Dr. Ambedkar Institute of Technology, Bengaluru-56

(An Autonomous Institution Affiliated to VTU, Belagavi)

(Accredited by NAAC with Grade “A”)

Department of Electronics and Communication Engineering

VI Semester
2023-24

Computer Organization & ARM Microcontrollers (21ECT602) Laboratory Manual

Compiled & Prepared by:

Mr. Anand H. D.
Assistant Professor,
Dept. of ECE, Dr. AIT

Student Name: ______________________________

USN: ______________________________

Section: ______________________________

Batch: ______________________________

Dr. AMBEDKAR INSTITUTE OF TECHNOLOGY, BENGALURU-560056

(An Autonomous Institution Affiliated TO VTU, BELAGAVI and Accredited by NAAC with Grade "A")

Department of Electronics and Communication Engineering

Vision Statement:

"To excel in education and research in Electronics and Communication Engineering and its related areas through its integrated activities for the society"

Mission Statement:

•To provide the high quality education in Electronics and Communication Engineering discipline and its related areas to meet the growing challenges of the industry and the society through research.

•To be a contributor to technology through value based quality technical education.

•To equip the students with strong foundations of Electronics and communication engineering.

Program Educational Objectives (PEOs):

PEO1: Graduates will have a solid foundation in electronics and communication engineering.

PEO2: Graduates are technically competent and able to analyze, design, develop and implement electronic and communication systems.

PEO3: Graduates will have sufficient breadth in electronics and its related fields so as to enable them to solve general engineering problems.

PEO4: Graduates are capable of communicating effectively and interact professionally with colleagues, clients, employers and the society.

PEO5: Graduates are capable of engaging in life - long learning and to keep themselves abreast of new developments in their fields of practice.

Program Outcomes (POs):

PO1:Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2:Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

PO3:Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4:Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

PO5:Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations

PO6:The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7:Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development

PO8:Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9:Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10:Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions

PO11:Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12:Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Syllabus

Subject Title : COMPUTER ORGANIZATION & ARM MICROCONTROLLERS

Course Learning Objectives:

1. to explain the basic organization of a computer system.
2. to understand functioning of different sub systems, such as processor, Input/output, and memory.
3. to describe the architectural features and instructions of 32-bit microcontroller ARM Cortex M3.
4. to Program ARM Cortex M3 for different applications.
5. to analyse the Thumb instruction set and different C-Programming concepts.

Unit No	Syllabus Contents	No. of Hours	Blooms Taxonomy level.
01.	Write an ALP to find the sum of first 10 integer numbers.	2	L1, L2, L3
02.	Write an ALP to calculate the value of the polynomial function.	2	L1, L2, L3
03.	Write an ALP to store data in desired Memory location.	2	L1, L2, L3
04.	Write a C program to Output the message using UART of LPC1768.	2	L1, L2, L3, L4
05.	Write a C Program to interface LED using LPC 1768.	2	L1, L2, L3, L4
06.	Write a C Program to interface Relay using LPC 1768.	2	L1, L2, L3, L4
07.	Write a C Program for DC motor/Stepper motor rotation using LPC 1768.	2	L1, L2, L3, L4
08.	Write a C Program to interface a DAC and generate Triangular and Square waveforms.	2	L1, L2, L3, L4
09.	Write a C program to demonstrate the use of an External interrupt in LPC 1768	2	L1, L2, L3, L4
Demonstration Experiments (For CIE only not for SEE)			
10.	Write a C program to interface a Real Time Clock (RTC) of LPC 1768.	2	L1, L2, L3, L4
11.	Write a program to read on-chip ADC value and display it on UART terminal using LPC 1768	2	L1, L2, L3, L4
12.	Write a C program to interface Keypad using LPC 1768.	2	L1, L2, L3, L4

Course Outcomes:

CO 1: Understand the basic structure and Input output organization of a computer system.

CO 2: Explain functioning of different sub systems, such as processor, Input/output, and memory.

CO 3: Describe the architectural features and instructions of 32-bit microcontroller ARM Cortex M3.

CO 4: Apply the instruction set to Program ARM Cortex M3 for different applications.

CO5: Analyse Thumb Instruction set and C-Programming Concepts to Program ARM Cortex M3.

Text Books:

1. Andrew N Sloss, Dominic System and Chris Wright, "ARM System Developers Guide", Elsevier, Morgan Kaufman publisher, 1st Edition, 2008

Video Links:

https://www.youtube.com/watch?v=bFT70WbBRrs&list=PLnFckqm074doPiUQkkhwpHzP0JI7tAg4p

--------0--------

Dos and Don‛ts in Laboratory

1. Wear your College ID card.
2. Be regular to the Lab.
3. Keep your bags in the rack.
4. Take care of your valuable things.
5. Keep your work area clean.
6. Bring Observation book, Record and Manual along with pen, pencil, and eraser Etc., no borrowing from others.
7. Before entering to lab, must prepare for Viva for which they are going to conduct experiment.
8. Check CRO probe (if using) before connecting it.
9. Strictly follow the instructions given by the teacher/Lab Instructor.
10. Before switching on the power supply, must show the connections to one of the faculties or instructors.
11. Remove the Connections and return the components to the respective lab instructors.
12. Before leaving the lab, switch off the power supplies of all equipments and arrange chairs properly.
13. Do not come late to the Lab.
14. Do not handle any equipment before reading the instructions/Instruction manuals.
15. Avoid loose connection and short circuits.
16. Do not panic if you do not get the output.
17. Do not throw connecting wires on the Floor.

Record Writing Format

Unruled Side	***Ruled side***
Diagram	Date
Design	Title
Observation table and Entries	Aim
Expected Graph/ Response	Components required
	Theory
	Procedure
	Result

Rubrics for Evaluation

Each program/ experiment needs to evaluated for 30 marks based on following 6 categories (**Introduction, Experimental Hypothesis, Procedure, Data representation, Analysis** and **Conclusion**) on a scale of 0 - 5 marks as stated in following table

Category ↓	Poor (0-1 Mark)	Fair (2 Marks)	Good (3 Marks)	Excellent (4 Marks)	Outstanding (5 Marks)
Introduction	There is no introduction OR introduction is abrupt	Introduction states the purpose of the lab, but not the variables that will be studied.	Introduction states the purpose of the lab and the variables to be studied.	Introduction clearly states the purpose of the lab and explicitly state the variables that are to be studied.	Introduction clearly states the purpose of the lab, state the variables that are to be studied and related Theory required.
Experimental Hypothesis	No hypothesis has been stated.	Hypothesized relationship between the variables and the predicted results are not clear	Hypothesized relationship between the variables and the predicted results has been stated, but appears to be based on flawed logic.	Hypothesized relationship between the variables and the predicted results is reasonable based on general knowledge	Hypothesized relationship between the variables and the predicted results is clear and reasonable based on what has been studied and observations
Procedure	Lacks the appropriate knowledge of the lab procedures. Procedures and steps of the experiment not written	Procedure do not accurately written according to the steps of the experiment.	Procedure is written but it is not in a logical order or it is difficult to follow	Procedure is written in a logical order, but steps are not numbered and/or are not in complete sentences	Procedures are listed in clear steps. Each step is numbered and is a complete sentence.
Data Representation	Adequate Data are not shown OR are inaccurate.	Accurate representation of the data in written form, but no graphs or tables are presented.	Accurate representation of the data in tables and/or graphs. Graphs and tables are labeled and titled.	Accurate representation of the data in tables and/or graphs. Graphs and tables are labeled and titled. Drawings are included when necessary.	Professional looking and accurate representation of the data in tables and/or graphs. Graphs and tables are labeled and titled. Drawings are included as necessary and are well labeled.

Cont…

	Poor (0-1 Mark)	Fair (2 Marks)	Good (3 Marks)	Excellent (4 Marks)	Outstanding (5 Marks)
Data Representation	Adequate Data are not shown OR Shown inaccurate data.	Accurate representation of the data in written form, but no graphs or tables are presented.	Accurate representation of the data in tables and/or graphs. Graphs and tables are labeled and titled.	Accurate representation of the data in tables and/or graphs. Graphs and tables are labeled and titled. Drawings are included when necessary.	Professional looking and accurate representation of the data in tables and/or graphs. Graphs and tables are labeled and titled. Drawings are included as necessary and are well labeled.
Analysis	Relationship between the variables is not analyzed.	Relationship between the variables is discussed but no patterns, trends or predictions are made based on the data.	Relationship between the variables is discussed and trends /patterns logically analyzed.	Relationship between the variables is discussed and trends/pattern s logically analyzed but outcomes not summarized	The relationship between the variables is discussed and trends/patterns are logically analyzed and the outcomes of analysis is clearly mentioned
Conclusion	No conclusion was included in the report OR shows little effort and reflection.	Conclusion written but not clear as it doesn't includes what was learned from the experiment.	Conclusion is accurate and includes what was learned from the experiment.	Conclusion is accurate and includes whether the findings supported the hypothesis and what was learned from the experiment	Conclusion is accurate and includes whether the findings supported the hypothesis, possible sources of error, and what was learned from the experiment

Final Internal Marks Evaluation
(Evaluated for 50 Marks)

30 Marks is for Laboratory Record (average of marks scored in individual experiment/program based on above rubrics)

Remaining **20 Marks** is divided as **05 Marks** for Write-up

10 Marks for Conduction

05 Marks for Viva-voce

ALL THE BEST!!!

CONTENTS

I.INTRODUCTION

The LPC-1768 is an ARM Cortex-M3 based microcontrollers for embedded applications featuring a high level of integration and low power consumption. The ARM Cortex-M3 is a next generation core that offers system enhancements such as enhanced debug features and a higher level of support block integration.

The LPC-1768 operate at CPU frequencies of up to 100 MHz. The LPC1768 operates at CPU frequencies of up to 120 MHz. The ARM Cortex-M3 CPU incorporates a 3-stage pipeline and uses a Harvard architecture with separate local instruction and data buses as well as a third bus for peripherals. The ARM Cortex-M3CPU also includes an internal prefetch unit that supports speculative branching.

The peripheral complement of the LPC-1768 includes up to 512 kB of flash memory, up to 64 kB of data memory, Ethernet MAC, USB Device/Host/OTG interface, 8-channel general purpose DMA controller, 4 UARTs, 2 CAN channels, 2 SSP controllers, SPI interface, 3 I2C-bus interfaces, 2 input plus 2-output I2S-bus interface, 8-channel 12-bit ADC, 10-bit DAC, motor control PWM, Quadrature Encoder interface, four general purpose timers, 6-output general purpose PWM, ultra-low power Real-Time Clock (RTC) with separate battery supply, and up to 70 general purpose I/O pins. The LPCLPC-1768 are pin-compatible to the 100-pin LPC236x ARM7-based microcontroller series.

Features:

• ARM Cortex-M3 processor, running at frequencies of up to 100 MHz (LPC-1768) A. Memory Protection Unit (MPU) supporting eight regions is included.

•ARM Cortex-M3 built-in Nested Vectored Interrupt Controller (NVIC).

•Up to 512 kB on-chip flash programming memory. Enhanced flash memory accelerator enables high-speed 120 MHz operation with zero wait states.

In-System Programming (ISP) and In-Application Programming (IAP) via on-chip boot loader software.

On-chip SRAM includes:

➢32/16 kB of SRAM on the CPU with local code/data bus for high-performance CPU access

➢Two/one 16 kB SRAM blocks with separate access paths for higher throughput. These SRAM blocks may be used for Ethernet, USB, and DMA memory, as well as for general purpose CPU instruction and data storage.

➢Eight channel General Purpose DMA controller (GPDMA) on the AHB multilayer matrix that can be used with SSP, I2S-bus, UART, Analog-to-Digital and Digital-to-Analog converter peripherals, timer match signals, and for memory-to-memory transfers.

➢Multilayer AHB matrixes interconnect provides a separate bus for each AHB master.AHB masters include the CPU, General Purpose DMA controller, Ethernet MAC, and the USB interface. This interconnect provides communication with no arbitration delays.

➢Split APB bus allows high throughput with few stalls between the CPU and DMA.

Serial interfaces:

➢Ethernet MAC with RMII interface and dedicated DMA controller. USB 2.0 full-speed device/Host/OTG controller with dedicated DMA controller and on-chip PHY for device, Host, and OTG functions.

➢Four UARTs with fractional baud rate generation, internal FIFO, and DMA support.

➢One UART has modem control I/O and RS-485/EIA-485 support, and one UART has IrDA support.

➢CAN 2.0B controller with two channels.

➢SPI controller with synchronous, serial, full duplex communication and programmable data length.

➢Two SSP controllers with FIFO and multi-protocol capabilities. The SSP interfaces can be used with the GPDMA controller.

➢Three enhanced I2C bus interfaces, one with an open-drain output supporting full I2C specification and Fast mode plus with data rates of 1 Mbit/s, two with standard port pins. Enhancements include multiple address recognition and monitor mode.

➢I2S (Inter-IC Sound) interface for digital audio input or output, with fractional rate control. The I2S-bus interface can be used with the GPDMA. The I2S-bus interface Supports 3-wire and 4-wire data transmit and receive as well as master clock input/output.

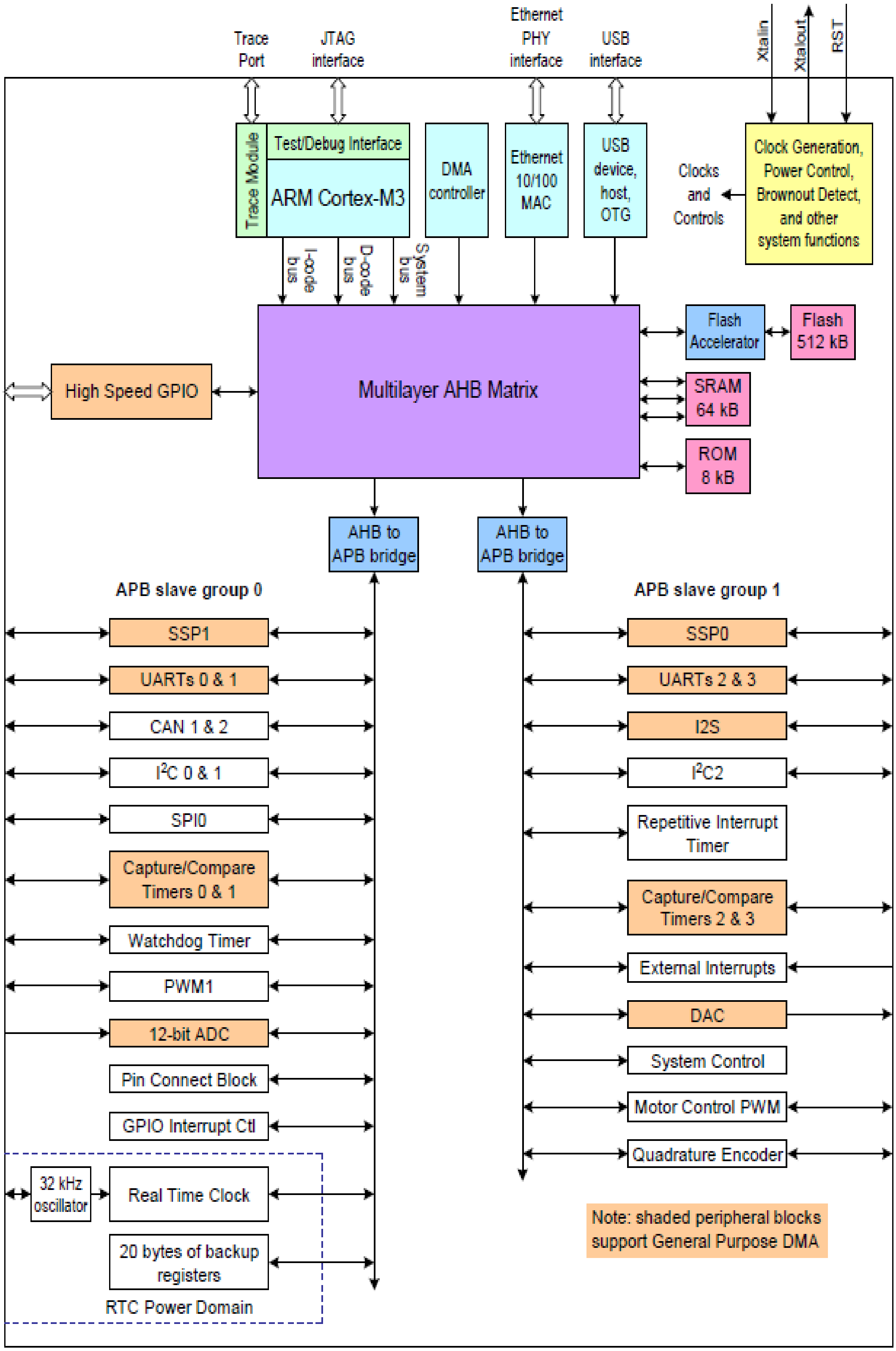


Fig 1. LPC1768 simplified block diagram

Other peripherals:

➢70 (100 pin package) General Purpose I/O (GPIO) pins with configurable pull-up/down resistors. All GPIOs support a new, configurable open-drain operating mode. The GPIO block is accessed through the AHB multilayer bus for fast access and located in memory such

➢that it supports Cortex-M3 bit banding and use by the General Purpose DMA Controller.

➢12-bit Analog-to-Digital Converter (ADC) with input multiplexing among eight pins, conversion rates up to 200 kHz, and multiple result registers. The 12-bit ADC can be used with the GPDMA controller.

➢10-bit Digital-to-Analog Converter (DAC) with dedicated conversion timer and DMA support.

➢Four general purpose timers/counters, with a total of eight capture inputs and ten compare outputs. Each timer block has an external count input. Specific timer events can be selected to generate DMA requests.

