COURSE LABORATORY MANUAL

A. LABORATORY OVERVIEW

Degree:	BE	Programme:	AIML/CSE
Semester:	4	Academic Year:	2021-22
Laboratory Title:	**Microcontroller and Embedded Systems Laboratory**	Laboratory Code:	**18CSL48**
L-T-P-S:	0-2-2-0	Duration of SEE:	3 Hrs
Total Contact Hours:	36 Hrs	SEE Marks:	60*
Credits:	2	CIE Marks:	40
Lab Manual Author:	Dr. Mahesh Prasanna K	Sign	Dt : 04/04/2022
Checked By:	Dr Govindaraj P	Sign	Dt : 04/04/2022

*The SEE will be conducted for 100 marks and proportionally reduced to 60 marks.

B. DESCRIPTION

1. PREREQUISITES:

- Microcontroller and Embedded Systems (18CS44)
- Basic Electronics (18ELN15/25)
- Analog and Digital Electronics(18CS33)
- Computer Programming Laboratory (18CPL17/27)
- Programming in C and Data Structures (18PCDS13/23).

2. BASE COURSE:

- Microcontroller and Embedded Systems (18CS44)

3. COURSE OUTCOMES:

At the end of the course, the student will be able to;

1. Apply the programming techniques in designing simple assembly language programs for solving simple problems by using ARM instruction set.
2. Gain hands-on experience in doing experiments on ARM by using simulator in the laboratory and present the report.
3. Understand the hardware, software tradeoffs involved in the design and interface hardware devices to ARM family.
4. Assess processors for various kinds of applications.

4. RESOURSES REQUIRED:

- ARM7TDMI/LPC2148 Evaluation Board
- Keil uVision-4 tool
- Interfacing Kits
- Embedded 'C'.

Prepared by: Dr Mahesh Prasanna K Checked by: Govindaraj P HOD

5. RELEVANCE OF THE COURSE:

- Microcontroller and Embedded Systems (18CS44)
- System Software and Compilers (18CS61) & Introduction to Operating System (18CS654)
- Artificial Intelligence and machine Learning (18CS71) & Advanced Computer Architecture (18CS733).

6. GENERAL INSTRUCTIONS:

Introduction to Keil Software:

The uVision3/4 IDE is a windows based software development platform that combines a robust editor, project manager, and makes facility. Uvision3 integrates all tolls including the C compiler, macro assembler, linker/locator, and HEX file generator. UVision3 provides,

- Full – featured source code editor
- Device database for configuring the development tool setting
- Project manager for creating and maintaining projects
- Integrated make facility for assembling, compiling, and linking your embedded applications
- Dialogs for all development tool settings
- True integrated source-level debugger with high speed CPU and peripheral simulator
- Advanced GDI interface for software debugging in the target hardware and for connection to KEIL ULINK
- Flash programming utility for downloading the application program into Flash ROM.

The Keil Software Flow:

Open the icon keil uvision and the following steps are given below. The menu bar provides you with manus for editor operations, project maintenance, development tool option settings, program debugging, external tool control, window selection and manipulation, and on-line help. The toolbar button allow you to rapidly execute uVision commands. A status bar provides editor and debugger information. The various toolbars and the status bar can be enabled or disabled from the view menu commands.

Creation of Project in Keil:

Step1: Go to “Project” and close the current project “Close Project”

Step2: Got o “Project” and click on “New Micro vision Project”

Step3: The new window named “Create New Project” popup and here select destination to save the project.

Step4: On new “Select Device for Target ‘Target 1’”, select NXP founded by Philips and then. select LPC2148 and click “Yes/No”. [Note: For .ASM – don't add header file & for .C file – add header file].

Step5: Go to Next, File>>New

Step6: Project workspace, editor window and Output window will appear. Write program on editor window.

Step7: Save the program file, if the prgram is in “C” save as “filename.C” or else it is in ASM save as “filename.ASM”.

Step8: Go to “Project workspace” >> “Target 1” >> “Add Files Group ‘source group1’” on Add files to the ‘Source Group’ window >> Add.

Step9: Go to “Project” >> “Build Target” or “F7”

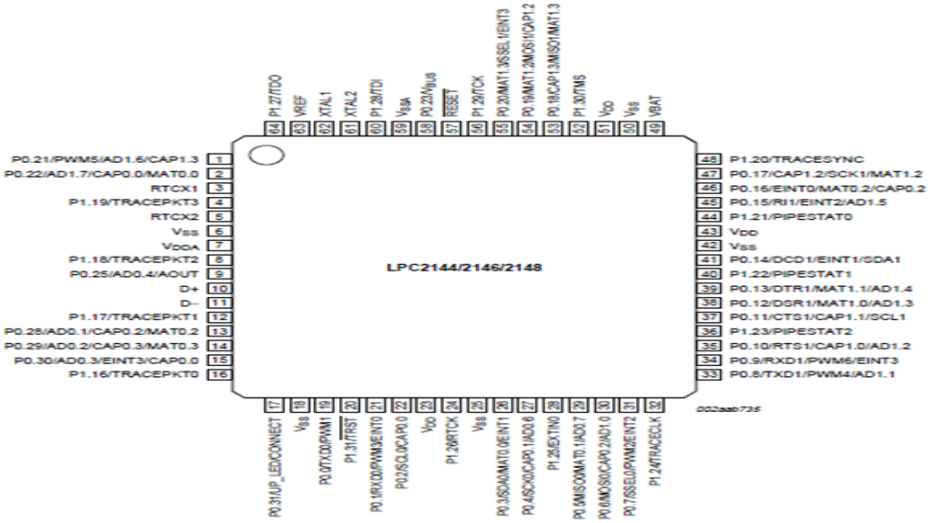
Step10: Debug >> “Satrt/stop Debug Session”. Use F11 key for single step execution.

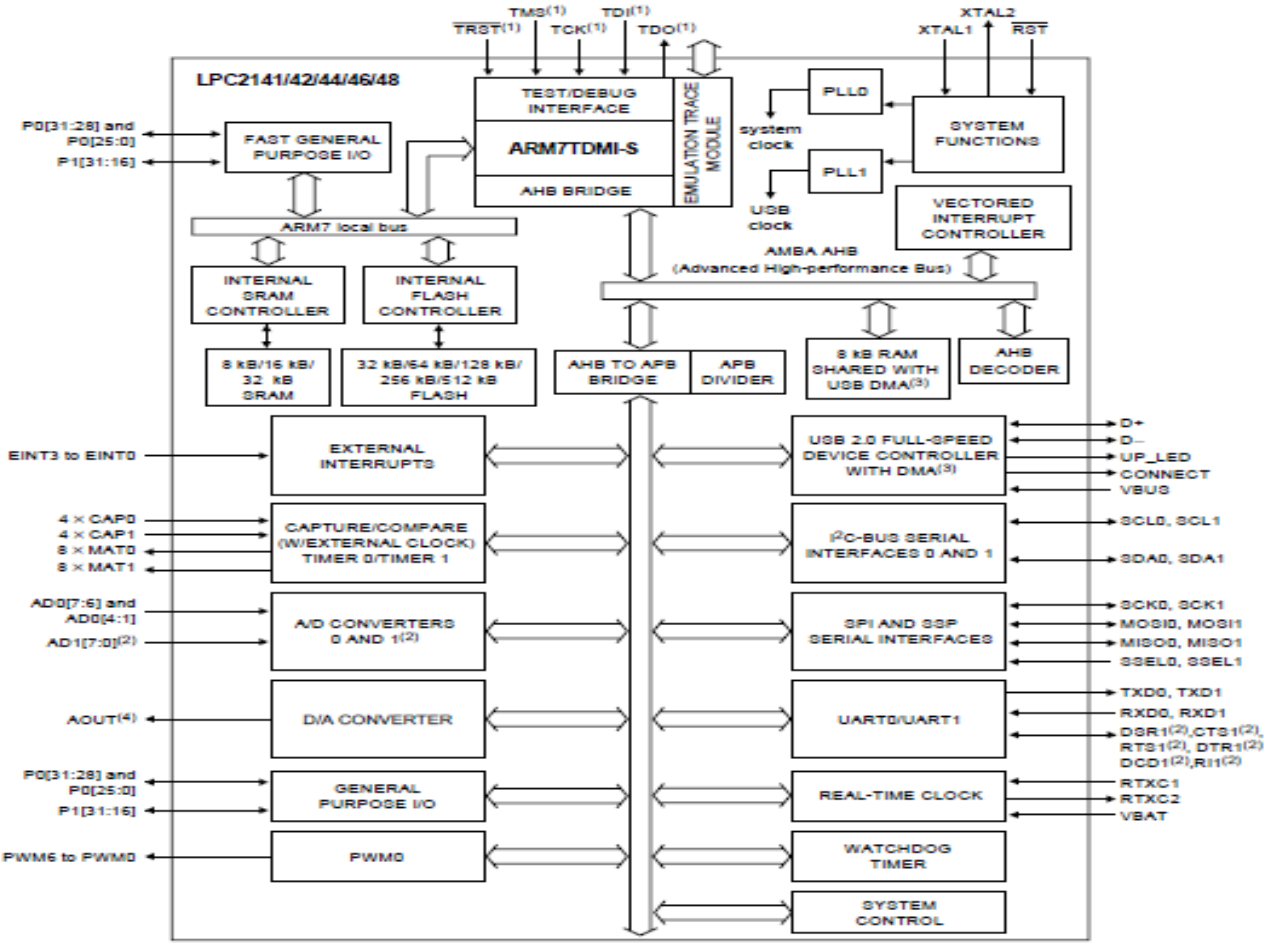
LPC2148

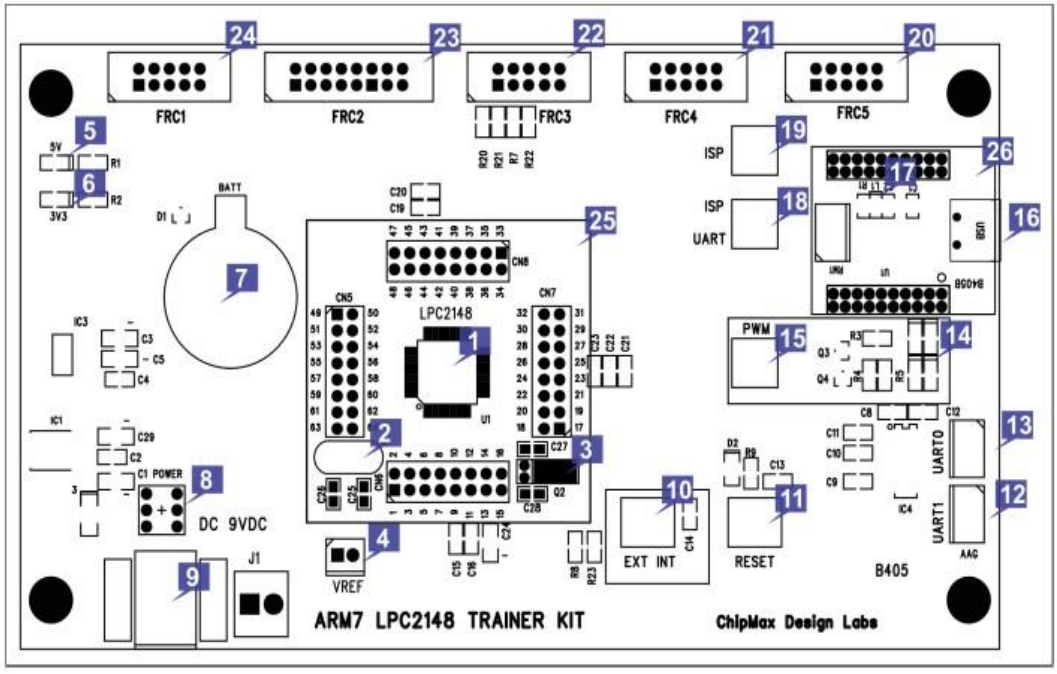
LPC2148 is the widely used IC from ARM-7 family. It is manufactured by Philips and it is pre-loaded with many inbuilt peripherals making it more efficient and a reliable for high end application developer.

- 8 to 40 kB of on-chip static RAM and 32 to 512 kB of on-chip flash program memory.
- 28 bit wide interface/accelerator enables high speed 60 MHz operation.
- In-System/In-Application Programming (ISP/IAP) via on-chip boot-loader software.
- Single flash sector or full chip erase in 400 ms and programming of 256 bytes in 1ms.
- USB 2.0 Full Speed compliant Device Controller with 2 kB of endpoint RAM. In addition, the LPC2146/8 provides 8 kB of on-chip RAM accessible to USB by DMA.
- Low power real-time clock with independent power and dedicated 32 kHz clock input.
- Multiple serial interfaces including two UARTs (16C550), two Fast I2C-bus(400kbit/s), SPI and SSP with buffering and variable data length capabilities.
- Vectored interrupt controller with configurable priorities and vector addresses.
- Up to nine edge or level sensitive external interrupt pins available.
- On-chip integrated oscillator operates with an external crystal in range from 1 MHz to30 MHz and with an external oscillator up to 50 MHz.
- Power saving modes include Idle and Power-down.
- Individual enable/disable of peripheral functions as well as peripheral clock scaling for additional power optimization.
- Processor wake-up from Power-down mode via external interrupt, USB, Brown- OutDetect (BOD) or Real-Time Clock (RTC).

LPC2148




LPC2141/42/44/46/48
TEST/DEBUG INTERFACE
ARM7TDMI-S
AHB BRIDGE
EMULATION TRACE MODULE
FAST GENERAL PURPOSE I/O
ARM7 local bus
INTERNAL SRAM CONTROLLER
INTERNAL FLASH CONTROLLER
8 kB/16 kB/ 32 kB SRAM
32 kB/64 kB/128 kB/ 256 kB/512 kB FLASH
AHB TO APB BRIDGE
APB DIVIDER
8 kB RAM SHARED WITH USB DMA
AHB DECODER
PLL0
PLL1
SYSTEM FUNCTIONS
VECTORED INTERRUPT CONTROLLER
AMBA AHB (Advanced High-performance Bus)
EXTERNAL INTERRUPTS
CAPTURE/COMPARE (W/EXTERNAL CLOCK) TIMER 0/TIMER 1
A/D CONVERTERS 0 AND 1
D/A CONVERTER
GENERAL PURPOSE I/O
PWM0
USB 2.0 FULL-SPEED DEVICE CONTROLLER WITH DMA
I2C-BUS SERIAL INTERFACES 0 AND 1
SPI AND SSP SERIAL INTERFACES
UART0/UART1
REAL-TIME CLOCK
WATCHDOG TIMER
SYSTEM CONTROL
TRST TMS TCK TDI TDO
XTAL1 XTAL2 RST
P0[31:28] and P0[25:0]
P1[31:16]
EINT3 to EINT0
4 × CAP0
4 × CAP1
8 × MAT0
8 × MAT1
AD0[7:6] and AD0[4:1]
AD1[7:0]
AOUT
P0[31:28] and P0[25:0]
P1[31:16]
PWM6 to PWM0
D+
D−
UP_LED
CONNECT
VBUS
SCL0, SCL1
SDA0, SDA1
SCK0, SCK1
MOSI0, MOSI1
MISO0, MISO1
SSEL0, SSEL1
TXD0, TXD1
RXD0, RXD1
RTXC1
RTXC2
VBAT


FRC1
FRC2
FRC3
FRC4
FRC5
ISP
ISP
UART
PWM
LPC2148
DC 9VDC
VREF
EXT INT
RESET
UART0
UART1
B405
ARM7 LPC2148 TRAINER KIT
ChipMax Design Labs

Hardware Design of ARM 7 LPC2148 Trainer Kit:

1. The device LPC2148 from NXP.
2. 12.00 MHz Crystal Oscillator Device speed.
3. 32.768 KHz crystal oscillator for RTC clock.
4. VREF jumper setting for ADC external Voltage reference.
5. 5V Red LED indication.
6. 3V3 Green LED indication.
7. Battery for RTC backup running.
8. Power ON/OFF switch.
9. 9VDC input Power Jack.
10. Input switch for external interrupt.

    Optional: can be use for ISP manual Programming Mode.
11. Input switch for Reset the device.
12. ART1 connectivity.

Note: For In System Programming (ISP), the switch which has been mentioned in the description line 18 and 19 has to be pressed.

13. UART0 connectivity.

    Optional: Can be use for ISP programming mode.
14. LEDs for PWM output.
15. PWM enable switch.
16. USB connector for ISP programming and Serial port connectivity.
17. LED indication for to confirm the USB connection has been established.
18. Switch selection for ISP/UART. The selection will be for ISP if the switch has been pressed else for UART.
19. Switch for to enable the ISP.
20. I/O port 10 pin FRC connector (FRC5).
21. I/O port 10 pin FRC connector (FRC4).
22. I/O port 10 pin FRC connector (FRC3).
23. I/O port 16 pin FRC connector (FRC2).
24. I/O port 10 pin FRC connector (FRC1).
25. LPC2148 Device Daughter card.
26. USB ISP programmer daughter card.

Device Daughter Board Details:

CN6	
PIN NO:	**DESCRIPTION**
1	P0.21/PWM5/AD1.6/CAP1.3
2	P0.22/AD1.7/CAP0.0/MAT0.0
3	RTCX1
4	P1.19/TRACEPKT3
5	RTCX2
6	VSS
7	VDDA
8	P1.18/TRACEPKT2
9	P0.25/AD0.4/AOUT
10	D+
11	D-
12	P1.17/TRACEPKT1
13	P0.28/AD0.1/CAP0.2/MAT0.2
14	P0.29/AD0.2/CAP0.3/MAT0.3
15	P0.30/AD0.3/CAP0.0/EINT3
16	P1.16/TRACEPKT0

CN8	
PIN NO:	**DESCRIPTION**
33	P0.8/TXD1/PWM4/AD1.1
34	P0.9/RXD1/PWM6/EINT3
35	P0.10/RTS1/CAP1.0/AD1.2
36	P1.23/PIPESTAT2
37	P0.11/CTS1/CAP1.1/SCL1
38	P0.12/DSR1/MAT1.0/AD1.3
39	P0.13/DTR1/MAT1.1/AD1.4
40	P1.22/PIPESTAT1
41	P0.14/DCD1/EINT1/SDA1
42	VSS
43	VDD
44	P1.21/PIPESTAT0
45	P0.15/RI1/EINT2/AD1.5
46	P0.16/EINT0/MAT0.2/CAP0.2
47	P0.17/CAP1.2/SCK1/MAT1.2
48	P1.20/TRACESYNC

CN7	
PIN NO:	**DESCRIPTION**
17	P0.31/UP_LED/CONNECT
18	VSS
19	P0.0/TXD0/PWM1
20	P1.31/TRST
21	P0.1/RXD0/PWM3/EINT0
22	P0.2/SCL0/CAP0.0
23	VDD
24	P1.26/RTCK
25	VSS
26	P0.3/SDA0/MAT0.0/EINT1
27	P0.4/SCK0/CAP0.1/AD0.6
28	P1.25/EXTIN0
29	P0.5/MISO0/MAT0.1/AD0.7
30	P0.6/MOSI0/CAP0.2/AD1.0
31	P0.7/SSEL0/PWM2/EINT2
32	P1.24/TRACECLK

CN5	
PIN NO:	**DESCRIPTION**
49	VBAT
50	VSS
51	VDD
52	P1.30/TMS
53	P0.18/CAP1.3/MISO1/MAT1.3
54	P0.19/CAP1.2/MOSI1/MAT1.2
55	P0.20/EINT3/SSEL1/MAT1.3
56	P1.29/TCK
57	RESET
58	P0.23/VBUS
59	VSSA
60	P1.28/TDI
61	XTAL2
62	XTAL1
63	VREF
64	P1.27/TDO

FRC Pin Details:

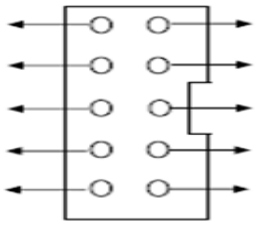
FRC 1	
PIN NO:	DESCRIPTION
1	P0.16
2	P0.17
3	P0.18
4	P0.19
5	P0.20
6	P0.21
7	P0.22
8	P0.23
9	+5V
10	GND

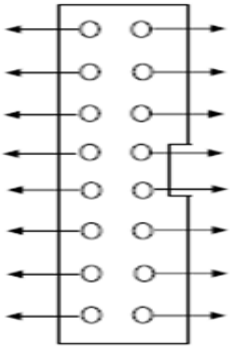
FRC 2	
PIN NO:	DESCRIPTION
1	P0.8
2	P0.9
3	P0.10
4	P0.11
5	P1.16
6	P1.17
7	P1.18
8	P1.19
9	P1.20
10	P1.21
11	P1.22
12	P1.23
13	+5V
14	NA
15	+3V3
16	GND

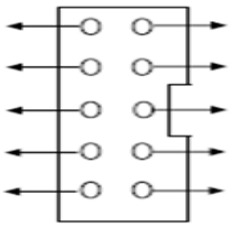
FRC 3	
PIN NO:	DESCRIPTION
1	P0.3(SDA0)
2	P0.2(SCL0)
3	P0.14(SDA1)
4	P0.11(SCL1)
5	P0.16 ENT1
6	P0.15 ENT2
7	RS232 TX0
8	RS232 RX0
9	+5V
10	GND

FRC 5	
PIN NO:	DESCRIPTION
1	P0.25
2	P0.28
3	P0.29
4	P0.30
5	P0.31
6	NA
7	RS232 TX1
8	RS232 RX1
9	+5V
10	GND

FRC 4	
PIN NO:	DESCRIPTION
1	P0.0
2	P0.1
3	P0.2
4	P0.3
5	P0.4
6	P0.5
7	P0.6
8	P0.7
9	+5V
10	GND







LED MODULE	
PIN NO	DESCRIPTION
1	Led1
2	Led2
3	Led3
4	Led4
5	Led5
6	Led6
7	Led7
8	Led8
9	+5V
10	GND

MATRIX KEYPAD	
PIN NO	DESCRIPTION
1	Row1
2	Row2
3	Row3
4	Row4
5	Column1
6	Column2
7	Column3
8	Column4
9	+5V
10	GND

STEPPER MOTOR	
PIN NO	DESCRIPTION
1	Input1
2	Input2
3	Input3
4	Input4
5	NA
6	NA
7	NA
8	NA
9	+5V
10	GND

LCD	
PIN NO	DESCRIPTION
1	NA
2	RS
3	R/W
4	EN
5	D0
6	D1
7	D2
8	D3
9	D4
10	D5
11	D6
12	D7
13	+5V
14	-5V
15	+3V3
16	GND

ADC / TEMPERATURE	
PIN NO	DESCRIPTION
1	NA
2	AD01
3	NA
4	NA
5	NA
6	NA
7	NA
8	NA
9	+5V
10	GND

DAC	
PIN NO	DESCRIPTION
1	NA
2	NA
3	NA
4	NA
5	A8
6	A7
7	A6
8	A5
9	A4
10	A3
11	A2
12	A1
13	+5V
14	-5V
15	NA
16	GND

TOGGLE SWITCH	
PIN NO	DESCRIPTION
1	SW1
2	SW2
3	SW3
4	SW4
5	SW5
6	SW6
7	SW7
8	SW8
9	+5V
10	GND

DC MOTOR	
PIN NO	DESCRIPTION
1	INPUT1
2	ENABLE
3	NA
4	INPUT2
5	NA
6	NA
7	NA
8	NA
9	+5V
10	GND

SEVEN SEGMENT	
PIN NO	DESCRIPTION
1	SHIFT CLK
2	DATA IN
3	LATCH CLK
4	NA
5	NA
6	NA
7	NA
8	NA
9	+5V
10	GND

FRC Connection Details:

Experiment	FRC1	FRC2	FRC3	FRC4	FRC5
DC Motor	Keypad			DC Motor	
Stepper Motor	Stepper Motor				
ADC		LCD			ADC (Change mode)
DAC					DAC
Keypad	Keypad	LCD			
Interrupt	LED				
Seven Segment Display				Seven Segment	

Steps to Interface Hardware Kits:

10) Steps 0 to 9, as explained earlier (Creation of Project in Keil – Slide No. 7).

