International Journal For Technological Research In Engineering
Volume 6, Issue 8, April-2019 *ISSN (Online): 2347 - 4718*

DESIGN SINGLE CYCLE MICROPROCESSOR USING VHDL

Naveen Kumar Saini
Asst. Prof. EE, RTU, Shekhawati Institute Of Engineering, Sikar, India

Abstract: In this paper, a A microprocessor is a digital electronic component with transistors on a single semiconductor integrated circuit (IC).One or more microprocessors typically serve as a central processing unit(CPU) in a computer system or handheld device .Microprocessors made possible the advent of the microcomputer .Before this electronic CPUs were typically made from bulky discrete switching device containing the equivalent of only a few transistors .By integrating the processor onto one or a very few large-scale integrated circuit package, the cost of processor power was greatly reduced. Since the advent of the IC in the mid-1970s, the microprocessor has become the most prevalent implementation of the CPU, nearly completely replacing all others forms. This project is trying to design an 8 bit microprocessor by using VHDL .VHDL is stand for very high speed integrated circuit hardware description language .It is one of the most popular design application uses by most designers nowadays .The microprocessor will be synthesize in VHDL using Xilinx ISE. The 8 bit microprocessor is widely use in microcontroller devise with specific task because it has a specific task because it has a specific instruction where it only done a given instruction.
Keorywords: integrated circuit (IC), VHDL, Xilinx

I. INTRODUCTION

In this paper, Microprocessors are the heart of all "smart" devices, whether they be electronic devices or otherwise. Their smartness comes as a direct result of the decisions and controls that microprocessors make. For example, we usually do not consider a car to be an electronic device. However, it certainly has many complex, smart electronic systems, such as the anti-lock brakes and the fuel-injection system. Each of these systems is controlled by a microprocessor. Yes, even the black, hardened blob that looks like a dried-up and pressed-down piece of gum inside a musical greeting card is a microprocessor. There are generally two types of microprocessors: general-purpose microprocessors and dedicated microprocessors.General-purpose microprocessors, such as the Pentium CPU, can perform different tasks under the control of software instructions. General-purpose microprocessors are used in all personal computers.

II. THE INTEL 486 FAMILY

When Intel finally did catch up, it did so in a big way with the release of the 80486 processor, better known as the 486, in 1989. In addition to adding an 8K primary cache, Intel decided to integrate a floating-point math coprocessor (FPU) to an improved 386 core with scalar architecture, and a full 32-bit data and address bus width. Because of the high transistor count, which numbered 1.2 million, the infant mortality rate was high. At first, only 30 percent of the chips survived the testing process from start to finish. Most of the failures occurred in the math coprocessor section, which occupied about two-third of the chip's real estate.

So it took no time at all for Intel to switch to testing its 486 chips for CPU performance first, and follow up with a math coprocessor check. Those chips that passed the first phase but flunked the math were labeled 486SX, and went into lower-priced, conventional desktops without math processors. Those chips that passed their math tests went into a higher-priced model, now called the 486DX. As yields improved, and demand for the lower-priced 486SX increased, the 486SX selection process was winnowed down to testing the CPU only, with a subsequent blowing of the fuse that fed power to the math coprocessor (just in case the math coprocessor was functional, but flawed). Another important feature of the 486 line was the introduction of 3.3-volt technology — something that Motorola couldn't match until two years later. Up to 1990 microprocessor logic was based on a 5-volt power supply. However, the amount of heat a semiconductor generates is a function of speed multiplied by voltage. By 1990, the speed of computer chips (both microprocessors and external logic chips) hit a heat barrier — if they went any faster, they'd simply burn up. Reducing the voltage reduced the heat build-up and let the chips run faster. By the year 2000, its expected computer chips will run at 500 MHz using 0.9-volt power sources. What was Motorola doing all this time? It was busy working on the 68060, a third generation 68000 chip that introduced the concept of superscalar pipelining, a technique again borrowed from mainframe technology, which permits multiple instructions to run at the same time. This chip saw the light of day in early 1994. Motorola was also busy developing a line of microcontroller chips, like the 68HC11.