➢One motor control PWM with support for three-phase motor control

➢Quadrature encoder interface that can monitor one external quadrature encoder.

➢One standard PWM/timer block with external count input.

➢RTC with a separate power domain and dedicated RTC oscillator. The RTC block includes 20 bytes of battery-powered backup registers.

➢WatchDog Timer (WDT). The WDT can be clocked from the internal RC oscillator, the RTC oscillator, or the APB clock.

➢ARM Cortex-M3 system tick timer, including an external clock input option.

➢Repetitive interrupt timer provides programmable and repeating timed interrupts.

➢Each peripheral has its own clock divider for further power savings.

➢Standard JTAG test/debug interface for compatibility with existing tools. Serial Wire Debug and Serial Wire Trace Port options.

➢Emulation trace module enables non-intrusive, high-speed real-time tracing of instruction execution.

➢Integrated PMU (Power Management Unit) automatically adjusts internal regulators to minimize power consumption during Sleep, Deep sleep, Power-down, and Deep power-down modes.

➢Four reduced power modes: Sleep, Deep-sleep, Power-down, and Deep power-down.

- Single 3.3 V power supply (2.4 V to 3.6 V).
- Four external interrupt inputs configurable as edge/level sensitive. All pins on Port 0 and Port 2 can be used as edge sensitive interrupt sources.
- Non-maskable Interrupt (NMI) input.
- Clock output function that can reflect the main oscillator clock, IRC clock, RTC clock, CPU clock, and the USB clock.
- The Wake-up Interrupt Controller (WIC) allows the CPU to automatically wake up from any priority interrupt that can occur while the clocks are stopped in deep sleep, Power-down,
- and Deep power-down modes.
- Processor wake-up from Power-down mode via any interrupt able to operate during Power down mode (includes external interrupts, RTC interrupt, USB activity, Ethernet wake-up interrupt, CAN bus activity, Port 0/2 pin interrupt, and NMI).
- Brownout detect with separate threshold for interrupt and forced reset.
- Power-On Reset (POR).
- Crystal oscillator with an operating range of 1 MHz to 25 MHz.
- 4 MHz internal RC oscillator trimmed to 1 % accuracy that can optionally be used as a system clock.
- PLL allows CPU operation up to the maximum CPU rate without the need for a high frequency crystal. May be run from the main oscillator, the internal RC oscillator, or the RTC oscillator.
- USB PLL for added flexibility.
- Code Read Protection (CRP) with different security levels.
- Unique device serial number for identification purposes.
- Available as LQFP100 (14 mm x 14 mm x 1.4 mm), TFBGA1001 (9 mm x 9 mm x 0.7 mm), and WLCSP100 (5.074 x 5.074 x 0.6 mm) package.

II Create application using KEIL µVision5

Create Application Using µVision5.

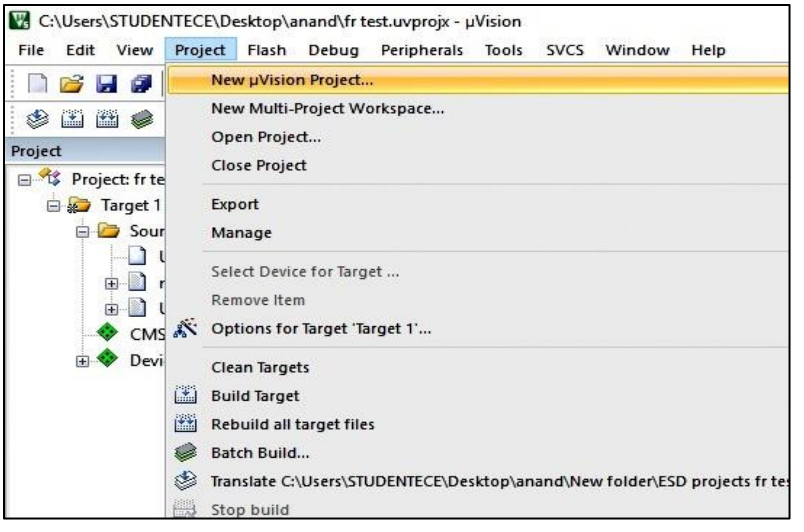
The required steps for creating application programs are listed below:

Step1.

Click on **Keil µVision5** shortcut available on the desktop.

Step2.

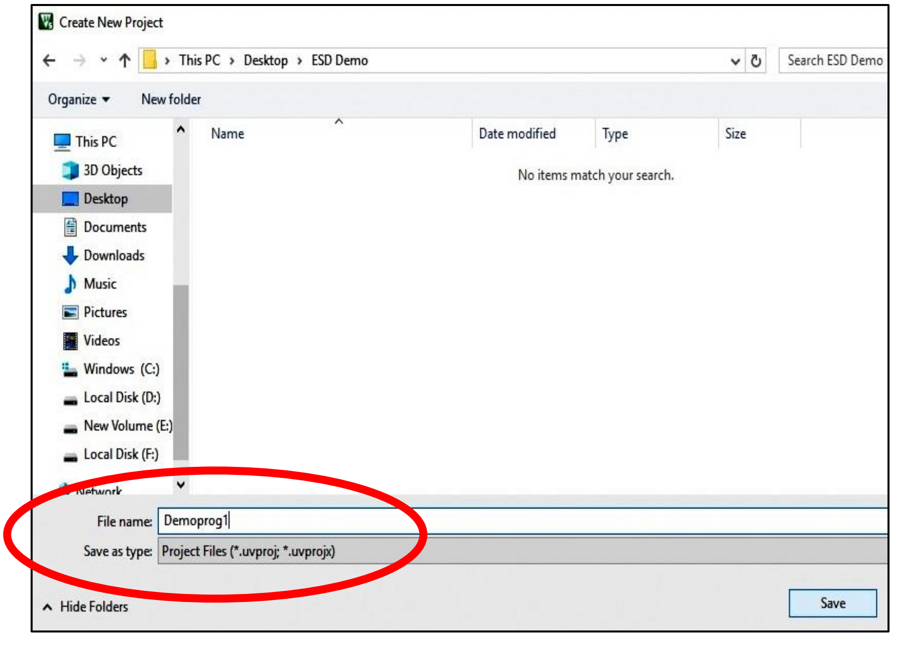
Select **Project – > New µVision Project** from the µVision5 menu.



Step 3.

This opens a standard Windows dialog, which prompts you for the new project file name.

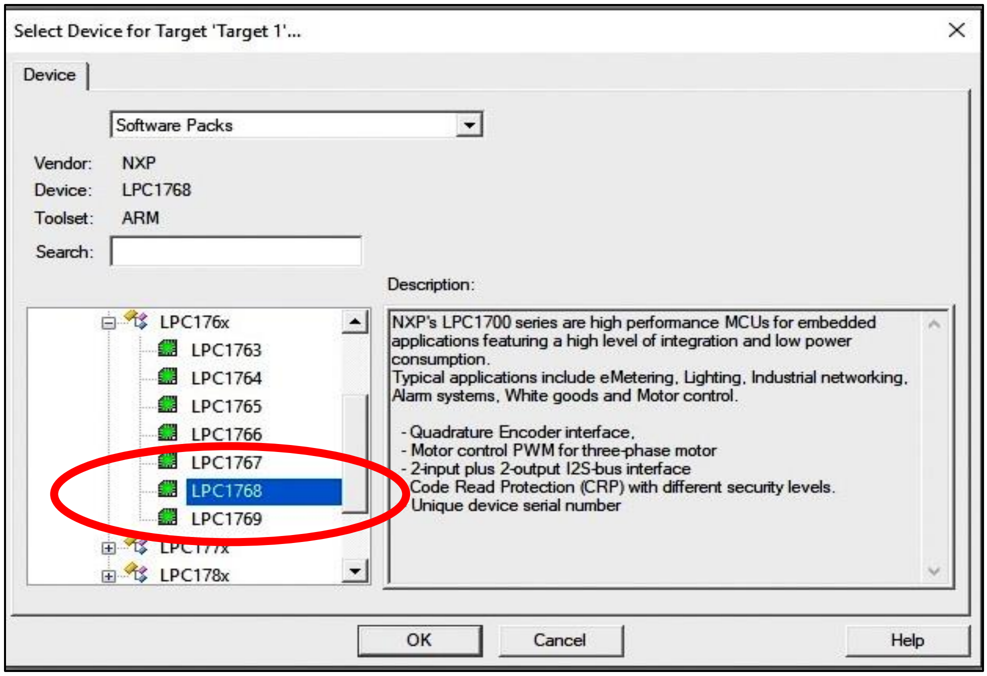
Create a new Project with user defined name (For Eg. Demoprog1) and press **Save**



Step 4.

Select Vendor as **NXP**, then

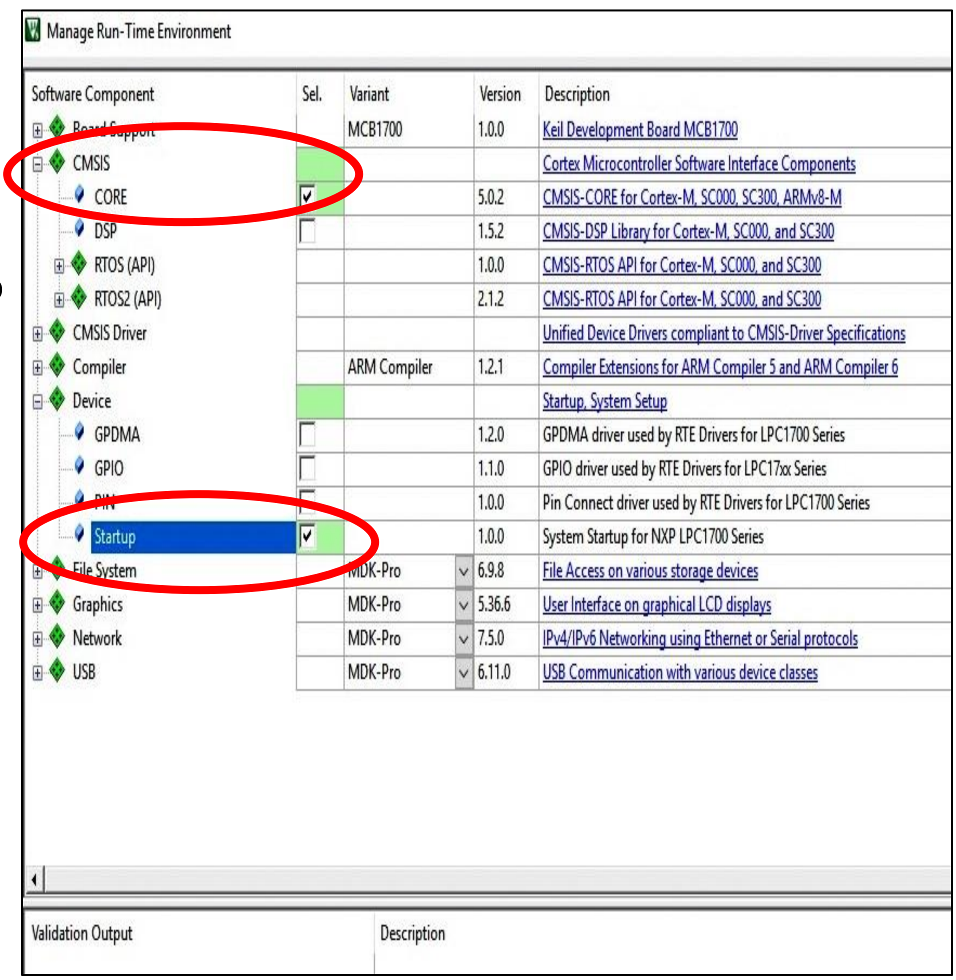
Device as **LPC176x -> LPC1768**



Step 5.

Select **CMSIS -> CORE** and **Device -> Startup**

Note: in case of Assembly Program select **CMSIS -> CORE** only **don't select Device**



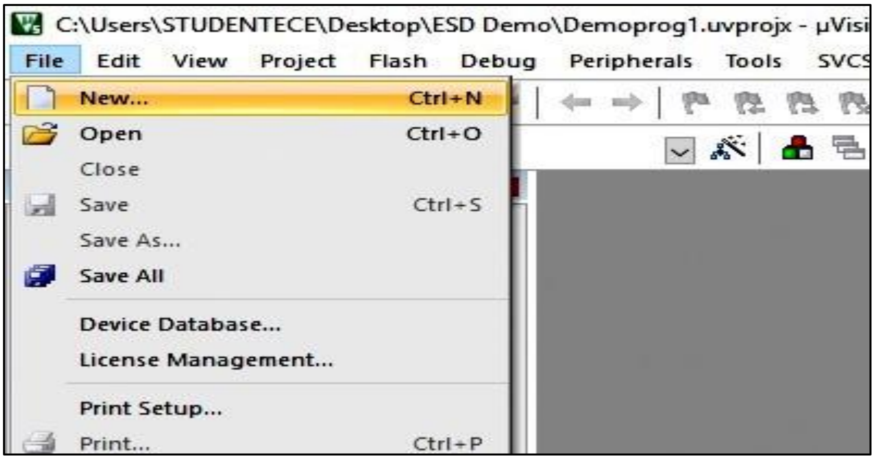
Step 6.

Select

File-> New

To open new editor window

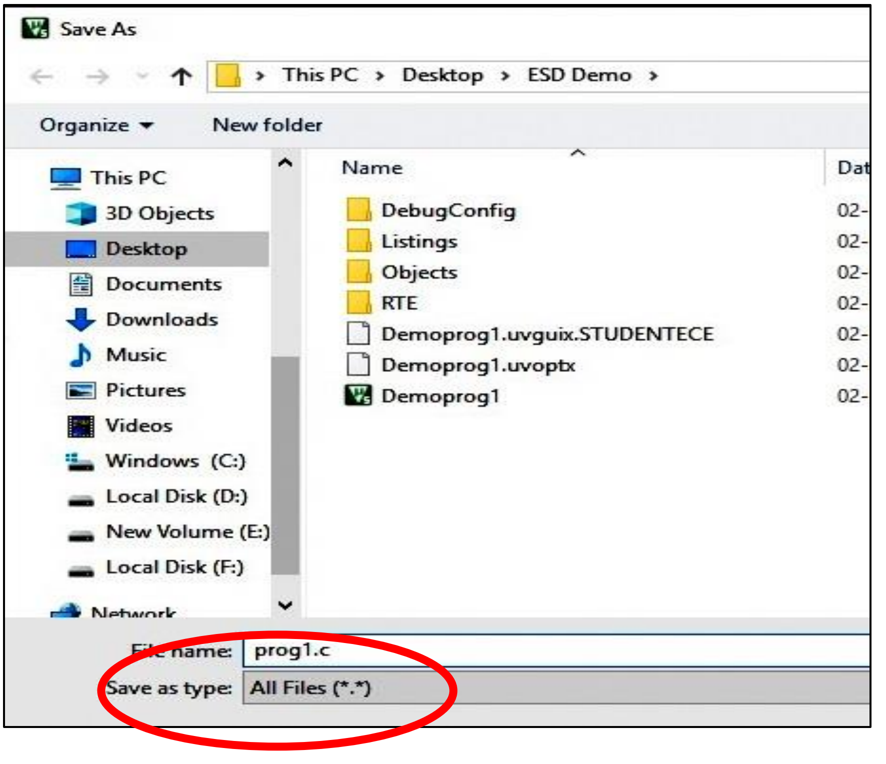
Type your **CODE** in this window and Save



Step 7.

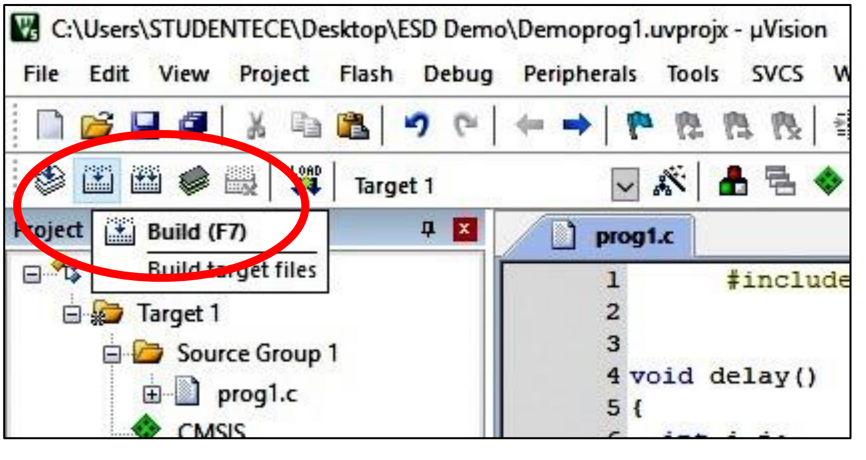
In case of **Assembly Program** Save with extension **.s** (i.e. file_name.s) & in case of **C-Program** Save with extension **.c** (i.e. file_name.c)

Note: See that you save your program in same directory in which you had created your **Project**.



Step 8.

