AVIT
AARUPADAI VEEDU INSTITUTE OF TECHNOLOGY

VINAYAKA MISSION'S RESEARCH FOUNDATION
(Deemed to be University under section 3 of the UGC Act 1956)

Accredited by NAAC

Approved by AICTE

EMBEDDED SYSTEM LAB – II

LAB MANUAL

HOD-ECE

LIST OF EXPERIMENTS

1. ATMEL CPLDs – Prochip designer a) Schematic entry b) VHDL entry
2. AT40K FPGA series – synthesis – design – simulation of application programs
3. Xilinx EDA design tools – device programming –PROM programming
4. Programming & Simulation in GUI Simulators /Tools
5. Code compressor studio for embedded DSP using Texas tool kit
6. Programming ARM processor :ARM7 / ARM9/ARM Cortex, Study on incircuit Emulators, cross compilers , debuggers
7. IPCORE usage in VOIP through SoC2 tools
8. Programming with Rasberry Pi Microcontroller Board :Study on incircuit Emulators, crosscompilers, debuggers
9. Third party tools for embedded java and embedded C++ applications through cadence tools

EXP No. 01 - ATMEL CPLDS PROCHIP DESIGNER A) SCHEMATIC ENTRY B) VHDL ENTRY

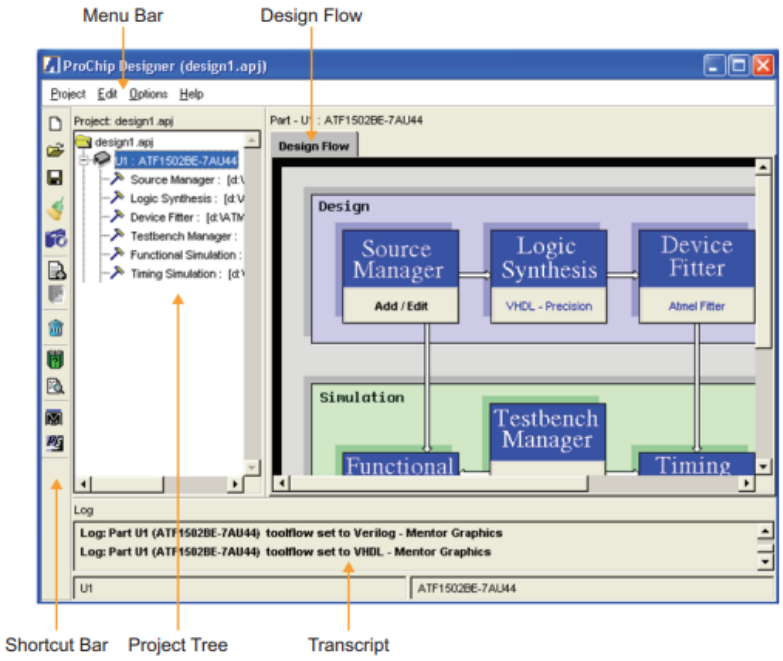
Aim

To formulate the design flow of a.) Prochip designer b.) Schematic Entry c.) VHDL entry

Procedure/Methodology

a.)Prochip Designer

The following are the flowgraph of Prochip Designer



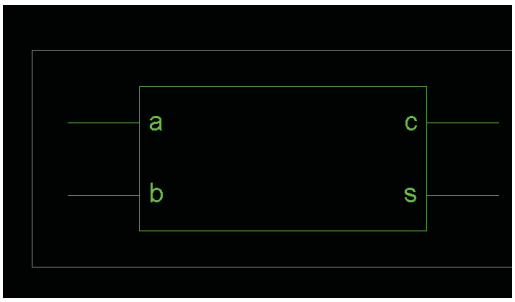
Design Flow Stage	Description
Source Manager	To specify new or existing design file(s). Launches the appropriate design entry tool for editing based on the selected tool-flow.
Logic Synthesis	To synthesize the design file(s) to generate the appropriate netlist file.
Device Fitter	To fit the netlist into the target device with the user selected fitter options.
Testbench Manager	To specify new or existing testbench file(s). Launches the appropriate testbench entry tool for editing based on the selected tool-flow.
Functional Simulation	To perform functional simulation on the design file(s).
Timing Simulation	To perform timing simulation on the output netlist generated by the fitter after fitting.
Atmel-ISP	To program the target ATF15xx CPLD through JTAG-ISP.

Schematic & VHDL Entry

Program:

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

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use  UNISIM.VComponents.all;

entity ha is
   Port ( a : in STD_LOGIC;b :
         in  STD_LOGIC;
         s : out STD_LOGIC;c :
         out  STD_LOGIC);
end ha;

