# Jawaharlal Nehru Engineering College

## Laboratory Manual

# EMBEDDED SYSTEMS

**For**

**BE (ECT/IE)**

Manual made by

Prof. P.A.JADHAV
Prof. A.B.PENTEWAR

# Jawaharlal Nehru Engineering College

## Technical Document

This technical document is a series of Laboratory manuals of Electronics & Telecommunication and Industrial Electronics and is a certified document of Jawaharlal Nehru Engineering College. The care has been taken to make the document error free but still if any error is found kindly bring it to the notice of subject teacher and HOD.

Technical document No. ECT/IE/Techdoc/21stOctober2009/…….

Recommended by,

HOD

Approved by,

Principal

Copies:

1. Departmental Library
2. Laboratory
3. HOD
4. Principal

# PREFACE

It is my great pleasure to present this laboratory manual for final year engineering students for the subject of Embedded Systems keeping in view the vast coverage required for visualization of concepts of basic electronic circuits.

As a student, many of you may be wondering with some of the questions in your mind regarding the subject and exactly that has been tried to answer through this manual.

Faculty members are also advised that covering these aspects in initial stage itself will greatly relieve them in future, as much of the load will be taken care by the enthusiastic energies of the students, once they are conceptually clear. Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions.

Prof. A. B. Pentewar
Prof. Miss. P. A. Jadhav
Author

**SUBJECT INDEX**

## 1. DOs and DON'Ts in Laboratory:

1. Do not handle kit without reading the instructions/Instruction manuals.

2. Refer Help for debugging the program.

3. Go through Demos of Signal Processing tool box.

4. Strictly observe the instructions given by the teacher/Lab Instructor.

.

## 2 Instruction for Laboratory Teachers:

**1.** Lab work completed during prior session should be corrected during the next lab session.

2. Students should be guided and helped whenever they face difficulties.

3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

# ***Study experiment:1***

***Aim*** **:-**Study Of EDU-ARM-2148 Trainer kit.

***Apparatus*** **:-** EDU-ARM-2148 Trainer kit.

## ***Theory*** **:-**

The trainer kit contains following items:

1. EDU-ARM7-2148 board
2. Serial communication Cable
3. Power Supply Adaptor
4. SPJETs CD-ROM

**Power Supply Requirement:**

The power adaptor works with 230VAC. It produces approximately 9V DC, and the EDU-ARM-2148 uses on board regulators to provide 5V, 3.3V and 1.8V DC to all components on board.

**Connecting the system:**

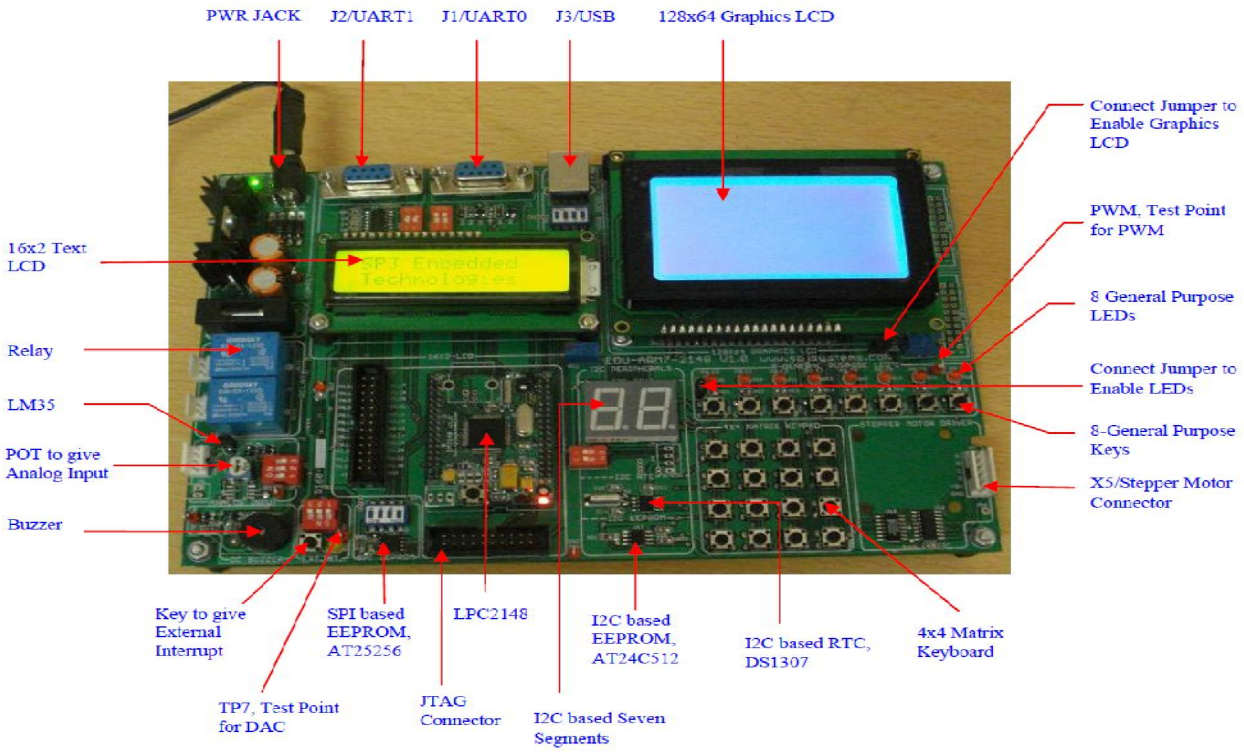
The serial communication cable supplied with the board should be used to connect the board to a PC running windows95/98/ME/2000/XP/Vista Operating system. Connect one end of the serial cable to UART0 of EDU-ARM7-2148 board and other end to PCs serial port.

**Powering ON:**

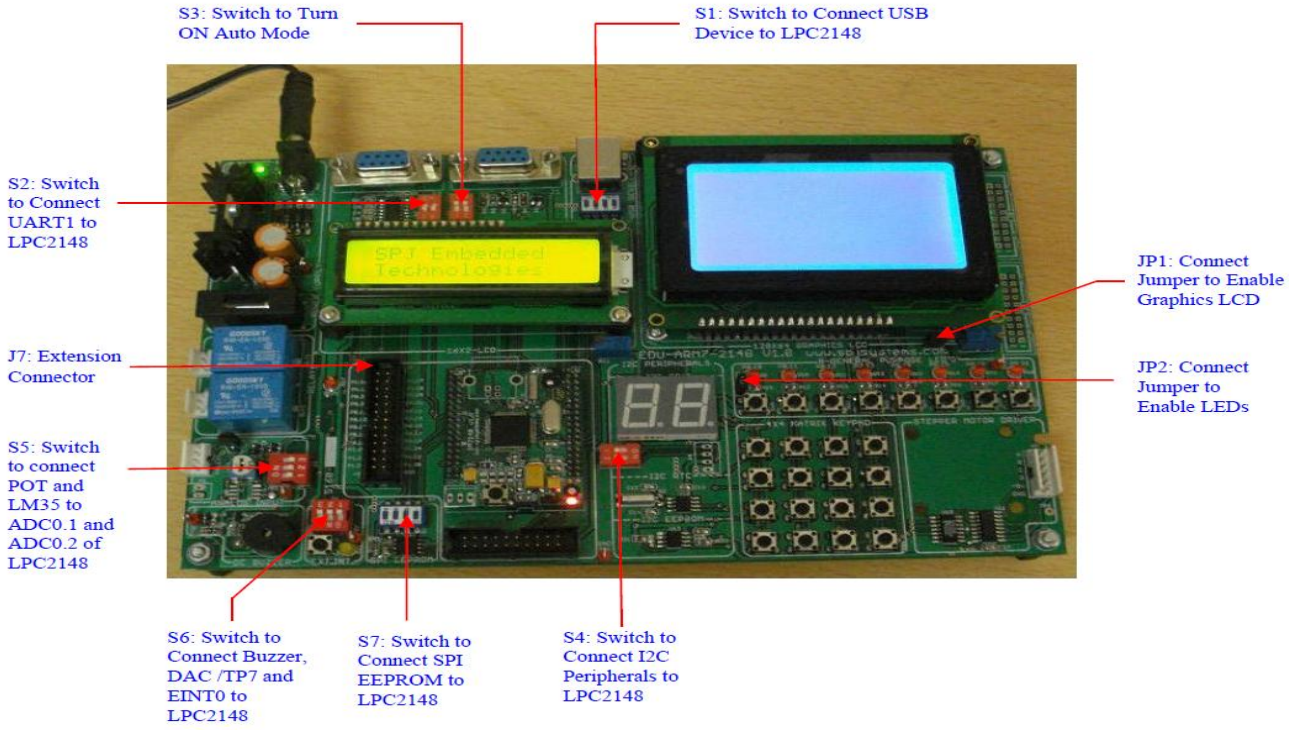
After connecting the serial communication cable as described above, you may insert the power adaptor output jack into the on-board power socket. Plug the power adaptor into 230 VAC mains outlet and turn it on. The power-on indication green LED will turn on.

