||Jai Sri Gurudev||

Sri Adichunchanagiri Shikshana Trust®

BGS INSTITUTE OF TECHNOLOGY

(Affiliated to VTU Belagavi, Approved by AICTE, New Delhi)

BG Nagara – 571448 (Bellur Cross)

Nagamangala Taluk, Mandya District.

EMBEDDED CONTROLLER LABORATORY MANUAL
15ECL67

For

VI Semester B.E.

2018-2019

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Prepared by:

1. Mr. Manoj Kumar S B
2. Mr.Balaji B S
3. Ms.Srividya C N

Approved by:

Professor & HOD
Dept. of Electronics & Communication Engg.
BGS Institute of Technology
BG Nagara - 571 448
Mandya District

Head of Department

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENFINEERING

VISION:

To develop high quality engineers with technical knowledge, skills and ethics in the area of Electronics and Communication Engineering to meet industrial and societal needs.

MISSION:

1. To provide high quality technical education with up-to-date infrastructure and trained human resources to deliver the curriculum effectively in order to impart technical knowledge and skills.
2. To train the students with entrepreneurship qualities, multidisciplinary knowledge and latest skill sets as required for industry, competitive examinations, higher studies and research activities.
3. To mould the students into professionally-ethical and socially- responsible engineers of high character, team spirit and leadership qualities.

PROGRAM EDUCATIONAL OBJECTIVES (PEO's):

After 3 to 5 years of graduation, the graduates of Electronics and Communication Engineering will;

1. Engage in industrial, teaching or any technical profession and pursue higher studies and research.
2. Apply the knowledge of Mathematics, Science as well as Electronics and Communication Engineering to solve social engineering problems.
3. Understand, Analyze, Design and Create novel products and solutions.
4. Display professional and leadership qualities, communication skills, team spirit, multidisciplinary traits and lifelong learning aptitude.

EMBEDDED CONTROLLER LAB SYLLABUS

Course Learning Objectives:

1. Choose the suitable instruction sets to multiply two 16 bit binary numbers and to find the sum of 10 numbers.
2. Apply the programming concept of GPIO registers to interact between ARM cortex M3 and programmer.
3. Adopt the keil software tools along with Assembly and C language for Interfacing applications.

Laboratory Experiments

PART-A

1. ALP to multiply two 16 bit binary numbers
2. ALP to find the sum of first 10 integer numbers.

PART-B

1. Display "Hello World" message using Internal UART.
2. Interface and control a DC motor.
3. Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.
4. Interface a DAC and generate Triangular and Square waveforms.
5. Interface a 4x4 keyboard and display the key code on an LCD.
6. using the Internal PWM module of ARM controller generate PWM and vary its duty cycle.
7. Demonstrate the use of an external interrupt
8. Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between.
9. Interface a simple Switch and display its status through Relay, Buzzer and LED
10. Measure ambient temperature using a sensor and SPI ADC IC.

Beyond Syllabus:

1. GPS MODULE

Course Outcomes: This course will enable students to:

· Understand the instruction set of ARM Cortex M3, a 32 bit microcontroller and the software tool required for programming in Assembly and C language.

· Program ARM Cortex M3 using the various instructions in assembly level language for different applications.

· Interface external devices and I/O with ARM Cortex M3.

· Develop C language programs and library functions for embedded system applications.

1. INTRODUCTION

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

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

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

Features:

- ARM Cortex-M3 processor, running at frequencies of up to 100 MHz (LPC-1768) A Memory Protection Unit (MPU) supporting eight regions is included.
- ARM Cortex-M3 built-in Nested Vectored Interrupt Controller (NVIC).
- Up to 512 kB on-chip flash programming memory. Enhanced flash memory accelerator enables high-speed 120 MHz operation with zero wait states.
- In-System Programming (ISP) and In-Application Programming (IAP) via on-chip boot loader software.

On-chip SRAM includes:

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

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

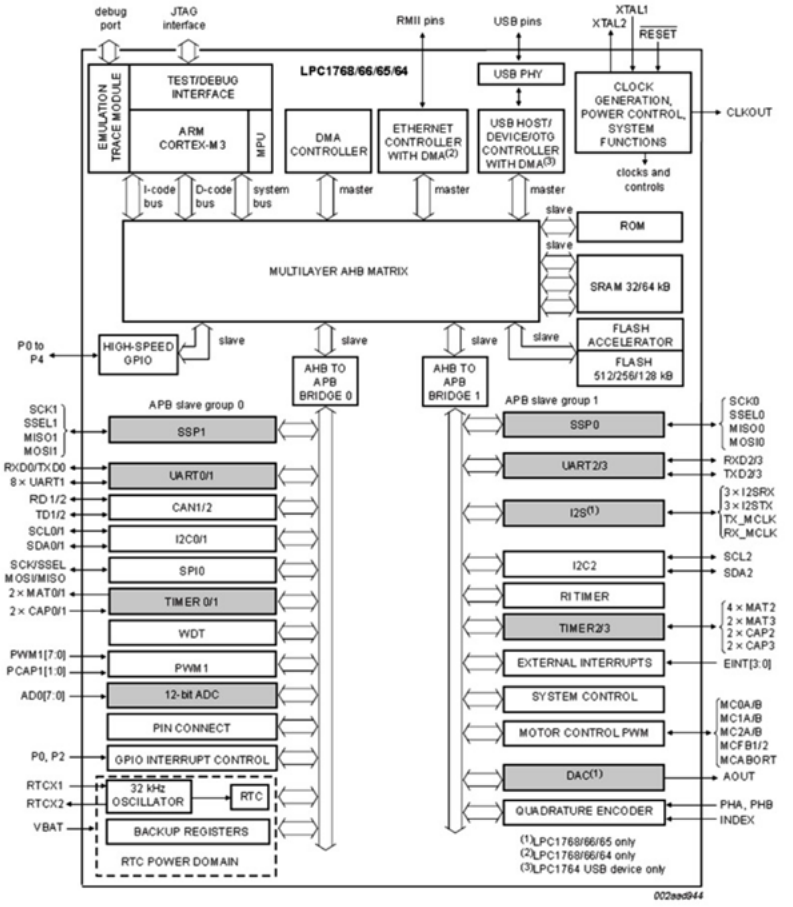
- Eight channel General Purpose DMA controller (GPDMA) on the AHB multilayer matrix that can be used with SSP, I2S-bus, UART, Analog-to-Digital and Digital-to-Analog converter peripherals, timer match signals, and for memory-to-memory transfers.

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

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

Serial interfaces:

- Ethernet MAC with RMII interface and dedicated DMA controller.
- USB 2.0 full-speed device/Host/OTG controller with dedicated DMA controller and on-chip PHY for device, Host, and OTG functions.
- Four UARTs with fractional baud rate generation, internal FIFO, and DMA support.
- One UART has modem control I/O and RS-485/EIA-485 support, and one UART has IrDA support.
- CAN 2.0B controller with two channels.
- SPI controller with synchronous, serial, full duplex communication and programmable data length.
- Two SSP controllers with FIFO and multi-protocol capabilities. The SSP interfaces can be used with the GPDMA controller.



Grey-shaded blocks represent peripherals with connection to the GPDMA.

Fig 1. Block diagram

- Three enhanced I2C bus interfaces, one with an open-drain output supporting full I2C specification and Fast mode plus with data rates of 1 Mbit/s, two with standard port pins. Enhancements include multiple address recognition and monitor mode.
- I2S (Inter-IC Sound) interface for digital audio input or output, with fractional rate control. The I2S-bus interface can be used with the GPDMA.

- The I2S-bus interface supports 3-wire and 4-wire data transmit and receive as well as master clock input/output.

Other peripherals:

- 70 (100 pin package) General Purpose I/O (GPIO) pins with configurable pull-up/down resistors. All GPIOs support a new, configurable open-drain operating mode. The GPIO block is accessed through the AHB multilayer bus for fast access and located in memory such that it supports Cortex-M3 bit banding and use by the General Purpose DMA Controller.
- 12-bit Analog-to-Digital Converter (ADC) with input multiplexing among eight pins, conversion rates up to 200 kHz, and multiple result registers. The 12-bit ADC can be used with the GPDMA controller.
- 10-bit Digital-to-Analog Converter (DAC) with dedicated conversion timer and DMA support.
- Four general purpose timers/counters, with a total of eight capture inputs and ten compare outputs. Each timer block has an external count input. Specific timer events can be selected to generate DMA requests.
- One motor control PWM with support for three-phase motor control
- Quadrature encoder interface that can monitor one external quadrature encoder.
- One standard PWM/timer block with external count input.
- RTC with a separate power domain and dedicated RTC oscillator. The RTC block includes 20 bytes of battery-powered backup registers.
- Watch Dog Timer (WDT). The WDT can be clocked from the internal RC oscillator, the RTC oscillator, or the APB clock.
- ARM Cortex-M3 system tick timer, including an external clock input option.
- Repetitive interrupt timer provides programmable and repeating timed interrupts.
- Each peripheral has its own clock divider for further power savings.
- Standard JTAG test/debug interface for compatibility with existing tools. Serial Wire Debug and Serial Wire Trace Port options.
- Emulation trace module enables non-intrusive, high-speed real-time tracing of instruction execution.
- Integrated PMU (Power Management Unit) automatically adjusts internal regulators to minimize power consumption during Sleep, Deep sleep, Power-down, and Deep power- down modes.
- Four reduced power modes: Sleep, Deep-sleep, Power-down, and Deep power-down.
- Single 3.3 V power supply (2.4 V to 3.6 V).
- Four external interrupt inputs configurable as edge/level sensitive. All pins on Port 0 and Port 2 can be used as edge sensitive interrupt sources.
- Non-mask able Interrupt (NMI) input.
- Clock output function that can reflect the main oscillator clock, IRC clock, RTC clock, CPU clock, and

the USB clock.