architecture Behavioral of ha is
begin
s <= a xor b;c
<= a and b;
end Behavioral;
```

Output



Result

Thus the formulation of the design flow of a.) Prochip designer b.) Schematic Entry c.) VHDL entry is done.

EXP No. 02 - AT40K FPGA SERIES - SIMULATION OF APPLICATION PROGRAMS

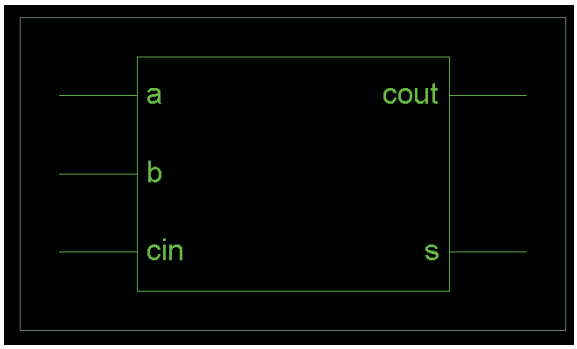
Aim

To formulate the design principle of AT40K FPGA SERIES with its application programmes

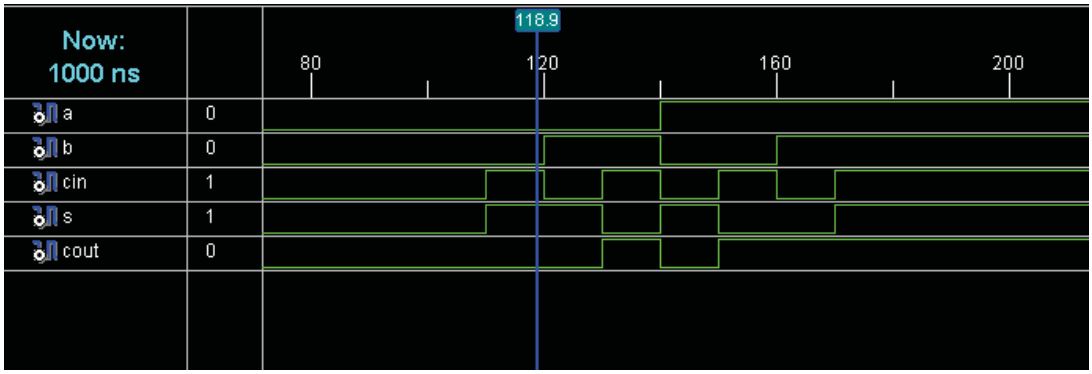
Procedure/program

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

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use  UNISIM.VComponents.all;

entity fa is
   Port ( a : in STD_LOGIC;b :
         in  STD_LOGIC;
         cin : in  STD_LOGIC;s :
         out  STD_LOGIC;
         cout : out  STD_LOGIC);
end fa;

architecture Behavioral of fa is

begin
s <= (a xor b) xor cin;
cout <= (a and b) or (b and cin) or (a and cin);end

Behavioral;
```

Model View



Simulation View



Result

To formulate the AT40K FPGA SERIES with its application programmes are executed

EXP No -3 - XILINX EDA DESIGN TOOLS – DEVICE PROGRAMMING –PROM PROGRAMMING

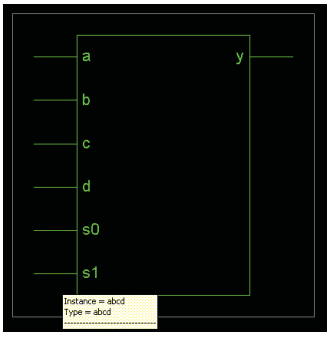
Aim

To formulate a program for PROM device programming with Xilinx EDA design tools

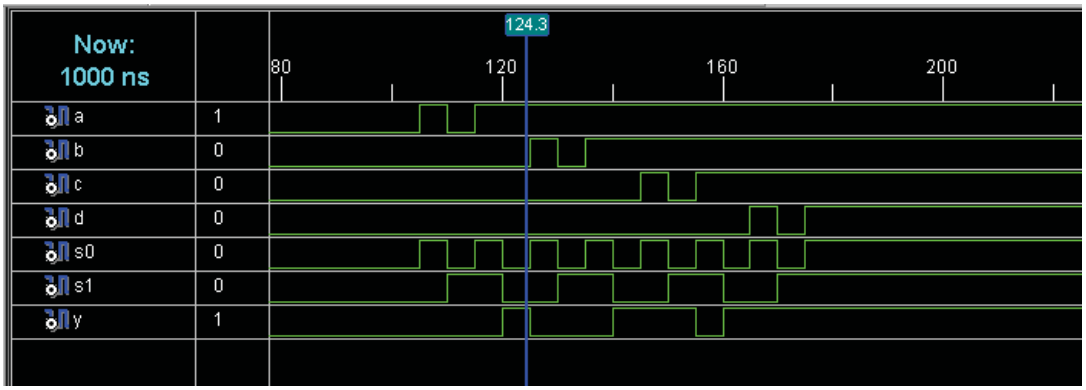
Procedure/Program

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

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use  UNISIM.VComponents.all;

entity abcd is
   Port ( a : in STD_LOGIC;b :
        in  STD_LOGIC; c : in
        STD_LOGIC;
        d : in STD_LOGIC;s0 :
        in  STD_LOGIC;s1 : in
        STD_LOGIC;
        y : out  STD_LOGIC);
end abcd;

architecture Behavioral of abcd is

begin
y <= a when s0 ='0' and s1 = '0' else
b when s0 ='0' and s1 = '1' else c
when s0 ='1' and s1 = '0' elsed;