## A. EDU-ARM7-2148 Block Diagram:

Below figure shows the locations of different components on the EDU-ARM7-2148 board.




Below figure shows the locations of different switches and jumpers on the EDU-ARM7-2148 board.



## B. Switches Details:

**S1:**

Turn ON this switch to connect USB device connector to USB lines of LPC2148.

**S2:**

Turn ON this switch to connect UART1 connector to UART1 lines (TxD1/P0.8 and RxD1/P0.9) of LPC2148.

**S3:**

Mode selection switch. The LPC21xx micro-controllers include on-chip flash for storing user program and non-volatile data. The LPC2148 have 512KBytes flash. This flash is In-System-Programmable (ISP). The LPC21xx micro-controllers have a built-in boot-load program. Upon power-on, this bootload program takes control; it passes control to the user program if pin P0.14 is HIGH and some other conditions are satisfied. Please refer to the LPC21xx data-sheet for further details. On the EDU-ARM7-2148 board, the P0.14 pin is made available on this S3 switch. Turn ON this switch to control the Mode (ISP mode or Run mode) by Flash Magic.

**S4:**

Turn ON this switch to connect Seven Segments, RTC (DS1307) and EEPROM (AT24C512) to I2C lines (SCL0/P0.2 and SDA0/P0.3) of LPC2148.

**S5:**

Turn ON this switch to connect POT and LM35 to ADC0.1/P0.28 and ADC0.2/P0.29 of LPC2148.

**S6:**

Turn ON this switch to connect Buzzer, DAC/TP7 and External Interrupt to ACOut/P0.25 and EINT0/P0.16 of LPC2148.

**S7:**

Turn ON this switch to connect SPI EEPROM (AT25256) to SPI lines (SCK0/P0.4, MISO0/P0.5, MOSI0/P0.6 and CS/P0.7) of LPC2148.

## C. Connector Details:

**UART0:**

This is a DB9 female connector, used for RS232 serial communication with the PC:

Pin 2 = UART0 RS232 TxD (output of mC)
Pin 3 = UART0 RS232 RxD (input to mC)
Pin 4 = RS232 DTR
Pin 5 = Ground
Pin 7 = RS232 RTS
All other pins of J1/UART0 are unused.

**UART1:**

This is a DB9 female connector, used for RS232 serial communication with the PC:

Pin 2 = UART1 RS232 TxD (output of mC)
Pin 3 = UART1 RS232 RxD (input to mC)
Pin 5 = Ground

**16x2 LCD:**

This is a 16 pin, single line connector, designed for connection to standard, text LCD modules. The pin/signal correspondence is designed to be matching with that required by such LCD modules.

Pin 1 = GND
Pin 2 = +5V

Pin 3 = Vlcd
Pin 4 = P1.25 (Used as RS of LCD)
Pin 5 = GND
Pin 6 = P1.24 (Used as EN of LCD)
Pin 7 to 10 = No Connection
Pin 11 = P0.15 (Used as D4 of LCD)
Pin 12 = P0.17 (Used as D5 of LCD)
Pin 13 = P0.22 (Used as D6 of LCD)
Pin 14 = P0.30 (Used as D7 of LCD)
Pin 15 = Backlighting
Pin 16 = GND

**128x64 Graphics LCD:**

This is a 20 pin, single line connector, designed for connection to standard, 128x64 Monochrome Graphics LCD modules. The pin/signal correspondence is designed to be matching with that required by such LCD modules.

Pin 1 = GND
Pin 2 = +5V
Pin 3 = Vlcd
Pin 4 = P1.25 (Used as RS of GLCD)
Pin 5 = P0.15 (Used as RW of GLCD)
Pin 6 = P1.24 (Used as EN of GLCD)
Pin 7 = P0.10 (Used as D0 of GLCD)
Pin 8 = P0.11 (Used as D1 of GLCD)
Pin 9 = P0.12 (Used as D2 of GLCD)
Pin 10 = P0.13 (Used as D3 of GLCD)
Pin 11 = P0.18 (Used as D4 of GLCD)
Pin 12 = P0.19 (Used as D5 of GLCD)
Pin 13 = P0.20 (Used as D6 of GLCD)
Pin 14 = P0.21 (Used as D7 of GLCD)
Pin 15 = P0.22 (Used as CS1 of GLCD)
Pin 16 = P0.30 (Used as CS2 of GLCD)
Pin 17 =
Pin 18 =
Pin 19 = +5V
Pin 20 = GND

**JTAG Connector:**

This standard 20 pin JTAG connector provides debugging support for the LPC21xx. This connector is mounted on top side of the board as shown in figure1. JTAG cables like SJT-S or SJT-U can be connected to this connector, while other end of the cable can be connected to PC COM port or USB port, respectively. Debugger software (like the debugger built into SCARM) allows JTAG based debugging. It is also possible to use third party JTAG based emulators /debuggers. The pin-out of JTAG Connector is given below:

| Pin | Signal name | | Pin | Signal name |
|---|---|---|---|---|
| 1 | 3.3V | | 2 | 3.3V |
| 3 | P1.31/NTRST | | 4 | GND |
| 5 | P1.28/TDI | | 6 | GND |
| 7 | P1.30/TMS | | 8 | GND |
| 9 | P1.29/TCK | | 10 | GND |
| 11 | P1.26/RTCK | | 12 | GND |
| 13 | P1.27/TDO | | 14 | GND |
| 15 | NRST | | 16 | GND |
| 17 | GND | | 18 | GND |
| 19 | GND | | 20 | GND |

**J7:**

This is 26 pin dual line headers. It brings out I/O and most of the pins of the LPC21xx micro-controller. Further, 5V and GND are also made available on these connectors. These connectors are intended for use to connect external peripherals.

The pin/signal details of J7 are as below:

| Pin | Signal name | | Pin | Signal name |
|---|---|---|---|---|
| 1 | P1.16 | | 2 | P1.17 |
| 3 | P1.18 | | 4 | P1.19 |
| 5 | P0.28 | | 6 | P0.29 |
| 7 | P0.30 | | 8 | P0.31 |
| 9 | P0.10 | | 10 | P0.11 |
| 11 | P0.12 | | 12 | P0.13 |
| 13 | P0.14 | | 14 | P0.15 |
| 15 | P0.16 | | 16 | P0.17 |
| 17 | P1.18 | | 18 | P1.19 |
| 19 | P1.25 | | 20 | P1.24 |
| 21 | P1.23 | | 22 | P1.22 |
| 23 | P1.21 | | 24 | P1.20 |
| 25 | 5V | | 26 | GND |

## ***Study experiment:2***

***Aim*** **:-** Study of IDE overview (Project creation, downloading & debugging)

***Apparatus*** **:-** EDU-ARM-2148 Trainer kit.

***Theory*** **:-**

## A. SCARM Installation:

As a part of the SCARM software package, you should have received a CDROM. Please insert it into the CD-ROM drive and run SETUP.EXE from it. Once you start the SETUP program, follow the instructions on the screen to complete the installation. The setup program will:

· Copy the required files to your hard disk
· If there are any compressed files, un-compress them
· Create a program group for SCARM

Once you have successfully installed the software on your hard disk, you can run it by clicking on

**Start/Programs/SPJ - SCARM/SIDEARM**. However, we recommend, that you go through this user's manual before you actually start using it.
**SCARM** is **SPJETs'** C Compiler for ARM. It also includes an **IDE** and other tools like **Debugger**, **Visual Code Generator (VCG)** and **Terminal Emulation Utility (SPJTerm)**. This document describes steps to create ARM applications in C using the SCARM.