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

General Description:

VTCM3_2 provides a hardware platform for developing embedded system using LPCLPC-1768 device. These features make VTCM3_3board ideal for instrumentation, communication and other demanding application areas where flexibility and in-circuit hardware upgradeability is of paramount important.VTCM3_3board comes with complete drivers for windows98/Me/2000/XP/2007.LPC- LPC-1768 code examples and windows DLL interface which can be used to interface it to most common programming language.

VTCM3_3board consists of the following:

- VTCM3_3board fitted in soft wooden box.
- USB Cable for Hex File downloading and 5v dc power for VTCM3_3 board .
- Inbuilt ADC, DAC, RTC
- RESET, ISP Program buttons.

- Eight digital input and Eight Digital output.
- 16X2 LCD Display.
- Four digit seven segment display.
- 20 pin box Header for JATAG debugging.
- I/O pin made available through 20pin, 26pin and 30 pin Box headers.
- Technical/Operational Manual with hardware Schematics and Sample Programs.
- Software CD.

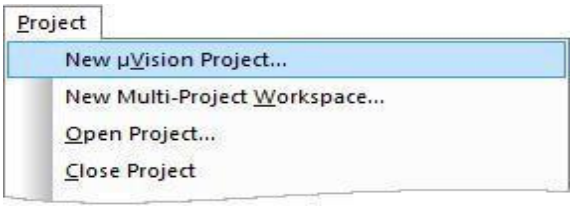
2. Create application using KEIL µVision5

Create Application Using µVision5.

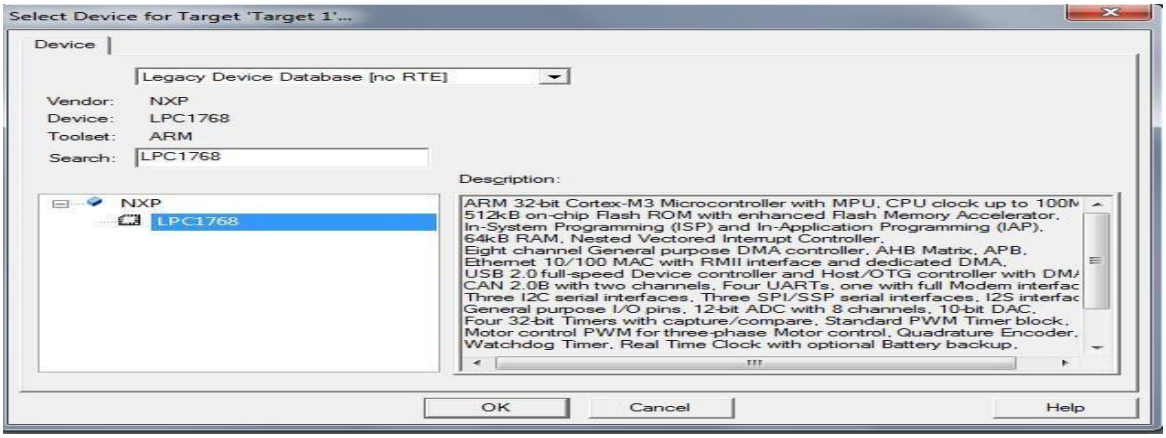
The required steps for creating application programs are listed below:

1. Select **Project - New Project** from the µVision5 menu.

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

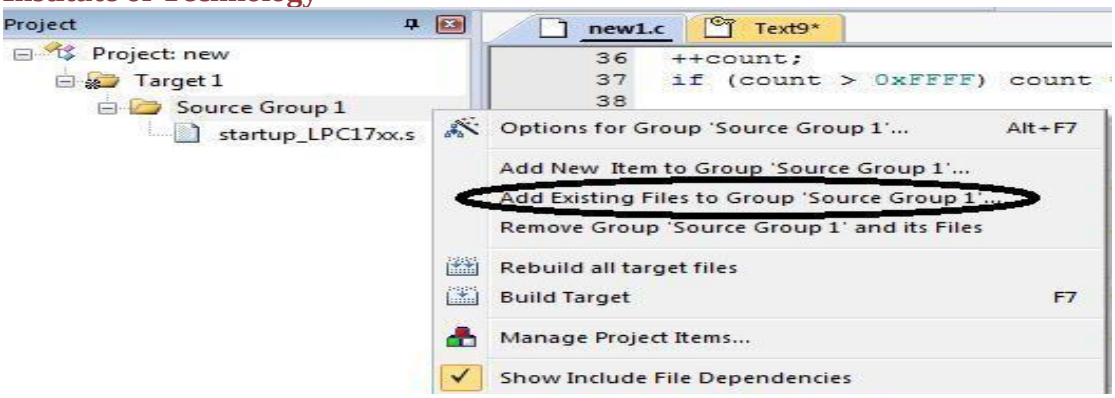


2. Create a new folder with a name **SMVITM** and while selecting target Select the device LPC1768 and click OK as shown below and press yes for LPC17xx.s file option



→ → → →

3. Go to the file New Now edit the program save file_name.c (given extension „.c")
4. Add your file_name.c : right click on source group1 as shown below

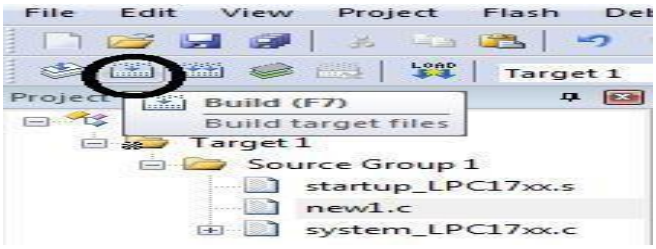


5. Also follow above 4 step to Include ***system_LPC17xx.c*** by browsing the link as shown below

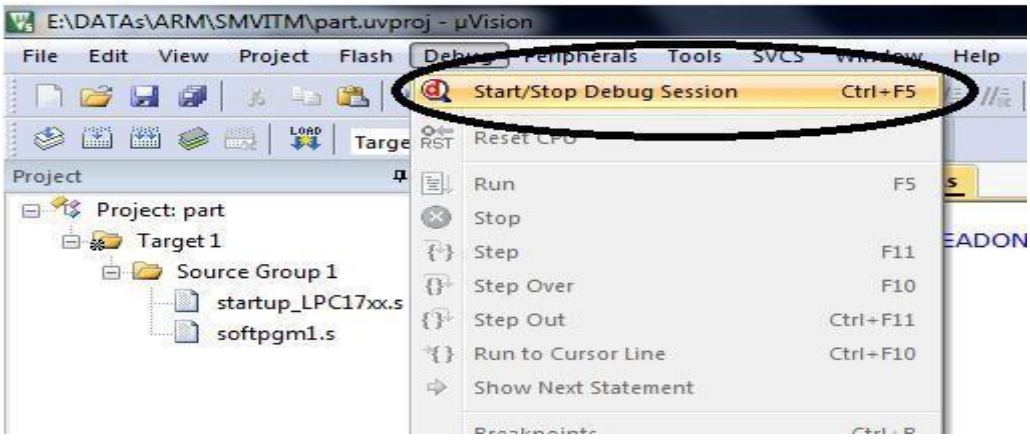
   C:\Keil_v5\ARM\Boards\Keil\MCB1700\Blinky

6. To interface LCD, make sure that in your folder keep a copy of **'LCD.h'** header file by browsing the link **C:\Keil_v5\ARM\Segger\emWin\Include**

7. Save the program and build the target as shown below

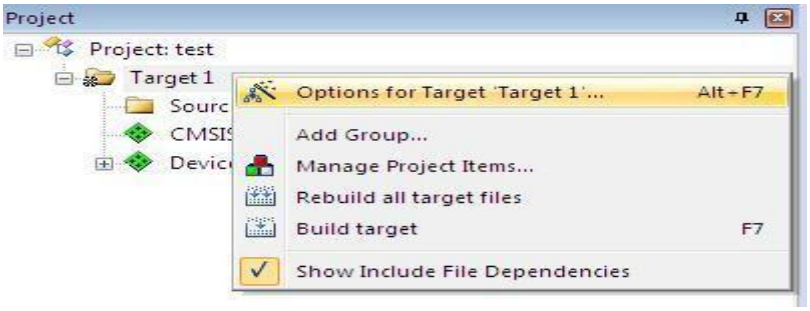


8. Errors or warnings are displayed in the **Build Output Window**. Double-click on a message to jump to the line where the incident occurred.
9. For Part-A program execution use Debug option (Exclude step 10)



- For single step execution press f11 and observe the output

10. If no error, then follow as shown below for PART-B execution (Exclude step 9)



Now the Option for Target „Target1" window will open.