end Behavioral;
```

Model View



Simulation View



Result

Thus the program for PROM device programming with Xilinx EDA design tools are executed

EXP No -4 PROGRAMMING & SIMULATION IN GUI SIMULATORS /TOOLS

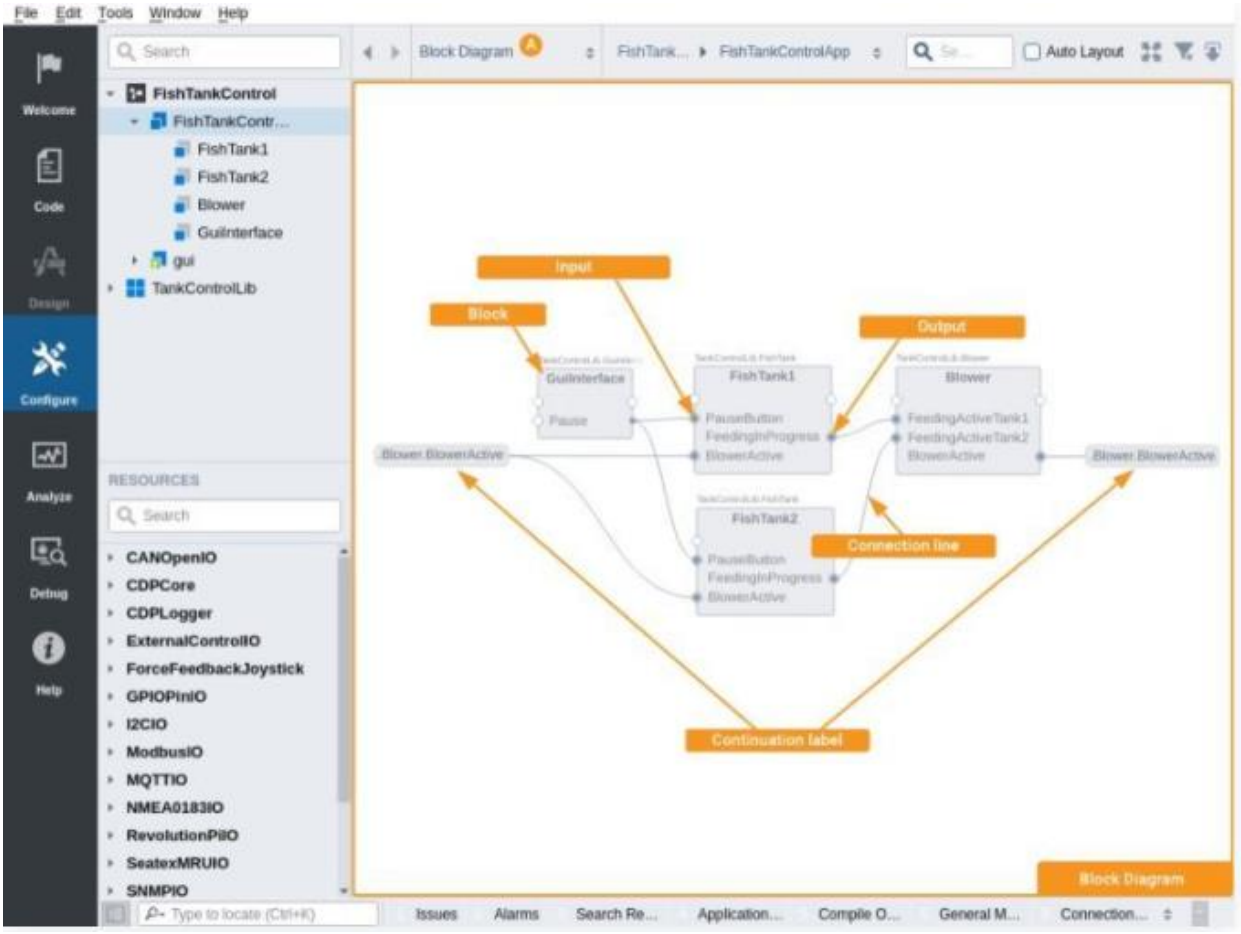
Aim

To formulate the programming code and Simulation in GUI Simulators/Tools

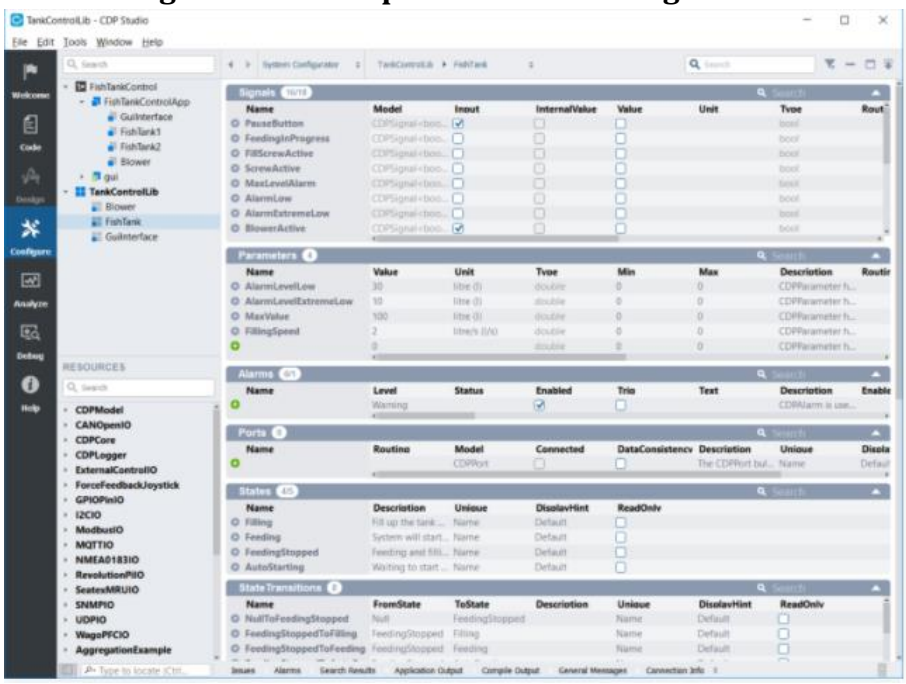
Procedure/Program

Here we use new visual programming editor – the Block Diagram. The editor allows you to create systems by adding new components (function blocks) with drag-and-drop and connect the blocks together to create a working system, all graphically.

The Block Diagram shows a visual representation of the system and gives a very good system overview.



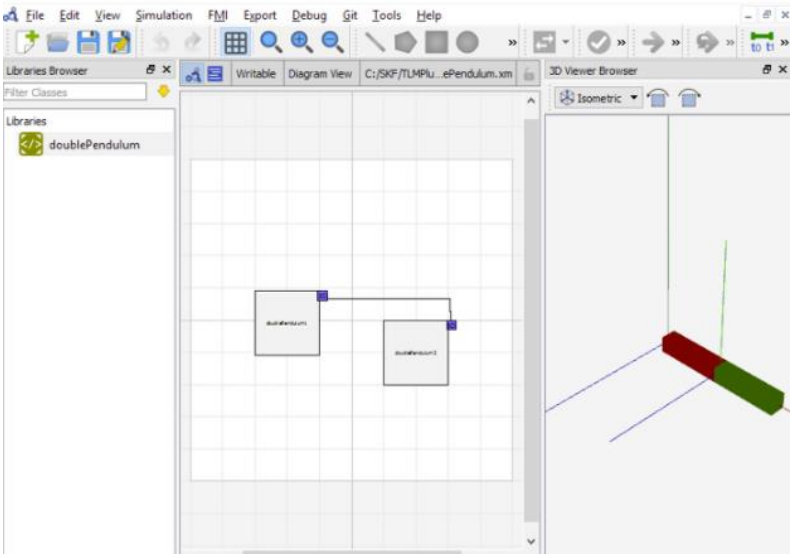
Model Editor – working with C++ components from Configure Mode



Program Code

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

entity encod1 is
    Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);q :
          out  STD_LOGIC_VECTOR (1 downto 0));
end encod1;

architecture Behavioral of encod1 is

begin
q<="00" when a="0001" else
  "01"   when   a="0010"   else
  "10"   when   a="0100"   else
  "11";


end Behavioral;
```