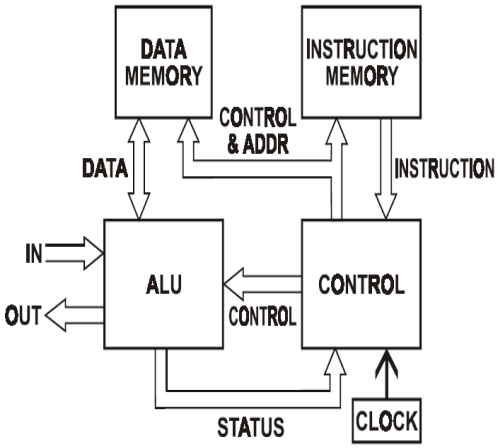

Figure1.2:-Harvard Architecture Microprocess

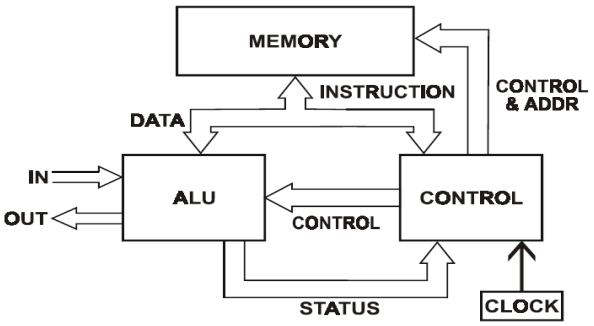

Figure1.3:-Princeton Architecture Microprocessor

III. CODING OF MICROPROCESSOR MULTIPLEXER

*A. 2*1 4-bit multiplexer :-*

```
libraryieee;
use ieee.std_logic_1164.all;
useieee.std_logic_unsigned.all;
useieee.numeric_std.all;
entity mux2 is port (
s:                      in std_logic;
x0, x1:                 in std_logic_vector(3 downto 0);
y: out std_logic_vector(3 downto 0);
end mux2;

architecture imp of mux2 iss
begin
process(s, x0, x1)
begin
if(s= ’0’) then
        y<= x0;
else
        y<= x1;
end if;
end process;
end imp;
```

*B. 4*1 8-bit multiplexer:-*

```
libraryieee;
use ieee.std_logic_1164.all;
useieee.std_logic_unsigned.all;
useieee.numeric_std.all;
entity mux2 is port (
s:                              in std_logic_vector(1 downto 0);
x0, x1, x2, x3:                 in std_logic_vector(7 downto 0);
y:                              out std_logic_vector(7 downto
0);
end mux4;
architecture imp of mux4 is
begin
process(s, x0, x1, x2, x3)
begin
case s is
when “00”   =>y <= x0;
when “01”   =>y <= x1;
when “10”   => y <= x2;
when “11”   =>y <= x3;
when others =>y <= (others => ‘x’);
end case;
end process;
end imp;
```

C. Full Adder:-

```
libraryieee;
use ieee.std_logic_1164.all;
useieee.std_logic_unsigned.all;

entity FA is port (
carryIn:                        in std_logic;
carryOut:                       out std_logic;
x, y:           in std_logic;
s:              out std_logic);
end FA;

architecture imp of FA is
begin
      s <= x xor y xorcarryIn;
carryout<= (x and y) or (carryIn and (x xor y));
end imp;
```

D. Add and Subtract:-

```
entity addsub8_pc is port (
A:                              in std_logic_vector(7 downto
0);
B:                              in std_logic_vector(7 downto
0);
F:                              out std_logic_vector(7 downto
0);
Sub:                            in std_logic);
end addsub8_pc;
architecture imp of addsub8_pc is
begin
process(A, B, sub)
begin
if (sub= ’0’)     then
                F <= A+B;
else
                F<= A-B;
end if ;
end process;
end imp;
```

E. Opcode Definition:-

```
PACKAGE opcodes  IS
    SUBTYPE t_cond1 IS std_logic_vector (2 DOWNTO
0);
      CONTANT sta                       : t_cond1 :=
“000”;
      CONSTANT lda                      : t_cond1 :=
“001”;
      CONSTANT movi                     : t_cond1 :=
“010”;
CONSTANT inp                    : t_cond1 := “011”;
CONSTANT outp                   : t_cond1 := “100”;
CONSTANT jnz                    : t_cond1 := “101”;
      CONSTANT adda                     : t_cond1 :=
“110”;
      CONSTANT suba                     : t_cond1 :=
```



```
"111";

SUBTYPE t_org IS std_logic_vecter (4 DOWNTO 0);
CONSTANT  A                    : t_oreg := "00000";
CONSTANT  B                    : t_oreg := "00001";
CONSTANT  C                    : t_oreg := "00010";
CONSTANT  D                    : t_oreg := "00011";
CONSTANT  E                    : t_oreg := "00100";
END opcodes;
```