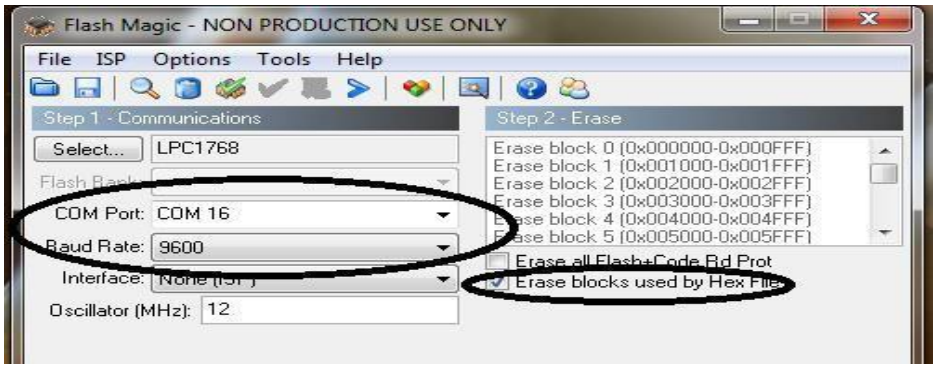
a.Select the **Xtal (MHz):12.0** in **Target** Menu

b. Select **Create HEX File** Check box in **Output** menu

3. Downloading Hex File to LPC 1768 (ONLY FOR PART-B)

Click on the Flash Magic Icon installed over desktop

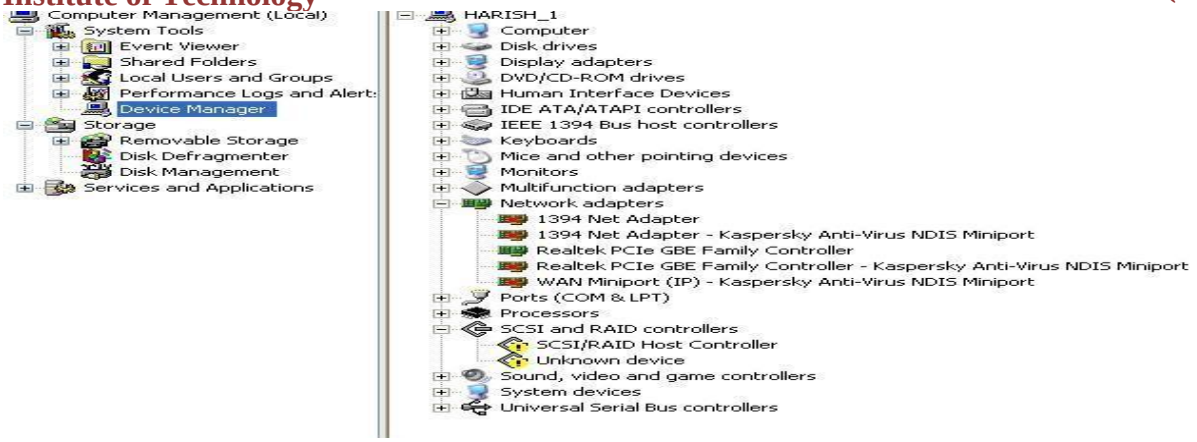
Select the COM Port Name; follow these Steps for knowing the PORT Name



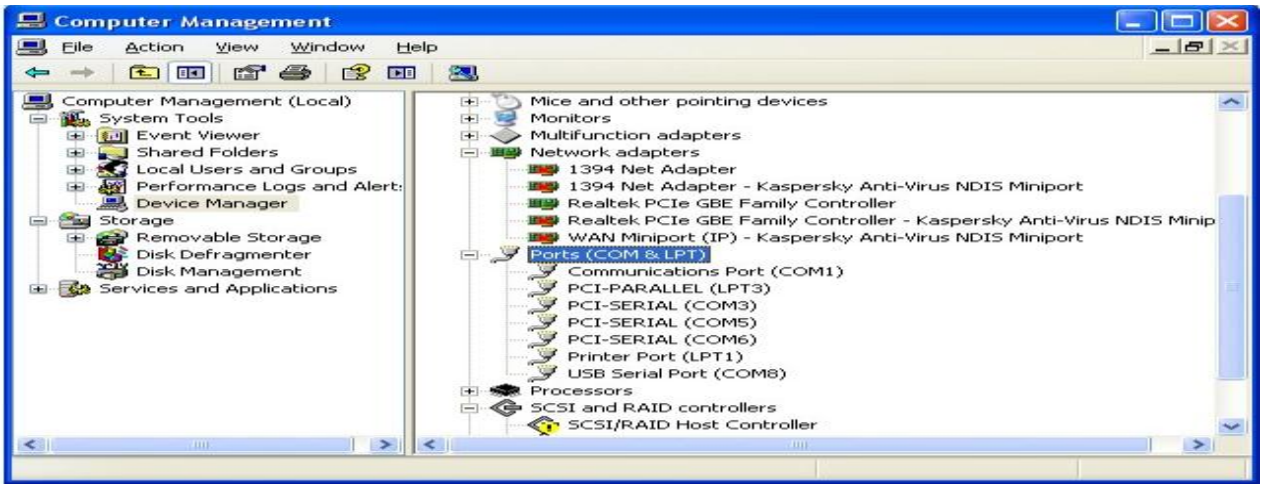
a. Insert one end of the USB cable to the USB Port and Other end to kit.

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

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



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



Once the PORT name is known Select from **COM Port** dropdown, Set the **Baud Rate** and **Oscillator (MHz)** to **9600** and **12.0** respectively & check the **Erase blocks** Used by Hex file.

Select the program from the desktop machine on click of Browse button

Once the program is selected click on the **Start Button** for downloading the code. Program is downloaded through USB Port & check for the desired Output.

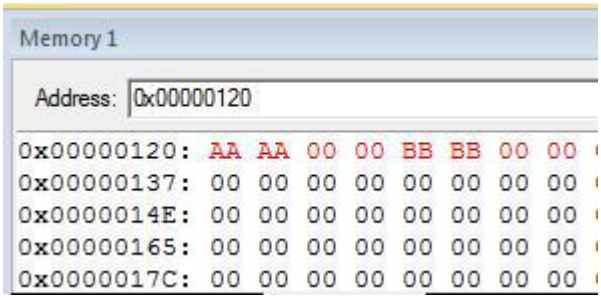
Now Press the **Reset Button** on VTCM3_B_1 kit and check for the desired output.

EXPERIMENT 1: ALP to multiply two 16 bit binary numbers.

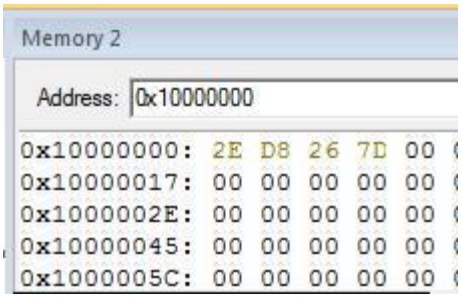
Aim of the Experiment: ALP to multiply two 16 bit binary numbers.

```
        AREA armEx, CODE, READONLY
        EXPORT load
load
        LDR r8, =SOURCE
        LDR r0, [r8]
        LDR r1, [r8, #4]
        mul r7,r0,r1

store
        LDR r8, =dest
        STR r7, [r8]

        AREA destination, DATA, READWRITE
dest
        SPACE 32

        AREA blockData, CODE, READONLY

SOURCE                          ; sorce data
        DCD 0x0000AAAA, 0x0000BBBB

End
```

Before Execution:



After Execution:

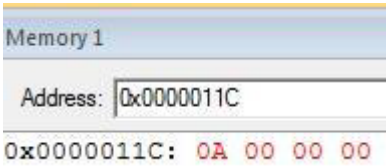


EXPERIMENT 2: ALP to find the sum of first 10 integer numbers.

AIM: ALP to find the sum of first 10 integer numbers.

```
        AREA armEx, CODE, READONLY
        EXPORT load
load
        LDR r8, =SOURCE
        LDR r0, [r8]
        mov r1,#0x00
loop
        ADD r1,r0
        sub r0,#0x01
        cmp r0,#0x00
        BNE loop
stop
        B stop

        AREA destination, DATA, READWRITE
dest
        SPACE 32
        AREA blockData, CODE, READONLY
SOURCE                          ; sorce data
        DCD 0xA
End
```

Before Execution:



After Execution:

PART-B

EXPERIMENT 1: Display "Hello World" message using Internal UART.

AIM: Display "Hello World" message using Internal UART.

Steps for Configuring UART0

Below are the steps for configuring the UART0.

1. Step1: Configure the GPIO pin for UART0 function using PINSEL register.
2. Step2: Configure the FCR for enabling the FIXO and Reste both the Rx/Tx FIFO.
3. Step3: Configure LCR for 8-data bits, 1 Stop bit, Disable Parity and Enable DLAB.
4. Step4: Get the PCLK from PCLKSELx register 7-6 bits.
5. Step5: Calculate the DLM,DLL vaues for required baudrate from PCLK.
6. Step6: Updtae the DLM,DLL with the calculated values.
7. Step6: Finally clear DLAB to disable the access to DLM,DLL.

After this the UART will be ready to Transmit/Receive Data at the specified baudrate.

Baudrate calculation

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

To obtain Baud rate 9600

PCLK= 24MHZ DLM=0

DLL=156 (in decimal) or 9C (in Hexa decimal)

Register	Description
RBR	Contains the recently received Data
THR	Contains the data to be transmitted
FCR	FIFO Control Register
LCR	Controls the UART frame formatting(Number of Data Bits, Stop bits)
DLL	Least Significant Byte of the UART baud rate generator value.
DLM	Most Significant Byte of the UART baud rate generator value.

```
#include<LPC17XX.H>
#define FOSC 12000000
#define FCCLK (FOSC*8)
#define FCCO (FCCLK*3)
#define FPCLK (FCCLK/4)
#define UART0_BPS 9600
int UART0_SendByte (int UCData)
{
  while(!(LPC_UART0->LSR&0x20));
            return(LPC_UART0->THR = UCData);
}
void UART0_Sendstring(unsigned char *s)
{
  while(*s!=0)
            UART0_SendByte(*s++);
}
int main(void)
{
  unsigned int rep;
            unsigned int USFdiv;
  LPC_PINCON->PINSEL0|=(1<<4);
  LPC_PINCON->PINSEL0|=(1<<6);
  LPC_UART0->LCR=0X83;
  USFdiv=(FPCLK/16)/UART0_BPS;
  LPC_UART0->DLM=USFdiv/256;
  LPC_UART0->DLL=USFdiv%256;
  LPC_UART0->LCR=0X03;
  LPC_UART0->FCR=0X06;
  UART0_Sendstring("HELLO WORLD");
  UART0_SendByte(0X0D);
  UART0_SendByte(0X0A);
  while(1)
```

```
        {
                while(!(LPC_UART0->LSR&0x01));
                        rep=LPC_UART0->RBR;
                while(!(LPC_UART0->LSR&0x20));
                        LPC_UART0->THR=rep;
        }
}
```

EXPERIMENT 2: Interface and control a DC motor.

AIM: Interface and control a DC motor.

DC Motor Pin Configuration

DC (direct current) motor rotates continuously. It has two terminals positive and negative. Connecting DC power supply to these terminals rotates motor in one direction and reversing the polarity of the power supply reverses the direction of rotation.