11) Go to Project Window >> Right Click "Target 1" >> Select "Target"

>> Select "Thumb Mode" from the drop-down list.

>> 'Check' – Use Cross Module Option.

>> No Init – 'Check' against IRAM1 (both sides).

>> Select "Output"

>> 'Check' – Create HEX File.

>> Select "Listing"

>> 'Check' – C Preprocessor Listing.

>> Select "Linker"

>> 'Check' – Use Memory Layout.

>> OK.

12) Go to "Project" >> "Build Target" or "F7".

13) Connect the Interfacing Kit through USB Port. Give Power supply & Switch on the Kit.

a) Make sure; PWM – Off, ISP & UART – ON Position.

14) My Computer >> Right Click >> Device Manager >> Identify the COM Port (Say COM3).

15) Double Click "Flash Magic".

a) Select COM3.

b) Select proper HEX File, by using 'Browse'.

c) 'Check' – Erase all Flash + Code.

d) 'Check' – Verify after Programming.

e) Give >> START

Example

TTL ARM IN ALP FOR DATA TRANSFER OPERATIONS

```
        AREA PROG, CODE, READONLY ENTRY
MAIN
        MOV r0, #1              ; r0 = 0x00000001
        MOV r1, #0x21           ; r1 = 0x00000021
        MOV r2, #21             ; r2 = 0x00000015
        MOV r3, r1              ; r3 = r1 = 0x00000021
        MVN r0, #0xff           ; r0 = 0xffffff00, move the NOT of 32-bit value
        MVN r0, r3              ; r0 = ~r3 = 0xfffffde
        LDR r0, [r1]            ; load r0 with the contents of the memory address pointed by r1
        STR r0, [r1]            ; store the contents of r0 to the memory address pointed by r1
HERE B HERE
        END
```

TTL ARM IN ALP FOR ARITHMETIC OPERATIONS

```
        AREA PROG, CODE, READONLY ENTRY
MAIN
        MOV r0, #0              ; r0 = 0x00000000
        MOV r1, #2              ; r1 = 0x00000002
        MOV r2, #1              ; r2 = 0x00000001
        MOV r3, #3              ; r3 = 0x00000003
        ADD r0, r1, r2          ; r0 = r1 + r2 = 0x00000003
        SUB r0, r1, r2          ; r0 = r1 – r2 = 0x00000001
        RSB r0, r1, #0          ; r0 = 0 – r1 = 0xfffffffe
        MUL r0, r1, r2          ; r0 = r1 * r2 = 0x00000002
        MLA r0, r1, r2, r3      ; r0 = (r1 * r2) + r3 = 0x00000005
HERE B HERE
        END
```

TTL ARM IN ALP FOR LOGICAL OPERATIONS

```
        AREA PROG, CODE, READONLY ENTRY
MAIN
; Illustration of LOGICAL Instructions */
        MOV r0, #0                  ;  r0  =  0x00000000
        MOV r1, #0x00000004 ; r1 = 0x00000004 MOV
        r2, #0x00000006 ; r2 = 0x00000006
        ORR r0, r1, r2              ; r0 = 0x00000006 (logical bitwise OR)
        AND r0, r1, r2              ; r0 = 0x00000004 (logical bitwise AND)
        EOR r0, r1, r2              ; r0 = 0x00000002 (logical bitwise XOR)
        BIC r0, r1, r2              ; r0 = 0x00000000 (logical bit clear (AND NOT))
; Illustration of SHIFT/ROTATE Instructions */
        MOV r5, #5                  ; r5 = 0x00000005
        MOV r6, #4                  ; r6 = 0x00000004
        MOV r7, #8                  ; r7 = 0x00000008
        MOV r8, #0x80000004         ; r8 = 0x80000004
        MOV r0, r7, LSL #2          ; r0 = 0x00000020, logical shift left by 2
        MOV r0, r7, LSR #2          ; r0 = 0x00000002, logical shift right by 2
        MOV r0, r8, ASR #2          ; r0 = 0xe0000001, arithmetic shift right by 2
        MOV r0, r7, ROR #2          ; r0 = 0x00000002, rotate right by 2
        MOV r0, r7, ROR r4          ; r0 = 0x00000008, rotate right by r4
        MOV r0, r7, ROR r5          ; r0 = 0x40000000, rotate right by r5
        MOV r7, r5, LSL #2          ; r7 = r5 * 4 = (r5 << 2) = 0x00000014
        MOV r1, #0x80000004         ; r1 = 0x80000004
        MOV r0, r1, LSL #1          ; r0 = 0x00000008, r1 = 0x80000004
HERE B HERE
        END
```

// C PROGRAM FOR ARM MICROPROCESSOR USING KEIL

```
#include "LPC21xx.H"
main ()
{
        int a=7, b=5,c,sum,mult,divi,modu;
        sum=a+b;
        mult=a*b;
```

```
    divi=a/b;
    modu=a%b;
    ++a;            // pre-increment
    --b;            // pre-decrement
    a++;            // post increment
    b--;            // post decrement
    c=a&b;          // bit-wise logical AND
    c=a|b;          // bit-wise logical OR
    c=a^b;          // bit-wise logical XOR
    c=~a;           // bit-wise logical NOT
    c=a<<1;         // left shift
    c=a>>1;         // right shift
}
```

Steps to Interface Hardware Kits:

10) Steps 1 to 9, as explained earlier (Creation of Project in Keil).
11) Go to Project Window >> Right Click “Target 1” >> Select “Target”
    >> Select “Thumb Mode” from the drop-down list.
    >> 'Check' – Use Cross Module Option.
    >> No Init – 'Check' against IRAM1 (both sides).
    >> Select “Output”
    >> 'Check' – Create HEX File.
    >> Select “Listing”
    >> 'Check' – Preprocessor Listing.
    >> Select “Linker”
    >> 'Check' – Use Memory Layout.
    >> OK.
12) Go to “Project” >> “Build Target” or “F7”.
13) Connect the Interfacing Kit through USB Port. Give Power supply & Switch on the Kit.
    a) Make sure; PWM – Off, ISP & UART – ON Position.
14) My Computer >> Right Click >> Device Manager >> Identify the COM Port (Say COM3).
15) Double Click “Flash Magic”.
    a) Select COM3.
    b) Select proper HEX File, by using 'Browse'.
    c) 'Check' – Erase all Flash + Code.
    d) 'Check' – Verify after Programming.

    Give >> START.

COURSE LABORATORY MANUAL

Expt No.	Title of the Experiments	RBT	CO
PART A: Conduct the following experiments by writing program using ARM7TDMI/LPC2148 using an evaluation board/simulator and the required software tool.			
1	Write a program to multiply two 16 bit binary numbers.	L2	CO1,2,3,4
2	Write a program to find the sum of first 10 integer numbers.	L2	CO1,2,3,4
3	Write a program to find factorial of a number.	L2	CO1,2,3,4
4	Write a program to add an array of 16 bit numbers and store the 32 bit result in internal RAM.	L2	CO1,2,3,4
5	Write a program to find the square of a number (1 to 10) using look-up table.	L2	CO1,2,3,4
6	Write a program to find the largest/smallest number in an array of 32 numbers.	L2	CO1,2,3,4
7	Write a program to arrange a series of 32 bit numbers in ascending/descending order.	L2	CO1,2,3,4
8	Write a program to count the number of ones and zeros in two consecutive memory locations.	L2	CO1,2,3,4
PART B: Conduct the following experiments on an ARM7TDMI/LPC2148 evaluation board using evaluation version of Embedded 'C' & Keil Uvision-4 tool/compiler.			
9	Display “Hello World” message using Internal UART.	L3	CO1,2,3,4
10	Interface and Control a DC Motor.	L3	CO1,2,3,4
11	Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.	L3	CO1,2,3,4
12	Determine Digital output for a given Analog input using Internal ADC of ARM controller.	L3	CO1,2,3,4
13	Interface a DAC and generate Triangular and Square waveforms.	L3	CO1,2,3,4
14	Interface a 4x4 keyboard and display the key code on an LCD.	L3	CO1,2,3,4
15	Demonstrate the use of an external interrupt to toggle an LED On/Off.	L3	CO1,2,3,4
16	16. Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between	L3	CO1,2,3,4
17	Open ended experiment – 1 Interfacing of temperature sensor with ARM freedom board (or any other ARM microprocessor board) and display temperature on LCD	L4	CO1,2,3,4
18	Open ended experiment – 2 To design ARM cortex based automatic number plate recognition system	L4	CO1,2,3,4

C. EVALUATION SCHEME

For CBCS 2017 and 2018 scheme:

- For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution (Coursed *to change in accordance with University regulations*)
- For laboratories having PART A and PART B
  - i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
  - ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks.

-

D1. ARTICULATION MATRIX

Mapping of CO to PO												
	POs											
COs	1	2	3	4	5	6	7	8	9	10	11	12
1. Apply the programming techniques in designing simple assembly language programs for solving simple problems by using ARM instruction set.	2	2	-	2	2	-	1	1	2	1	-	1
2. Gain hands-on experience in doing experiments on ARM by using simulator in the laboratory and present the report.	2	2	-	2	2	-	1	1	2	1	-	1
3. Understand the hardware, software tradeoffs involved in the design and interface hardware devices to ARM family.	2	2	-	2	2	-	1	1	2	1	-	1
4. Assess processors for various kinds of applications.	3	3	3	3	2	1	1	1	2	2	2	2
Note: Mappings in the Tables D1 (above) and D2 (below) are done by entering in the corresponding cell the Correllation Levels in terms of numbers. For Slight (Low): 1, Moderate (Medium): 2, Substantial (High): 3 and for no correllation: " - ".												

D2. ARTICULATION MATRIX CO v/s PSO

Mapping of CO to PSO			
	PSOs		
COs	1	2	3
1. Apply the programming techniques in designing simple assembly language programs for solving simple problems by using ARM instruction set.	1	2	2
2. Gain hands-on experience in doing experiments on ARM by using simulator in the laboratory and present the report.	1	2	2
3. Understand the hardware, software tradeoffs involved in the design and interface hardware devices to ARM family.	1	2	2
4. Assess processors for various kinds of applications.	2	2	2

1. EXPERIMENT NO: 1

2. TITLE: 16-BIT MULTIPLICATION

3. LEARNING OBJECTIVES:
   - TO TO LEARN THE PROCESS OF WRITING AND SIMULATING ARM PROGRAMS FOR MULTIPLICATION OPERATION ON 16-BIT DATA

4. AIM:
   - MULTIPLY TWO 16 BIT BINARY NUMBERS

5. MATERIAL / EQUIPMENT REQUIRED:
   - PC WITH WINDOWS 7 OS AND MINIMUM 1GB RAM
   - KEIL-5 MDK-ARM WITH LEGACY SUPPORT

6. THEORY /ALGORITHM:

TWO NUMBERS ARE STORED IN DATA MEMORY USING DCW (DEFINE CONSTANT WORD) DIRECTIVE.

* STEP1: GET THE FIRST NUMBER IN R1 REGISTER.
* STEP2: GET THE SECOND NUMBER IN R2 REGISTER.
* STEP3: MULTIPLY THE TWO 16-BIT NUMBERS, AND SAVE THE RESULT IN R0.

7. PROGRAM:

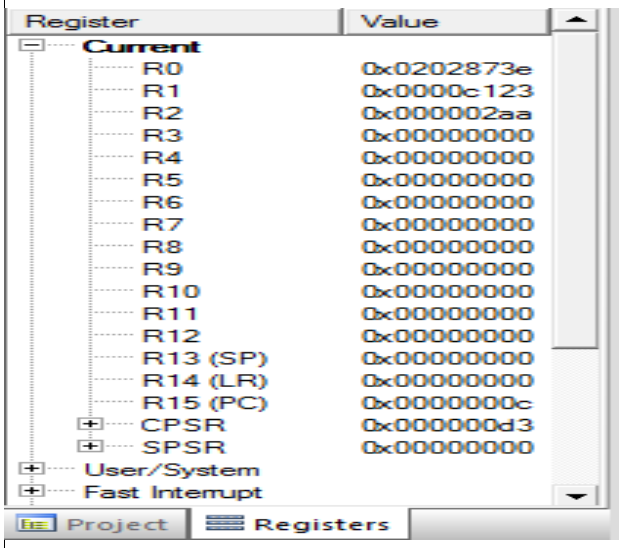
```
        TTL 16-BIT MULTIPLICATION
        AREA PROG1, CODE, READONLY

        ENTRY
START LDRH R1, NUM1     ;
        LOAD THE FIRST NUMBER.
        LDRH R2, NUM2     ; LOAD THE SECOND NUMBER.

        MUL R0, R1, R2      ; MULTIPLY THEM TOGETHER INTO R0 (R0=R1*R2).
STOP B STOP
        AREA DATA1, DATA, READONLY

NUM1  DCW  0XC123         ; FIRST VALUE TO BE MULTIPLIED.

NUM2  DCW  0X02AA         ; SECOND VALUE TO BE MULTIPLIED.

        ALIGN
```

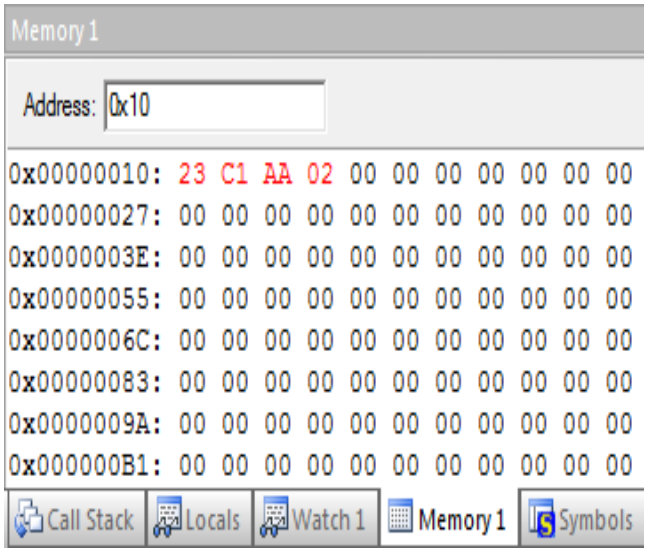
8. PROCEDURE: NA

9. BLOCK DIAGRAM: NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE: NA

11. OUTPUTS:





12. RESULTS & CONCLUSIONS:

•

13. LEARNING OUTCOMES:

- THE ARM PROCESSOR CAN BE PROGRAMMED IN ASSEMBLY LANGUAGE WHICH GIVES GOOD CONTROL OVER THE PROCESSOR

14. APPLICATION AREAS:
-

15. REMARKS:
-

1. EXPERIMENT NO: 2

2. TITLE: SUM OF FIRST 10 INTEGER NUMBERS

3. LEARNING OBJECTIVES:
- TO LEARN THE PROCESS OF WRITING AND SIMULATING ARM PROGRAMS FOR ADDING FIRST 10 INTEGER NUMBER

4. AIM:
- TO FIND THE SUM OF FIRST 10 INTEGER NUMBERS

5. MATERIAL / EQUIPMENT REQUIRED:
- PC WITH WINDOWS 7 OS AND MINIMUM 1GB RAM
- KEIL-5 MDK-ARM WITH LEGACY SUPPORT

6. THEORY/ALGORITHM:

INITIALLY, SUM IS SET TO 0 IN DATA MEMORY USING DCD (DEFINE CONSTANT DATA) DIRECTIVE.

STEP1: LOAD THE INITIAL VALUE OF SUM INTO REGISTER R0. LET R1 BE THE ITERATER & INITIAL SUM VALUE 0 BE IN REGISTER R2.

STEP2: CHECK ITERATER R1 TO THE MAXIMUM VALUE 10; AND IF IT IS NOT 10, ADD EACH NUMBER AND SAVE TO R2.

STEP3: REPEAT STEP2; UNTIL ITERAER R1 BECOMES 10; AND IF IT IS 10, STORE THE FINAL SUM VALUE.

7. PROGRAMS:

```
        TTL INTEGER SUM

        AREA PROG2, CODE, READONLY

        ENTRY
START   LDR R0,=Sum

        MOV R1, #1          ;Loop iterater.

        MOV R2, #0          ;Initial Sum = 0.
Next    CMP R1, #10         ;repeat for summing first 10 numbers.

        BEQ Store

        ADC R2, R2, R1      ;add each number and save to
        R2. ADD R1, R1, #1  ;increment loop iterater.
Loop    B Next              ;repeat for summing next number.

Store   STR R2, [R0]        ;store the final Sum.
        AREA DATA2, DATA
Sum     DCD 0

        END
```

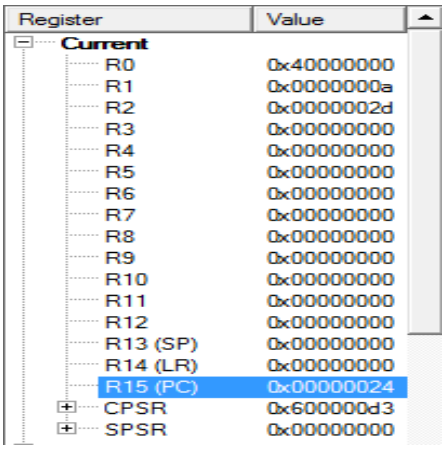
8. PROCEDURE: NA

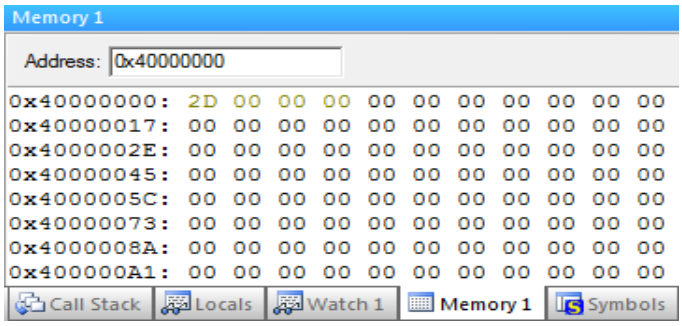
9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 2D





12. RESULTS & CONCLUSIONS:

-

13. LEARNING OUTCOMES:

- The ARM processor can be programmed in assembly language which gives good control over the processor

14. APPLICATION AREAS:

-

15. REMARKS:

-

1. EXPERIMENT NO: 3

2. TITLE: FACTORIAL OF A NUMBER

3. LEARNING OBJECTIVES:

- TO LEARN THE ARM PROGRAM IMPLEMENTATION OF COMPUTATION OF FACTORIAL OF A NUMBER

4. AIM:

- TO FIND FACTORIAL OF A NUMBER

5. MATERIAL / EQUIPMENT REQUIRED:

- PC WITH WINDOWS 7 OS AND MINIMUM 1GB RAM
- KEIL-5 MDK-ARM WITH LEGACY SUPPORT

6. THEORY/ALGORITHM:

NUMBER (NUM) WHOSE FACTORIAL IS TO BE CALCULATED IS STORED IN DATA MEMORY USING DCW (DEFINE CONSTANT WORD) DIRECTIVE.

STEP1: GET THE FIRST NUM IN R0 REGISTER & MAKE A COPY INTO R1 REGISTER.

STEP2: CHECK WHETHER THE NUM IS 2; AND IF 2, THEN MAKE THE FACTORIAL VALUE 1, AND IF NUM IS LESS THAN 1 THEN STOP.

STEP3: IF NUM IS GREATER THAN 2; SUBTRACT ONE FROM THE NUM AND MULTIPLY NUM WITH THE NUM1 VALUE, UNTIL NUM BECOMES EQUAL TO ONE.

STEP4: IF NUM BECOMES EQUAL TO 1, THEN STOP.

