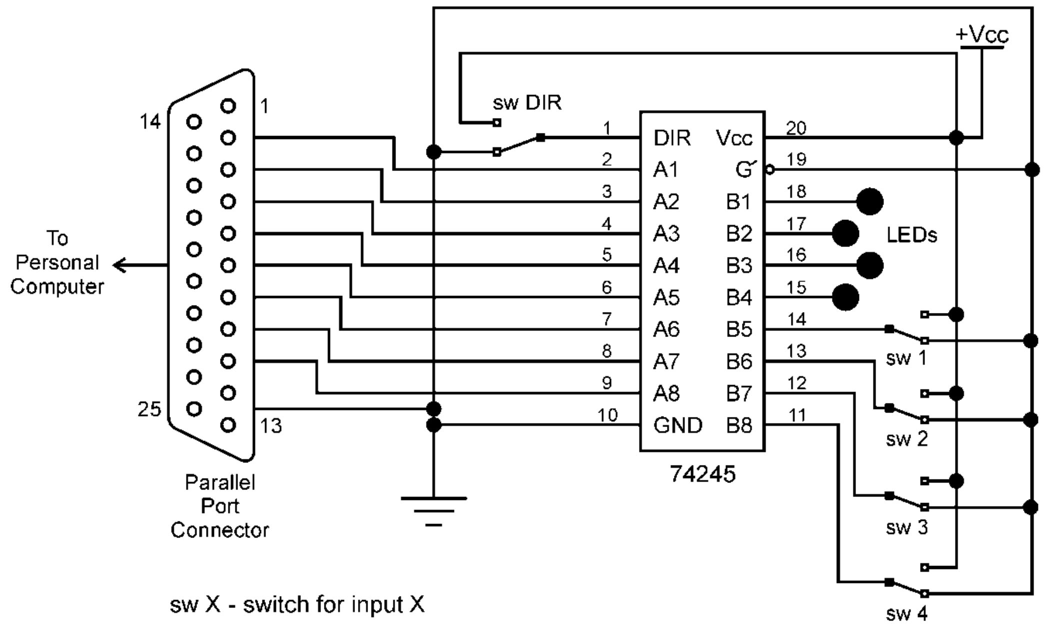


Figure 15.4: *Interface of 74245 to PC via Parallel Port*

Procedure

1. Connect the DB-25 connector to the PC at one end and connect it to the circuit board at the other end.
2. Connect various pins of the connector and 74245 according to the circuit diagram shown in figure 15.4.
3. Using the functions listed in the previous section, write a program in C language to operate various pins of the buffer. The user interface of the program should provide option to the user whether to send or receive data from the port.
4. Set DIR switch to 1.
5. Try sending some data bits to the port. The result should be displayed at the LEDs.
6. Set DIR switch to 0.
7. Set some binary 4-bit value through the switches. Try reading it via parallel port.

C Language Program

```
/* A program to access parallel port for data transfers */

/* use header file*/
#include <stdio.h>

/* main program */
void main()
{
    /* define variable */
    unsigned int data=0;
    char choice;

    while(1)
    {
      //printing option menu
      clrscr();
      printf("This program will send or receive a BCD digit from the
              parallel port\n");
      printf("\nPress your choice number\n");
      printf("\n1. Send Data");
      printf("\n2. Receive Data");
      printf("\n3. Exit\n");

      choice = getch();

      //sending entered digit to the port
      if(choice == '1')
      {
          //input BCD digit
          printf("\nEnter BCD digit to be sent to the port:");
          data = getche();

          //masking the first four bits
          data = data & 0x0f;

          //sending to the port
          outportb(0x378,data);
          printf("\nThe BCD digit %c has been sent to the port", data);
      }

      //printing on screen the data received from the port
      else if(choice == '2')
      {
          //receiving data from the port
          data = inportb(0x378);

          //masking the last four bits
          data = data & 0xf0;                                    \

          //shifting first four bits to make them the last four bits
          data>>=4;
          printf("\nThe BCD digit set at the port is: %c", data);
      }
      //exit
      else if(choice == '3')
          break;

      //key pressed is not in the menu
      else
          printf("\nEnter valid choice number");

          printf("\n\nPress any key to continue");
      getch();
    }
}
```

EXERCISES

1. In the above circuit, DIR and G´ inputs are hardwired. Connect them to status or control pins of the parallel port connector and control their function through your software program.

Circuit Diagram

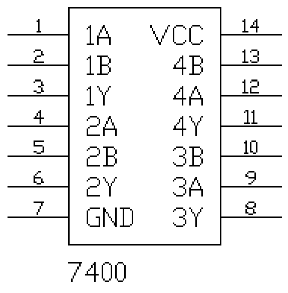
Program Code

Appendix A

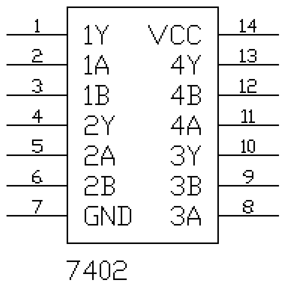
PIN DIAGRAMS OF THE ICS REQUIRED FOR THE LABORATORY SESSIONS

Consult the TTL/IC data book for internal diagrams and electrical characteristics of these ICs.