The speed of Dc motor can maintained at a constant speed for a given load by using "Pulse Width Modulation (PWM)" technique. By changing the width of the pulse of applied to dc motor, the power applied is varied thereby DC motor speed can be increased or decreased. Wider the pulse Faster is the Speed, Narrower is the Pulse, and Slower is the Speed

U9 is L293 driver IC used to drive the dc motor. It has enable lines which is used to switch on the DC motor. It is connected to P4.28. Logic „1" enables the driver and logic „0" disables the driver.P4.28 and P4.29 are used for Motor 1 direction and speed control.

DC Motor Pin connection table.

Motor Selection	Motor Direction	LPC-1768 Pin No	LPC-1768 Port No
Motor	DCM0-Clk	8 2	P4.28=1 & P4.29=0
	DCM1-Aclk	8 5	P4.28=0 & P4.29=1
	DCM_EN	6 5	P2.8=1

```
// clk wise
#include<LPC17XX.H>
int main(void)
{
    LPC_GPIO4->FIODIR=0x30000000;
    while(1)
    {
LPC_GPIO4->FIOPIN=1<<29;
LPC_GPIO2->FIOPIN=0x00000100;
```

```
    }
}

// counter clk wise
#include<LPC17XX.H>
int main(void)
{
    LPC_GPIO4->FIODIR=0x30000000;
    while(1)
    {
LPC_GPIO4->FIOPIN=1<<28;
LPC_GPIO2->FIOPIN=0x00000100;
    }
}




// both

#include<LPC17xx.H>
void delay (unsigned int count)
{
unsigned int j=0,i=0;
for(j=0;j<count;j++)
{
for(i=0;i<12000;i++);
}
}
int main(void)
{
unsigned int del=200;
delay(1000);
```

```
LPC_GPIO4->FIODIR=0x30000000;
while(1)
{
LPC_GPIO4->FIOPIN=1<<29;
LPC_GPIO2->FIOPIN=0x00000100;
delay(del);
LPC_GPIO4->FIOPIN=1<<28;
LPC_GPIO2->FIOPIN=0x00000100;
delay(del);
}
}
```

```
// clk wise using key
#include<LPC17XX.H>
#define KEY1 (1<<14)
#define KEY_PIN LPC_GPIO->FIOPIN
int main(void)
{
    LPC_GPIO4->FIODIR=0x30000000;
    while(1)
    {
if(!(KEY_PIN & KEY1))
{
LPC_GPIO4->FIOPIN=1<<29;
LPC_GPIO2->FIOPIN=0x00000100;
    }
}
}
```

```
// counter clk wise using keys
#include<LPC17XX.H>
#define KEY1 (1<<14)
```

```
#define KEY_PIN LPC_GPIO->FIOPIN
int main(void)
{
    LPC_GPIO4->FIODIR=0x30000000;
    while(1)
    {
if(!(KEY_PIN & KEY1))
{
LPC_GPIO4->FIOPIN=1<<28;
LPC_GPIO2->FIOPIN=0x00000100;
    }
}
}

// using clk and aclk wise using keys
#include<LPC17XX.H>
#define KEY1 (1<<14)
#define KEY2 (1<<15)
#define KEY_PIN LPC_GPIO->FIOPIN
int main(void)
{
    LPC_GPIO4->FIODIR=0x30000000;
    while(1)
    {
if(!(KEY_PIN & KEY1))
{
LPC_GPIO4->FIOPIN=1<<29;
LPC_GPIO2->FIOPIN=0x00000100;
    }
if(!(KEY_PIN & KEY2))
{
LPC_GPIO4->FIOPIN=1<<28;
LPC_GPIO2->FIOPIN=0x00000100;
```

```
        }
    }
    }
```

EXPERIMENT 3: Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.

AIM: Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.

Stepper Motor Configuration:

A ***stepper motor*** is a special type of electric motor that moves in increments, or steps, rather than turning smoothly as a conventional motor does. Typical increments are 0.9 or 1.8 degrees, with 400 or 200 increments thus representing a full circle. The speed of the motor is determined by the time delay between each incremental movement.

U8 is a Driver Buffer (ULN2003) *device connected to LPC-1768 Device and can be used for driving Stepper Motor*. On the LPC-1768, P1.22 to P1.25 is used to generate the pulse sequence required to run the stepper Motor. Also, the Stepper Motor is powered by its own power supply pin (COM), which is connected to a 12V supply. Table shows connections for stepper Motor.

Stepper Motor Pin connection table.

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

```
//clockwise
#include <LPC17XX.H>
#define MOTOR_CTRL_DIR LPC_GPIO1
void delay()
```

```
{
unsigned int j=0,i=0;
for(j=0;j<20;j++)
{
for(i=0;i<12000;i++);
}
}
int main(void)
{
LPC_GPIO1->FIODIR=0x03C00000;
while(1)
{
LPC_GPIO1->FIOPIN=1<<25;
delay();
LPC_GPIO1->FIOPIN=1<<24;
delay();
LPC_GPIO1->FIOPIN=1<<23;
delay();
LPC_GPIO1->FIOPIN=1<<22;
delay();
}
}

//anti clock wise
#include <LPC17XX.H>
#define MOTOR_CTRL_DIR LPC_GPIO1
void delay()
{
unsigned int j=0,i=0;
for(j=0;j<20;j++)
{
for(i=0;i<12000;i++);
}
```

```
}
int main(void)
{
LPC_GPIO1->FIODIR=0x03C00000;
while(1)
{
LPC_GPIO1->FIOPIN=1<<22;
delay();
LPC_GPIO1->FIOPIN=1<<23;
delay();
LPC_GPIO1->FIOPIN=1<<24;
delay();
LPC_GPIO1->FIOPIN=1<<25;
delay();
}
}
```

```
// Both clk and aclk
#include <LPC17XX.H>
#define MOTOR_CTRL_DIR LPC_GPIO1
void delay()
{
unsigned int j=0,i=0;
for(j=0;j<20;j++)
{
for(i=0;i<12000;i++);
}
}
int main(void)
{
int k=0;
unsigned int count=0;
```

```
LPC_GPIO1->FIODIR=0x03C00000;
while(count<=100)
{
for(k=0;k<50;k++)
{
LPC_GPIO1->FIOPIN=1<<25;
delay();
LPC_GPIO1->FIOPIN=1<<24;
delay();
LPC_GPIO1->FIOPIN=1<<23;
delay();
LPC_GPIO1->FIOPIN=1<<22;
delay();
count=count+4;
}
for(k=0;k<50;k++)
{
LPC_GPIO1->FIOPIN=1<<22;
delay();
LPC_GPIO1->FIOPIN=1<<23;
delay();
LPC_GPIO1->FIOPIN=1<<24;
delay();
LPC_GPIO1->FIOPIN=1<<25;
delay();
count=count+4;
}
}
}

// rotate 90
#include<LPC17XX.H>
#define MOTOR_CTRL_DIR
```

```
void delay()
{
    unsigned int j=0,i=0;
    for(j=0;j<20;j++)
    {
            for(i=0;i<12000;i++);
    }
}
int main(void)
{
    unsigned int count=0;
    LPC_GPIO1->FIODIR=0X03C00000;
    while(count<=50)
    {
            LPC_GPIO1->FIOPIN=1<<25;
            delay();
            LPC_GPIO1->FIOPIN=1<<24;
            delay();
            LPC_GPIO1->FIOPIN=1<<23;
            delay();
            LPC_GPIO1->FIOPIN=1<<22;
            delay();
            count=count+4;
    }
    }


// rotate 180
#include<LPC17XX.H>
#define MOTOR_CTRL_DIR
void delay()
{
    unsigned int j=0,i=0;
    for(j=0;j<20;j++)
```

```
    {
        for(i=0;i<12000;i++);
    }
}
int main(void)
{
    unsigned int count=0;
    LPC_GPIO1->FIODIR=0X03C00000;
    while(count<=100)
    {
        LPC_GPIO1->FIOPIN=1<<25;
        delay();
        LPC_GPIO1->FIOPIN=1<<24;
        delay();
        LPC_GPIO1->FIOPIN=1<<23;
        delay();
        LPC_GPIO1->FIOPIN=1<<22;
        delay();
        count=count+4;
    }
    }


// rotate 360
#include<LPC17XX.H>
#define MOTOR_CTRL_DIR
void delay()
{
    unsigned int j=0,i=0;
    for(j=0;j<20;j++)
    {
        for(i=0;i<12000;i++);
    }
```

```
}
int main(void)
{

    unsigned int count=0;
    LPC_GPIO1->FIODIR=0X03C00000;
    while(count<=200)
    {

    {
            LPC_GPIO1->FIOPIN=1<<25;
            delay();
            LPC_GPIO1->FIOPIN=1<<24;
            delay();
            LPC_GPIO1->FIOPIN=1<<23;
            delay();
            LPC_GPIO1->FIOPIN=1<<22;
            delay();
            count=count+4;
            }
            }
    }


// rotate both
#include <LPC17XX.H>
#define MOTOR_CTRL_DIR LPC_GPIO1
void delay()
{
unsigned int j=0,i=0;
for(j=0;j<20;j++)
{
for(i=0;i<12000;i++);
```

```
}
}
int main(void)
{
int k=0;
unsigned int count=0;
LPC_GPIO1->FIODIR=0x03C00000;
while(count<=100)
{
for(k=0;k<50;k++)
{
LPC_GPIO1->FIOPIN=1<<25;
delay();
LPC_GPIO1->FIOPIN=1<<24;
delay();
LPC_GPIO1->FIOPIN=1<<23;
delay();
LPC_GPIO1->FIOPIN=1<<22;
delay();
count=count+4;
}
for(k=0;k<50;k++)
{
LPC_GPIO1->FIOPIN=1<<22;
delay();
LPC_GPIO1->FIOPIN=1<<23;
delay();
LPC_GPIO1->FIOPIN=1<<24;
delay();
LPC_GPIO1->FIOPIN=1<<25;
delay();
count=count+4;
}
```

```
}
}

// rotate both using keys
#include <LPC17XX.H>
#define MOTOR_CTRL_DIR LPC_GPIO1
#define KEY1 (1<<14)
#define KEY2 (1<<15)
#define KEY_PIN LPC_GPIO1->FIOPIN
void delay()
{
unsigned int j=0,i=0;
for(j=0;j<20;j++)
{
for(i=0;i<12000;i++);
}
}
int main(void)
{
LPC_GPIO1->FIODIR=0x03C00000;
while(1)
{
if(!(KEY_PIN & KEY1))
{
LPC_GPIO1->FIOPIN=1<<25;
delay();
LPC_GPIO1->FIOPIN=1<<24;
delay();
LPC_GPIO1->FIOPIN=1<<23;
delay();
LPC_GPIO1->FIOPIN=1<<22;
delay();
}
```

```
if(!(KEY_PIN & KEY2))
{
LPC_GPIO1->FIOPIN=1<<22;
delay();
LPC_GPIO1->FIOPIN=1<<23;
delay();
LPC_GPIO1->FIOPIN=1<<24;
delay();
LPC_GPIO1->FIOPIN=1<<25;
delay();
}
}
}
```

```
// rotate both using switches
#include <LPC17XX.H>
#define MOTOR_CTRL_DIR LPC_GPIO1->FIODIR
#define MOTOR_CTRL_SET LPC_GPIO1->FIOSET
#define MOTOR_CTRL_CLR LPC_GPIO1->FIOCLR
#define MOTOR_MASK 0X03C00000
#define KEY_DIR LPC_GPIO1->FIODIR
#define KEY_SET LPC_GPIO1->FIOSET
#define KEY_CLR LPC_GPIO1->FIOCLR
#define KEY_PIN LPC_GPIO1->FIOPIN
#define KEY_START (1<<14)
#define KEY_STOP (1<<15)
#define KEY_INC (1<<16)
#define KEY_DEC (1<<17)
#define KEY_CW (1<<18)
#define KEY_CCW (1<<19)
void delay()
{
```

```
    unsigned int j=0,i=0;
    for(j=0;j<10;j++)
    {
        for (i=0;i<12000;i++);
    }
}
void motor_write(uint32_t data)
{
    uint32_t temp;
    temp=(data<<22)& MOTOR_MASK;
 MOTOR_CTRL_CLR|= MOTOR_MASK;
 MOTOR_CTRL_SET|= temp;
}
int main(void)
{
    unsigned int del=10;
    uint32_t stpval=0x01;
    unsigned char dir=0;
    unsigned char run=1;
    MOTOR_CTRL_DIR |= MOTOR_MASK;
    motor_write (stpval);
    while(1)
    {
        if(!(KEY_PIN&KEY_START))
            run=1;
        if(!(KEY_PIN&KEY_STOP))
            run=0;
        if(!(KEY_PIN&KEY_CW))
            dir=0;
        if(!(KEY_PIN&KEY_CCW))
            dir=1;
        if(!(KEY_PIN&KEY_INC))
        if(del!=10) del=del-1;
```

```
                if(run==1)
                {
                        if(dir==0)
                        {
                                if(stpval==8)
                                        stpval=1;
                                else
                                        stpval<<=1;
                        }
                        else
                        {
                                if(stpval==1)
                                        stpval=8;
                                else
                                        stpval>>=1;
                        }
                        motor_write(stpval);
                        delay(del);
                }
        }

}
```