Output



Result

Thus the Programming & Simulation in GUI Simulators /Tools has been simulated

EXP No -5 CODE COMPRESSOR STUDIO FOR EMBEDDED DSP USING TEXAS TOOL KIT

Aim

To execute the code compressor file for embedded DSP using Texas tool Kit

Procedure/Methodology

Code for File Compressor

```
#include <stdio.h>
#include "volume.h"
/* Global declarations */
int inp_buffer[BUFSIZE]; /* processing data buffers */
int out_buffer[BUFSIZE];
int gain = MINGAIN; /* volume control variable */
unsigned int processingLoad = BASELOAD; /* processing load */
/* Functions */
extern void load(unsigned int loadValue);
static int processing(int *input, int *output);
static void dataIO(void);
/* ======== main ======== */
void main()
{
 int *input = &inp_buffer[0];
 int *output = &out_buffer[0];
 puts("volume example started\n");
```

Reviewing the Source Code

Testing Algorithms and Data from a File 4-5

```
 /* loop forever */
 while(TRUE)
 {
 /* Read using a Probe Point connected to a host file. */
 dataIO();

 /* apply gain */
 processing(input, output);
 }
}
/* ======== processing ======== *
 * FUNCTION: apply signal processing transform to input signal.
 * PARAMETERS: address of input and output buffers.
 * RETURN VALUE: TRUE. */
static int processing(int *input, int *output)
{
 int size = BUFSIZE;
 while(size--){
 *output++ = *input++ * gain;
 }

 /* additional processing load */
 load(processingLoad);

 return(TRUE);
```

```
}
/* ======== dataIO ======== *
 * FUNCTION: read input signal and write output signal.
 * PARAMETERS: none.
 * RETURN VALUE: none. */
static void dataIO()
{
 /* do data I/O */
 return;
```

Executing the Program

1) Choose Project→Build or click the (Incremental Build) toolbar button.

2) Choose File→Load Program. Select hostio.out and click Open.

3) Choose Tools→RTDX.

4) Click Configure in the RTDX area of the window. In the General Settings tab of the RTDX Properties dialog, select Continuous RTDX mode. Then, click OK.

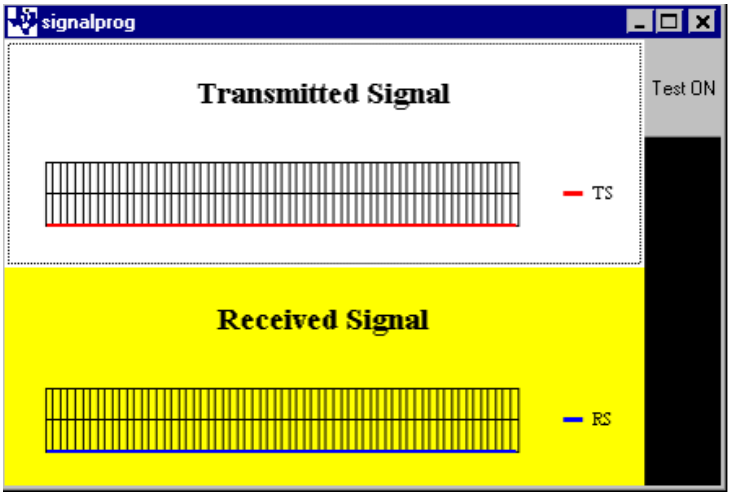
5) Change RTDX Disable to RTDX Enable in the RTDX area. This changes the Configure button to Diagnostics.

6) Choose Tools→DSP/BIOS→Message Log. Right-click on the Message Log area and choose Property Page from the pop-up window. Select trace as the name of the log to monitor and click OK.

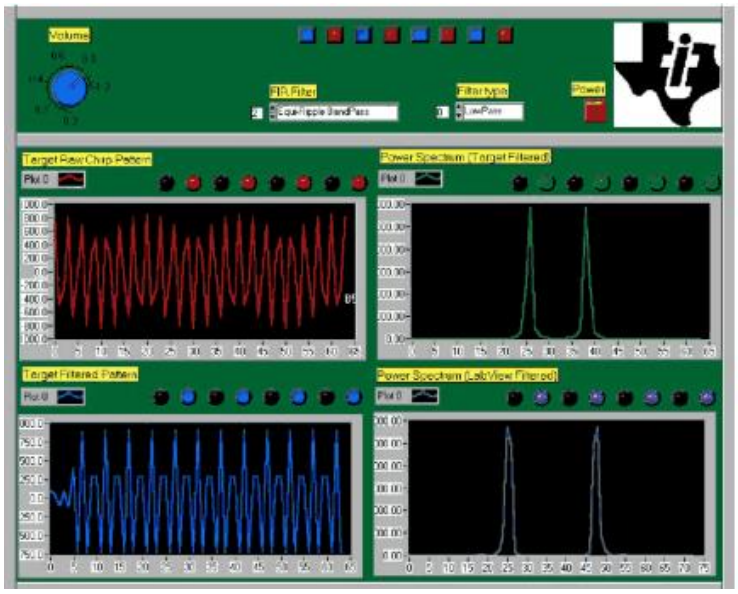
7) Choose Debug→Run or click the (Run) toolbar button.

