Competence, Commitment and Teamwork

SDME Society's

**SDM COLLEGE OF ENGINEERING AND TECHNOLOGY, DHARWAD – 580 002**

(An Autonomous Institution affiliated to Visvesvaraya Technological University,Belagavi - 590 018)


## Department of Computer Science and Engineering

# Laboratory Manual

**Course Title:** ARM Processor Laboratory

**Course Code:** 18UCSL406

**Semester: IV**

**Course Instructors**

1. Nita Kakhandaki
2. Basavaraj Vadatti

**Academic Year 2023-2024**

**Course Learning Objectives (CLOs):** This course focuses on the following learning perspectives:

- Understand the internal architecture, instruction set of ARM7 microcontroller, assembling process & implement small programs.
- Design & develop Assembly Language Program /& C program for a given real time application.
- Understand the use of interrupts & other advanced concepts related to ARM7
- Demonstrate working knowledge of the necessary steps and methods used to interface ARM7 to devices such as motors, LCD, ADC, and DAC etc.

**Course Outcomes (COs)**

| CO | Description of the Course Outcome: At the end of the course, the student will be able to: | Mapping to POs/PSOs | | |
|---|---|---|---|---|
| | | Substantial | Moderate | Low |
| CO-1 | **Execute** assembly level codes for a given specific problem using ARM processor. | - | 2, 4 | 3,15 |
| CO-2 | **Execute** embedded C programs for a given specific problem using ARM processor. | - | 4,14 | 15,16 |
| CO-3 | **Implement** programs for interfacing with real world devices such as LCD's Keyboards, DAC, ADC, Relays Motors etc. | 13 | 4,5,16 | 3,12 |

| POs | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mapping Level | - | 2.0 | 1.0 | 2.0 | 2.0 | - | - | - | - | - | - | 1.0 | 3.0 | 2.0 | 1.0 | 1.5 |

**Course Contents:**

**PART A:** Conduct the following experiments to learn ALP using ARM:

- Arithmetic and logical operations
- Interrupts related operations
- Timer related applications.

**PART B:** Conduct interfacing experiments to learn embedded C for ARM:

- LCD- interfacing
- Stepper Motor Interfacing
- Real time sensors Interfacing
- 7-segment LED interface

**Reference Books:**


1. Andrew N. Sloss, ARM System Developer's guide, ELSEVIER Publications, 2016
2. William Hohl, ARM Assembly Language, CRC Press.
3. Steve Furber, ARM System-on-chip Architecture by, Pearson Education, 2012
4. James K. Peckol, Embedded Systems: A Contemporary Design Tool, 2008
5. Jonathan W. Valvano, Brookes / Cole, Embedded Microcomputer Systems, Real Time Interfacing, 1999
6. LPC 2148 USER MANUAL.


**List of Term-Works:**

**1. Learning Outcome:**

**a) Write appropriate assembly code using ARM processor syntax**

1. Write an Assembly language program to
2. Write an Assembly language program to
3. Write an Assembly language program to
4. Write an Assembly language program to
5. Write an Assembly language program to

**b) Write appropriate embedded C code using ARM processor syntax**

6. Write an embedded C program to
7. Write an embedded C program to

**c) Write appropriate embedded C code for interfacing with real world devices using ARM processor**

8. Write an embedded C program to
9. Write an embedded C program to
10. Write an embedded C program to

# Contents

| | | | | |
|---|---|---|---|---|
| | | | | |
| **9** | Conversion of **upper case to lower case** and vice versa. | **2** | **4** | **12** |
| **10** | Routine that **reverses the bits** in a register | **2** | **4** | **12** |
| **11** | Find the **maximum value** in a list of 32-bit numbers(two's complement representations) located in memory | **3** | **4** | **12** |
| **Appendix** | **A Quick Tour of the IDE** | | | **12** |
| | **Flash Magic** | | | **21** |

## Mapping of COs to Marks

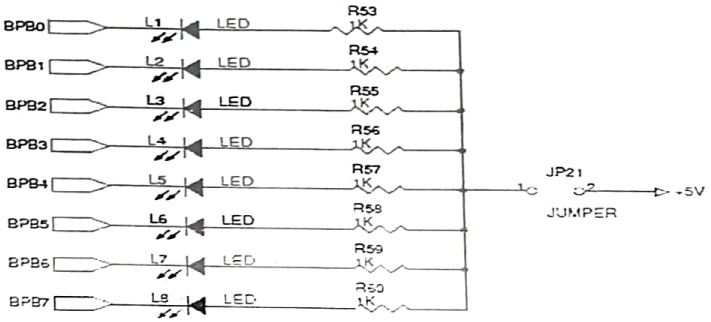
| CO | **Description of the Course Outcome:** At the end of the course, the student will be able to: | **Marks allotted** |
|---|---|---|
| **CO-1** | **Execute** assembly level codes for a given specific problem using ARM processor. | **10** |
| **CO-2** | **Execute** embedded C programs for a given specific problem using ARM processor. | **20** |
| **CO-3** | **Implement** programs for interfacing with real world devices such as LCD's Keyboards, DAC, ADC, Relays Motors etc. | **20** |

## 1. Write a C program to blink the LEDs with fixed delay

Understand the board schematics, LPC2148 Ports, Port/Pin Configuration. Develop an understanding of the startup.s file. To be able to use the compiler with microlib,linker.To develop an understanding of .elf,.axf,.sct and other file formats.

Procedure: Configure PINSEL, IOPIN, IODIR, IOSET, IOCLR registers. and the write a C code to blink the LEDs in required fashion. Short JP21.

There are 8 LEDs on board and these are connected to P0.16(PB0) to P0.23(PB7) refer Board Manual for the same.




Code for the same is

Startup.s | Blink.c | LPC21xx.h | stdint.h

```
1  #include <LPC21xx.h>
2
3  unsigned int delay;
4
5  int main ()
6  {
7    PINSEL1 = 0x00000000 ;     // Configure P0.16 to P0.31 as GPIO
8     IO0DIR  = 0x00FF0000 ;     // Configure P0.16 to P0.23 as Output
9
10   while(1)
11   {
12     IO0CLR = 0x00FF0000;          // CLEAR (0) P0.10 to P0.13 and P0.18 to P0.21, LEDs ON
13     for(delay=0; delay<500000; delay++);  // delay
14     IO0SET = 0x00FF0000;          // SET (1) P0.10 to P0.13 and P0.18 to P0.21, LEDs OFF
15     for(delay=0; delay<500000; delay++);  // delay
16   }
17
18 }
19
```

Compile and build the code and download it on to kit referring the steps on annexure.