EXPERIMENT 4: Interface a DAC and generate Triangular and Square waveforms.

AIM: Interface a DAC and generate Triangular and Square waveforms.

Digital to Analog Converter [DAC]:

Features

• 10-bit digital to analog converter

• Resistor string architecture

• Buffered output

• Selectable speed vs. power

• Maximum update rate of 1 MHZ.

As VTCM3_3 board comes with one DAC output for generation different wave forms. AOUT (P0.26) is connected to TEST point TP1. The generated waveforms can be viewed through TP1 (DAC) and TP2 (GND) by connecting CRO.

```
// Square waveform
#include<LPC17XX.H>
int main (void)
{
uint32_t m;
LPC_PINCON->PINSEL1=0x00200000;
while(1)
{
LPC_DAC->DACR=(512<<6);
for(m=120000; m>1; m--);
LPC_DAC->DACR=(1023<<6);
for(m=120000;m>1;m--);
}
```

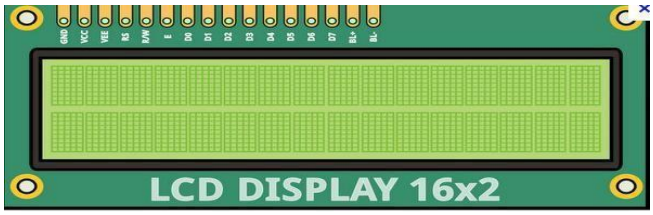
```
}

// Triangle waveform
#include <LPC17XX.H>
int main(void)
{
uint32_t i=0;
uint32_t m;
LPC_PINCON->PINSEL1=0X00200000;
while(1)
{
for (i=0;i<1023;i++)
{
LPC_DAC -> DACR=(i<<6);
for(m=10;m>1;m--);
}
for(i=1023;i>0;i--)
{
LPC_DAC -> DACR =(i<<6);
for(m=10;m>1;m--);
}
}
}
```

EXPERIMENT 5: Interface a 4x4 keyboard and display the key code on an LCD.

AIM: Interface a 4x4 keyboard and display the key code on an LCD.

16 X 2 LCD Display

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



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

LCD Pin Configuration

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

HEX KEY PAD Pin configurations:

The hex keypad is a peripheral that is organized in rows and Columns. Hex key Pad 16 Keys arranged in a 4 by 4 grid, labeled with the hexadecimal digits 0 to F. An example of this can been seen in Figure 1, below. Internally, the structure of the hex keypad is very simple. Wires run in vertical columns (we call them C0 to C3) and in horizontal rows (called R0 to R3). These 8 wires are available externally, and will be connected to the lower 8 bits of the port. Each key on the keypad is essentially a switch that connects a row wire to a column wire. When a key is pressed, it makes an electrical connection between the row and column. Table shows connections for HEX KEY Pad matrix.

HEX KEY Pad matrix table.

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

USER KEY PAD Pin configurations:

VT1768 3_3board provides eight individual KEYS connected to LPC-1768 device through PORT1. S1 to S6,S11and S16 are connected to general purpose I/O pins on 2148 device as shown in table. 2148 device port pin drives to Logic „0" when the corresponding key is pressed. Table shows connections for USER KEY Pad connection.

USER KEY Pad connection table.

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

```
// LCD
#include<LPC17XX.H>
#include "lcd.h"
#define COL1 (1<<14)
#define COL2 (1<<15)
#define COL3 (1<<16)
#define COL4 (1<<17)
#define ROW1 (1<<18)
#define ROW2 (1<<19)
#define ROW3 (1<<20)
#define ROW4 (1<<21)
#define COLMASK (COL1|COL2|COL3|COL4)
#define ROWMASK (ROW1|ROW2|ROW3|ROW4)
```

```
#define KEY_DIR LPC_GPIO1->FIODIR
#define KEY_SET LPC_GPIO1->FIOSET
#define KEY_CLR LPC_GPIO1->FIOCLR
#define KEY_PIN LPC_GPIO1->FIOPIN
void col_write(unsigned char data)
{
unsigned int temp=0;
temp=(data<<14) & COLMASK;
KEY_CLR|=COLMASK;
KEY_SET|=temp;
}
int main(void)
{
unsigned char key,i;
unsigned char rval[]={0X7,0XB,0XD,0XE,0X0};
unsigned char KeyPadMatrix[]=
{
0xC,0x5,0x4,0x0,
0xD,0x9,0x5,0x1,
0xE,0xA,0x6,0x2,
0xF,0xB,0x7,0x3
};
init_lcd();
LPC_GPIO1->FIODIR|= (COL1|COL2|COL3|COL4);
KEYDIR &= ~(ROWMASK);
lcd_putstring 16(0,"press HEX_keys..");
lcd_putstring 16(1,"key_pressed= ");
LPC_GPIO3->FIODIR=0X02000000;
while(1)
{
LPC_GPIO3->FIOCLR=0X02000000;
key=0;
for(i=0;i<4;i++)
```

```
{
col_write (rval[i]);
if(!(KEY_PIN & ROW1))
break;
key++;
if(!(KEY_PIN & ROW2))
break;
key++;
if(!(KEY_PIN & ROW3))
break;
key++;
if(!(KEY_PIN & ROW4))
break;
key++;
}
if(key==0X10)
lcd_putstring 16(1,"key_pressed= ");
else
{
lcd_gotoxy(1,14);
lcd_putchar(KeyPadMatrix[key]);
}
}
}
```

```
// 7 Segment Display
#include<LPC17XX.H>
#define dig1 (1<<26)
unsigned char
data7[]={0x88,0xEB,0x4C,0x49,0x2B,0x19,0x18,0xCB,0x8,0x9,0xA,0x38,0x9C,0x68,0x1C,0x1E};
#define COL1 (1<<14)
#define COL2 (1<<15)
```

```
#define COL3 (1<<16)
#define COL4 (1<<17)
#define ROW1 (1<<18)
#define ROW2 (1<<19)
#define ROW3 (1<<20)
#define ROW4 (1<<21)
#define COLMASK (COL1|COL2|COL3|COL4)
#define ROWMASK (ROW1|ROW2|ROW3|ROW4)
#define KEY_PIN LPC_GPIO1->FIOPIN
void col_write(unsigned char data)
{
unsigned int temp=0;
temp=(data<<14) & COLMASK;
LPC_GPIO1->FIOCLR|=COLMASK;
LPC_GPIO1->FIOSET|=temp;
}
int main(void)
{
unsigned char key,i;
unsigned char rval[]={0X7,0XB,0XD,0XE,0X0};
unsigned char KeyPadMatrix[]=
{
0xC,0x5,0x4,0x0,
0xD,0x9,0x5,0x1,
0xE,0xA,0x6,0x2,
0xF,0xB,0x7,0x3
};
LPC_GPIO1->FIODIR|=COLMASK;
LPC_GPIO1->FIODIR &= ~(ROWMASK);
LPC_GPIO2->FIODIR=0X000000FF;
LPC_GPIO2->FIOPIN=0X000000FF;
LPC_GPIO1->FIODIR|=0X3C000000;
LPC_GPIO1->FIOPIN &= ~(0X3C000000);
```

```
LPC_GPIO1->FIOSET=dig1;
while(1)
{
key=0;
for(i=0;i<4;i++)
{
col_write (rval[i]);
if(!(KEY_PIN & ROW1))
break;
key++;
if(!(KEY_PIN & ROW2))
break;
key++;
if(!(KEY_PIN & ROW3))
break;
key++;
if(!(KEY_PIN & ROW4))
break;
key++;
}
if(key==0X10)
LPC_GPIO2->FIOPIN=0XFF;
else
{
LPC_GPIO2->FIOPIN=data7[KeyPadMatrix[key]];
}
}
}
```

EXPERIMENT 6: Using the Internal PWM module of ARM controller generate PWM and vary its duty cycle

AIM: Using the Internal PWM module of ARM controller generate PWM and vary its duty cycle.