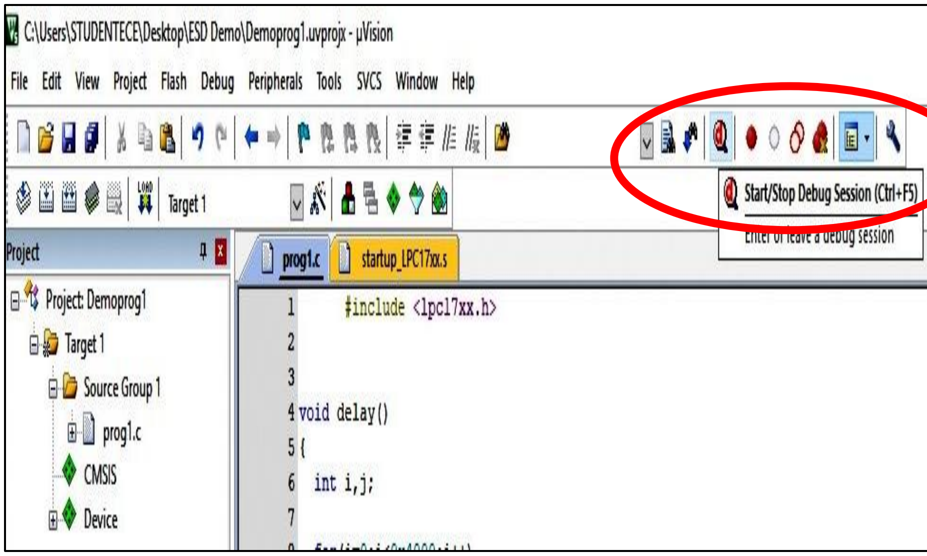
Save the program and **Build** the target as shown in figure.



Errors or warnings are displayed in the **Build Output Window**. Double-click on a message to jump to the line where the error/warning occurred.

Step 10.

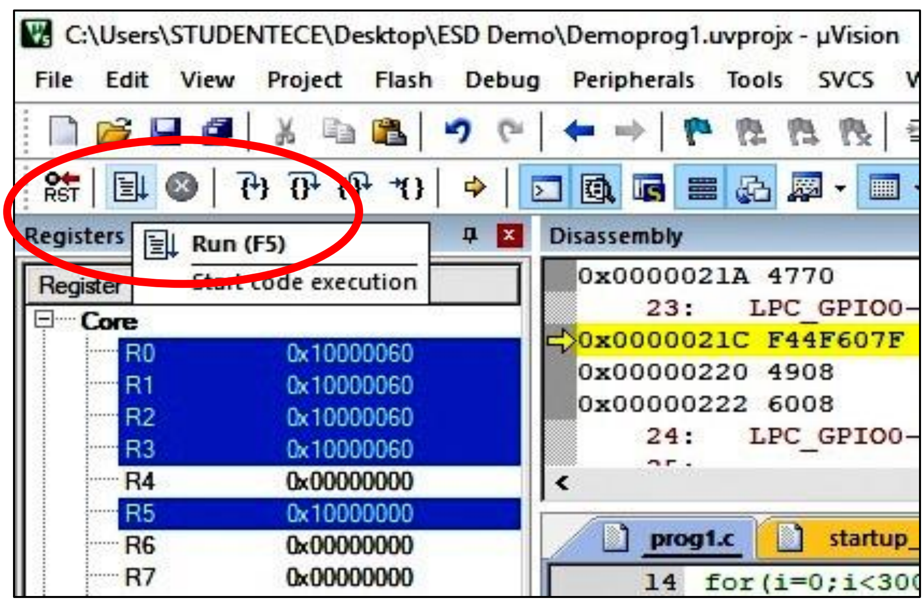
For **Assembly program execution** use **Debug**(Start/ Stop Debug) option as shown in figure.



Step 11.

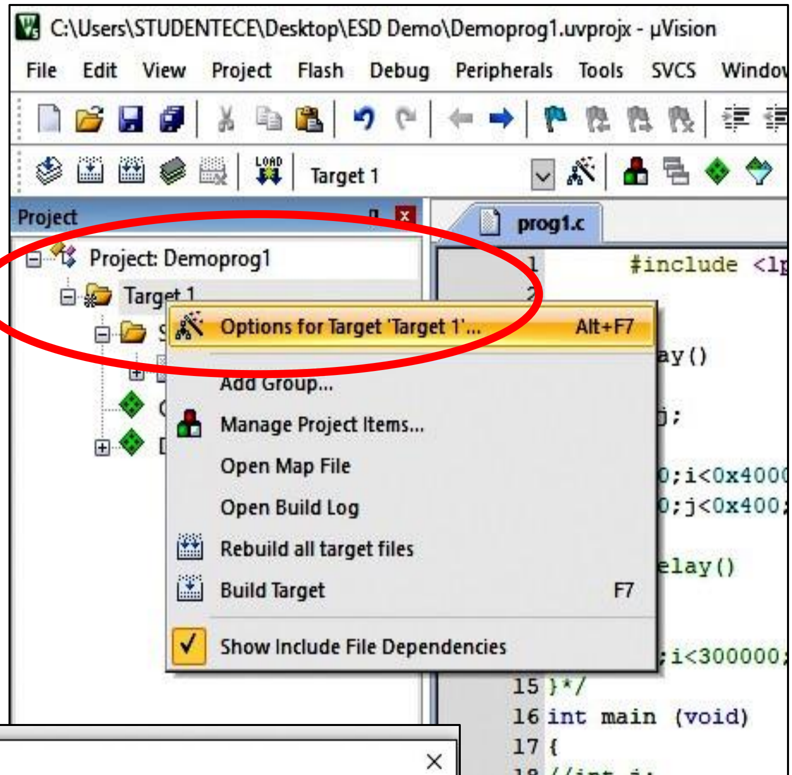
Then **Run** your program and observe the output at **Registers / Memory** window.

For **single step execution** press **f11** and observe the output



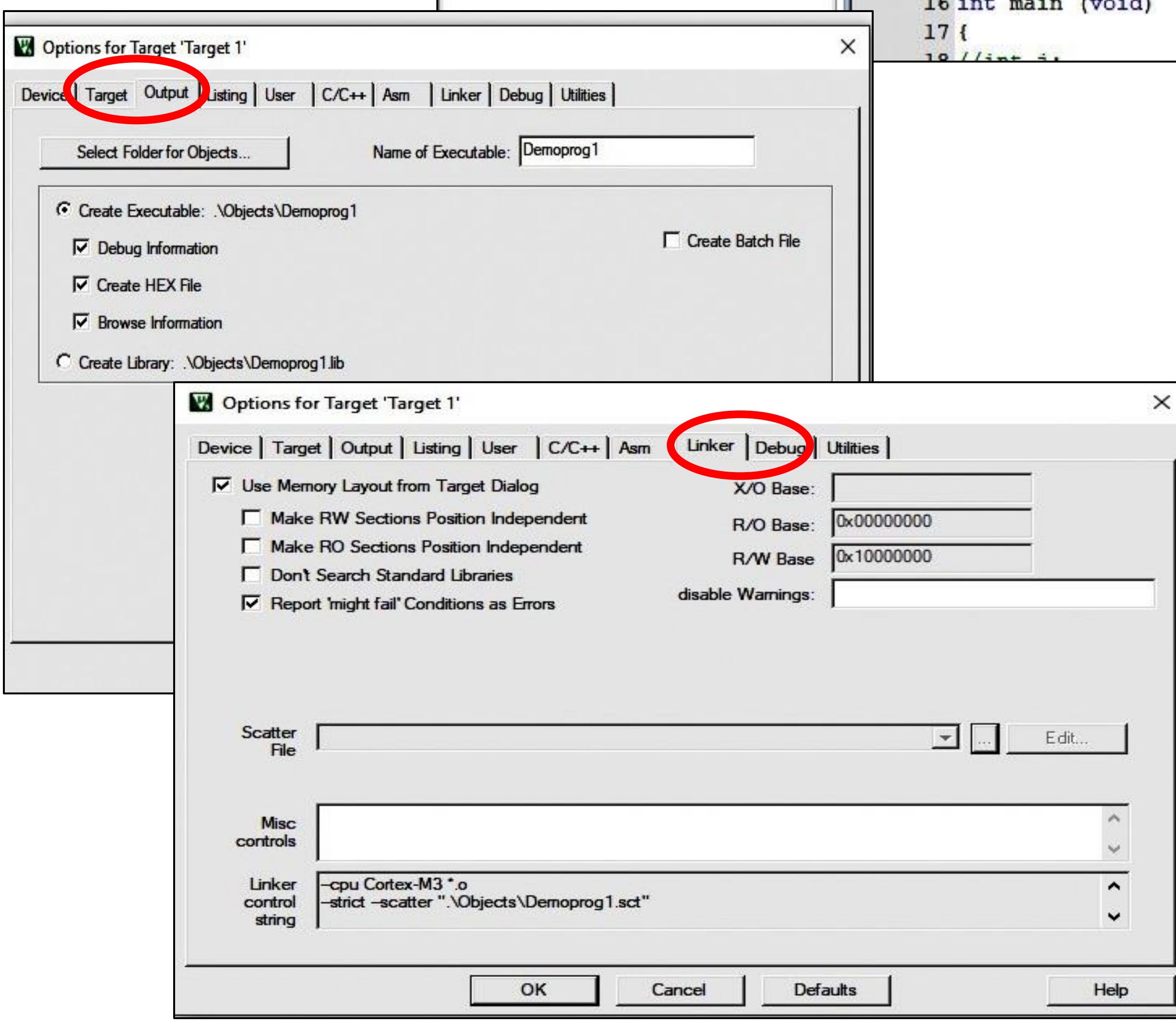
In case of C Program

To create **.hex file Click on Target 1 -> Options for Target**



Select **Output** and choose **Create HEX File**

Then Select **Linker** and choose **Use Memory Layout from Target Dialog**



III. Downloading Hex File to LPC 1768 (ONLY FOR C-Programs)

The **Flash magic** can be used to load the created **Hex file** of the program into **LPC1768 MCU,** after connecting the **MCU with PC** follow the below steps.

➢**Click** on the **Flash Magic** Icon installed over desktop

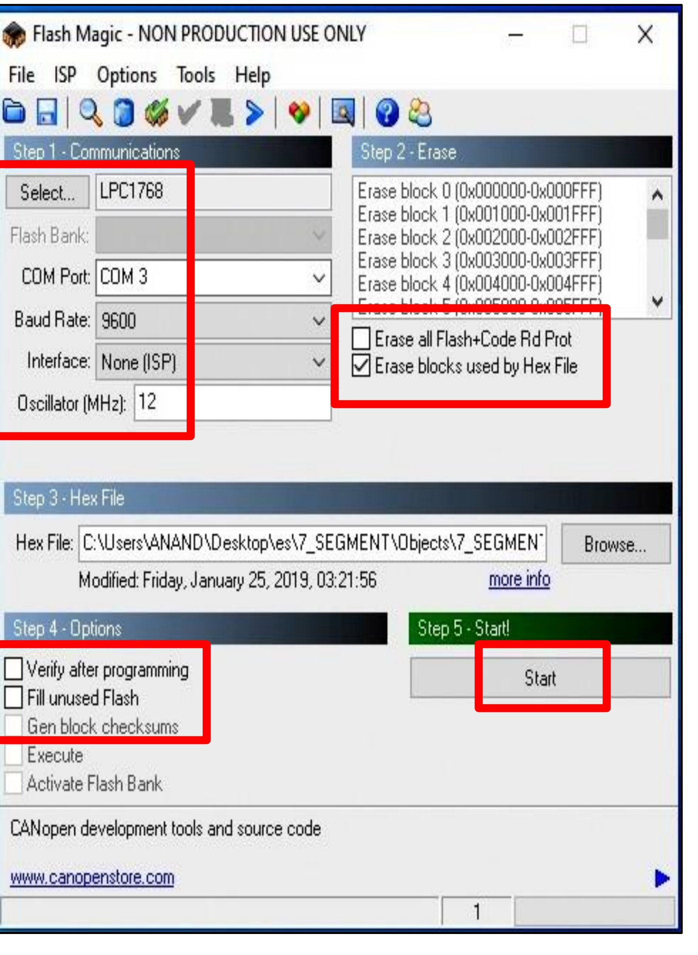
➢Click select device and choose **LPC1768**, select **COM port**, **Baud rate**, **Interface**, and **oscillator** fields, as shown below.

➢The **COM Port** selection is based on the **USB port of PC** to which the board is connected.

➢Select **Erase blocks used by Hex file**.

➢Select **Verify after Programming**.

➢Click **Browse** and select **.Hex file** from the project directory, inside **Objects** Folder.



➢Click **'START'**.

➢The Program will be downloaded into trainer kit by displaying **Programming device**….. **Finished** indicates the Successful downloading of the program.

➢The **Appropriate Output** can be seen in the **Peripherals of the LPC1768 Kit** or the **Hyper Terminal Window**.

Right Click on My Computer option from the desktop and select Manage Option as shown in the below figure.

Click on Manage button and select the Device Manager Option as shown below

Expand Ports option on the right hand side to see the Serial port name of the USB Connected. For Example : **USB Serial Port(COM8)**

ASSEMBLY PROGRAMS

PROGRAM 1A

Write an Assembly language program to calculate 10+9+8+.........+1

```
;;; Assembler Directives

    PRESERVE8           ;The PRESERVE8 directive specifies that the current file
                         preserves eight-byte alignment of the stack.
     THUMB

; Vector Table Mapped to Address 0 at Reset
; Linker requires __Vectors to be exported

     AREA    RESET, DATA, READONLY
     EXPORT  __Vectors

__Vectors
     DCD  0x20001000    ; stack pointer value when stack is empty
     DCD  Reset_Handler     ; reset vector

     ALIGN

; The program
; Linker requires Reset_Handler

     AREA    MYCODE, CODE, READONLY

             ENTRY
             EXPORT Reset_Handler

 Reset_Handler

;;;;;;;;;;User Code Starts from the next line;;;;;;;;;;;;
                        MOV         R1, #0x0A       ;R1 -> 10
                        MOV         R2, #0X00       ;R2 -> 0
LOOP1       ADD         R2, R1                      ;R2 -> R2 + R1
                        SUBS        R1, #0X01       ;R1 -> R1 - 1
                        BNE         LOOP1
HERE        B HERE                                  ;Stop here
                        END                         ;End of the program
```

Expected Output:

Before Execution:

Register	Value
Core	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000008
xPSR	0x01000000

After Execution:

Register	Value
Core	
R0	0x00000000
R1	0x00000000
R2	0x00000037
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000016
xPSR	0x61000000

Results & Observations:

Challenge for you:

Try writing an Assembly language program to calculate 1+2+3+.........+10

PROGRAM 1B

Write a Assembly language program to link Multiple object files and link them together.

;test1.s contains the main program as experiment1
;test2.s contains the addition subroutine
;test3.s contains the subtraction subroutine

;test1.s

```
;;; Directives
        PRESERVE8
        THUMB

; Vector Table Mapped to Address 0 at Reset
; Linker requires __Vectors to be exported

        AREA    RESET, DATA, READONLY
        EXPORT  __Vectors

__Vectors
        DCD  0x20001000                    ; stack pointer value when stack is empty
        DCD  Reset_Handler                 ; reset vector

        ALIGN

         AREA    MYCODE, CODE, READONLY

                    IMPORT add_handler          ;import the add and sub handlers
                    IMPORT sub_handler
                    ENTRY
                    EXPORT Reset_Handler

 Reset_Handler
;;;;;;;;;;User Code Starts from the next line;;;;;;;;;;;

                            MOV          R1, #10        ;R1 -> 10
                            MOV          R2, #0         ;R2 -> 0
LOOP1         BL            add_handler
                            BL  sub_handler

              BNE           LOOP1
HERE          B HERE

                            END                         ;End of the program
```

;test2.s

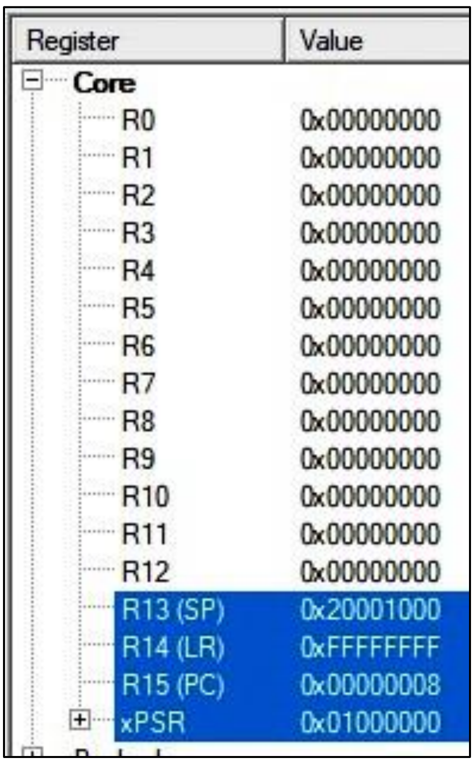
```
                AREA    subroutine1, CODE, READONLY
                EXPORT add_handler

add_handler
__add          ADD R2, R1
               BX lr   ;Branch  indirect - Link Register (which contains the return address)
               END
```

;test3.s

```
                AREA    subroutine2, CODE, READONLY
                EXPORT sub_handler
sub_handler
__sub           SUBS R1, #1
                BX lr     ;Branch indirect-Link Register (which contains the return address)
                END
```

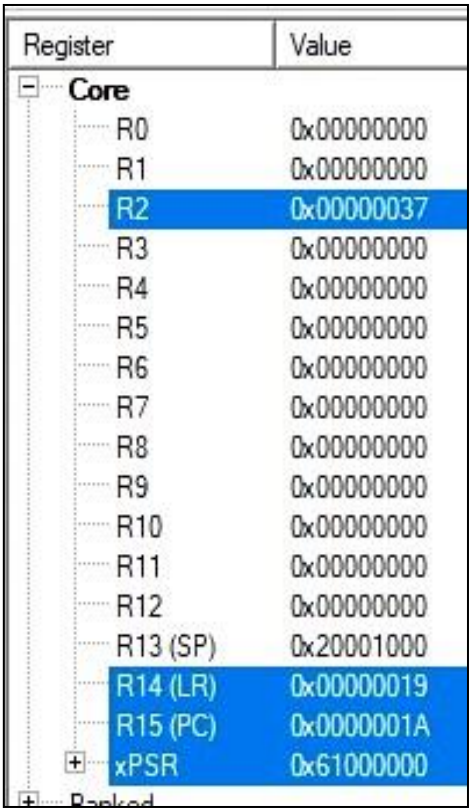
Expected Output:

Before Execution:



Register	Value
Core	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000008
xPSR	0x01000000

After Execution:



NOTE:

Observe that the **Link Register (LR)** contents changes twice during **RUN** time, once when branching to add_handler as **0x00000015** and again when branching to sub_handler as **0x00000019**

Results & Observations:

Challenge for you:

Try writing an Assembly language program to calculate $1^2 + 2^2 + 3^2 + \dots\dots + 10^2$

PROGRAM 2

Write an Assembly language program to solve the given Polynomial Function

Let the given Function be: **5x^2-6x+8** when **x=7**

```
;;; Directives
        PRESERVE8
        THUMB

; Vector Table Mapped to Address 0 at Reset
; Linker requires __Vectors to be exported

        AREA    RESET, DATA, READONLY
        EXPORT  __Vectors

__Vectors
        DCD  0x20001000                ; stack pointer value when stack is empty
        DCD  Reset_Handler             ; reset vector

        ALIGN

; The program
; Linker requires Reset_Handler

        AREA    MYCODE, CODE, READONLY

                ENTRY
                EXPORT Reset_Handler


Reset_Handler
;;;;;;;;;;User Code Starts from the next line;;;;;;;;;;;;

            MOV   R0, #7                       R0-> x=7
            MUL   R1, R0,R0                    R1->x^2
            MOV   R4, #5
            MUL   R1, R1,R4
            MOV   R5, #6
            MUL   R2, R0,R5
            SUB   R3, R1,R2
            ADD   R3,R3,#8
HERE        B  HERE                            ;Stop here
            END                                ;End of the program
```

Expected Output:

Before Execution:

Register	Value
Core	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000008
xPSR	0x01000000

After Execution:

Register	Value
Core	
R0	0x00000007
R1	0x000000F5
R2	0x0000002A
R3	0x000000D3
R4	0x00000005
R5	0x00000006
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000028
xPSR	0x01000000

Results & Observations:

Challenge for you:

Try writing an Assembly language program to solve Functions with different values of x, different operations and x^3/^4 powers.