**Adjust delays, observe patterns such as Ring counter Johnson counter and implement over kit.**

## 2. Write a C program to implement a. Hex Counter b. Binary displays counter on 7-segment LEDs.

Therearefourmultiplexed7-segmentdisplaysTL543(**U8-U11**)ontheboard.Eachdisplayhas8-inputsSEG_A(Pin-7),SEG_B(Pin-6),SEG_C(Pin-4),SEG_D(Pin-2),SEG_E(Pin-1),SEG_F(Pin-9),SEG_G(Pin-10)andDP(Pin-5).Theremainingpinspin-3&8isCommonCathode.TheportlinesP0.28toP0.31areusedtoselectoneofthe4digitsasshowninthetablebelow.TheportlinesP0.16toP0.23areusedassegmentlinesfortheEIGHTdigitsthroughthe74HCT244buffer(**U6**).

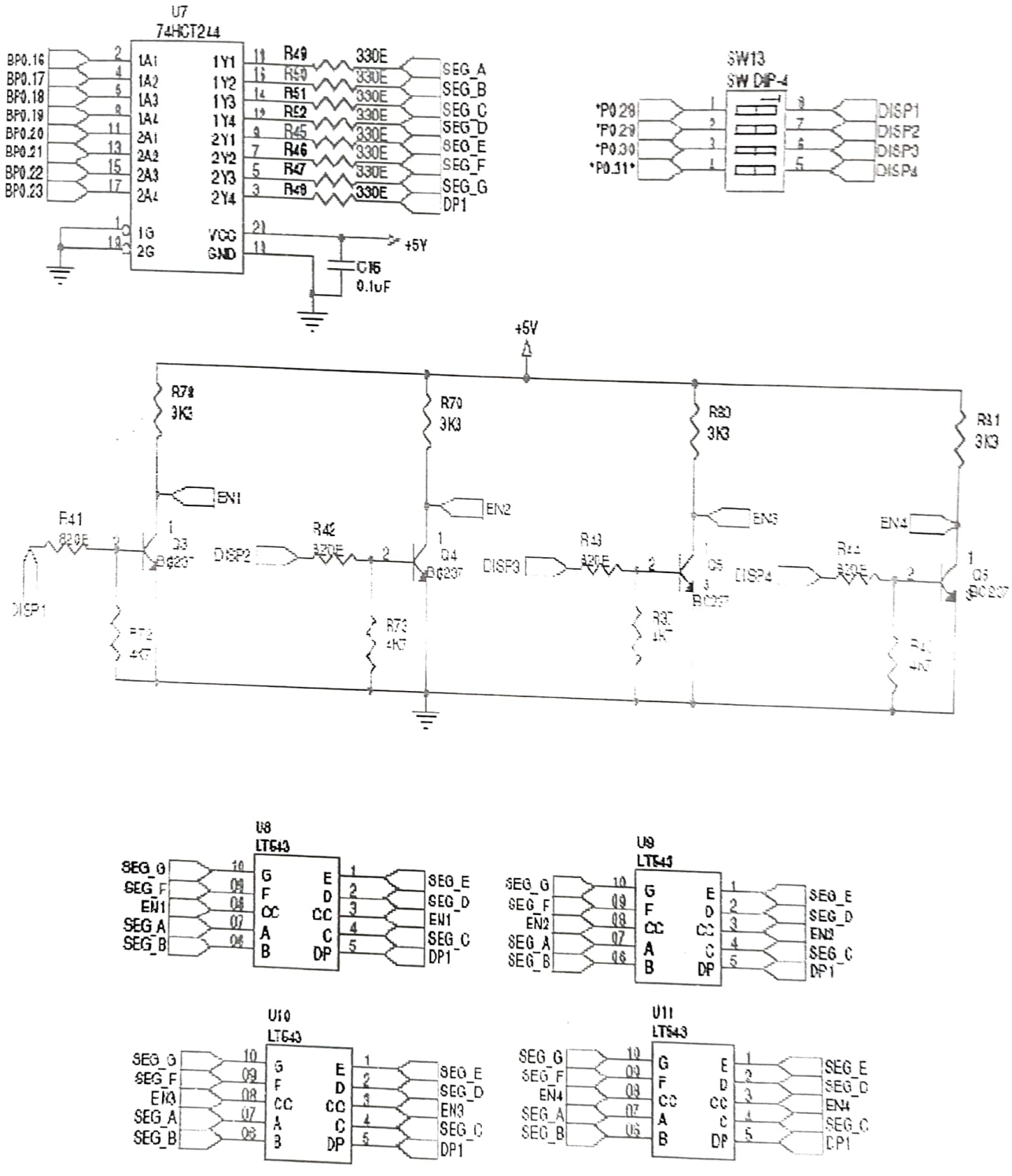
**Selection Of seven segment displays:**

| P0.28 | P0.29 | P0.30 | P0.31 | Displayunitselected |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | U |
| 0 | 1 | 0 | 0 | U |
| 0 | 0 | 1 | 0 | U10 |
| 0 | 0 | 0 | 1 | U11 |

7-segment LED should be identified as common cathode from board manual and 7-segment LED modules can be turned on or off with U8 to U11 through Hardware (DIP switches or through coding by pumping appropriate values to P0.28 to P0.31 by making them as GPIO by configuring PINSEL1)

Concept of Scanning through 7-Segment has to be understood and then implemented.

Hint: 100/sec to be a good number for the display update rate. Going faster than required just burns up CPU cycles for no purpose. With a lower number like 50-80/sec, but then some people may start to notice flicker, especially in certain lighting conditions. 100/sec and run the numbers. To display all 4 digits 100 times/sec means we display each digit in turn for 1/400 of a second or every 2.5ms. For 6 digits the number is 1/600 of a second or every 1.67ms.