Pulse Width Modulation [PWM]:

Features

• Counter or Timer operation (may use the peripheral clock or one of the capture inputs as the clock source).

• Seven match registers allow up to 6 single edge controlled or 3 double edge controlled PWM outputs, or a mix of both types. The match registers also allow:

– Continuous operation with optional interrupt generation on match.

– Stop timer on match with optional interrupt generation.

– Reset timer on match with optional interrupt generation.

• Supports single edge controlled and/or double edge controlled PWM outputs. Single edge controlled PWM outputs all go high at the beginning of each cycle unless the output is a constant low. Double edge controlled PWM outputs can have either edge occur at any position within a cycle. This allows for both positive going and negative going pulses.

• Pulse period and width can be any number of timer counts. This allows complete flexibility in the trade-off between resolution and repetition rate. All PWM outputs will occur at the same repetition rate.

• Double edge controlled PWM outputs can be programmed to be either positive going or negative going pulses.

• Match register updates are synchronized with pulse outputs to prevent generation of Erroneous pulses. Software must "release" new match values before they can become effective.

• May be used as a standard timer if the PWM mode is not enabled.

• A 32-bit Timer/Counter with a programmable 32-bit prescaler.

• Two 32-bit capture channels take a snapshot of the timer value when an input signals transitions. A capture event may also optionally generate an interrupt. As VTCM3_3 board comes with one DAC output for generation different wave forms. AOUT (P0.26) is connected to TEST point TP1. The generated waveforms can be viewed through TP1 (DAC) and TP2 (GND) by connecting CRO.

As VTCM3_3 board comes with one PWM output for generation PWM wave forms. AOUT (P0.26) is connected to TEST point TP1. The generated waveforms can be viewed through TP1 (DAC) and TP2 (GND) by connecting CRO.

Steps to Configure PWM

1. Configure the GPIO pins for PWM operation in respective PINSEL register.
2. Configure TCR to enable the Counter for incrementing the TC, and Enable the PWM block.
3. Set the required pre-scalar value in PR. In our case it will be zero.
4. Configure MCR to reset the TC whenever it matches MR0.
5. Update the Cycle time in MR0. In our case it will be 100.
6. Load the Duty cycles for required PWMx channels in respective match registers MRx(x: 1-6).
7. Enable the bits in LER register to load and latch the new match values.
8. Enable the required pwm channels in PCR register.

Register Description

TCR Timer Control Register: The TCR is used to control the Timer Counter functions (enable/disable/reset). **TC Timer Counter**: The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.

PR Prescalar Register: This is used to specify the Prescalar value for incrementing the TC.

MCR: Match Control Register: The MCR is used to control the resetting of TC and generating of interrupt whenever a Match occurs.

MR0 Match Register: This register hold the max cycle Time (Ton+Toff). **MR1-MR6: Match Registers**: These registers hold the Match value (PWM Duty) for corresponding PWM channels (PWM1-PWM6).

PCR: PWM Control Register: PWM Control Register. Enables PWM outputs and selects PWM channel types as either single edge or double edge controlled.

LER Load Enable Register: Enables use of new PWM values once the match occurs. TC is incremented.

```
// PWM 0 to 100

#include<LPC17xx.H>
void delay_ms(unsigned int ms)
{
unsigned int i,j;
for(i=0;i<ms;i++)
for(j=0;j<50000;j++);
}
int main(void)
{
int dutycycle;
LPC_PINCON->PINSEL7=3<<20;
LPC_PWM1->TCR=(1<<0);
LPC_PWM1->PR=0X0;
LPC_PWM1->MCR=(1<<1);
LPC_PWM1->MR0=100;
LPC_PWM1->MR3=0;
LPC_PWM1->LER=1<<3;
LPC_PWM1->PCR=1<<11;
while(1)
{
for(dutycycle=0;dutycycle<100;dutycycle++)
{
LPC_PWM1->MR3=dutycycle;
LPC_PWM1->LER=1<<3;
delay_ms(5);
}
```

```
for(dutycycle=100;dutycycle>0;dutycycle--)
{
LPC_PWM1->MR3=dutycycle;
LPC_PWM1->LER=1<<3;
delay_ms(5);
}
}
}

// 50 to 100

#include<LPC17xx.H>
void delay_ms(unsigned int ms)
{
unsigned int i,j;
for(i=0;i<ms;i++)
for(j=0;j<50000;j++);
}
int main(void)
{
int dutycycle;
LPC_PINCON->PINSEL7=3<<20;
LPC_PWM1->TCR=(1<<0);
LPC_PWM1->PR=0X0;
LPC_PWM1->MCR=(1<<1);
LPC_PWM1->MR0=100;
LPC_PWM1->MR3=50;
LPC_PWM1->LER=1<<3;
LPC_PWM1->PCR=1<<11;
while(1)
{
for(dutycycle=50;dutycycle<100;dutycycle++)
{
```

```
LPC_PWM1->MR3=dutycycle;
LPC_PWM1->LER=1<<3;
delay_ms(5);
}
for(dutycycle=100;dutycycle>50;dutycycle--)
{
LPC_PWM1->MR3=dutycycle;
LPC_PWM1->LER=1<<3;
delay_ms(5);
}
}
}
```

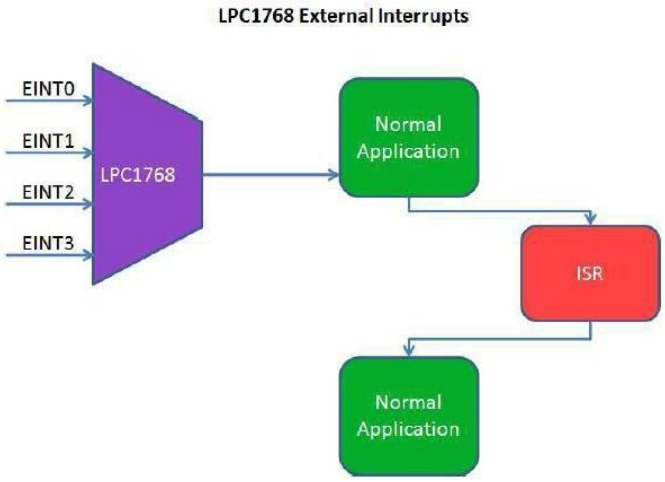
EXPERIMENT 7: Demonstrate the use of an external interrupt

AIM: Demonstrate the use of an external interrupt

VTCM3_3board provides eight individual SMD led‟s connected to LPC-1768 device through 74HC151driver IC. D1 to D8 are connected to general purpose I/O pins on LPC-1768 device as shown in table(14) When LPC-1768 device drives Logic „1‟ the corresponding LED turns on.

LED Pin connection table.

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



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

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

// **Program to demonstrate the External interrupt: When an interrupt occurs INT0, main program halts and wait till a key to be pressed (key P1.14). As soon as key pressed, it resumes the main operation.**

```
#include<LPC17XX.H>
void EINT0_IRQHandler(void)
{
    LPC_SC->EXTINT=(1<<0);
    LPC_GPIO0->FIOPIN^=(1<<4);
}
void EINT1_IRQHandler (void)
{
    LPC_SC->EXTINT=(1<<1);
    LPC_GPIO0->FIOPIN^=(1<<5);
}
int main()
{
    LPC_SC->EXTINT=(1<<0)|(1<<1);
    LPC_PINCON->PINSEL4=(1<<20)|(1<<22);
    LPC_SC->EXTMODE=(1<<0)|(1<<1);
    LPC_SC->EXTPOLAR=(1<<0)|(1<<1);
    LPC_GPIO0->FIODIR=((1<<4)|(1<<5));
    LPC_GPIO0->FIOPIN=0X00;
    NVIC_EnableIRQ(EINT0_IRQn);
    NVIC_EnableIRQ(EINT1_IRQn);
    while(1)
```

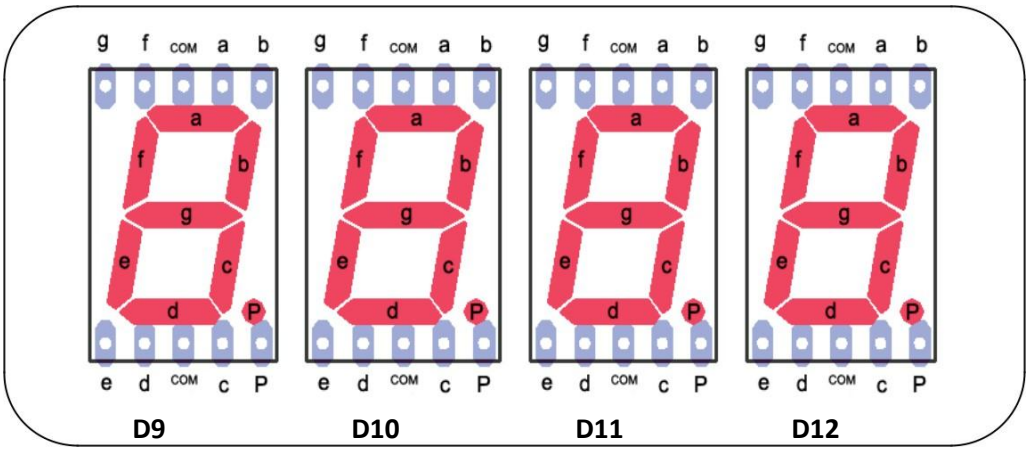
```
        {
        }
    }
```

EXPERIMENT 8: Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between.