7. PROGRAM:

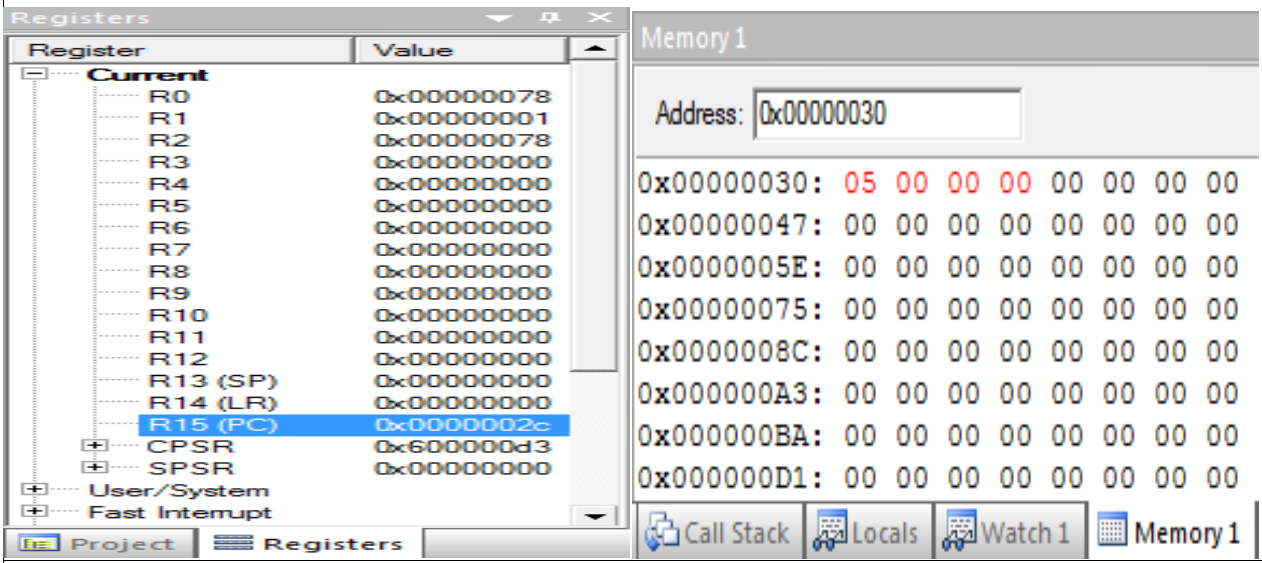
```
        TTL FACTORIAL OF A NUMBER
        AREA PROG3, CODE, READONLY ENTRY
START
        LDR R0,Num      ;load the value into R0.
        MOV R1,R0       ;make a copy.
        CMP R0, #02     ;check whether the value is less than 2.
        MOV R2, #01     ;if less than 2, then make factorial value 1.
        BLT STOP        ;if value is less than 1 then Stop.
REPEAT
        SUBS R1,R1,#01  ;subtract the number by 1.
        CMP R1,#01      ;compare whether the number is 1.
        BEQ STOP        ;if value is equal to 1 then Stop.
        MUL R2,R0,R1    ;multiply to get factorial
        MOV R0, R2      ;store result, R0 has the final value.
        B REPEAT
STOP B STOP
        AREA DATA3, DATA, READONLY
Num     DCW  0x5        ;Number whose factorial is to be calculated.
        ALIGN
        END
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:



12. RESULTS & CONCLUSIONS:

-

13. LEARNING OUTCOMES:
- THUS THE FACTORIAL OF NUMBER CAN BE CALCULATED MORE EFFICIENTLY USING ARM PROGRAM.

14. APPLICATION AREAS:
-

15. REMARKS:
-

1. EXPERIMENT NO: 4

2. TITLE: 16-BIT ADDITION

3. LEARNING OBJECTIVES:
- TO LEARN THE PROCESS OF WRITING AND SIMULATING ARM PROGRAMS FOR ADDING AN ARRAY OF 16BIT NUMBERS

4. AIM: TO ADD AN ARRAY OF 16 BIT NUMBERS AND STORE THE 32 BIT RESULT IN INTERNAL RAM

5. MATERIAL / EQUIPMENT REQUIRED:
- PC WITH WINDOWS 7 OS AND MINIMUM 1GB RAM
- KEIL-5 MDK-ARM WITH LEGACY SUPPORT

6. THEORY/ALGORITHM:
ARRAY OF 10 ELEMENTS IS SUM IS STORED IN DATA MEMORY USING DCW (DEFINE CONSTANT WORD) DIRECTIVE & SUM IS SET TO 0 IN DATA MEMORY USING DCD (DEFINE CONSTANT DATA) DIRECTIVE.

STEP1: LOAD ARRAY ADDRESS TO R0 & SUM ADDRESS TO R5. LET R1 BE THE ITERATOR & INITIAL SUM IN R4 IS 0.

STEP2: LOAD THE FIRST ARRAY NUMBER TO R3, ADD WITH CARRY (IF ANY), INCREMENT THE LOOP ITERATOR.

STEP3: REPEAT STEP2, UNTIL THE ITERATOR BECOMES 10.

STEP4: IF ALL NUMBERS ADDED, SAVE THE RESULT TO SUM LOCATION FROM R4

7. PROGRAM:

```
        TTL 16-BIT ADDITION
        AREA PROG4, CODE, READONLY
        ENTRY
START
        LDR R0, =Array      ;load array address and sum. LDR R5, =Sum
        MOV R1, #0  ;Loop Iterator.
        MOV R4, #0  ;Initial Sum =0.
Next    CMP R1, #10  ;load total numbers in array, say 10.
        BEQ Store
        LDRH R3, [R0], #2   ;load the number from array and point to next word (16-bit number).
        ADC R4,R4,R3        ;add & update carry if any.
        ADD R1, R1, #1      ;increment loop iterator, stop if reached 10
Loop    B Next
```

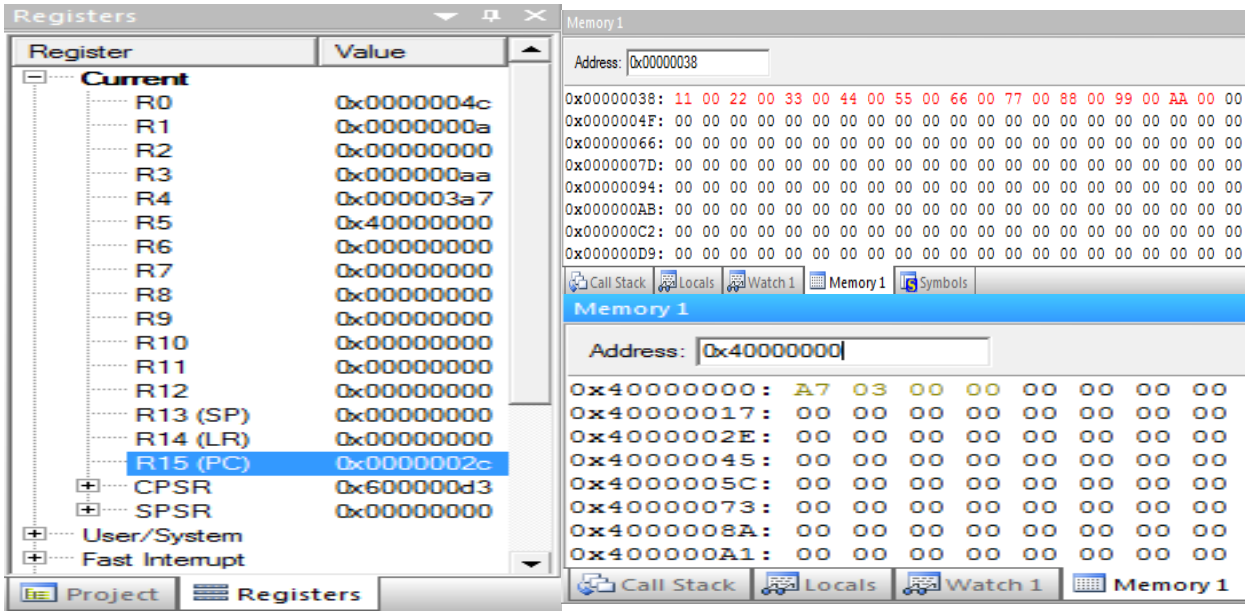
```
Store   STR R4, [R5]
STOP B STOP

        AREA Data4, DATA, READONLY
Array   DCW 0x0011,0x0022,0x0033,0x0044,0x0055,0x0066,0x0077,0x0088,0x0099 ,0x00AA
        ALIGN
        AREA Data41, DATA
Sum     DCD 0 END
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:



12. RESULTS & CONCLUSIONS:

-

13. LEARNING OUTCOMES:

- THE ARM PROCESSOR CAN BE PROGRAMMED GIVES GOOD CONTROL OVER THE PROCESSOR

14. APPLICATION AREAS:

-

15. REMARKS:

-

1. EXPERIMENT NO: 5

2. TITLE: SQUARE OF A NUMBER

3. LEARNING OBJECTIVES:

- TO LEARN THE PROCESS OF WRITING AND SIMULATING ARM PROGRAMS FOR SQUARING OF A NUMBER USING LOK-UP-TABLE

4. AIM: TO FIND THE SQUARE OF A NUMBER (1 TO 10) USING LOOK-UP TABLE.

5. MATERIAL / EQUIPMENT REQUIRED:

- PC WITH WINDOWS 7 OS AND MINIMUM 1GB RAM

- KEIL-5 MDK-ARM WITH LEGACY SUPPORT

6. THEORY/ALGORITHM:

ARRAY OF 10 ELEMENTS (LOOKUP TABLE) & THE NUMBER/ VALUE WHOSE SQUARE IS TO BE RETRIEVED FROM THE LOOKUP TABLE ARE STORED IN DATA MEMORY USING DCD (DEFINE CONSTANT DATA) DIRECTIVES.

RESULT IS SET TO 0 IN DATA MEMORY USING DCD (DEFINE CONSTANT DATA) DIRECTIVE.

STEP1: LOAD THE ADDRESS OF LOOKUP TABLE TO R0, VALUE TO R1, AND ADDRESS OF RESULT TO R2.

STEP2: BASED ON THE VALUE (R1), ADJUST THE POINT LOCATION WHERE SQUARE IS STORED. ADJUST IS REQUIRED FOR 32-BIT (4 BYTES) DATA.

STEP3: GET THE RESULT FROM LOOKUP TABLE MEMORY LOCATION TO REGISTER R3

7. PROGRAM:

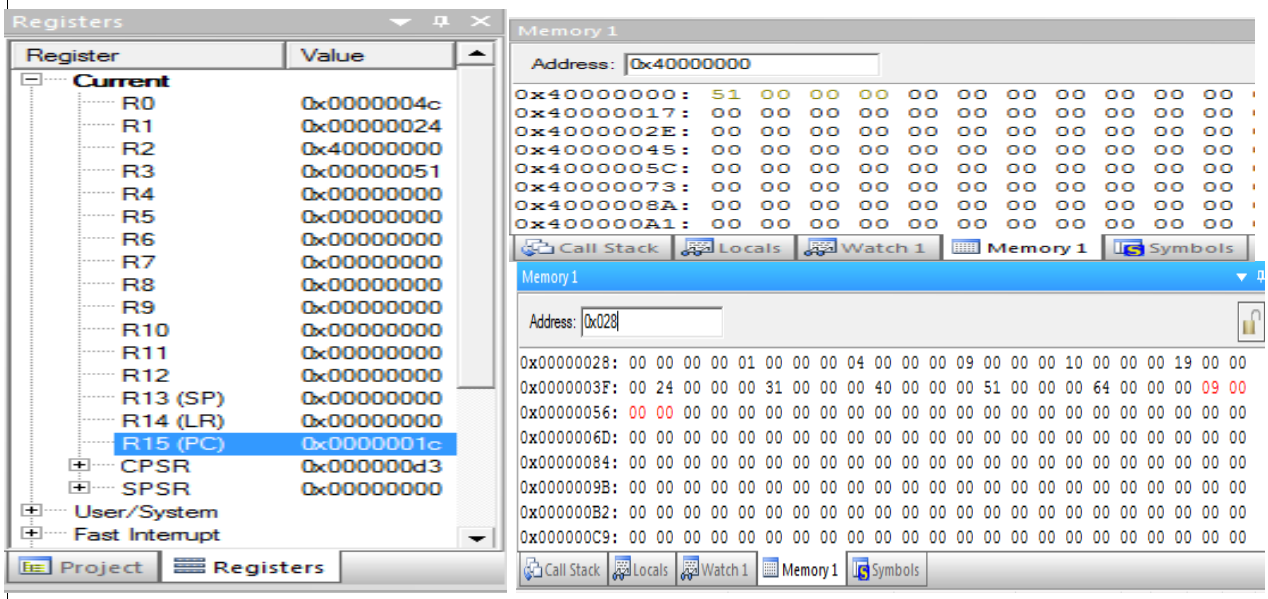
```
        TTL SQUARE OF A MUNBER
        AREA PROG5, CODE, READONLY
        ENTRY
START
        LDR R0, =Lookup          ;to point base of lookup table.
        LDR R1, Value            ;value whose square is to be retrieved from lookup table.
        LDR R2, =Result          ;point to result.
        MOV R1, R1, LSL #0x2 ;adjust to point location where square is stored in look -up table.
        ADD R0, R0, R1           ;adjust is required for 32-bit data (4 bytes).
        LDR R3, [R0]             ;get result from memory location (lookup table) to R3.
        STR R3, [R2]             ;store result.
STOP B STOP

        AREA SRC, DATA, READONLY
Lookup DCD 0,1,4,9,16,25,36,49,64,81,100 ;Lookup table contains Squares of Nos from 0 to 10.
Value   DCD 0x00000009     ;Number whose square is to be retrieved from lookup table.
        ALIGN
        AREA DST, DATA
Result  DCD 0x00000000     ;memory location to save the result.
        ALIGN
        END
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:



12. RESULTS & CONCLUSIONS:

-

13. LEARNING OUTCOMES:

- THE ARM PROCESSOR CAN BE PROGRAMMED AND GIVES GOOD CONTROL OVER THE PROCESSOR

14. APPLICATION AREAS:

-

15. REMARKS:

-

1. EXPERIMENT NO: 6

2. TITLE: LARGEST/ SMALLEST NUMBER

3. LEARNING OBJECTIVES:

- TO LEARN THE PROCESS OF WRITING AND SIMULATING ARM PROGRAMS FOR FINDING LARGEST/ SMALLEST NUMBER

4. AIM:

- TO FIND THE LARGEST/ SMALLEST NUMBER IN AN ARRAY OF 32 (- BIT?) NUMBERS

5. MATERIAL / EQUIPMENT REQUIRED:

- PC WITH WINDOWS 7 OS AND MINIMUM 1GB RAM
- KEIL-5 MDK-ARM WITH LEGACY SUPPORT

6. THEORY/ALGORITHM:

ARRAY OF ELEMENTS IS STORED IN DATA MEMORY USING DCD (DEFINE CONSTANT DATA) DIRECTIVES. LOCATION (LS) IS RESERVED IN DATA MEMORY USING DCD (DEFINE CONSTANT DATA) DIRECTIVE TO STORE THE LARGEST/ SMALLEST NUMBER.

STEP1: LOAD THE FIRST ELEMENT OF THE ARRAY INTO R3 AND INCREMENT THE POINTER, LOAD THE SECOND ELEMENT OF THE ARRAY INTO R4 AND INCREMENT THE POINTER; THEN COMPARE.

STEP2: IF SECOND ELEMENT IS GREATER, COPY THAT INTO FIRST REGISTER, R3.

STEP3: DECREMENT THE COUNTER, REPEAT STEP 1 AND 2 UNTIL COMPLETION OF ARRAY.

7. PROGRAM:

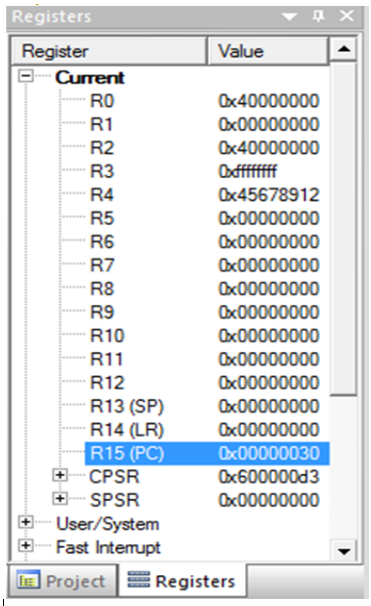
```
        TTL LARGEST/ SMALLEST NUMBER
        AREA PROG6, CODE, READONLY

        ENTRY
START
        LDR R0, =Array      ;to point base of Array.
        MOV R1, #9          ;length of the Array.
        LDR R2, =LS         ;point to result location.
        LDR R3, [R0], #04   ;load first number and increment address by 4.
Up      LDR R4, [R0], #04   ;load second number and increment address by 4.
        CMP R3, R4          ;compare the two numbers.
        MOVLS R3, R4        ;Largest-LS:unsigned lower or same (Z or c); Smallest-HI:unsigned
                             higher (zC).
        SUB R1, R1, #1      ;decrement the counter.
        CMP R1, #00         :check for end of the array
        BNE Up
        LDR R0, =LS ;get the address of memory location to store largest.
        STR R3, [R0] ;save the largest to memory location.
STOP B STOP

        AREA SRC, DATA, READONLY
       ARRAY DCD 0x33333333, 0x22222222, 0x55555555, 0x11111111, 0x44444444,
                 0x77777777, 0xFFFFFFFF, 0x99999999, 0x88888888, 0x45678912
    ALIGN
    AREA DST, DATA
LS DCD 0
    ALIGN
    END
```

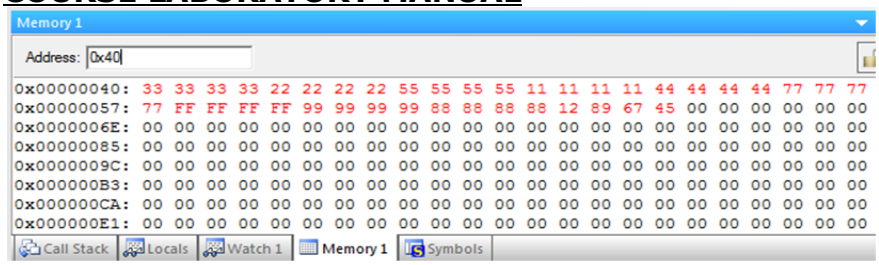
8. PROCEDURE:NA

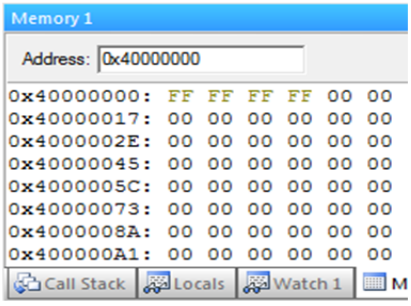
9. BLOCKDIAGRAM:NA

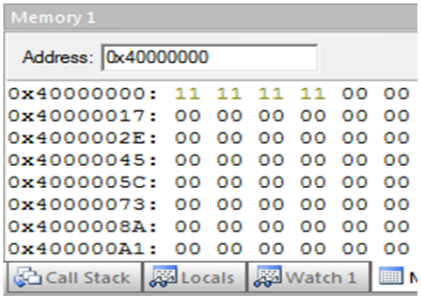
10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:









12. RESULTS & CONCLUSIONS:
-

13. LEARNING OUTCOMES:
- THE ARM PROCESSOR CAN BE PROGRAMMED AND GIVES GOOD CONTROL OVER THE PROCESSOR

14. APPLICATION AREAS:
-

15. REMARKS:
-

1. EXPERIMENT NO:7

2. TITLE: NUMBERS IN ASCENDING/DESCENDING ORDER

3. LEARNING OBJECTIVES:
- TO LEARN THE PROCESS OF WRITING AND SIMULATING ARM PROGRAMS FOR ARRANGING GIVEN NUMBER IN ASCENDING/ DESCENDING ORDER

4. AIM:
- TO ARRANGE A SERIES OF 32-BIT NUMBERS IN ASCENDING/ DESCENDING ORDER

5. MATERIAL / EQUIPMENT REQUIRED:
- PC WITH WINDOWS 7 OS AND MINIMUM 1GB RAM
- KEIL-5 MDK-ARM WITH LEGACY SUPPORT

6. THEORY/ALGORITHM:

STEP1: COPY THE ELEMENTS OF ARRAY ELEMENTS FROM READONLY DATA MEMORY TO THE MEMORY SECTION.

STEP2: INITIALIZE THE COUNTER, FLAG, AND THE ADDRESS OF THE MEMORY.

STEP3: LOAD THE FIRST AND SECOND NUMBERS INTO REGISTER, COMPARE, AND (IF REQUIRED) SWAP THE VALUES IN MEMORY; SET THE FLAG.

STEP4: DECREMENT THE COUNTER AND REPEAT STEP 3, IF COUNTER IS NOT ZERO.

STEP5: NOW, CHECK FLAG FOR IF NOT, REPEAT STEP 2,3 AND 4.

7. PROGRAM:

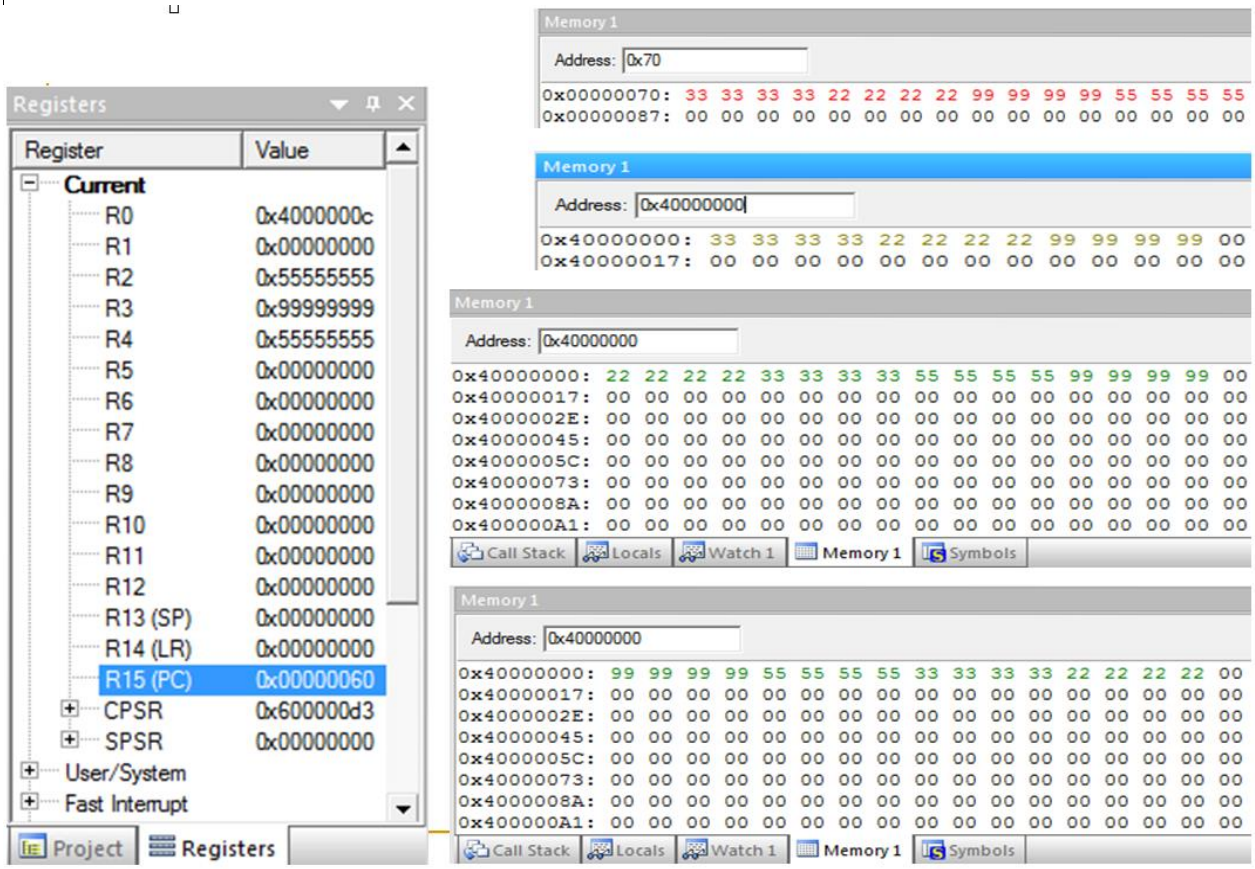
```
        TTL ASCENDING/ DESCENDING
        AREA PROG7, CODE, READONLY
        ENTRY
START
        MOV R1,#4          ;initialize counter to 4(i.e. number of elements are 4).
        LDR R2, =Array     ;load the address of the first element of the Array.
        LDR R3, =DST       ;load the address of the first element position of destination.
UP      LDR R4, [R2], #4   ;load the first element of Array into R4.
        STR R4, [R3], #4   ;store the first element to the destination region.
        SUBS R1, R1, #1    ;decrement the counter. CMP R1, #0  ;compare the counter to 0.
        BNE UP             ;loop back till Array ends.
Initial
        MOV R1, #3         ;length of the Array = N-1.
        MOV R7, #0         ;flag to denote exchange has occurred.
        LDR R0, =DST       ;load the address of the first element of the memory.
Inner   LDR R2, [R0], #4   ;load the first number.
        LDR R3, [R0]       ;load the second number.
        CMP R2, R3         ;compare two numbers.
        BLS Outer          ;if the first number is < then go to Loop2. ;(Largest-LS:unsigned
                           lower or same (Z or c); Smallest-HI:unsigned higher (zC))
        STR R2, [R0],#-4   ;Interchange the position of first and second numbers.
        STR R3, [R0]
        MOV R7, #1         ;flag denoting exchange has taken place.
        ADD R0, #4         ;restore the pointer.
Outer
        SUBS R1, R1, #1    ;decrement the counter.
        CMP R1, #0         ;compare counter to 0.
        BNE Inner          ;Loop back till Array ends.
        CMP R7, #0         ;check the flag.
        BNE Initial        ;if flag is not zero, then go to Initial.
STOP B STOP
        AREA SRC, DATA, READONLY
Array   DCD 0x33333333, 0x22222222, 0x99999999, 0x55555555 ALIGN
AREA DST, DATA SPACE 10
        ALIGN
        END
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:



12. RESULTS & CONCLUSIONS:

-

13. LEARNING OUTCOMES:

- THE ARM PROCESSOR CAN BE PROGRAMMED AND GIVES GOOD CONTROL OVER THE PROCESSOR

14. APPLICATION AREAS:

-

15. REMARKS:

-

1. EXPERIMENT NO: 8

2. TITLE: NUMBER OF ONES & ZEROS

3. LEARNING OBJECTIVES:

- TO LEARN THE PROCESS OF WRITING AND SIMULATING ARM PROGRAMS FOR COUNTING NUMBER OF ONES AND ZEROS.