PROGRAM 3

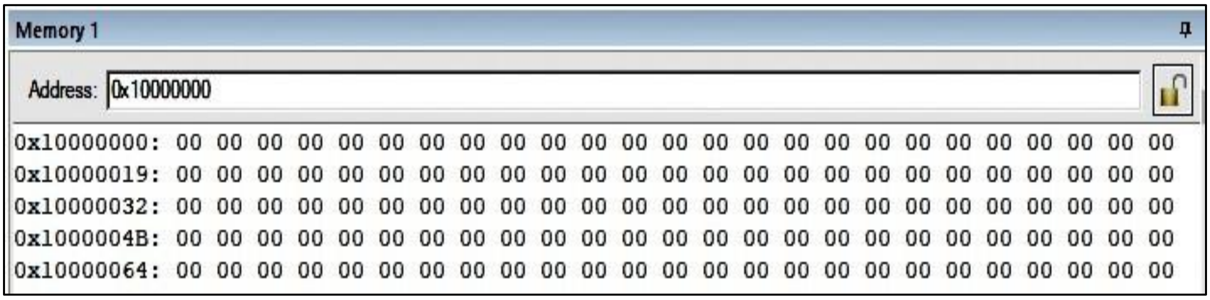
Write an Assembly language program to store data in RAM.

```
;;; Directives
        PRESERVE8
        THUMB

; Vector Table Mapped to Address 0 at Reset
; Linker requires __Vectors to be exported

        AREA    RESET, DATA, READONLY
        EXPORT  __Vectors

__Vectors
        DCD  0x20001000     ; stack pointer value when stack is empty
        DCD  Reset_Handler  ; reset vector

        ALIGN

; The program
; Linker requires Reset_Handler

        AREA    MYCODE, CODE, READONLY

                ENTRY
                EXPORT Reset_Handler


Reset_Handler
;;;;;;;;;;User Code Starts from the next line;;;;;;;;;;;;

                MOV R1, #0x10000000             ;address -> 0x10000000
                MOV R0, #0x501                  ;value -> 0x00000501
                STR R0, [R1]

                ADD R1, R1, #4                  ;address -> 0x10000004
                MOV R0, R0, LSL #1              ;value -> 0x00000A02
                STR R0, [R1]

                ADD R1, R1, #4                  ;address -> 0x10000008
                MOV R0, R0, LSL #1              ;value -> 0x00001404
                STR R0, [R1]
```

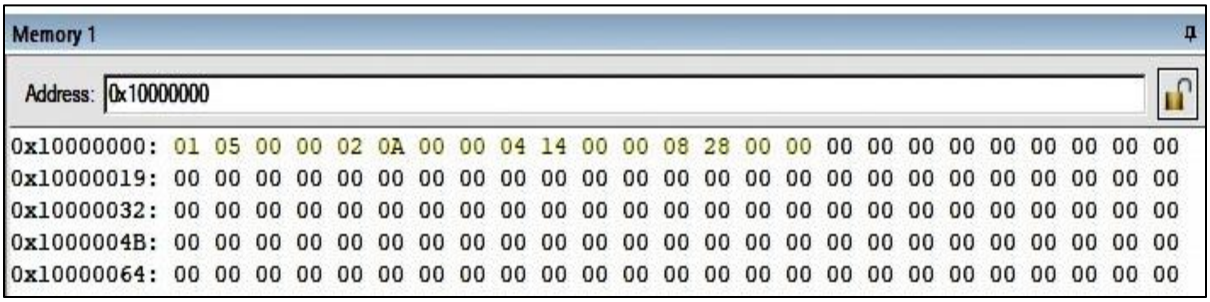
```
        ADD R1, R1, #4              ;address -> 0x1000000C
        MOV R0, R0, LSL #1          ;value -> 0x00002808
        STR R0, [R1]
here  B here

        END
        ;End of the program
```

Expected Output:

Before Execution:



After Execution:



Results & Observations:

Challenge for you:

Try writing an Assembly Program for Loading back the data in memory into Registers (R2, R3, R4 & R5)

INTERFACING/C PROGRAMS

PROGRAM 4

Write a C-Program to display "Hello world" by using UART Serial Communication Protocol

Steps for Configuring UART0:

Step1: Configure the GPIO pin for UART0 function using PINSEL register.
Step2: Configure the FCR for enabling the FIXO and Reset both the Rx/Tx FIFO.
Step3: Configure LCR for 8-data bits, 1 Stop bit, Disable Parity and Enable DLAB.
Step4: Get the PCLK from PCLKSELx register 7-6 bits.
Step5: Calculate the DLM,DLL values for required baudrate from PCLK.
Step6: Update the DLM,DLL with the calculated values.
Step7: Finally clear DLAB to disable the access to DLM,DLL. After this the UART will be ready to Transmit/Receive Data at the specified baudrate

Baud rate calculation:

$$UARTn_{baudrate} = \frac{PCLK}{16 \times (256 \times UnDLM + UnDLL)}$$

To obtain Baud rate 9600:
PCLK= 24MHZ
DLM=0
DLL=156 (in decimal) or 9C (in Hexa decimal)

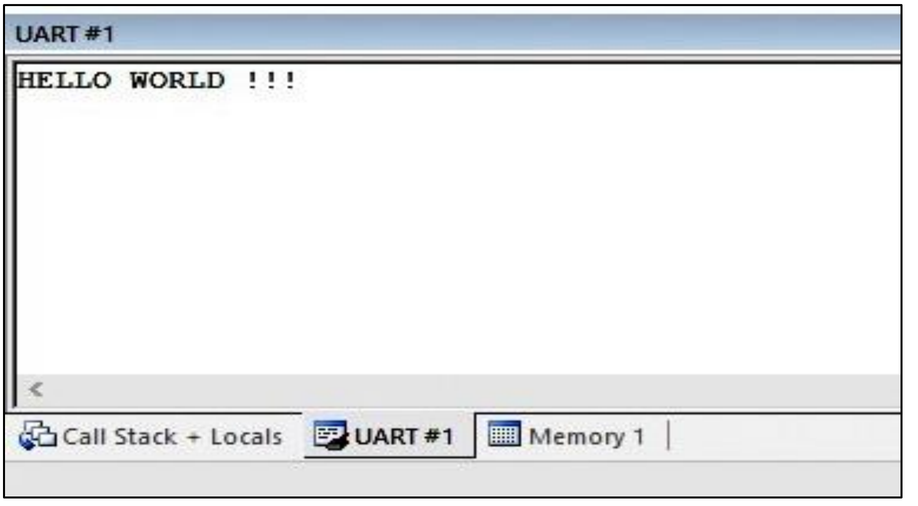
Register	Description
RBR	Contains recently received Data
THR	Contains Data to be Transmitted
FCR	FIFO Contol Register
LCR	Controls UART frame from atting(No. of Data Bits, Stop Bits)
DLL	LSB of the UART Baud Rate Generator value.
DLM	MSB of the UART Baud Rate Generator value.

Program:

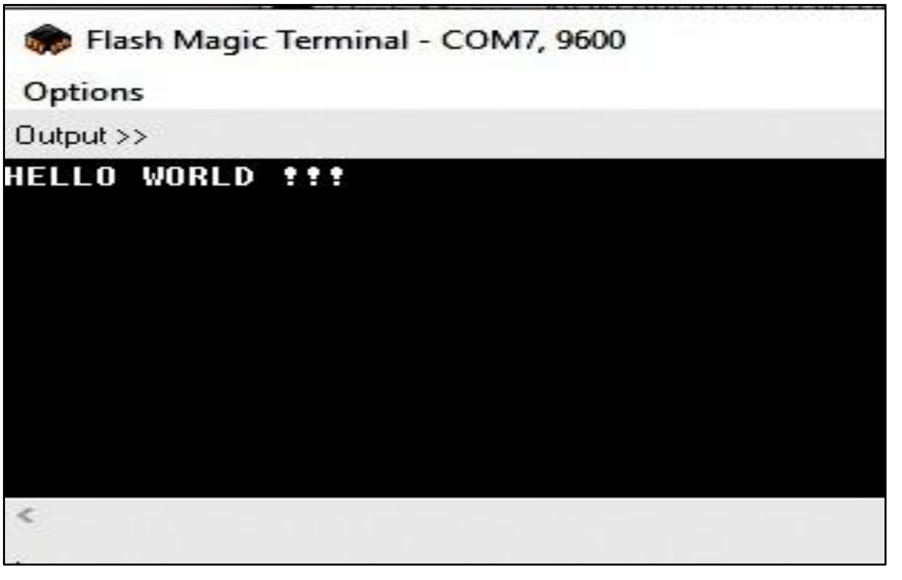
```
#include "LPC17xx.h"

#define FOSC      12000000          /* oscillator frequency */
#define FCCLK   (FOSC  * 8)         /*  master clock frequency <= 100Mhz */
#define FCCO    (FCCLK * 3)         /*  PLL frequency (275Mhz to 550Mhz) */
#define FPCLK   (FCCLK / 4)         /*  peripheral clock frequency */
#define UART0_BPS     9600          /*  Serial communication baud rate 9600 */

void UART0_Init (void)
{
 unsigned int usFdiv;
 LPC_PINCON->PINSEL0 |= (1 << 4);         /* Pin P0.2 used as TXD0 (Com0) */
 LPC_PINCON->PINSEL0 |= (1 << 6);         /* Pin P0.3 used as RXD0 (Com0) */
 LPC_UART0->LCR  = 0x83;                  /* allows you to set the baud rate  */
 usFdiv = (FPCLK / 16) / UART0_BPS;       /* set the baud rate*/
 LPC_UART0->DLM  = usFdiv / 256;
 LPC_UART0->DLL  = usFdiv % 256;
 LPC_UART0->LCR  = 0x03;               /* Lock the baud rate*/
 LPC_UART0->FCR  = 0x06;
}

int UART0_SendByte (int ucData)
{
   while (!(LPC_UART0->LSR & 0x20));
  return (LPC_UART0->THR = ucData);
}

void UART0_SendString (unsigned char *s)
{
                while (*s != 0)
                UART0_SendByte(*s++);
}
int main(void)
{
                UART0_Init();
                UART0_SendString("HELLO WORLD !!!");
                UART0_SendByte(0x0D); UART0_SendByte(0x0A); //CR LF
}
```

Expected Output:

@ Serial Window UART#1:



@ Hyper Terminal Window:



Results & Observations:

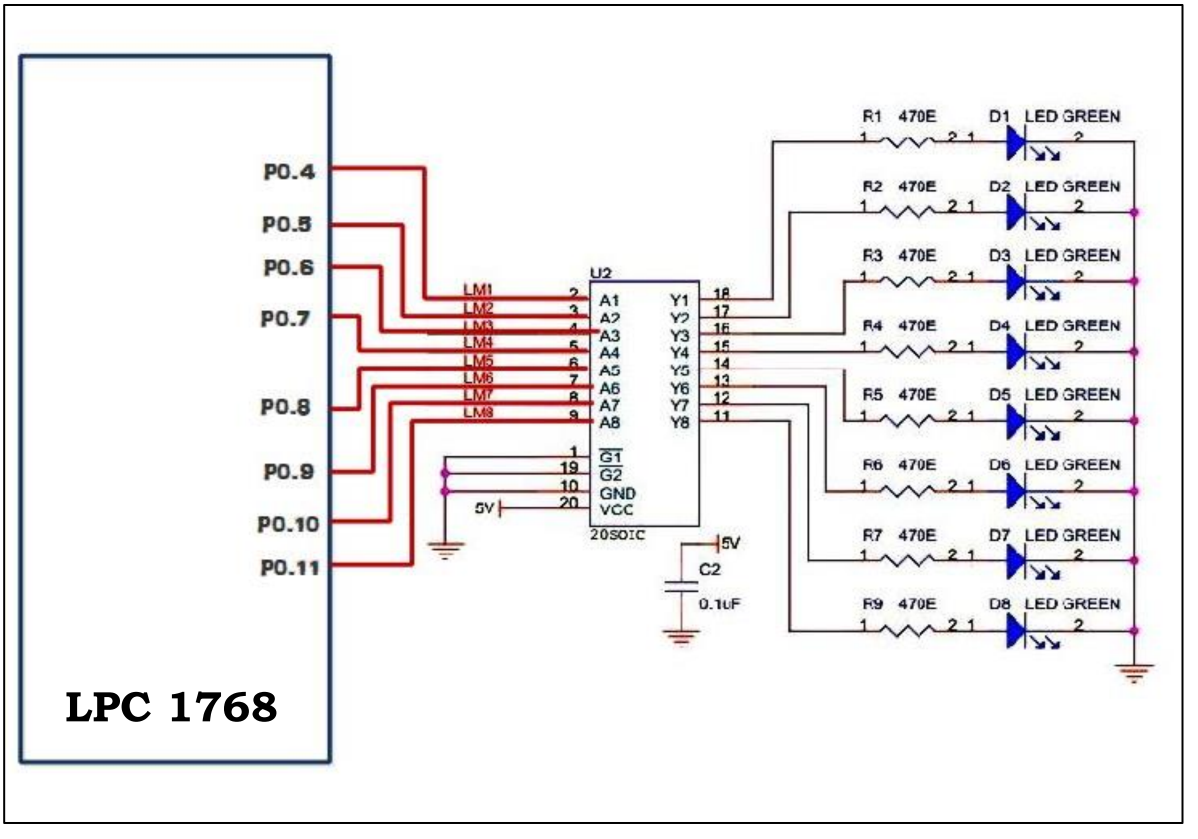
Challenge for you:

Try Displaying more than 1 Line Message & more characters.

PROGRAM 5

Write a C-Program to interface 8 LEDs with LPC1768 to BLINK in given Pattern

Interfacing Diagram:



Interfacing Details:

This board provides eight individual **COMMON CATHODE** SMD LED's connected to LPC-1768 device through 74HC151driver IC. D1 to D8 are connected to general purpose I/O pins on LPC-1768 device as shown in table. When LPC-1768 device drives Logic "1" the corresponding LED turns on.