AIM: Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between.

Pin Configuration:

D11, D12, D13 and D14 are Common Cathode segments connected to LPC-1768 device so that each segment is individually controlled by a general purpose I/O pin. When the LPC-1768 device drives logic „0" the corresponding segment turns on. See fig and table for more details.



Seven Segment Pin connection table.

Seven Segment Data Lines	g	f	a	b	p	c	d	e
LPC-1768 Pin No	75	74	73	70	69	68	67	66
LPC-1768 Port No	P2.0	P2.1	P2.2	P2.3	P2.4	P2.5	P2.6	P2.7

SEVEN SEGMENT Selection Pin Configurations:

As VTCM3_3 board comes with 4 digit seven segment unit. Displays connected to the microcontroller usually occupy a large number of valuable I/O pins, which can be a big problem especially if it is needed to display multi digit numbers. The problem is more than obvious if, for example, it is

needed to display four digit numbers (a simple calculation shows that 32 output pins are needed in this case). The solution to this problem is called MULTIPLEXING. This is how an optical illusion based on the same operating principle as a film camera is made. Only one digit is active at a time, but they change their state so quickly making impression that all digits of a number are simultaneously active. Each digit can made active using switching transistors Q1, Q2, Q3 and Q4 and these on switched on and off by selection lines which are in turn connected to *LPC-1768* ports. Table shows the details of seven segment selection lines.

Seven Segment Selection table.

Seven Segment Selection Lines	Disp1 (D9)	Disp2 (D10)	Disp3 (D11)	Disp4 (D12)
LPC-1768 Pin No	40	43	44	45
LPC-1768 Port No	P1.26	P1.27	P1.28	P1.29

Display Encoding:

Digits to display	**Display Segments**								**Hex code**
	G	*f*	*a*	*b*	*p*	*c*	*d*	*e*	
0	1	0	0	0	1	0	0	0	88
1	1	1	1	0	1	0	1	1	EB
2	0	1	0	0	1	1	0	0	4C
3	0	1	0	0	1	0	0	1	49
4	0	0	1	0	1	0	1	1	2B
5	0	0	0	1	1	0	0	1	19
6	0	0	0	1	1	1	1	0	18
7	1	1	0	0	1	0	1	1	CB
8	0	0	0	0	1	0	0	0	08
9	0	0	0	0	1	0	0	1	09
A	0	0	0	0	1	0	1	0	0A
B	0	0	1	1	1	0	0	0	38
C	1	0	0	1	1	0	0	0	98
D	0	1	1	0	1	0	0	0	68
E	0	0	0	1	1	1	0	0	1C
F	0	0	0	1	1	1	1	0	1E

```
// 0 to F
#include<LPC17XX.H>
unsigned char
data7[]={0x88,0xeb,0x4c,0x49,0x2b,0x19,0x18,0xcb,0x8,0x9,0xa,0x38,0x9c,0x68,0x1c,0x1e};
int main (void)
{
unsigned int i,j;
unsigned int count=0;
LPC_GPIO2->FIODIR=0x000000FF;
LPC_GPIO2->FIOPIN=0X000000FF;
LPC_GPIO1->FIODIR=1<<26;
LPC_GPIO1->FIOPIN=~(1<<26);
while(1)
{
if(count<0)
count=0;
for(i=0;i<10000;i++)
{
LPC_GPIO2->FIOPIN=data7[count];
LPC_GPIO1->FIOSET=0X04000000;
for(j=0;j<500;j++)
LPC_GPIO1->FIOCLR=0X04000000;
}
++count;
}
}

// BGSIT
#include<LPC17XX.H>
unsigned char data7[]={0x38,0x9,0x19,0xBE,0x3C,0x7F,0x1C,0x9C,0x1C};
```

```
int main (void)
{
unsigned int i,j;
unsigned int count=0;
LPC_GPIO2->FIODIR=0x000000FF;
LPC_GPIO2->FIOPIN=0X000000FF;
LPC_GPIO1->FIODIR=1<<26;
LPC_GPIO1->FIOPIN=~(1<<26);
while(1)
{
if(count<0)
count=0;
for(i=0;i<10000;i++)
{
LPC_GPIO2->FIOPIN=data7[count];
LPC_GPIO1->FIOSET=0X04000000;
for(j=0;j<500;j++)
LPC_GPIO1->FIOCLR=0X04000000;
}
++count;
}
}

// F to 0
#include <LPC17XX.H>
unsigned char
data7[]={0X88,0Xeb,0X4c,0X49,0X2b,0X19,0X18,0Xcb,0X8,0X9,0Xa,0X38,0X9c,0X68,0X1c,0X1e};
int main(void)
{
unsigned int i,j;
unsigned int count=0X0F;
LPC_GPIO2-> FIODIR=0x000000FF;
LPC_GPIO2-> FIOPIN=0x000000FF;
```

```
LPC_GPIO1-> FIODIR=1<<26;
LPC_GPIO1-> FIOPIN=~(1<<26);
while(1)
{
if(count>0X0F) count=0;
for(i=0;i<10000;i++)
{
LPC_GPIO2->FIOPIN=data7[count];
LPC_GPIO1->FIOSET=0X04000000;
for(j=0;j<500;j++);
LPC_GPIO1->FIOCLR=0X04000000;
}
--count;
}
}
```

EXPERIMENT 9: Interface a simple Switch and display its status through Relay, Buzzer and LED

AIM: Interface a simple Switch and display its status through Relay, Buzzer and LED

Relay Configurations:

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

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

Relay connection table.

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

Buzzer & Speaker Pin Configuration.

A buzzer or beeper is an audio signaling device, which may be mechanical, electromechanical or piezoelectric. Typical uses of buzzers and beepers include alarm devices, timers and confirmation of user input such as a mouse click or keystroke. LS1 is a small 12mm round buzzer that operates around the audible 2kHz range. We drove it through Q8 which in turn connected to P3.25.We need to send Logic „1" to generate beep.

A speaker is a device which converts electrical signal to audible signal.J7 is a two pin RMC where we are connecting external speaker. We drove it through Q5 which in turn connected to P3.25 via Jumper JP2.We need to send Logic „1" to generate Tone.

Speaker /Buzzer pin connection Table.

Speaker/ Buzzer Pin	LPC-1768 Pin No	LPC-1768 Port No
1	27	P3.25

```
#include <LPC17XX.H>
#define KEY1 (1<<14)
#define KEY2 (1<<15)
#define KEY3 (1<<16)
#define LED (1<<11)
#define RELAY (1<<25)
#define BUZZ (1<<25)
#define KEY_PIN LPC_GPIO1->FIOPIN
int main(void)
{
LPC_GPIO0->FIODIR|=LED;
LPC_GPIO0->FIOCLR=LED;
LPC_GPIO0->FIODIR|=RELAY;
LPC_GPIO0->FIOCLR=RELAY;
LPC_GPIO3->FIODIR|=BUZZ;
LPC_GPIO3->FIOCLR=BUZZ;
LPC_GPIO1->FIODIR=~(KEY1|KEY2|KEY3);
LPC_GPIO1->FIOSET=(KEY1|KEY2|KEY3);
while(1)
{
if(!(KEY_PIN & KEY1))
LPC_GPIO0->FIOSET=LED;
else
LPC_GPIO0->FIOCLR=LED;
if(!(KEY_PIN & KEY2))
LPC_GPIO0->FIOSET=RELAY;
```

```
else
LPC_GPIO0->FIOCLR=RELAY;
if(!(KEY_PIN & KEY3))
LPC_GPIO3->FIOSET=BUZZ;
else
LPC_GPIO3->FIOCLR=BUZZ;
}
}
// blinking of a LED
#include<LPC17xx.H>
void delay (unsigned int count)
{
    unsigned int j=0,i=0;
    for(j=0;j<count;j++)
    {
            for(i=0;i<12000;i++);
 }
}
int main(void)
{
    unsigned int del=200;
    delay(1000);
    LPC_GPIO0->FIODIR=0x000000F0;
    while(1)
    {
            LPC_GPIO0->FIOSET=0x000000E0;
            delay(del);
            LPC_GPIO0->FIOCLR=0x000000E0;
            delay(del);
    }
}
```

EXPERIMENT 10: Measure ambient temperature using a sensor and SPI ADC IC.

AIM: Measure ambient temperature using a sensor and SPI ADC IC.

Description

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

As VTCM3_3 board comes with two ADC inputs. One is connected to a potentiometer for external analog voltage and another is for temperature measurement.A 5K variable POT is connected to AD0(P0.23) input of LPC1768. By varying this we are applying 0 to 3.3V to ADC0 input. This analog voltage input can be converted into digital output.An LM35 temperature sensor is connected through ADC SSP IC to SSEL input of LPC1768 for ambient temperature measurement.