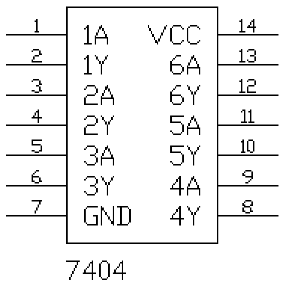
7400 Quad 2-Input NAND



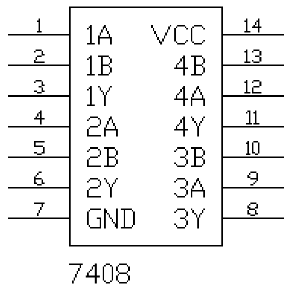
7402 Quad 2-Input NOR



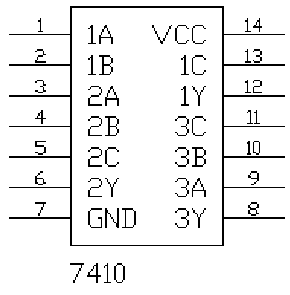
7404 Hex Inverter



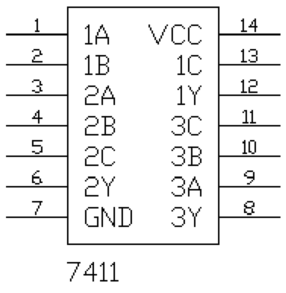
7408 Quad 2-Input AND



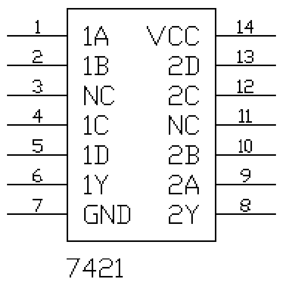
7410 Triple 3-Input NAND



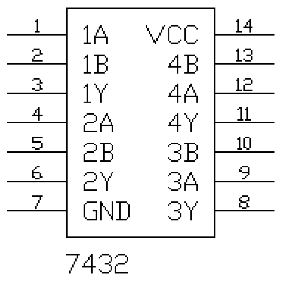
7411 Triple 3-Input AND



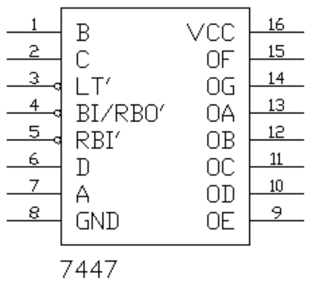
7421 Dual 4-Input AND



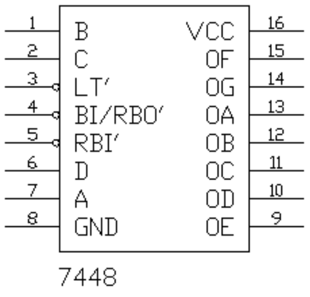
7432 Quad 2-Input OR



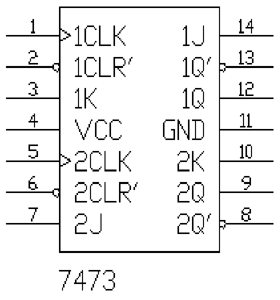
7447 BCD-to-Seven Segment Decoder (15V, O.C.)



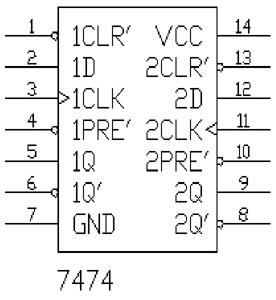
7448 BCD-to-Seven Segment Decoder (2kΩ pull-up output)



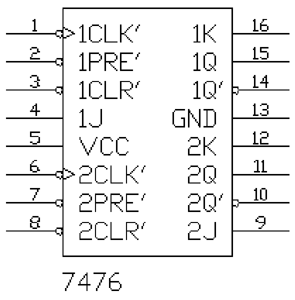
7473 Dual JK Flip-Flop



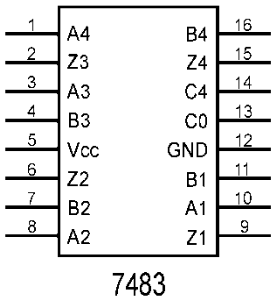
7474 Dual D-Type Flip-Flop



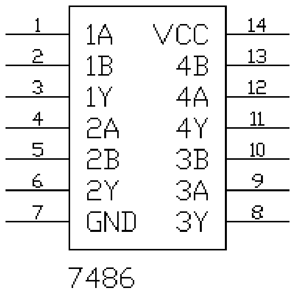
7476 Dual JK Flop-Flop



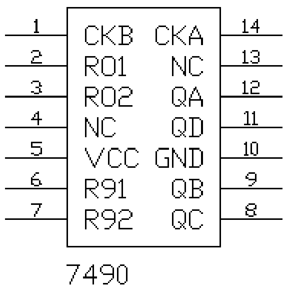
7483 4-Bit Binary Full Adder with Fast Carry



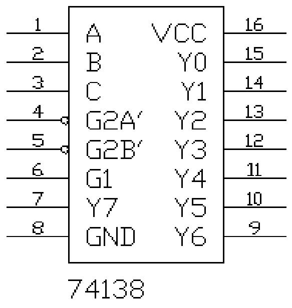
7486 Quad 2-Input Exclusive OR



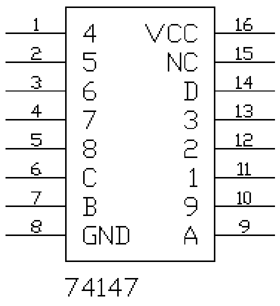
7490 Decade Counter



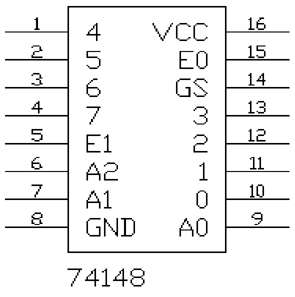
74138 3-to-8 Line Decoder / Demultiplexer



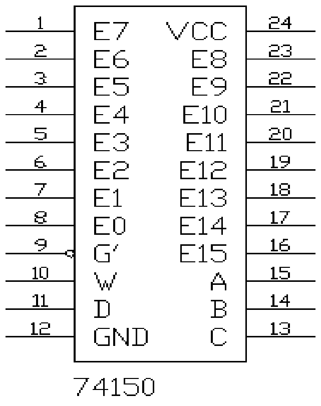
74147 10-to-4 Priority Encoder



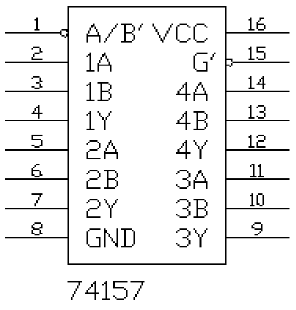
74148 8-to-3 Line Priority Encoder



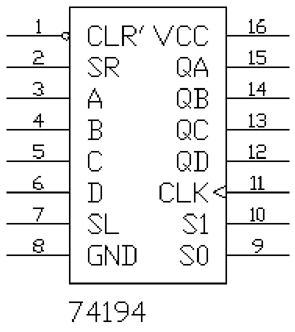
74150 16-to-1 Line Data Selector / Multiplexer