**About "Project":**
What is a project?
A project is a file in which SIDEARM stores all information related to an application. E.g. it stores the name(s) of 'C' and/or Assembler source file(s), memory size to be used and other options for compiler, assembler and linker.

**Opening a project:**
To open an existing project file, select **Project / Open Project** from the menu.
**Creating a new project:**
To create a new project, select **Project / New Project** from the menu.
**Changing project settings:**
To change the project settings (such as adding or removing 'C' and/or Assembler source file(s), changing memory settings etc.), select Project / Settings from the menu.

## B. SCARM Quick Start for creating 'C' applications:

**1.** Start the **SIDE_ARM** program (i.e. the Integrated Development Environment) from start\Programs\SPJ-SCARM.

**2.** From Project menu, select **Close project** (if any project is open).

**3.** From Project menu, select **New Project**. The Open dialog window will be displayed. Select the desired path where you wish to create this new project. (For example, C:\SPJ).

CAUTION: The path and filename must not contain space or other special characters such as tab, comma, semicolon etc. In the “File name” field, type the name of the project, without any extension. For example, you may type “**PROG1**”. Then click on the “**Open**” button.

**4.** The action in the previous step will display the “**Project Settings**” dialog window. This dialog window has 3 different parts named “**Compiler Options**”, “**Linker Options**”, and “**Source Files**”. Any of these 3 parts can be displayed by clicking on the corresponding name near the top of this dialog window. Currently, the “**Compiler Options**” will be automatically displayed. If the target micro-controller (must be a member of ARM family) is known, you may select the appropriate Manufacturer from the list; and then select the appropriate micro-controller from the device list. If the target micro-controller is not known or if you cannot find it in the list, then you may simply select “**Philips**” as the manufacturer and “**LPC2148**” as the micro-controller.

**5.** Click on “**Linker Options**” to display that part of the dialog window. In this window, you will see a list of 8 “Memory Banks”, with names such as “Memory #1”, “Memory #2” and so on. In your target hardware, there may be none or 1 or more number of contiguous memory blocks connected to the ARM micro-controller. Check the appropriate number of memory banks to reflect the target’s memory blocks. For each checked memory bank, specify memory start address (in Hexadecimal) and memory block size (in decimal). Size maybe specified either in number of Kilobytes (KB) or Megabytes (MB). Some of the memory blocks maybe “read-only” (e.g. flash or conventional EPROM). Accordingly, you may check or uncheck the “Read only” box. Based on this information about memory banks, the IDE will automatically create the Linker Script. This auto-generated script is adequate for most users. However, if you wish to use your own script file instead of this autogenerated script, you may check the “Use different linker script” box and further click on the browse button (marked “…”) and select appropriate linker script file.

**6.** Click on “**Source Files**” to display that part of the dialog window. This window will indicate that IDE has automatically added 2 files in this new project: **PROG1.C** and **STARTUP.ASM**. The STARTUP.ASM file is automatically created by the IDE and is required for all C projects. Similarly, the IDE has automatically created an empty C file (PROG1.C). If the file PROG1.C already exists in the same path, then IDE would neither create/overwrite it nor modify it; but it will anyway add it to the project automatically. If you wish to add more files in this project, then click on the “**Add file**” button, select the desired filename and then click on “**Open**” button. Now the Project Settings dialog will indicate that selected file has been added into the project. When all necessary files have been added to the project, click “**OK**” button to create this new project.

**7.** The PROG1.C file created by the IDE will be an empty file containing only the frame of “main” function. You may write the desired program statements in this file (or other files that you may have added to the project). When done, select **Save** from **File menu**. If you have modified more than one source files, then select **Save All** from **File menu**.

**8.** From the **Compile** menu, select **Build**. This will invoke the Compiler to compile the file PROG1.C; and further (assuming no errors) invoke the linker to create the **.HEX** file. If there are any errors or warnings during the process of compiling, assembling or linking, then those will be displayed in the output window (below the editor window). If there are errors, then you may correct those by making appropriate changes to the program; select Save

from File menu to save the changes and then again select Build from Compile menu. Repeat this until there are no errors.

**9.** You may inspect contents of the folder where your project files reside. When there are no errors and build has completed successfully and then you will see a filename with same name as the project name and extension .HEX (in above example, **PROG1.HEX**). This is the file that you will need to use to program your micro-controller.

## C. Downloading & Running Programs:

The LPC2148 micro-controllers include on-chip flash for storing user program and non-volatile data. LPC2148 on EDU-ARM7-2148 have 512KBytes flash. This flash is In-System-Programmable (ISP). Therefore it is possible to download user program into on-chip flash of LPC2148, through serial port connected to PC. For doing so, a certain position of S3 switch is required. **S3 Switch should be continuously ON**. This section describes how to use the software Flash Magic to download program into LPC2148.

### 1. How to install Flash Magic:

The CD you have received with this board contains SCARM, C Compiler for ARM. Install it. After installation go to folder **C:\SCARM\Utilities**. This folder contains 5 zip files. Install Flash Magic from FlashMagic3.71.zip. Extract the **FlashMagic3.71.zip** and then run FlashMagic.exe from the extracted files. (If you have wrong version of Flash Magic already installed, then please uninstall it first and then install new version).

### 2. Download and Run program using Flash Magic into LPC2148:

· After installation of Flash Magic, open it.
· In Flash Magic go to Options -> Advanced Options-> Communications. Check High Speed Communications and keep Maximum Baud Rate as 19200. Click on OK.
· Again in Flash Magic go to Options -> Advanced Options-> Hardware Config. “Use DTR and RTS to control RST and P0.14” option should be checked. Click on OK.
(After doing above mentioned settings, Flash Magic stores it means for the next time just verify if these setting are proper or not. If they are proper then you can directly follow below mentioned procedure)
**a)** Connect the J1/UART0 connector of EDU-ARM7-2148 board to COM1 or COM2 of a PC, using the serial communication cable (supplied with the board).
**b)** Keep S3 switch in ON position. (You can keep S3 switch continuously ON) Switch ON power to the EDU-ARM7-2148.
**c)** Do proper settings in Flash Magic (COM Port: COM1 (if other choose it), Baud Rate: 38400, Device: LPC2148, Interface: None (ISP), Enable “Erase blocks used by Hex File”, Browse the file which you want to download) and click on Start button.
**d)** Flash Magic will download the program. Wait till Finished comes.
**e)** After downloading Flash Magic automatically resets the EDU-ARM7-2148 board and program executes. You can see output according to the program.
**f)** If again you want to Reset the board then Switch OFF and ON the power to the EDU-ARM7- 2148 board. You can see output according to the program.
**Note**: Flash Magic can be used to download the program into other Philips Microcontrollers also. See the list in Flash Magic itself.

## ***Study experiment:3***

***Aim*** **:-**Study of JTAG Debugger.

***Apparatus*** **:-** EDU-ARM-2148 Trainer kit.

***Theory*** **:-**

## A. JTAG Cable for Debugging:

**1. Type:**

**SJT-S:** Serial JTAG Cable.

**2. Contents:**

The "JTAG Cable" consists of following parts:

· Dongle (a small box with connectors on both ends).

· Cable

The "dongle" consists of some electronic circuit for interfacing the JTAG port of target processor to the host computer. The cable is a bunch of wires to connect the dongle with the JTAG port of target.

**3. Power Supply Requirements:**

The JTAG cable draws power from the target board. Thus it does not require a separate power source.

**4. Connecting JTAG Cable:**

**SJT-S:**

There is a DB9 female connector on one end of the dongle. This directly mates with the PC COM port –which has a DB9 male connector (or you can connect yellow color serial cable supplied with SJT-S between DB9 female connector on one end of the dongle and PC COM port –which has a DB9 male connector). The other end of the dongle has DB25 female connector. There is a DB25 male connector on one end of the cable. These DB25 female and DB25 male connectors are designed to mate with each other directly. The other end of the cable has a 20-pin header. This should be connected to the JTAG connector of the target board.

**CAUTION:**

The JTAG Cable must not be connected or dis-connected when power is applied to the target board. Turn off power to the target board, connect the JTAG Cable and then you may turn on power to the target board. Connecting the JTAG Cable with incorrect polarity / orientation may permanently damage the EDUARM7- 2148 board and/or the JTAG Cable. It will also make the warranty void for both the products.