4. AIM:

- TO COUNT THE NUMBER OF ONES AND ZEROS IN TWO CONSECUTIVE MEMORY LOCATIONS.

5. MATERIAL / EQUIPMENT REQUIRED:

- PC WITH WINDOWS 7 OS AND MINIMUM 1GB RAM
- KEIL-5 MDK-ARM WITH LEGACY SUPPORT

6. THEORY/ALGORITHM:

The two 8-bit numbers whose number of ones and zeros is to be calculated are stored in DATA memory using DCB (Define Constant Byte) directives.

Step1: Load the address of the first number, clear the contents of two registers to hold the result, initialize counter for two consecutive memory locations.

Step2: Initialize the counter for 8-bits of a number, load the first number to R3, test for 1's, and increment the register (r6 or R5) accordingly. Check all the 8-bits of a number.

Step3: Access the second number and repeat Step 2.
Step4: Save the values of ones and zeros to memory location.

7. PROGRAM:

```
        TTL NUMBER OF ONES & ZEROS
        AREA PROG8, CODE, READONLY

        ENTRY
START
        LDR R0, =SRC        ;load the first num address.
        EOR R5,R5,R5        ;R5 and R6 are used to save result.
        EOR R6,R6,R6        ;Initially, R5 = R6 = 0.
        MOV R2, #2          ;two 8-bit numbers (consecutive memory locations) to check.
Up      MOV R1, #8          ;each number is of 8-bit size.
        LDRB R3, [R0]       ;load the first 8-bit number to R3.
Top     TST R3, #01         ;test for 1's.
        BEQ Inc_Zero
        ADD R6, #1          ;increment ones count.
        B Down
Inc_Zero  ADD R5, #1        ;increment zeros count.
Down    LSR R3, #1          ;logical shift right.
        SUB R1, #1          ;decrement the bit counter.
        CMP R1, #0          ;check all the bits. BNE Top
        ADD R0, #1          ;access the second 8-bit number.
        SUB R2, #1          ;last memory location to check.
        CMP R2, #0          ;repeat checking ones & zeros for the second location.
        BNE Up
        LDR R0, =ONES       ;address of the memory location to save number of ones.
        LDR R0, =ONES       ;address of the memory location to save number of ones.
        STRB R6, [R0]       ;R6 holds the number of ones.
        LDR R0, =ZEROS      ;address of the memory location to save number of zeros.
        STRB R5, [R0]       ;R6 holds the number of zeros.
STOP B STOP
        AREA SRC,DATA,READONLY
        DCB 0x01, 0x0F      ;two 8-bit numbers in consecutive memory locations.

        AREA DST, DATA, READWRITE
```

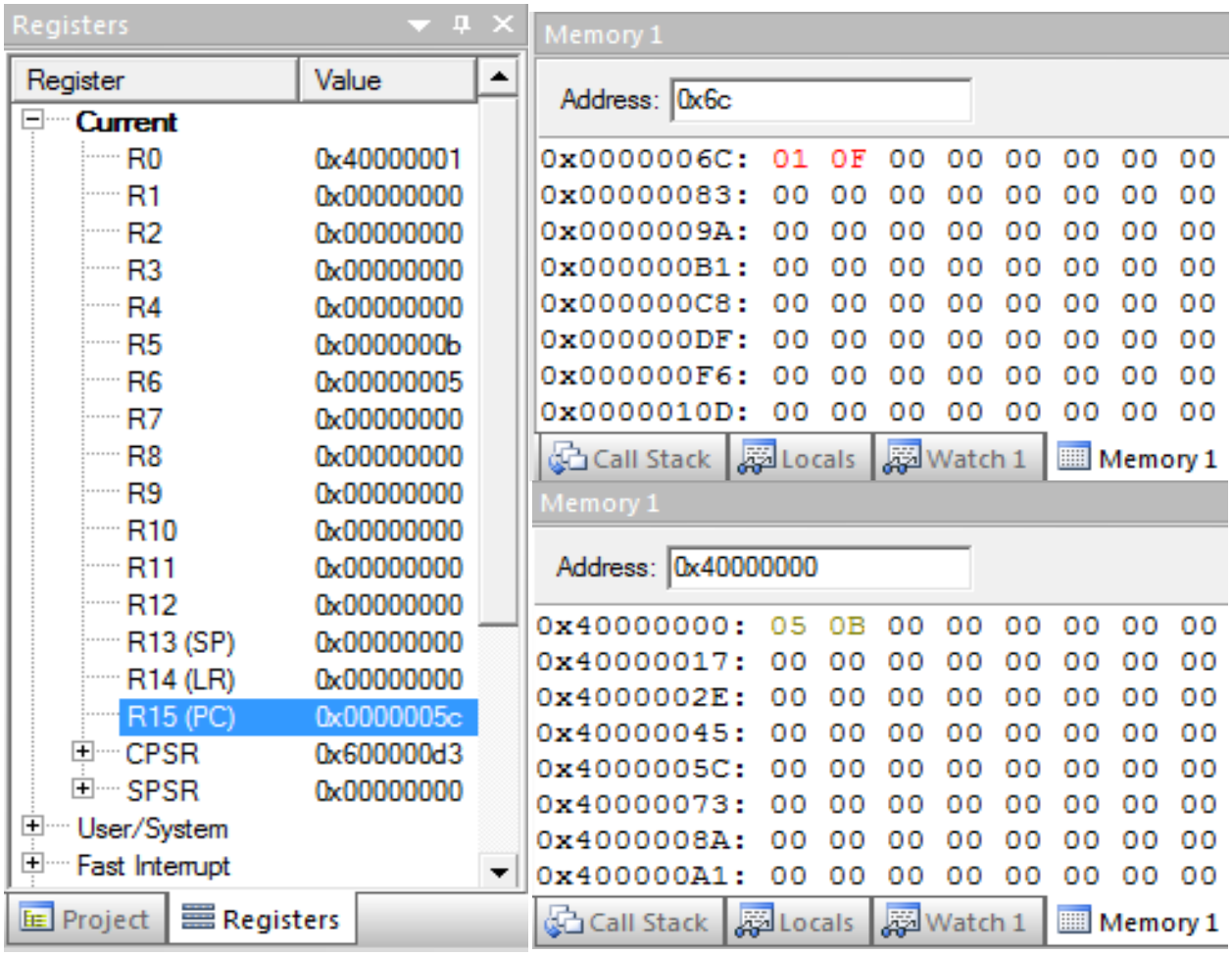
```
ONES   DCB 0          ;to hold number of ones.
ZEROS DCB 0           ;to hold number of zeros.
       END
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:



12. RESULTS & CONCLUSIONS:

-

13. LEARNING OUTCOMES:

- THE ARM PROCESSOR CAN BE PROGRAMMED AND GIVES GOOD CONTROL OVER THE PROCESSOR

14. APPLICATION AREAS:

-

15. REMARKS:

-

1. EXPERIMENT NO: 9

2. TITLE: UART-HELLO WORLD

COURSE LABORATORY MANUAL

3. LEARNING OBJECTIVES:

- TO LEARN THE INTERFACING TO 'HELLO WORLD' MESSAGE USING INTERNAL UART.

4. AIM:

- DISPLAY "HELLO WORLD" MESSAGE USING INTERNAL UART.

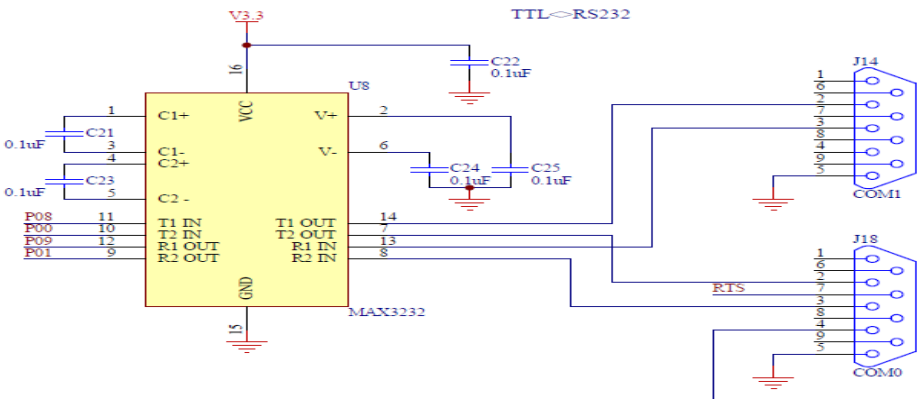
5. MATERIAL / EQUIPMENT REQUIRED:

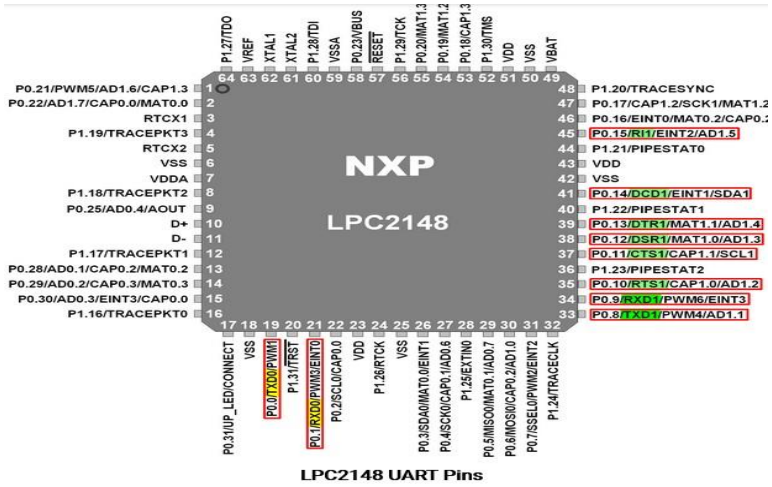
- INTEL CPU BASED COMPUTER
- UART INTERFACE KIT
- FRC CONNECTOR.

6. THEORY/ALGORITHM:

The microcontroller has to communicate to external world through the UARTs only. LPC2148 microcontroller has two UARTs – UART0 & UART1.

* Importance for developing serial port driver
* Built-in baud rate generator.
* 16 byte Receive and Transmit FIFOs.
* Receiver FIFO trigger points at 1, 4, 8, and 14 bytes.
* Register locations conform to '550 industry standard.
* Mechanism that enables software and hardware flow control implementation.





LPC2148 UART Pins

The Circuit Diagram for Interfacing COM Port with LPC2148 is as follow:

- First of all you have to assign the PINs Rx and Tx for UART0, this is done by the help of PIN SELECT BLOCK 0, by setting its value to 0x05; the pin P0.0 will behave as Tx0 and P0.1 will behave a Rx0.
- After that you need to configure the UART0 bits like Data Bit, Stop Bits and Parity Bits, this can be set by the U0LCR (Line Control Register).
- We will write value 0x83 to U0LCR, as we are using 8-bit character length, 1 stop bit and will disable the parity bits.

U0LCR (UART0 Line Control Register)

- It is an 8-bit read-write register.
- It determines the format of the data character that is to be transmitted or received.

7	6	5	4	3	2	1 0
DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	No. of Stop Bits	Word Length Select

U0LCR (UART0 Line Control Register)

- **Bit 1:0 - Word Length Select**
  00 = 5-bit character length
  01 = 6-bit character length
  10 = 7-bit character length
  11 = 8-bit character length
- **Bit 2 - Number of Stop Bits**
  0 = 1 stop bit
  1 = 2 stop bits
- **Bit 3 - Parity Enable**
  0 = Disable parity generation and checking
  1 = Enable parity generation and checking
- **Bit 5:4 - Parity Select**
  00 = Odd Parity
  01 = Even Parity
  10 = Forced "1" Stick Parity
  11 = Forced "0" Stick Parity
- **Bit 6 - Break Control**
  0= Disable break transmission
  1 = Enable break transmission
- **Bit 7 - Divisor Latch Access Bit (DLAB)**
  0 = Disable access to Divisor Latches
  1 = Enable access to Divisor Latches

The UART0 is operating at a Baud rate of 9600bps, the baud rate is determined by the Crystal Frequency, U0DLM (Divisor Latch MSB), U0DLL (Divisor Latch LSB), DivAddVal, MulVal Values. The Formula for Calculation is as follow:

$$UART0_{baudrate} = \frac{PCLK}{16 \times (256 \times U0DLM + U0DLL) \times \left(1 + \frac{DivAddVal}{MulVal}\right)}$$

where, baud rate = 9600, PCLK = 12 MHz, U0DLM = 0, DivAddVal = 0, and MulVal = 1.

- These values are to be loaded in the U0DLL and U0DLM Register in order to obtain the Baud Rate of 9600 bps.

- After loading this step we are going to disable the DLAB bit, as this will protect the further changes to these values.

U0DLL and U0DLM (UART0 Divisor Latch Registers)

- U0DLL is the Divisor Latch LSB.
- U0DLM is the Divisor Latch MSB.
- These are 8-bit read-write registers.
- UART0 Divisor Latch holds the value by which the PCLK(Peripheral Clock) will be divided. This value must be 1/16 times the desired baud rate.
- A 0x0000 value is treated like a 0x0001 value as division by zero is not allowed.
- The Divisor Latch Access Bit (DLAB) in U0LCR must be one in order to access the UART0 Divisor Latches. (DLAB = 1)

U0DLL

U0DLM

To send data over UART:

All Data which is to sent over UART is entered in U0THR (Transmit Holding Register); when there is data in U0THR Register it will start sending data and if data is sent then the bit THRE in U0LSR Register gets set other zero, hence we can write a small piece of code for sending data as follow:

while (! (U0LSR & 0x20));

U0THR = value; where value is data to be sent over UART.

* To receive data over UART:

* All Data received from UART is received U0RBR (Received Buffer Register); upon reception of data from UART RDR (Receiver Data Ready bit) in U0LSR sets otherwise cleared, and by this we can collect the data, hence we can write a small piece of code for receiving data as follow:

```
while(! (U0LSR & 0x01));

return (U0RBR);
```

U0LSR (UART0 Line Status Register)

- It is an 8-bit read only register.

U0LSR (UART0 Line Status Register)

- It provides status information on UART0 RX and TX blocks.
- **Bit 0 - Receiver Data Ready**
  0 = U0RBR is empty
  1 = U0RBR contains valid data
- **Bit 1 - Overrun Error**
  0 = Overrun error status inactive
  1 = Overrun error status active
  This bit is cleared when U0LSR is read.
- **Bit 2 - Parity Error**
  0 = Parity error status inactive
  1 = Parity error status active
  This bit is cleared when U0LSR is read.
- **Bit 3 - Framing Error**
  0 = Framing error status inactive
  1 = Framing error status active
  This bit is cleared when U0LSR is read.
- **Bit 4 - Break Interrupt**
  0 = Break interrupt status inactive
  1 = Break interrupt status active
  This bit is cleared when U0LSR is read.
- **Bit 5 - Transmitter Holding Register Empty**
  0 = U0THR has valid data
  1 = U0THR empty
- **Bit 6 - Transmitter Empty**
  0 = U0THR and/or U0TSR contains valid data
  1 = U0THR and U0TSR empty
- **Bit 7 - Error in RX FIFO (RXFE)**
  0 = U0RBR contains no UART0 RX errors
  1 = U0RBR contains at least one UART0 RX error
  This bit is cleared when U0LSR is read.

U0THR (UART0 Transmit Holding Register)

- It is an 8-bit write only register.
- Data to be transmitted is written to this register.
- It contains the "newest" received byte in the transmit FIFO.
- The Divisor Latch Access Bit (DLAB) in U0LCR must be zero in order to access theU0THR. (DLAB = 0)

U0THR (UART0 Transmit Holding Register)

U0RBR (UART0 Receive Buffer Register)

- It is an 8-bit read only register.
- This register contains the received data.
- It contains the "oldest" received byte in the receive FIFO.
- If the character received is less than 8 bits, the unused MSBs are padded with zeroes.
- The Divisor Latch Access Bit (DLAB) in U0LCR must be zero in order to access the U0RBR. (DLAB = 0)

U0RBR (UART0 Receive Buffer Register)

7. PROGRAM:

```
#include <LPC214x.H>
#include <String.H>

void init_serial (void) //The init_serial function initializes the serial driver.
{
PINSEL0 = PINSEL0 | 0x00000005; //To assign the PINs Rx (P0.1) and Tx (P0.0) for UART0.
U0LCR = 0x83;
/*U0LCR: UART0 Line Control Register 0x83: Enable access to Divisor Latches and set 8-bit word length with 1 stop bit and Parity bit Disabled; access to Divisor Latches is Enabled, in order to write Baud rate Generator Registors.*/
U0DLL = 0x5E;          //U0DLL: UART0 Divisor Latch (LSB).
U0DLM = 0x00;          //U0DLM: UART0 Divisor Latch (MSB).
/*Formula is Baud Rate = PCLK * MulVal / [(16*(256*U0DLM+U0DLL)*(MulVal + DivAddVal))]
Example: MulVal = 1, DivAddVal = 0, Baud_Rate = 9600, PCLK = 14.44 MHz, U0DLM =0
Hence, U0DLL = 14440000/(9600*16) = 94 = 0x5E*/
U0LCR = 0x03;
/*U0LCR: UART0 Line Control Register 0x03: same as above, but Disable access to Divisor Latches.*/
}
char sendchar (char SDat)
{
while (!(U0LSR & 0x20));     //wait until Transmit Holding Register is empty.
return (U0THR = SDat);         //then store to Transmit Holding Register.
/*THRE bit can be extracted by this U0LSR & 0x20
THRE = 0 means data is present, and THRE = 1 means register is empty.
In order to transmit data, we have to wait till the THRE = 1, then only we can transmit data.*/
}
int getchar (void)
{
while (!(U0LSR & 0x01));     //wait until there's a character to be read (RDR = 0 stay here).
return (U0RBR);         //then read from the Receiver Buffer Register.
```

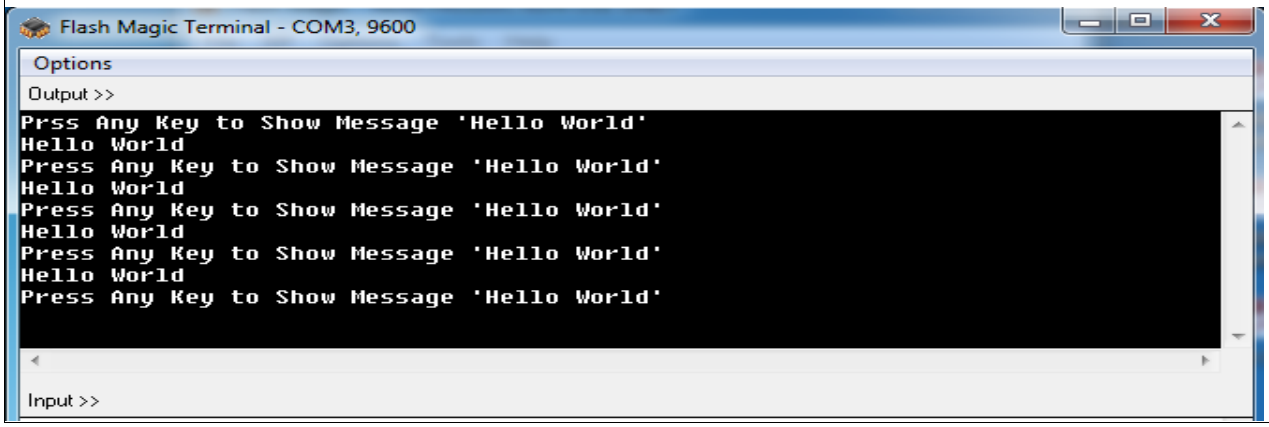
```
/*Receiver Data Ready = U0LSR.0 bit. RDR bit can be extracted by this U0LSR & 0x01. RDR = 0 means no Data is Received in U0RBR, and RDR = 1 means that Data is present in U0RBR.*/
}
void sendstr (char *MSG)
{
unsigned char COUNT, LENGTH; LENGTH = strlen (MSG);
sendchar (0x0D); sendchar (0x0A);
for(COUNT=0;COUNT<LENGTH;COUNT++)
{

sendchar(*MSG); MSG++;
}
}
int main(void)
{
init_serial();

/*The init_serial function initializes the serial driver.
The function: Sets the baud rate, Initializes the UART, Enables the interrupts for transmission and reception of data.*/
while(1)
{
sendstr ("Press Any Key to Show Message 'Hello World'"); getchar();
sendstr ("Hello World");
}
}
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUT:



12. RESULTS & CONCLUSIONS:

13. LEARNING OUTCOMES:

- ANY ALPHANUMERIC CHARACTER CAN BE DISPLAYED.