8) Using Windows Explorer, run signalprog.exe and slider.exe. You see these two Visual Basic applications.

Output



Compressed Output



Result

Thus the code compressor file for embedded DSP using Texas tool Kit is executed

EXP No -6 PROGRAMMING ARM PROCESSOR : ARM7 / ARM9/ARM CORTEX & STUDY ON IN-CIRCUIT EMULATORS, CROSS COMPILERS & DEBUGGERS

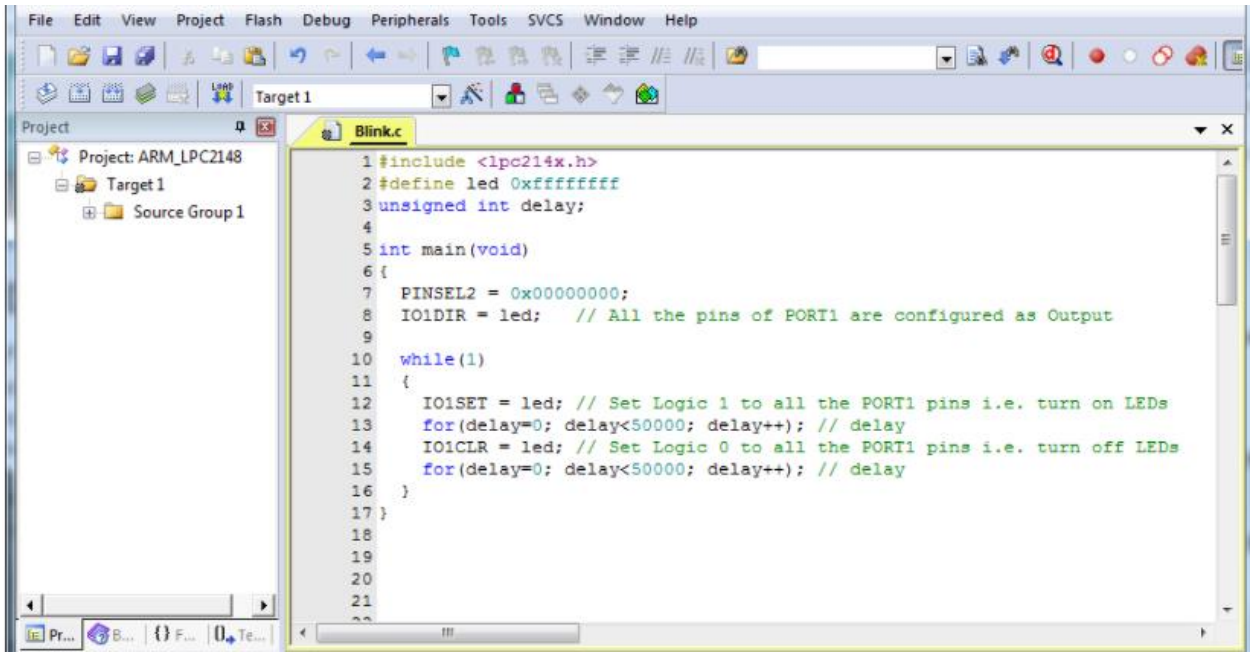
Aim

To formulate the programming structure of ARM processor

Procedure/Methodology

ARM7/ARM9/ARM Cortex

Sample layout for program



Code for Printing a 32-bit Number in Hex

Step 1: standard initialization:

.globl _start

_start:

ldr r4, =0x101f1000

@ ASCII codes stored

@ at [r4] get printed

Step 2: store some number to r0.

• Note: the mov instruction does not allow us to use arbitrary

32-bit constants, we can only use 8-bit constants. (Why?)

```
@ set r0 := 0x12ad730f
mov r0, #0x12
lsl r0, r0, #8
add r0, r0, #0xad
lsl r0, r0, #8
add r0, r0, #0x73
lsl r0, r0, #8
add r0, r0, #0x0f
@ set r0 := 0x12ad730f
ldr r0, =0x12ad730f
mov r2, #28
my_loop:
cmp r2, #0
blt my_exit
lsr r1, r0, r2
and r1, r1, #0x0000000f
cmp r1, #10
addlt r1, r1, #48
addge r1, r1, #55
str r1, [r4]
```

```
sub r2, r2, #4

b my_loop

my_exit: @do infinite loop at the end

b my_exit
 mov r2, #28

my_loop:

cmp r2, #0

blt my_exit

lsr r1, r0, r2

and r1, r1, #0x0000000f

cmp r1, #10

addlt r1, r1, #48

addge r1, r1, #55

str r1, [r4]

sub r2, r2, #4

b my_loop

my_exit: @do infinite loop at the end

b my_exit

mov r2, #28

my_loop:

cmp r2, #0

blt my_exit

lsr r1, r0, r2

and r1, r1, #0x0000000f

cmp r1, #10
```

```
addlt r1, r1, #48
addge r1, r1, #55
str r1, [r4]
sub r2, r2, #4
b my_loop
my_exit: @do infinite loop at the end
b my_exit
mov r2, #28
my_loop:
cmp r2, #0
blt my_exit
lsr r1, r0, r2
and r1, r1, #0x0000000f
cmp r1, #10
addlt r1, r1, #48
addge r1, r1, #55
str r1, [r4]
sub r2, r2, #4
b my_loop
my_exit: @do infinite loop at the end
b my_exit
CODE END
```

A Short glimpse of essentials in ARM processors

Compiler

A compiler is a computer program (or a set of programs) that transforms the source code written in a programming language (the source language) into another computer language (normally binary format). The most common reason for conversion is to create an executable program. The name "compiler" is primarily used for programs that translate the source code from a highlevel programming language to a low-level language (e.g., assembly language or machine code).