## B. Verifying correct cable connection:

When the JTAG Cable is correctly connected to PC as well as the target board, it serves as a link between the JTAG port of target processor and the PC. This link is used by SPJ - SCARM software Tools (e.g. Debugger for ARM microcontrollers). This software tool, Debugger will work correctly only when the JTAG Cable is connected correctly.

**For SJT-S:**

There is a crude test to verify SJT-S JTAG Cable connection. You may please follow these steps:

1. Connect JTAG Cable between PC COM port and EDU-ARM7-2148, as per instructions in this manual.
2. Turn ON power to the target board.
3. On the PC, run SPJTerminal software.
4. In the Port Settings, select appropriate COM port (to which the JTAG cable is connected). Select 115200 baud, no parity, 8 bits per char, 1 stop bit and no flow control.
5. Open the COM Port connection.
6. Type character ‘G’ in the terminal window. I.e. send the character ‘G’ to the PC COM port.
7. If the JTAG Cable connection is correct, it will send back character ‘1’ or ‘0’. As a result, you will see either ‘1’ or ‘0’ appearing in the terminal window. This indicates that JTAG Cable connection is ok.
8. If you don’t see any character in the terminal window, probably the JTAG Cable is not connected appropriately.
9. If you see some character other than ‘1’ and ‘0’, then either the cable is not connected appropriately or the COM port settings are not as specified above.

## C. How to Debug Program:

1. Connect SJT-S as mentioned above.
2. Open project in SIDEARM. Rebuild it.
3. Download the same code in the target board.
4. In SIDEARM go to Tools -> Debugger.
5. In Debugger go to Run -> Click on “Not connected to target (click here to connect)”.
6. Device ID starting from 0x4..... will be displayed and program will run.
7. To stop program go to Run -> Stop. Now you can insert breakpoint and say Run.
8. You can use all the functions visible in Run option.
9. In variable watch window you can see only global variables.
10. If you have declared any global variables then find their addresses from .map file.
11. Insert these addresses in variable watch window and you can see global variables also.

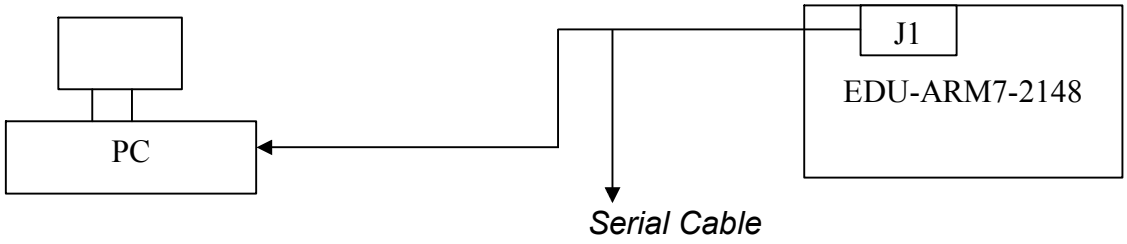
## ***Experiment No. 4 :***

***Aim*** :- Write a program for serial communication using UART0.W.A.P. to transfer message "Hello World !" serially at 19200 –baud rate 8-bit data and 1 stop bit using UART0.

***Apparatus*** **:-** EDU-ARM-2148 Trainer kit,SCRAM,PC

***Theory*** :-Write theory related to serial communication using UART0.

***Block Diagram:-***




***Programm:-*** *Program is given along with the compiler software.*

**Source Code:**

```
#include <Philips\LPC2148.h>
#include <stdio.h>

void main ()
{
        PINSEL0 = 0x00000005 ;
        InitUart0();
        printf("Hello World! \n");
        while(1)
        {
                putchar(getchar());
        }
}
```

**Function: UART0**

```
  #include <Philips\LPC2148.h>
//#include <Philips\LPC2138.h>
#define       VPBDIV                  (*((volatile WORD32 *) 0xE01FC100))
#define       U0RBR                   (*((volatile WORD32 *) 0xE000C000))
#define DESIRED_BAUDRATE 19200
#define CRYSTAL_FREQUENCY_IN_HZ 12000000
#define PCLK CRYSTAL_FREQUENCY_IN_HZ        /* since VPBDIV=0x01       */
#define DIVISOR (PCLK/(16*DESIRED_BAUDRATE))
void InitUart0(void)
```

```
{
/*      U0LCR: UART0 Line Control Register
        0x83: enable Divisor Latch access, set 8-bit word length,
        1 stop bit, no parity, disable break transmission       */
        U0LCR=0x83;

/*      VPBDIV: VPB bus clock divider
        0x01: PCLK = processor clock            */
        VPBDIV=0x01;

/*      U0DLL: UART0 Divisor Latch (LSB)        */
        U0DLL=DIVISOR&0xFF;

/*      U0DLM: UART0 Divisor Latch (MSB)        */
        U0DLM=DIVISOR>>8;

/*      U0LCR: UART0 Line Control Register
        0x03: same as above, but disable Divisor Latch access       */
        U0LCR=0x03 ;

/*      U0FCR: UART0 FIFO Control Register
        0x05: Clear Tx FIFO and enable Rx and Tx FIFOs              */
        U0FCR=0x05 ;
}
char putchar(char ch)
{
        if (ch=='\n')
        {
                //wait until Transmit Holding Register is empty
                while (!(U0LSR&0x20)) {}
                //then store to Transmit Holding Register
                U0THR='\r';
        }
        //wait until Transmit Holding Register is empty
        while (!(U0LSR&0x20)) {}
        //then store to Transmit Holding Register
        U0THR=ch;
        return ch;
}
char getchar(void)
{
        char ch;
        //wait until there's a character to be read
        while (!(U0LSR&0x01)) {}
        //then read from the Receiver Buffer Register
        ch=U0RBR;
        return ch;
}
```

***Output:*** *You can see output on SPJ Terminal.*

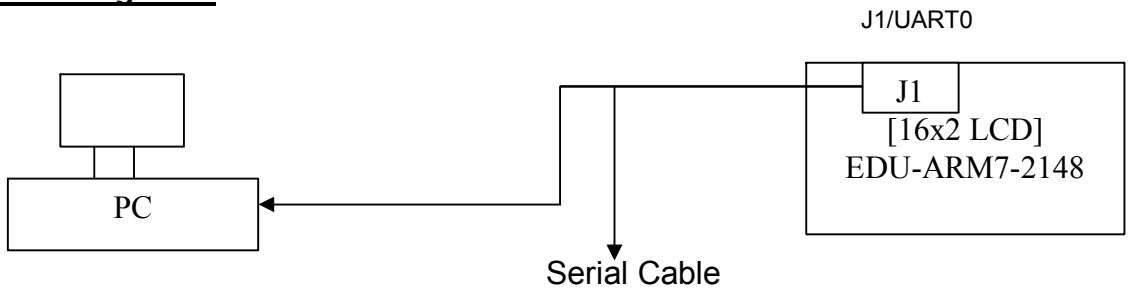
# ***Experiment No. 5:***

***Aim*** : Write a program to interface LCD.

***Apparatus*** **:-** EDU-ARM-2148 Trainer kit with 16x2 LCD,SCRAM,PC

***Theory*** **:-** Write theory related to LDC Interfacing with ARM Processors.

***Block Diagram:-***




***Programm:-*** *Program is given along with the compiler software.*

**Source Code:**

```
#include <Philips\LPC2148.h>
#include "lcd.h"
void main ()
{
        LcdInit();
        DisplayRow (1," JNEC  ");
        DisplayRow (2," Aurangabad  ");
        while(1)
        {}
}
```

