INDEX

DEPARTMENT VISION & MISSION

VISION

To become a pioneer in developing competent professionals with societal and ethical values through transformational learning and interdisciplinary research in the field of Electronics and Communication Engineering.

MISSION

The department of Electronics and Communication is committed to:

M1: Offer quality technical education through experiential learning to produce competent engineering professionals.

M2: Encourage a culture of innovation and multidisciplinary research in collaboration with industries/universities.

M3: Develop interpersonal, intrapersonal, entrepreneurial and communication skills among students to enhance their employability.

M4: Create a congenial environment for the faculty and students to achieve their desired goals and to serve society by upholding ethical values.

EMBEDDED SYSTEM LAB SYLLABUS

Sub Code: 18ECL66 **Exam Hours**: 03 Hrs
Hrs/Week: 03 Exam **Marks**: 60 Marks
IA Marks: 40

PART-A:

(Conduct the following experiments on an ARM CORTEX M3 evaluation board to learn ALP and using evaluation version of Embedded 'C' &Keil Uvision-4tool/compiler.)

1. ALP to multiply two 16 bit binary numbers.
2. ALP to find the sum of first 10 integers.
3. ALP to find the number of 0's and 1's in a 32 bit data.
4. ALP to determine the given 16 bit number is ODD or EVEN.
5. ALP to write data in RAM.

PART-B:

(Conduct the following experiments on an ARM CORTEX M3evaluation board using evaluation version of Embedded 'C' &Keil Uvision-4tool/compiler.)

6. Display "Hello World" message using Internal UART.
7. Interface and Control a DC Motor.
8. Interface a Stepper motor and rotate it in clockwise and anti-clockwisedirection.
9. Interface a DAC and generate Triangular and Square waveforms.
10. Interface a 4x4 keyboard and display the key code on an LCD.
11. Deonstrate the use of an external interrupt to toggle an LED On/Off.
12. Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between.
13. Measure Ambient temperature using a sensor and SPI ADC IC.

Beyond Syllabus:

i).Using the Internal PWM module of ARM controller generate PWM and vary itsduty cycle.

ii).Interface a simple Switch and display its status through Relay, Buzzer andLED.

Conduction of Practical Examination:

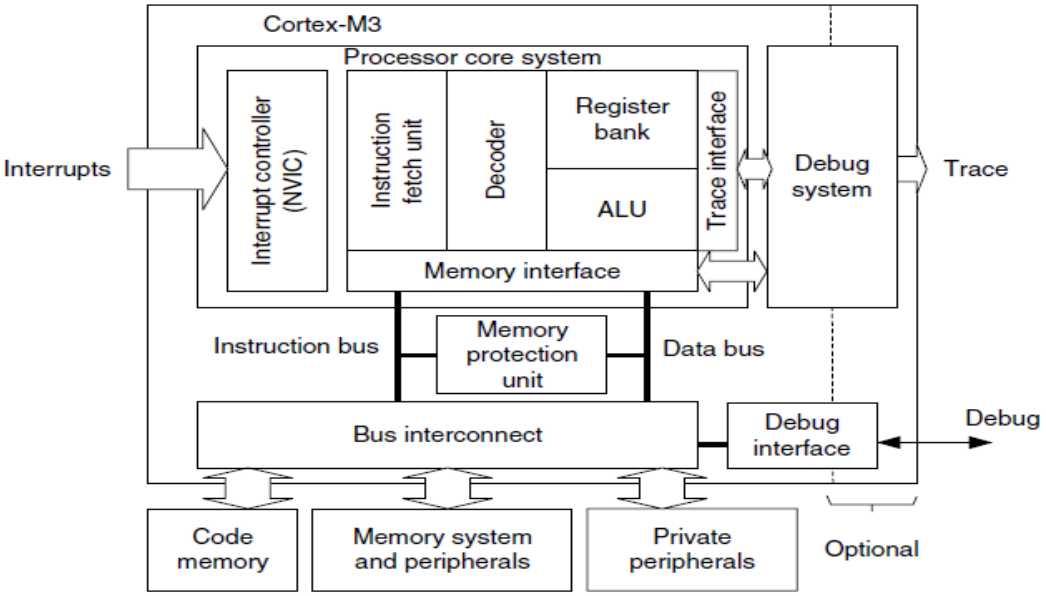
One question from PART A and One question from PART-B experiments to be asked in the practical examination.

INTRODUCTION TO ARM Cortex M3 PROCESSOR

Introduction

The ARM Cortex-M3 is a general purpose 32-bit microprocessor, which offers high performance and very low power consumption. The Cortex-M3 offers many new features, including a Thumb-2 instruction set, low interrupt latency, hardware divide, interruptible/continuable multiple load and store instructions, automatic state save and restore for interrupts, tightly integrated interrupt controller with Wake-up Interrupt Controller and multiple core buses capable of simultaneous accesses.

Pipeline techniques are employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.
The processor has a Harvard architecture, which means that it has a separate instruction bus and data bus. This allows instructions and data accesses to take place at the same time, and as a result of this, the performance of the processor increases because data accesses do not affect the instruction pipeline. This feature results in multiple bus interfaces on Cortex-M3, each with optimized usage and the ability to be used simultaneously. However, the instruction and data buses share the same memory space (a unified memory system). In other words, you cannot get 8 GB of memory space just because you have separate bus interfaces.
A simplified block diagram of the Cortex-m3 architecture is shown below



It is worthwhile highlighting that the Cortex-M3 processor is not the first ARM processor to be used to create generic micro controllers. The venerable ARM7 processor has been very successful in this market, The Cortex-M3 processor builds on the success of the ARM7 processor to deliver devices that are significantly easier to program and debug and yet deliver a higher processing capability.

Background of ARM architecture

ARM was formed in 1990 as Advanced RISC Machines Ltd., a joint venture of Apple Computer, Acorn Computer Group, and VLSI Technology. In 1991, ARM introduced the ARM6 processor family, and VLSI

became the initial licensee. Subsequently, additional companies, including Texas Instruments, NEC, Sharp, and ST Microelectronics, licensed the ARM processor designs, extending the applications of ARM processors into mobile phones, computer hard disks, personal digital assistants (PDAs), home entertainment systems, and many other consumer products.

Nowadays, ARM partners ship in excess of 2 billion ARM processors each year. Unlike many semiconductor companies, ARM does not manufacture processors or sell the chips directly. Instead, ARM licenses the processor designs to business partners, including a majority of the world's leading semiconductor companies. Based on the ARM low-cost and power-efficient processor designs, these partners create their processors, micro controllers, and system-on-chip solutions. This business model is commonly called intellectual property (IP) licensing.

Architecture versions

Over the years, ARM has continued to develop new processors and system blocks. These include the popular ARM7TDMI processor and, more recently, the ARM1176TZ (F)-S processor, which is used in high-end applications such as smart phones. The evolution of features and enhancements to the processors over time has led to successive versions of the ARM architecture. Note that architecture version numbers are independent from processor names. For example, the ARM7TDMI processor is based on the ARMv4T architecture (the *T* is for *Thumb* instruction mode support).

The ARMv5E architecture was introduced with the ARM9E processor families, including the ARM926E-S and ARM946E-S processors. This architecture added "Enhanced" Digital Signal Processing (DSP) instructions for multimedia applications. With the arrival of the ARM11 processor family, the architecture was extended to the ARMv6. New features in this architecture included memory system features and Single Instruction–Multiple Data (SIMD) instructions. Processors based on the ARMv6 architecture include the ARM1136J (F)-S, the ARM1156T2 (F)-S, and the ARM1176JZ (F)-S.

Over the past several years, ARM extended its product portfolio by diversifying its CPU development, which resulted in the architecture version 7 or v7. In this version, the architecture design is divided into three profiles:

- The ***A profile***is designed for high-performance open application platforms.
- The ***R profile***is designed for high-end embedded systems in which real-time performance is needed.
- The ***M profile***is designed for deeply embedded micro controller-type systems.

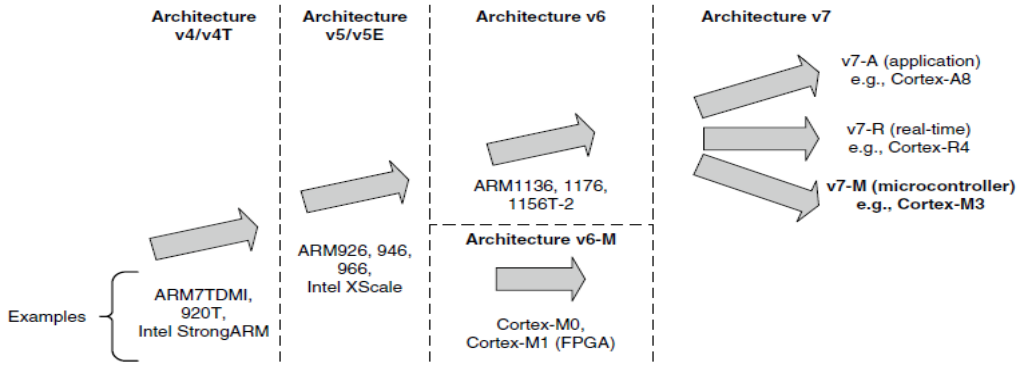
Bit more details on these profiles

***A Profile (ARMv7-A)*:** Application processors which are designed to handle complex applications such as high-end embedded operating systems (OSs) (e.g., Symbian, Linux, and Windows Embedded). These processors requiring the highest processing power, virtual memory system support with memory management units (MMUs), and, optionally, enhanced Java support and a secure program execution environment. Example products include high-end mobile phones and electronic wallets for financial transactions.

***R Profile (ARMv7-R)*:** Real-time, high-performance processors targeted primarily at the higher end of the real-time market, those applications, such as high-end breaking systems and hard drive controllers, in which high processing power and high reliability are essential and for which low latency is important.

***M Profile (ARMv7-M)*:** Processors targeting low-cost applications in which processing efficiency is important and cost, power consumption, low interrupt latency, and ease of use are critical, as well as industrial control applications, including real-time control systems. The Cortex processor families are the first products developed on architecture v7, and the Cortex- M3 processor is based on one profile of the v7 architecture, called ARM v7-M, an architecture specification for micro controller products.

Below figure shows the development stages of ARM versions



The Thumb-2 Technology and Instruction Set Architecture

The Thumb-2 technology extended the Thumb Instruction Set Architecture (ISA) into a highly efficient and powerful instruction set that delivers significant benefits in terms of ease of use, code size, and performance. The extended instruction set in Thumb-2 is a super set of the previous 16-bit Thumb instruction set, with additional 16-bit instructions alongside 32-bit instructions. It allows more complex operations to be carried out in the Thumb state, thus allowing higher efficiency by reducing the number of states switching between ARM state and Thumb state.

Focused on small memory system devices such as micro controllers and reducing the size of the processor, the Cortex-M3 supports only the Thumb-2 (and traditional Thumb) instruction set. Instead of using ARM instructions for some operations, as in traditional ARM processors, it uses the Thumb-2 instruction set for all operations. As a result, the Cortex-M3 processor is not backward compatible with traditional ARM processors.

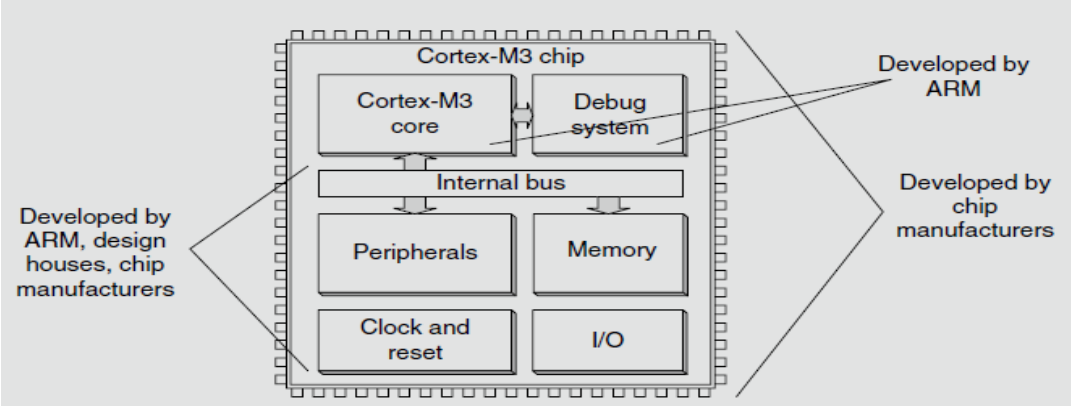
Nevertheless, the Cortex-M3 processor can execute almost all the 16-bit Thumb instructions, including all 16-bit Thumb instructions supported on ARM7 family processors, making application porting easy. With support for both 16-bit and 32-bit instructions in the Thumb-2 instruction set, there is no need to switch the processor between Thumb state (16-bit instructions) and ARM state (32-bit instructions). For example, in ARM7 or ARM9 family processors, you might need to switch to ARM state if you want to carry out complex calculations or a large number of conditional operations and good performance is needed, whereas in the Cortex-M3 processor, you can mix 32-bit instructions with 16-bit instructions without switching state, getting high code density and high performance with no extra complexity.

The Thumb-2 instruction set is a very important feature of the ARMv7 architecture. Compared with the instructions supported on ARM7 family processors (ARMv4T architecture), the Cortex-M3 processor instruction set has a large number of new features. For the first time, hardware divide instruction is available on an ARM processor, and a number of multiply instructions are also available on the Cortex-M3 processor to improve data-crunching performance. The Cortex-M3 processor also supports unaligned data accesses, a feature previously available only in high-end processors.

Applications of Cortex M3 processors

- *Low-cost micro controllers*:
- *AutomotiveIndustry*
- *Data communications*
- *Industrial control applications*
- *Consumer products*:

The Cortex-M3 Processor versus Cortex-M3-Based Micro Controllers



The Cortex-M3 processor is the central processing unit (CPU) of a micro controller chip. In addition, a number of other components are required for the whole Cortex-M3 processor-based micro controller. After chip manufacturers license the Cortex-M3 processor, they can put the Cortex-M3 processor in their silicon designs, adding memory, peripherals, input/output (I/O), and other features. Cortex-M3 processor-based chips from different manufacturers will have different memory sizes, types, peripherals, and features.

INTRODUCTION TO MICRO CONTROLLER LPC1768

Architectural Overview

The LPC1768FBD100 is an ARM Cortex-M3 based micro controller for embedded applications requiring a high level of integration and low power dissipation. The ARM Cortex-M3 is a next generation core that offers system enhancements such as modernized debug features and a higher level of support block integration. LPC1768 operate up to 100 MHz CPU frequency.

The peripheral complement of the LPC1768 includes up to 512 kilo bytes of flash memory, up to 64KB of data memory, Ethernet MAC, a USB interface that can be configured as either Host, Device, or OTG, 8 channel general purpose DMA controller, 4 UARTs, 2 CAN channels, 2 SSP controllers, SPI interface, 3 I2C interfaces, 2-input plus 2-output I2S interface, 8 channel 12-bit ADC, 10-bit DAC, motor control PWM, Quadrature Encoder interface, 4 general purpose timers, 6-output general purpose PWM, ultra-low power RTC with separate battery supply, and up to 70 general purpose I/O pins.

The LPC1768 use a multi layer AHB(Advanced High Performance Bus) matrix to connect the ARM Cortex-M3 buses and other bus masters to peripherals in a flexible manner that optimizes performance by allowing peripherals that are on different slaves ports of the matrix to be accessed simultaneously by different bus masters.