F. Program Counter:-

```
libraryieee;
use ieee.std_logic_1164.all;
useieee.std_logic_unsigned.all;
useieee.numeric_std.all;

entity PC is port (
clk:                            in std_logic;
reset:                          in std_logic;
load:                           in std_logic;
  INPUT:                              in std_logic_vector(7
downto 0);
  OUTPUT:                             out std_logic_vector(7
downto 0));
end PC;

architecture imp of PC is
component FF is Port(
clk:          in std_logic;
reset:        in std_logic;
load:         in std_logic;
     D:           in std_logic;
     Q:           in std_logic);
end component;
begin
  U0:          FF port map (clk, reset, load, INPUT(0),
OUTPUT(0));
  U1:          FF port map (clk, reset, load, INPUT(1),
OUTPUT(1));
  U2:          FF port map (clk, reset, load, INPUT(2),
OUTPUT(2));
  U3:          FF port map (clk, reset, load, INPUT(3),
OUTPUT(3));
  U4:          FF port map (clk, reset, load, INPUT(4),
OUTPUT(4));
  U5:          FF port map (clk, reset, load, INPUT(5),
OUTPUT(5));
  U6:          FF port map (clk, reset, load, INPUT(6),
OUTPUT(6));
  U7:          FF port map (clk, reset, load, INPUT(7),
OUTPUT(7));
end imp;
```