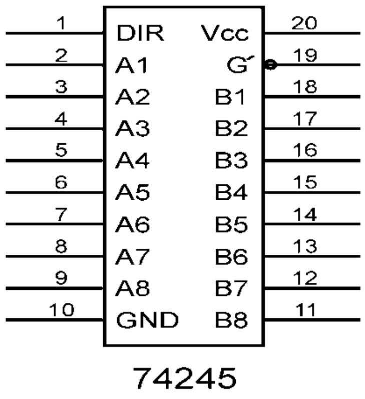
74157 Quad 2x1 Data Selector / Multiplexer



74194 4-Bit Bidirectional Universal Shift Register



74245 Octal Bus Transceivers (3-State)



Appendix B

COMMON COMPONENTS (BESIDE LOGIC ICS) USED IN BUILDING CIRCUITS

Battery

Battery supplies a voltage which drives an electric *current* round the circuit from the positive (+) terminal of the battery to its negative (–) terminal. Voltage is measured in volts (V) and current in *amperes* (A).

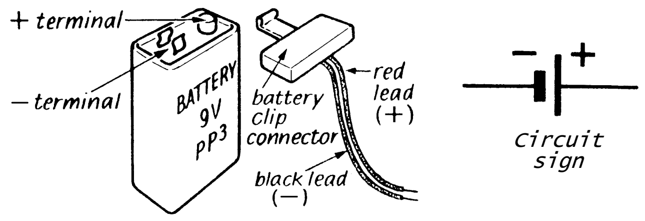


Figure B.1: *Battery (9V, type PP3)*

Connecting Wires

A connecting wire allows current to flow through it easily because it is made of copper which is a good electrical conductor. Insulators like PVC (polyvinyl chloride - a plastic) and enamel are used to cover connecting wires.

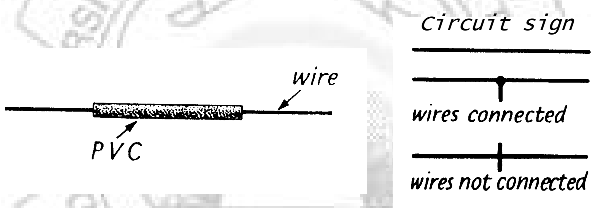


Figure B.2: *Connecting wire (PVC-covered tinned copper wire 1/0.6 mm, i.e. 1 wire of diameter 0.6 mm)*

Resistors

A resistor reduces the current in a circuit because of its resistance.

Color Coded Resistor

The colored bands give the resistance in *ohms*.

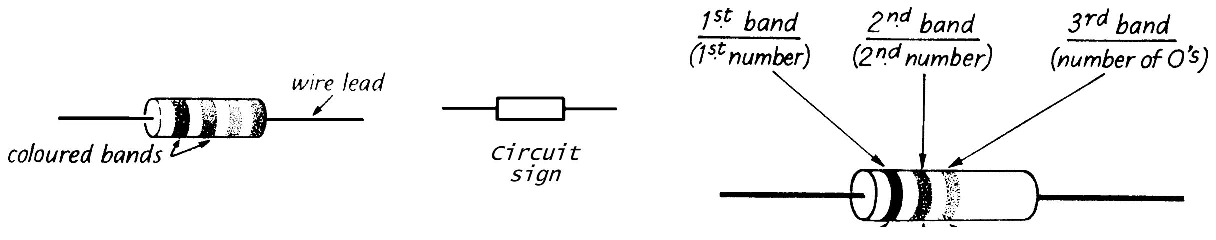


Figure B.3: *Color coded resistor (carbon, ½ watt)*

To distinguish left from right there is a gap between the C and D bands.

- Band A is the first significant figure of component value (left side)
- Band B is the second significant figure

- Band C is the decimal multiplier
- Band D if present, indicates tolerance of value in percent (no color means 20%)

The values are interpreted as given in table B.1.

Color	Significant figures	Multiplier	Tolerance
Black	0	x 10^0	-
Brown	1	x 10^1	±1%
Red	2	x 10^2	±2%
Orange	3	x 10^3	-
Yellow	4	x 10^4	-
Green	5	x 10^5	±0.5%
Blue	6	x 10^6	±0.25%
Violet	7	x 10^7	±0.1%
Gray	8	x 10^8	±0.05%
White	9	x 10^9	-
Gold	-	x 10^{-1}	5%
Silver	-	x 10^{-2}	10%
None	-	-	20%

Table B.1: *Standard color codes*

For example, a resistor with bands of yellow, violet, red and gold will have first digit 4, second digit 7, followed by 2 zeros: 4700 ohms. Gold signifies that the tolerance is ±5%, so the resistance could lie anywhere between 4465 and 4935 ohms.

Variable Resistor

Variable resistors are used when it is necessary to dynamically change the resistance in order to control the current in a circuit, and may also be used when a voltage divider is needed. For example, they are used to control the volume in a radio or the brightness of a lamp.

Variable resistors consist of a resistance track with connections at both ends and a wiper which moves along the track as you turn the spindle (see figure B.4). The track may be made from carbon, cermet (ceramic and metal mixture) or a coil of wire (for low resistances). The track is usually rotary but straight track versions, usually called sliders, are also available.