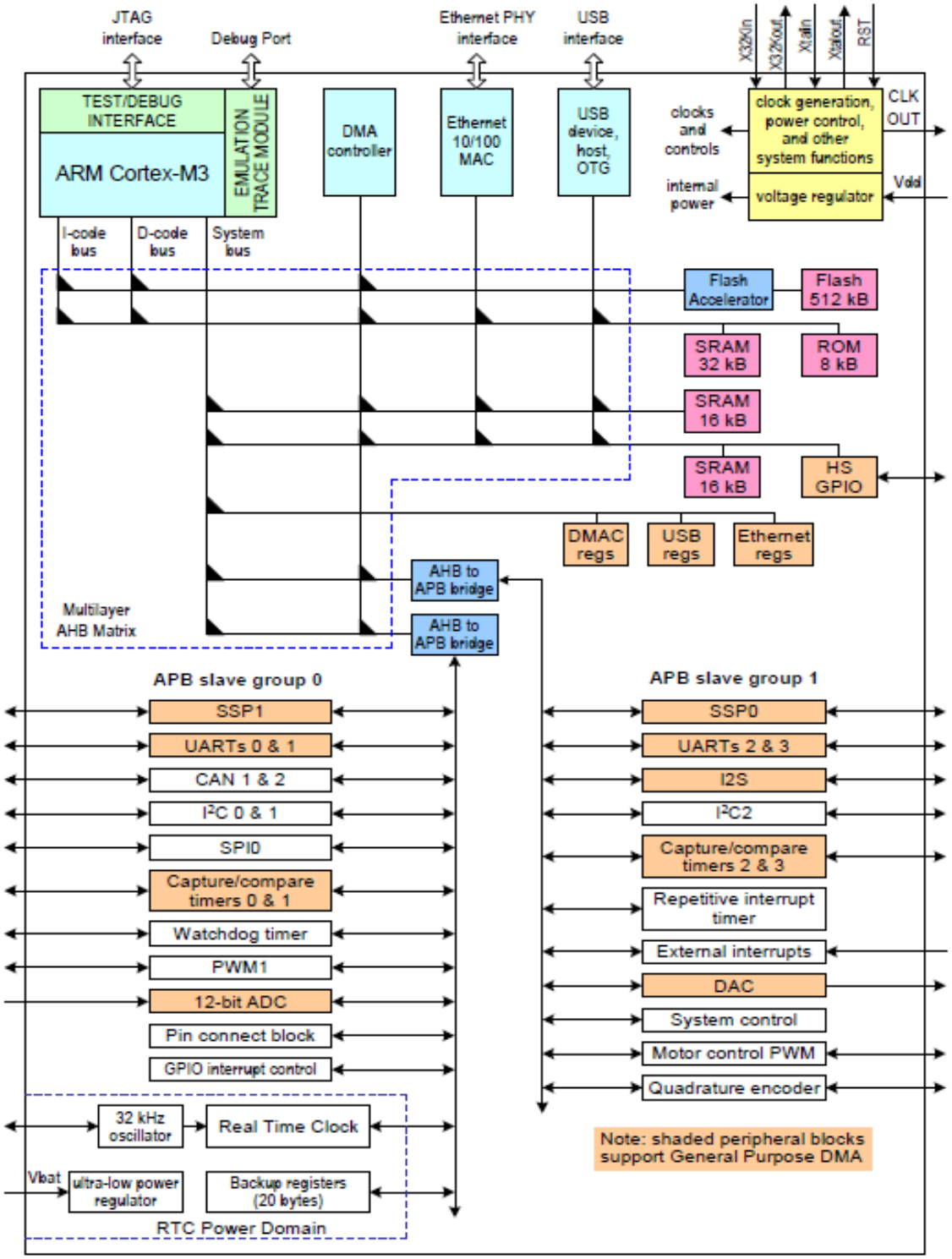
On-chip flash memory system

The LPC1768 contains up to 512 KB of on-chip flash memory. A flash memory accelerator maximizes performance for use with the two fast AHB Lite buses. This memory may be used for both code and data storage. Programming of the flash memory may be accomplished in several ways. It may be programmed In System via the serial port. The application program may also erase and/or program the flash while the application is running, allowing a great degree of flexibility for data storage field firmware upgrades, etc.

On-chip Static RAM

The LPC1768 contains up to 64 KB of on-chip static RAM memory. Up to 32 KB of SRAM, accessible by the CPU and all three DMA controllers are on a higher-speed bus. Devices containing more than 32 KB SRAM have two additional 16 KB SRAM blocks, each situated on separate slave ports on the AHB multilayer matrix. This architecture allows the possibility for CPU and DMA accesses to be separated in such a way that there are few or no delays for the bus masters.

Block Diagram of LPC1768



A brief description of the blocks:

Nested vector interrupt controller

The NVIC is an integral part of the Cortex-M3. The tight coupling to the CPU allows for low interrupt latency and efficient processing of late arriving interrupts.

Features

- Controls system exceptions and peripheral interrupts
- In the LPC1768, the NVIC supports 33 vectored interrupts
- 32 programmable interrupt priority levels, with hardware priority level masking
- Relocatable vector table
- Non-Maskable Interrupt (NMI)
- Software interrupt generation

Interrupt sources

Each peripheral device has one interrupt line connected to the NVIC but may have several interrupt flags. Individual interrupt flags may also represent more than one interrupt source.

Any pin on Port 0 and Port 2 (total of 42 pins) regardless of the selected function, can be programmed to generate an interrupt on a rising edge, a falling edge, or both.

General purpose DMA controller

The GPDMA (General Purpose Direct Memory Access) is an AMBA AHB (Advanced Micro controller Bus Architecture Advance high performance bus) compliant peripheral allowing selected peripherals to have DMA support.

The GPDMA enables peripheral-to-memory, memory-to-peripheral, peripheral-to-peripheral, and memory-to-memory transactions. The source and destination areas can each be either a memory region or a peripheral, and can be accessed through the AHB master. The GPDMA controller allows data transfers between the USB and Ethernet controllers and the various on-chip SRAM areas. The supported APB peripherals are SSP0/1, all UARTs, the I2S-bus interface, the ADC, and the DAC. Two match signals for each timer can be used to trigger DMA transfers.

Function Configuration block

The selected pins of the micro controller to have more than one function. Configuration registers control the multiplexers to allow connection between the pin and the on-chip peripherals. Peripherals should be connected to the appropriate pins prior to being activated and prior to any related interrupt(s) being enabled.

Activity of any enabled peripheral function that is not mapped to a related pin should be considered undefined. Most pins can also be configured as open-drain outputs or to have a pull-up, pull-down, or no resistor enabled.

Fast general purpose parallel I/O

Device pins that are not connected to a specific peripheral function are controlled by the GPIO registers. Pins may be dynamically configured as inputs or outputs. Separate registers allow setting or clearing any number of outputs simultaneously. The value of the output register may be read back as well as the current state of the port pins.

USB interface

The Universal Serial Bus (USB) is a 4-wire bus that supports communication between a host and one or more (up to 127) peripherals. The host controller allocates the USB bandwidth to attached devices through a token-based protocol. The bus supports hot plugging and dynamic configuration of the devices. All transactions are initiated by the host controller.

The USB interface includes a device, Host, and OTG controller with on-chip PHY for device and Host functions. The OTG switching protocol is supported through the use of an external controller.

USB device controller enables 12 Mbit/s data exchange with a USB Host controller. It consists of a register interface, serial interface engine, endpoint buffer memory, and a DMA controller. The serial interface engine decodes the USB data stream and writes data to the appropriate endpoint buffer. The status of a completed USB transfer or error condition is indicated via status registers. An interrupt is also generated if enabled. When enabled, the DMA controller transfers data between the endpoint buffer and the on-chip SRAM.

12-bit ADC

The LPC1768 contain a single 12-bit successive approximation ADC with eight channels and DMA support.

10-bit DAC

The DAC allows to generate a variable analog output. The maximum output value of the DAC is VREFP.

UART's

The LPC1768 contain four UART's. In addition to standard transmit and receive data lines, UART1 also provides a full modem control handshake interface and support for RS-485/9-bit mode allowing both software address detection and automatic address detection using 9-bit mode.

The UART's include a fractional baud rate generator. Standard baud rates such as 115200 Baud can be achieved with any crystal frequency above 2 MHz

SPI serial I/O controller

The LPC1768 contain one SPI controller. SPI is a full duplex serial interface designed to handle multiple masters and slaves connected to a given bus. Only a single master and a single slave can communicate on the interface during a given data transfer. During a data transfer the master always sends 8 bits to 16 bits of data to the slave, and the slave always sends 8 bits to 16 bits of data to the master.

SSP serial I/O controller

The LPC1768 contain two SSP controllers. The SSP controller is capable of operation on a SPI, 4-wire SSI, or Micro wire bus. It can interact with multiple masters and slaves on the bus. Only a single master and a single slave can communicate on the bus during a given data transfer. The SSP supports full duplex transfers, with frames of 4 bits to 16 bits of data flowing from the master to the slave and from the slave to the master. In practice, often only one of these data flows carries meaningful data.

I2C-bus serial I/O controllers

The LPC1768 each contain three I2C-bus controllers. The I2C-bus is bidirectional for inter-IC control using only two wires: a Serial Clock line (SCL) and a Serial DAta line (SDA). Each device is recognized by a unique address and can operate as either a receiver-only device or a transmitter with the capability to both receive and send information (such as memory). Transmitters and/or receivers can operate in either master or slave mode, depending on whether the chip has to initiate a data transfer or is only addressed. The I2C is a multi-master bus and can be controlled by more than one bus master connected to it.

General purpose 32-bit timers/external event counters

The LPC1768 include four 32-bit timer/counters. The timer/counter is designed to count cycles of the system derived clock or an externally-supplied clock. It can optionally generate interrupts, generate timed DMA requests, or perform other actions at specified timer values, based on four match registers. Each timer/counter also includes two capture inputs to trap the timer value when an input signal transitions, optionally generating an interrupt.

Pulse width modulator

The PWM is based on the standard Timer block and inherits all of its features, although only the PWM function is pinned out on the LPC1768. The Timer is designed to count cycles of the system derived clock and optionally switch pins, generate interrupts or perform other actions when specified timer values occur, based on seven match registers. The PWM function is in addition to these features, and is based on match register events.

Watchdog timer

The purpose of the watchdog is to reset the micro controller within a reasonable amount of time if it enters an

erroneous state. When enabled, the watchdog will generate a system reset if the user program fails to 'feed' (or reload) the watchdog within a predetermined amount of time.

RTC and backup registers

The RTC is a set of counters for measuring time when system power is on, and optionally when it is off. The RTC on the LPC1768 is designed to have extremely low power consumption, i.e. less than 1 uA. The RTC will typically run from the main chip power supply, conserving battery power while the rest of the device is powered up. When operating from a battery, the RTC will continue working down to 2.1 V. Battery power can be provided from a standard 3 V Lithium button cell.

An ultra-low power 32 kHz oscillator will provide a 1 Hz clock to the time counting portion of the RTC, moving most of the power consumption out of the time counting function.

Clocking and Power Control

Crystal oscillators

The LPC1768 include three independent oscillators. These are the main oscillator, the IRC oscillator, and the RTC oscillator. Each oscillator can be used for more than one purpose as required in a particular application. Any of the three clock sources can be chosen by software to drive the main PLL and ultimately the CPU.

Following reset, the LPC1768 will operate from the Internal RC oscillator until switched by software. This allows systems to operate without any external crystal and the boot loader code to operate at a known frequency.

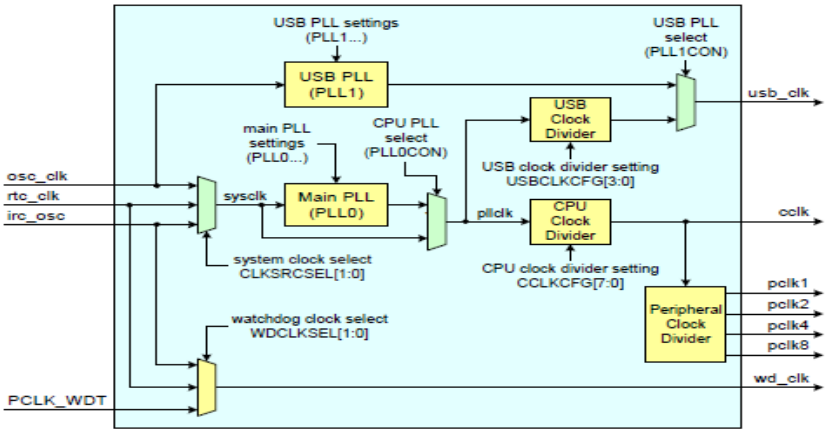
Power control

The LPC1768 support a variety of power control features. There are four special modes of processor power reduction: Sleep mode, Deep-sleep mode, Power-down mode, and Deep power-down mode. The CPU clock rate may also be controlled as needed by changing clock sources, reconfiguring PLL values, and/or altering the CPU clock divider value. This allows a trade-off of power versus processing speed based on application requirements. In addition, Peripheral Power Control allows shutting down the clocks to individual on-chip peripherals, allowing fine tuning of power consumption by eliminating all dynamic power use in any peripherals that are not required for the application. Each of the peripherals has its own clock divider which provides even better power control.

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

The LPC1768 also implement a separate power domain to allow turning off power to the bulk of the device while maintaining operation of the RTC and a small set of registers for storing data during any of the power-down modes.

Clock generation block diagram for LPC1768 is shown below



System Control

Reset

Reset has four sources on the LPC1768: the RESET pin, the Watchdog reset, power-on reset (POR), and the Brown-Out Detection (BOD) circuit.

The RESET pin is a Schmitt trigger input pin. Assertion of chip Reset by any source, once the operating voltage attains a usable level, causes the RSTOUT pin to go LOW. Once reset is de-asserted, or, in case of a BOD- triggered reset, once the voltage rises above the BOD threshold, the RSTOUT pin goes HIGH. In other words RSTOUT is high when the controller is in its active state.

Emulation and debugging

Debug and trace functions are integrated into the ARM Cortex-M3. Serial wire debug and trace functions are supported in addition to a standard JTAG debug and parallel trace functions. The ARM Cortex-M3 is configured to support up to eight breakpoints and four watch points.

Note: For further details on Controller blocks refer the User manual of LPC176x/5x – UM10360 available at www.nxp.com

TECHNICAL SPECIFICATIONS of LPC1768

Specifications of LPC1768:

- ARM Cortex-M3 processor runs up to 100 MHz frequency.
- ARM Cortex-M3 built-in Nested Vectored Interrupt Controller (NVIC).
- Up to 512kB on-chip flash program memory with In-System Programming (ISP) and In-Application Programming (IAP) capabilities. The combination of an enhanced flash memory accelerator and location of the flash memory on the CPU local code/data bus provides high code performance from flash.
- Up to 64kB on-chip SRAM includes:
  - Up to 32kB of SRAM on the CPU with local code/data bus for high-performance CPU access.
  - Up to two 16kB 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 instruction and data storage.
- Eight channel General Purpose DMA controller (GPDMA) on the AHB multilayer matrix that can be used with the SSP, I2S, UART, the Analog-to-Digital and Digital-to-Analog converter peripherals, timer match signals, GPIO, and for memory-to-memory transfers.
- Serial interfaces:
  - Ethernet MAC with RMII interface and dedicated DMA controller.
  - USB 2.0 full-speed controller that can be configured for either device, Host, or OTG operation with an on-chip PHY for device and Host functions and a dedicated DMA controller.
  - Four UART's with fractional baud rate generation, internal FIFO, IrDA, and DMA support. One UART has modem control I/O and RS-485/EIA-485 support.
  - Two-channel CAN controller.
  - Two SSP controllers with FIFO and multi-protocol capabilities. The SSP interfaces can be used with the GPDMA controller.
  - SPI controller with synchronous, serial, full duplex communication and programmable data length. SPI is included as a legacy peripheral and can be used instead of SSP0.
  - Three enhanced I2C-bus interfaces, one with an open-drain output supporting the full I2C specification and Fast mode plus with data rates of 1Mbit/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 interface can be used with the GPDMA. The I2S interface supports 3- wire data transmit and receive or 4-wire combined transmit and receive connections, as well as master clock output.
- Other peripherals:
  - 70 General Purpose I/O (GPIO) pins with configurable pull-up/down resistors, open drain mode, and repeater mode. All GPIOs are located on an AHB bus for fast access, and support Cortex-M3 bit-banding. GPIOs can be accessed by the General Purpose DMA Controller. Any pin of ports 0 and 2 can be used to generate an interrupt.
  - 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.