G. Control Unit:-

```
entity controller is port (
clk:                            in std_logic;
reset:                          in std_logic;
Aeq0:                           in std_logic;
IR:                                 in std_logic_vector(7
downto 5);
ALUSel:                             out std_logic_vector(1
downto 0);
Asel:                               out std_logic_vector(1
downto 0);
writeAcc:                       out std_logic;
IRload:                          out std_logic;
PCload:                         out std_logic;
Oload:                          out std_logic;
jmpMux:                         out std_logic;
opfetch:                        out std_logic;
we:                             out std_logic;
rbe:                            out std_logic;
end controller;

architecture imp of controller is
Type state_type is (
s_start,
s_fetch,
s_decode,
s_jnz,
s_in,
s_out,
s_add,
s_sub,
s_store,
s_load,
s_mov);
signal state: state_type := s_start;
signalclkcount: std_logic_vector(7 downto 0);
begin
NEXT_STATE_LOGIC: process(reset, clk)
begin
if(reset= '1')   then
state<= s_start;
clkcount<= X"00";
elsif(clk ' event and clk='1')  then
clkcount<= clkcount + 1;
case state is
whens_start => state <= s_fetch:
whens_fetch => state <= s_decode;
whens_decode =>
case IR(7 downto 5) is
when "000" => state <= s_store;
when "001" => state <= s_load;
when "010" => state <= s_mov;
when "011" => state <= s_in;
when "100" => state <= s_out;
when "101" => state <= s_jnz;
when "110" => state <= s_add;
when "111" => state <= s_sub;
when others => state <= s_start;
end case;
when others => state <= s_fetch;
end case;
end if;
end process;
OUTPUT_LOGIC: process (state)
begin
```



```
case state is
whens_start =>jmpMux<=  ‘0’;
writeAcc<= ‘0’;
whens_fetch =>IRload<= ‘1’;
PCload<= ‘1’;
Jmpmux<= ‘0’;
opfetch<= ‘1’;
ALUSel<= “XX”;
Asel<= “XX”;
Oload<= ‘0’;
we<= ‘0’;
writeAcc<= ’0’;
rbe<= ‘X’;
whens_decode  =>
IRload<= ‘0’;
PCload<= ‘0’;
Jmpmux<= ‘0’;
opfetch<= ‘0’;
ALUSel<= “XX”;
Asel<= “XX”;
Oload<= ‘0’;
we<= ‘0’;
writeAcc<= ’0’;
rbe<= ‘X’;
whens_jnz   =>
IRload<= ‘0’;
PCload<= ‘Aeq0’;
Jmpmux<= ‘1’;
opfetch<= ‘0’;
ALUSel<= “XX”;
Asel<= “XX”;
Oload<= ‘0’;
we<= ‘0’;
writeAcc<= ’0’;
rbe<= ‘X’;
whens_in    =>
IRload<= ‘0’;
PCload<= ‘0’;
Jmpmux<= ‘0’;
opfetch<= ‘0’;
ALUSel<= “00”;
Asel<= “01”;
Oload<= ‘0’;
we<= ‘1’;
writeAcc<= ’0’;
rbe<= ‘1’;
whens_add   =>
IRload<= ‘0’;
PCload<= ‘0’;
Jmpmux<= ‘0’;
opfetch<= ‘0’;
ALUSel<= “01”;
Asel<= “11”;
Oload<= ‘0’;
we<= ‘0’;
writeAcc<= ’1’;
rbe<= ‘1’;
whens_sub  =>
IRload<= ‘0’;
PCload<= ‘0’;
Jmpmux<= ‘0’;
opfetch<= ‘0’;
ALUSel<= “11”;
Asel<= “11”;
Oload<= ‘0’;
we<= ‘0’;
writeAcc<= ’1’;
rbe<= ‘1’;
whens_store  =>
IRload<= ‘0’;
PCload<= ‘0’;
Jmpmux<= ‘0’;
opfetch<= ‘0’;
ALUSel<= “00”;
Asel<= “11”;
Oload<= ‘0’;
we<= ‘1’;
writeAcc<= ’0’;
rbe<= ‘0’;
whens_load  =>
IRload<= ‘0’;
PCload<= ‘0’;
Jmpmux<= ‘0’;
opfetch<= ‘0’;
ALUSel<= “00”;
Asel<= “00”;
Oload<= ‘0’;
we<= ‘0’;
writeAcc<= ’1’;
rbe<= ‘1’;
whens_mov   =>
IRload<= ‘0’;
PCload<= ‘0’;
Jmpmux<= ‘0’;
opfetch<= ‘0’;
ALUSel<= “01”;
Asel<= “10”;
Oload<= ‘0’;
we<= ‘0’;
writeAcc<= ’1’;
rbe<= ‘0’;

whens_out =>
IRload<= ‘0’;
PCload<= ‘0’;
Jmpmux<= ‘0’;
opfetch<= ‘0’;
ALUSel<= “XX”;
Asel<= “XX”;
Oload<= ‘1’;
we<= ‘0’;
rbe<= ‘1’;

whens_others   =>
IRload<= ‘0’;
PCload<= ‘0’;
Jmpmux<= ‘0’;
opfetch<= ‘0’;
```



```
ALUSel<= “XX”;
Asel<= “XX”;
Oload<= ‘0’;
we<= ‘0’;
writeAcc<= ’0’;
rbe<= ‘X’;
end case;
end process;
end imp;
Data Path:-
libraryieee;
use ieee.std_logic_1164.all;
useieee.std_logic_unsigned.all;
useieee.numeric_std.all;

entitydatapath is port (
clk:                    in std_logic;
reset:                  in std_logic;
input:                  in std_logic_vector(7
downto 0);
ouput:                  out std_logic_vector(7
downto 0);
  Aeq0:                 out std_logic;
IROut:                  out std_logic_vector(7
downto 5);
ALUSel:                 in std_logic_vector(1
downto 0);
Asel:                   in std_logic_vector(1
downto 0);
writeAcc:               in std_logic;
IRload:                 in std_logic;
PCload:                 in std_logic;
Oload:                  in std_logic;
jmpmux:                 in std_logic;
PCload:                 in std_logic;
opfetch:                in std_logic;
we:                     in std_logic;
rbe:                    in std_logic;
enddatapath;

architecture imp of datapath is
signaldp_ROMData, dp_IR, dp_IR2, dp_ALU_Out:
std_logic_vector(7 downto 0);
signaldp_PC, dp_PCnext, dp_Adder_out:
std_logic_vector(7 downto 0);
signaldp_regfile_i, dp_regfile_B: std_logic_vector(7
downto 0);
signal dp_mux4_out: std_logic_vector(7 downto 0);
signal dp_mux2_out: std_logic_vector(3 downto 0);
signal dp_mux2_out8: std_logic_vector(7 downto 0);
signalf_unsigned_overflow: std_logic;
signalsub_jmp: std_logic;
begin

Aeq<= dp_regfile_A(0) or dp_regfile_A(1) or
dp_regfile_A(2) or dp_regfile_A(3) or dp_regfile_A(4) or
dp_regfile_A(5) or
dp_regfile_A(6) or dp_regfile_A(7);
dp_IR2 <=”000” &dp_IR(4 downto 0);
Bus_select: entity work.mux4 port map (Asel,
dp_regfile_B, Input, dp_IR2, dp_regfile_A, dp_mux4_out);
instruction_register: entity work.IR port map (clk, reset,
IRload, dp_ROMData, dp_IR);
ProgramCounter: entity work.PC port map (clk, reset,
PCload, dp_PCnext, dp_PC);
PC_mux: entity work.mux2 port map (jmpmux, “0001”,
dp_IR(3 downto 0), dp_mux_out);
dp_mux2_out8 <= “0000” & dp_mux2_out;
Sub_jmp<= jmpMux and dp_IR(4);
Adder_8_bit: entity work.Addsub8_pc port map (dp_PC,
dp_PCnext, dp_mux2_out8, sub_jmp);
ProgramMemory: entity work.rom_256_8 port map
(opfetch, dp_PC, dp_ROMData);
RegisterFile: entity work.regfile port map (clk, reset, we
,writeAcc, dp_IR(4 downto 0), dp_ALU_Out,
rbe, dp_regfile_A, dp_regfile_B);
ALU8: entity work.ALU port map (ALUSel,
dp_mux4_out, dp_regfile_B, dp_ALU_Out,
f_unsigned_overflow);
OutputRegister: entity work.OReg port map (clk, reset,
Oload, dp_regfile_B, output);
IROut<= dp_IR(7 downto 5);
end imp;
Register File:-
entityregfile is port (
clk:                    in std_logic;
reset:                  in std_logic;
we:                     in std_logic;
writeAcc:               in std_logic;
Adr:                    in std_logic_vector(4
downto 0);
      D:                in std_logic_vector(7
downto 0);
rbe:                    in std_logic;
portA:                  out std_logic_vector(7
downto 0);
portB:                  out std_logic_vector(7
downto 0));
endregfile;

architecture  imp of regfile is
subtypereg is std_logic_vector(7 downto 0);
typeregArray is array(0 to 31) of reg;
signal RF: regArray;
begin

WritePort: process (clk, reset)
begin
if (clk’event and clk=’1’)    then
if (reset=’1’)  then
RF(0) <= (others => ‘0’);
RF(1) <= (others => ‘0’);
RF(2) <= (others => ‘0’);
RF(3) <= (others => ‘0’);
elsif (we=’1’)      then
RF(conv_integer(Adr))  <=D;
elsif (writeAcc=’1’)   then
RF (0)  <= D;
```



```
end if;
end if;
end process;

ReadPortB:  Process(rbe, Adr)
begin
if (rbe='1')   then
portB<=  RF(conv_integer (Adr));
else
PortB<=  (others  =>  'X');
end if;
end process;
ReadPortA: PortA<=  RF(0);
end imp;
Arithmetic Logic Unit:-
libraryieee;
use ieee.std_logic_1164.all;
useieee.std_logic_unsigned.all;
useieee.numeric_std.all;
entity ALU is port (
S:                              in std_logic_vector(1 downto 0);
A, B:                           in std_logic_vector(7 downto 0);
F:                                 out std_logic_vector(7 downto
0);
Unsigned_overflow:    out std_logic);
end ALU;

architecture imp of ALU is
signal X, Y:           std_logic_vector(7downto 0):
signal C:              std_logic_vector(7 downto 0);
begin
C(0)  <=  S(1);
Y(0)  <=  s(1) xor (S(0) and B(0));
Y(1)  <=  s(1) xor (S(0) and B(1));
Y(2)  <=  s(1) xor (S(0) and B(2));
Y(3)  <=  s(1) xor (S(0) and B(3));
Y(4)  <=  s(1) xor (S(0) and B(4));
Y(5)  <=  s(1) xor (S(0) and B(5));
Y(6)  <=  s(1) xor (S(0) and B(6));
Y(7)  <=  s(1) xor (S(0) and B(7));
        U0 entity work.FA port map (C(0), C(1), A(0), Y(0),
F(0));
        U1 entity work.FA port map (C(1), C(2), A(1), Y(1),
F(1));
U2 entity work.FA port map (C(2), C(3), A(2), Y(2), F(2));
  U3 entity work.FA port map (C(3), C(4), A(3), Y(3), F(3));
 U4 entity work.FA port map (C(4), C(5), A(4), Y(4), F(4));
 U5 entity work.FA port map (C(5), C(6), A(4), Y(5), F(5));
 U6 entity work.FA port map (C(6), C(7), A(4), Y(6), F(6));
U7 entity work.FA port map (C(0), unsigned_overflow, A(4),
Y(7), F(7));
end imp;
```