14. APPLICATION AREAS:

15. REMARKS:

1. EXPERIMENT NO: 10

2. TITLE: DC MOTOR

3. LEARNING OBJECTIVES:

- TO LEARN THE INTERFACING OF DC MOTOR TO THE MICROPROCESSOR AND TO ROTATE IT IN A SPECIFIED.

4. AIM:

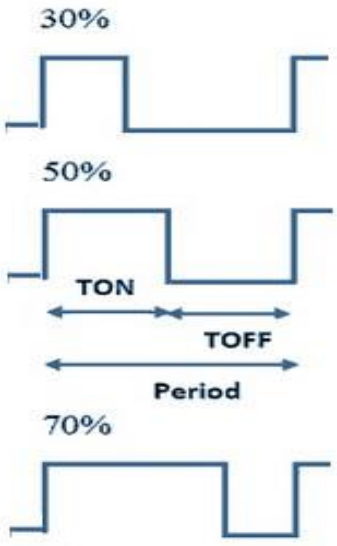
- INTERFACE AND CONTROL A DC MOTOR

5. MATERIAL / EQUIPMENT REQUIRED:

- INTEL CPU BASED COMPUTER
- DC MOTOR DRIVER INTERFACE KIT
- FRC CONNECTOR.

6. THEORY/ALGORITHM:

- DC Motor converts electrical energy (in the form of Direct Current) into mechanical energy.
- DC Motor speed can be controlled by applying varying DC Voltage.
- For applying varying DC Voltage, we can make use of PWM technique.
- The direction of rotation of the motor can be changed by reversing the direction of current through it.
- For reversing the current, we can make use of H-Bridge Circuit or Motor Driver ICs that employ the H-Bridge technique or any other mechanisms.

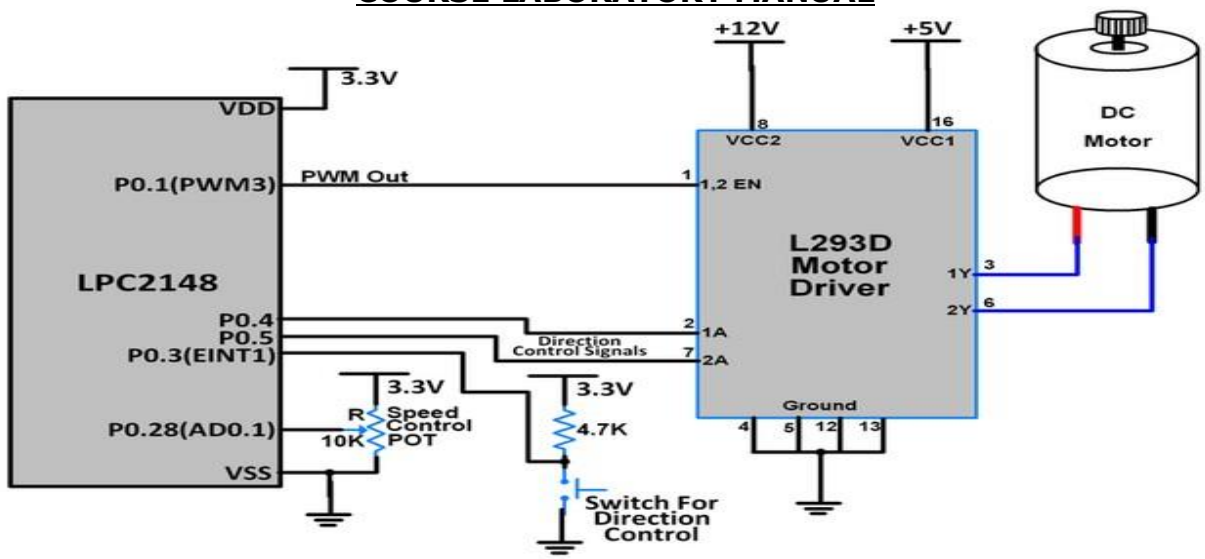


$$\text{Duty Cycle (In \%)} = \frac{Ton}{Ton + Toff} \times 100$$

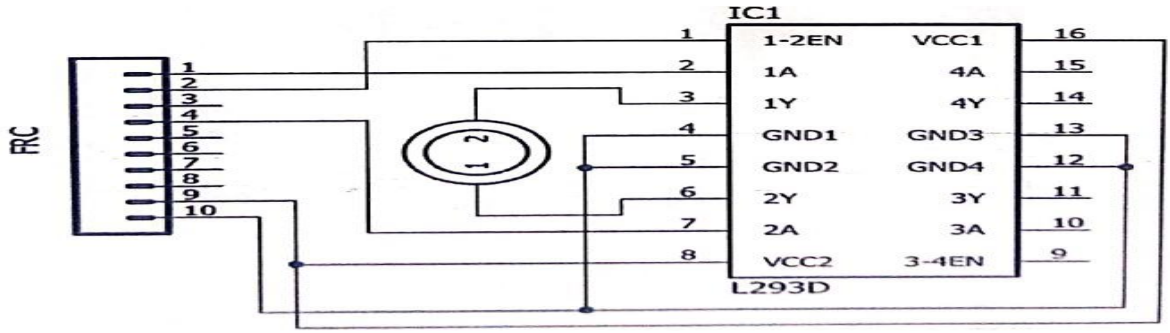
Consider a pulse with a period of 10 ms, which remains ON (high) for 3 ms.

The duty cycle of this pulse will be

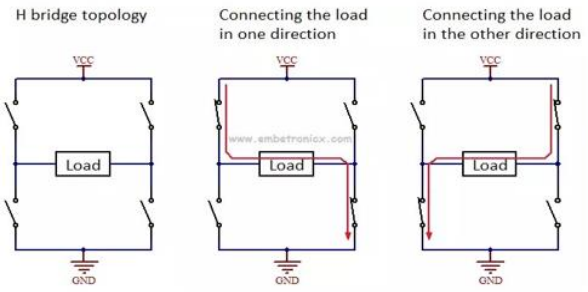
D = (3 ms / 10 ms) x 100 = 30 %

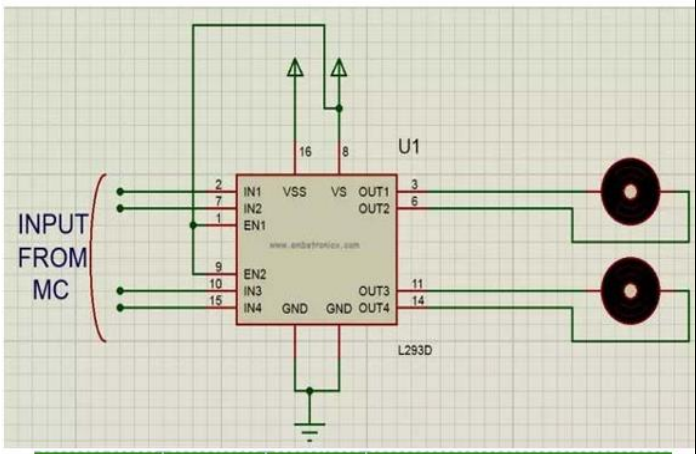


Interfacing DC Motor with LPC2148



L293D Motor Driver





EN1	IN1	IN2	FUNCTION
1	1	0	Rotates Anti-clockwise (Reverse)
1	0	1	Rotates Clockwise (Forward)
1	1	1	OFF
1	0	0	OFF

*Working Algorithm

*Forward

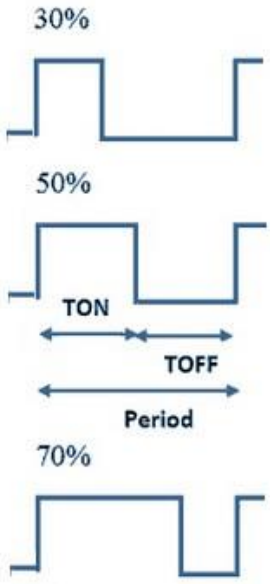
* EN Pin High (En1 = 1 or En2 = 1)
* Input 1 or Input 3 Pin High (In1 = 1 or In3=1)
* Input 2 or Input 4 Pin Low (In2 = 0 or In4 = 0)

*Reverse

* EN Pin High (En1 = 1 or En2 = 1)
* Input 1 or Input 3 Pin Low (In1 = 0 or In3=0)
* Input 2 or Input 4 Pin Low (In2 = 1 or In4 = 1)

Pulse Width Modulation (PWM) is a technique by which width of a pulse is varied while keeping the frequency constant.

* A period of a pulse consists of an ON cycle (HIGH) and an OFF cycle (LOW).



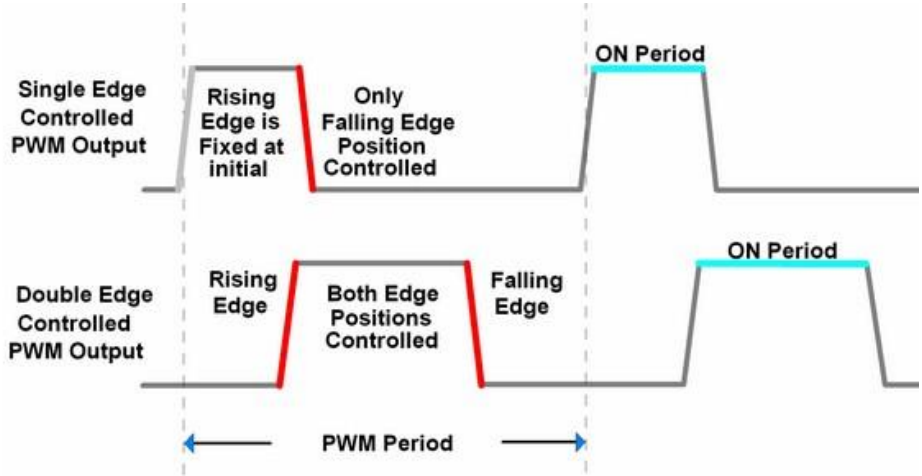
$$\text{Duty Cycle (In \%)} = \frac{Ton}{Ton + Toff} \times 100$$

Consider a pulse with a period of 10 ms, which remains ON (high) for 3 ms.

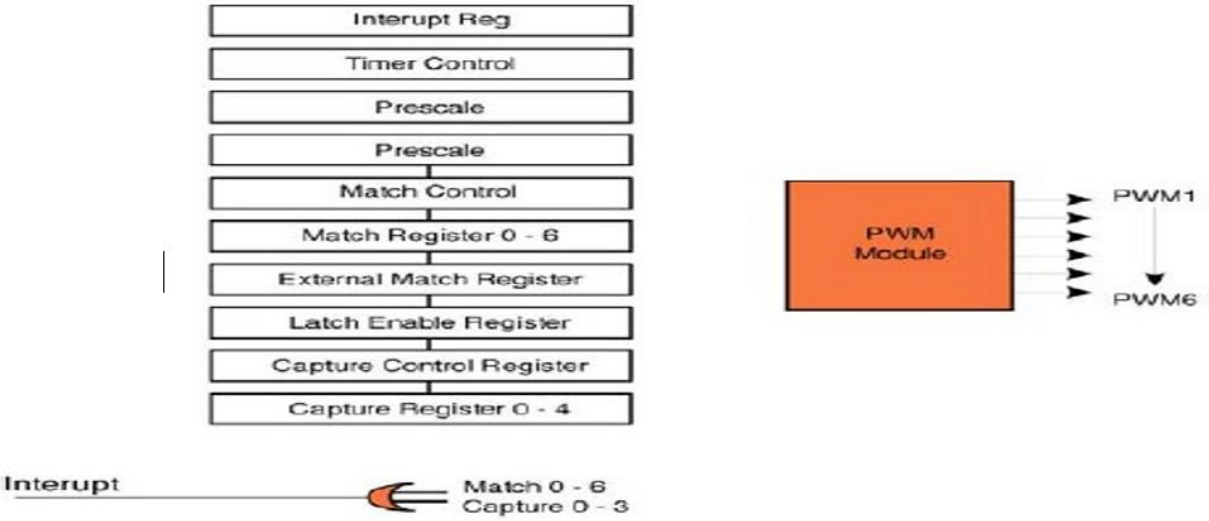
The duty cycle of this pulse will be

D = (3 ms / 10 ms) x 100 = 30 %

- The fraction for which the signal is ON over a period is known as duty cycle. LPC2148 has PWM peripheral through which we can generate multiple PWM signals on PWM pins.
- LPC2148 supports two types of controlled PWM outputs as,
- Single Edge Controlled PWM Output
- Only falling edge position can be controlled.
- Double Edge Controlled PWM Output Both Rising and Falling edge positions can be controlled



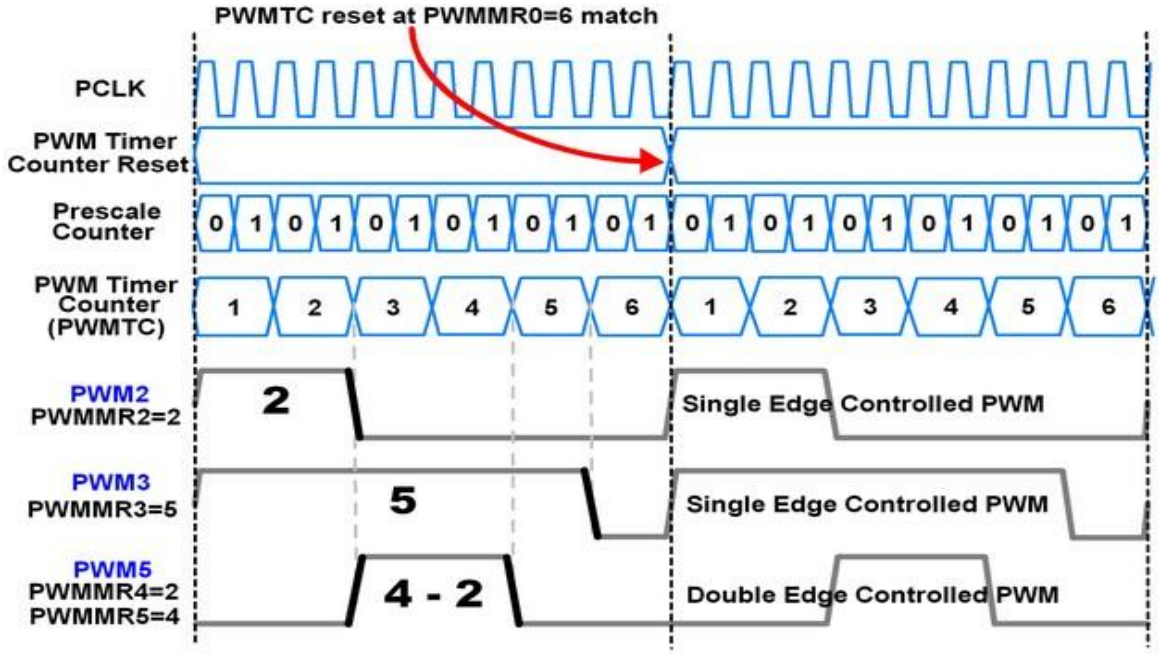
PWM Modulator

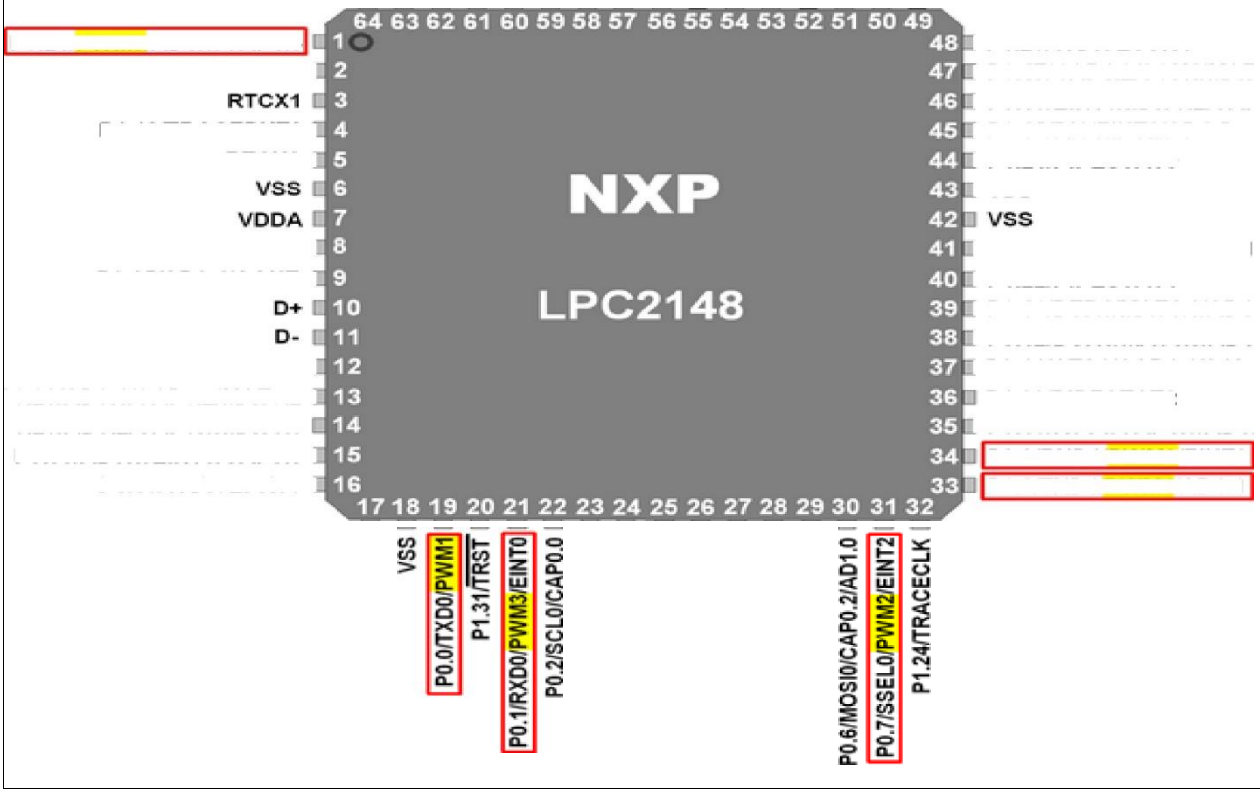


CLPC2148 PWM

- The PWM in LPC2148 is based on standard 32-bit Timer Counter, i.e. PWM Timer Counter (PWM TC).
- This Timer Counter counts the cycles of peripheral clock (PCLK).
- Also, we can scale this timer clock counts using 32-bit PWM Prescale Register (PWM PR).
- LPC2148 has 7 PWM Match Registers (PWM MR0 – PWM MR6).
- One Match Register (PWM MR0) is used to set PWM frequency.
- Remaining 6 Match Registers are used to set PWM width for
- 6 different PWM signals in Single Edge Controlled PWM or

- 3 different PWM signals in Double Edge Controlled PWM.
- Whenever PWM Timer Counter (PWM TC) matches with these Match Registers then, PWM Timer Counter resets, or stops, or generates match **interrupt, depending upon settings in PWM Match Control Register(PWM MCR)**

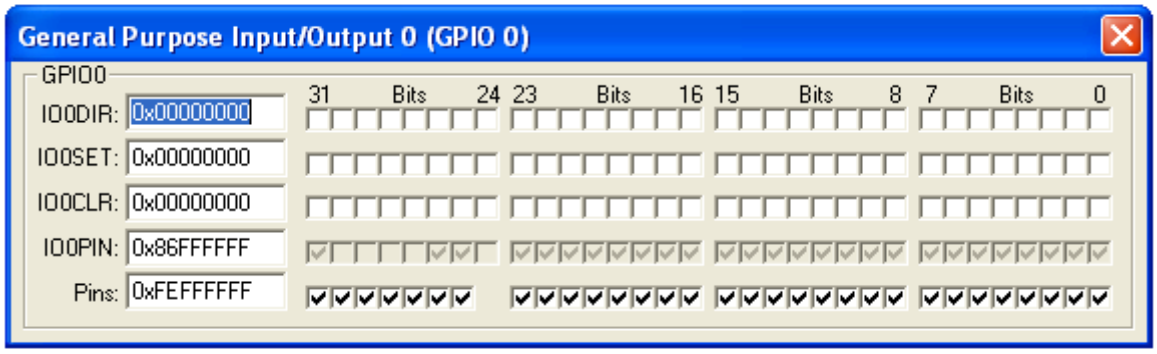




General Purpose Input/ Output 0 (GPIO 0) Dialog

- GPIO 0 controls the direction of the general purpose port 0 pins.
- The following controls can be used to select and configure the external interrupt settings:
- IO0DIR (Input Output Direction Register) a 32-bit register, controls the direction of each port pin. The checkboxes are checked for output (1-Output) and unchecked for input (0-Input).
- IO0SET (Input Output Set Register) a 32-bit register, used to make pins of Port (PORT0/ PORT1) HIGH. Writing one to specific bit makes that pin HIGH. Writing zero has no effect.
- IO0CLR (Input Output Clear Register) a 32-bit register, used to make pins of Port LOW. Writing one to specific bit makes that pin LOW. Writing zeroes has no effect.
- IO0PIN (Input Output Pin Value Register) a 32-bit register, used to read/ write the value on Port (PORT0/ PORT1).