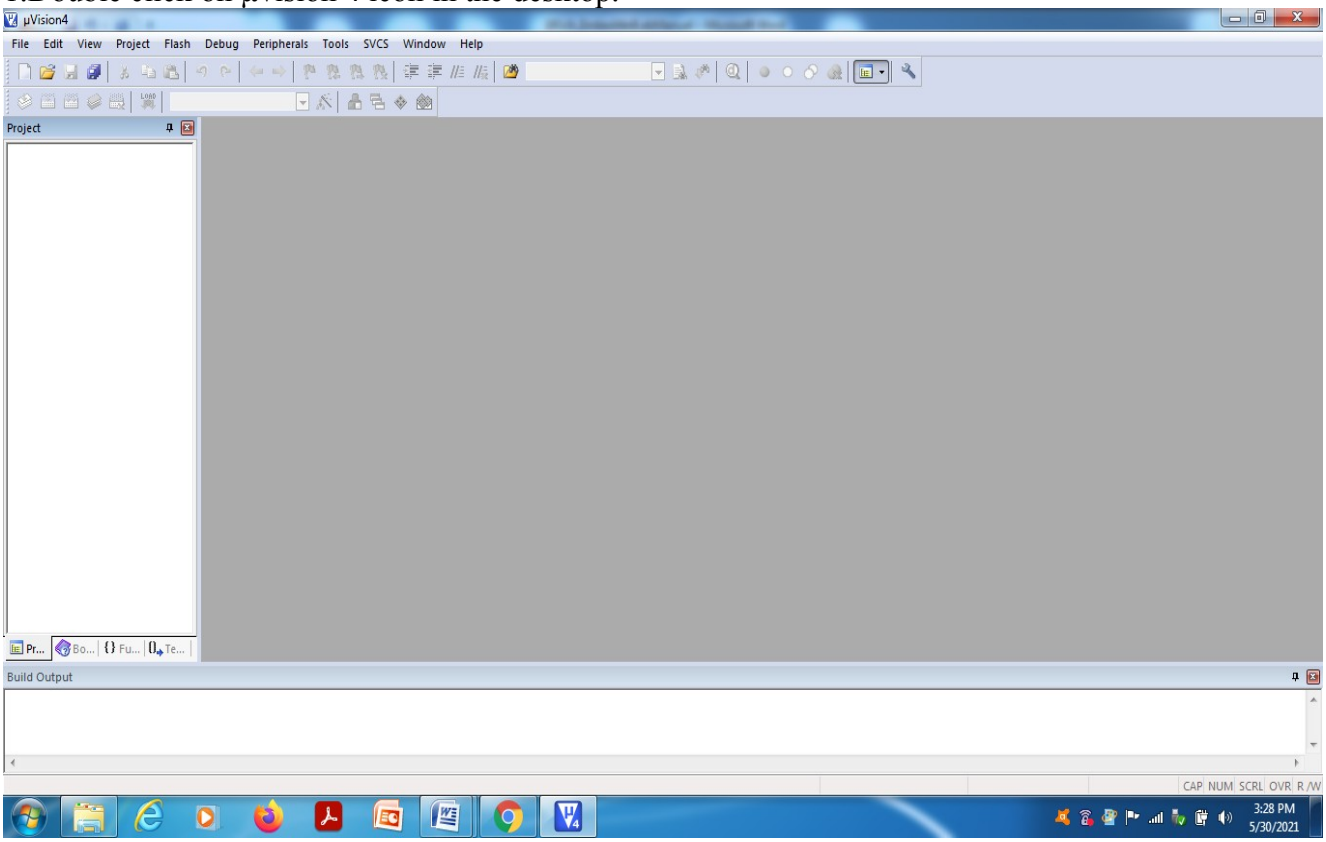
- One motor control PWM with support for three-phase motor control.
- Quadrature encoder interface that can monitor one external quadrature encoder.
- One standard PWM/timer block with external count input.
- Real-Time Clock (RTC) with a separate power domain. The RTC is clocked by a dedicated RTC oscillator. The RTC block includes 20 bytes of battery-powered backup registers, allowing system status to be stored when the rest of the chip is powered off. Battery power can be supplied from a standard 3 V Lithium button cell. The RTC will continue working when the battery voltage drops to as low as 2.1 V. An RTC interrupt can wake up the CPU from any reduced power mode.
- Watchdog Timer (WDT). The WDT can be clocked from the internal RC oscillator, the RTC oscillator, or the APB clock.
- Cortex-M3 system tick timer, including an external clock input option.
- Repetitive interrupt timer provides programmable and repeating timed interrupts.

- Standard JTAG test/debug interface as well as Serial Wire Debug and Serial Wire Trace Port options.
- Emulation trace module supports real-time trace.
- 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). Temperature range of -40 °C to 85 °C.
- Four external interrupt inputs configurable as edge/level sensitive. All pins on PORT0 and PORT2 can be used as edge sensitive interrupt sources.
- Non Maskable Interrupt (NMI) input.
- Clock output function that can reflect the main oscillator clock, IRC clock, RTC clock, CPU clock, or 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, PORT0/2 pin interrupt, and NMI).
- Each peripheral has its own clock divider for further power savings.
- Brownout detect with separate threshold for interrupt and forced reset.
- On-chip Power-On Reset (POR).
- On-chip 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.
- An on-chip 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.
- A second, dedicated PLL may be used for the USB interface in order to allow added flexibility for the Main PLL settings.
- Versatile pin function selection feature allows many possibilities for using on-chip peripheral functions.

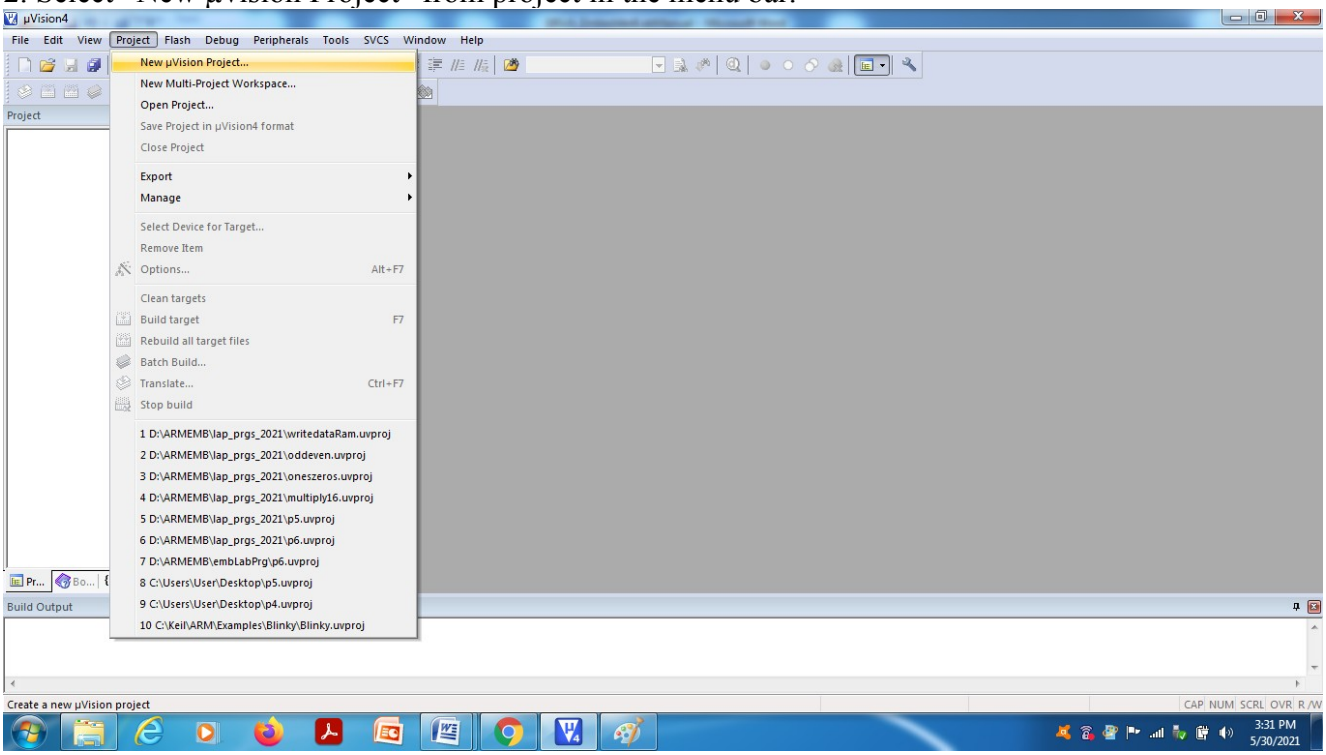
PART A
(Assembly Level Programming-ARM Cortex M3)
Software : Keil µvision 4

Software Handling Procedure:

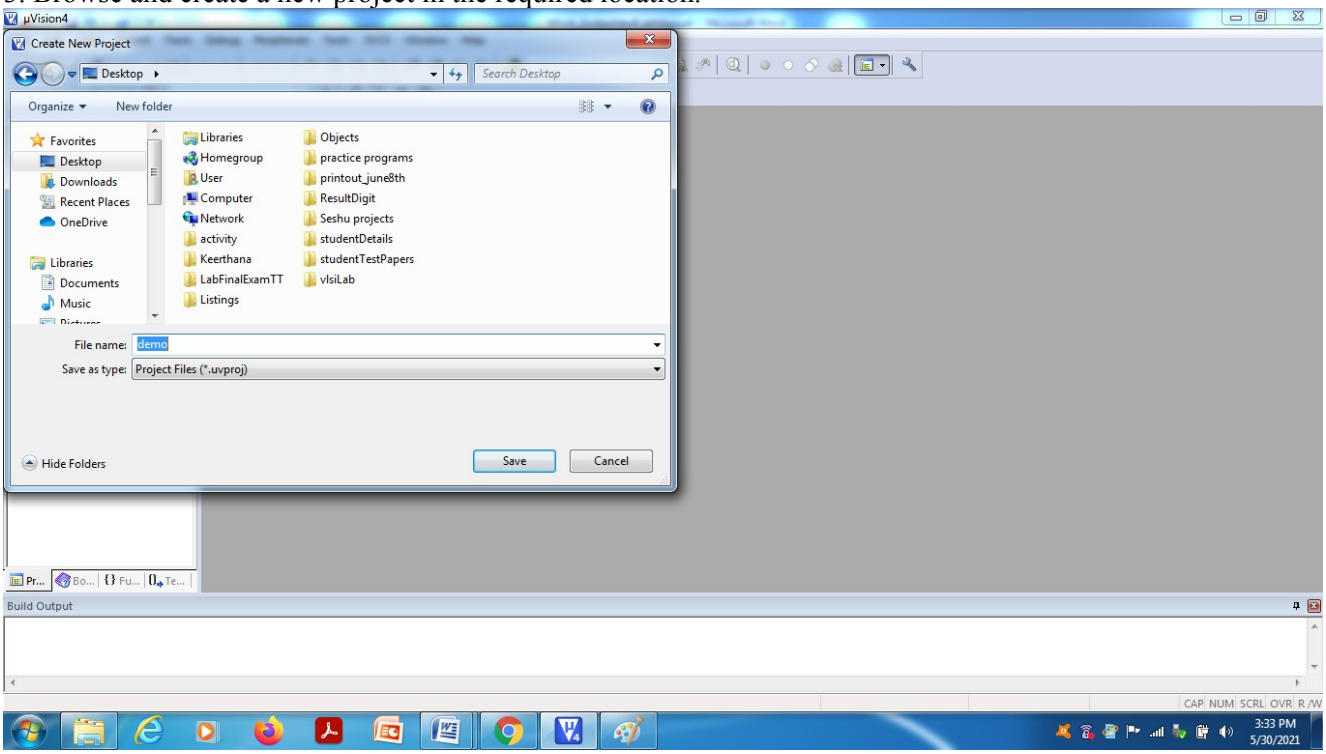
1.Double click on µvision 4 icon in the desktop.



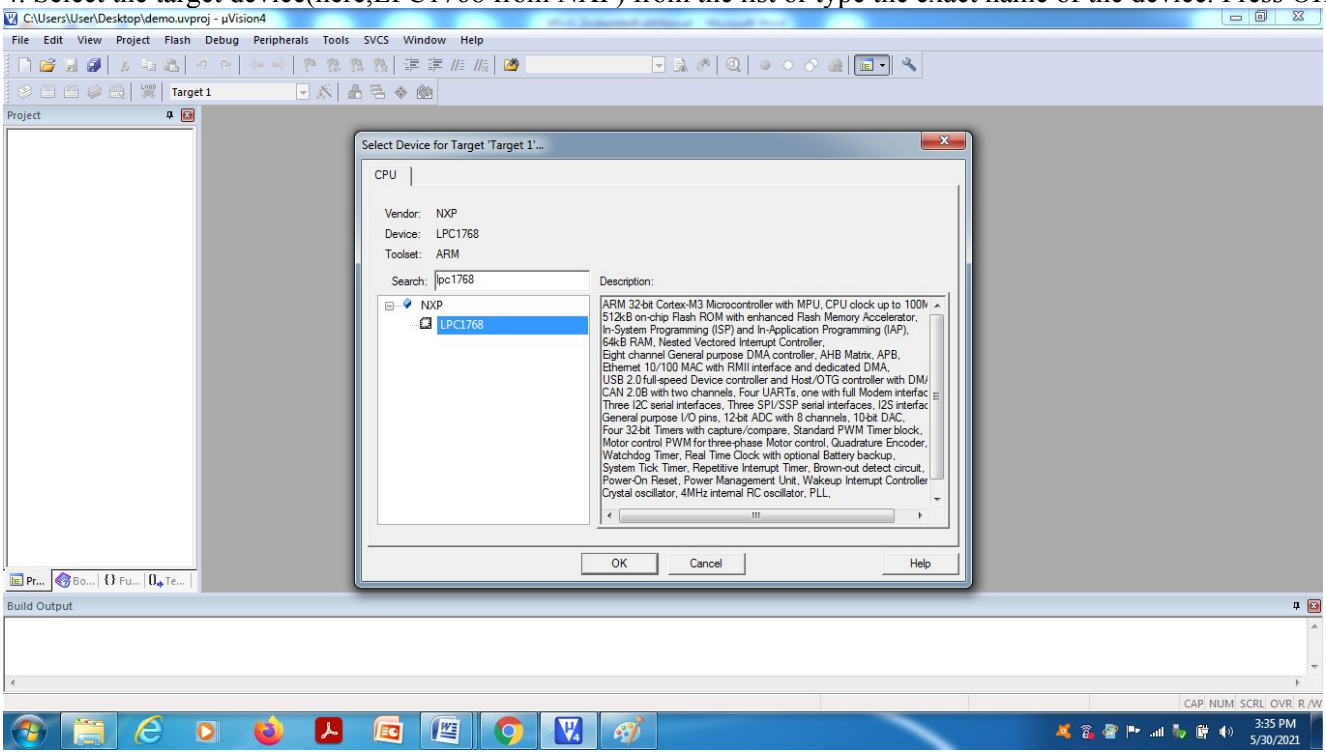
2. Select “New µvision Project” from project in the menu bar.



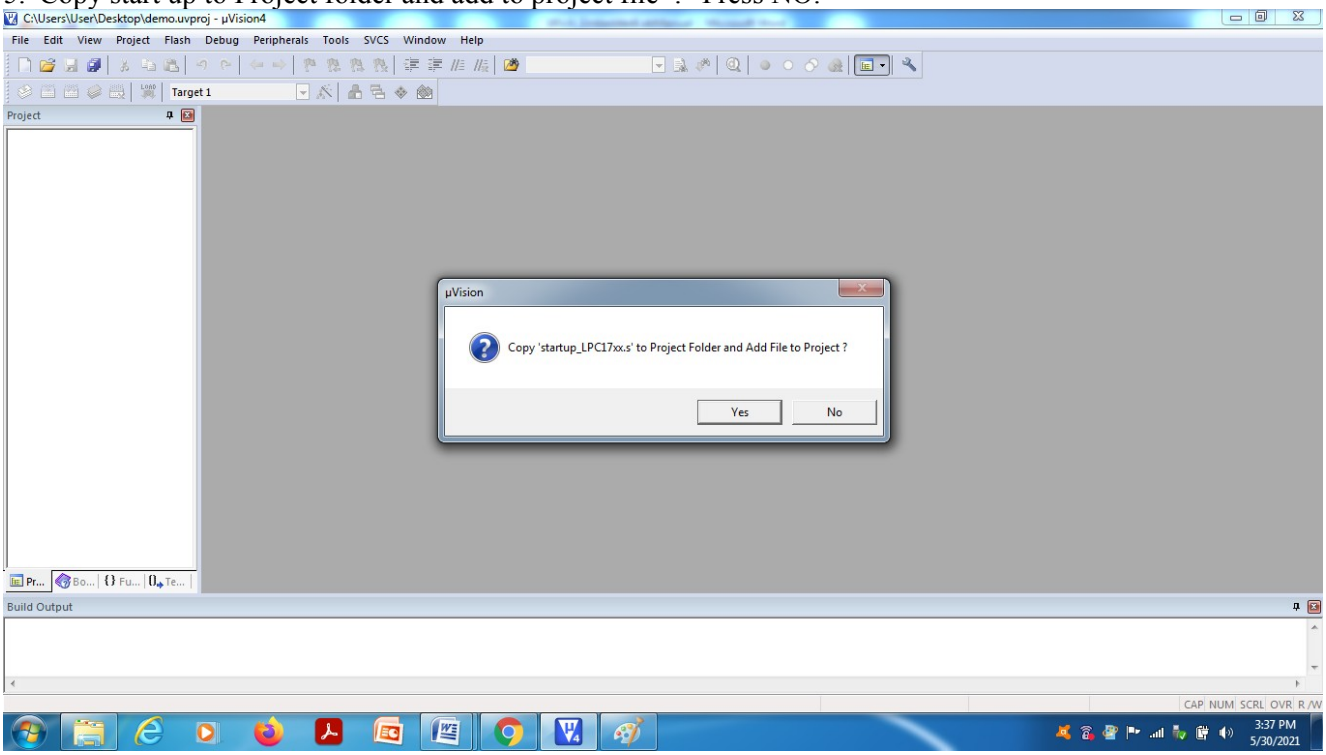
3. Browse and create a new project in the required location.