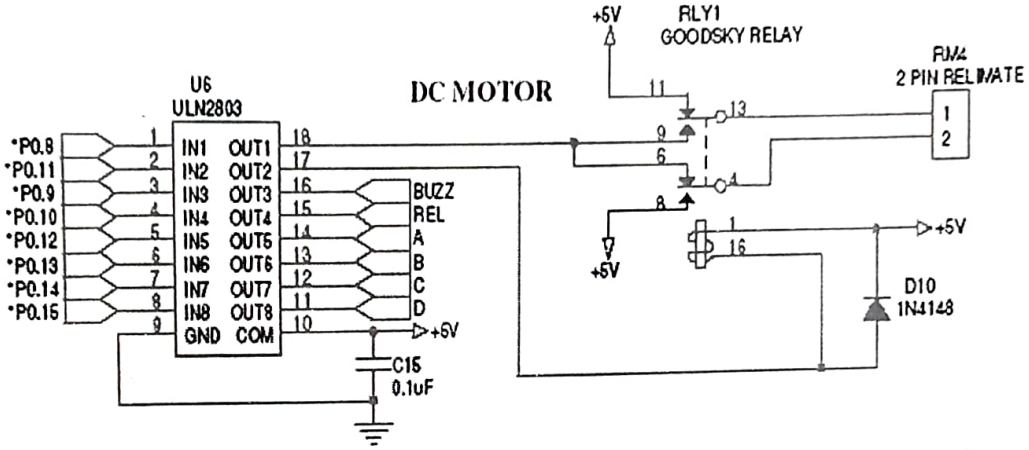



## 3. Write the C Program

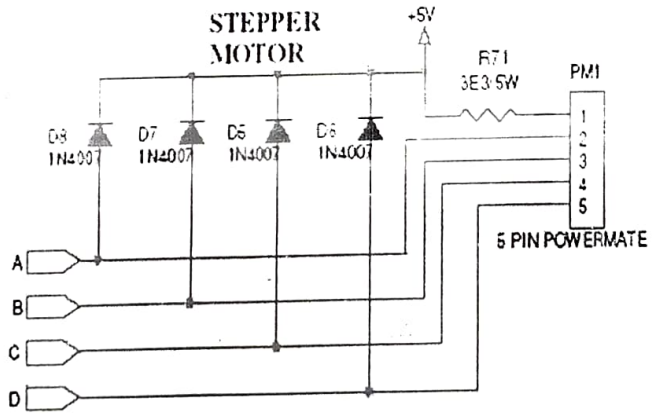
### a. To rotate DC and Stepper Motor

TheDCMotorcanalsobeinterfacedtotheboardbyconnectingittotheReliamate

RM4.Thedirectionoftherotationcanbechangedthroughsoftwareusing RelayR **LY1**.PortlinesusedforDCmotorareP0.8andP0.11.




The Stepper motor can be interfaced to the board by connecting it into the Power Mate PM1. The rotating direction of the stepper motor can be changed through software. Port lines used for Stepper motor are P0.12 – P0.15.




**For Clockwise**

```
voidclock_wise(void)
 {
       var1 = 0x00000800;              //For Clockwise
   for(i=0;i<=3;i++)                 // for A B C D Stepping
       {
             var1 = var1<<1;           //For Clockwise
      var2 = ~var1;
      var2 = var2 & 0x0000F000;

             IO0PIN = ~var1;

      for(k=0;k<3000;k++);       //for step speed variation
         }
```

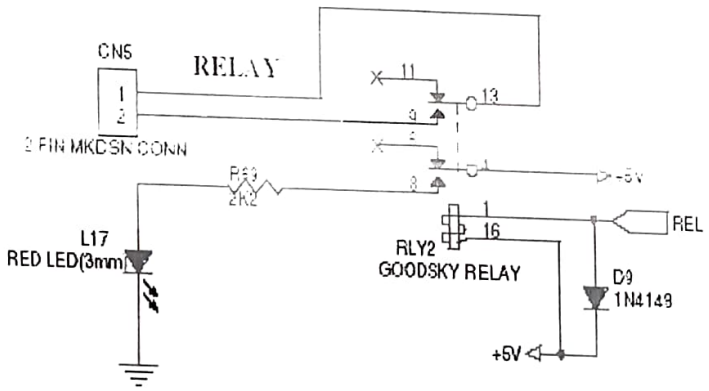
**For Anticclockwise**

```
void anti_clock_wise(void)
 {
      var1 = 0x00010000;        //For Anticlockwise
   for(i=0;i<=3;i++)        // for A B C D Stepping
    {
     var1 = var1>>1;       //For Anticlockwise
     var2 = ~var1;
     var2 = var2 & 0x0000F000;

            IO0PIN = ~var1;
     for(k=0;k<3000;k++);       //for step speed variation

    }
```

**b. To toggle the relay, based on key press**

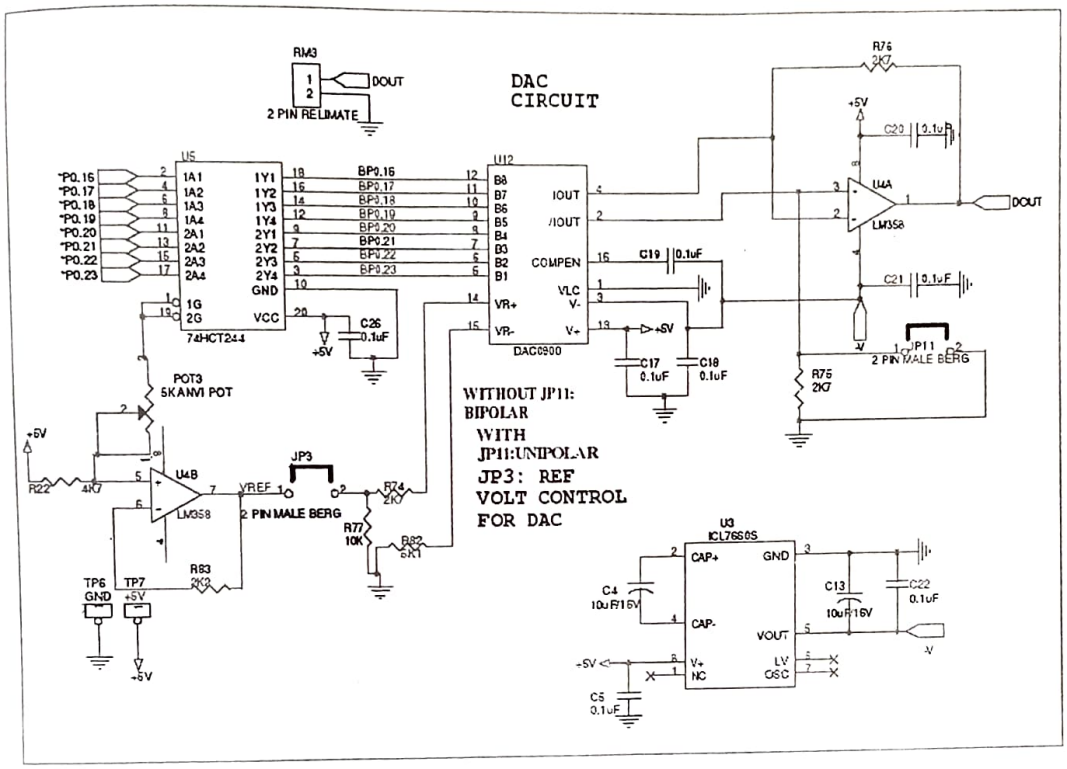



```
IO0DIR = 0x00000400;                    //Set P0.10 as output
IO0SET = 0x00000400;                    //P0.10 is set to a HI
```

Further setting or clearing P0.10 will toggle the relay.