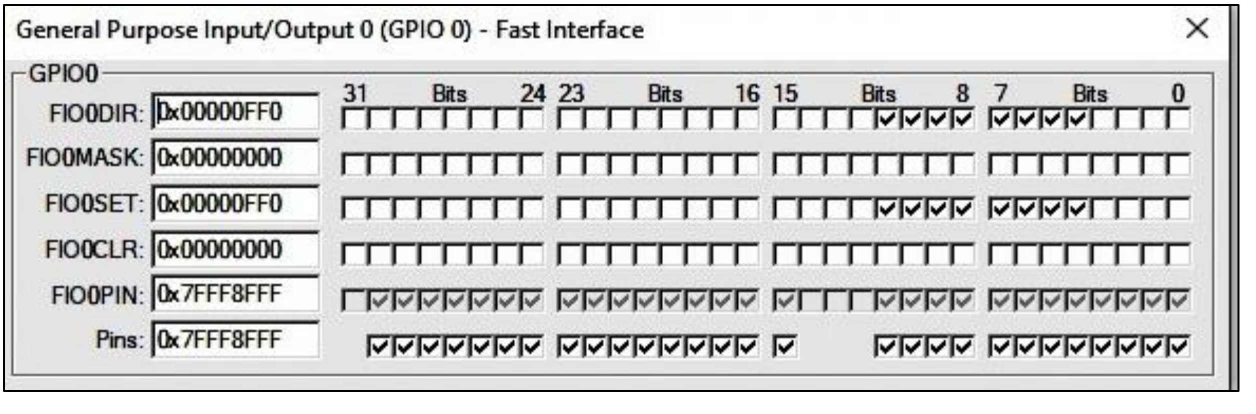
LED	D1	D2	D3	D4	D5	D6	D7	D8
LPC-1768 Pin No	81	80	79	78	77	76	48	49
LPC-1768 Port No	P0.4	P0.5	P0.6	P0.7	P0.8	P0.9	P0.10	P0.11

Program:

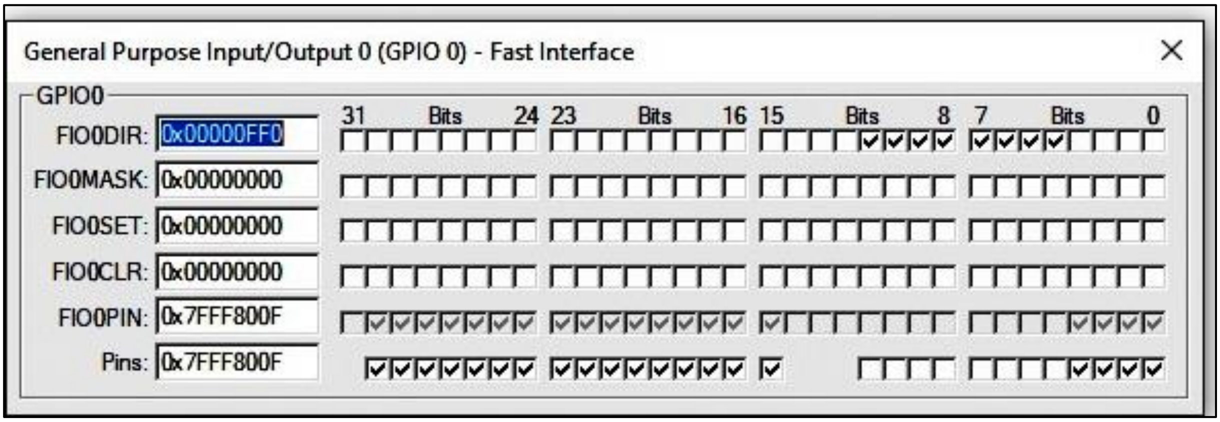
```
#include <lpc17xx.h>
void delay()
{
            int i,j;

            for(i=0;i<0x4000;i++)
            for(j=0;j<0x400;j++);
}

int main (void)
{
 LPC_GPIO0->FIODIR = 0x0000FF0;         /* P2.xx defined as Outputs */
 LPC_GPIO0->FIOCLR = 0x0000FF0;         /* turn off all the LEDs */

  while(1)
 {

             LPC_GPIO0->FIOSET = 0x0000FF0;
             delay();
             LPC_GPIO0->FIOCLR = 0x0000FF0;
             delay();
            }

}
```

<u>Expected Output:</u>

All 8 LEDs ON State:



All 8 LEDs OFF State:



Results & Observations:

Challenge for you:

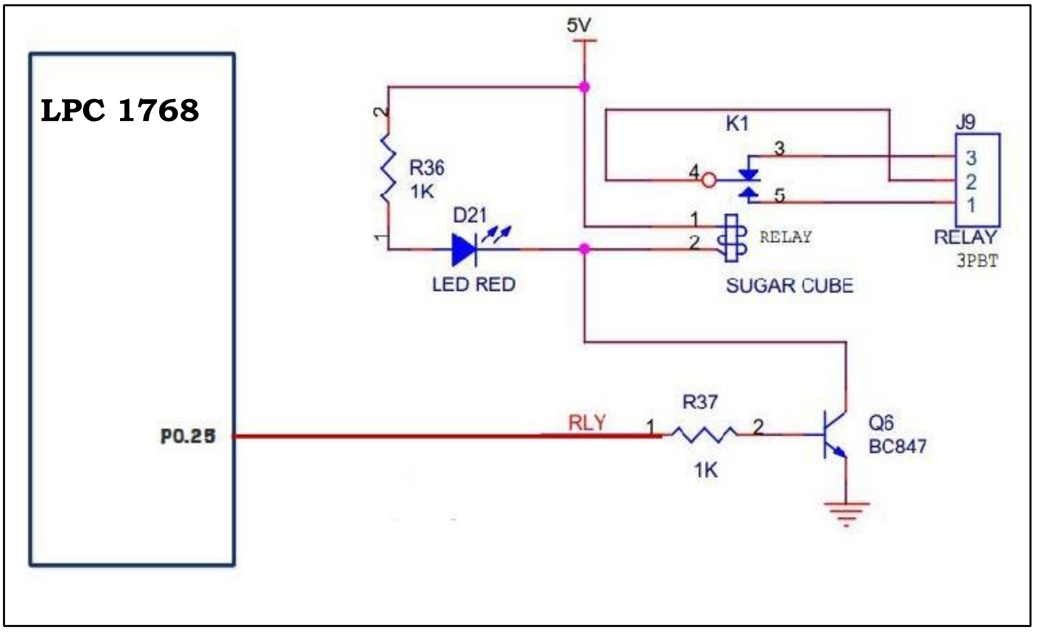
Try writing a program to blink LEDs with different Patterns (like alternate LEDs glowing, 1 after the other, binary series etc.) and different delays.

PROGRAM 6

Write an C-Program to interface Relay with LPC1768

A **relay** is an electrically operated switch. Many relays use an electromagnet to operate a switching mechanism mechanically, but other operating principles are also used. Relays are used where it is necessary to control a circuit by a low-power signal (with complete electrical isolation between control and controlled circuits), or where several circuits must be controlled by one signal. The first relays were used in long distance telegraph circuits, repeating the signal coming in from one circuit and re-transmitting it to another. Relays were used extensively in telephone exchanges and early computers to perform logical operations.

Interfacing Diagram:



Interfacing Details:

K1 is an electromagnetic relay which is connected to P0.25 through Dip switch. We need to send logic "1" to switch on relay.J9 is three pin PBT terminal used to connect external device to relay. Table shows connections for Relay.

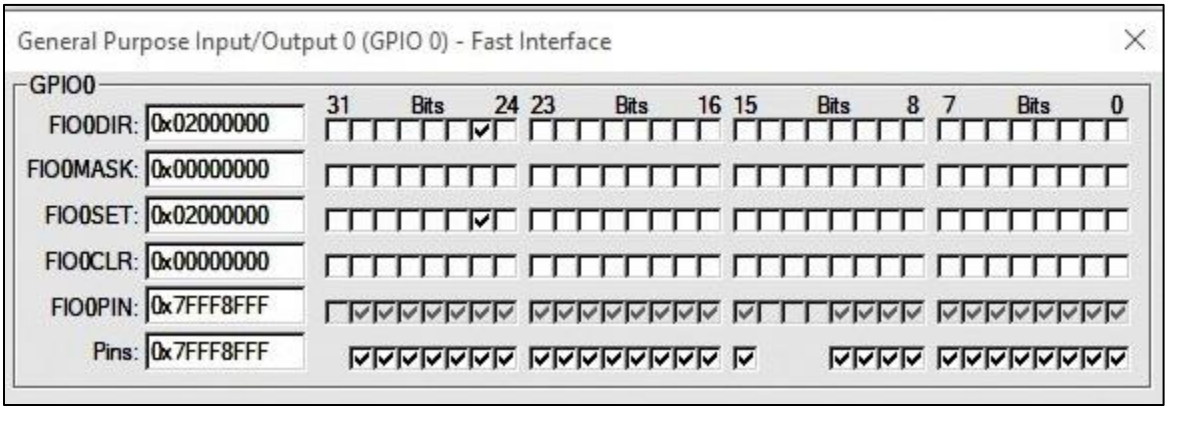
Relay coil	LPC-1768 Pin No	LPC-1768 Port No
Relay	07	P0.25

PROGRAM:

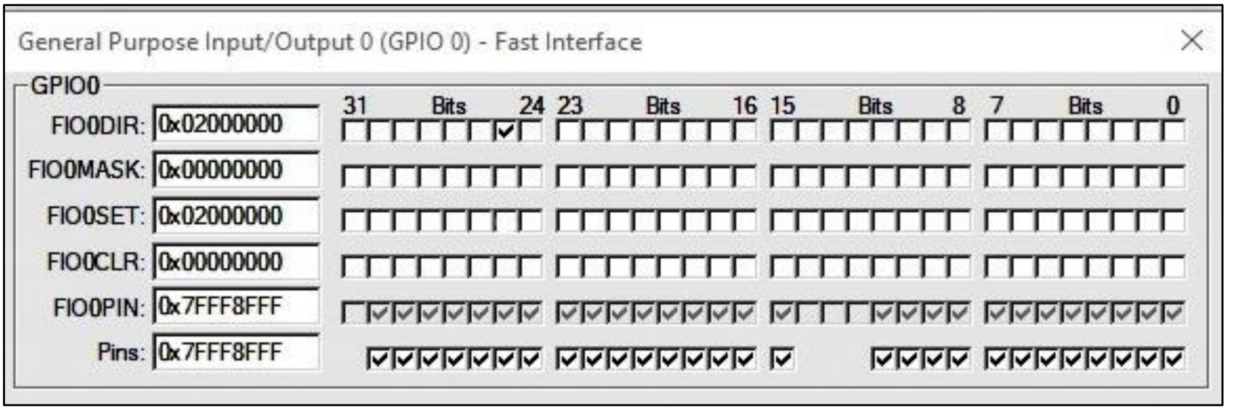
```
#include "lpc17xx.h“

void delay()
{
                int i,j;

                for(i=0;i<0x2000;i++)
                for(j=0;j<0x200;j++);
}

int main (void)
{
  LPC_PINCON->PINSEL7 = 0X02000000;
  LPC_GPIO0->FIODIR = 0X02000000;
  LPC_GPIO0->FIOCLR =0X02000000;

  while(1)
  {
                 LPC_GPIO0->FIOSET =0X02000000;
                 delay();
                 LPC_GPIO0->FIOCLR =  0X02000000;
                 delay();
                }
}
```

Expected Output:

Relay ON/CLOSED State:



Relay OFF/OPEN State:



Results & Observations:

Challenge for you:

Try writing a program to interface BUZZER with the LPC1768 to generate audio output.

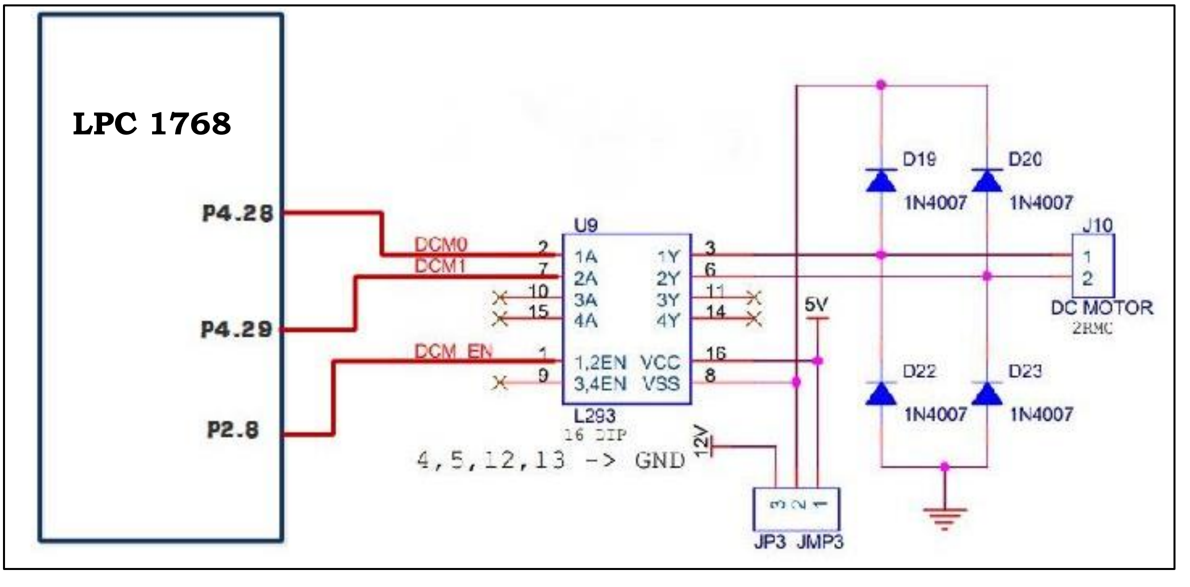
Relay coil	LPC-1768 Pin No	LPC-1768 Port No
Buzzer	27	P3.25

PROGRAM 7A

<u>Write a C Program to control direction of DC motor rotation using LPC 1768</u>.

DC Motor Pin Configuration DC (direct current) motor rotates continuously. It has two terminals positive and negative. Connecting DC power supply to these terminals rotates motor in one direction and reversing the polarity of the power supply reverses the direction of rotation. The speed of Dc motor can maintained at a constant speed for a given load by using “Pulse Width Modulation (PWM)” technique. By changing the width of the pulse of applied to dc motor, the power applied is varied thereby DC motor speed can be increased or decreased. Wider the pulse Faster is the Speed, Narrower is the Pulse, and Slower is the Speed U9 is L293 driver IC used to drive the dc motor. It has enable lines which is used to switch on the DC motor. It is connected to P4.28. Logic “1‟ enables the driver and logic “0‟ disables the driver.P4.28 and P4.29 are used for Motor 1 direction and speed control.

Interfacing Diagram:



Interfacing Details:

Motor Selection	Motor Direction	LPC-1768 Pin No	LPC-1768 Port No
Motor	DCM0-Clk	82	P4.28=1 & P4.29=0
	DCM1-Aclk	85	P4.28=0 & P4.29=1
	DCM_EN	65	P2.8=1

PROGRAM:

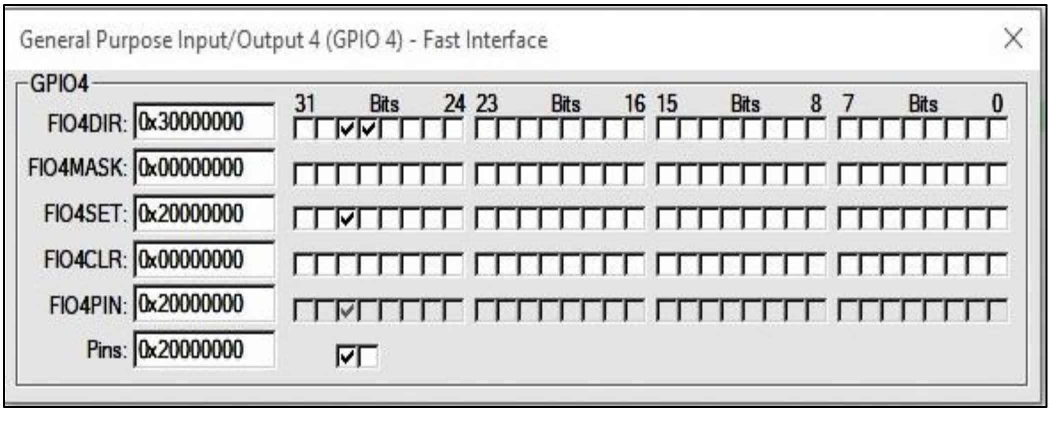
```
#include <LPC17xx.H>
int main (void)
{
LPC_GPIO4->FIODIR = 0x30000000 ;
LPC_GPIO1->FIODIR&=~(0x0001C000);
LPC_GPIO2->FIOSET =(1<<8);
while(1)
{
if (!(LPC_GPIO1->FIOPIN & (1<<14)))          //START key pressed
LPC_GPIO4->FIOPIN=0X10000000;               // Motor CCW

if (!(LPC_GPIO1->FIOPIN &(1<<15)))           //STOP key pressed
LPC_GPIO4->FIOPIN=0X30000000;               // switch OFF Motor

if (!( LPC_GPIO1->FIOPIN & (1<<16)))         //CW key pressed
LPC_GPIO4->FIOPIN=0X20000000;
}
}
```

Expected Output:

If the SWITCH 1 is Pressed MOTOR rotates in CLOCKWISE Direction.