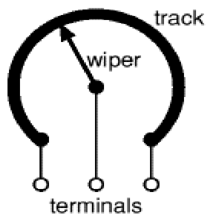


Figure B.4: *Variable resistor*

They are specified by their maximum resistance, linear or logarithmic track, and their physical size. The standard spindle diameter is 6mm. The resistance and type of track are marked on the body (for example: 4K7 LIN means 4.7 kΩ linear track ; 1M LOG means 1 MΩ logarithmic track). Linear (LIN) track means that the resistance changes at a constant rate as you move the wiper. This is the standard arrangement which is assumed if a project does not specify the type of track. Logarithmic (LOG) track means that the resistance changes slowly at one end of the track and rapidly at the other end, so halfway along the track is not half the total resistance. This arrangement is used for volume (loudness) controls because the human ear has a logarithmic response to loudness so fine control (slow change) is required at low volumes and coarser control (rapid change) at high volumes. It is important to connect the ends of the track the correct way round, if turning the spindle increases the volume rapidly followed by little further change you should swap the connections to the ends of the track.

Variable resistors may be used as a rheostat with two connections (the wiper and just one end of the track) or as a potentiometer with all three connections in use. Miniature versions called presets are made for setting up circuits which will not require further adjustment.

The terminal in the middle is the *wiper*. When a potentiometer is used as a voltage divider, all three terminals are wired separately. But when a potentiometer is used strictly as a rheostat, only need two connections are needed. Either side of the variable resistor may be attached to the circuit board, with the remaining side unattached or grounded, but it is important to always connect the wiper. The wiper must be grounded or affixed to the voltage source. For example, the left terminal of the pot may be attached to the voltage source and the wiper to ground, or the right terminal may be substituted for the left. Changing the connection to a different side alters the direction the knob must be turned in order to achieve maximum resistance. The unused side of the variable resistor may be left unconnected, wired to an unused portion of the breadboard, or wired to ground.

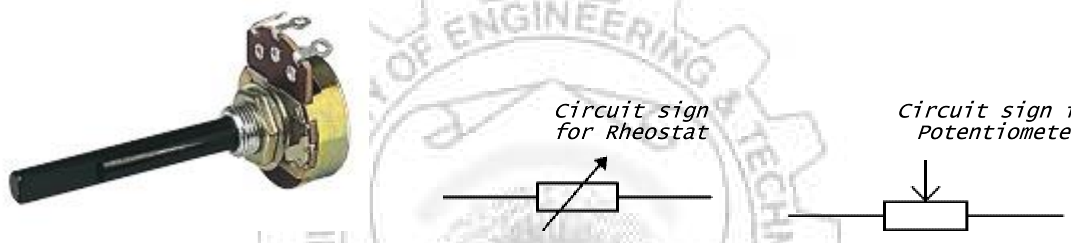


Figure B.5: *Potentiometer and Rheostat*

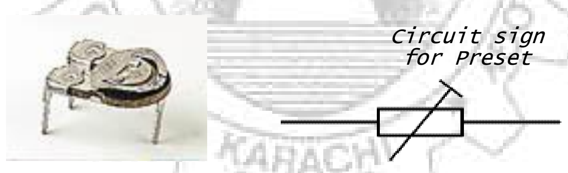


Figure B.6: *Preset*

Light Dependent Resistor (LDR)

When light falls on it, its resistance becomes small; in the dark resistance is high.

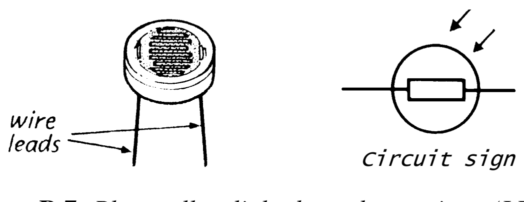


Figure B.7: *Photocell or light dependent resistor (LDR)*

Capacitors

A capacitor stores electricity; the greater the capacitance the more does it store. Capacitance values are measured in microfarads shortened to μF or, less correctly, to mfd. On a capacitor, 0.1 μF may be marked as .1 mfd and 0.01μF as 10n. The greatest voltage it can stand is also shown, e.g. 30V.

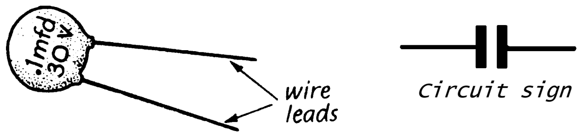


Figure B.8: *Capacitor (disc ceramic type)*

Electrolytic Capacitor

Electrolytic capacitor stores electricity: values usually larger than 1µF. Greatest voltage marked on it. Must be connected the correct way round.

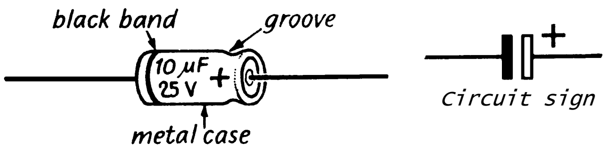


Figure B.9: *Electrolytic Capacitor*

Variable Capacitor

It varies the capacitance in a circuit by moving one set of metal plates in or out of a fixed set when the spindle is rotated. The sets of plates are separated by sheets of an insulator (also called a dielectric).

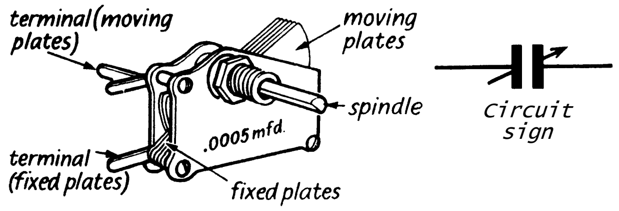


Figure B.10: *Variable Capacitor (0.0005 microfarads)*

Loud Speaker

It changes electric currents into sound.

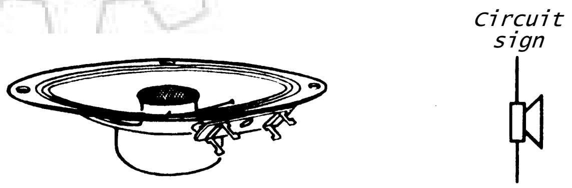


Figure B.11: *Loud Speaker (2.5 inch, 25 to 80*

Aerial

Changes radio waves into electric currents.