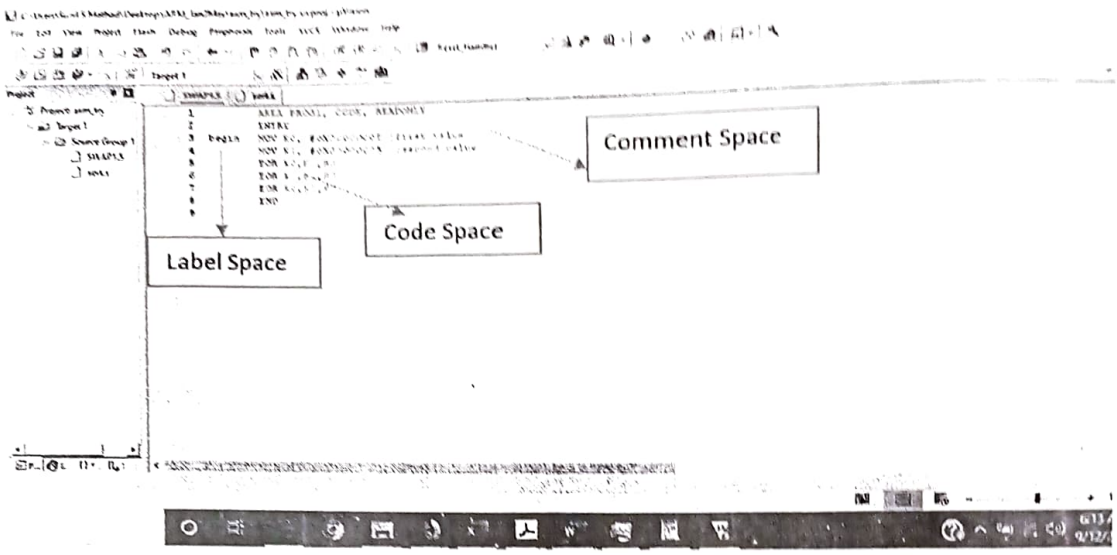
**4. Write the C- Program to generate the following a. Sine wave b. Square wave c. Ramp and d. Triangular wave.**

**DAC0800**isusedtoconvertthedigitaldataintoanalogsignals.Digitaldatafromspeci fiedportlinesisgiventoDACinput.Amplitudeofoutputwaveformcanbe variedbyvarying**POT**3(5KPot)thatisbyvaryingthe referencevoltageofDAC0800whenJP3isclosed. For**Bipolar**modeopen jumperJP11.For **Unipolar**modeshort jumperJP11.Portlines usedforDACareP0.16–P0.23.




Configure P0.0 to P0.15 as GPIO.
For sine wave use the formula 127+128 Sin$\Theta$ and form a lookup table which has to be fed to DAC continuously. Square wave can be generated by toggling Ox00 to 0XFF alternatively with equal delay (with 50% duty cycle) supplying it to DAC. Similarly Saw tooth with incremental steps from 0X00 to 0XFF continuously and triangular incremental steps from 0X00 to 0Xff and then decremental steps from 0xff to 0x00
continuously.

**5.** Swap 2 register contents with and without using temporary register.
For all Assembly Programs startup.s is not necessary unless there is a need to alter the status of interfacing circuitry. In Assembly Level Programming (ALP) after compilation an .axf ( arm executable file will be built with build report indicating number of code bytes, data bytes and Zero-initialised bytes. Further in linker Read Only and Read Write locations have to be different and only one source file should have ENTRY directive if there are multiple source files. Scatter file changes have to be noted when project is built.




Assembly codes will be having ".s" extension and editing space is virtually divided as shown above. The code for swapping register contents with out using another register

```
        AREA PROG1, CODE, READONLY
START   ENTRY
begin   MOV R0, #0X00000008     ;first value
        MOV R1, #0X00000005     ;second value
        EOR R0,R1,R0
        EOR R1,R1,R0
stop    EOR R0,R1,R0
        END
```

6. Implement the given expressions using ARM instruction set $6x^2 - 9x - 12$.
   The program targets at usage of Arithmetic instructions along with fetching and storing of data from main memory using LDR and STR instructions.
7. Perform32 bit multiplication, producing a 64-bit result, using only **UMULL** and logical operations.
   As program is indicating it targets usage of UMULL (unsigned long multiplication). Further fetching the operands and storing the result is to be demonstrated.

8. **Add 128-bit** numbers together, placing the result in registers r0, r1, r2, and r3. The first operand should be placed in registers r4, r5, r6, and r7, and the second operand should be in registers r8, r9, r10, and r11.
   The order of addition and carry management is highlighted.
9. Conversion of **upper case to lower case** and vice versa.
   Usage of Relational instruction set majorly CMP and Conditional execution has to be utilised.
10. Routine that reverses the bits in a register

This program targets the usage of shift and rotate instruction. Usage of RRX and LSL are to be explored. Reverse the bits in another register then check for equality with original data if they are same its palindrome.

11. Find the maximum value in a list of 32-bit numbers(two's complement representations) located in memory
Usage of LDR and STR with addressing modes (pre-indexed, pre-indexed with write back, post indexed) with within array boundary access through element count and usage CMP ,TEQ can be tested.
Further different sorting algorithms ( bubble sort, bucket sort) and binary search can be explored.

**A Quick Tour of the IDE**

The full documentation for the µVision IDE can be found at the following URL:
http://www.keil.com/product/brochures/uv4.pdf
That document is over 180 pages long, and most of it is very introductory in nature. It also covers multiple different processor types that are not used in our labs. You should not need to refer to that document.The IDE consists of a number of elements

## Getting started with ARM LPC2148 using Keil uVision IDE

There are various development environments available in the market for ARM processors.
Some of these are mentioned below :

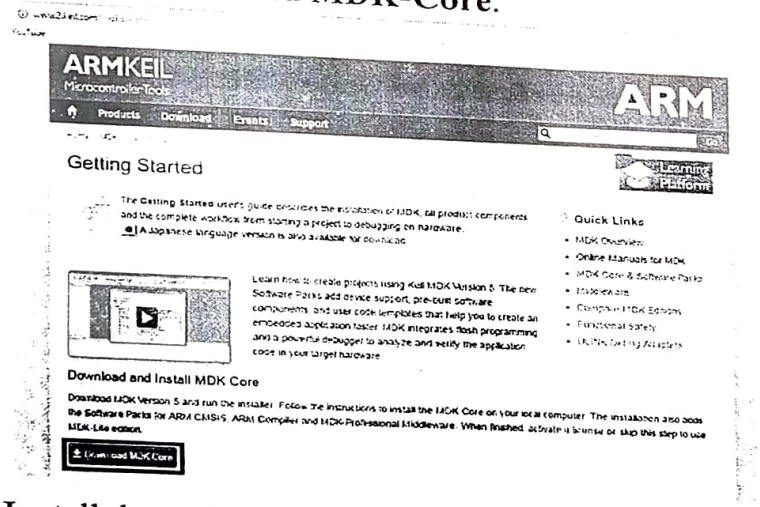
- CrossWorks for Arm
- Keil µVision
- IAR Embedded Workbench

We will see how to install and setup the µVision IDE by Keil.
We will see the steps that need to be followed for installing this software correctly.
When this is done, we will setup the environment for LPC2148 and write a basic code for LED blinking.