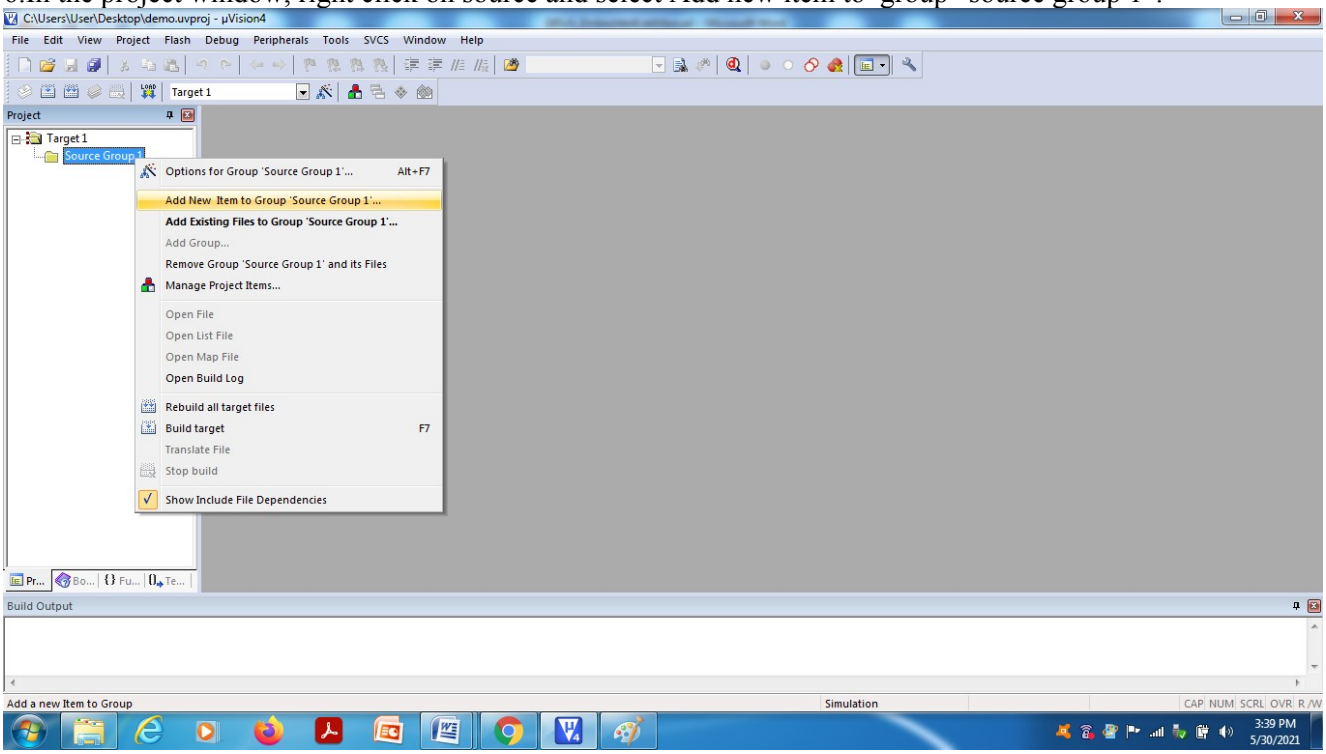
4. Select the target device(here,LPC1768 from NXP) from the list or type the exact name of the device. Press OK.



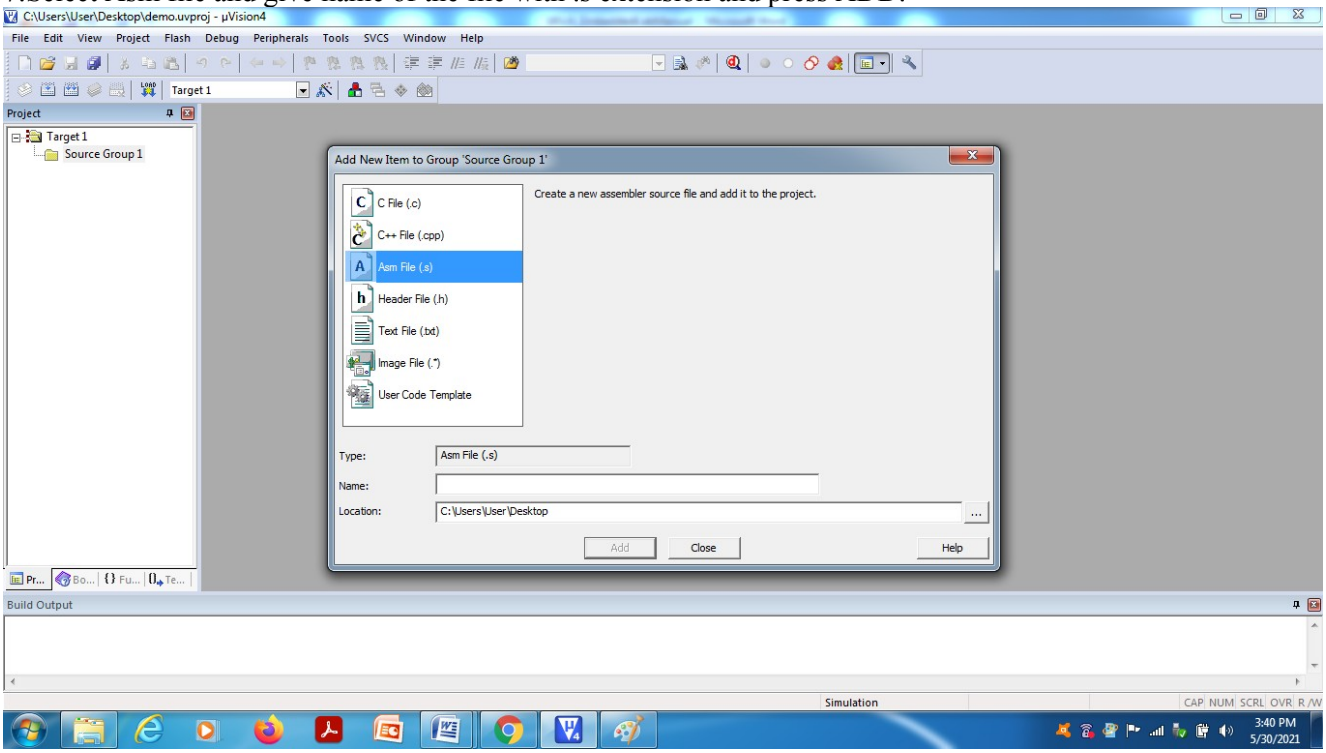
5."Copy start up to Project folder and add to project file"?- Press NO.



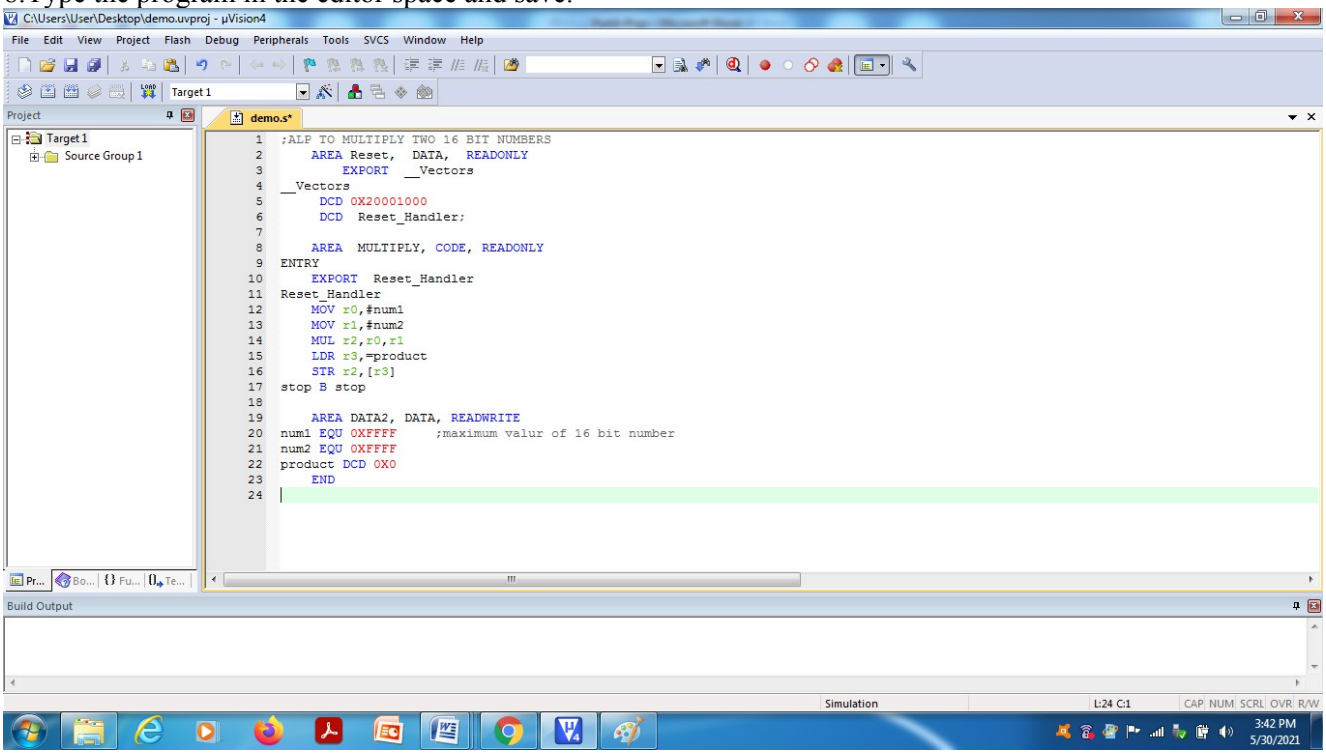
6.In the project window, right click on source and select Add new item to group “source group 1”.



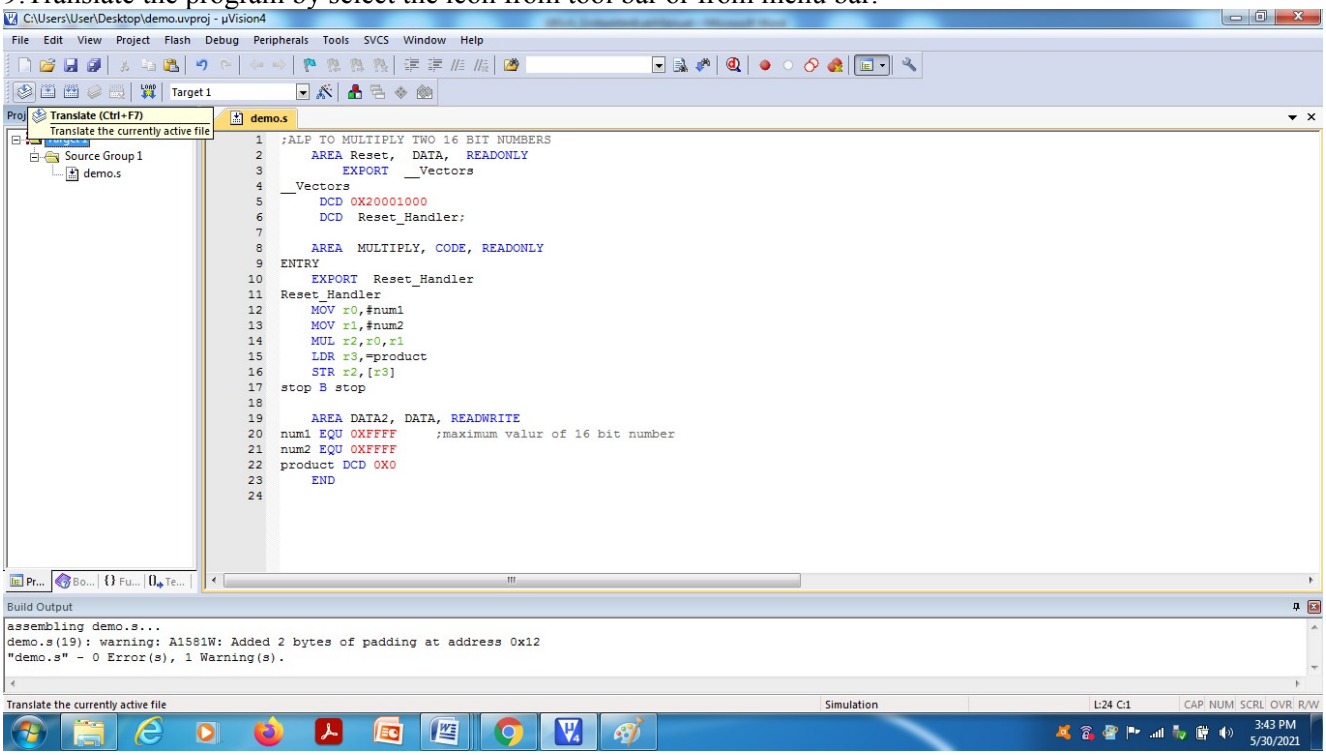
7.Select Asm file and give name of the file with .s extension and press ADD.



8.Type the program in the editor space and save.

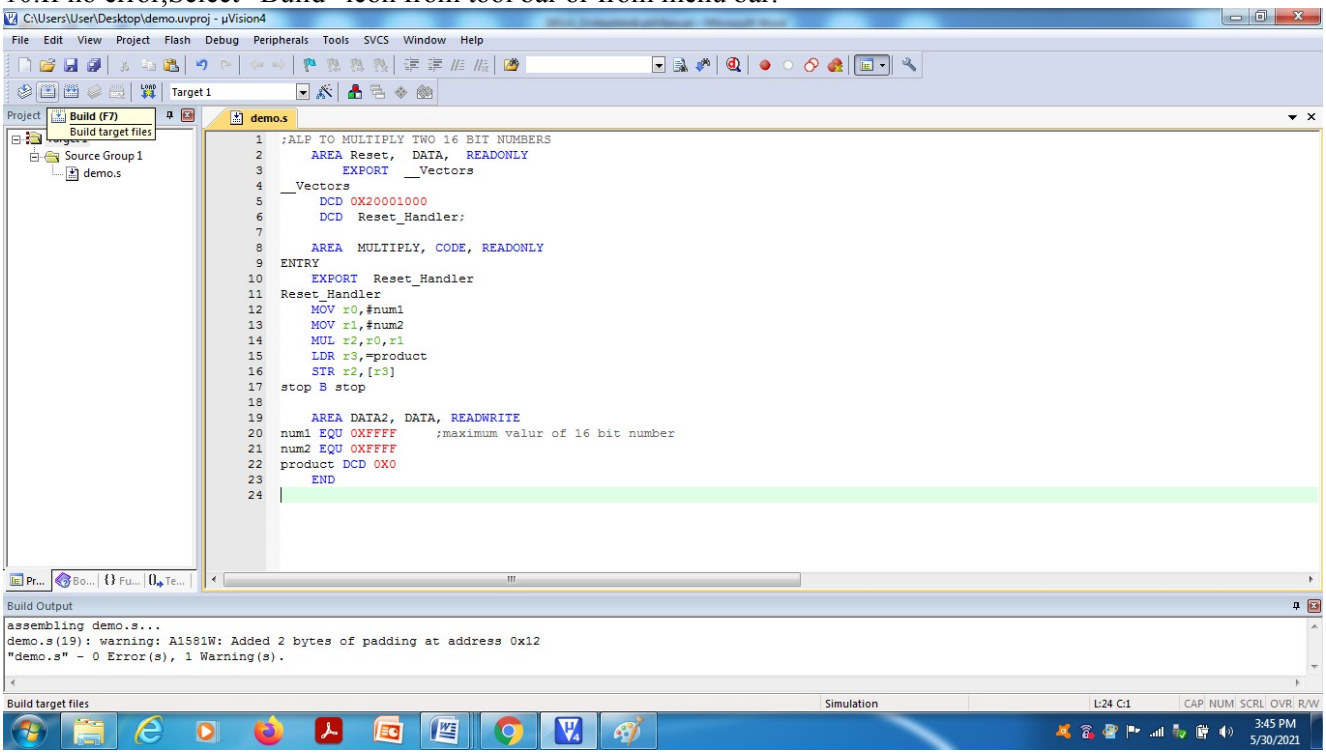


9.Translate the program by select the icon from tool bar or from menu bar.