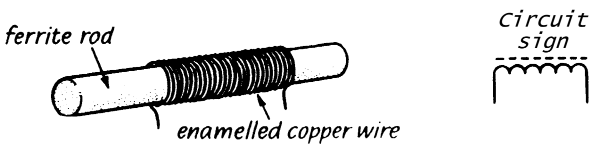


Figure B.12: Ferrite rod aerial

Light Emitting Diode (LED)

An LED lets current flow in one direction but not in the other. When it conducts, light is emitted. Must have a current limiting resistor in series with it. The cathode lead is nearest the 'flat' and may be shorter than the anode lead (but this is not always so). The arrow on the sign shows the conducting direction

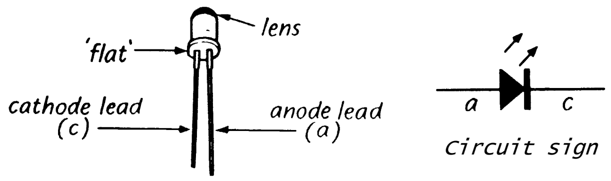


Figure B.13: *LED (light emitting diode)*

Transistor

Transistor amplifies small currents into much larger copies. It acts as a very fast switch. It must be correctly connected with a positive voltage to the collector

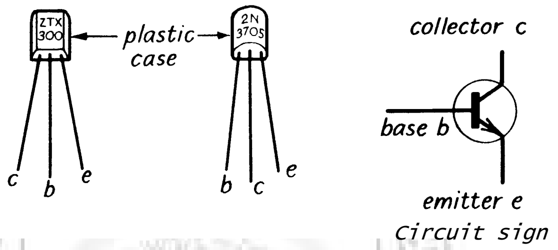


Figure B.14: *Transistor (npn)*
(e.g. ZTX300 or 2N3705)

Switches

A switch is an electrical component that can break an electrical circuit, interrupting the current or diverting it from one conductor to another.

SPDT Switch

Connects terminal A to terminal B or C, i.e. it is a change-over switch.

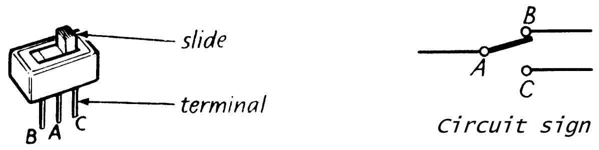


Figure B.15: *Miniature slide switch SPDT (single pole double throw)*

DIP Switch

A DIP switches are manual electric switches that are packaged in a group in a standard dual in-line package (DIP) (the whole package unit may also be referred to as a DIP switch in the singular). It provides an easy way of inputting values to a digital circuit. The Dip Switch requires a Resistor Network Pack for its operation.

The connection configuration is shown in figure B.17. Resistor Network Pack has a pin configuration that must be followed. The side of the resistor pack with the print is the "front." At the far left end of the component there is a black circle. This pin must be connected to the power supply. The other pins do not required to be connected to power. Once the dipswitch and resistor pack is placed on the board, the next thing that needs to be done is to connect wires from all the columns connected to the dip-switch to the ground rail. This way one side of the dip-switch is "tied" to ground.

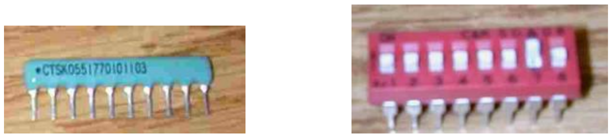

(a) *(b)*

Figure B.16: *(a) Resistor Network Pack, (b) DIP Switch*

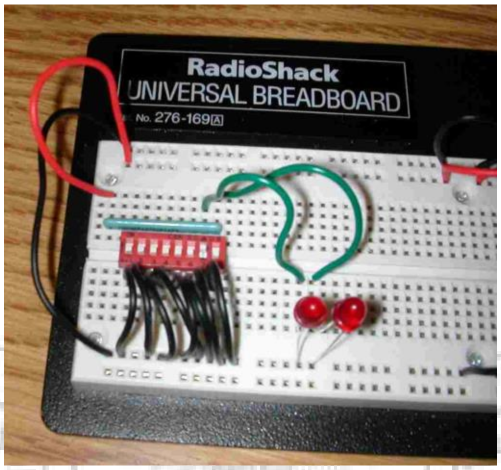


Figure B.17: *Circuit Connections of DIP switch with resistor network pack*

The dip-switch can now be used to "input" values. When the switch is pushed to one side it disconnects the top column from the bottom column on the breadboard. Voltage appears on the column due to the pin of the resistor pack pin connected to that column. When switch is pushed to the other side, the dip-switch connects the top column to the bottom column. Since the top column along with the pin of the resistor pack is "tied" to ground, no voltage appears on the top column. The idea of the dip-switch is that when the dip switch is open, or there is no connection between the top and bottom column, current flows from the resistor pack to the electrical component. However, when the dip-switch is closed, or the top and bottom column are tied together, current flows from the resistor pack to ground rather than to the electrical component. This resistor-pack/dip-switch circuit should be placed as far to the left side of the breadboard as possible to leave space for other components.

Push Button

A Push Switch or Push to make switch, allows electricity to flow between its two contacts when held in. When the button is released, the circuit is broken. Other forms are push to break which, does the opposite.

Relay

A relay is an electrically operated switch. Relays are used where it is necessary to control a circuit by a low-power signal (with complete electrical isolation between control and controlled circuits), or where several circuits must be controlled by one signal. These devices use a solenoid to control a heavy-duty switch. The wiring for the solenoid may require only 0.5 amps to activate, while the switch it controls carries 10 to 30 amps.

A Solid State Relay (SSR) is an electronic switch that works without moving parts. Here the low current control and a high current load are isolated optically or with transformers. They are activated by AC control signals or DC control signals from Programmable logic controller (PLCs), PCs, Transistor-transistor logic (TTL) sources, or other microprocessor and microcontroller controls. Since relays are switches, the terminology applied to switches is also applied to relays. A relay will switch one or more *poles*, each of whose contacts can be *thrown* by energizing the coil in one of three ways:

- Normally-open (NO) contacts connect the circuit when the relay is activated; the circuit is disconnected when the relay is inactive.
- Normally-closed (NC) contacts disconnect the circuit when the relay is activated; the circuit is connected when the relay is inactive.
- Change-over (CO), or double-throw (DT), contacts control two circuits: one normally-open contact and one normally-closed contact with a common terminal.