**Downloading and installation**

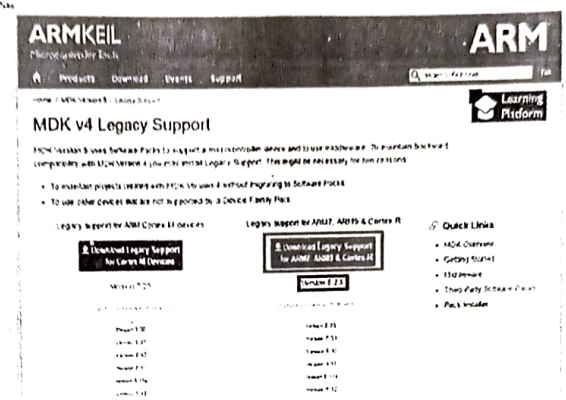
Follow the steps given below:

1. Download the MDK-lite (Microcontroller Development Kit) by Keil from their website. Here is the link to the page from where you can download this : http://www2.keil.com/mdk5/install
Click on **Download MDK-Core**.




Install the software by following the simple instructions provided during installation process.

2. The new version µVision5 does not support many of the devices that were supported in the older versions yet. LPC2148 is one of the devices that are not supported. Hence, we need to add this device after successfully installing µVision5.
To do this, go to the following link and download the executable file for Legacy Support for ARM7, ARM9, and Cortex-R: http://www2.keil.com/mdk5/legacy
Download the Legacy support for the version of MDK downloaded and installed.

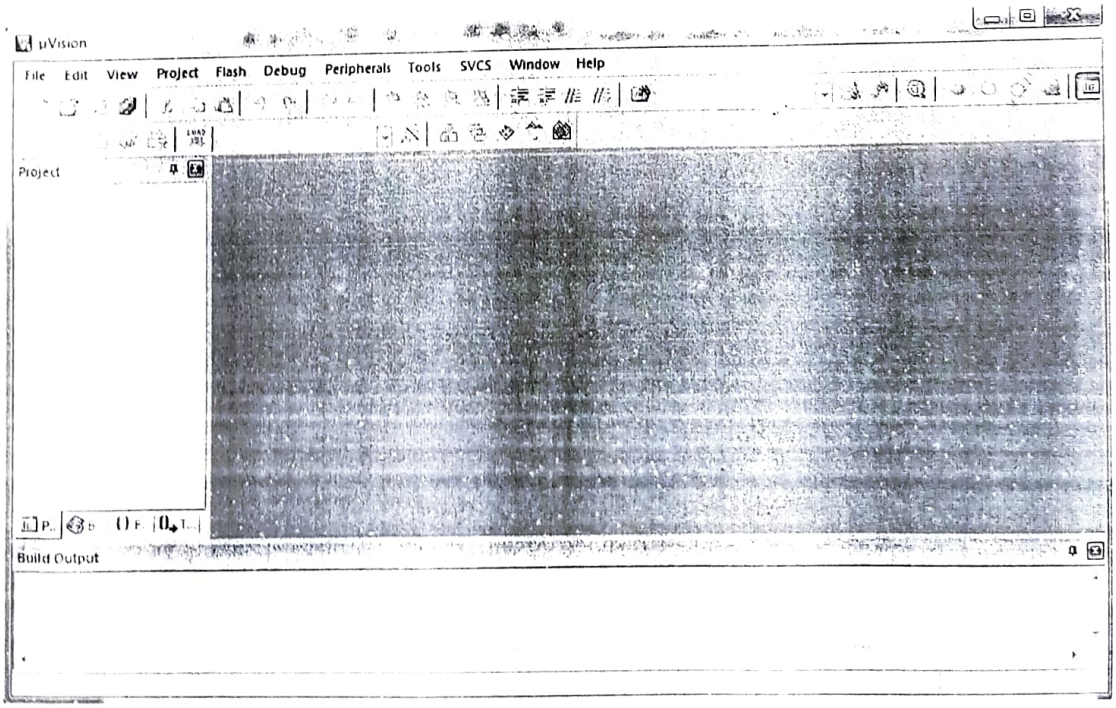



Install the executable file that will be downloaded. Follow the simple instructions provided during the installation process. When the above described steps are completed, we will have the IDE installed and ready to use with support for the device we intend to use, i.e. LPC2148.

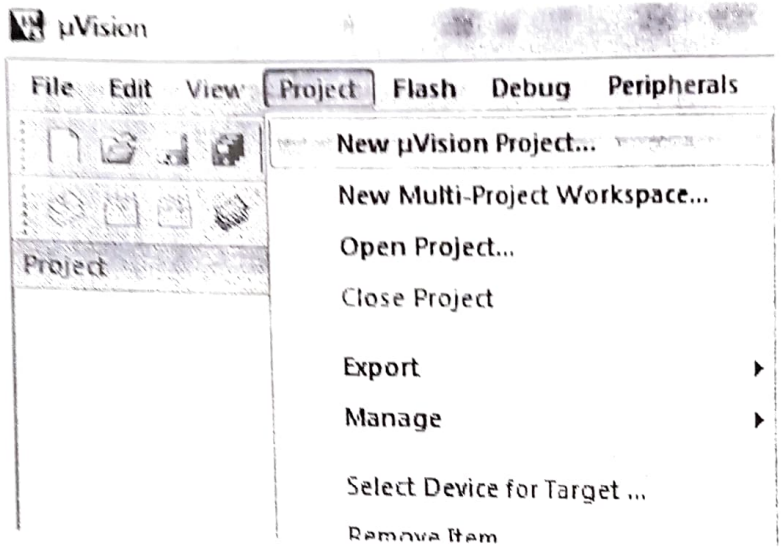
## Using µVision IDE

We will create a simple LED blinking project. Following are steps which show how to create and built project using the Keil uVision IDE:

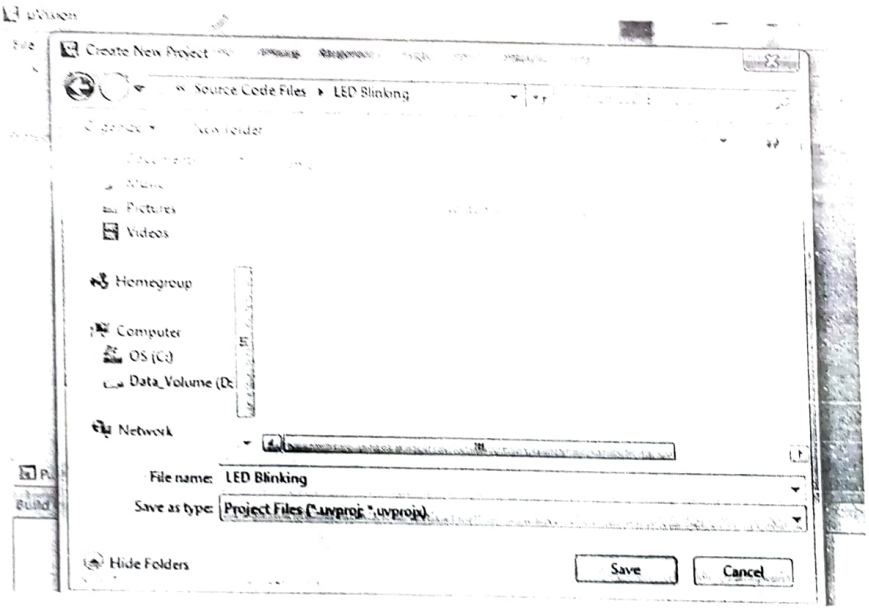
1. Open Keil µVision from the icon created on your desktop