CLOCKWISE Rotation:

@ P4.28

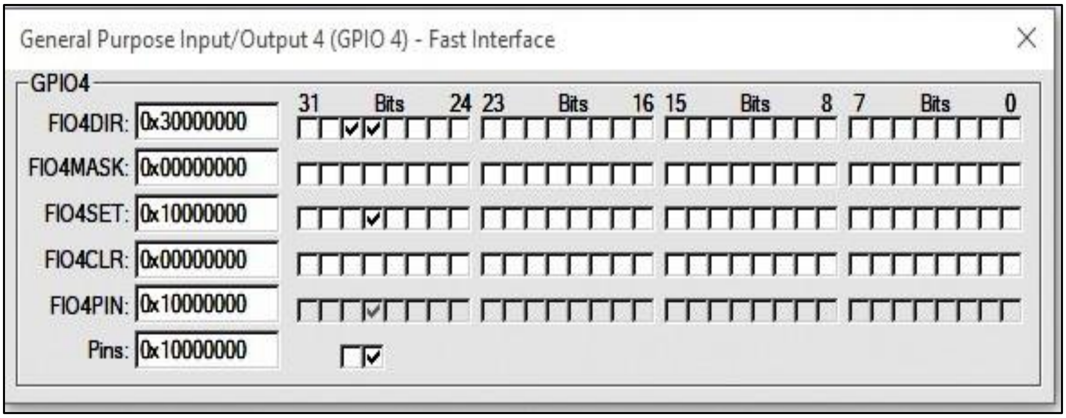


➢**If the SWITCH 2 is Pressed MOTOR STOPs.**

➢**If the SWITCH 3 is Pressed MOTOR rotates in COUNTER-CLOCKWISE Direction.**

COUNTER- CLOCKWISE Rotation:

@ P4.28



<u>Results & Observations:</u>

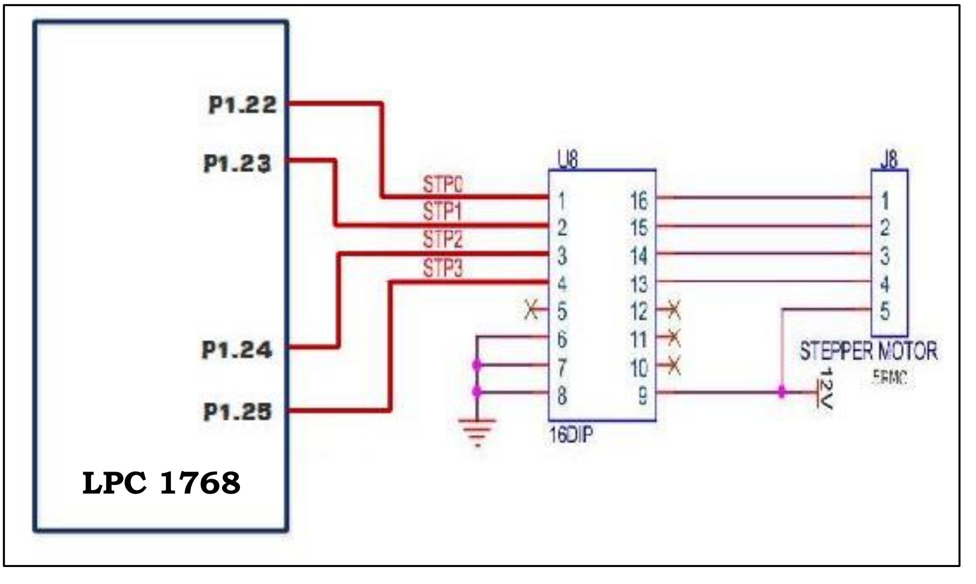
PROGRAM 7B

Write a C Program to control direction of Stepper motor rotation using LPC 1768.

Stepper Motor Configuration:

A **stepper motor** is a special type of electric motor that moves in increments, or steps, rather than turning smoothly as a conventional motor does. Typical increments are 0.9 or 1.8 degrees, with 400 or 200 increments thus representing a full circle. The speed of the motor is determined by the time delay between each incremental movement.
U8 is a Driver Buffer (ULN2003) device connected to LPC-1768 Device and can be used for driving Stepper Motor. On the LPC-1768, P1.22 to P1.25 is used to generate the pulse sequence required to run the stepper Motor. Also, the Stepper Motor is powered by its own power supply pin (COM), which is connected to a 12V supply. Table shows connections for stepper Motor.

Interfacing Diagram:



Interfacing Details:

Stepper Motor Coil	LPC-1768 Pin No	LPC-1768 Port No
A	39	P1.22
B	37	P1.23
C	38	P1.24
D	39	P1.25

PROGRAM:

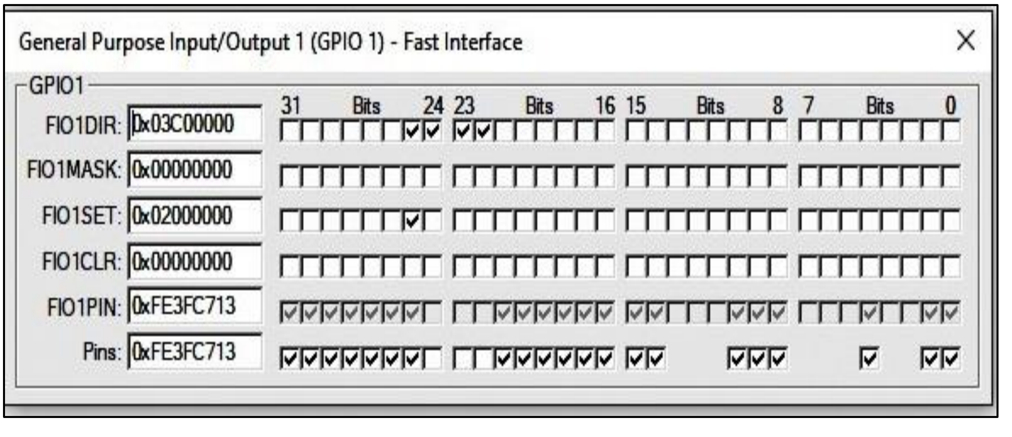
```
#include <LPC17xx.H>

void delay(unsigned int count)
{
unsigned int j=0,i=0;
for(j=0;j<count;j++)
for(i=0;i<5000;i++);
}

int main (void)
{
unsigned int del=50;
LPC_GPIO1->FIODIR = 0x03C00000;
uint32_t i;
while(1)
    {
            if (!(LPC_GPIO1->FIOPIN & (1<<14)))
            {
            for ( i=0; i<500; i++)
            {
            LPC_GPIO1->FIOPIN =0x02000000;
            delay(del);
            LPC_GPIO1->FIOPIN =0x01000000;
            delay(del);
            LPC_GPIO1->FIOPIN =0x00800000;
            delay(del);
```

```
            LPC_GPIO1->FIOPIN =0x00400000;
            delay(del);
            if (!(LPC_GPIO1->FIOPIN & (1<<15)))
            break;
            }}

            if (!(LPC_GPIO1->FIOPIN & (1<<15)))
            {
            for ( i=0; i<500; i++)
            {
            LPC_GPIO1->FIOPIN =0x00400000;
            delay(del);
            LPC_GPIO1->FIOPIN =0x00800000;
            delay(del);
            LPC_GPIO1->FIOPIN =0x01000000;
            delay(del);
            LPC_GPIO1->FIOPIN =0x02000000;
            delay(del);
            if (!(LPC_GPIO1->FIOPIN & (1<<14)))
            break;
            }}
        }
}
```

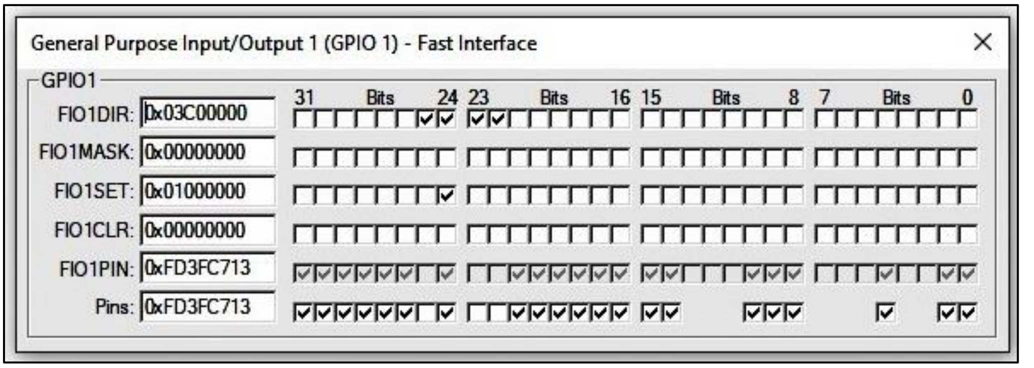
Expected Output:

If SWITCH 1 is Pressed MOTOR rotates in CLOCKWISE Direction:

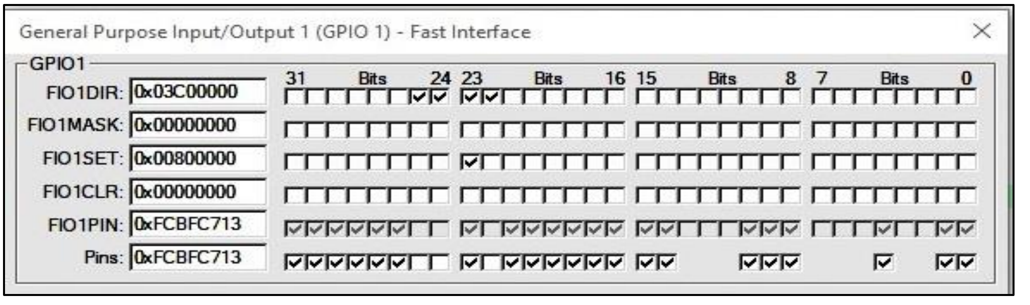
@ P1.25



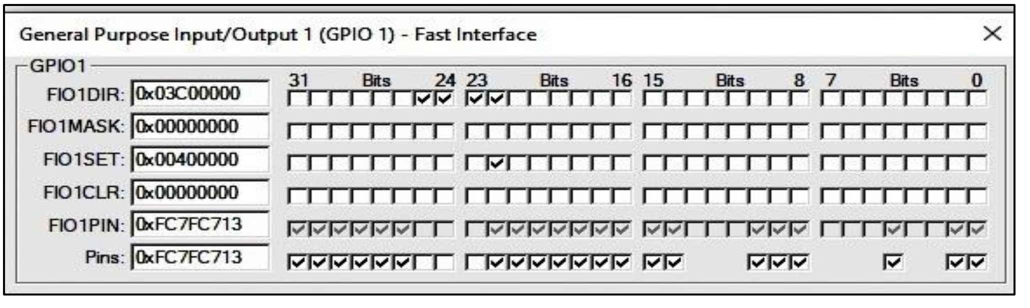
@ P1.24



@ P1.23



@ P1.22



If SWITCH 2 is Pressed MOTOR rotates in COUNTER CLOCKWISE Direction and output is seen at P1.22, P1.23, P1.24, & P1.25 respectively.

Results & Observations:

Challenge for you:

Try writing a program to interface both DC MOTOR and STEPPER MOTOR and Control their direction of rotation with different SWITCHS.

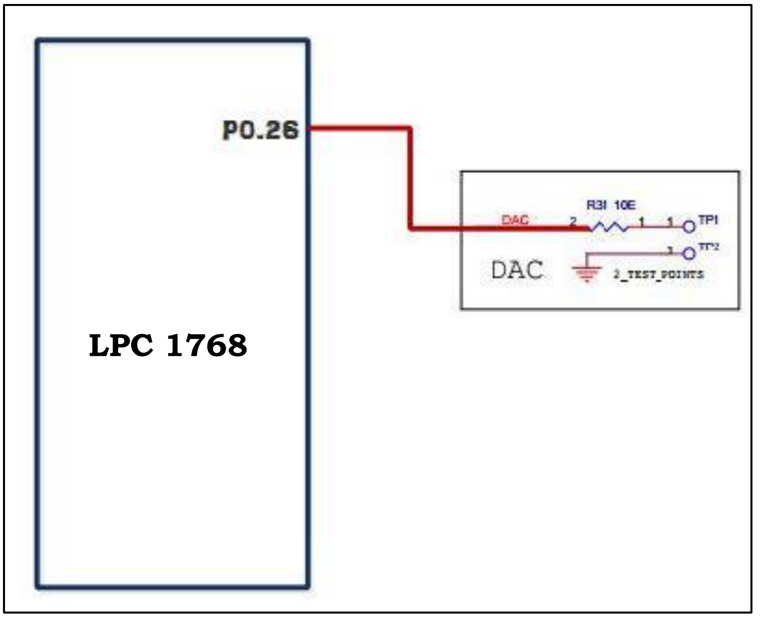
PROGRAM 8

<u>Write a C program to Interface a DAC and generate Triangular and Square waveforms using LPC 1768..</u>

Features of Digital to Analog Converter[DAC]:

- 10-bit digital to analog converter
- Resistor string architecture
- Buffered output
- Selectable speed vs. power
- Maximum update rate of 1 MHZ.

Interfacing Diagram:



Interfacing Details:

As this board comes with one DAC output for generation different wave forms. **AOUT (P0.26)** is connected to TEST point **TP1**. The generated waveforms can be viewed through **TP1** (DAC) and TP2 (GND) by connecting CRO.

a. Program for Square Wave generation:

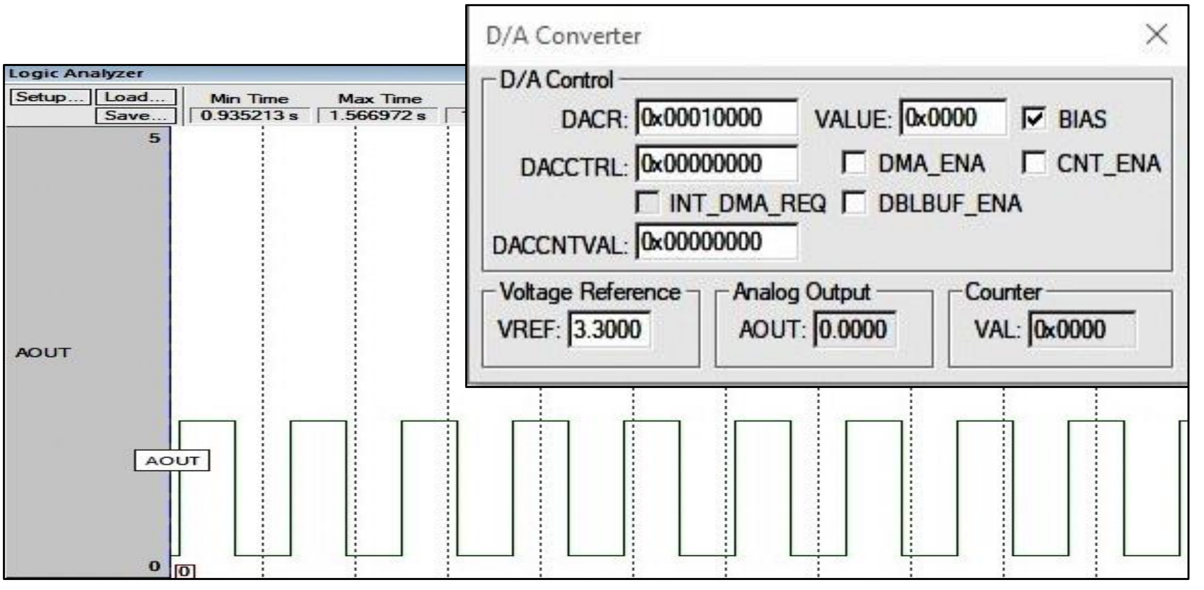
```
#include <LPC17xx.H>
#define voltage 1024
#define freq 120000

int main (void)
{
uint32_t m;
LPC_PINCON->PINSEL1 |= (1<<21);
while(1)
{
LPC_DAC->DACR = (voltage/2 << 6);
for(m = freq; m > 1; m--);
LPC_DAC->DACR = (voltage << 6);
for(m = freq; m > 1; m--);
}
}
```

b. Program for Triangle wave generation:

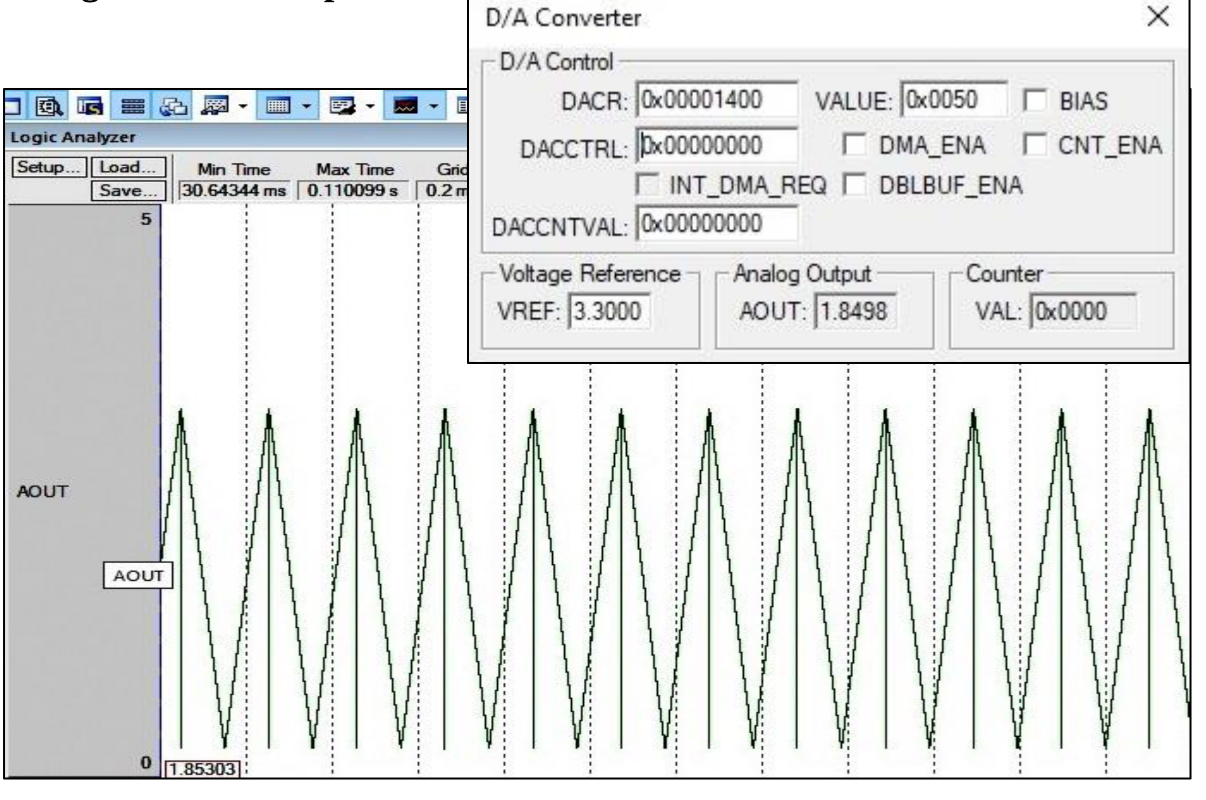
```
#include <LPC17xx.H>
#define voltage 1024
int main (void)
{
uint32_t i = 0;
LPC_PINCON->PINSEL1 |= (1<<21);
while(1)
{
for(i = 0; i < voltage; i++)
LPC_DAC->DACR = (i << 6);
for(i = voltage; i > 0; i--)
LPC_DAC->DACR = (i << 6);
}
}
```