H. Program Memory:-

```
entity rom_256_8 is port (
cs:                              in std_logic;
addr:                           in std_logic_vector(7 downto 0);
data:                              out std_logic_vector(7 downto
0));
end rom_256_8;

architecture imp of rom_256_8 is
subtype cell is std_logic_vector (7 downto 0);
typerom_type is array (0 to 255) of cell;
constant ROM: rom_type :=(
Inp& B,
movi& "00001",
sta& C,
movi& "00000",
sta& D,
lda& D,
adda& B,
sta& D,
lda& B,
suba& C,
sta& B,
jnz& "10111",
outp& D,
others => (others => '0')
);

begin
process(cs)
begin
if(cs  = '1')  then
data<= ROM(conv_integer (addr));
else
data<= (others => 'Z');
end if;
end process;
end imp;
Processor:-
libraryieee;
use ieee.std_logic_1164.all;
useieee.std_logic_unsigned.all;
useieee.numeric_std.all;

entityup_abs is port(
clk:             in std_logic;
reset:          in std_logic;
input:          in std_logic_vector(7 downto 0);
output:        in std_logic_vector(7 downto 0));
end  up_abs;

architecture imp of up_abs is
Signal IR:  std_logic_vecter(7 downto 5);
Signal ALUSel:  std_logic_vecter(1 downto 0);
Signal ASel:  std_logic_vecter(1 downto 0);
Signal writeAcc:  std_logic;
Signal Aeq0, IRload, PC load, opfetch, jmpmux, Oload, we,
rbe :std_logic;
begin
ControlUnit :   controller port map(clk ,   reset, aeq0, IR,
ALUSel, Asel, writeAcc, IRload,
PCload, Oload, jmpMux, opfetch, we, rbe);
Datapath8 :datapath  port map(clk ,  reset, aeq0, IR, ALUSel,
Asel, writeAcc, IRload, PCload,
Oload, jmpMux, opfetch, we, rbe);
```



```
end imp;
Test Bench:-
entity test is port (
--input_testbench: in std _logic_ vecter (7 downto 0);
Output_testbench: out std_logic_vecter(7 downto 0));
end entity:

architecture imp of test is
signalinput_signals: std_logic _vecter (7 down 0):=”00000100”;
signaloutput_signals: std_logic _vecter (7 down 0);
signalclkin: std_logic :=’0’;
signal reset :std_logic :=’1’;
begin
process(clkin)
begin
clkin<= ‘not’ clkin after 5 ns:
end process;
process(reset)
begin
reset<= ‘1’ reset after 30 ns ;
end process:
processor: entity work.up_absportmap(reset, clkin, input_signals, output_signals);
end imp;
```