Cross-Compiler

If the compiled program can run on a computer having different CPU or operating system than the computer on which the compiler compiled the program, then that compiler is known as a cross-compiler.

Decompiler

A program that can translate a program from a low-level language to a high-level language is called a decompiler.

Language Converter

A program that translates programs written in different high-level languages is normally called a language translator, source to source translator, or language converter.

A compiler is likely to perform the following operations –

- Preprocessing
- Parsing
- Semantic Analysis (Syntax-directed translation)
- Code generation
- Code optimization

Assemblers

An assembler is a program that takes basic computer instructions (called as assembly language) and converts them into a pattern of bits that the computer's processor can use to perform its basic operations. An assembler creates object code by translating assembly instruction mnemonics into opcodes, resolving symbolic names to memory locations.

Debugging Tools in an Embedded System

Debugging is a methodical process to find and reduce the number of bugs in a computer program or a piece of electronic hardware, so that it works as expected. Debugging is difficult when subsystems are tightly coupled, because a small change in one subsystem can create bugs in another. The debugging tools used in embedded systems differ greatly in terms of their development time and debugging features. We will discuss here the following debugging tools.

Functions of Simulators

- A simulator performs the following functions
- Defines the processor or processing device family as well as its various versions for the target system.
- Monitors the detailed information of a source code part with labels and symbolic arguments as the execution goes on for each single step.
- Provides the status of RAM and simulated ports of the target system for each single step execution.
- Monitors system response and determines throughput.
- Provides trace of the output of contents of program counter versus the processor registers.
- Provides the detailed meaning of the present command.
- Monitors the detailed information of the simulator commands as these are entered from the keyboard or selected from the menu.
- Supports the conditions (up to 8 or 16 or 32 conditions) and unconditional breakpoints.
- Provides breakpoints and the trace which are together the important testing and debugging tool.
- Facilitates synchronizing the internal peripherals and delays.

EXP No -7 IPCORE USAGE IN VOIP THROUGH SOC2 TOOLS

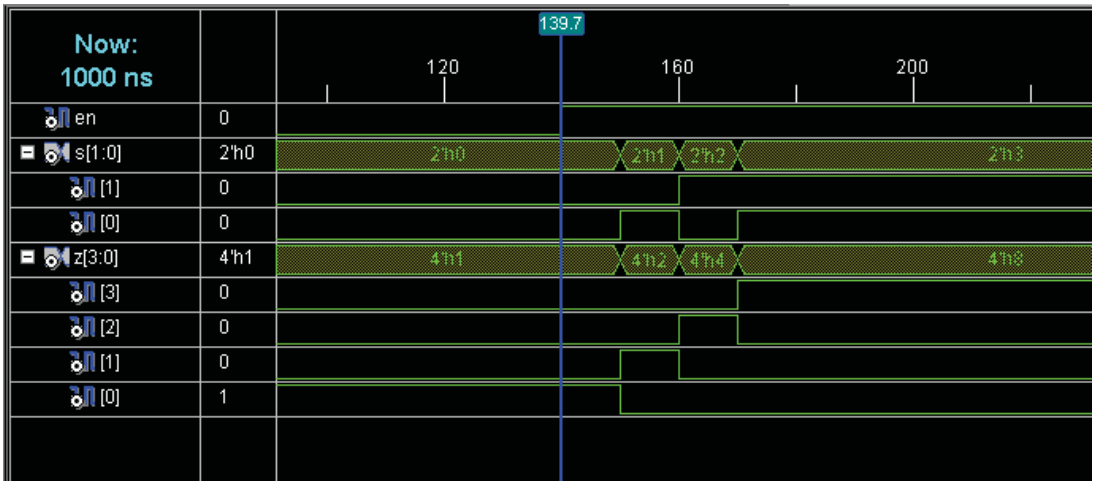
Aim

To code a program of Design a 2 to 4 Line Decoder that can be executed on SoC2 tools

Program Code

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

--library UNISIM;
--use UNISIM.VComponents.all;

entity decoder is
Port ( en : in STD_LOGIC;
s : in STD_LOGIC_VECTOR (1 downto 0);
z : out STD_LOGIC_VECTOR (3 downto 0)); end decoder;

architecture arch of decoder is signal p,q : STD_Logic; begin
p <= not s(1); q <= not s(0);

z(0) <= p and q; z(1) <= p and s(0);
z(2) <= q and s(1);
z(3) <= s(0) and s(1); end arch;
```

Output



Result

Thus the program of Design a 2 to 4 Line Decoder is executed using SoC2 tools.

EXP No -8 PROGRAMMING WITH RASBERRY PI MICROCONTROLLER BOARD

Aim

To formulate and execute the code of automatic cart using Rasberry Pi Microcontrollers

Program Code

```
# import sys
#
# sys.path.insert(0, '/home/pi/Ali Proj/Libraries')
import gspread
from time import sleep
from RPLCD.i2c import CharLCD
lcd = CharLCD('PCF8574', 0x27)
from escpos.printer import Serial
from datetime import datetime
now = datetime.now()
dt_string = now.strftime("%b/%d/%Y %H:%M:%S")
lcd.cursor_pos = (0, 0)
lcd.write_string("Initiallising...")
#locating the spread sheet json file
gc = gspread.service_account(filename='/home/pi/Ali Proj/Proj 4 Shoping Cart with
Thermal Printer/Shopping Cart on 20_4 LCD/shop-data-thermal-585dc7bffa1f.json')
#sheet name is to be passed
sh = gc.open("Shop Data for Thermal")
worksheet=sh.get_worksheet(0)
count=0
```