Expected Output:

Square Wave output:



Triangular Wave output:



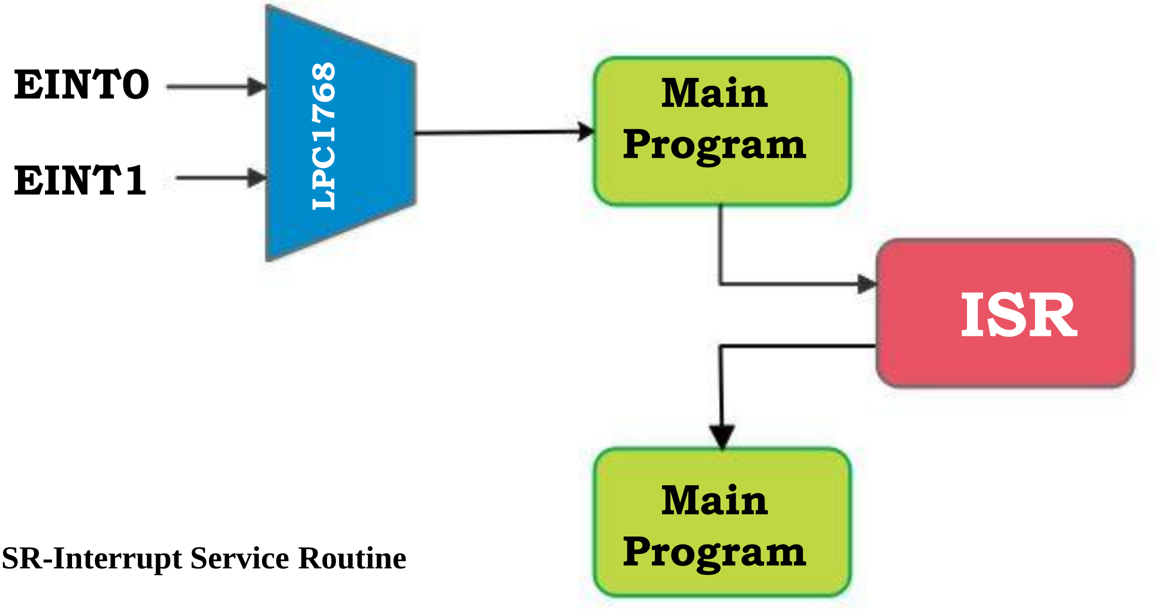
Results & Observations:

Challenge for you:

Try writing a program to generate both Square & Triangular Wave using a single program and selection made through a SWITCH.

PROGRAM 9

Write a C program to demonstrate the use of an external interrupt in LPC 1768



*ISR-Interrupt Service Routine

Register	Description
PINSELx	To configure pins as External interrupts
EXTINT	External Interrupt Flag Register contains flags for EINT0, EINT1, EINT2 & EINT3
EXTMODE	External Interrupt Mode Register (Level/Edge triggered)
EXTPOLAR	External Interrupt Polarity(Falling/Rising, Active High/Low)

Configurations Steps for External Interrupt:

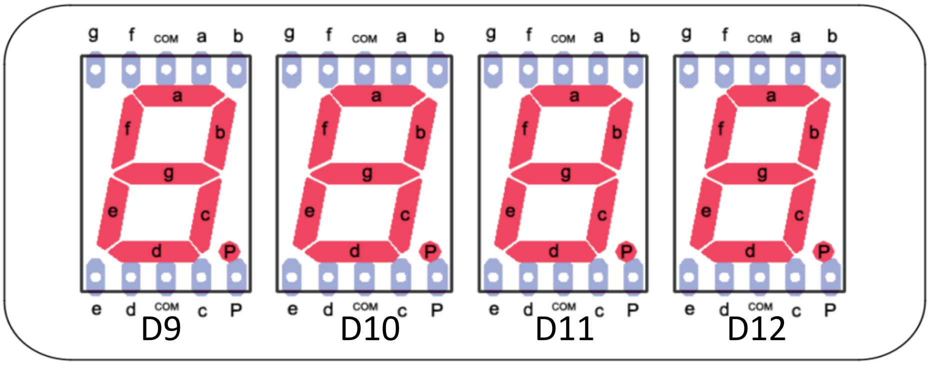
1. Configure the pins as external interrupts in PINSELx register.
2. Clear any pending interrupts in EXTINT.
3. Configure the EINTx as Edge/Level triggered in EXTMODE register.
4. Select the polarity(Falling/Rising Edge, Active Low/High) of the interrupt in EXTPOLAR register.
5. Finally enable the interrupts by calling NVIC_EnableIRQ() with IRQ number

Program to demonstrate the External interrupt:

When an interrupt occurs INT0, main program (7 Segment Display Execution) halts and wait till a key to be pressed (key P1.14). As soon as key pressed, it resumes the main operation

7-Segment Display Pin Configuration:

D12, D11, D10 and D9 are Common Cathode segments connected to LPC-1768 device so that each segment is individually controlled by a general purpose I/O pin. When the LPC-1768 device drives logic "0" the corresponding segment turns on.



Seven Segment Pin connection table:

7-Segment Data Lines	g	f	a	b	p	c	d	e
LPC1768 Port No.	P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7

7- SEGMENT Selection Pin Configurations:

This board comes with 4 digit seven segment unit. Displays connected to the microcontroller usually occupy a large number of valuable I/O pins, which can be a big problem especially if it is needed to display multi digit numbers. The problem is more than obvious if, for example, it is needed to display four digit numbers (a simple calculation shows that 32 output pins are needed in this case). The solution to this problem is called MULTIPLEXING. Only one digit is active at a time, but they change their state so quickly making impression that all digits of a number are simultaneously active. Each digit can made active using switching transistors Q1, Q2, Q3 and Q4 and these on switched on and off by selection lines which are in turn connected to LPC-1768 ports.

Seven Segment Selection table:

7-Segment selection Lines	Disp1 (D9)	Disp1 (D9)	Disp1 (D9)	Disp1 (D9)
LPC-1768 Port No.	P1.26	P2.27	P1.28	P1.29

PROGRAM:

```
#include <lpc17xx.h>

unsigned char data7[] = {0x88,0xeb,0x4c,0x49,0x2b,0x19,0x18,0xcb,
0x8,0x9,0xa,0x38,0x9C,0x68,0x1c,0x1e};

void intrupt_init()
{
LPC_PINCON->PINSEL4 = (1<<20);
LPC_SC->EXTINT = (1<<0) ;
LPC_SC->EXTMODE = (1<<0) ;
LPC_SC->EXTPOLAR = (1<<0);
}

void EINT0_IRQHandler(void)
{
    if (!(LPC_GPIO1->FIOPIN & (1<<14)))
    LPC_SC->EXTINT = (1<<0);
}

int main()
{
unsigned int i,j;
unsigned int count=0;
intrupt_init();
LPC_SC->EXTINT = (1<<0);
NVIC_EnableIRQ(EINT0_IRQn);
LPC_GPIO2->FIODIR = 0x000000FF;
LPC_GPIO1->FIODIR = 0x3C000000;
LPC_GPIO1->FIOSET=(1<<29);
LPC_GPIO1->FIODIR&=~(0x00008000);
```

```
while(1)
{
++count;
if (count > 0xF)
count = 0;
for (i=0; i < 25000; i++)
{
LPC_GPIO2->FIOPIN = data7[count];
for (j=0;j<1000;j++);

} }
}
```

Results & Observations:

Challenge for you:

Try writing a program to configure all FOUR 7-Segment Displays and count simultaneously.

DEMONSTRATION PROGRAMS

PROGRAM 10

Write a C program to Interface Real Time Clock and display the current time using LPC 1768.

Using inbuilt **Real Time Clock (RTC)** we will be writing a program to Display current TIME in **Hrs : Mins : Secs**

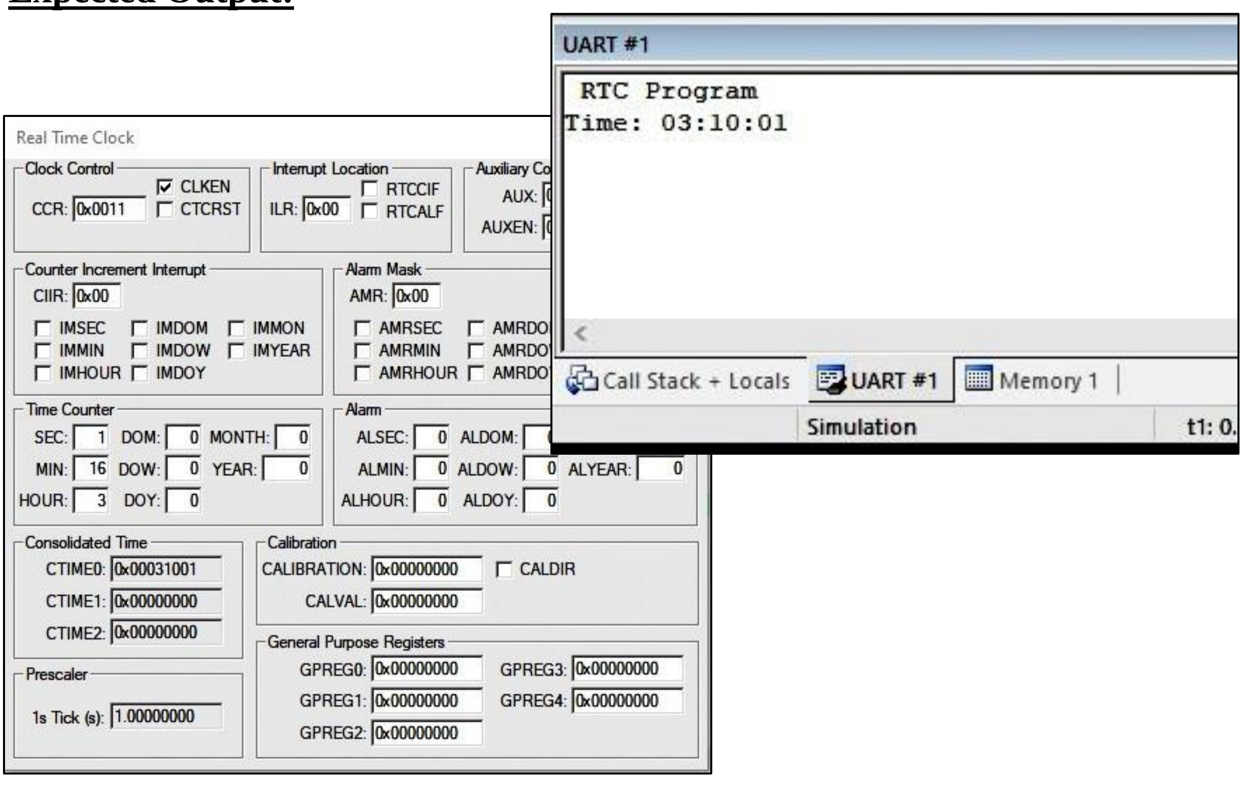
```
#include "LPC1768_Includes.h"
#include "UART.h"
#include <stdio.h>

void RTC_init(void)
{
        CIIR = 0x00;                    // no interrupt
        CCR = 0x11;
}

void RTC_SetTime(int hours, int mins, int sec)
{
        SEC  = sec;             // program the secs
        MIN  = mins;            // program the mins
        HOUR = hours;           // program the hours
        CCR =  0x11;            // start the clock
}

void RTC_GetTime(int *hour, int *min, int *sec)
{
        *sec = SEC;
        *min = MIN;
        *hour = HOUR;
}

void delay(unsigned int time)
{
        unsigned int i,j;
        for(i=0;i<time;i++)
        for(j=0;j<10000;j++);
}
```

```
int main()
{
        int hours, min, sec;
        UartInit(9600);
        RTC_init();
        RTC_SetTime(0x03, 0x10, 0x01);
        printf("RTC Program\r\n");
        while(1)
        {
                RTC_GetTime(&hours, &min, &sec);
                printf("Time: %02x:%02x:%02x\r\n",hours, min, sec);
                delay(600);
        }
}
```

NOTE:

Along with this Program add **LPC1768_Includes.h, UART.c and UART.h** supporting files and then BUILD the Program.

Expected Output:



Results & Observations:

Challenge for you:

Try writing a program to display Day of the Month(DOM), Day of the Week(DOW), Day of the Year(DOY), Month and Year.

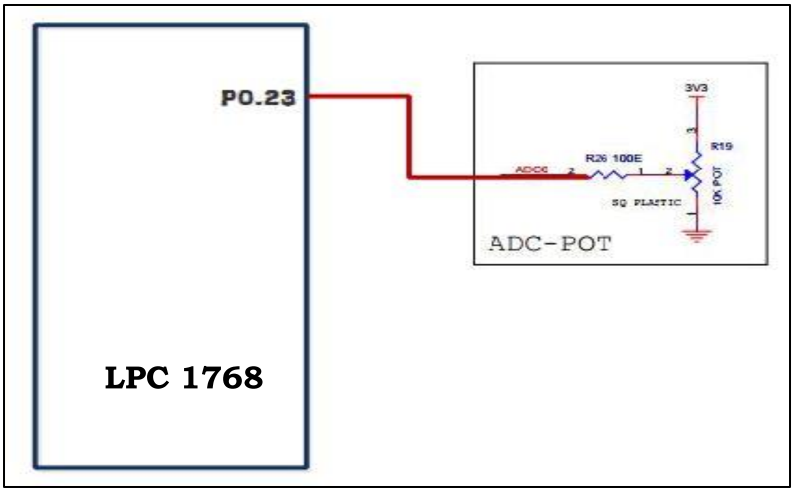
PROGRAM 11

<u>Write a C program to read on-chip ADC value and display it on LCD using LPC 1768.</u>

Features of ADC:

- 12-bit successive approximation analog to digital converter.
- Input multiplexing among 8 pins.
- Power-down mode.
- Measurement range VREFN to VREFP (typically 3 V; not to exceed VDDA voltage level).
- 12-bit conversion rate of 200 kHz.
- Burst conversion mode for single or multiple inputs.
- Optional conversion on transition on input pin or Timer Match signal. Basic clocking for the A/D converters is provided by the APB clock. A programmable divider is included in each converter to scale this clock to the clock (maximum 13 MHz) needed by the successive approximation process. A non-burst mode conversion requires 65 clocks and a burst mode conversion requires 64 clocks.

Interfacing Diagram:



Interfacing Details:

This board comes with two ADC inputs. One is connected to a potentiometer for external analog voltage and another is for temperature measurement. A **5K variable POT** is connected to **AD0(P0.23)** input of LPC1768. By varying this we are applying 0 to 3.3V to ADC0 input. This analog voltage input can be converted into digital output. An LM35 temperature sensor is connected through ADC SSP IC to SSEL input of LPC1768 for ambient temperature measurement.

PROGRAM:

```
 #include <LPC17xx.H>
#include <stdio.h>
#include "lcd.h“
#define ADC_DONE          0x80000000
#define ADC_OVERRUN       0x40000000
#define ADC_CLK   1000000                        /* set to 1Mhz */

volatile uint32_t ADCValue;

void ADCInit( uint32_t Clk )
{
  uint32_t pclk;
  LPC_SC->PCONP |= (1 << 12);
  LPC_PINCON->PINSEL1 &= ~0x0000C000;
  LPC_PINCON->PINSEL1 |= 0x00004000;
  LPC_PINCON->PINMODE1 &= ~0x0000C000;
  LPC_PINCON->PINMODE1 |= 0x00008000;

  pclk = SystemCoreClock/4;

  LPC_ADC->ADCR = ( 0x01 << 0 ) |
                  ( ( pclk  / Clk - 1 ) << 8 ) |
                  ( 0 << 16 ) |
                  ( 0 << 17 ) |
                  ( 1 << 21 ) |
                  ( 0 << 24 ) |
                  ( 0 << 27 );
  return;
}
uint32_t ADCRead( )
{
  uint32_t regVal, ADC_Data;

  LPC_ADC->ADCR &= 0xFFFFFF00;
  LPC_ADC->ADCR |= (1 << 24) | 1;
while ( 1 )
 {
```

```
regVal = LPC_ADC->ADDR0;
                if ( regVal & ADC_DONE ) break;
  }

  LPC_ADC->ADCR &= 0xF8FFFFFF;              /* stop ADC now */
  if ( regVal & ADC_OVERRUN )
   return ( 0 );

  ADC_Data = ( regVal >> 4 ) & 0xFFF;
  return ( ADC_Data );
}
int main (void)
{
                char lstr[10];
                double volt;
   ADCInit( ADC_CLK );

  init_lcd();

                lcd_putstring16(0,"RAW ADC = 0000  ");
                lcd_putstring16(1,"VOLTAGE = 0.00v ");

  while(1)
  {

                                ADCValue = ADCRead();
                                sprintf (lstr, "%4u", ADCValue);
                                lcd_gotoxy(0,10);
                                lcd_putstring(lstr);

                                volt = (double)ADCValue * 0.0008056640625;
                                sprintf (lstr, "%0.2f", volt);
                                lcd_gotoxy(1,10);
                                lcd_putstring(lstr);
                                delay(500);
                                }
}
```

NOTE:

Along with the above program add **lcd.c** and **lcd.h** Supporting files and then build the program.