**Function: LCD**

```
#include <Philips\LPC2148.h>
#include "lcd.h"
void SmallDelay (void)
{
int i;
        for(i=0;i<100;i++);
}
void LcdCmd1 (unsigned char cmd)
{
        if (cmd & 0x01)
                IO0SET = (1<<15);
        else
                IO0CLR = (1<<15);
```

```
	if (cmd & 0x02)
		IO0SET = (1<<17);
	else
		IO0CLR = (1<<17);
	if (cmd & 0x04)
		IO0SET = (1<<22);
	else
		IO0CLR = (1<<22);
	if (cmd & 0x08)
		IO0SET = (1<<30);
	else
		IO0CLR = (1<<30);
	IO1CLR = 0x03000000  ;   // make rs and en low
	SmallDelay() ;
	IO1SET = 0x01000000  ;   // enable en
	SmallDelay() ;
	IO1CLR = 0x01000000  ;   // disable en
	SmallDelay() ;
}
void LcdDat1 (unsigned char dat)
{
	if (dat & 0x01)
		IO0SET = (1<<15);
	else
		IO0CLR = (1<<15);
	if (dat & 0x02)
		IO0SET = (1<<17);
	else
		IO0CLR = (1<<17);
	if (dat & 0x04)
		IO0SET = (1<<22);
	else
		IO0CLR = (1<<22);
	if (dat & 0x08)
		IO0SET = (1<<30);
	else
		IO0CLR = (1<<30);
	IO1SET = 0x02000000  ;   // make RS high
	SmallDelay() ;
	IO1CLR = 0x01000000  ;   // disable en
	SmallDelay() ;
	IO1SET = 0x01000000  ;   // enable en
	SmallDelay() ;
	IO1CLR = 0x01000000  ;   // disable en
	SmallDelay() ;
}
void Delay250 (void)
{
	int k,j ;
```

```
        j =200 ;
        for(k = 0 ; k < 100 ; k ++)
        {
                j-- ;
        }
}
void DelayMs (int n)
{
        int k ;
        for(k = 0 ; k < n ; k ++)
        {
                Delay250() ;
                Delay250() ;
        }
}
void LcdCmd (unsigned char cmd)
{
        LcdCmd1(cmd >> 4) ;
        LcdCmd1(cmd) ;
        Delay250() ;
        Delay250() ;
}
void LcdDat (unsigned char dat)
{
        LcdDat1(dat >> 4) ;
        LcdDat1(dat) ;
        Delay250() ;
        Delay250() ;
}
void LcdInit (void)
{
        IO1DIR = 0x03000000  ;
        IO1CLR = 0x03000000  ;
        IO0DIR = 0x40428000  ;
        IO0CLR = 0x40428000  ;

        DelayMs(6) ;
        LcdCmd1(0x03) ;
        DelayMs(6) ;
        LcdCmd1(0x03) ;
        Delay250() ;
        LcdCmd1(0x03) ;
        Delay250() ;
        LcdCmd1(0x02) ;
        Delay250() ;
        LcdCmd(0x28) ;
        LcdCmd(0x08) ;
        LcdCmd(0x0c) ;
        LcdCmd(0x06) ;
```

```
}
void DisplayRow (int row, char *str)
{
/*
        pass pointer to 16 character string
        displayes the message on line1 or line2 of LCD, depending on whether row is 1 or 2.
*/
        int k ;
        if (row == 1)
                LcdCmd(0x80) ;
        else
                LcdCmd(0xc0) ;
        for(k = 0 ; k < 16 ; k ++)
        {
                if (str[k])
                        LcdDat(str[k]) ;
                else
                        break ;
        }
        while(k < 16)
        {
                LcdDat(' ') ;
                k ++ ;
        }
}
```

***Output:*** *You can see the message  Hello World on LCD.If required reset the board.*

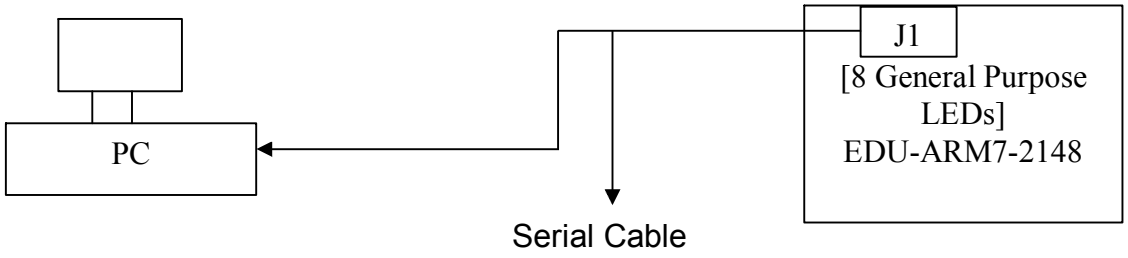
# ***Experiment No. 6***

***Aim*** **:** Write a program to Blink LED ‘s present on EDU-ARM-2148.

***Apparatus*** **:-** EDU-ARM-2148 Trainer kit ,SCRAM,PC

***Theory*** **:-** Write theory related to LED interfacing using ARM.

***Block Diagram:-***




***Programs:-*** *Program is given along with the compiler software.*

**Source Code:**

```
#include <Philips\LPC2148.h>
#include <stdio.h>

void main ()
{
        PINSEL0 = 0x00000005 ;
        InitUart0();
        printf("Hello World! \n");
        while(1)
        {
                putchar(getchar());
        }
}
```

***Output:*** *You can see the message Hello World on LCD.If required reset the board.*

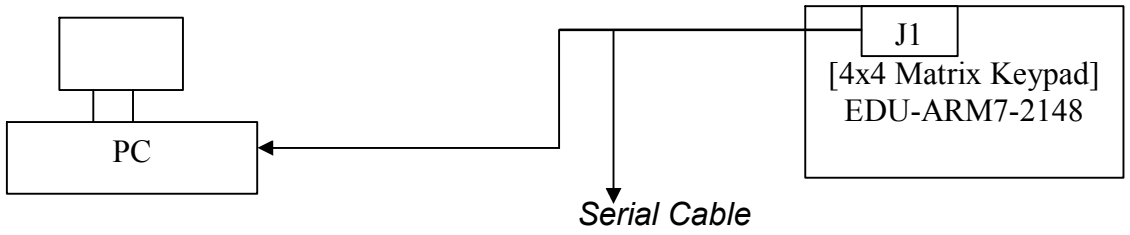
# ***Experiment No.7***

***Aim*** **:** Write a program to interface 4*4 matrix keyboard.

***Apparatus*** **:-** EDU-ARM-2148 Trainer kit ,SCRAM,PC

***Theory*** **:-** Write theory related to interfacing of matrix keypads.

***Block Diagram:-***




***Programm:-*** *Program is given along with the compiler software.*

**Source Code:**

```
#include <Philips\LPC2148.h>
#include <stdio.h>
#include "KBD.h"
#include "UART0.h"
#include "TYPE.h"
#include "lcd.h"

int i8ch ;
char szTemp[16] ;

void main (void)
{
        PINSEL0 |= 0x00000005 ;
        PINSEL1 |= 0x00000000 ;
        UART0_Init();
        LcdInit();
        DisplayRow(1,"Keypad Test ");
        DisplayRow(2,"Press Any Key  ");
        KBD_Init();
        {

                puts("This is a keyboard test program\nPress any key\nKeycode will be
displayed\n") ;
```

```
                while(1)
                {
                        i8ch = KBD_rdkbd() ;
                        printf("Keycode = %02X\n", i8ch) ;
                        sprintf(szTemp,"KeyCode = %02X",i8ch);
                        DisplayRow(2,szTemp) ;
                }
        }
}
```