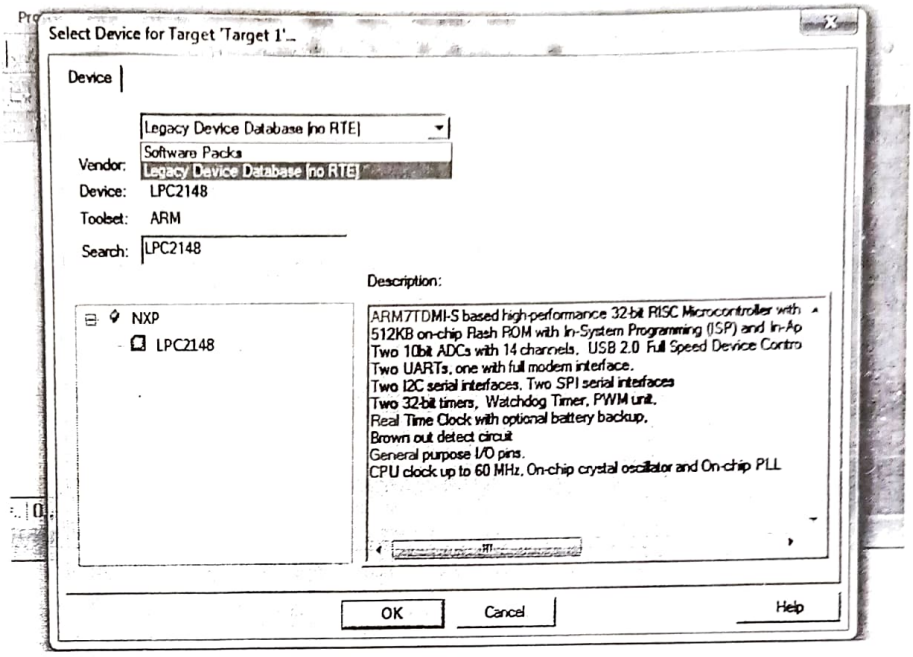

2. Go to the Project tab. Select New µVision Project ...from that menu.




3. Create New Project window will pop up. Select the folder where you want to create project and give a suitable name to the project. Then click on Save.

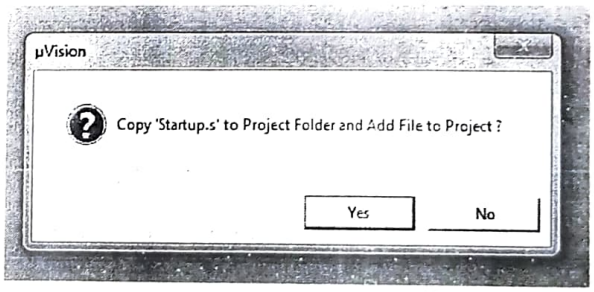



**4. Select Device for Target: 'Target1'...window will pop up next. It has a select window to choose between Software Packs or Legacy Device Database. As LPC2148 is in Legacy Device Database, choose Legacy Device Database.**

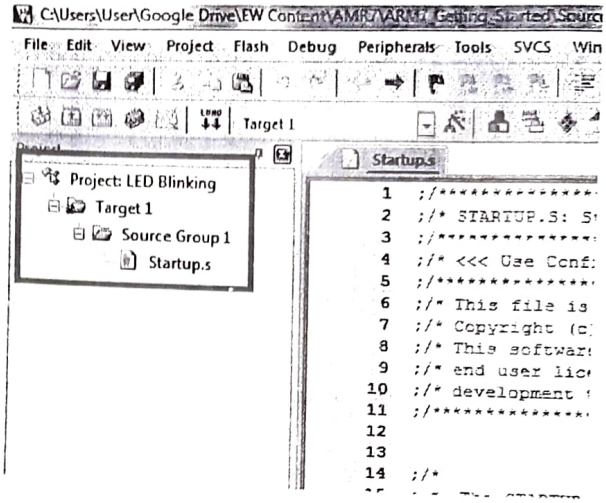



Type in LPC2148 in search and select the device under NXP with the name LPC2148 and click on OK.

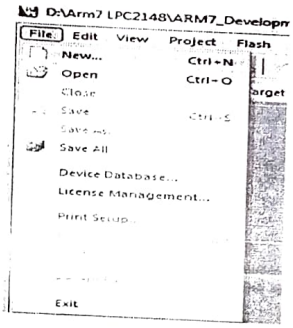
5. A window will pop up asking whether to copy Startup.s to project folder and add file to project. Click on **Yes**.



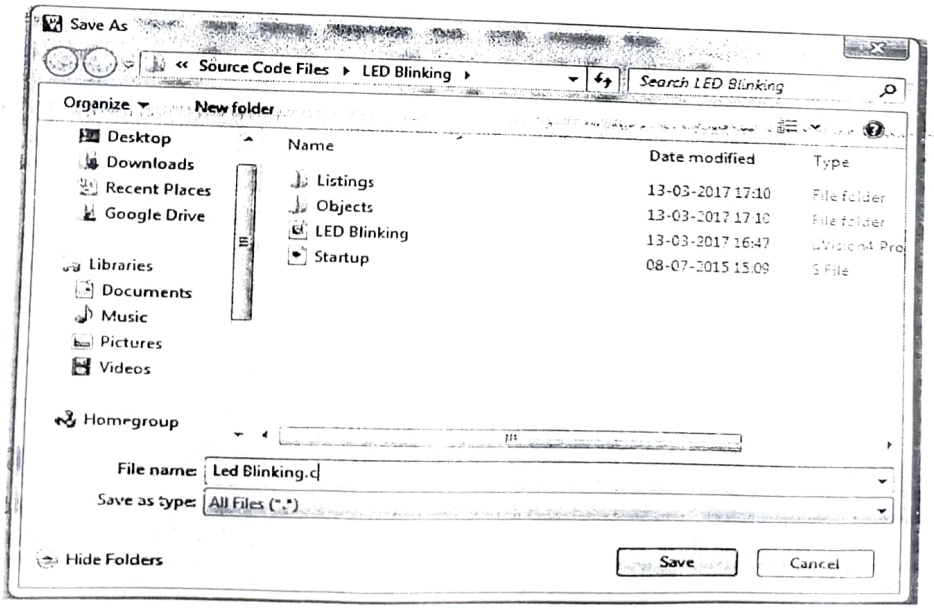

6. The project name and its folders can be seen on the left side in the project window after the previous step is completed as shown below.