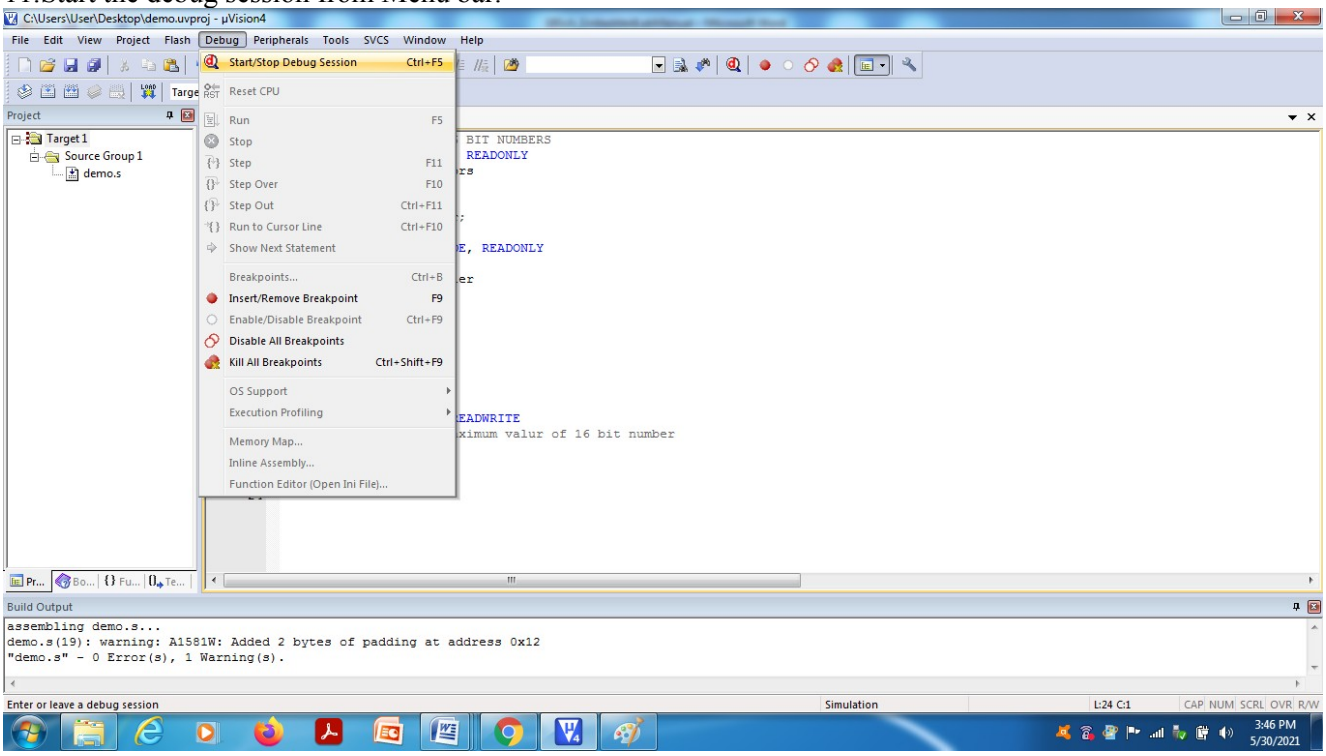


Check for errors and warnings in the bottom window.

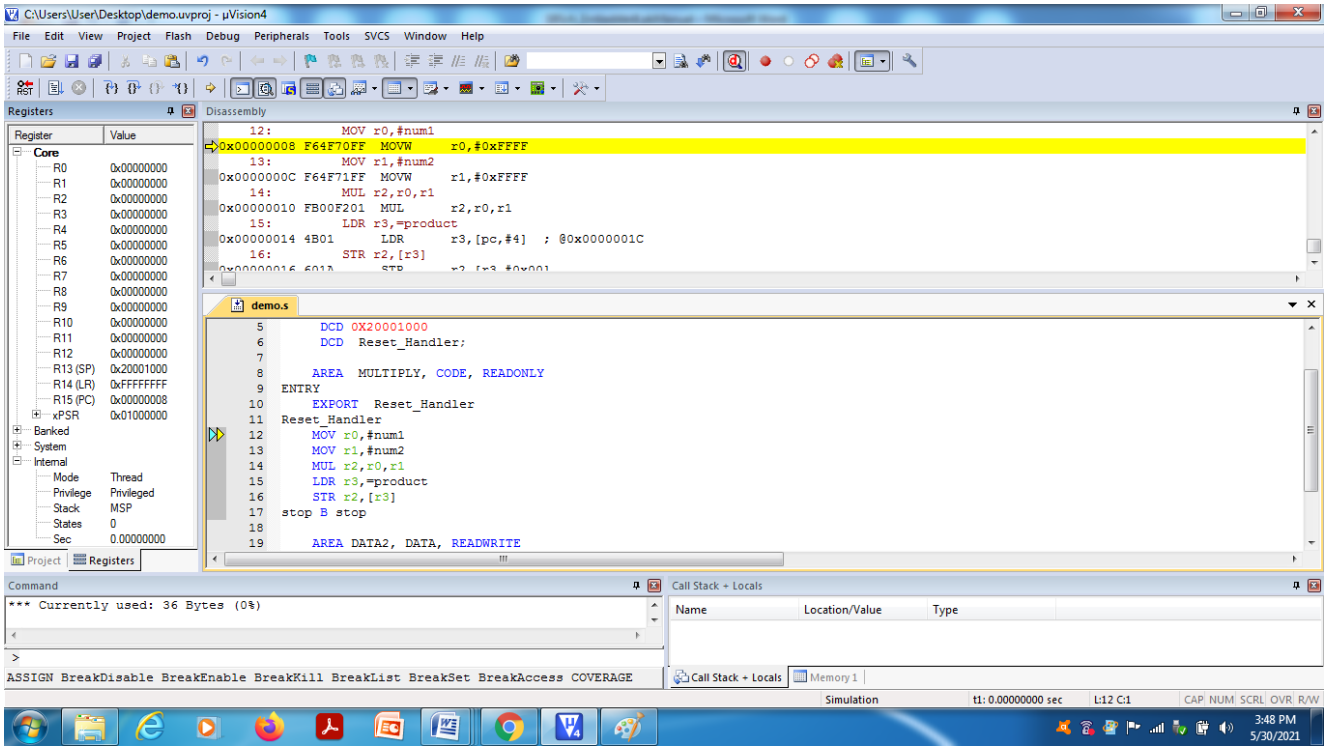
10.If no error,Select “Build” icon from tool bar or from menu bar.



11.Start the debug session from Menu bar.



12.PressOK

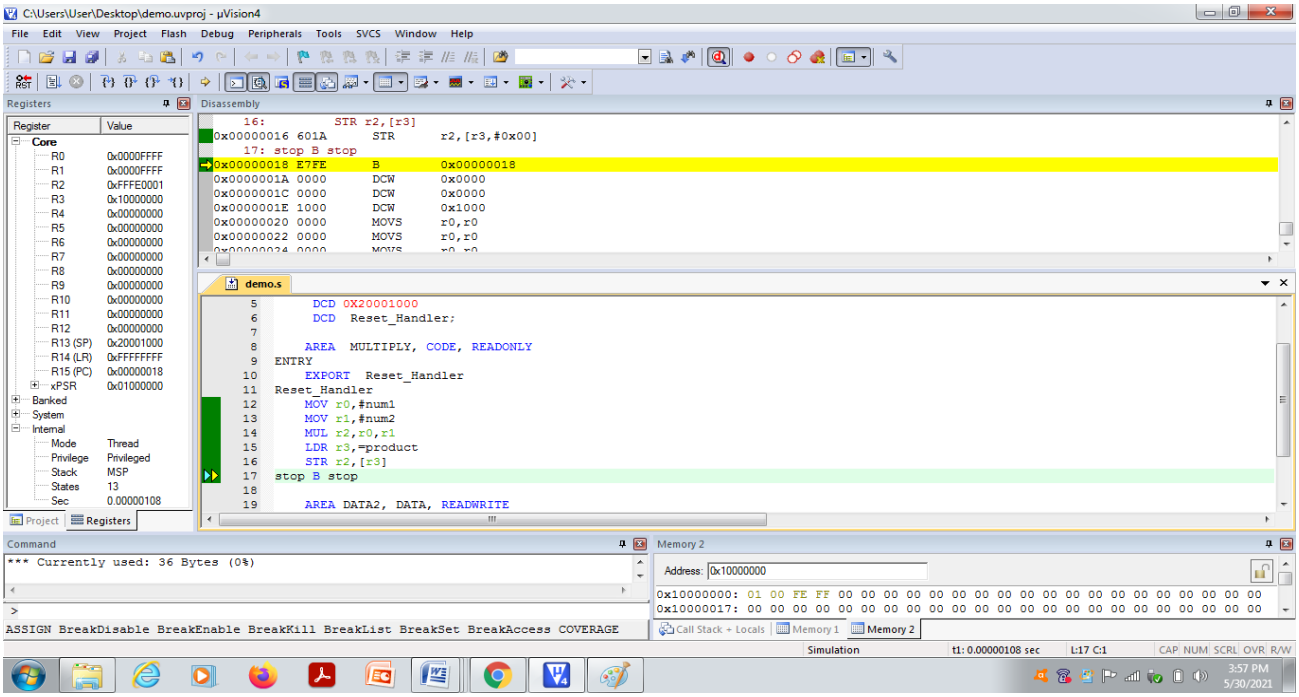


13.Press function key F11 or select “step” option under Debug menu for single step execution and verify the output in register window/Memory window/xPSR.

1.ALP TO MULTIPLY TWO 16 BIT NUMBERS

```
        AREA Reset,  DATA,  READONLY
          EXPORT  __Vectors
__Vectors
         DCD 0X20001000
         DCD  Reset_Handler;

        AREA  MULTIPLY, CODE, READONLY
ENTRY
        EXPORT  Reset_Handler
Reset_Handler
        MOV r0,#num1
        MOV r1,#num2
        MUL r2,r0,r1
        LDR r3,=product
        STR r2,[r3]
stop B stop

        AREA DATA2, DATA, READWRITE
num1 EQU 0XFFFF             ;maximum value of 16 bit number
num2 EQU 0XFFFF
product DCD 0X0
        END
```

Result:
(0xFFFF) x(0xFFFF) =0xFFFE0001 in the product memory location.



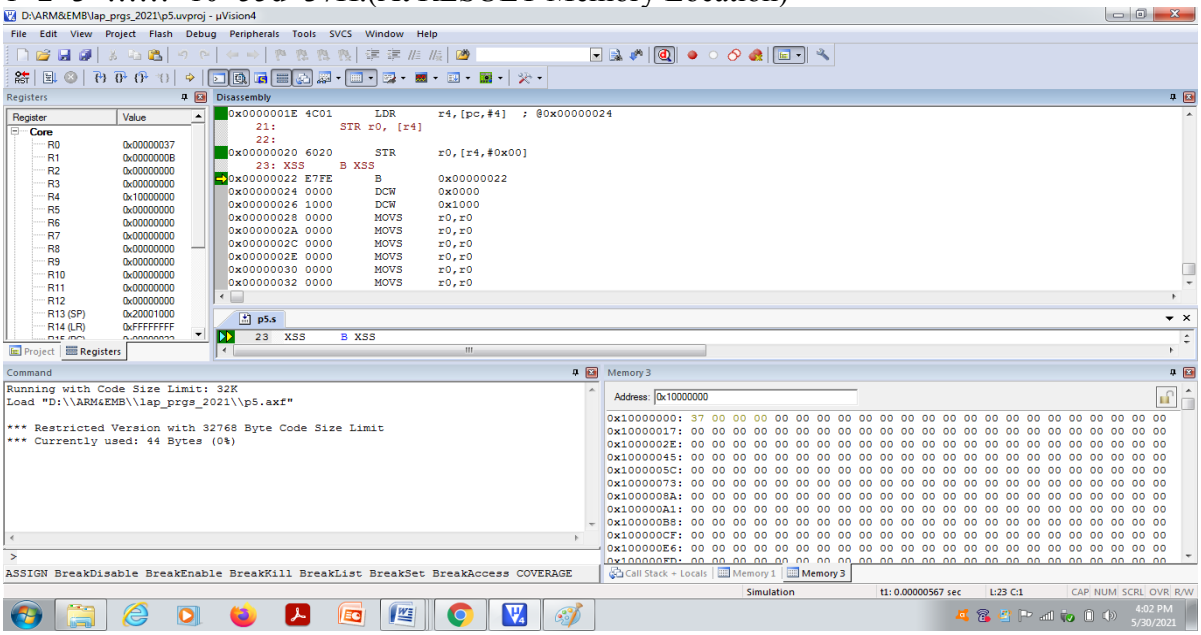
2.ALP TO FIND THE SUM OF FIRST 10 INTEGERS

```
        AREA Reset,  DATA,  READONLY
          EXPORT  __Vectors
__Vectors
        DCD 0X20001000
        DCD  Reset_Handler;

        AREA  SUM, CODE, READONLY
ENTRY
        EXPORT  Reset_Handler
Reset_Handler
        MOV r3,#10
        MOV r0,#0
        MOV r1,#1
l1        ADD r0,r0,r1
        ADD r1,r1,#1
        SUBS r3,#1
        BNE l1

        LDR r4, =RESULT
        STR r0, [r4]

XSS     B XSS

        AREA DATA2, DATA, READWRITE
```

```
RESULT  DCD 0X0
        END                          ;Mark the end
```

Result:

1+2+3+……+10=55d=37H.(At RESULT Memory Location)

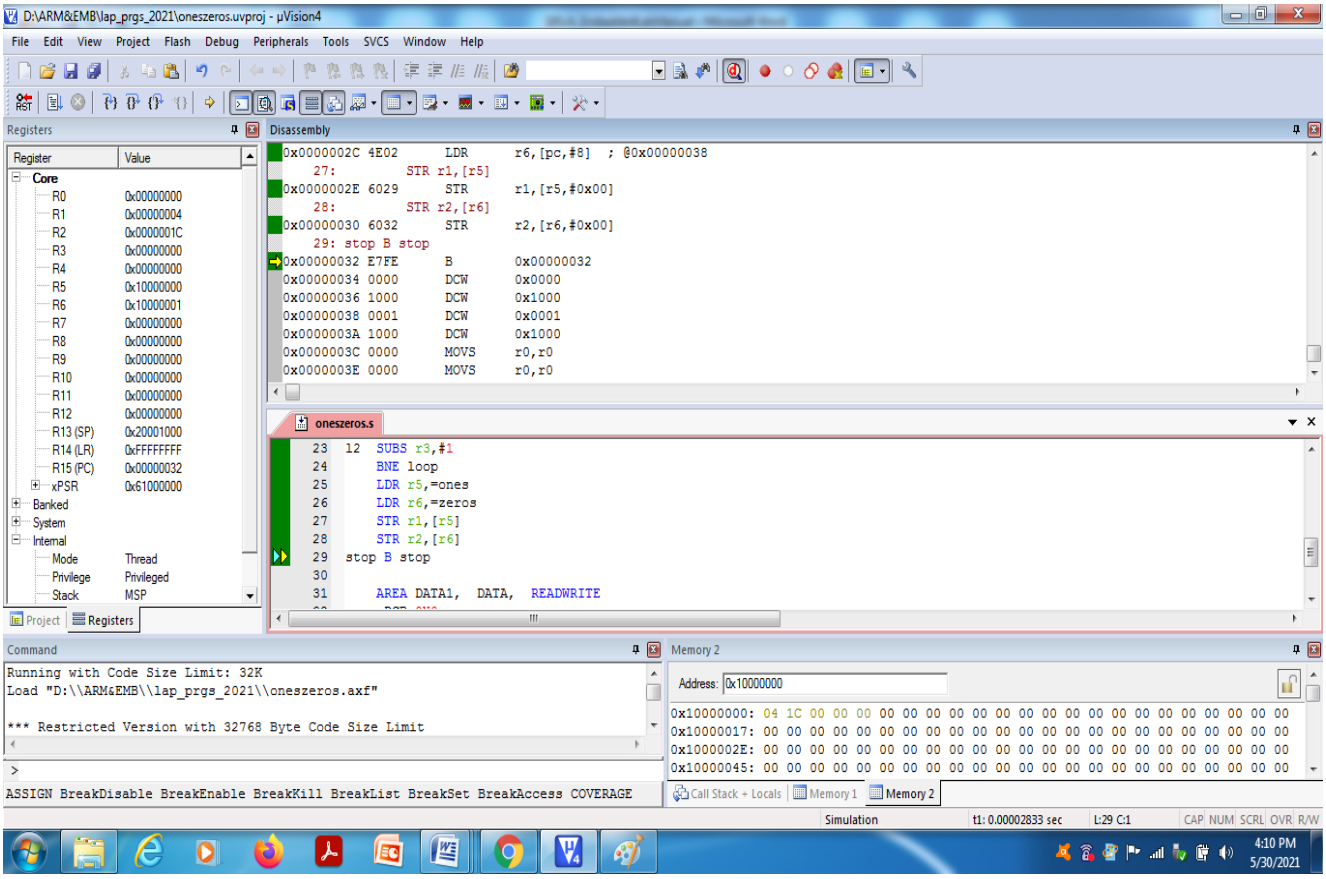


3.ALP TO FIND THE 1'S AND 0' IN THE GIVEN 32 BIT DATA.

```
        AREA Reset,  DATA,  READONLY
          EXPORT  __Vectors
__Vectors
         DCD 0X20001000
         DCD  Reset_Handler;

        AREA  onezero, CODE, READONLY
num EQU 15

ENTRY
        EXPORT  Reset_Handler
Reset_Handler
        MOV r0,#num
        MOV r1,#0
        MOV r2,#0
        MOV r3,#32
loop LSRS r0,r0,#1
        BCS l1
        ADD r2,#1
        B l2
l1  ADD r1,#1
l2      SUBS r3,#1
        BNE loop
```

```
        LDR r5,=ones
        LDR r6,=zeros
        STR r1,[r5]
        STR r2,[r6]
stop B stop

        AREA DATA1,  DATA,  READWRITE
ones DCB 0X0
zeros DCB 0X0
         END
```

Result:
If num=15d→no of 1's=4 and No.of 0's=28d=1Ch.