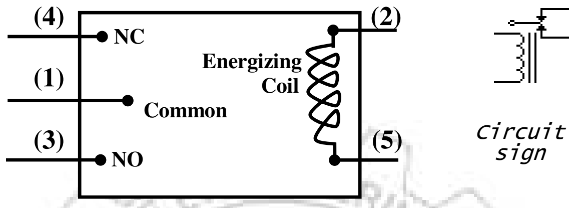
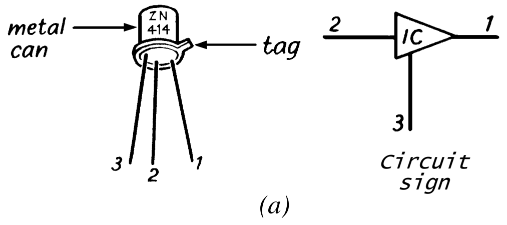


Figure B.18: *Relay*

For circuit connections, consider figure B.18, pins 2 and 5 seem to go to the coil. Pin 1 is the common pin. Pin 4 is the NC (normally closed) pin and pin 3 is the NO (normally open) pin. Connect the control device to the two solenoid pins (pins 2 and 5) of the relay. Supplying a small voltage to these two pins will turn the relay switch on or off. The lower the better until you are sure what voltage the relay can take (can start from 3V or 5V). When the relay is not energized, there should be continuity between pins 1 and 4. When the relay is energized, there should be continuity between pins 1 and 3. Connect the negative terminal of the battery to pin 1. This pin brings the power into the relay for powering the external device, which could be the electric motor of a fan or light. If the external device should only be on when the relay is energized, connect a wire from the negative terminal of the external device to pin 3 of the relay. If the external device should be on at all times except when the relay is energized, connect to pin 4 instead of pin 3. Complete the circuit by connecting a wire from the positive terminal of the external device to the positive terminal of the battery.

Integrated Circuits (ICs)

Transistors, diodes, resistors and capacitors are connected together on a tiny ‘chip’ of silicon (sand is mostly silicon oxide) to give any desired circuit, e.g. a multistage amplifier; an astable, bistable or monostable multivibrator; a counter; a logic gate for a computer; several stages of a TRF (tuned radio frequency) radio.



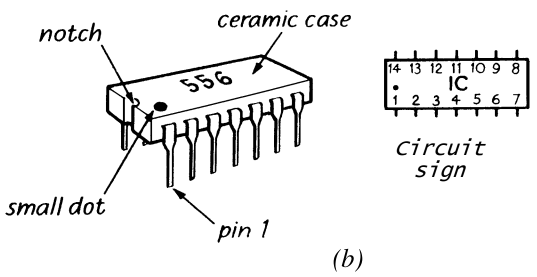


Figure B.19: *Integrated circuits (‘chips’)*
(a) Can type
(b) Dual in line (d.i.l) type (14 or 16 pins)

They must be correctly connected. Pin 1 is next to the 'tag' in the can type and on the d.i.l. type it is identified from the 'notch' or 'small dot' on the case. TTL and CMOS are two technologies in which logic gate ICs are available. CMOS 'chips' (standing for *C*omplementary *M*etal *O*xide *S*emiconductors and pronounced 'see-moss') need special care.

CMOS (4000 series) General Characteristics

- Supply: 3 to 15V, small fluctuations are tolerated.
- Inputs have very high impedance (resistance), this is good because it means they will not affect the part of the circuit where they are connected. However, it also means that unconnected inputs can easily pick up electrical noise and rapidly change between high and low states in an unpredictable way. This is likely to make the chip behave erratically and it will significantly increase the supply current. To prevent problems all unused inputs MUST be connected to the supply (either +Vs or 0V), this applies even if that part of the chip is not being used in the circuit.
- Outputs can sink and source only about 1mA if you wish to maintain the correct output voltage to drive CMOS inputs. If there is no need to drive any inputs the maximum current is about 5mA with a 6V supply, or 10mA with a 9V supply (just enough to light an LED). To switch larger currents you can connect a transistor.
- Fan-out: one output can drive up to 50 inputs.
- Gate propagation time: typically 30ns for a signal to travel through a gate with a 9V supply, it takes a longer time at lower supply voltages.
- Frequency: up to 1MHz, above that the 74 series is a better choice.
- Power consumption (of the chip itself) is very low, a few µW. It is much greater at high frequencies, a few mW at 1MHz for example.
- Damage occurs if static charges build up on input pins when, for example, they touch insulating materials (e.g. clothes, plastic pen) in warm, dry conditions.
  1. Keep the IC in the carrier in which it is supplied until it is inserted in the circuit.
  2. Do not finger the pins or hold them in contact with an insulator.
  3. Connect all unused inputs of the IC to either the positive or the negative of the battery, depending on the circuit.

74LS series TTL Characteristics

- Supply: 5V ±0.25V, it must be very smooth, a regulated supply is best. In addition to the normal supply smoothing, a 0.1µF capacitor should be connected across the supply near the chip to remove the 'spikes' generated as it switches state, one capacitor is needed for every 4 chips.
- Inputs 'float' high to logic 1 if unconnected, but do not rely on this in a permanent (soldered) circuit because the inputs may pick up electrical noise. 1mA must be drawn out to hold inputs at logic 0. In a permanent circuit it is wise to connect any unused inputs to +Vs to ensure good immunity to noise.
- Outputs can sink up to 16mA (enough to light an LED), but they can source only about 2mA. To switch larger currents you can connect a transistor.
- Fan-out: one output can drive up to 10 74LS inputs, but many more 74HCT inputs.
- Gate propagation time: about 10ns for a signal to travel through a gate.
- Frequency: up to about 35MHz (under the right conditions).
- Power consumption (of the chip itself) is a few mW.