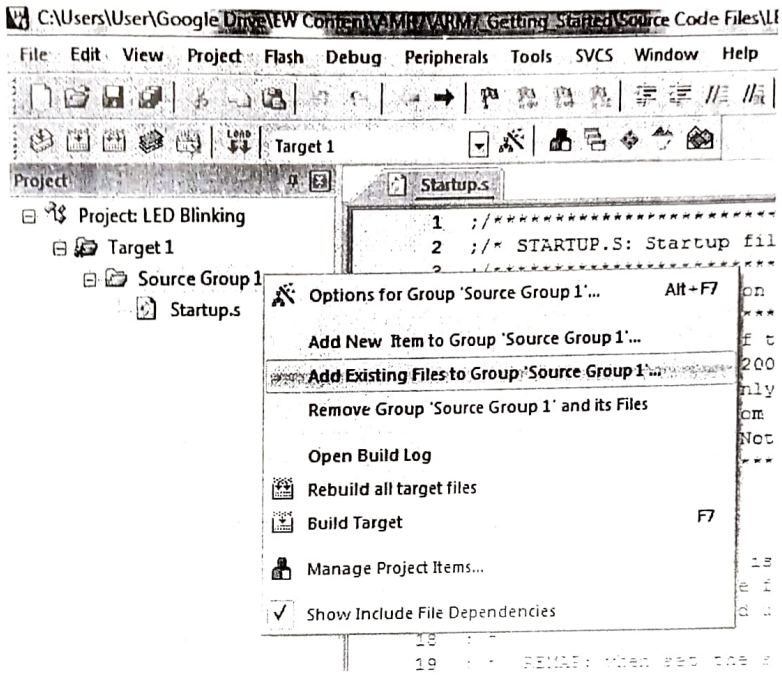

7. Now go to File tab and add **New** file from the menu.

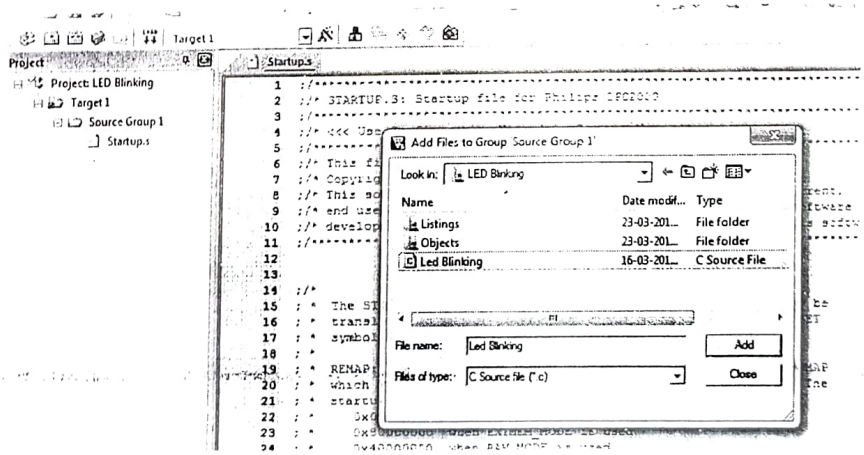



8. Save the file from the previous step with a specific name. Add .c extension to the file name.



9. Add this file to Source Group folder in the project window by right clicking on Source Group1 folder and selecting **Add Existing Files to Group 'Source Group1'...**

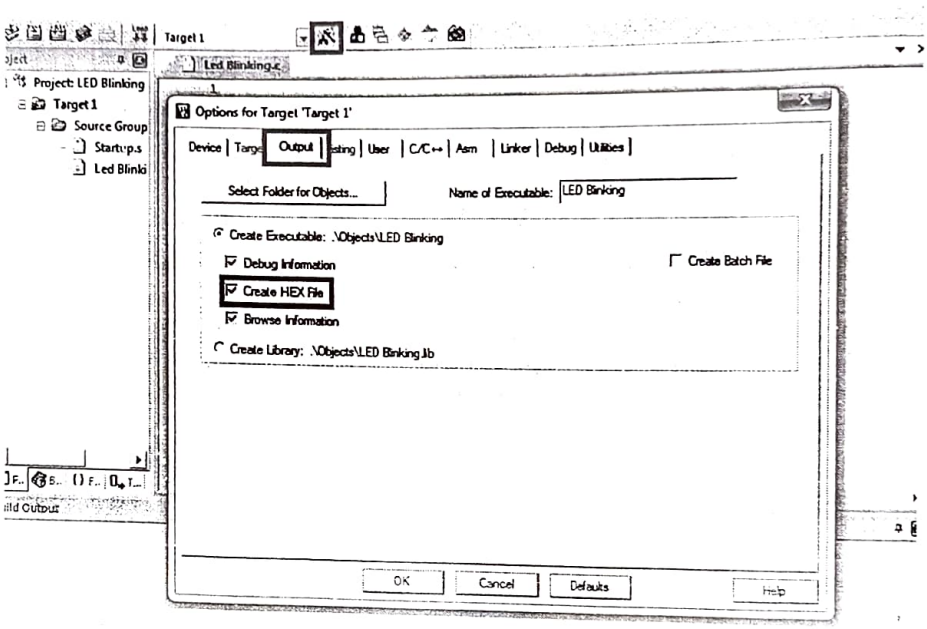


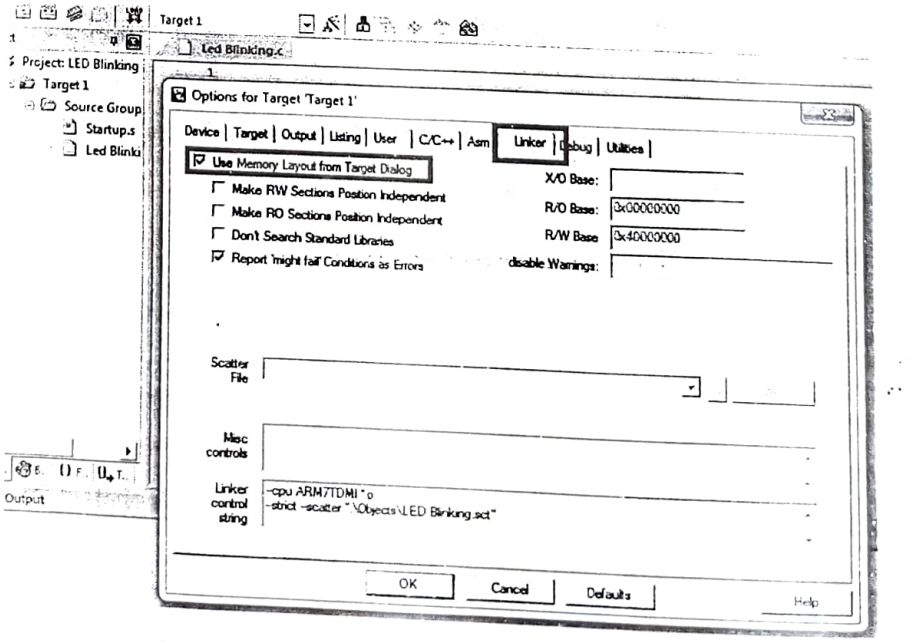

Select the previously saved file from the window that pops up and add it to the Source Group1. In our case, LED Blinking.c

10. Now click on the **Options for Target 'Target1'...** symbol shown in red box in the image below or press **Alt+F7** or right click on Target1 and click on **Options for Target 'Target1'....**

Options for target window will open. Go to the **Output** tab in that window. Tick '√' **Create HEX File** option. We need to produce HEX file to burn it into the microcontroller.