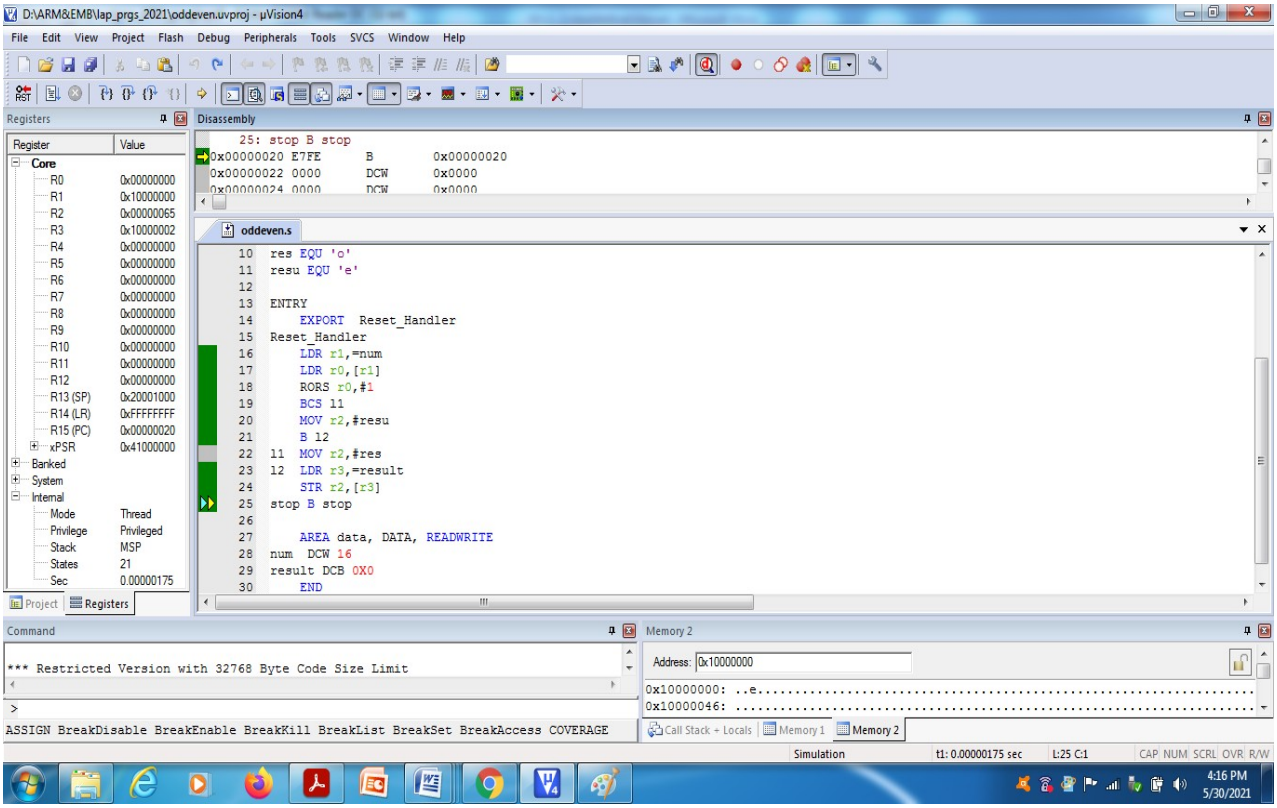

D:\ARM&EMB\lap_prgs_2021\oneszeros.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Core
R0 0x00000000
R1 0x00000004
R2 0x0000001C
R3 0x00000000
R4 0x00000000
R5 0x10000000
R6 0x10000001
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x20001000
R14 (LR) 0xFFFFFFFF
R15 (PC) 0x00000032
xPSR 0x61000000
Banked
System
Internal
Mode Thread
Privilege Privileged
Stack MSP
Project Registers
Disassembly
0x0000002C 4E02 LDR r6,[pc,#8] ; @0x00000038
27: STR r1,[r5]
0x0000002E 6029 STR r1,[r5,#0x00]
28: STR r2,[r6]
0x00000030 6032 STR r2,[r6,#0x00]
29: stop B stop
0x00000032 E7FE B 0x00000032
0x00000034 0000 DCW 0x0000
0x00000036 1000 DCW 0x1000
0x00000038 0001 DCW 0x0001
0x0000003A 1000 DCW 0x1000
0x0000003C 0000 MOVS r0,r0
0x0000003E 0000 MOVS r0,r0
oneszeros.s
23 12 SUBS r3,#1
24 BNE loop
25 LDR r5,=ones
26 LDR r6,=zeros
27 STR r1,[r5]
28 STR r2,[r6]
29 stop B stop
30
31 AREA DATA1, DATA, READWRITE
Command
Running with Code Size Limit: 32K
Load "D:\\ARM&EMB\\lap_prgs_2021\\oneszeros.axf"
*** Restricted Version with 32768 Byte Code Size Limit
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE
Memory 2
Address: 0x10000000
0x10000000: 04 1C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10000017: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1000002E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10000045: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Call Stack + Locals Memory 1 Memory 2
Simulation t1: 0.00002833 sec L:29 C:1 CAP NUM SCRL OVR R/W
4:10 PM
5/30/2021

4. ALP TO FIND WHETHER THE GIVEN 16 BIT NUMBER IS ODD OR EVEN

```
        AREA Reset,  DATA,  READONLY
          EXPORT  __Vectors
__Vectors
         DCD 0X20001000
         DCD  Reset_Handler;

         AREA  oddeven, CODE, READONLY

res  EQU   'o'
resu  EQU  'e'

ENTRY
         EXPORT  Reset_Handler
Reset_Handler
         LDR r1,=num
         LDR r0,[r1]
         RORS r0,#1
         BCS l1
         MOV r2,#resu
         B l2
l1       MOV r2,#res
l2  LDR r3,=result
         STR r2,[r3]
stop B stop

         AREA data, DATA, READWRITE
num  DCW 16
result DCB 0X0
         END
```

Result:

num=16d.Hence it is EVEN



5. ALP TO MOVE A BLOCK OF DATA FROM CODE TO RAM MEMORY

Method1:

```
AREA Reset,  DATA,  READONLY
          EXPORT  __Vectors
__Vectors
         DCD 0X20001000
         DCD  Reset_Handler;

         AREA  writedata, CODE, READONLY
src DCD 0x11,0X22,0X33,0X44,0X55
ENTRY
         EXPORT  Reset_Handler
Reset_Handler
                  LDR r0,=src
                  LDR r1,=dst
                  MOV r2,#5
l1       LDR r3,[r0],#4
                  STR r3,[r1],#4
                  SUBS r2,#1
                  BNE l1

stop     B  stop


         AREA data,  DATA,READWRITE
dst      DCD   0X0
         END
```

Method 2:

;ALP TO MOVE A BLOCK OF DATA FROM CODE TO RAM MEMORY-USING LDM and STM INSTRUCTIONS(MULTIPLE DATA TRANSFER)

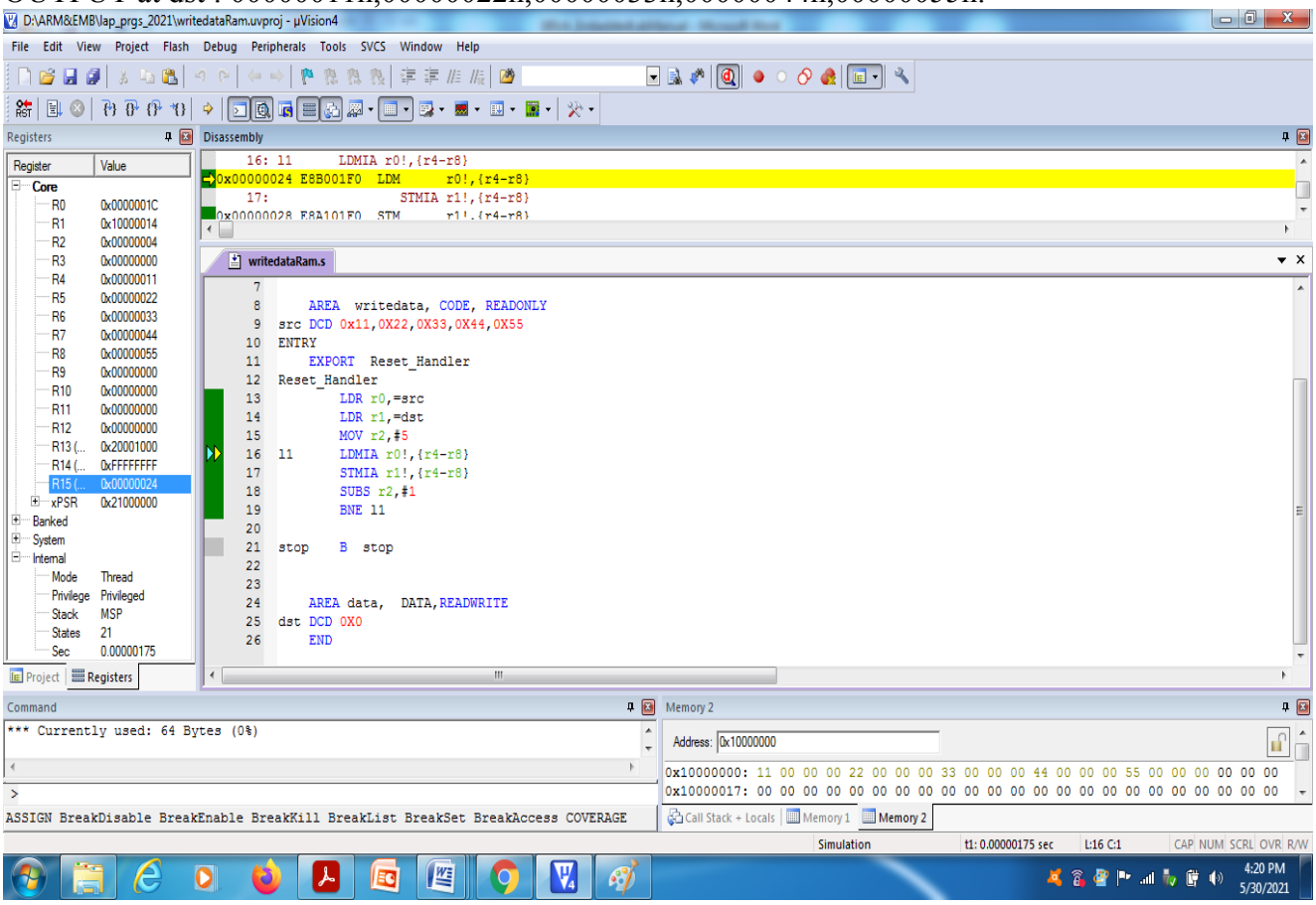
```
         AREA Reset,  DATA,  READONLY
           EXPORT  __Vectors
__Vectors
         DCD 0X20001000
         DCD  Reset_Handler;

         AREA  writedata, CODE, READONLY
src DCD 0x11,0X22,0X33,0X44,0X55
ENTRY
         EXPORT  Reset_Handler
Reset_Handler
```

```
        LDR r0,=src
        LDR r1,=dst
        MOV r2,#5
l1      LDMIA r0!,{r4-r8}
        STMIA r1!,{r4-r8}
        SUBS r2,#1
        BNE l1

stop    B  stop


        AREA data,  DATA,READWRITE
dst     DCD   0X0
        END
```

Result:

INPUT: 00000011h,00000022h,00000033h,00000044h,00000055h.

OUTPUT at dst : 00000011h,00000022h,00000033h,00000044h,00000055h.



PART B

(INTERACING HARDWARE WITH LPC 1768 MICROCONTROLLER)

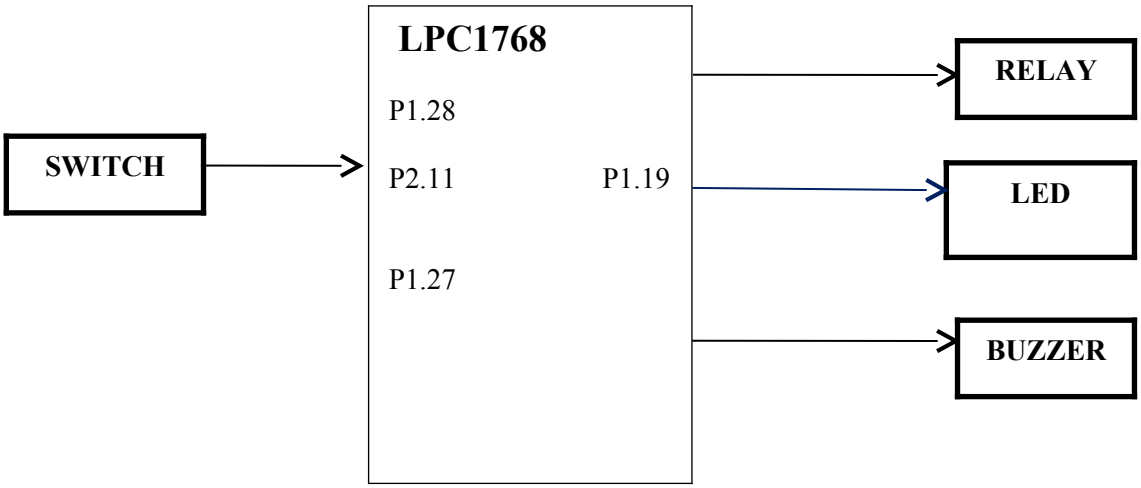
SOFTWARE: Keil µVision 4, Flash Magic

Language: Embedded C

Exp.No:1: (Beyond Syllabus-Practice session):

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

Connection details:



Algorithm:

1. Configure PORT 1 and PORT 2 as GPIO.
2. Configure the direction of Port 2 as input and Port 1 as output .
3. Read the status of the switch.
4. If the switch is pressed, turn on LED,Relay and Buzzer else turn them off.
5. Repeat from step 3 unconditionally.

Program:

```
#include<LPC17xx.h>
#define switch 11
#define LED 19
#define relay 28
#define buzzer 27
int main(void)
{
LPC_PINCON->PINSEL3=0x00000000;
LPC_PINCON->PINSEL4=0x00000000;
LPC_GPIO2->FIODIR=0x00000000;
LPC_GPIO1->FIODIR=0xFFFFFFFF;
LPC_GPIO1->FIOCLR=0xFFFFFFFF;
while(1)
 {
if (!((LPC_GPIO2->FIOPIN>>switch)& 0x1))
{
LPC_GPIO1->FIOPIN=(1<<LED)|(1<<relay)|(1<<buzzer);
```

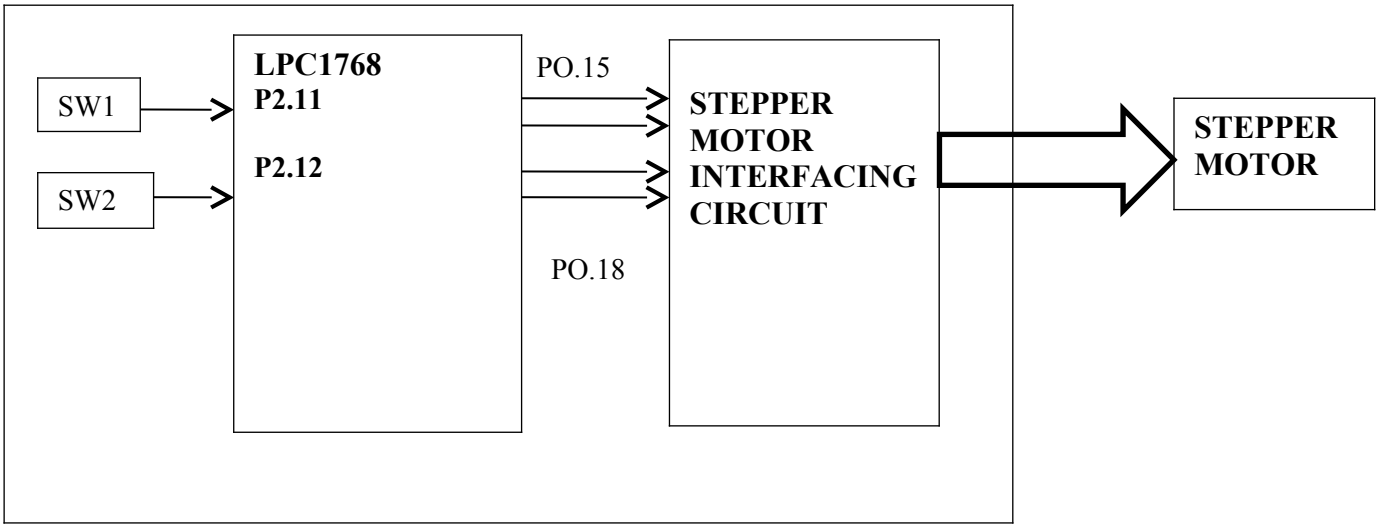
```
}
else
{
LPC_GPIO1->FIOPIN=(0<<LED)|(0<<relay)|(0<<buzzer);

}
  }
}
```

Exp.No:2

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

Connection Details:

LPC1768 TRAINER KIT



1. Configure the Port 0 and Port 2 as GPIO.
2. Configure the Port 2 in input direction and Port 0 in output direction.
3. Read the status of the switch 1. If it is pressed, set the direction as 0 for clock wise rotation.
4. Else read the status of switch 2. If it is pressed, set the direction as 1 for anticlock wise rotation.
5. If the direction is 0, send the data to energize the stepper motor coils in a sequence A-B-C-D else in D-C-B-A sequence.
6. Insert an appropriate delay between energizing two consecutive coils.
7. Repeat from steps 3 unconditionally.