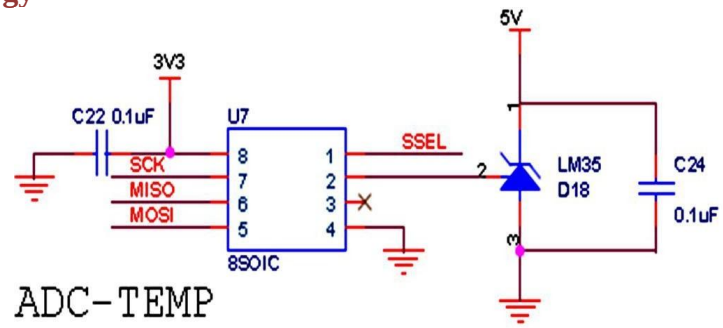
SPI Registers

SPCR SPI Control Register : used to configure SPI

SPSR SPI status Register :

SPDR SPI Data Register : contains received data or data to be transmitted

SPCCR SPI Clock Counter Register : used to control master SCK frequency



MOSI -- P0.18

MISO -- P0.17

SCK -- P0.15

SSEL -- P0.16

```
#include<lpc17xx.h>
#include<stdio.h>
uint8_t dummy_u8;
void delay(unsigned int count)
{
unsigned int j=0,k=0; for(j=0;j<count;j++)
{
for(k=0;k<12000;k++);
}}
void lcd_command_write(uint32_t val1)
{
LPC_GPIO2->FIOPIN= val1;
 LPC_GPIO0->FIOCLR=(1<<28);
LPC_GPIO0->FIOSET=(1<<27);
delay(100);
LPC_GPIO0->FIOCLR=(1<<27);
delay(100);
}
void LCD_DATA( char val)
```

```
{
LPC_GPIO2->FIOPIN= val;
LPC_GPIO0->FIOSET=(1<<28);
LPC_GPIO0->FIOSET=(1<<27);
delay(100);
LPC_GPIO0->FIOCLR=(1<<27);
delay(100);
}

void lcd_init(void)
{
LPC_GPIO0->FIODIR =0X18000000;
LPC_GPIO2->FIODIR=0x000000FF;
delay(1000);
lcd_command_write(0x38); /*    8-bit interface, two line, 5X7 dots.     */
lcd_command_write(0x10); /*    display shift     */
lcd_command_write(0x0C); /*    display on        */
lcd_command_write(0x06) ; /*   cursor move direction */
lcd_command_write(0x01) ; /*   cursor home     */
delay(1000);
}
void lcd_putstring( char *string )
{
while(*string != '\0')
{
LCD_DATA( *string );
string++;
}}
void SPI_Init(void)
{
LPC_PINCON->PINSEL0 = 0xC0000000;       //SCK
LPC_PINCON->PINSEL1 = 0x0000003C;       //SSEL, MISO, MOSI
LPC_GPIO0->FIODIR |= 0x00058000;        //SSEL, MOSI, SCK are Outputs 15,16,18
```

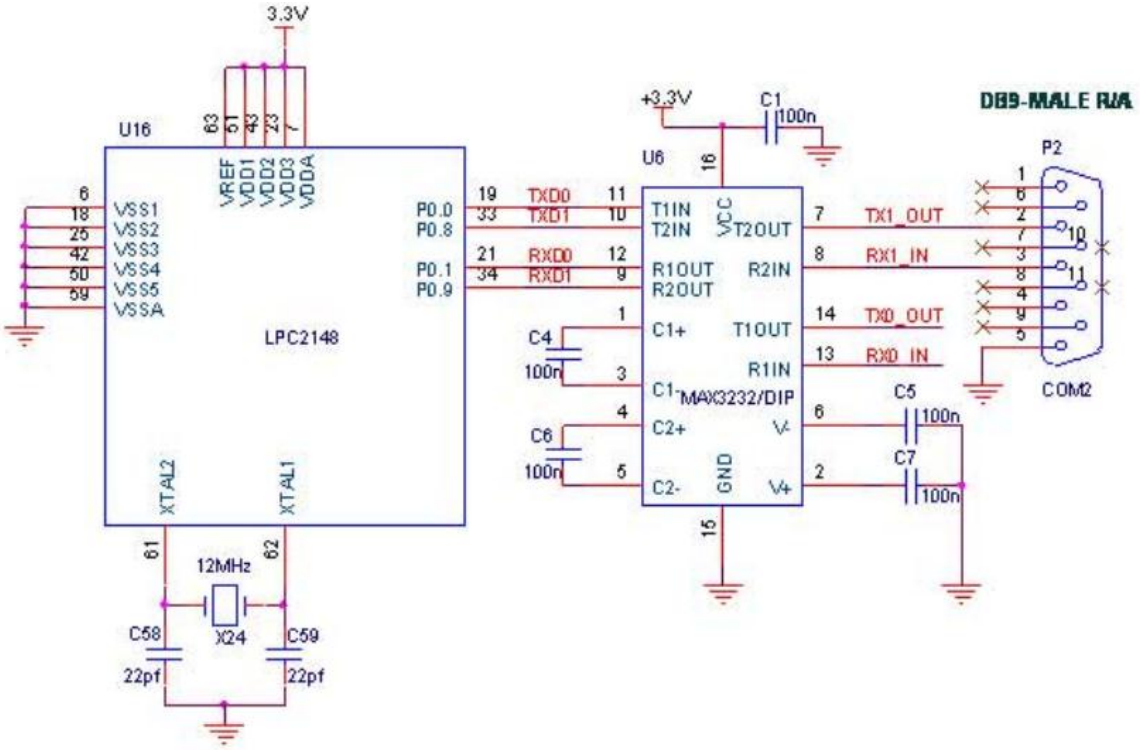
```
LPC_GPIO0->FIODIR &= ~0x00020000;        //MISO is Input 17
LPC_GPIO0->FIOSET |= (1 << 16);      // Disable the Slave Select
LPC_SC->PCONP |= (1 << 8);                    // enable power to spi cloc
LPC_SPI->SPCCR =6;
LPC_SPI->SPCR = ((0<<3) | (0<<4) | (1<<5)); dummy_u8 = LPC_SPI->SPSR;
dummy_u8 = LPC_SPI->SPDR; }
// Set Spi Clock
/* Dummy read to clear the flags */
/* Dummy read to clear the flags */
uint8_t SPI_Write (uint8_t spiData_u8)
{
LPC_SPI->SPDR = spiData_u8; dummy_u8 = 0;
while(dummy_u8 == 0)
{
dummy_u8 = LPC_SPI->SPSR & 0x80; // wait until data is sent
}
dummy_u8 = LPC_SPI->SPSR; spiData_u8 = (uint8_t)LPC_SPI->SPDR; return spiData_u8;
}

int main (void)
{
uint8_t msb, lsb; uint16_t ADCValue; float temp;
char lstr[10];
lcd_init(); SPI_Init();
lcd_command_write(0x80);       /*Display line if 0xC0 its line*/ lcd_putstring("ADC value=000 ");
lcd_command_write(0xC0);       /*Display line if 0xC0 its line*/ lcd_putstring("Tempr=00 degC");
while(1)
{
LPC_GPIO0->FIOCLR |= (1 << 16);
delay(1); SPI_Write(0x01);
msb = SPI_Write(0xA0); // SGL=1, ODD=0, MSBF=1
lsb = SPI_Write(0x00); LPC_GPIO0->FIOSET |= (1 << 16);
msb &= 0x0F;
```

```
ADCValue = (msb << 8) | lsb; sprintf (lstr,"%4u",ADCValue); lcd_command_write(0x8A);
lcd_putstring(lstr);
temp = ((ADCValue * 3.3) / 4096) * 100; sprintf (lstr, "%0.1f", temp); lcd_command_write(0xC6);
lcd_putstring(lstr);
delay(500);
}}
```

Beyond Syllabus:

Aim : To write an embedded C code for interfacing GPS Module with ARM CORTEX M3-LPC2148.

Circuit diagram



Procedure:

Give +3.3V power supply to LPC2148 Primer Board; connect +5V adapter with GPS module is connected with the LPC2148 Primer Board. Open the Hyper Terminal screen, select which port you are using and set the default settings. Now the screen should show some text messages. If you are not reading any data from UART0, then you just check the jumper connections & just check the serial cable is working. Otherwise you just check the code with debugging mode in Keil. If you want to see more details about debugging just see the videos in below link.

Program:

```
#define CR 0x0D
#include <LPC21xx.H>
void init_serial (void);
int putchar (int ch);
int getchar (void);
unsigned char test;
int main(void)
```

```
{
char *Ptr = "*** UART0 Demo ***\n\n\rType Characters to be echoed!!\n\n\r";
VPBDIV = 0x02; // Divide Pclk by two init_serial();
while(1)
{
while (*Ptr)
{
putchar(*Ptr++);
}
 putchar(getchar()); // Echo terminal
}
}
 void init_serial (void)
{
PINSEL0 = 0x00000005; // Enable RxD0 and TxD0
U0LCR = 0x00000083; //8 bits, no Parity, 1 Stop bit
 U0DLL = 0x000000C3; //9600 Baud Rate @ 30MHz VPB Clock
 U0LCR = 0x00000003;
}
int putchar (int ch)
{
if (ch == '\n')
{
while (!(U0LSR & 0x20));
 U0THR = CR;
}
while (!(U0LSR & 0x20));
return (U0THR = ch);
}
int getchar (void)
{
while (!(U0LSR & 0x01));
return (U0RBR);
```

}

Result:

Thus, interfacing of GPS module with ARM CORTEX M3- LPC2148 using embedded C code was executed and verified successfully.

VIVA QUESTIONS

VIVA QUESTIONS FOR EMBEDDED CONTROLLER LAB

1) Explain what is embedded system in a computer system?
2) Mention what are the essential components of embedded system?
3) Mention how I/O devices are classified for embedded system?
4) Why embedded system is useful?
5) Explain what are real-time embedded systems?
6) Explain what is microcontroller?
7) Mention what is the difference between microprocessor and microcontroller?
8) What does DMA address will deal with?
9) Explain what is interrupt latency? How can you reduce it?
10) Mention what are buses used for communication in embedded system?
11) List out various uses of timers in embedded system?
12) Explain what is a Watchdog Timer?
13) Explain what is the need for an infinite loop in embedded systems?
14) List out some of the commonly found errors in Embedded Systems?
15) Explain what is semaphore?