**Function: KBD**

```
#include <Philips\LPC2148.h>
#include "KBD.h"
#include "TYPE.h"
#define RET0        0x00010000  // P1.16
#define RET1        0x00020000  // P1.17
#define RET2        0x00040000  // P1.18
#define RET3        0x00080000  // P1.19
#define        ALL_RET     (RET0|RET1|RET2|RET3)
#define SCAN0       0x00100000  // P1.20
#define SCAN1       0x00200000  // P1.21
#define SCAN2       0x00400000  // P1.22
#define SCAN3       0x00800000  // P1.23

const uint32 KBD_u32scanport1[4]={SCAN0,SCAN1,SCAN2,SCAN3};
const uint32 KBD_u32retport[4]={RET0,RET1,RET2,RET3};

#define CRYSTAL_FREQUENCY_IN_HZ 12000000
#define PLL_MULTIPLIER 1
#define DEL_LOOPS       CRYSTAL_FREQUENCY_IN_HZ*PLL_MULTIPLIER/500000

int32 KBD_i32keydown;
void KBD_cdelay(void)
{
        int32 i32i;
        for (i32i=0; i32i<5*DEL_LOOPS; i32i++) {}
}

void KBD_Init(void)
{
        //IO1DIR &= (0xFFFFFFFF^(RET0|RET1|RET2|RET3));
        //printf("IO1PIN  %02bx\n",IO1PIN) ;

        KBD_i32keydown=-1;
}

int32 KBD_i32the_sc ;
int32 KBD_kbhit (void)
{
```

```
int32 i32sc , i32sc0 ;
int32 i32ret, i32ret0 ;

for(i32sc = 0;i32sc < 4 ; i32sc ++)
{
IO1DIR = KBD_u32scanport1[i32sc] | (0x03000000)    ; // For RS and RW of LCD
IO1CLR = KBD_u32scanport1[i32sc] ;

        KBD_cdelay() ;
        i32ret = IO1PIN & ALL_RET ;

        if (KBD_i32keydown == -1)
        {
                switch(i32ret)
                {
                        case   (ALL_RET & (~RET0))      :
                                        KBD_i32the_sc = i32sc ;
                                        KBD_i32keydown = (i32sc * 4) ;
                                        //printf("RET0") ;
                                        //printf("%u\n",i32sc) ;
                                        return 1 ;
                        case   (ALL_RET & (~RET1))      :
                                        KBD_i32the_sc = i32sc ;
                                        KBD_i32keydown = (i32sc * 4) + 1 ;
                                        //printf("RET1") ;
                                        //printf("%u\n",i32sc) ;
                                        return 1 ;
                        case   (ALL_RET & (~RET2))      :
                                        KBD_i32the_sc = i32sc ;
                                        KBD_i32keydown = (i32sc * 4) + 2 ;
                                        //printf("RET2") ;
                                        //printf("%u\n",i32sc) ;
                                        return 1 ;
                        case   (ALL_RET & (~RET3))      :
                                        KBD_i32the_sc = i32sc ;
                                        KBD_i32keydown = (i32sc * 4) + 3 ;
                                        //printf("RET3") ;
                                        //printf("%u    \n",i32sc) ;
                                        return 1 ;
                }
        }
        else
        {
                // i.e. key is already pressed, wait until it is released
                if (i32sc == KBD_i32the_sc)
                {
                        if (i32ret == ALL_RET)
                        {
                                KBD_i32keydown = -1 ;     // key has been released
```

```
                                }
                        }
                }
        }
        return 0 ;
}
int32 KBD_rdkbd(void)
{
        while(!KBD_kbhit()) {}
        return KBD_i32keydown ;
}
```

**Function: LCD**

```
#include <Philips\LPC2148.h>
#include "lcd.h"

void SmallDelay (void)
{

        int i;
        for(i=0;i<100;i++);
}
void LcdCmd1 (unsigned char cmd)
{
        if (cmd & 0x01)
                IO0SET = (1<<15);
        else
                IO0CLR = (1<<15);
        if (cmd & 0x02)
                IO0SET = (1<<17);
        else
                IO0CLR = (1<<17);
        if (cmd & 0x04)
                IO0SET = (1<<22);
        else
                IO0CLR = (1<<22);
        if (cmd & 0x08)
                IO0SET = (1<<30);
        else
                IO0CLR = (1<<30);
        IO1CLR = 0x03000000  ;   // make rs and en low
        SmallDelay() ;
        IO1SET = 0x01000000  ;   // enable en
        SmallDelay() ;
        IO1CLR = 0x01000000  ;   // disable en
        SmallDelay() ;
}
```

```
void LcdDat1 (unsigned char dat)
{
	if (dat & 0x01)
		IO0SET = (1<<15);
	else
		IO0CLR = (1<<15);
	if (dat & 0x02)
		IO0SET = (1<<17);
	else
		IO0CLR = (1<<17);
	if (dat & 0x04)
		IO0SET = (1<<22);
	else
		IO0CLR = (1<<22);
	if (dat & 0x08)
		IO0SET = (1<<30);
	else
		IO0CLR = (1<<30);

	IO1SET = 0x02000000  ;   // make RS high
	SmallDelay() ;
	IO1CLR = 0x01000000  ;   // disable en
	SmallDelay() ;
	IO1SET = 0x01000000  ;   // enable en
	SmallDelay() ;
	IO1CLR = 0x01000000  ;   // disable en
	SmallDelay() ;
}
void Delay250 (void)
{
	int k,j ;
	j =200 ;
	for(k = 0 ; k < 100 ; k ++)
	{
		j-- ;
	}
}
void DelayMs (int n)
{
	int k ;
	for(k = 0 ; k < n ; k ++)
	{
		Delay250() ;
		Delay250() ;
	}
}
void LcdCmd (unsigned char cmd)
{
	LcdCmd1(cmd >> 4) ;
```

```
	LcdCmd1(cmd) ;
	Delay250() ;
	Delay250() ;
}
void LcdDat (unsigned char dat)
{
	LcdDat1(dat >> 4) ;
	LcdDat1(dat) ;
	Delay250() ;
	Delay250() ;
}
void LcdInit (void)
{
	IO1DIR |= 0x03000000  ;
	IO1CLR |= 0x03000000  ;
	IO0DIR |= 0x40428000  ;
	IO0CLR |= 0x40428000  ;

	DelayMs(6) ;
	LcdCmd1(0x03) ;
	DelayMs(6) ;
	LcdCmd1(0x03) ;
	Delay250() ;
	LcdCmd1(0x03) ;
	Delay250() ;
	LcdCmd1(0x02) ;
	Delay250() ;
	LcdCmd(0x28) ;
	LcdCmd(0x08) ;
	LcdCmd(0x0c) ;
	LcdCmd(0x06) ;
}
void DisplayRow (int row, char *str)
{
/*
	pass pointer to 16 character string
	displayes the message on line1 or line2 of LCD, depending on whether row is 1 or 2.
*/
	int k ;
	if (row == 1)
		LcdCmd(0x80) ;
	else
		LcdCmd(0xc0) ;
	for(k = 0 ; k < 16 ; k ++)
	{
		if (str[k])
			LcdDat(str[k]) ;
		else
			break ;
```

```
        }
        while(k < 16)
        {
                LcdDat(' ') ;
                k ++ ;
        }
}
```

***Output:*** *In this program after pressing any ,its code is send to serial port using UART0.You can see output onb SPJT terminal.*

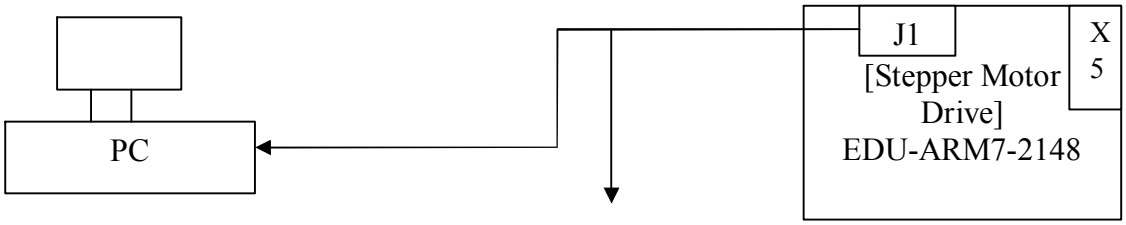
# ***Experiment No.8***

***Aim*** **:** Write a program to interface Stepper motor.

***Apparatus*** **:-** EDU-ARM-2148 with stepper motor Trainer kit ,SCRAM,PC

***Theory*** **:-** Write theory related to stepper motor interfacing.

***Block Diagram:-***




*Serial Cable*

***Programm:-*** *Program is given along with the compiler software.*

**Source Code:**

```
#include <Philips\LPC2148.h>
#define PHASEA 0x00002400
#define PHASEB 0x00001400
#define PHASEC 0x00001800
#define PHASED 0x00002800

unsigned int delay ;
void main ()
{
        PINSEL0 = 0x00000000 ;
        PINSEL1 = 0x00000000 ;
        IO0DIR = 0x003C3C00 ;
        IO0SET = 0x003C0000 ;
        //PINSEL2 = 0x00000000 ;
        //IO1DIR = 0x00000000 ;
        while(1)
        {
                IO0SET = PHASEA ;
                IO0CLR = (~PHASEA) & 0x00003C00 ;
                for(delay=0; delay<30000; delay++) ;
                IO0SET = PHASEB ;
                IO0CLR = (~PHASEB) & 0x00003C00 ;
```

```
                for(delay=0; delay<30000; delay++) ;
                IO0SET = PHASEC ;
                IO0CLR = (~PHASEC) & 0x00003C00 ;
                for(delay=0; delay<30000; delay++) ;
                IO0SET = PHASED ;
                IO0CLR = (~PHASED) & 0x00003C00 ;
                for(delay=0; delay<30000; delay++) ;
        }
}
```