Program:

```
#include<lpc17xx.h>
#define SW1 11
#define SW2 12
void delay(unsigned int x)
{
unsignedinti,j;
for(i=0;i<x;i++)
{
for(j=0;j<90000;j++);
}
}
int main(void)
```

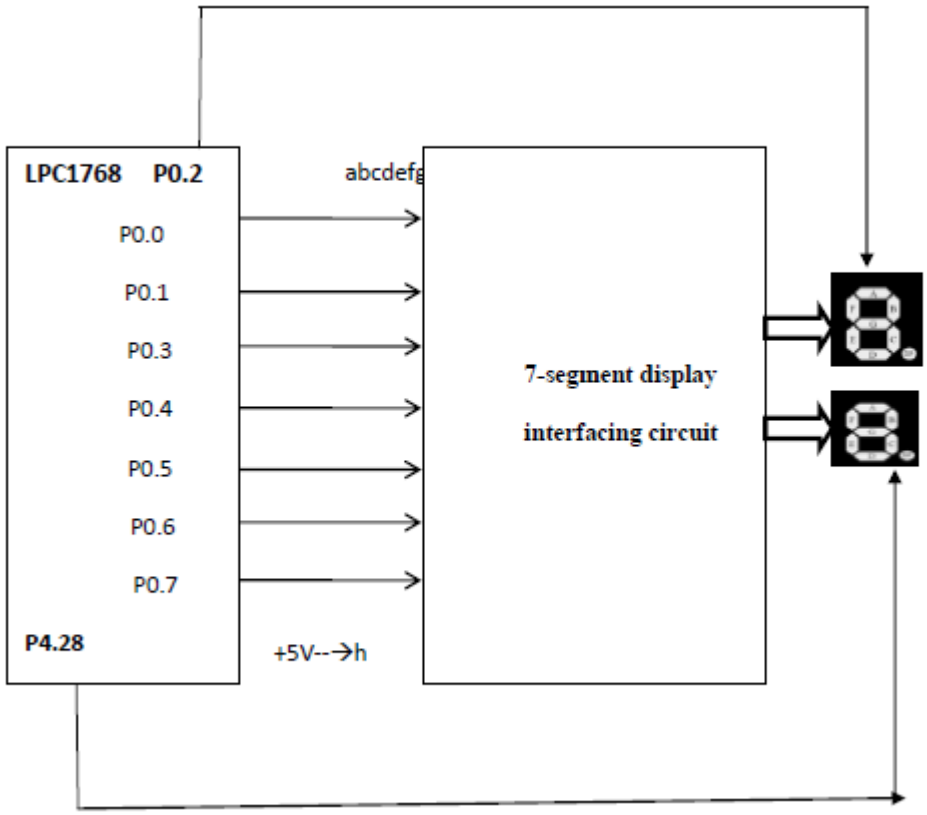
```
{
unsignedint direct;
LPC_PINCON->PINSEL0=0X00000000;
LPC_PINCON->PINSEL1=0X00000000;
LPC_PINCON->PINSEL4=0X00000000;
LPC_GPIO0->FIODIR=0xFFFFFFFF;
LPC_GPIO2->FIODIR=0X00000000;
LPC_GPIO0->FIOCLR=0X00078000;// CLEAR P0.15 TO p0.18
while(1)
  {
if(!((LPC_GPIO2->FIOPIN>>SW1)& 0X1))
{
while(!((LPC_GPIO2->FIOPIN>>SW1) & 0X1));
direct=1;
}
else if(!((LPC_GPIO2->FIOPIN>>SW2) & 0X1))
{
while(!((LPC_GPIO2->FIOPIN>>SW2) & 0X1));
direct=0;
}
if(direct==1)
{
LPC_GPIO0->FIOPIN=0X00008000;
delay(15);
LPC_GPIO0->FIOPIN=0X00010000;
delay(15);
LPC_GPIO0->FIOPIN=0X00020000;
delay(15);
LPC_GPIO0->FIOPIN=0X00040000;
delay(15);
}
else
{
LPC_GPIO0->FIOPIN=0X00040000;
delay(15);
LPC_GPIO0->FIOPIN=0X00020000;
delay(15);
LPC_GPIO0->FIOPIN=0X00010000;
delay(15);
LPC_GPIO0->FIOPIN=0x00008000;
```

```
delay(15);
}
}
 }
```

Exp.No:3

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

Connection Details:



ALGORITHM:

1. Configure port 0 and Port 4 as GPIO.
2. Configure the direction of Port 0 and Port 4 as output.
3. Create a look up table containing 7 segment equivalent code for the digits 0 to 9 and hexa digits A to F.
4. Select the display unit. Send logic 1 to Port line P0.2 for display unit 1 or to Port 4.28 for display unit 2.
5. Send each 7 segment equivalent code taken from look up table toPort 0.0 to Port line P0.7 with appropriate delay in between.
6. Clear the selected display before sending the next data to display and insert a delay.
7. Repeat from step 4.

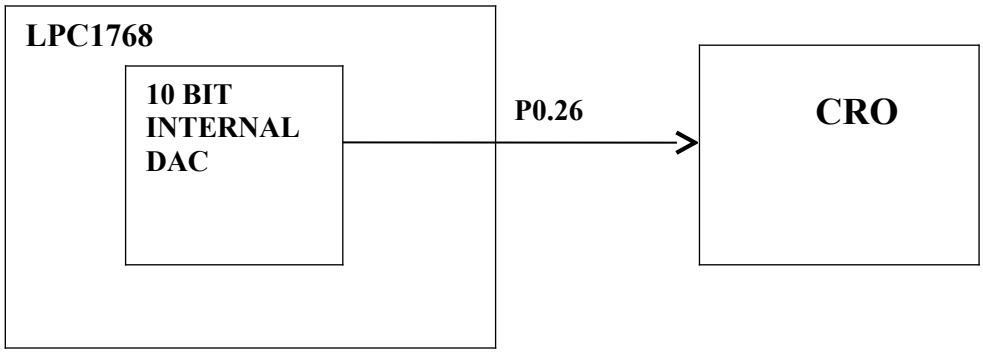
PROGRAM:

```
#include<lpc17xx.h>
```

```
void delay (unsigned int x)
{
unsignedinti,j;
for(i=0;i<x;i++)
{
for(j=0;j<90000;j++);
}
}
int main(void)
{
unsignedint k;
unsignedint a[]={0x104,0x1E5,0x094,0x0c4,
                 0x065,0x046,0x006,0xE4,
                 0x004,0x064,0x024,0x004,
                 0x116,0x104,0x016,0x36};
LPC_PINCON->PINSEL0=0X0000000;
LPC_PINCON->PINSEL9=0X0000000;
LPC_GPIO0->FIODIR=0XFFFFFFFF;
LPC_GPIO4->FIODIR=0XFFFFFFFF;
LPC_GPIO0->FIOPIN=0X1F7;
LPC_GPIO4->FIOPIN=0X10000000;
while(1)
{
for(k=0;k<16;k++)
{
LPC_GPIO0->FIOPIN=a[k];
delay(80);
LPC_GPIO0->FIOPIN=0X1F7;
delay(80);
}
}
}
```

Exp.No: 4

Interface a DAC and generate Triangular and Square waveforms.
Connection Details:



It is a string DAC consisting of 2^N resistors in series where N = no. of bits LPC176x DAC has only 1 output pin, referred to as **AOUT**. The Analog voltage at the output of this pin is given as:

$$V_{AOUT} = \frac{VALUE * (V_{REFP}-V_{REFN})}{1024} + V_{REFN}$$

When we have $V_{REFN} = 0$, the equation boils down to:

$$V_{AOUT} = \frac{VALUE * V_{REFP}}{1024}$$

Where VALUE is the 10-bit digital value which is to be converted into its Analog counterpart and V_{REF} is the input reference voltage.

Pins relating to LPC1768 DAC block:

Pin Description

AOUT (P0.26) Analog Output pin. Provides the converted Analog signal which is referenced to VSSA i.e. the Analog GND. Set Bits[21:20] in PINSEL1 register to "10" to enable this function.

DACR register Format: (32 bit register):

D/A Converter Register (DACR - 0x4008 C000)

This read/write register includes the digital value to be converted to analog, and a bit thattrades off performance vs. power. Bits 5:0 are reserved for future, higher-resolution D/Aconverters.

Bit	Symbol	Value	Description	Reset Value
5:0	-	Reserved	User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:6	VALUE	10 bit data	the voltage onthe AOUT pin is VALUE × ((VREFP - VREFN)/1024) + VREFN.	0
16	BIAS	0	The settling time of the DAC is 1 μs max, and the maximum current is 700 μA. This allows a maximum update rate of 1 MHz.	0
31:17	-	Reserved	User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

ALGORITHM:

TRIANGULAR WAVEFORM:

1. Configure the Port 0.26 as second alternate function to carry the analog output by using the PINSEL1 register.
2. Initialize 10 bit digital dataas 0.
3. Send the digital data to the DACR[6:15].
4. Increment the digital data and check whether it is equal to 1024_{10}.
5. If it is less than 1024_{10}, repeat from step 3.
6. Else decrement the digital value and send to the DACR[6:15]
7. Check whether the digital data is greater than 0.
8. if yes, repeat from step 6 else repeat from step 3.

PROGRAM FOR TRIANGULAR WAVEFORM:

```
#include<lpc17xx.h>
 #define p0_26 21
 #define ddata 6
uint32_tdacv = 0x0;

int main()
 {
    SystemInit();
    LPC_PINCON -> PINSEL1 = (1<<p0_26);
    while(1)
    {
        while(1)
        {
                dacv++;
```

```
                LPC_DAC->DACR=(dacv<<ddata);
                if(dacv>=0x3FF)
                {
                        break;
                }
        }
        while(1)
        {
                dacv--;
                LPC_DAC->DACR=(dacv<<ddata);
                if(dacv<=0x0)
                {
                        break;
                }
        }
    }
}
```

SQUARE WAVEFORM:

1. Configure the Port 0.26 as second alternate function to carry the analog output by using the PINSEL1 register.
2. Initialize 10 bit digital dataas 0.
3. Send the digital data to the DACR[6:15].
4. Insert the required delay.
5. Send the digital data equivalent to the required analog signal amplitude.(it should be less than 3.3v or 0x3ff).
6. Insert the same delay as used in step 4.
7. Repeat from step 2 unconditionally.

Program:

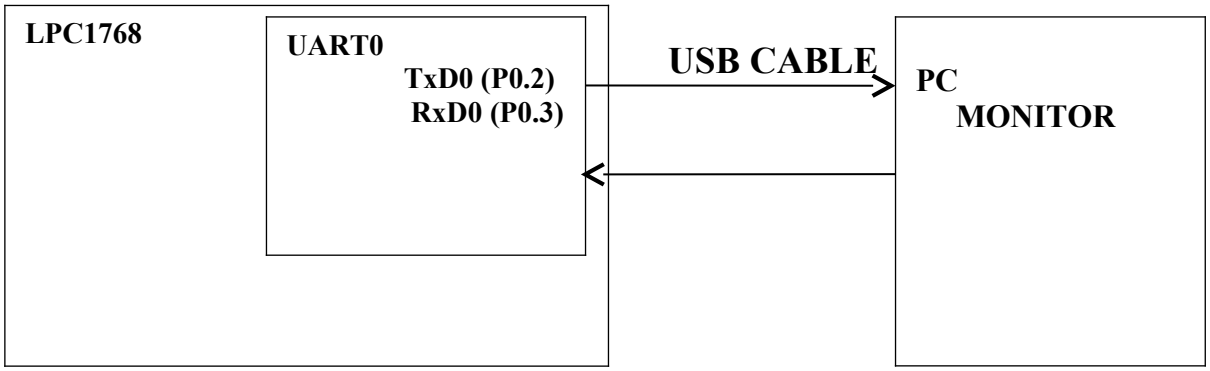
```
#include<lpc17xx.h>
 #define p0_26 21
 #define ddata 6
uint32_tdacv = 0x0;
void delay(unsigned int x)
 {
        unsignedinti,j;
        for (i=0;i<x;i++)
        {
```

```
            for(j=0;j<9000;j++);
        }
 }
int main()
 {
        SystemInit();
        LPC_PINCON -> PINSEL1 = (1<<p0_26);
        while(1)
        {
                while(1)
                {
                        dacv = 0x0;
                        LPC_DAC->DACR=(dacv<<ddata);
                        delay (15);
                dacv = 0x3ff;
                  LPC_DAC ->DACR =(dacv<<ddata);
                delay(15);
                 }
         }
}
```

Exp.No:5:

Display "Hello World" message using Internal UART.

Connection Details:



UART Registers:

The below table shows the registers associated with LPC1768 UART.

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.

UART Register formats or configuration:

FCR (FIFO Control Register):

LPC1768 has inbuilt 16byte FIFO for Receiver/Transmitter. Thus it can store 16-bytes of data received on UART without overwriting. If the data is not read before the Queue(FIFO) is filled then the new data will be lost and the OVERRUN error bit will be set.

FCR

31:8	7:6	5:4	3	2	1	0
RESERVED	**RX TRIGGER**	**RESERVED**	**DMA MODE**	**TX FIFO RESET**	**RX FIFO RESET**	**FIFO ENABLE**

Bit 0 – FIFO: This bit is used to enable/disable the FIFO for thedata received/transmitted. 0--FIFO is Disabled, 1--FIFO is Enabled for both Rx and Tx.

Bit 1 – RX_FIFO: This is used to clear the 16-byte Rx FIFO.0-No impact.

1-Clears the 16-byte Rx FIFO and the resets the FIFO pointer.

Bit 2 – Tx_FIFO: This is used to clear the 16-byte Tx FIFO.
0-No impact.
1-Clears the 16-byte Tx FIFO and the resets the FIFO pointer.

Bit 3 – DMA_MODE:
This is used for Enabling/Disabling DMA mode.
0--Disables the DMA.
1--Enables DMA only when the FIFO(bit-0) bit is SET.

Bit 7:6 – Rx_TRIGGER: This bit is used to select the number of bytes of the receiver data to be written so as to enable the interrupt/DMA.
00-- Trigger level 0 (1 character or 0x01)
01-- Trigger level 1 (4 characters or 0x04)
10-- Trigger level 2 (8 characters or 0x08)
11-- Trigger level 3 (14 characters or 0x0E)

LCR (Line Control Register):

This register is used for defining the UART frame format ie. Number of Data bits, STOP bits etc.

Format:

31:8	7	6	5:4	3	2	1:0
Reserved	DLAB	Break COntrol	Parity Select	Parity Enable	Stop Bit Select	Word Length Select

Bit 1:0 : Word Length Select: These two bits are used to select the character length
00-- 5-bit character length
01-- 6-bit character length
10-- 7-bit character length
11-- 8-bit character length

Bit 2 – Stop Bit Selection: This bit is used to select the number(1/2) of stop bits
0-- 1 Stop bit
1-- 2 Stop Bits

Bit 3 – Parity Enable: This bit is used to Enable or Disable the Parity generation and checking.
0-- Disable parity generation and checking.
1-- Enable parity generation and checking.

Bit 5:4 – Parity Selection: These two bits will be used to select the type of parity.
00-- Odd parity. Number of 1s in the transmitted character and the attached parity bit will be odd.
01-- Even Parity. Number of 1s in the transmitted character and the attached parity bit will be even.
10-- Forced "1" stick parity.
11-- Forced "0" stick parity

Bit 6 – Break Control: 0-- Disable break transmission.
1-- Enable break transmission. Output pin UARTn TXD is forced to logic 0

Bit 8 – DLAB: Divisor Latch Access Bit:

This bit is used to enable the access to divisor latch.
0-- Disable access to divisor latch
1-- Enable access to divisor latch

LSR (Line Status Register):

The is a read-only register that provides status information of the UART TX and RX blocks.

LSR Format:

31:8	7	6	5	4	3	2	1	0
Reserved	RXFE	TEMT	THRE	BI	FE	PE	OE	RDR

Bit 0 – RDR: Receive Data Ready

This bit will be set when there is a received data in RBR register. This bit will be automatically cleared when RBR is empty.
0-- The UARTn receiver FIFO is empty.
1-- The UARTn receiver FIFO is not empty.