Pins is used to manually control a pin value



ACK 0x00E00000	0000 0000 1110 0000 0000 0000 0000 0000
CLK 0x00D00000	0000 0000 1101 0000 0000 0000 0000 0000
DIR 0x000E0000	0000 0000 0000 1110 0000 0000 0000 0000
CLR 0x00F00000	0000 0000 1111 0000 0000 0000 0000 0000
PWM1 0x00000001	0000 0000 0000 0000 0000 0000 0000 0001
PWM2 0x00000008	0000 0000 0000 0000 0000 0000 0000 1000
ENBL 0x00000002	0000 0000 0000 0000 0000 0000 0000 0010
IO0DIR 0x00F0000B	0000 0000 1111 0000 0000 0000 0000 1011

7. PROGRAM:

```
#include<LPC214X.H>                    FRC4: DC MOTOR MODULE
                                       FRC1:KEYBOARD
#define ACK 0x00E00000    //DC Motor in Anticlockwise direction.
#define CLK 0x00D00000    //DC Motor in Clockwise direction.
#define DIR 0x000E0000    //DC Motor Direction Control. #define
CLR 0x00F00000

unsigned int PWM1 = 0x00000001;          //P0.0 (Pin 19)

unsigned int PWM2 = 0x00000008;          //P0.3 (Pin 26)

unsigned int ENBL = 0x00000002;          //P0.1 (Pin 21)

char scan (int);                         //For Scan the key.

void delay(int);                         //For Delay.

int main (void)

{

IO0DIR = 0x00F0000B;                     //Set the direction (I/O Ports).

while (1)

{

IO0SET |= ENBL;

/*IO0SET = IO0SET | ENBL = IO0SET | 0x00000002; Set PORT0.1 (i.e., Pin 21)*/

IO0CLR = CLR;                            //In1 = In2 = 0

IO0SET = ACK;                            //DC Motor in Anticlockwise direction.

while (scan (DIR))                       //S1

{

IO0SET |= PWM1;                          //In1 = 1

IO0CLR |= PWM2;                          //In2 = 0

}

IO0CLR = CLR;                  //In1 = In2 = 0
```

```
IO0SET = CLK;                          //DC Motor in Clockwise direction.
while(scan(DIR))                       //S2
{
IO0SET |= PWM2;                        //In2 = 1
IO0CLR |= PWM1;                        //In1 = 0
}
}
}
char scan (int keystatus)              //Scanning for a key.
{
while ((IO0PIN & 0x000F0000) == keystatus)
/*Current State of the GPIO configured port pins can always be read from IO0PIN register*/
/*Current State of the GPIO configured port pins can always be read from IO0PIN register*/
{
  return(1);          //Evaluates to True for a 'HIGH' on P0.16 to P0.
}
  Return(0);
 }
Void delay (int n) // generate one mili second delay
 {
Int i. j;
for (I = 1; i<=n; i++)
for (j=0; j<=10000; j++);
 }
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:

- DC MOTOR RUNS IN COUNTER-CLOCKWISE DIRECTION

12. RESULTS & CONCLUSIONS:

-

13. LEARNING OUTCOMES:

-

14. APPLICATION AREAS:

-

15. REMARKS:

-

1. EXPERIMENT NO: 11

2. TITLE: STEPPER MOTOR

3. LEARNING OBJECTIVES:

- TO LEARN THE INTERFACING OF STEPPER MOTOR TO THE MICROPROCESSOR AND TO ROTATE IT IN A SPECIFIED.

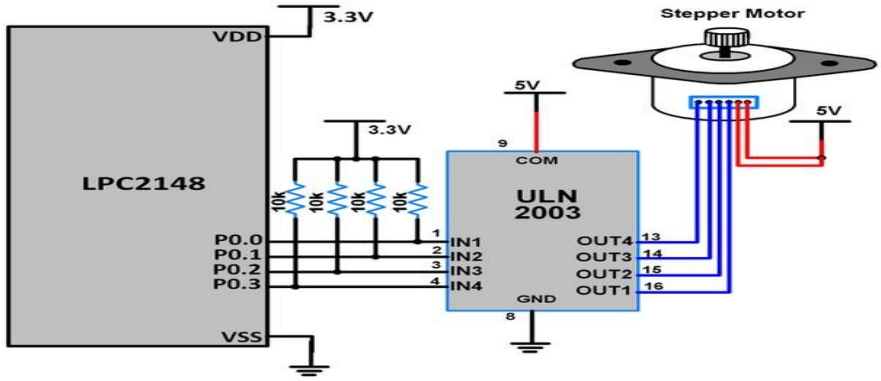
4. AIM:

- TO INTERFACE A STEPPER MOTOR AND ROTATE IT IN CLOCKWISE AND ANTI-CLOCKWISE DIRECTION.

5. MATERIAL / EQUIPMENT REQUIRED:

6. THEORY/ALGORITHM:

- A stepper motor is a brush less electric motor that divides a full rotation into a number of equal steps. The motor's position can then be commanded to move and hold at one of these steps without any position sensor for feedback, as long as the motor is carefully sized to the application in respect to torque and speed.



Stepper motors are widely used in industrial embedded applications, consumer electronic products and robotics control systems.

The paper feed mechanism of a printer/ fax makes use of stepper motors for its functioning.

* Based on the coil winding arrangements, a two-phase stepper motor is classified into two – Unipolar & Bipolar.

* The stepping of stepper motor can be implemented in different ways by changing the sequence of activation of the stator windings. The different stepping modes supported by stepper motor are: Full Step, Wave Step & Half Step.

STEP	FULL STEP			
	COIL A	COIL B	COIL C	COIL D
1	H	H	L	L
2	L	H	H	L
3	L	L	H	H
4	H	L	L	H

7. PROGRAM:

```
#include <LPC214x.H>        // Header file for LPC2148. FRC1:    Stepper Motor Module

#define MOTOR_PORT(DATA) (DATA << 16)
                                        //Pin from P0.16 to P1.19 configured as motor port.
unsigned char COUNT = 0;
unsigned int j = 0;
unsigned int CLOCK[4] = {0x03, 0x09, 0x0C, 0x06};        //Data for clockwise rotation.
unsigned int ANTI_CLOCK[4] = {0x06, 0x0C, 0x09, 0x03};        //Data for anti-clockwise rotatior


void DELAY_1S (unsigned int n)      //1s delay.
{
unsigned int i, j; for(i=1; i<=n; i++)
          for(j=0; j<=10000; j++);
}
void CLOCK_WISE_DIRECTION (unsigned int STEP, unsigned int TIME)
                                                //Function for clockwise direction.
{
for(j=0; j<=STEP; j++)
{


IO0PIN = MOTOR_PORT (CLOCK[COUNT]); COUNT ++;
DELAY_1S (TIME);
if (COUNT == 4) COUNT = 0;
}
}
void ANTI_CLOCK_WISE_DIRECTION (unsigned int STEP, unsigned int TIME)
                                                //Function for anti-clockwise direction
```

```
{
for(j=0; j<=STEP; j++)
{
IO0PIN = MOTOR_PORT (ANTI_CLOCK[COUNT]); COUNT ++;
DELAY_1S (TIME);
if (COUNT == 4) COUNT = 0;
}
}
int main ()     //Main function.
{
IO0DIR = 0x000F0000;          // Pin from P0.16 to P1.19 configured as output pins.
while(1)          //Infinite loop.
{
CLOCK_WISE_DIRECTION (200, 10);
//Clockwise direction with step and speed.
ANTI_CLOCK_WISE_DIRECTION (200, 10);
//Anti-clockwise direction with step and speed
}
}
```

8. PROCEDURE:

9. BLOCKDIAGRAM:

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:

11. OUTPUTS:
- STEPPER MOTOR RUNS IN COUNTER-CLOCKWISE DIRECTION.

12. RESULTS & CONCLUSIONS:
-

13. LEARNING OUTCOMES:
- THE STEPPER MOTOR CAN BE INTERFACED TO THE MICROPROCESSOR AND CAN BE CONTROLLED.

14. APPLICATION AREAS:
-

15. REMARKS:
-

1. EXPERIMENT NO: 12

2. TITLE: ADC

3. LEARNING OBJECTIVES:
- LEARN TO GENERATE DIGITAL OUTPUT FOR A GIVEN ANALOG INPUT USING INTERNAL ADC OF ARM CONTROLLER

4. AIM:
- TO DETERMINE DIGITAL OUTPUT FOR A GIVEN ANALOG INPUT USING INTERNAL ADC OF ARM CONTROLLER.

5. MATERIAL / EQUIPMENT REQUIRED:

- INTEL CPU BASED COMPUTER
- ADC INTERFACE KIT
- FRC CONNECTOR.

6. THEORY/ALGORITHM:

* Analog to Digital Converter (ADC) is used to convert input analog signal into digital form.

* LPC2148 has two inbuilt 10-bit ADC: ADC0 & ADC1.

* ADC0 has 6 channels & ADC1 has 8 channels (AD0.1:4, AD0.6:7 & AD1.7:0).

* Hence, we can connect 6 distinct types of input analog signals to ADC0 and 8 distinct types of input analog signals to ADC1.

* ADCs in LPC2148 use Successive Approximation technique (10-bit) to convert analog signal into digital form.

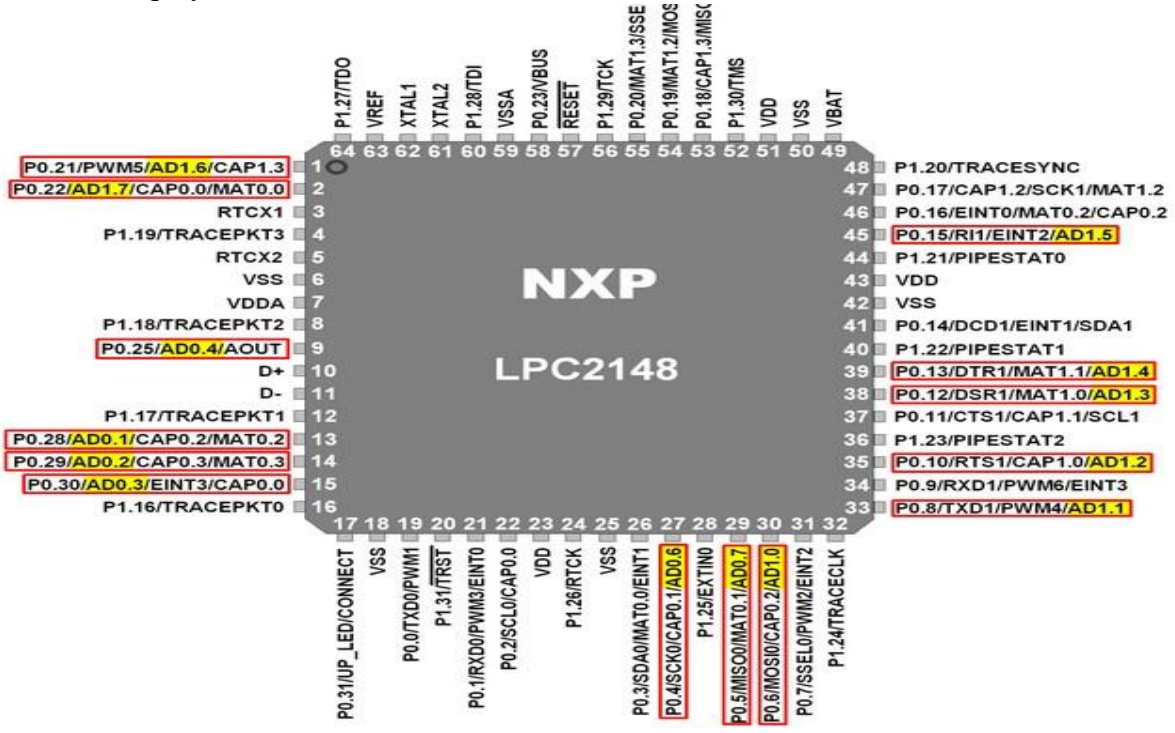
* Successive Approximation process requires a clock less than or equal to 4.5 MHz. We can adjust this clock using clock divider settings.

* LCDs (Liquid Crystal Displays) are used for displaying status or parameters in embedded systems.

* LCD 16x2 is 16 pin device which has 8 data pins (D0-D7) and 3 control pins (RS, RW, EN). The remaining 5 pins are for supply and backlight for the LCD.

* The control pins help us configure the LCD in command mode or data mode. They also help configure read mode or write mode and also when to read or write.

* LCD 16x2 can be used in 4-bit mode or 8-bit mode depending on the requirement of the application. In order to use it we need to send certain commands to the LCD in command mode and once the LCD is configured according to our need, we can send the required data in data mode to display on it.



COURSE LABORATORY MANUAL

No.	PIN	Function
1	VSS	Ground
2	VCC	+5 Volt
3	VEE	Contrast control 0 Volt: High contrast.

No.	PIN	Function
4	RS	Register Select 0: Commannd Reg. 1: Data Reg.
5	RW	Read / write 0: Write 1: Read
6	E	Enable H-L pulse
7-14	D0 - D7	Data Pins D7: Busy Flag Pin
15	LED+	+5 Volt
16	LED-	Ground

16x2 LCD Pin Description

```
/*GPIO Port 0 for LCD Command & GPIO Port 1 for LCD Data*/
unsigned int  EN = 0x00000800;      //P0.11 ENABLE LCD.
unsigned int  RW = 0x00000400;      //P0.10 READ/WRITE LCD.
unsigned int  RS = 0x00000200;      //P0.9  REGISTER SELECT LCD.
```

```
void INIT_GPIO()        //Controles the direction of Port Pins.
{
    IO0DIR = 0x00000E00;    //P0.9 to P0.11 for LCD Command (GP Port 0).
    IO1DIR = 0x00FF0000;    //P1.16 to P1.23 for LCD Data (GP Port 1).
}
```

Code (HEX)	Command to LCD
0x01	Clear the display screen
0x06	Shift the cursor right (e.g. data gets written in an incrementing order, left to right)
0x0C	Display on, cursor off
0x0E	Display on, cursor blinking
0x80	Force the cursor to the beginning of the 1st line
0xC0	Force the cursor to the beginning of the 2nd line
0x10	Shift cursor position to the left
0x14	Shift cursor position to the right
0x18	Shift entire display to the left
0x1C	Shift entire display to the right
0x38	2 lines, 5x8 matrix, 8-bit mode
0x28	2 lines, 5x8 matrix, 4-bit mode
0x30	1 line, 8-bit mode
0x20	1 line, 4-bit mode

EB 0x00000800 0000 0000 0000 0000 0000 1000 0000 0000

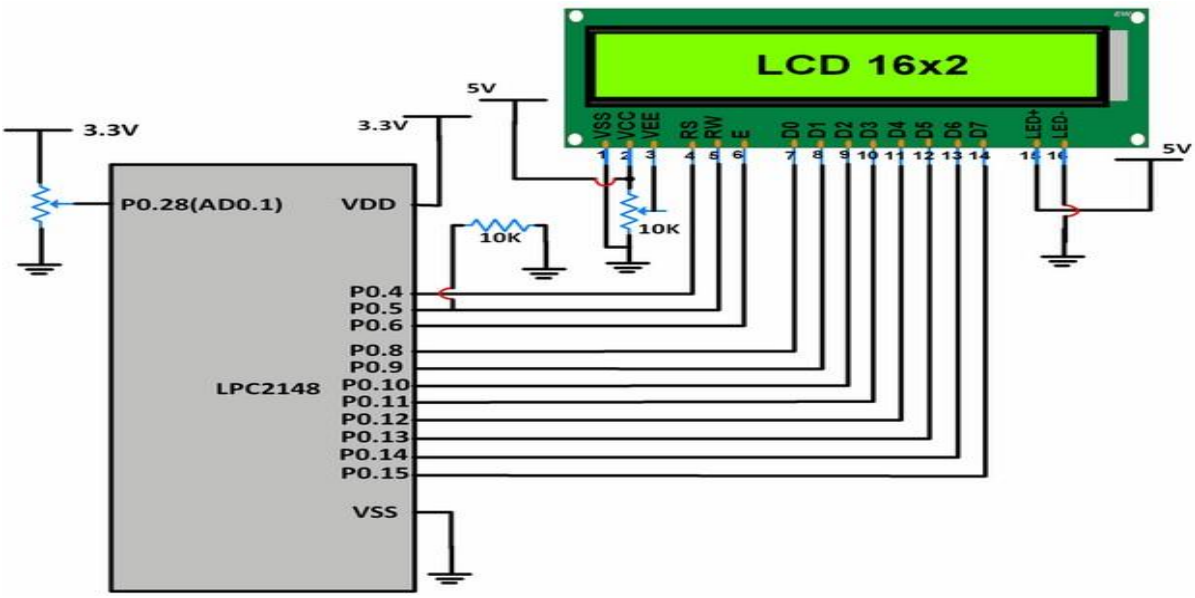
RW 0x00000400 0000 0000 0000 0000 0000 0100 0000 0000

RS 0x00000200 0000 0000 0000 0000 0000 0010 0000 0000

IO0DIR 0x00000E00 0000 0000 0000 0000 0000 1110 0000 0000

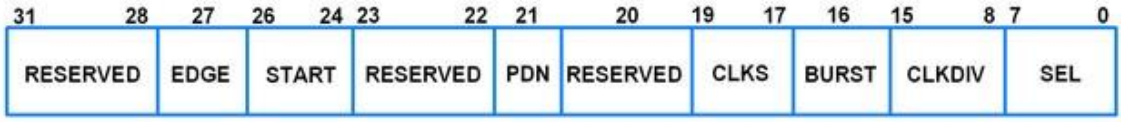
IO1DIR 0x00FF0000 0000 0000 1111 1111 0000 0000 0000 0000

EB 0x00000800	0000 0000 0000 0000 0000 1000 0000 0000
RW 0x00000400	0000 0000 0000 0000 0000 0100 0000 0000
RS 0x00000200	0000 0000 0000 0000 0000 0010 0000 0000
IO0DIR 0x00000E00	0000 0000 0000 0000 0000 1110 0000 0000
IO1DIR 0x00FF0000	0000 0000 1111 1111 0000 0000 0000 0000



AD0CR (ADC0 Control Register)

- AD0CR is a 32-bit register.
- This register must be written to select the operating mode before A/D conversion can occur.
- It is used for selecting channel of ADC, clock frequency for ADC, number of clocks or number of bits in result, start of conversion and few other parameters.



AD0CR (ADC0 Control Register)

- **Bits 7:0 – SEL**
  These bits select ADC0 channel as analog input. In software-controlled mode, only one of these bits should be 1.e.g. bit 7 (10000000) selects AD0.7 channel as analog input.
- **Bits 15:8 – CLKDIV**
  The APB(ARM Peripheral Bus)clock is divided by this value plus one, to produce the clock for ADC.
  This clock should be less than or equal to 4.5MHz.

- **Bit 16 – BURST**
  0 = Conversions are software controlled and require 11 clocks
  1 = In Burst mode ADC does repeated conversions at the rate selected by the **CLKS** field for the analog inputs selected by **SEL** field. It can be terminated by clearing this bit, but the conversion that is in progress will be completed.
  When Burst = 1, the **START** bits must be 000, otherwise the conversions will not start.
- **Bits 19:17 – CLKS**
  Selects the number of clocks used for each conversion in burst mode and the number of bits of accuracy of Result bits of AD0DR.
  e.g. 000 uses 11 clocks for each conversion and provide 10 bits of result in corresponding **ADDR** register.
  000 = 11 clocks / 10 bits
  001 = 10 clocks / 9 bits
  010 = 9 clocks / 8 bits
  011 = 8 clocks / 7 bits
  100 = 7 clocks / 6 bits
  101 = 6 clocks / 5 bits
  110 = 5 clocks / 4 bits
  111 = 4 clocks / 3 bits
- **Bit 21 – PDN**
  0 = ADC is in Power Down mode 1 = ADC is operational
- **Bit 26:24 – START**
  When **BURST** bit is 0, these bits control whether and when A/D conversion is started
  000 = No start (Should be used when clearing PDN to 0)
  001 = Start conversion now
  010 = Start conversion when edge selected by bit 27 of this register occurs on CAP0.2/MAT0.2 pin
  011= Start conversion when edge selected by bit 27 of this register occurs on CAP0.0/MAT0.0 pin
  100 = Start conversion when edge selected by bit 27 of this register occurs on MAT0.1 pin
  101 = Start conversion when edge selected by bit 27 of this register occurs on MAT0.3 pin
  110 = Start conversion when edge selected by bit 27 of this register occurs on MAT1.0 pin
  111 = Start conversion when edge selected by bit 27 of this register occurs on MAT1.1 pin
- **Bit 27 – EDGE**
  This bit is significant only when the Start field contains 010-111. In these cases,
  0 = Start conversion on a rising edge on the selected CAP/MAT signal
  1 = Start conversion on a falling edge on the selected CAP/MAT signal

AD0GDR (ADC0 Global Data Register)

- AD0GDR is a 32-bit register.
- This register contains the ADC's DONE bit and the result of the most recent A/D conversion.

- **Bit 5:0 – RESERVED**
- **Bits 15:6 – RESULT**
  When **DONE** bit is set to 1, this field contains 10-bit ADC result that has a value in the range of 0 (less than or equal to VSSA) to 1023 (greater than or equal to VREF).
- **Bits 26:24 – CHN**
  These bits contain the channel from which ADC value is read.
  e.g. 000 identifies that the **RESULT** field contains ADC value of channel 0.
- **Bit 30 – Overrun**
  This bit is set to 1 in burst mode if the result of one or more conversions is lost and overwritten before the conversion that produced the result in the **RESULT** bits.
  This bit is cleared by reading this register.
- **Bit 31 – DONE**
  This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read and when the AD0CR is written.
  If AD0CR is written while a conversion is still in progress, this bit is set and new conversion is started.

ADGSR (A/D Global Start Register)

- ADGSR is a 32-bit register.
- Software can write to this register to simultaneously start conversions on both ADC.

ADGSR (A/D Global Start Register)

- **BURST (Bit 16), START (Bit <26:24>) & EDGE (Bit 27)**
  These bits have same function as in the individual ADC control registers i.e. AD0CR & AD1CR. Only difference is that we can use these function for both ADC commonly from this register.

AD0STAT (ADC0 Status Register)

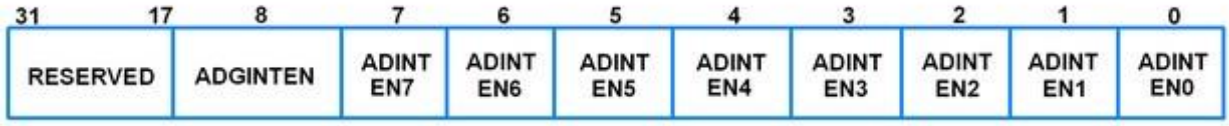
- AD0STAT is a 32-bit register.
- It allows checking of status of all the A/D channels simultaneously.

AD0STAT (ADC0 Status Register)

- **Bit 7:0 – DONE7:DONE0**
  These bits reflect the **DONE** status flag from the result registers for A/D channel 7 - channel 0.
- **Bit 15:8 – OVERRUN7:OVERRUN0**
  These bits reflect the **OVERRUN** status flag from the result registers for A/D channel 7 - channel 0.
- **Bit 16 – ADINT**
  This bit is 1 when any of the individual A/D channel DONE flags is asserted and enables ADC interrupt if any of interrupt is enabled in AD0INTEN register.

AD0INTEN (ADC0 Interrupt Enable)

- AD0INTEN is a 32-bit register.
- It allows control over which channels generate an interrupt when conversion is completed.

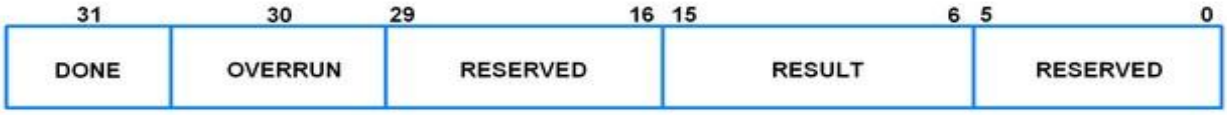


AD0INTEN (ADC0 Interrupt Enable)

- **Bit 0 – ADINTEN0**
  0 = Completion of a A/D conversion on ADC channel 0 will not generate an interrupt
  1 = Completion of a conversion on ADC channel 0 will generate an interrupt
- Remaining **ADINTEN** bits have similar description as given for **ADINTEN0**.
- **Bit 8 – ADGINTEN**
  0 = Only the individual ADC channels enabled by **ADINTEN7:0** will generate interrupts
  1 = Only the global **DONE** flag in A/D Data Register is enabled to generate an interrupt

AD0DR0-AD0DR7 (ADC0 Data Registers)

- These are 32-bit registers.
- They hold the result when A/D conversion is completed.
- They also include flags that indicate when a conversion has been completed and when a conversion overrun has occurred.



AD0 Data Registers Structure

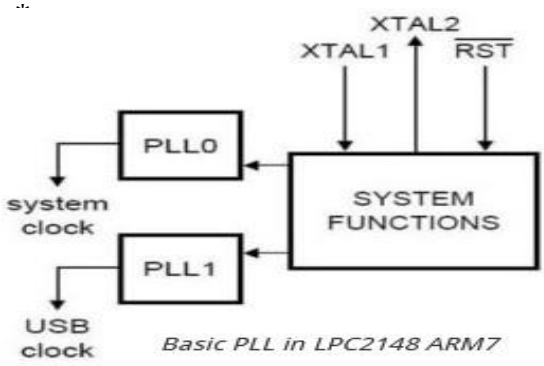
- **Bits 15:6 – RESULT**
  When **DONE** bit is set to 1, this field contains 10-bit ADC result that has a value in the range of 0 (less than or equal to VSSA) to 1023 (greater than or equal to VREF).
- **Bit 30 – Overrun**
  This bit is set to 1 in burst mode if the result of one or more conversions is lost and overwritten before the conversion that produced the result in the **RESULT** bits. This bit is cleared by reading this register.
- **Bit 31 – DONE**
  This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read.

PHASE LOCKED LOOP (PLL):

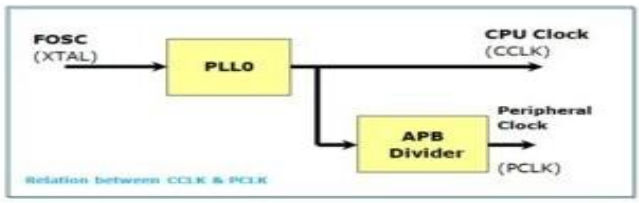
* PLL is used to generate system clock.

* PLL may multiply frequency to range from 10 MHz to 60 MHz (LPC21xx Series).

* PLL0 for System Clock & PLL1 for USB Clock



Basic PLL in LPC2148 ARM7



Setting PCLK in LPC2148 ARM7

CCLK to generate PCLK as shown below:

To control ABP Divider we have register called VPBDIV. The value in controls the division of CCLK to generate PCLK as shown below:

VPBDIV=0x00 APB bus clock (PCLK) is one fourth of the processor clock (CCLK)

VPBDIV=0x01 APB bus clock (PCLK) is the same as the processor clock (CCLK)

VPBDIV=0x02 APB bus clock (PCLK) is one half of the processor clock (CCLK)

VPBDIV=0x03 Reserved.

PLL Registers in LPC2148

Gen. Name	Description	PLL0	PLL1
PLLCON	PLL Control Register. Holding register for updating PLL control bits	0xE01F C080 PLL0CON	0xE01F C0A0 PLL1CON
PLLCFG	PLL Configuration Register. Holding register for updating PLL configuration values	0 0xE01F C084 PLL0CFG	0xE01F C0A4 PLL1CFG
PLLSTAT	PLL Status Register. Read-back register for PLL control and configuration information	0xE01F C088 PLL0STAT	0xE01F C0A8 PLL1STAT
PLLFEED	PLL Feed Register. This register enables loading of the PLL control and configuration information from the PLLCON and PLLCFG registers into the shadow registers that actually affect PLL operation.	0xE01F C08C PLL0FEED	0xE01F C0AC PLL1FEED

SETTING PLL IN LPC 2148:

* CCLK = M * FOSC = 5 * 12 MHz = 60 MHz
* CCLK – CPU Clock; M – PLL Multiplier; FOSC – Frequency of Oscillator.
* **FCCO = CCLK * 2 * P = 60*2*2 = 240 MHz**
* FCCO – Frequency of PLL Current Controlled Oscillator (should be with in the range 150 MHz to 320 MHz); P – PLL Divider.
* So for our calculation M = 5 and P = 2; then PLLCFG = 0b00100100 = 0x24.
* To avoid the accidental miss-configuration of the PLL, we need to use a Feed Sequence. A feed sequence is initialized whenever we are trying to configure PLL.
* Feed sequence acts as a key to unlock the PLL configuration.

PLL Divider values

PSEL Bits (PLLCFG bits [6:5])	Value of P
00	1
01	2
10	4
11	8

PLL Multiplier values

MSEL Bits (PLLCFG bits [4:0])	Value of M
00000	1
00001	2
00010	3
00011	4
...	...
11110	31
11111	32

7. PROGRAM:

```
#include <LPC214x.H>              FRC5: ADC Module
#include <STRING.H>               FRC2: LCD Card
void DELAY(unsigned int);         //Function declaration for Delay.
void INIT_GPIO(void);             //Controls the direction of Port Pins.
void INIT_PLL(void);              //PLL is used to generate system clock
void INIT_LCD(void);
void INIT_ADC(void);
void LCDSTR(unsigned char ADDRESS, char *MSG);
void LCD_DATA(unsigned char);
void LCD_CMD(unsigned int); int READ_ADC(void);
void CONVERT_ADC(unsigned int);
/*GPIO Port 0 for LCD Command & GPIO Port 1 for LCD Data*/
unsigned int    EN = 0x00000800;   //P0.11    ENABLE LCD.
unsigned int    RW = 0x00000400;   //P0.10    READ/WRITE LCD.
unsigned int    RS = 0x00000200;   //P0.9     REGISTER SELECT LCD.
unsigned int TEMP = 0, ADC = 0;
int main()
{
 INIT_GPIO();          //Controls the direction of Port Pins.
 INIT_PLL(); //Function to generate the PCLK.
 INIT_LCD();
 INIT_ADC();
 LCDSTR(0x00000080,"ADC");
 LCDSTR(0x000000C0,"VOLTAGE =");
 while(1)
```

```
{
 TEMP = READ_ADC();
 ADC = 1000*(TEMP*(3.901/1024));
          LCD_CMD(0x000000CA);
CONVERT  ADC(ADC); } }
void DELAY(unsigned int VALUE)   //Function for Delay.
{
unsigned int i, j; for(i=0;i<VALUE;i++)
{
for(j=1;j<1200;j++);
}
}
void INIT_GPIO()      //Controls the direction of Port Pins.
{
IO0DIR = 0x00000E00;         //P0.9 to P0.11 for LCD Command (GP Port 0).
IO1DIR = 0x00FF0000;         //P1.16 to P1.23 for LCD Data (GP Port 1).
}
void INIT_PLL()                 //Function to generate the PCLK.
{
PLL0CFG = 0x00000024;      //Set Multiplier and Divider of PLL to generate 60.0 MHz CCLK.
PLL0CON = 0x00000001; //Enable the PLL (PLL0CON-PLL Control Register-to Enable &
Control the PLL).
PLL0FEED = 0x000000AA;  //Update the PLL register with feed sequence.
PLL0FEED = 0x00000055;
while(!(PLL0STAT & 0x00000400)); //Test Lock bit (10th bit indicates lock status of the PLL.
PLL0CON = 0x00000003;      //Connect PLL after PLL is locked.
PLL0FEED = 0x000000AA;  //Feed sequence.
PLL0FEED = 0x00000055;
VPBDIV = 0x00000002;         //Set the Peripheral Clock: PCLK=1/2*CCLK = 30.0 MHz.
}
void LCD_DATA(unsigned char VALUE)
{
unsigned int b, TEMP;
TEMP = VALUE;
for(b=0;b<16;b++)
{          TEMP = TEMP<<1;   }
IO1SET = TEMP;
IO1CLR = ~TEMP;
```

```
IO0CLR = RW;          //RW Low (RW = 0, for write operation).
IO0SET = RS;          //RS High (For Data Mode, RS = 1).
IO0SET = EN;          //EN High.
DELAY(50);
IO0CLR = EN;          //EN Low.
}
void LCD_CMD(unsigned int LCMD)     //Command write function.
{
unsigned int b; for(b=0;b<16;b++)
{         LCMD = LCMD<<1;  }
IO1SET = LCMD;
IO1CLR = ~LCMD;
IO0CLR = RW;          //RW Low (RW = 0, for write operation).
IO0CLR = RS;          //RS Low (For Command Mode, RS = 0).
IO0SET = EN;          //EN High.
DELAY(50);
IO0CLR = EN;          //EN Low.
}
void LCDSTR(unsigned char ADDRESS,char *MSG)
{
unsigned char COUNT,LENGTH;
LCD_CMD(ADDRESS);
LENGTH = strlen(MSG);
for(COUNT=0;COUNT<LENGTH;COUNT++)
{
LCD_DATA(*MSG); MSG++;
}
}
 void INIT_LCD()
{
LCD_CMD(0x00000038);     //Initialize LCD Command.
LCD_CMD(0x0000000C);     //Display ON, Curzor OFF.
LCD_CMD(0x00000001);     //Display Clear.
LCD_CMD(0x00000006);     //Auto increment Curzor.
}
void INIT_ADC()
{
```

```
PINSEL1 |= 0x01000000;      //Select P0.28 as AD0.1
AD0CR = 0x00200602;          //AD0 Operational; with Clock = PCLK/(6+1) = 4.286 KHz.
}
int READ_ADC()
{
int VAL;
AD0CR |= 0x01000000;         //Start A/D Conversion by writing the value (24th bit) in AD0CR.
do
{
VAL = AD0DR1;        //Read A/D Data Register.
} while (!(VAL & 0x80000000));      //Wait for end of A/D Conversion (Check 31st bit of AD0DR1).
AD0CR &= ~0x01000000;    //Stop A/D Conversion on AD0.1.
VAL >>=6;     //Extract AD Result.
VAL = VAL & 0x3FF;
return(VAL);
}
void CONVERT_ADC(unsigned int ADC)
{
unsigned int CNT1, CNT2, CNT3, CNT4;
unsigned char DATA1, DATA2, DATA3, DATA4;
CNT1 = ADC % 10000/1000;
CNT2 = ADC % 1000/100;
CNT3 = ADC % 100/10;
 CNT4 = ADC % 10/1;
DATA1 = CNT1 | 0x30;
DATA2 = CNT2 | 0x30;
DATA3 = CNT3 | 0x30;
DATA4 = CNT4 | 0x30;
LCD_DATA(DATA1);
LCD_DATA('.');
LCD_DATA(DATA2); LCD_DATA(DATA3); LCD_DATA(DATA4);
return;
}
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:

- STEPPER MOTOR RUNS IN COUNTER-CLOCKWISE DIRECTION, FOR PRECISE STEPS AND THEN STOPS.

12. RESULTS & CONCLUSIONS:
    -

13. LEARNING OUTCOMES:
    - DIGITAL OUTPUT FOR A GIVEN ANALOG INPUT USING INTERNAL ADC OF ARM CONTROLLER ARE GENEERATED.

14. APPLICATION AREAS:
    -

15. REMARKS:
    -

1. EXPERIMENT NO: 13

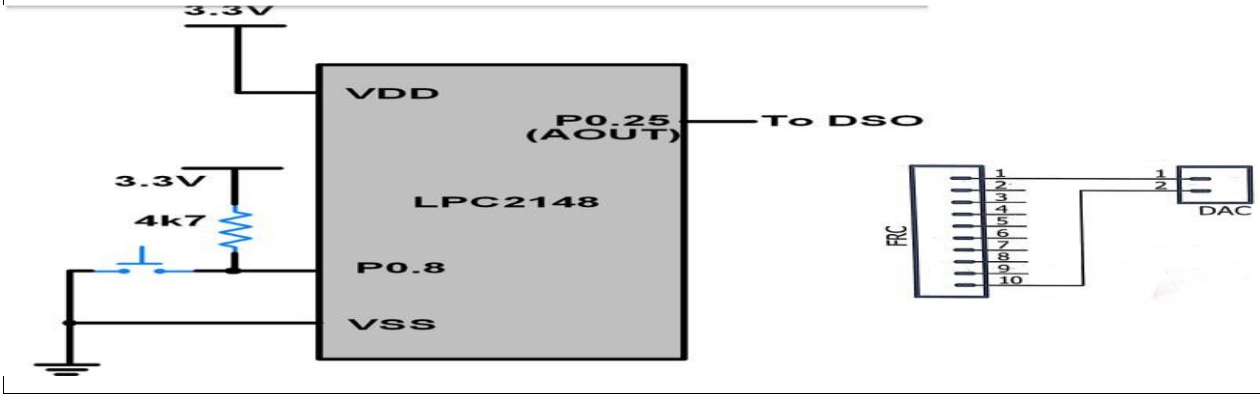
2. TITLE: DAC

3. LEARNING OBJECTIVES:
    - LEARN THE GENERATION OF TRIANGLAR AND SQUARE WAVE FORMS USING MICROPROCESSORS

4. AIM:
    - INTERFACE A DAC AND GENERATE TRIANGULAR AND SQUARE WAVEFORMS

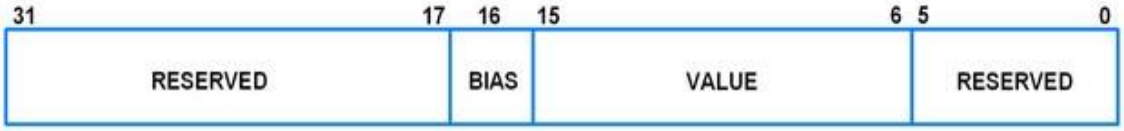
5. MATERIAL / EQUIPMENT REQUIRED:
    - INTEL CPU BASED COMPUTER
    - DAC INTERFACE KIT
    - FRC CONNECTOR.

6. THEORY/ALGORITHM:
    - Digital to Analog Converter (DAC) are mostly used to generate analog signals (e.g. Sine wave, Square wave, triangular wave etc.) from digital values.
    - LPC2148 has 10-bit DAC with resistor string architecture.
    - LPC2148 has Analog output pin (AOUT) on chip, where we can get digital value in the form of Analog output voltage.
    - The Analog voltage on AOUT pin is calculated as ((VALUE/1024) * VREF). Hence, we can change voltage by changing VALUE (10-bit digital value) field in DACR (DAC Register).
    - E.g.: If we set VALUE = 512, then, we can get analog voltage on AOUT pin as ((512/1024) * VREF) = VREF/2.



DACR (DAC Register)

- DACR is a 32-bit register.
- It is a read-write register.



DACR (DAC Register)

- **Bit 5:0 – RESERVED**
- **Bits 15:6 – VALUE**
  This field contains the 10-bit digital value that is to be converted in to Analog voltage. We can get Analog output voltage on AOUT pin and it is calculated with the formula (VALUE/1024) * VREF.
- **Bit 16 – BIAS**
  0 = Maximum settling time of 1µsec and maximum current is 700µA
  1 = Settling time of 2.5µsec and maximum current is 350µA
  Note that, the settling times are valid for a capacitance load on the AOUT pin not exceeding 100 pF. A load impedance value greater than that value will cause settling time longer than the specified time.
- **Bit 31:17 – RESERVED**

Programming Steps:

- First, configure P0.25/AOUT pin as DAC output using PINSEL Register.
- Then set settling time using BIAS bit in DACR Register.
- Now write 10-bit value (which we want to convert into analog form) in VALUE field of DACR Register.

7. PROGRAM:

```
#include<lpc214x.h>
#include"math.h"
unsigned int t;
void Delay(unsigned long b)
{
while (--b!=0);
}
// FRC5: DAC Module Sine wave
int main()
```

```
{
int value = 0x3ff; PINSEL1 = 0x00080000;
/*P0.25 as DAC output.*/ Delay(1);
while(1)
{
value = 512+400*sin(t*0.01745);
DACR = value<<6;
if(++t==360)t=0;
}
}
#include<lpc214x.h>
void delay(int n)
{
int i,j;
for (i=1; i<=n; i++) for(j=0; j<=10000; j++);
}
```

FRC5: DAC Module Square wave

```
int main()
{
int value=0x3ff; PINSEL1 = 0x00080000;
/*P0.25 as DAC output. */
while(1)
{
DACR = value<<6; delay(1);
DACR = 0;
delay(1);
}
}
#include<lpc214x.h>
unsigned int J;
void delay(int n)
{
int i,j;
for (i=1; i<=n; i++) for(j=0; j<=10000; j++);
}
```

FRC5: DAC Module

Triangular wave

```
int main()
{
{
PINSEL1 = 0x00080000;
/*P0.25 as DAC output. */
while(1)
{       for (J=0;J<1023;J++)
{       DACR = J<<6;        } for (J=1023;J>0;J--)
{       DACR = J<<6;        }
}
}
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:
    -

12. RESULTS & CONCLUSIONS:
    -

13. LEARNING OUTCOMES:
    - THE TRAINGULAR AND SQUARE WAVEFORM CAN BE GENERATED BY THE MICROPROCESSOR BY GIVING SUITABLE VALUES TO THE ADC.

14. APPLICATION AREAS:
    -

15. REMARKS:
    -

1. EXPERIMENT NO: 14

2. TITLE: 4x4 KEYBOARD

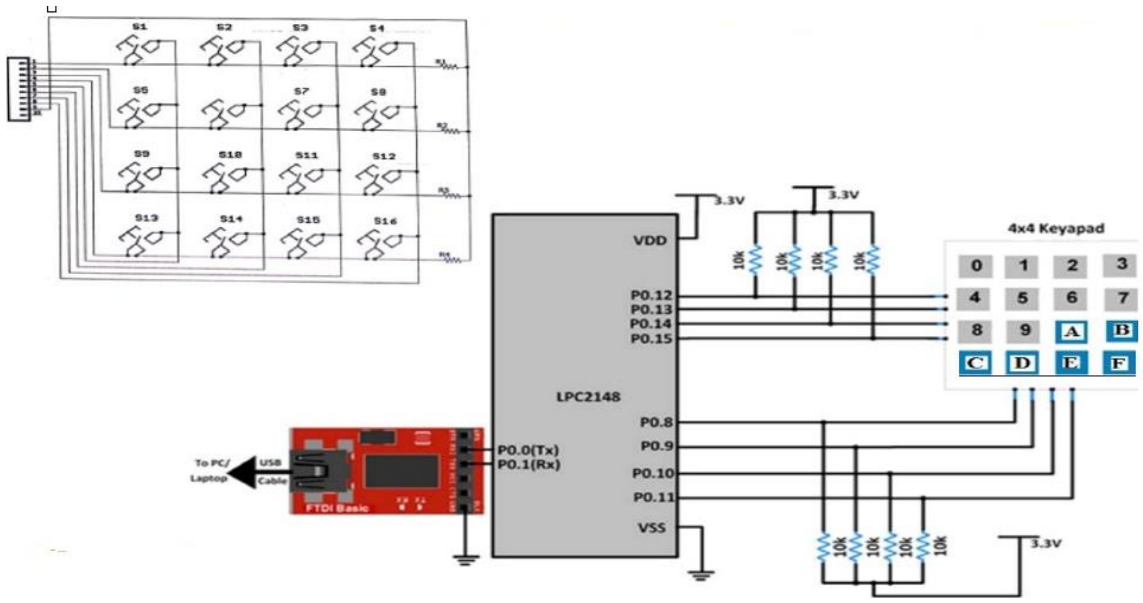
3. LEARNING OBJECTIVES:
    - LEARN TO INTERFACE A 4X4 KEYBOARD AND DISPLAY THE KEY CODE ON AN KCD

4. AIM:
    - INTERFACE A 4X4 KEYBOARD AND DISPLAY THE KEY CODE ON AN LCD

5. MATERIAL / EQUIPMENT REQUIRED:
    - INTEL CPU BASED COMPUTER
    - KEYBOARD INTERFACE KIT
    - FRC CONNECTOR.

6. THEORY/ALGORITHM:
    - Keypad is used as an input device to read the key pressed by the user and to process it.

- 4x4 keypad consists of 4 rows and 4 columns. Switches are placed between the rows and columns. A key press establishes a connection between corresponding row and column between which the switch is placed.
- In order to read the key press, we need to configure the rows as outputs and columns as inputs.
- Columns are read after applying signals to the rows in order to determine whether or not a key is pressed and if pressed, which key is pressed.



O1 0x00E00000	0000 0000 1110 0000 0000 0000 0000 0000
O2 0x00D00000	0000 0000 1101 0000 0000 0000 0000 0000
O3 0x00B00000	0000 0000 1011 0000 0000 0000 0000 0000
O4 0x00700000	0000 0000 0111 0000 0000 0000 0000 0000
I1 0x000E0000	0000 0000 0000 1110 0000 0000 0000 0000
I2 0x000D0000	0000 0000 0000 1101 0000 0000 0000 0000
I3 0x000B0000	0000 0000 0000 1011 0000 0000 0000 0000
I4 0x00070000	0000 0000 0000 0111 0000 0000 0000 0000
CLR 0x00F00000	0000 0000 1111 0000 0000 0000 0000 0000

Key.c