***Output:*** *You can see stepper motor moving in particular direction and corresponding phase changes you can observed LED's D9-D12.*

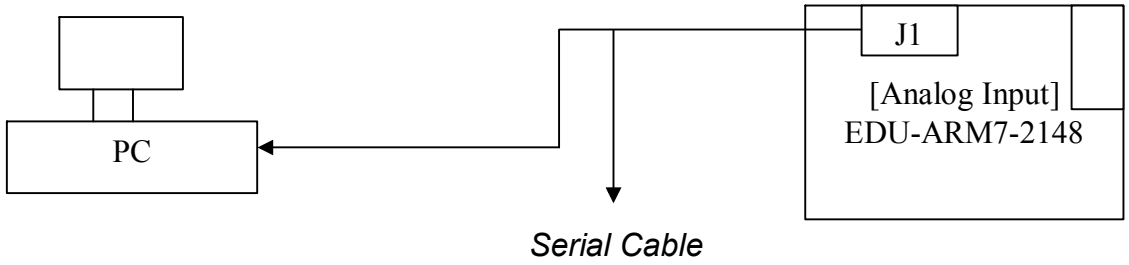
# ***Experiment No.9***

***Aim*** **:** Write a program for interfacing of LM35 temperature sensor to on chip ADC(ADC0.Channel 2,P0.29)

***apparatus*** **:-** EDU-ARM-2148 Trainer kit ,SCRAM,PC

***Theory*** **:-** Write theory related to interfacing of sensors.

***Block Diagram:-***




***Programm:-*** *Program is given along with the compiler software.*

**Source Code:**

```
#include <Philips\LPC2138.h>
#include <stdio.h>
#include "TYPE.h"
#include "uart0.h"

void ADC_Init (void)
{
        PINSEL1 = 0x04000000  ; //P0.29,AD0.2
}
unsigned int ADC_GetAdcReading (uint32 u32ChnlNum)
{
        AD0CR = 0x01200300 + (1 << u32ChnlNum) ;
        while(!((u32ChnlNum = AD0DR) & 0x80000000))
        {
        }
        return((u32ChnlNum >> 6) & 0x3ff) ;
}
int main (void)
{
        uint32 u32k = 2;
        UART0_Init() ;
        ADC_Init() ;
```

```
	while(1)
	{
		printf("Press any key to get channel 2 reading...\n") ;
		getchar() ;
		if(u32k)
		{
			printf("Chnl %d: 0x%03X\n", u32k, ADC_GetAdcReading(u32k)) ;
		}
	}
	return 0 ;
}
```

**Function: UART0**

```
#include <Philips\LPC2148.h>
#include "TYPE.h"
#define DESIRED_BAUDRATE 19200
#define CRYSTAL_FREQUENCY_IN_HZ 12000000
#define PCLK CRYSTAL_FREQUENCY_IN_HZ		// since VPBDIV=0x01
#define DIVISOR (PCLK/(16*DESIRED_BAUDRATE))

void UART0_Init(void)
{
	PINSEL0 |= 0x00000005 ;
/*	U0LCR: UART0 Line Control Register
	0x83: enable Divisor Latch access, set 8-bit word length,
	1 stop bit, no parity, disable break transmission	*/
	U0LCR=0x83;
/*	VPBDIV: VPB bus clock divider
	0x01: PCLK = processor clock		*/
	VPBDIV=0x01;
/*	U0DLL: UART0 Divisor Latch (LSB)	*/
	U0DLL=DIVISOR&0xFF;
/*	U0DLM: UART0 Divisor Latch (MSB)	*/
	U0DLM=DIVISOR>>8;
/*	U0LCR: UART0 Line Control Register
	0x03: same as above, but disable Divisor Latch access	*/
	U0LCR=0x03 ;
/*	U0FCR: UART0 FIFO Control Register
	0x05: Clear Tx FIFO and enable Rx and Tx FIFOs		*/
	U0FCR=0x05 ;
}
int8 putchar(int8 i8ch)
{
	if (i8ch=='\n')
	{
		//wait until Transmit Holding Register is empty
		while (!(U0LSR&0x20)) {}
		//then store to Transmit Holding Register
		U0THR='\r';
	}
```

```
        //wait until Transmit Holding Register is empty
        while (!(U0LSR&0x20)) {}
        //then store to Transmit Holding Register
        U0THR=i8ch;
        return i8ch;
}
int8 getchar(void)
{
        int8 i8ch;
        //wait until there's a character to be read
        while (!(U0LSR&0x01)) {}
        //then read from the Receiver Buffer Register
        i8ch=U0RBR;
        return i8ch;
}
```

***Output:*** *you can see digital reading of the corresponding analog input from LM35 on SPJ terminal.*

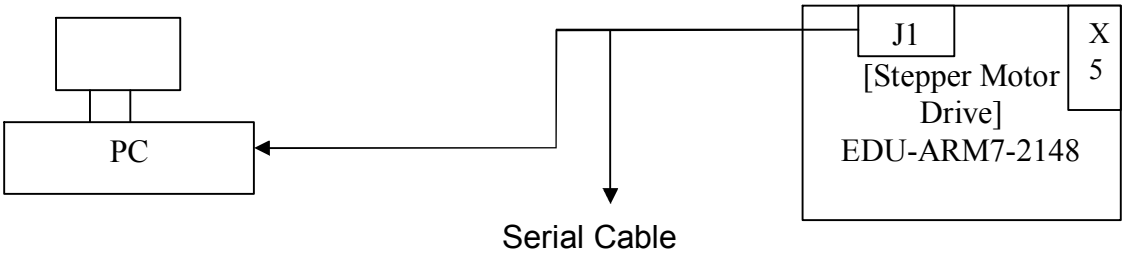
# ***Experiment No.10***

***Aim*** **:** Write a program to generate RAMP wave using on chip DAC(P0.25)

***apparatus*** **:-** EDU-ARM-2148 Trainer kit ,SCRAM,PC,Oscilloscope

***Theory*** **:-** Write theory related to DAC.

***Block Diagram:-***




***Programm:-*** *Program is given along with the compiler software.*

**Source Code:**

```
#include <Philips\LPC2148.h>
#define       DACR                (*((volatile WORD32 *) 0xE006C000))
#include "UART0.h"
#include "LCD.h"
#include <stdio.h>
unsigned int g_uiAoutValue = 0 ;
int main (void)
{
        PINSEL0 = 0x55 ;
        PINSEL1 = 0x00080000 ;  // to use P0.25 as DAC output pin.
        InitUart0() ;
        LcdInit() ;
        DisplayRow(1,"DAC/MiniARM-2148") ;
        DisplayRow(2,"spjsystems.com  ") ;
        puts("DAC example program by:\n\n\nwww.spjsystems.com\n\n\nRuns on MiniARM-
2148 Evaluation board\nGenerates a ramp\n") ;
        for(;;)
        {
                for(g_uiAoutValue = 0 ; g_uiAoutValue < 1024 ; g_uiAoutValue ++)
                {
                        DACR = g_uiAoutValue << 6 ;
                }
        }
}
```