```
item_cost=0
totalCost=0
SNo=0
scode=""
qty=1
scodePrev=0
item_name=""
entryF=[]
""" 9600 Baud, 8N1, Flow Control Enabled """
p = Serial(devfile='/dev/serial0',
        baudrate=9600,
        bytesize=8,
        parity='N',
        stopbits=1,
        timeout=1.00,
        dsrdtr=True)
p.set(
   align="center",
   font="a",
   width=1,
   height=1,
   )
def print_receipt():
   p.text("\n")
   p.set(
```

```
        align="center",
        font="a",
        width=1,
        height=1,
      )
   #Printing the image
   p.image("/home/pi/Ali Proj/Proj 3 Interfacing thermal printer with
pi/CD_new_Logo_black.png",impl="bitImageColumn")
   #printing the initial data
   p.set(width=2,
      height=2,
      align="center",)
   p.text(" ===============\n")
   p.text("Tax Invoice\n")
   p.text(" ===============\n")
   p.set(width=1,
      height=1,
      align="left",)
   p.text("CIRCUIT DIGEST\n")
   p.text("AIRPORT ROAD\n")
   p.text("LOCATION : JAIPUR\n")
   p.text("TEL : 0141222585\n")
   p.text("GSTIN : 08AAMFT88558855\n")
   p.text("Bill No. : \n\n")
   p.text("DATE : ")
```

```
p.text(dt_string)
p.text("\n")
p.text("CASHIER : \n")
p.text(" ============================\n")
p.text("S.No     ITEM   QTY   PRICE\n")
p.text(" -------------------------------\n")
print(text_F)
p.text(text_F)
p.text(" -------------------------------\n")
p.set(
      # underline=0,
      align="right",
   )
p.text("     SUBTOTAL:  ")
p.text(totalCostS)
p.text("\n")
p.text("     DISCOUNT:  0\n")
p.text("     VAT @ 0%: 0\n")
p.text(" ============================\n")
p.set(align="center",
   )
p.text("    BILL TOTAL: ")
p.text(totalCostS)
p.text("\n")
p.text(" -------------------------\n")
```

```
    p.text("THANK YOU\n")
    p.set(width=2,
        height=2,
        align="center",)
    p.text(" ===============\n")
    p.text("Please scan\nto Pay\n")
    p.text(" ===============\n")
    p.set(
          align="center",
          font="a",
          width=1,
          height=1,
          density=2,
          invert=0,
          smooth=False,
          flip=False,
       )
    p.qr("9509957951@ybl",native=True,size=12)
    p.text("\n")
    p.barcode('123456', 'CODE39')
    #if your printer has paper cuting facility then you can use this function
    p.cut()
    print("prnting done")
lcd.cursor_pos = (0, 0)
lcd.write_string('Please Scan...')
```

```
while 1:
    try:
        scode=input("Scan the barcode")
        if scode=="8906128542687": #Bill Printing Barcode
            print("done shopping ")
            lcd.clear()
            print(*entryF)
            print("in string")
            print(len(entryF))
            text_F=" "
            for i in range(0,len(entryF)):
                text_F=text_F+entryF[i]
                i=i+1
            lcd.cursor_pos = (0, 0)
            lcd.write_string("Thanks for Shopping")
            lcd.cursor_pos = (1, 7)
            lcd.write_string("With Us")
            lcd.cursor_pos = (3, 0)
            lcd.write_string("Printing Invoice...")
            print_receipt()
        else:
            cell=worksheet.find(scode)
            print("found on R%sC%s"%(cell.row,cell.col))
            item_cost = worksheet.cell(cell.row, cell.col+2).value
            item_name = worksheet.cell(cell.row, cell.col+1).value
```

```
        lcd.clear()
        SNo=SNo+1
        entry = [SNo,item_name,qty,item_cost]
        entryS=str(entry)+'\n'
        print("New Item ",*entry)
        lcd.cursor_pos = (0, 2)
        lcd.write_string(str(SNo))
        lcd.cursor_pos = (0, 5)
        lcd.write_string("Item(s) added")
        lcd.cursor_pos = (1, 1)
        lcd.write_string(item_name)
        lcd.cursor_pos = (2, 5)
        lcd.write_string("of Rs.")
        lcd.cursor_pos = (2, 11)
        lcd.write_string(item_cost)
        item_cost=int(item_cost)
        totalCost=item_cost+totalCost
        lcd.cursor_pos = (3, 4)
        lcd.write_string("Cart Total")
        lcd.cursor_pos = (3, 15)
        lcd.write_string(str(totalCost))
        entryF.append(entryS)   #adding entry in Final Buffer
        sleep(2)
    except:
       print("Unknown Barcode or Item Not Registered")
```