Building Circuits

The circuit board shown in figure B.20 accepts ICs as well as separate components. It has 47 rows of 5 interconnected sockets on each side of a central channel across which d.i.l. ICs can be fitted. A wire inserted in a socket in a certain row becomes connected to wires in any of the other 4 sockets in that row by a metal strip under the board. For example, wires in sockets B5, C5, D5, E5 and F5 are all joined. Metal strips under the board connect the sockets.

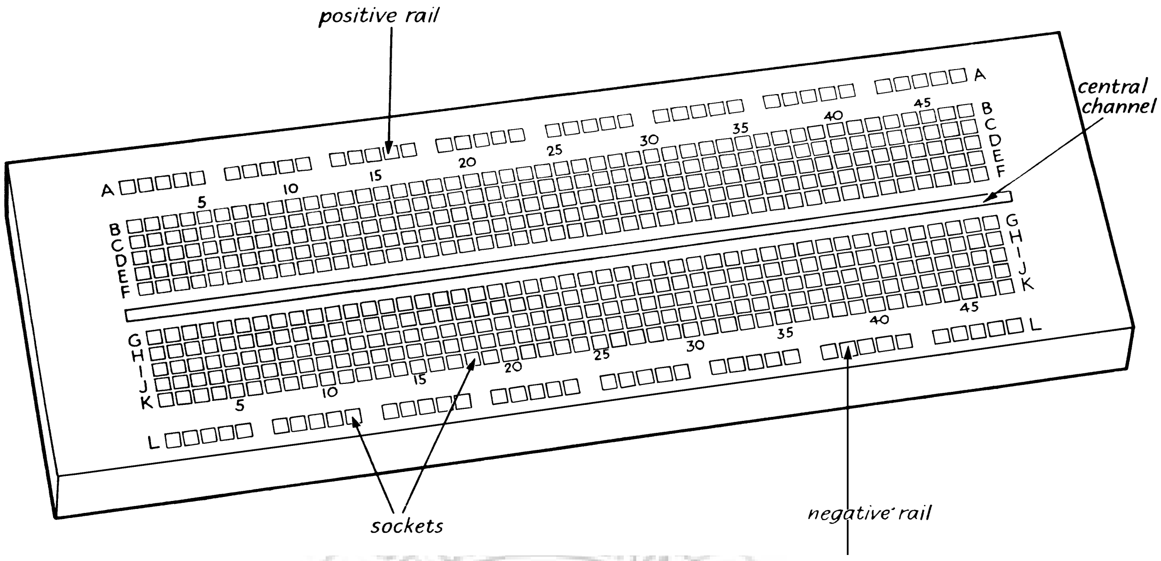


Figure B.20: *Circuit board*

The circuit board shown above accepts ICs as well as separate components. It has 47 rows of 5 interconnected sockets on each side of a central channel across which d.i.l. ICs can be fitted. A wire inserted in a socket in a certain row becomes connected to wires in any of the other 4 sockets in that row by a metal strip under the board. For example, wires in sockets B5, C5, D5, E5 and F5 are all joined. Metal strips under the board connect the sockets. There is a row of 40 interconnected sockets along the top of the board and a similar row along the bottom act as the positive and negative power supply rails (called 'bus bars'). Various makes of circuit board are available, some with vertically mounting removable panels for supporting controls.

To make a connection push about 1 cm of the bare end of a wire (0.25 to 0.85 mm diameter) straight into the socket (not at an angle) so that it is gripped by the metal strip under the board. Do not use wires that are dirty or have kinked ends. Only put one wire in each socket. Bend leads on resistors etc., as shown in figure B.21 before inserting them in the board.

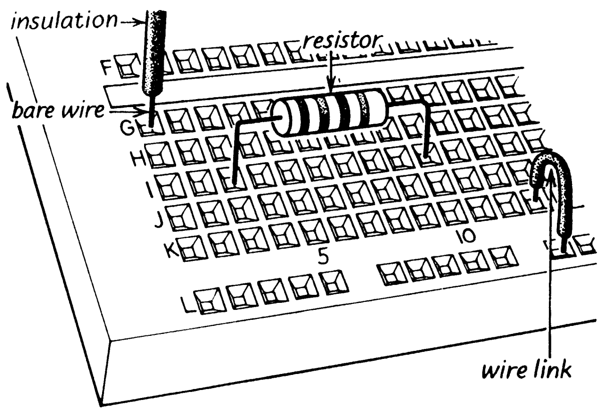


Figure B.21: *Placing various components on the circuit board*

Bare the ends of connecting wire (PVC-covered tinned copper wire 0.6 mm diameter) by removing the insulation (PVC) either with wire strippers or using a pair of blunt-nosed pliers and a pair of side cutters as shown in figure B.22. With practice you should be able to judge just how much the side cutters have to be squeezed and pulled to remove the insulation without cutting the wire.

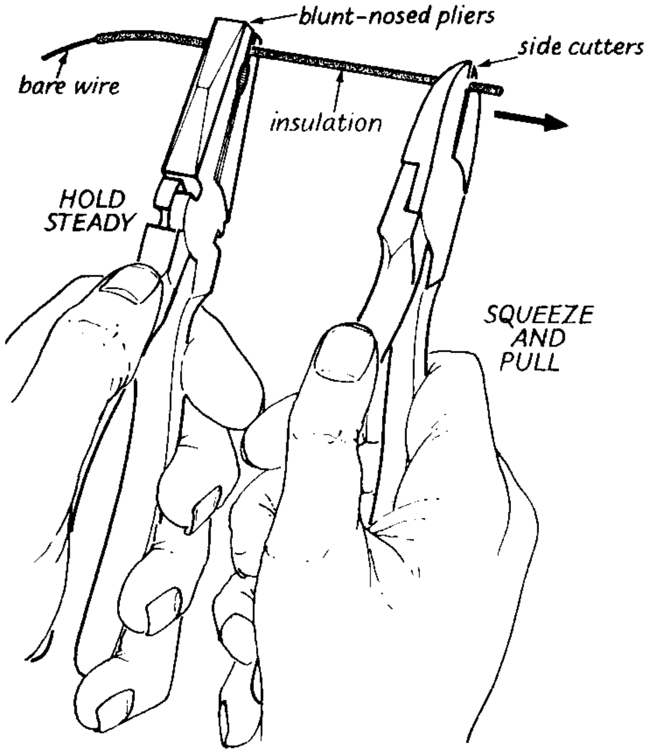


Figure B.22: Stripping wire

Appendix C

C.1 CLOCK GENERATION USING 555 TIMER IC

The 555 timer is a versatile and widely used device which can be used effectively without understanding the function of each pin in detail. It can be configured in two different modes as either a *monostable multivibrator (one-shot)* or as an *astable multivibrator (oscillator)*. An astable multivibrator has no stable state and therefore changes back and forth (oscillates) between two unstable states without any external triggering, hence can be used for clock generation of desired frequency and duty cycle. The circuit given in figure C.1 shows the connections of 555 timer in astable mode.