Expected Output:

Results & Observations:

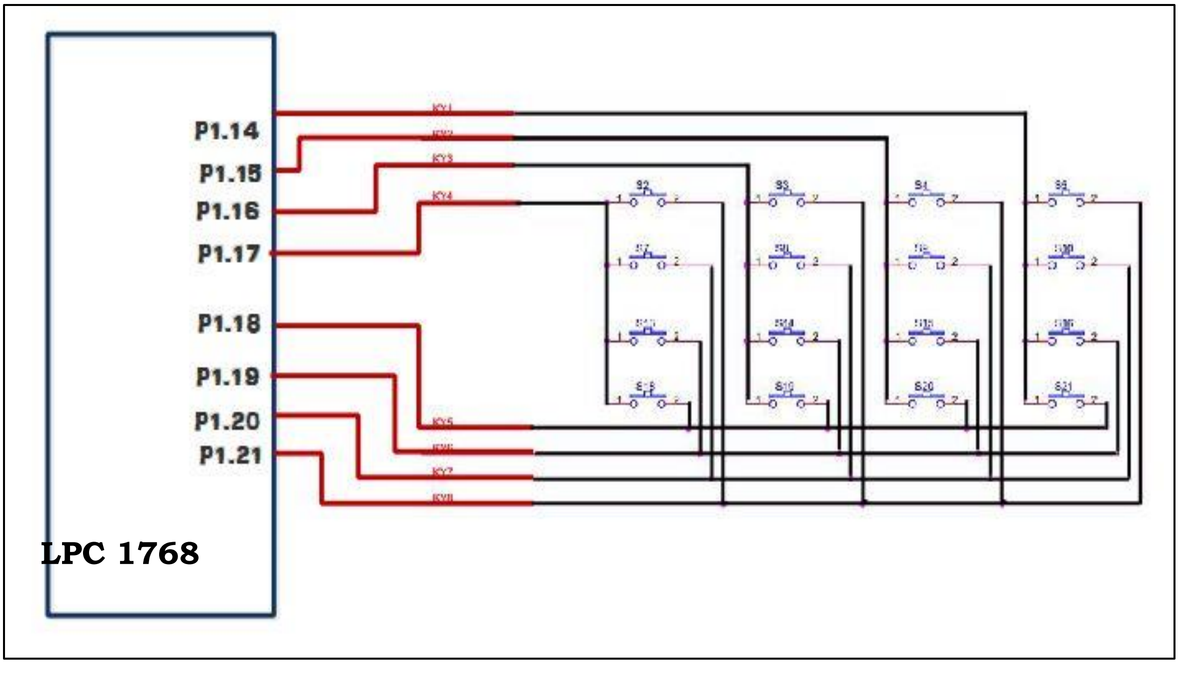
PROGRAM 12

Write a C program to Interface a 4x4 keyboard and display the key pressed on an LCD using LPC 1768.

HEX KEY PAD Pin configurations:

The hex keypad is a peripheral that is organized in rows and Columns. Hex key Pad 16 Keys arranged in a 4 by 4 grid, labeled with the hexadecimal digits 0 to F. Internally, the structure of the hex keypad is very simple. Wires run in vertical columns (we call them C0 to C3) and in horizontal rows (called R0 to R3). These 8 wires are available externally, and will be connected to the lower 8 bits of the port. Each key on the keypad is essentially a switch that connects a row wire to a column wire. When a key is pressed, it makes an electrical connection between the row and column. Table shows connections for HEX KEY Pad matrix.

Interfacing Diagram:



Interfacing Details:

ROW/COLOUMNS	ROW1	ROW2	ROW3	ROW4	COL1	COL2	COL3	COL4
LPC-1768 Pin No	32	33	34	35	89	88	87	86
LPC-1768 Port No	P1.18	P1.19	P1.20	P1.21	P1.14	P1.15	P1.16	P1.17

PROGRAM:

```
#include <LPC17xx.H>
#include <stdio.h>
#include "lcd.h"

void delay(unsigned int count)

{
unsigned int j=0,k=0;
for(j=0;j<count;j++)
{
for(k=0;k<12000;k++);
}
}

void LCD_DATA(unsigned char val)
{
LPC_GPIO2->FIOPIN= val;
LPC_GPIO0->FIOSET=(1<<28);

LPC_GPIO0->FIOSET=(1<<27);
delay(100);
LPC_GPIO0->FIOCLR=(1<<27);
delay(100);
}

void col_write( unsigned char data )
{
unsigned int temp=0;

temp=(data << 14) & 0X0003C000;
LPC_GPIO1->FIOCLR = 0X0003C000;
LPC_GPIO1->FIOSET = temp;
}

int main()
{
unsigned char array[100]={"PRESSED KEY IS"};
```

```
unsigned char i,j,key,key1;
unsigned char rval[ ] = {0x7,0xB,0xD,0xE,0x0};
unsigned char keyPadMatrix[ ] = {      'C','8','4','0',
                                       'D','9','5','1',
                                       ‘E','A','6','2',
                                       'F','B','7','3‘     };

LPC_GPIO0->FIODIR =0X18000000;
LPC_GPIO2->FIODIR=0x000000FF;
LPC_GPIO1->FIODIR=0X0003C000;
LPC_GPIO1->FIODIR&=~0X003C0000;

 init_lcd();

for(j=0; j<14; j++)
{
key1=array[j];
LCD_DATA(key1);
delay(100);
}
while(1)
{

key = 0;
for( i = 0; i < 4; i++ )
{
col_write(rval[i]);

if (!(LPC_GPIO1->FIOPIN & (1<<18)))
break;
key++;

if (!(LPC_GPIO1->FIOPIN & (1<<19)))
break;
key++;

if (!(LPC_GPIO1->FIOPIN & (1<<20)))
break;
key++;
```

```
if (!(LPC_GPIO1->FIOPIN & (1<<21)))
break;
key++;
}

lcd_command_write(0xC8);

if (!(key == 0x10))

LCD_DATA(keyPadMatrix[key]);
delay(10);
}
}
```

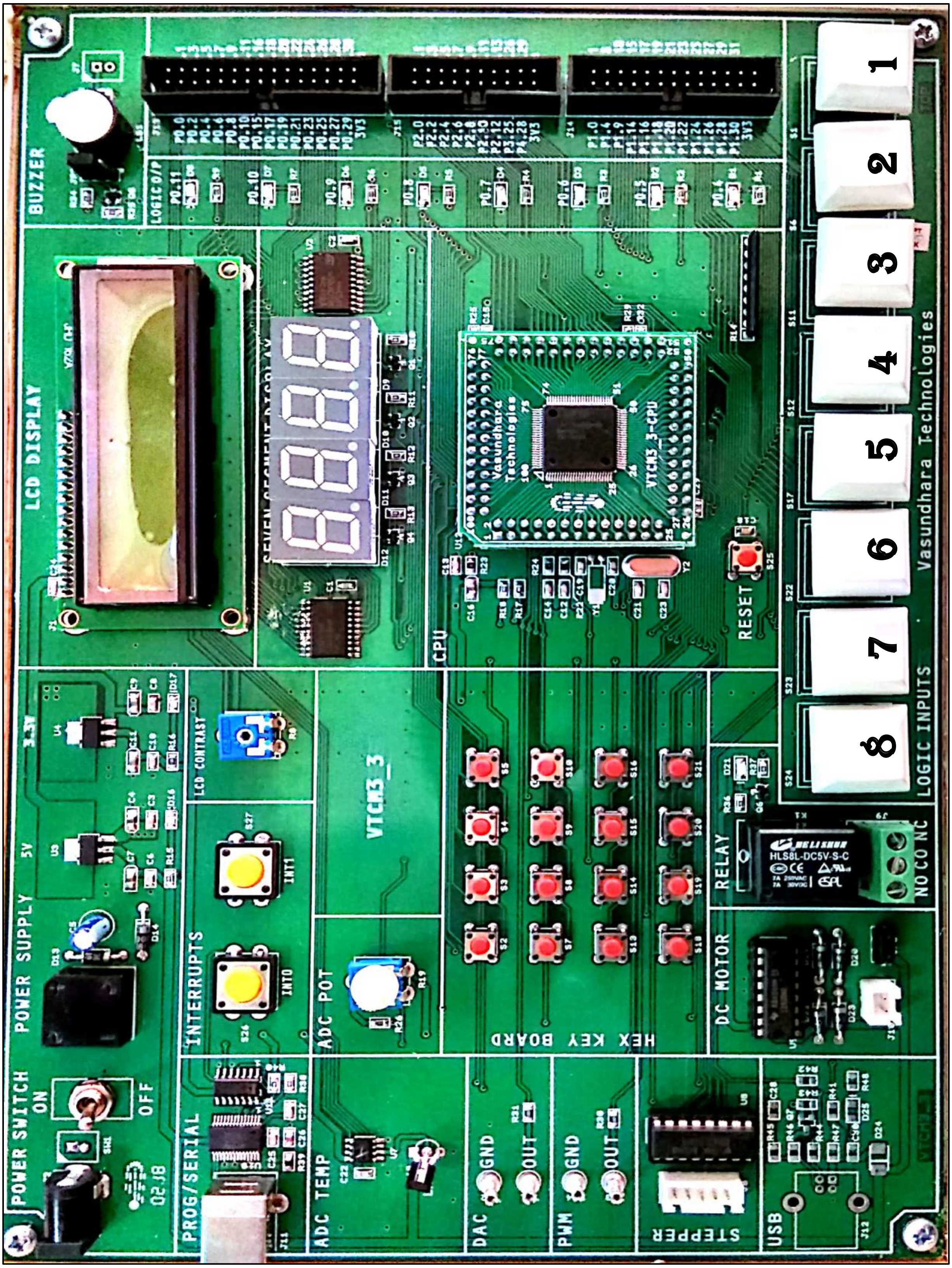
NOTE:

Along with the above program add **lcd.c** and **lcd.h** Supporting files and then build the program.

Results & Observations:

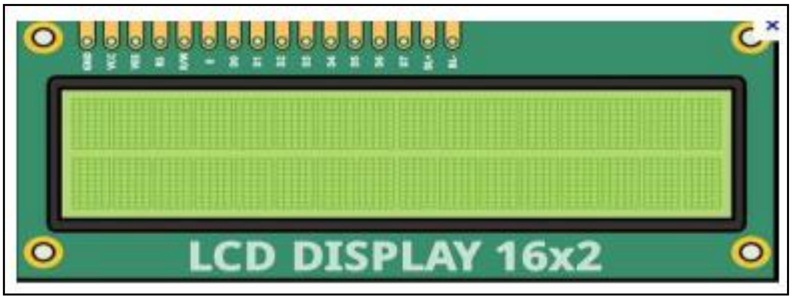
APENDIX

a. LPC-1768 Board:



b. 16 X 2 LCD Display:

An LCD display is specifically manufactured to be used with microcontrollers, which means that it cannot be activated by standard IC circuits. It is used for displaying different messages on a miniature liquid crystal display. It displays all the letters of alphabet, Greek letters, punctuation marks, mathematical symbols etc. In addition, it is possible to display symbols made up by the user. Other useful features include automatic message shift (left and right), cursor appearance, LED backlight etc.



There are pins along one side of a small printed board. These are used for connecting to the microcontroller. There are in total of 14 pins marked with numbers (16 if it has backlight). Their function is described in the table below:

LCD Pin No	LCD Pin Functions	LPC-1768 Pin No	LPC-1768 Port No
1	GND	-	-
2	VCC	-	-
3		-	-
4	RS	24	P2.28
5	R/W	GND	GND
6	EN	25	P2.25
7	DAT1	75	P2.0
8	DAT2	74	P2.1
9	DAT3	73	P2.2
10	DAT4	70	P2.3
11	DAT5	69	P2.4
12	DAT6	68	P2.5
13	DAT7	67	P2.6
14	DAT8	66	P2.7
15	VCC	-	-
16	GND	-	-

// lcd.c Program

```
#include <LPC17xx.H>
#include "lcd.h“
void lcd_command_write( unsigned char command )
{

                LCD_DATA_SET = command;
                LCD_CTRL_CLR |= LCDRS;
                LCD_CTRL_SET |= LCDEN;
  delay(1);
  LCD_CTRL_CLR |= LCDEN;
  delay(1);
}

void lcd_data_write( unsigned char data )
{

  LCD_DATA_SET = data;
  LCD_CTRL_SET |= LCDRS;
  LCD_CTRL_SET |= LCDEN;
  delay(1);
  LCD_CTRL_CLR |= LCDEN;
  delay(1);
}

void lcd_clear( void)
{
  lcd_command_write( 0x01 );
}

int lcd_gotoxy( unsigned char x, unsigned char y)
{
  unsigned char retval = TRUE;

  if( (x > 1) && (y > 15) )
   {
    retval = FALSE;
   }
  else
   {
    if( x == 0 ) lcd_command_write( 0x80 + y );
                else if( x==1 ) lcd_command_write( 0xC0 + y );
   }
   return retval;
}
```

```
void lcd_putchar( unsigned char c )
{
 lcd_data_write( c );
}

void lcd_putstring( char *string )
{
  while(*string != '\0')
 {
   lcd_putchar( *string );
   string++;
 }
}
void lcd_putstring16( unsigned char line, char *string )
{
 unsigned char len = 16;

 lcd_gotoxy( line, 0 );
 while(*string != '\0' && len--)
 {
   lcd_putchar( *string );
   string++;
 }
}

void init_lcd( void )
{
SEG_CTRL_DIR |= SEG_DIG_MASK;
SEG_CTRL_SET &= ~SEG_DIG_MASK;

LPC_GPIO0->FIODIR |= 0x00000FF0;
 LPC_GPIO0->FIOPIN &= ~0x00000FF0;
 LCD_CTRL_DIR |= ( LCDEN | LCDRS );
 LCD_DATA_DIR = LCD_DATA_MASK;

 delay(1000);
 lcd_command_write(0x38);    /*  8-bit interface, two line, 5X7 dots.      */
 lcd_command_write(0x38);
 lcd_command_write(0x38);
 lcd_command_write(0x10);    /*   display shift      */
 lcd_command_write(0x0C);    /*   display on         */
 lcd_command_write(0x06) ;   /*  cursor move direction */
 lcd_command_write(0x01) ;   /*  cursor home      */
 delay(1000);
}
```

// lcd.h Program

```
#ifndef _LCD_H
#define _LCD_H

#define TRUE 1
#define FALSE 0

#define LINE1 0x80
#define LINE2 0xC0

#define CONTROL_REG 0x00
#define DATA_REG    0x01

#define LCD_DATA_DIR        LPC_GPIO2->FIODIR0
#define LCD_DATA_SET        LPC_GPIO2->FIOPIN0
#define LCD_CTRL_DIR        LPC_GPIO0->FIODIR
#define LCD_CTRL_SET        LPC_GPIO0->FIOSET
#define LCD_CTRL_CLR        LPC_GPIO0->FIOCLR

#define SEG_DATA_DIR        LPC_GPIO2->FIODIR
#define SEG_DATA_SET        LPC_GPIO1->FIOPIN
#define SEG_CTRL_DIR        LPC_GPIO1->FIODIR
#define SEG_CTRL_SET        LPC_GPIO1->FIOPIN

#define LCDEN       (1 << 27)
#define LCDRS       (1 << 28)

//scale
31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16,15,14,13,12,11,10,09,08,07,06,05,04,03,
02,01,00
#define LCD_DATA_MASK               0x000000FF
#define SEG_DIG_MASK                0x0F000000

void delay(unsigned int count);
void init_lcd(void);
void lcd_putstring(char *string);
void lcd_putstring16(unsigned char line, char *string);
void lcd_clear(void);
int lcd_gotoxy(unsigned char x, unsigned char y);
void lcd_putchar(unsigned char c);
#endif
```

c. User Key Pad :

This board provides eight individual KEYS connected to LPC-1768 device through PORT1. S1 to S6,S11and S16 are connected to general purpose I/O pins on 2148 device as shown in table. 2148 device port pin drives to Logic "0" when the corresponding key is pressed. Table shows connections for USER KEY Pad connection.

USER DEFINED KEYS	S1	S6	S11	S12	S17	S22	S23	S24
LPC-1768 Pin No	1	2	3	4	5	6	7	8
LPC-1768 Port No	P1.14	P1.15	P1.16	P1.17	P1.18	P1.19	P1.20	P1.21

Sample Viva-Voce Question

1. Name the Device/Board used in your lab.
2. Difference b/w Microcontroller and Microprocessor.
3. LPC-1768 belongs to which microcontroller Family.
4. Name the different Profiles of ARM Microcontroller.
5. You are working with which Profile of ARM Microcontroller.
6. Cortex-M3 is based on which Architecture?
7. List Special Features of Cortex-M3.
8. Difference b/w Assembly level Programs and C-Programs.
9. Explain Assembly level Instructions: DCD, BNE, SUBS, BX lr, STR etc.
10. Name Communication Interfaces used in Embedded Systems and their features.
11. Explain Characteristics of UART Communication Protocol.
12. What is Baud Rate?
13. Explain Steps involved in Configuring UART in LPC-1768.
14. Name any Input and Output Devices used in Embedded System.
15. Explain features of LEDs and their applications.
16. Explain function of Relay Coil and its applications.
17. Differences b/w DC Motor and Stepper Motor and their operation.
18. Explain features of ADC available in LPC-1768 and its applications.
19. Explain Steps involved in Configuring ADC in LPC-1768.
20. Explain features of DAC available in LPC-1768 and its applications.
21. Explain Steps involved in Configuring DAC in LPC-1768.
22. Explain Interrupt Service Routine (ISR) and Context Switching.
23. Explain features of 7-Segment Displays and their applications.
24. Explain about interrupts of Cortex-M3.
25. Explain features of Hex Keypad.

………