***Output:*** *you can see Ramp wave on oscilloscope.*

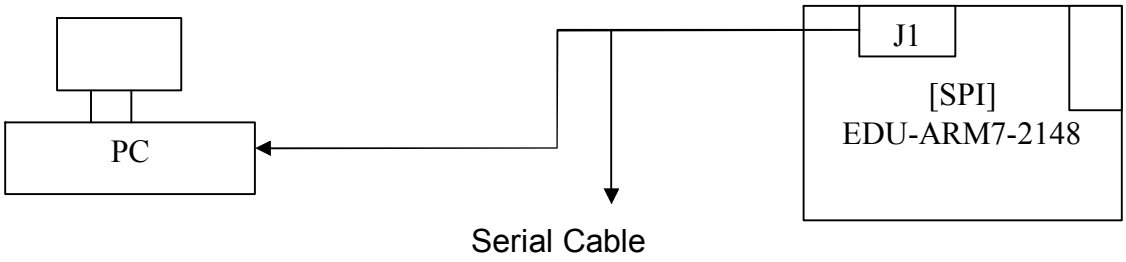
# ***Experiment No.11***

***Aim*** **:** Write a program to interface AT25256 EEprom using SPI protocol.

***Apparatus*** **:-** EDU-ARM-2148 Trainer kit ,SCRAM,PC

***Theory*** **:-** Write theory related to SPI Protocol.

***Block Diagram:-***




***Programm:-*** *Program is given along with the compiler software.*

**Source Code:**

```
#include <Philips\LPC2148.h>
#include <Stdio.h>
#include "Type.h"
#include "UART0.h"
#include "TIMER.h"
#include "SPI.h"

/*************************************************
Main function.
*************************************************
*/
void main(void)
{
        uint8 u8buffer1[20] = {'H','E','L','L','O',0};
        uint8 u8buffer2[20] ;
        UART0_Init();
        SPI_Init();
        TIMER_Init();
        printf("Init done.");
        while(1)
        {
                if (!SPI_WriteToSPIEEPROM(0,  u8buffer1, 6))
                        printf("\nMemory write error.");

                if (!SPI_ReadFromSPIEEPROM(0, u8buffer2, 6))
```

```
                        printf("\nMemory Read error..");
                printf("\nData: %s",u8buffer2);
        }
}
```

**Function: Timer**

```
#include <Philips\LPC2148.h>
#include "Type.h"
#include "Timer.h"

/************************************************
Timer0 is used for delay in micro sec
Timer1 is used for time out delay
Timer is initialized to simply count at a specified
frequency(Configuration.h).
Count can be read from register TC.
************************************************
*/
/*
Initialises the timers
*/
void TIMER_Init (void)
{
        // Power ON timer peripheral
        PCONP       |= 0x00000006;
        // TPC: Timer Prescaler counter
        // the counter is incremented once every TPC+1 cycles of PCLK
        T0PR = PRESCALER0;
        //T1PR         = PRESCALER1;
        // TCR: Timer  Control Register
        // 2: reset counters (both timer and prescaler)
        // 1: enable counting
        T0TCR=2;
        //T1TCR=2;
}
/*
waits for next tick in timer
*/
void TIMER_WaitForNextTick (void)
{
        uint32 start=T0TC;
        while (T0TC==start) {}       // wait until timer counter changes, then leave
}
```

**Function: SPI**

```
#include <Philips\LPC2148.h>
#include <Stdio.h>
#include "Type.h"
#include "UART0.h"
```

```
#include "TIMER.h"
#include "SPI.h"
/*****************************************************
Contains all SPI related functions.
For Interfacing SPI0 is used.
*****************************************************
*/
/*
Initialises the SPI protocol and port pins.
*/
void SPI_Init (void)
{
        // Power on SPI0 peripheral
        PCONP       |= 0x00000100;
        // Define port pin as SCL0, MISO0 and MOSI0
        PINSEL0     |= 0x00001500;
        // Set #CS pin
        Set_SPI_CS0_AS_OP();
        Set_SPI_CS0();
        // Set SPI clock speed.
   S0SPCCR = SPI_EEPROM_SPEED_DIVISOR & 0xFE;
        // Operates in master mode
   S0SPCR    = 0x00000020 ;
}
/*
Reads data from SPIEEPROM.
Length should be > 0.
Return:         True on valid data and
                False on time out or any error with device
*/
BOOL SPI_ReadFromSPIEEPROM (uint32 u32startAddr, uint8 *u8ptr2arr, uint32 u32len)
{
        uint32 u32i;
        uint32 u32dummyData;
        // Check for upper limit
        if ((u32startAddr + u32len >= SPIEEPROM_SIZE) || (u32len ==0))
                return FALSE;
        // Select device
        Clr_SPI_CS0();
        // Send Read opcode
        S0SPDR = CMD_SPIEEPROM_READ;
        while (!(S0SPSR & 0x80));
        u32dummyData = S0SPDR;
        // Send start address
        S0SPDR = (u32startAddr>>8) & 0xFF;
        while (!(S0SPSR & 0x80));
        u32dummyData = S0SPDR;
        S0SPDR = u32startAddr & 0xFF;
        while (!(S0SPSR & 0x80));
```

```
	u32dummyData = S0SPDR;
	// Read byte wise
	for (u32i=0;u32i<u32len;u32i++)
	{
		S0SPDR = 0;// Dummy write
		while (!(S0SPSR & 0x80));
		// Read data
		u8ptr2arr[u32i] = S0SPDR;
	}
	// Deselect device
	Set_SPI_CS0();
	return TRUE;
}
/*
Writes data to SPIEEPROM.
Length should be > 0.
Return:	True on successful write and
		False on time out or any error with device
*/
BOOL SPI_WriteToSPIEEPROM (uint32 u32startAddr, uint8 *u8ptr2arr, uint32 u32len)
{
	uint32 u32i;
	uint32 u32dummyData;
	// Check for upper limit
	if ((u32startAddr + u32len >= SPIEEPROM_SIZE) || (u32len ==0))
		return FALSE;

	// write data byte wise
	for (u32i = 0; u32i < u32len; u32i++)
	{
		// Write enable device
		Clr_SPI_CS0();
		// Send Write Enable opcode
		S0SPDR = CMD_SPIEEPROM_WREN;
		while (!(S0SPSR & 0x80));
		u32dummyData = S0SPDR;
		Set_SPI_CS0();
		// Select device
		Clr_SPI_CS0();
		// Send Write opcode
		S0SPDR = CMD_SPIEEPROM_WRITE;
		while (!(S0SPSR & 0x80));
		u32dummyData = S0SPDR;
		// Send start address
		S0SPDR = (u32startAddr>>8) & 0xFF;
		while (!(S0SPSR & 0x80));
		u32dummyData = S0SPDR;
		S0SPDR = u32startAddr & 0xFF;
		while (!(S0SPSR & 0x80));
```

```
                u32dummyData = S0SPDR;
                u32startAddr++;
                // Send data
                S0SPDR = u8ptr2arr[u32i];
                while (!(S0SPSR & 0x80));
                u32dummyData = S0SPDR;
                Set_SPI_CS0();
                // Wait for writing is complete
                if (!SPI_WaitForEEPROMReady())
                        return FALSE;
        }
        return TRUE;
}
/*
Waits until byte program is done.
Return:         True on EEPROM is ready with write done and False on time out
*/
BOOL SPI_WaitForEEPROMReady (void)
{
        uint32 u32dummyData;
        TIMER0_RESET();
        TIMER0_ENABLE();
        while (T0TC < SPIEEPROM_WRITE_TIME_OUT)
        {
                // Poll for ready bit
                Clr_SPI_CS0();
                S0SPDR = CMD_SPIEEPROM_RDSR;
                while (!(S0SPSR & 0x80));
                u32dummyData = S0SPDR;
                S0SPDR = 0;
                while (!(S0SPSR & 0x80));
                if (!(S0SPDR & 0x01))
                {
                        Set_SPI_CS0();
                        TIMER0_DISABLE();
                        return TRUE;
                }
                Set_SPI_CS0();
        }
#ifdef DEBUG
        printf("Timed out..WaitForEEPROMReady");
#endif
        TIMER0_DISABLE();
        return FALSE;
}
```

***Output:*** *you can see output on SPJ terminal.*

**4. Quiz on the subject**

1. Write short notes on:
    a. Priority Scheduling
    b. Multitasking
    c. Seamthort

**5. Conduction of Viva-Voce Examination:**

Teacher should conduct oral exams of student with full preparation. Normally the objective questions with guesses are to be avoided. To make it meaningful, the questions should be such that depth of the students in the subject is tested. Oral examinations are to be conducted in cordial environment amongst the teacher taking the examination. Teachers taking such examinations should not have ill thoughts about each other and courtesies should be offered to each other in case of difference of opinion, which should be critically suppressed in front of the students.

**6. Evaluation and Marking systems:**

Basic honesty in the evaluation and marking system is absolutely essential and in the process impartial nature of the evaluator is required in the examination system to become successful. It is wrong approach or concept to award the students by way of easy making to get cheap popularity among the students which they do not deserve. It is a primary responsibility of the teacher to see that right students are really putting up lot of hard work with right kind of intelligence are correctly awarded.
The marking pattern should be justifiable to the students without any ambiguity and teacher should see that the students are faced with just circumstances.