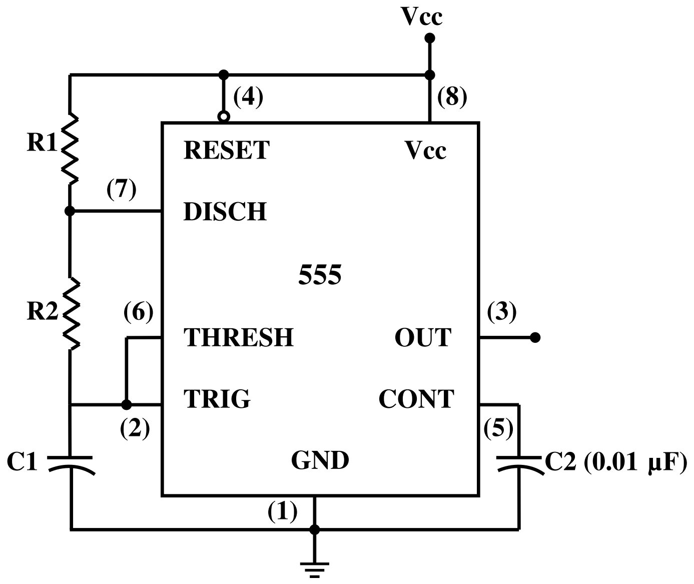


Figure C.1: *Connections of 555 timer IC in astable mode*

The frequency, or repetition rate, of the output pulses is determined by the values of two resistors, R1 and R2 and by the timing capacitor, C1. The design formula for various parameters are as follows:

1. Frequency , $f = \dfrac{1.44}{(R1 + 2R2) \times C1}$
2. Period , $t = \dfrac{1}{f} = 0.69\,(R1 + 2R2) \times C1$
3. HIGH time = $0.69\,(R1 + R2) \times C1$
4. LOW time = $0.69\,(R2 \times C1)$
5. Duty Cycle = $\dfrac{\text{HIGH time}}{\text{period}}$

Before calculating a frequency, it is usual to make R1=1 kΩ because this helps to give the output pulses a duty cycle close to 50%, that is, the HIGH and LOW times of the pulses are approximately equal. By selecting C1 = 10μF and R2 = 140kΩ, output pulses having approximately 1Hz frequency can be generated. Another usual practice is to replace *R*2 with a potentiometer (usually 500kΩ) so that the output frequency can be changed at runtime.

C.2 DEBOUNCING CIRCUITRY FOR MECHANICAL SWITCHES

Mechanical switches form the interface between human beings and computers or other digital systems. For example, a keyboard is a matrix of switches used to supply alphanumeric data to a computer. Whatever the switch design, it is a potential source of problems due to contact bounce. The contact bounce creates a sense of narrow pulses when a switch is opened or closed. An example of device whose operation would be adversely affected by contact bounce is a digital counter used to count the number of times a switch is depressed. Eliminating the effects of contact bounce is called *debouncing.*

When data is entered into a computer via a keyboard, a *software debounce* is often used. This type of debouncing is a program that causes the computer to sample the switch terminal (i.e., to input data from it) many times in succession during the interval of time that contact bounce occurs. If the data is sensed to be 1s (or 0s) for a specific number of consecutive samples, then it is assumed that contact bounce has ended and the last value sensed is valid.

Hardware debouncing is the use of electronic circuitry to eliminate the effects of contact bounce. There are numerous versions of such circuitry, including those that use monostable multivibrators (one-shots), but the most straightforward is simply an RS latch. Figure B.1 shows the circuit. When the switch is in position 1, *R* = 0 and *S* = 1, so the latch is set and the output (*Q*) is 1. When the switch is in position 0, *R* = 1and *S* = 0, so the latch is reset and the output is 0. When the switch is moved from one position to the other, the latch changes state and bouncing occurs at either the *R* or the *S* input. The bouncing does not affect the latch after it has change state. An RS latch will remain set, for example, when its *R* input is 0 and its *S* input is alternately changed form 1 to 0.

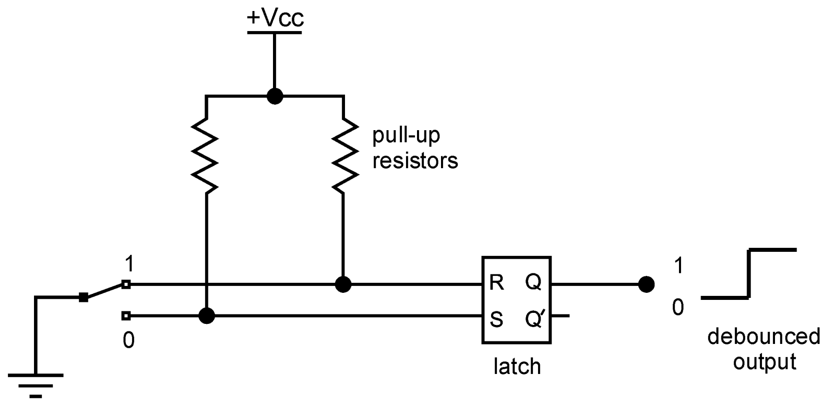


Figure B.1: *Use of RS latch to debounce a mechanical switch*