In the options for target window, go to the **Linker** tab. Select the **Use Memory Layout from Target Dialogue** option.




**Then click on OK.**

11. Now write the code for LED Blinking.

```
/*
  LED Blinking on LPC2148(ARM7)

*/

#include <lpc214x.h>
#include <stdint.h>

void delay_ms(uint16_t j) /* Function for delay in milliseconds
*/
{
    uint16_t x,i;
      for(i=0;i<j;i++)
      {
    for(x=0; x<6000; x++);
  /* loop to generate 1 millisecond delay with 12MHz Fosc. */
      }
}

int main(void)
{
      IO0DIR = 0x000000FF;
      /* Set P0.0 to P0.7 bits as output bits by writing 1 in
IO0DIR register corresponding to those bits. */
      while(1)

            IO0PIN =  IO0PIN | 0x000000FF;
 /* Make P0.0 to P0.7 HIGH while keeping other bits unchanged.
*/
            delay_ms(300);
            IO0PIN = IO0PIN & 0xFFFFFF00;
/* Make P0.0 to P0.7 LOW while keeoing other bits unchanged. */
            delay_ms(300);
      }
}
```

12. Once the code is written, **Build** the code by clicking on the button shown in red in the image below. You can also build the project from the **Build Target** option in the Project tab or by
13. pressing **F7** on the keyboard


You can see **creating hex file** ... in the Build Output window as shown in the image.

14. Once the project is built, a **hex file** is created in the **Objects** folder inside the folder of your project. Use **Flash Magic** software to burn this hex file in your microcontroller.

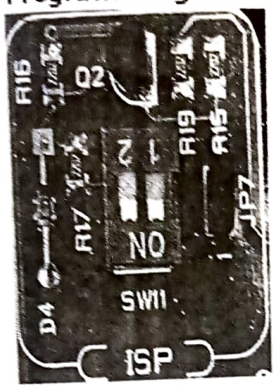
### Flash Magic

Download Flash Magic from the following Link : http://www.flashmagictool.com
After installation open the flash magic software and follow the below steps.

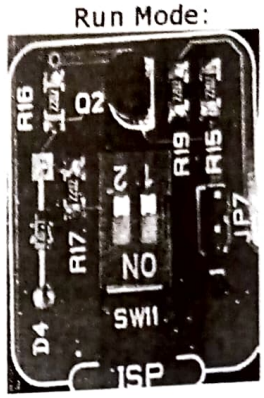
1. Select the IC from Select Menu.
2. Select the COM Port. **Check the device manager for detected Com port.**
3. Select Baud rate from 9600-115200
4. Select None Isp Option.
5. Oscillator Freq 12.000000(12Mhz).
6. Check the Erase blocks used by Hex file option
7. Browse and Select the hex file.
8. Check the Verify After Programming Option.
9. If DTR and RTS are used then go to Options->Advanced Options-> Hardware Config and select the Use DTR and RTS Option.

SW11: 2-WAY dip switch to control RTS & DTR lines for ISP. During programming the switches are kept ON and JP7 is also shorted. In RUN mode the switches are OFF and JP7 is open.

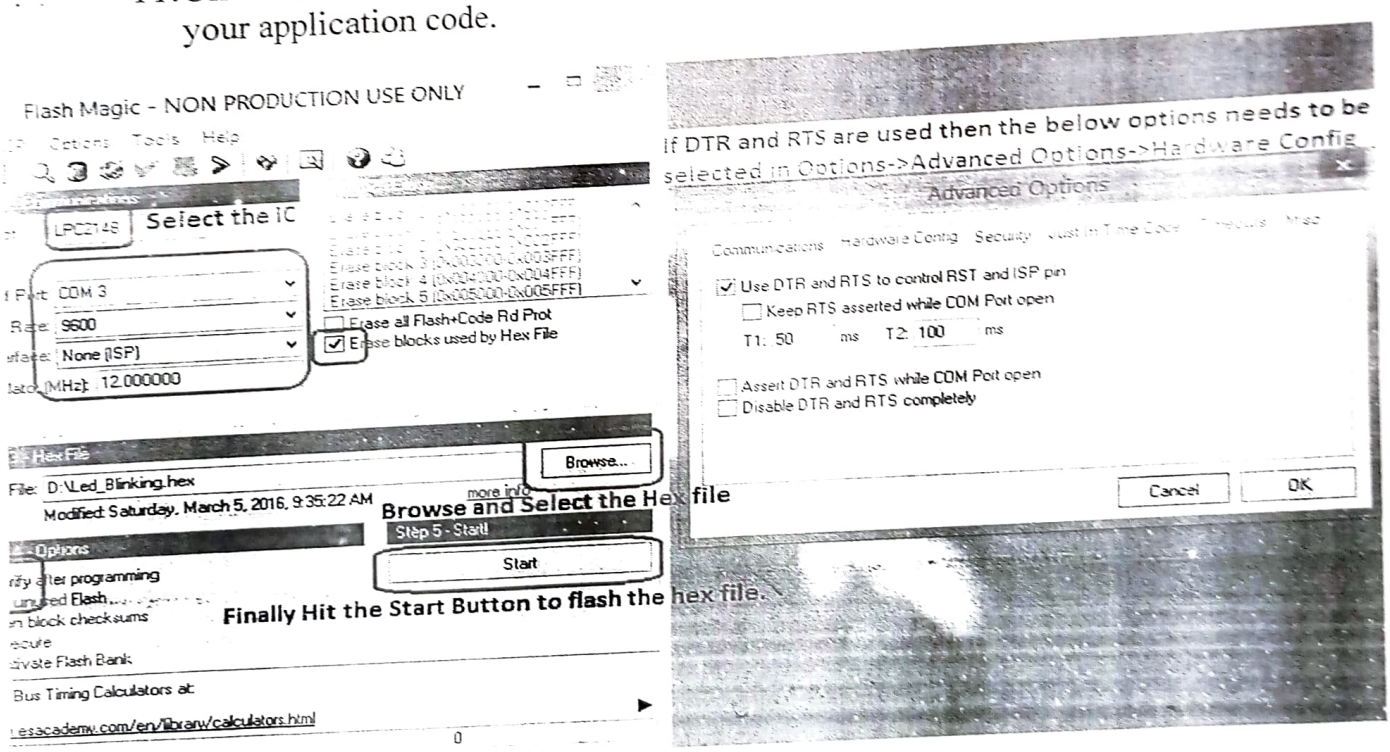
Programming Mode:




Run Mode:




10. Hit the Start Button to flash the hex file.
11. Once the hex file is flashed, Reset the board. Now the controller should run your application code.




Once successfully programmed expected behavior can be observed on board.