IV. RESULT

In this paper we done a design of a single cycle microprocessor using a VHDL. The final outcome of the project was that all function and instruction working properly. We done a five instruction of microprocessor using Modelsim software to simulate a code of microprocessor

V. CONCLUSIONS

The In this paper, we done a part of successfully studied about the single cycle microprocessor fetching, load, store and jump instruction using all part of processor like program counter, instruction memory, data memory, arithmetic and logic unit, adder, register file and bit manipulation. In this we done a part of history of computer, instruction set architecture of different type of processor and effect of performance on processor. The design was verified through exhaustive simulations. The processor achieves higher performance, lower area and lower power dissipation. This processor can be used as a systolic core to perform mathematical functions and we use this for load store a data.The final outcome of the project was that all the instructions and function properly.

REFERENCE

[1] Anshulkumar“Computer Architecture”.
[2] 8 Bit Microprocessor Design Using VHDL.
[3] R. Gaonkar“Microprocessor architecture programming and applications with 8085”.
[4] Douglas V. Hall, SSSP Rao.“Microprocessors and its Interfacing 3rd Edition 3rd EditionAuthor(s)”.
[5] K. M. Bhurchandi, A. K. Ray “Advanced Microprocessor andPeripherals 3rd Edition 3rd EditionAuthor(s)”.
[6] Kenneth Ayala “The 8051 Microcontroller (With CD) 3 EditionAuthor(s)”.
[7] Kenneth Ayala “The 8051 Microcontroller & Embedded Systems Using Assembly and C (With CD) 1st Edition Author(s)”.
[8] Muhammad Ali Mazidi, Janice GillispieMazidi, RolinD.Mckinlay “The 8051 Microcontroller and Embedded Systems: Using Assembly and C 2 EditionAuthor(s)”.
[9] Sampath K. Venkatesh “8051 Microcontroller & Embedded System 1st Edition Author(s)”.
[10] A. P. Godse, D. A. Godse “Microcontrollers and RISC Architecture forAnna University of Technology 2nd EditionAuthor(s)”.