```
lcd.clear()
lcd.cursor_pos = (0, 0)
lcd.write_string("Item Not Found...")
lcd.cursor_pos = (2, 0)
lcd.write_string("Scan Again...")
sleep(2)
```

Result

Thus the program of automatic cart has been formulated and executed via Raspberry Pi motherboard and the results are verified.

EXP No -9 THIRD PARTY TOOLS FOR EMBEDDED JAVA AND EMBEDDED C++ APPLICATIONS THROUGH CADENCE TOOLS

Aim

To Study the various third party tools for Embedded Java and Embedded C++

Materials

Embedded Tools in JAVA

It is designed to be used on systems with at least 32 MB of RAM, and can work on Linux ARM, x86, or Power ISA, and Windows XP and Windows XP Embedded architectures. Java ME embedded used to be based on the Connected Device Configuration subset of Java Platform, Micro Edition

Suitable language for embedded systems

For many embedded systems, C or C++ will be the best choices. In part, that's because they are “compiled” languages and extremely efficient. In compiled languages, the machine (or embedded device) directly translates the code, which means the language is fast and stable

C++ & Java

The main difference between C++ and Java is that C++ is only a compiled language while Java is both compiled and interpreted. The C++ compiler converts the source code into machine code and therefore, it is platform dependent.

Java Encapsulation

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. ... Therefore, it is also known as data hiding. To achieve encapsulation in Java − Declare the variables of a class as private.

Garbage Collection in Java

Garbage collection in Java is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. The garbage collector finds these unused objects and deletes them to free up memory.

Programming Embedded Systems

As mentioned earlier, Embedded Systems consists of both Hardware and Software. If we consider a simple Embedded System, the main Hardware Module is the Processor. The Processor is the heart of the Embedded System and it can be anything like a Microprocessor, Microcontroller, DSP, CPLD (Complex Programmable Logic Device) or an FPGA (Field Programmable Gated Array). All these devices have one thing in common: they are programmable i.e., we can write a program (which is the software part of the Embedded System) to define how the device actually works.

Embedded Software or Program allow Hardware to monitor external events (Inputs / Sensors) and control external devices (Outputs) accordingly. During this process, the program for an Embedded System may have to directly manipulate the internal architecture of the Embedded Hardware (usually the processor) such as Timers, Serial Communications Interface, Interrupt Handling, and I/O Ports etc.

From the above statement, it is clear that the Software part of an Embedded System is equally important as the Hardware part. There is no point in having advanced Hardware Components with poorly written programs (Software).

There are many programming languages that are used for Embedded Systems like Assembly (low-level Programming Language), C, C++, JAVA (high-level programming languages), Visual Basic, JAVA Script (Application level Programming Languages), etc.

In the process of making a better embedded system, the programming of the system plays a vital role and hence, the selection of the Programming Language is very important.

Factors for Selecting the Programming Language

The following are few factors that are to be considered while selecting the Programming Language for the development of Embedded Systems.

Size: The memory that the program occupies is very important as Embedded Processors like Microcontrollers have a very limited amount of ROM (Program Memory).

Speed: The programs must be very fast i.e., they must run as fast as possible. The hardware should not be slowed down due to a slow running software.

Portability: The same program can be compiled for different processors.

Ease of Implementation

Ease of Maintenance

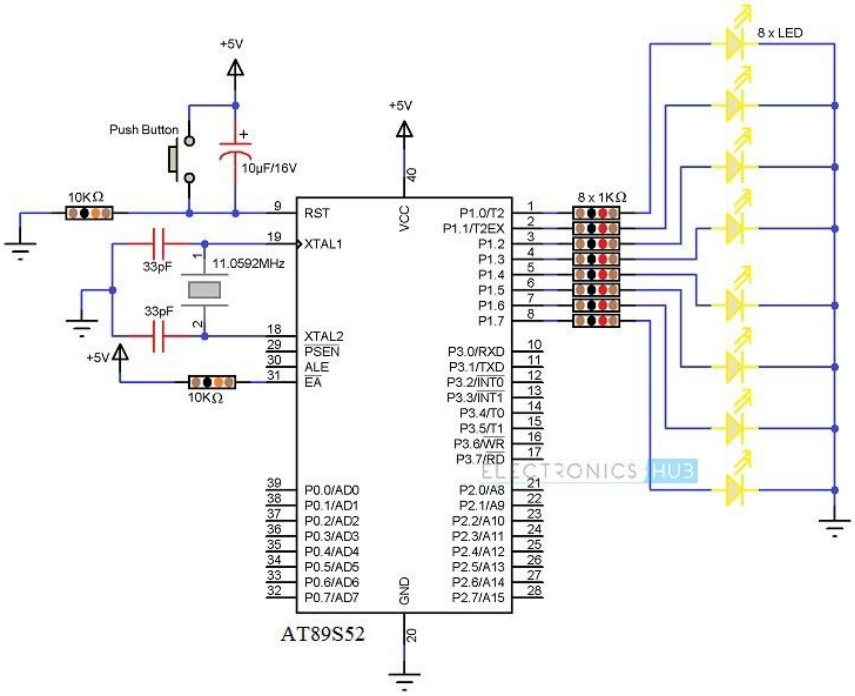
Readability

Earlier Embedded Systems were developed mainly using Assembly Language. Even though Assembly Language is closest to the actual machine code instructions and produces small size hex files, the lack of portability and high amount of resources (time and man power) spent on developing the code, made the Assembly Language difficult to work with.

There are other high-level programming languages that offered the above mentioned features but none were close to C Programming Language. Some of the benefits of using Embedded C as the main Programming Language:

- Significantly easy to write code in C
- Consumes less time when compared to Assembly
- Maintenance of code (modifications and updates) is very simple
- Make use of library functions to reduce the complexity of the main code
- You can easily port the code to other architecture with very little modifications
- Introduction to Embedded C Programming Language
- Before going in to the details of Embedded C Programming Language and basics of Embedded C Program, we will first talk about the C Programming Language.

Circuit design for blinking LED using microcontrollers using cadence tool



Sample Code

```
#include<reg51.h> // Preprocessor Directive
void delay (int); // Delay Function Declaration


void main(void) // Main Function
{
P1 = 0x00;
/* Making PORT1 pins LOW. All the LEDs are OFF.
 * (P1 is PORT1, as defined in reg51.h) */


while(1) // infinite loop
{
P1 = 0xFF; // Making PORT1 Pins HIGH i.e. LEDs are ON.
delay(1000);
/* Calling Delay function with Function parameter as 1000.
 * This will cause a delay of 1000mS i.e. 1 second */


P1 = 0x00; // Making PORT1 Pins LOW i.e. LEDs are OFF.
delay(1000);
}
}


void delay (int d) // Delay Function Definition
{
unsigned int i=0; // Local Variable. Accessible only in this function.

/* This following step is responsible for causing delay of 1000mS
 * (or as per the value entered while calling the delay function) */


for(; d>0; d–)
{
for(i=250; i>0; i – -);
for(i=248; i>0; i – -);
}
}
```

Result

Thus the circuit and code for blinking an LED using Cadence tools has been simulated and executed