```
001 #include <LPC214x.H>
002 #include <STRING.H>
003
004 #define O1 0x00E00000
005 #define O2 0x00D00000
006 #define O3 0x00B00000
007 #define O4 0x00700000
008
009 #define I1 0x000E0000
010 #define I2 0x000D0000
011 #define I3 0x000B0000
012 #define I4 0x00070000
013
014 #define CLR 0x00F00000
```

7. PROGRAM:

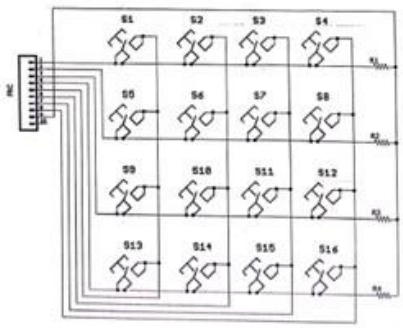
```
#include <LPC214x.H>
#include <STRING.H>
#define O1 0x00E00000
#define O2 0x00D00000
```

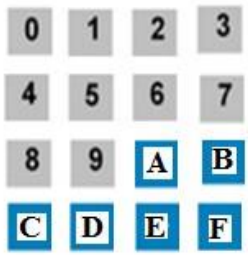
FRC1: 4 x 4 Keypad

FRC2: LCD Card

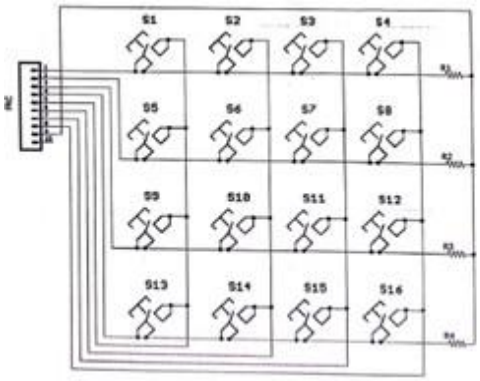
```
#define O3 0x00B00000
#define O4 0x00700000
#define I1 0x000E0000
#define I2 0x000D0000
#define I3 0x000B0000
#define I4 0x00070000
void DELAY(unsigned int);   //Function declaration for Delay.
void INIT_GPIO(void);       //Controls the direction of Port Pins.
void INIT_PLL(void);        //PLL is used to generate system clock.
void INIT_LCD(void);
void LCDSTR(unsigned char ADDRESS, char *MSG);
void LCD_DATA(unsigned char);
void LCD_CMD(unsigned int);
char scan (int);
void delay(int);
/*GPIO Port 0 for LCD Command & GPIO Port 1 for LCD Data*/
unsigned int
unsigned int    EN = 0x00000800;
RW = 0x00000400;    //P0.11
//P0.10 ENABLE LCD.
READ/WRITE LCD.
unsigned int    RS = 0x00000200;     //P0.9   REGISTER SELECT LCD.
int main()
{
INIT_GPIO(); //Controls the direction of Port Pins.
INIT_PLL();   //Function to generate the PCLK. INIT_LCD();
LCDSTR(0x00000080,"Matrix KeyPad");
LCDSTR(0x000000C0,"Key Pressed:");
while(1)
{
IO0CLR = CLR;
IO0SET = O1;
delay(10);
if(scan(I1))LCDSTR(0x000000CC,"0");      //S1
if(scan(I2))LCDSTR(0x000000CC,"4");      //S5
```

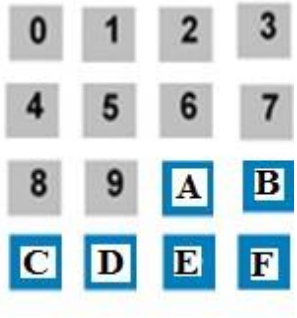
```
if(scan(I3))LCDSTR(0x000000CC,"8");      //S9
if(scan(I4))LCDSTR(0x000000CC,"C");      //S13
```





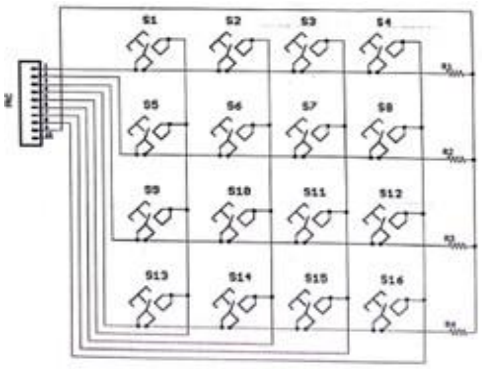

```
IO0CLR = CLR;
IO0SET = O2;
if(scan(I1))LCDSTR(0x000000CC,"1");      //S2
if(scan(I2))LCDSTR(0x000000CC,"5");      //S6
if(scan(I3))LCDSTR(0x000000CC,"9");      //S10
if(scan(I4))LCDSTR(0x000000CC,"D");      //S14
```

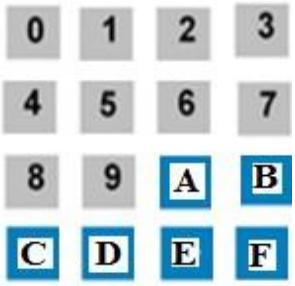





```
IO0CLR = CLR;
IO0SET = O3;
if(scan(I1))LCDSTR(0x000000CC,"2");      //S3
if(scan(I2))LCDSTR(0x000000CC,"6");      //S7
if(scan(I3))LCDSTR(0x000000CC,"A");      //S11
```

```
if(scan(I4))LCDSTR(0x000000CC,"E");      //S15
```






```
IO0CLR = CLR;
IO0SET = O4;
if(scan(I1))LCDSTR(0x000000CC,"3");      //S4
if(scan(I2))LCDSTR(0x000000CC,"7");      //S8
if(scan(I3))LCDSTR(0x000000CC,"B");      //S12
if(scan(I4))LCDSTR(0x000000CC,"F");      //S16
}
}
char scan(int keystatus)          /* Scanning for a key */
{
while((IO0PIN & 0x000F0000) == keystatus)
{
delay(50);
if((IO0PIN & 0x000F0000) == 0x000F0000)return(1);
}
return(0) ;
}
void delay(int n)                 /* Generates one millisecond delay */
{
int i, j;
for (i=1; i<=n; i++) for(j=0; j<=10000; j++);
}

void DELAY(unsigned int VALUE)
{
unsigned int i, j; for(i=0;i<VALUE;i++)
```

```
{
for(j=1;j<1200;j++);
 }
}
void INIT_GPIO()                    //Controls the direction of Port Pins.
{
IO0DIR =0x00F00E00;        // P0.8 to P0.15 LCD DATA (GP Port 0).
IO1DIR =0X00FF0000;        // P1.16 to P1.23 LCD CONTROL (GP Port 1).
}
void INIT_PLL()                     //Function to generate the PCLK.
{
PLL0CFG = 0x00000024; //Set Multiplier and Divider of PLL to generate 60.0 MHz CCLK.
PLL0CON = 0x00000001;     //Enable the PLL (PLL0CON-PLL Control Register-to Enable &
Control the PLL).
PLL0FEED = 0x000000AA;  //Update the PLL register with feed sequence.
PLL0FEED = 0x00000055;
while(!(PLL0STAT & 0x00000400));//Test Lock bit (10th bit indicates lock status of the PLL.
PLL0CON = 0x00000003;     //Connect PLL after PLL is locked. PLL0FEED = 0x000000AA;
          //Feed sequence.
PLL0FEED = 0x00000055;
VPBDIV = 0x00000002;        //Set the Peripheral Clock: PCLK=1/2*CCLK = 30.0 MHz.
}
void LCD_DATA(unsigned char VALUE)
{
unsigned int b, TEMP; TEMP = VALUE;
for(b=0;b<16;b++)
{         TEMP = TEMP<<1;   }
IO1SET = TEMP;
IO1CLR = ~TEMP;
IO0CLR = RW;          //RW Low (RW = 0, for write operation).
IO0SET = RS;//RS High (For Data Mode, RS = 1).
IO0SET = EN;//EN High.
DELAY(50);
IO0CLR = EN;          //EN Low.
}
```

```
void LCDSTR(unsigned char ADDRESS,char *MSG)
{
unsigned char COUNT,LENGTH;
LCD_CMD(ADDRESS);
LENGTH = strlen(MSG);
for(COUNT=0;COUNT<LENGTH;COUNT++)
{
LCD_DATA(*MSG); MSG++;
}
}
void INIT_LCD()
{
LCD_CMD(0x00000038);     //Initialize LCD Command.
LCD_CMD(0x0000000C);     //Display ON, Curzor OFF.
LCD_CMD(0x00000001);     //Display Clear.
LCD_CMD(0x00000006);     //Auto increment Curzor.
}
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:
    -

12. RESULTS & CONCLUSIONS:
    -

13. LEARNING OUTCOMES:
    - A 4X4 KEYBOARD AND DISPLAY THE KEY CODE ON AN LCD

14. APPLICATION AREAS:
    -

15. REMARKS:
    -

1. EXPERIMENT NO: 15

2. TITLE: INTERRUPT TO TOGGLE LED

3. LEARNING OBJECTIVES:
    - LEARN TO USE AN EXTERNAL INTERRUPT TO TOGGLE AN LED ON/OFF

4. AIM:
    - DEMONSTRATE THE USE AN EXTERNAL INTERRUPT TO TOGGLE AN LED ON/OFF

5. MATERIAL / EQUIPMENT REQUIRED:
    - INTEL CPU BASED COMPUTER

- INTERFACE KIT
- FRC CONNECTOR.

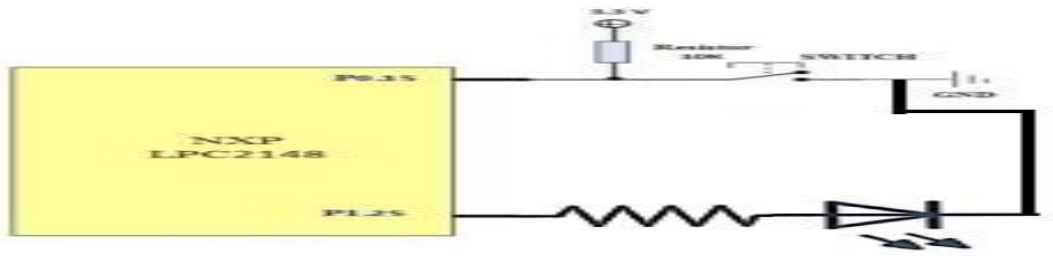
6. THEORY/ALGORITHM:

- External Interrupts are caused by an external source such as external switch, sensor or monitoring device.
- These interrupts are special events that require immediate attention.
- An interrupt can be configured to be set off by a variety of triggers; say for example, the chip awaits a signal change on a pin connected to a switch or sensor and timed interrupts.

Pin Name	Pin Direction	Pin Description
EINT0	External Interrupt Input 0	Pins P0.1 and P0.16 can be selected to perform EINT0 function
EINT1	External Interrupt Input 1	Pins P0.3 and P0.14 can be selected to perform EINT1 function
EINT2	External Interrupt Input 2	Pins P0.7 and P0.15 can be selected to perform EINT2 function
EINT3	External Interrupt Input 3	Pins P0.9, P0.20 and P0.30 can be selected to perform EINT3 function

- The external interrupt function has four registers associated with it.
- The EXTINT register contains the interrupt flags, and the EXTWAKEUP register contains bits that enable Individual external interrupts to wake up the microcontroller from Power-down mode. The EXTMODE and EXTPOLAR registers specify the level and edge sensitivity parameters.

Name	Description
EXTINT	The External Interrupt Flag Register contains interrupt flags for EINT0, EINT1, EINT2 and EINT3. See
INTWAKE	The Interrupt Wakeup Register contains four enable bits that control whether each external interrupt will cause the processor to wake up from Power-down mode. See
EXTMODE	The External Interrupt Mode Register controls whether each pin is edge- or level sensitive.
EXTPOLAR	The External Interrupt Polarity Register controls which level or edge on each pin will cause an interrupt.



7. PROGRAM:

FRC1: LED Module

```
#include<LPC21xx.h>
void DELAY(unsigned long VALUE);
void ext_interrupt(void) _irq  //Initialize interrupt vector and call ISR when interrupt occurs.
{
IO0SET |= (1<<16);
DELAY(5000000);
IO0CLR |= (1<<16);
EXTINT = 0x02;        //Clear Interrupt Flag.
VICVectAddr = 0x00000000;//Acknowledge Interrupt.
}
void init_ext(void)
{
IO0DIR |= (0xFF<<16);         //To turn LED on.
IO0CLR |= (0xFF<<16);
PINSEL0 |= 0x20000000;
/*Modify PINSEL0 register to change the function of pin P0.30 from General Purpose IO i.e. GPIO to External Interrupt 3 (EINT3).*/
EXTMODE = 0x02;   //Edge sensitive.
EXTPOLAR = 0x02;  //On rising edge.
VICVectAddr0 =(unsigned int)ext_interrupt;//Address of ISR.
VICVectCntl0 = 0x0000002F;          //Use it for External Interrupt.
/*There are 16 of these registers (VICVectCntl0 to VICVectCntl15), each controlling a vector IRQ Slot.
VICVectCntl0 has highest priority, while VICVectCntl15 has the lowest priority. Let's modify VICVectCntl0.*/
VICIntEnable = 0x00008000;//Enable the External Interrupt.
}
int main(void)
{
init_ext(); while(1)
{
}
}
void DELAY(unsigned long VALUE)
{
while(VALUE>0)
{
```

```
VALUE--;
        }
}
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:

-

12. RESULTS & CONCLUSIONS:

-

13. LEARNING OUTCOMES:

- DEMONSTRATED THE USE AN EXTERNAL INTERRUPT TO TOGGLE AN LED ON/OFF

14. APPLICATION AREAS:

-

15. REMARKS:

-

1. EXPERIMENT NO: 16

2. TITLE: 7-SEGMENT LED

3. LEARNING OBJECTIVES:

- TO LEARN THE INTERFACING OF 7SEGMENT LED DISPLAY KIT TO THE MICROPROCESSOR.

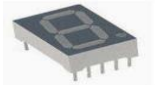
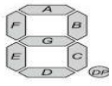
4. AIM:

- DISPLAY THE HEX DIGITS 0 TO F ON A 7-SEGMENT LED INTERFACE, WITH AN APPROPRIATE DELAY IN BETWEEN.

5. MATERIAL / EQUIPMENT REQUIRED:

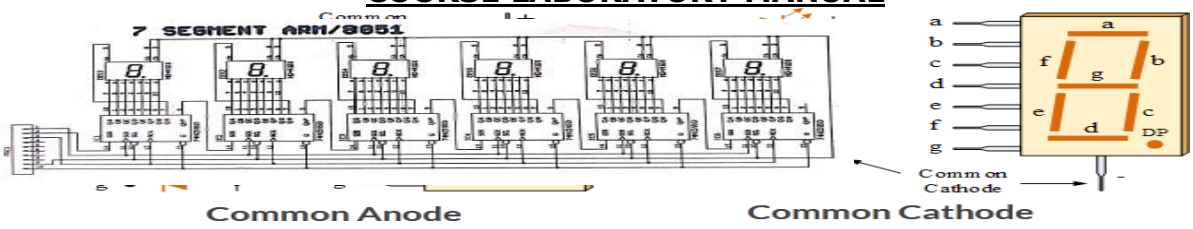
6. THEORY/ALGORITHM:

- 7-segment displays are the simplest display units to display the numbers and characters. It is generally used to display numbers and has brighter illumination and simpler construction.
- 7-segment display consists of 8 LEDs, each LED is used to illuminate one segment of unit and the 8th LED is used to illuminate DOT.
- A single (7-segment) module is used to display single digit or character. To display more than one digit or character, multiple 7-segments are used.

Pins of 7-Segment Display:

- There are 10 pins, in which 8 pins are used to refer a, b, c, d, e, f, g and h/dp; the two middle pins are common anode/ cathode of all LEDs. These common anode/ cathode are internally shorted so we need to connect only one COM pin

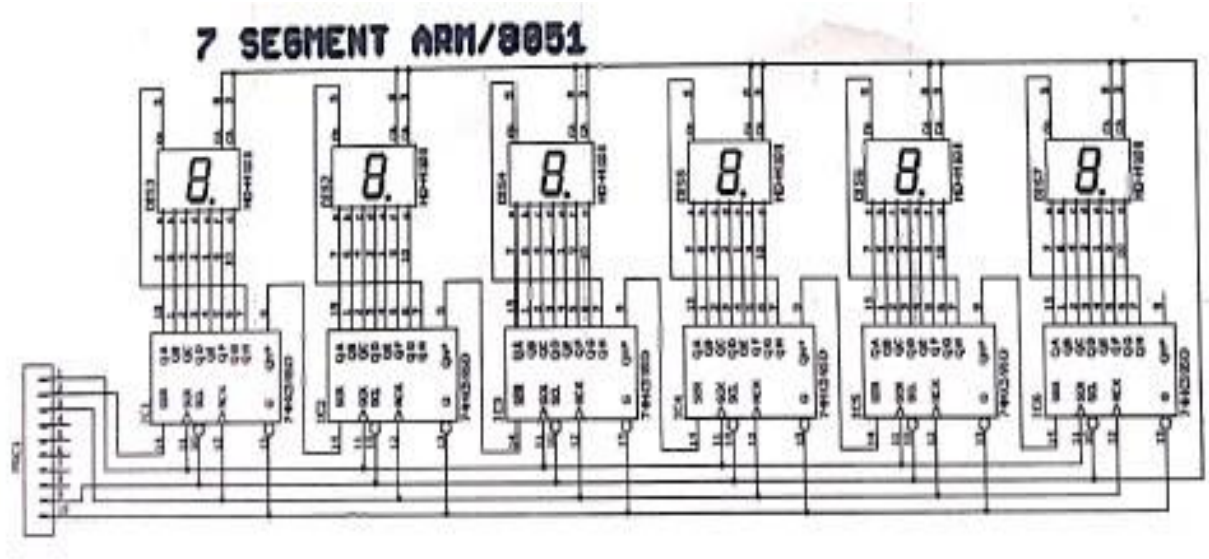


Common Anode Common Cathode

- Hence, for Common Cathode 7-segmewnt display – hgfe dcba hgfe dcba

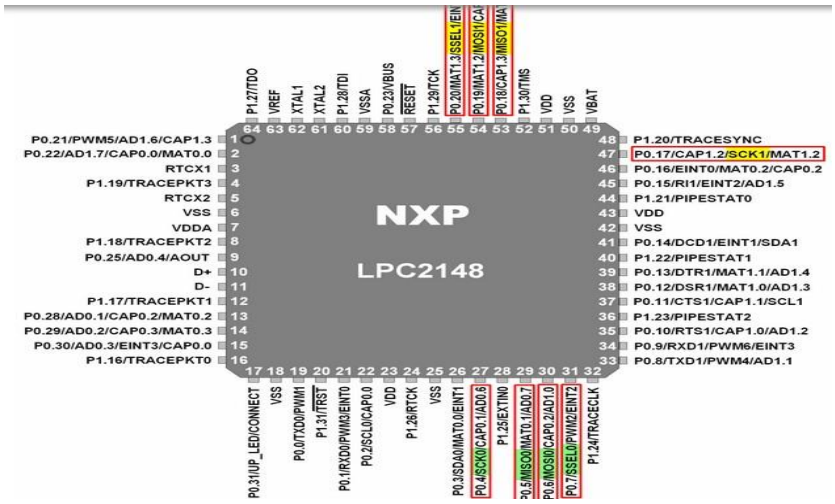
0 – 1100 1111 = 0xC0 A – 1000 1000 = 0x88

9 – 1001 0000 = 0x90 F – 1000 1110 = 0xC0



* The Serial Peripheral Interface (SPI) is a bus interface connection protocol originally started by Motorola Corp.

* LPC2148 has two inbuilt SPI modules i.e. SPI0 and SPI1/SSP (Synchronous Serial Port).



7. PROGRAM:

#include <LPC214x.H> FRC4: 7 Segment Module

```
/*Header file for LPC214x series Microcontrollers.*/

unsigned int    SCK = 0x00000001;   //P0.0 (Serial Clock for synchronized data transfer).
unsigned int    ser = 0x00000002;     //P1.1
unsigned int    rck = 0x00000004;     //P1.3


void delay(int);
void convert_display (unsigned int);
void Clear (void);       //Function to clear all 7-Segment Display Units.
int display(unsigned char);
unsigned char disp[10] =
{0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
unsigned int count = 0;
int main()
{
IO0DIR |= 0x00000007;        //Configure P0.0 to P0.2 as Output.
Clear();          //Clear all 7-Segment Display Units. while(1)
{
convert_display(count); count = count + 111111; delay(1000);
Clear();
if(count > 999999) count = 0;
}
}
void Clear (void)                        //Function to clear all 7-Segment Display Units.
{
unsigned int x;
for (x = 0; x < 6; x++)
{         display(0xFF);}
}
void delay(int n)        // Generates one millisecond delay.
{
int i,j;
for (i = 1; i <= n; i++) for(j = 0; j <= 10000; j++);
}
int display(unsigned char value)
{
char m; int buffer;
```

```
buffer = value;
for(m = 0; m < 8; m++)
{

if(buffer & 0x80) IO0SET |= ser; else IO0CLR |= ser;
IO0SET |= SCK;
buffer <<= 1; IO0CLR |= SCK;
}
IO0SET |= rck;
delay(1);
IO0CLR |= rck;
return 0;
}
void convert_display(unsigned int value)
{
display(disp[(value % 1000000/100000)]);
display(disp[(value % 100000/10000)]);
display(disp[(value % 10000/1000)]);
display(disp[(value % 1000/100)]);
display(disp[(value % 100/10)]);
display(disp[(value % 10/1)]);
return;
}
```

8. PROCEDURE:NA

9. BLOCKDIAGRAM:NA

10. OBSERVATION TABLE / LOOKUP TABLE / TRUTH TABLE:NA

11. OUTPUTS:
    -

12. RESULTS & CONCLUSIONS:
    -

13. LEARNING OUTCOMES:
    - ANY ALPHANUMERIC CHARACTER CAN BE DISPLAYED ON THE 7 SEGMENT LED UNIT.

14. APPLICATION AREAS:
    - CHARACTER DISPLAY ON A VIDEO DEVICE
    - MOVING MESSAGE DISPLAYS.

15. REMARKS:
    -