Bit 1 – OE: Overrun Error

The overrun error condition is set when the UART Rx FIFO is full and a new character is received. In this case, the UARTn RBR FIFO will not be overwritten and the character in the UARTn RSR will be lost.
0-- No overrun
1-- Buffer over run

Bit 2 – PE: Parity Error

This bit is set when the receiver detects a error in the Parity.
0-- No Parity Error
1-- Parity Error

Bit 3 – FE: Framing Error

This bit is set when there is error in the STOP bit(LOGIC 0)
0-- No Framing Error
1-- Framing Error

Bit 4 – BI: Break Interrupt

This bit is set when the RXDn is held in the spacing state (all zeroes) for one full character transmission
0-- No Break interrupt
1-- Break Interrupt detected.

Bit 5 – THRE: Transmitter Holding Register Empty

THRE is set immediately upon detection of an empty THR. It is automatically cleared when the THR is written.
0-- THR register is Empty
1-- THR has valid data to be transmitted

Bit 6 – TEMT: Transmitter Empty

TEMT is set when both UnTHR and UnTSR are empty; TEMT is cleared when any of them contain valid data.
0-- THR and/or the TSR contains valid data.
1-- THR and the TSR are empty.

Bit 7 – RXFE: Error in Rx FIFO

This bit is set when the received data is affected by Framing Error/Parity Error/Break Error.
0-- RBR contains no UARTn RX errors.

1-- RBR contains at least one RX error.

TER (Transmitter Enable register): This register is used to Enable/Disable the transmission

TER Format:

31:8	7	6-0
Reserved	TXEN	Reserved

Bit 7 – TXEN: Trsnamitter Enable

When this bit is 1, the data written to the THR is output on the TXD pin.
If this bit is cleared to 0 while a character is being sent, the transmission of that character is completed, but no further characters are sent until this bit is set again.
In other words, a 0 in this bit blocks the transfer of characters.
•Note: By default this bit will be set after Reset.

Baudrate Calculation

LPC1768 generates the baud rate depending on the values of DLM,DLL.

Baudrate = PCLK/ (16 * ((256 * DLM) + DLL) * (1+ DivAddVal/MulVal))

where, DLM=0, DLL= , (DivaddVal/MulVal)=0.

Steps for Configuring UART0

Below are the steps for configuring the UART0.

1. Configure the P0.2 and P0.3 as first alternate function UART0 function using PINSEL0 register.
2.Configure the FCR for enabling the FIFO and Reset both the Rx/Tx FIFO.
3. Configure LCR for 8-data bits, 1 Stop bit, Disable Parity and Enable DLAB.
4. Calculate the DLM,DLL values for required baudrate from PCLK.
6. Update the DLM,DLL with the calculated values(i.e DLM=0;DLL=163).
7. Finally clear DLAB to disable the access to DLM,DLL.

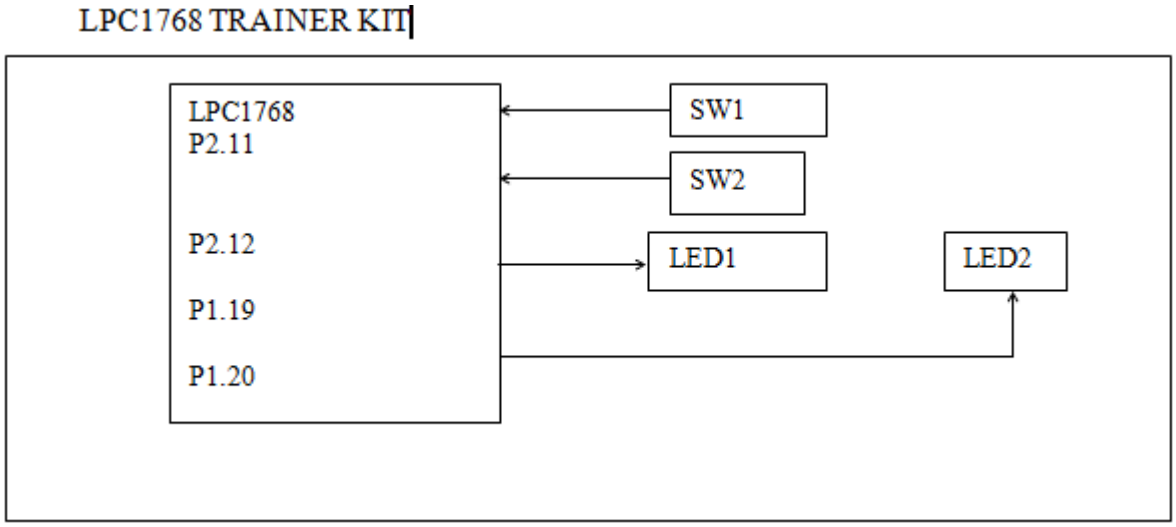
After this the UART will be ready to Transmit/Receive Data at the specified baudrate, by sending the string character by character.

Program:

```
#include<lpc17xx.h>
void U0Write( char txdata)
{
while(!(LPC_UART0->LSR & 0x20));
LPC_UART0->THR=txdata;
}

void initUART0(void)
{
LPC_PINCON->PINSEL0 =(1<<4)|(1<<6);
LPC_UART0->LCR=0x83;
LPC_UART0->DLL=163;
LPC_UART0->DLM=0;
LPC_UART0->FCR =0x7;
LPC_UART0->FDR=0x0;
LPC_UART0->LCR = 0x03;
}

int main(void)
{
charmsg[]= "Hello World";
int i=0;
initUART0();

for(i=0;msg[i];i++)

{
U0Write(msg[i]);

}
}
```

Exp.No:6:

Demonstrate the use of an external interrupt to toggle an LED On/Off.
Connection details:

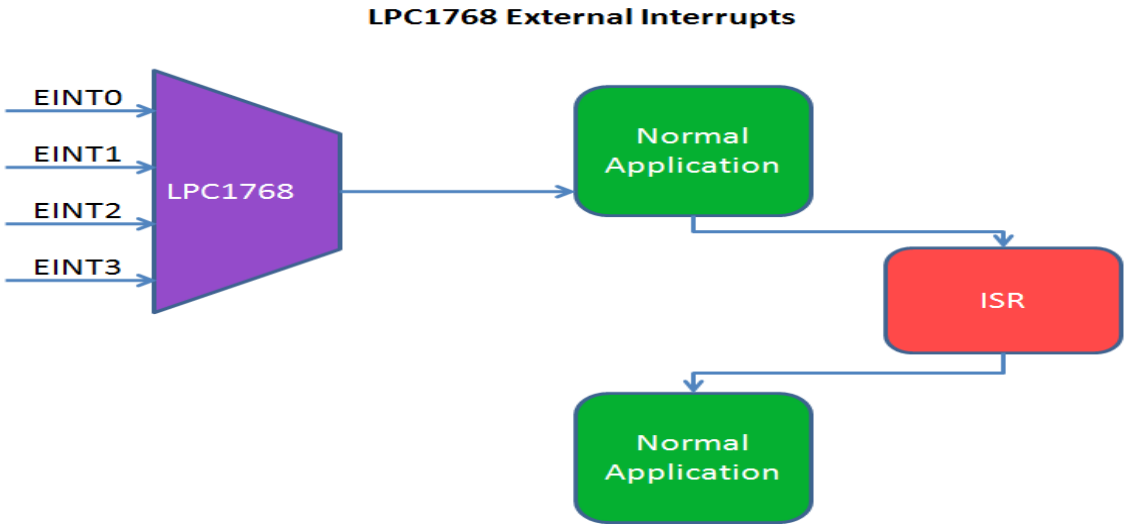


-

Description:

LPC1768 has four external interrupts EINT0-EINT3.

Port Pin	PINSEL_FUNC_0	PINSEL_FUNC_1	PINSEL_FUNC_2
P2.10	GPIO	**EINT0**	NMI
P2.11	GPIO	**EINT1**	I2STX_CLK
P2_12	GPIO	**EINT2**	I2STX_WS
P2.13	GPIO	**EINT3**	I2STX_SDA



Note:

- since the two general purpose switches have been connected with port lines P2.11 and P2.12, only two interrupts EINT1 and EINT2 have been used in this experiment.

- By pressing the switch, the corresponding interrupt signal will be generated.

EINT Registers:

Below table shows the registers associated with LPC1768 external interrupts.

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)

EXTINT Format:

31:4	3	2	1	0
RESERVED	EINT3	EINT2	EINT1	EINT0

EINTx: Bits will be set whenever the interrupt is detected on the particular interrupt pin.If the interrupts are enabled then the control goes to ISR.
Writing one to specific bit will clear the corresponding interrupt.

EXTMODE Format:

31:4	3	2	1	0
RESERVED	EXTMODE3	EXTMODE2	EXTMODE1	EXTMODE0

EXTMODEx: This bits is used to select whether the EINTx pin is level or edge Triggered.

0: EINTx is Level Triggered.
1: EINTx is Edge Triggered.

EXTPOLAR Format:

31:4	3	2	1	0
RESERVED	EXTPOLAR3	EXTPOLAR2	EXTPOLAR1	EXTPOLAR0

EXTPOLARx: This bits is used to select polarity(LOW/HIGH,FALLING/RISING) of the EINTx interrupt depending on the EXTMODE register.
0: EINTx is Active Low or Falling Edge (depending on EXTMODEx).
1: EINTx is Active High or Rising Edge (depending on EXTMODEx).

ALGORITHM:

1. Configure the pins p2.11 AND p2.12 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.
6. Define ISR1 to toggle the status of LED 1 for EINT1 and ISR2 to toggle the status of LED2 for EINT2.

PROGRAM:

```
#include <lpc17xx.h>

#define PINSEL_EINT1    22 // interrupt 1
#define PINSEL_EINT2    24 // interrupt 2

#define LED1          25  // led at p1.25
#define LED2          26  // led at p1.26

#define SBIT_EINT1     1  //extint bit 1
#define SBIT_EINT2     2  //extint bit 2

#define SBIT_EXTMODE1   1        //extint mode bit 1
#define SBIT_EXTMODE2   2        //extint mode bit 2

#define SBIT_EXTPOLAR1  1  //extint polarity mode bit 1
#define SBIT_EXTPOLAR2  2  //extint polarity mode bit 2


void EINT1_IRQHandler(void)
{
   LPC_SC->EXTINT = (1<<SBIT_EINT1);  /* Clear Interrupt Flag */
   LPC_GPIO1->FIOPIN ^= (1<< LED1);   /* Toggle the LED1 everytime INTR1 is generated */
}


void EINT2_IRQHandler(void)
{
   LPC_SC->EXTINT = (1<<SBIT_EINT2);  /* Clear Interrupt Flag */
   LPC_GPIO1->FIOPIN ^= (1<< LED2);   /* Toggle the LED2 everytime INTR2 is generated */
}
```

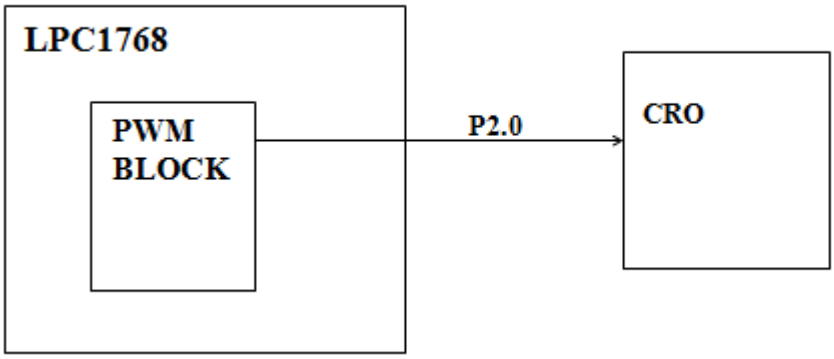
```
int main()
{
SystemInit();

   LPC_SC->EXTINT      = (1<<SBIT_EINT1)    | (1<<SBIT_EINT2);   /* Clear Pending interrupts */
   LPC_PINCON->PINSEL4 = (1<<PINSEL_EINT1)  | (1<<PINSEL_EINT2);   /* Configure
P2_11,P2_12 as EINT1/2 */
   LPC_SC->EXTMODE     = (1<<SBIT_EXTMODE1) | (1<<SBIT_EXTMODE2);
 /* Configure EINTx as Edge Triggered*/
   LPC_SC->EXTPOLAR    = (1<<SBIT_EXTPOLAR1)| (1<<SBIT_EXTPOLAR2); /* Configure
EINTx as Falling Edge */

   LPC_GPIO1->FIODIR   = (1<<LED1)  | (1<<LED2);                /* Configure LED pins as OUTPUT */
   LPC_GPIO1->FIOPIN   =  0x00;

NVIC_EnableIRQ(EINT1_IRQn);    /* Enable the EINT1,EINT2 interrupts */
NVIC_EnableIRQ(EINT2_IRQn);
while(1)
   {
     // Do nothing
   }
}
```

•

Exp.No:7 (Beyond Syllabus)

Using the Internal PWM module of ARM controller generate PWM and vary itsduty cycle.

Connection Details:



LPC1768 has 6 PWM output pins which can be used as 6-Single edged or 3-Double edged. There as seven match registers to support these 6 PWM output signals. Below block diagram shows the PWM pins and the associated Match(Duty Cycle) registers.

PWM channel 1(Port Line P2.0) has been connected to the PWM output point in a trainer Kit.So configure the P2.0 as a first alternate function to carry the PWM channel 1output using PINSEL4 register.

LPC1768 PWM Registers

The below table shows the registers associated with LPC1768 PWM Module.

Register	Description
IR	Interrupt Register: The IR can be read to identify which of eight possible interrupt sources are pending. Writing Logic-1 will clear the corresponding interrupt.
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.
PC	Prescale Counter: The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented.
MCR	Match Control Register: The MCR is used to control the reseting 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 holds 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.

Register Configuration

The below table shows the registers associated with LPC1768 PWM.

TCR				
31:4	3	2	1	0
Reserved	PWM Enable	Reserved	Counter Reset	Counter Enable

Bit 0 – Counter Enable

This bit is used to Enable or Disable the PWM Timer and PWM Prescalar Counters

0- Disable the Counters

1- Enable the Counter incrementing.

Bit 1 – Counter reset

This bit is used to clear the PWM Timer and PWM Prescalar Counter values.

0- Do not Clear.

1- The PWM Timer Counter and the PWM Prescale Counter are synchronously reset on the next positive edge of PCLK.

Bit 3 – PWM Enable

Used to Enable or Disable the PWM Block.

0- PWM Disabled

1- PWM Enabled

MCR

31:21	20	19	18	- 5	4	3	2	1
Reserved	PWMMR6S	PWMMR6R	PWMMR6I	-	PWMMR1S	PWMMR1R	PWMMR1I	PWMMR0S	PWMMR0R

PWMMRxI

This bit is used to Enable or Disable the PWM interrupts when the PWMTC matches PWMMRx (x:0-6)

0- Disable the PWM Match interrupt

1- Enable the PWM Match interrupt.

PWMMRxR

This bit is used to Reset PWMTC whenever it Matches PWMRx(x:0-6)

0- Do not Clear.

1- Reset the PWMTC counter value whenever it matches PWMRx.

PWMMRxS

This bit is used to Stop the PWMTC,PWMPC whenever the PWMTC matches PWMMRx(x:0-6).

0- Disable the PWM stop o match feature

1- Enable the PWM Stop feature. This will stop the PWM whenever the PWMTC reaches the Match register value.

PCR

31:15	14-9	8-7	6-2	1-0
Unused	PWMENA6-PWMENA1	Unused	PWMSEL6-PWMSEL2	Unused

PWMSELx

This bit is used to select the single edged and double edge mode form PWMx (x:2-6)

0- Single Edge mode for PWMx

1- Double Edge Mode for PWMx.

PWMENAx

This bit is used to enable/disable the PWM output for PWMx(x:1-6)
0- PWMx Disable.
1- PWMx Enabled.

LER

31-7	6	5	4	3	2	1	0
Unused	LEN6	LEN5	LEN4	LEN3	LEN2	LEN1	LEN0

LENx
This bit is used Enable/Disable the loading of new Match value whenever the PWMTC is reset(x:0-6)
PWMTC will be continously incrementing whenever it reaches the PWMMRO, timer will be reset depeding on PWMTCR configuraion. Once the Timer is reset the New Match values will be loaded from MR0-MR6 depending on bits set in this register.
0- Disable the loading of new Match Values
1- Load the new Match values from MRx when the timer is reset.

PWM Working
The TC is continuously incremented and once it matches the MR1(Duty Cycle) the PWM pin is pulled Low. TC still continues to increment and once it reaches the Cycle time(Ton+Toff) the PWM module does the following things:

- Reset the TC value.
- Pull the PWM pin High.
- Loads the new Match register values.

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. Here, it will be 100.
6. Load the Duty cycles for required PWM1 channel in respective match register MR1.
7. Enable the bits in LER register to load and latch the new match values.
8. Enable the pwm channel 1 in PCR register.

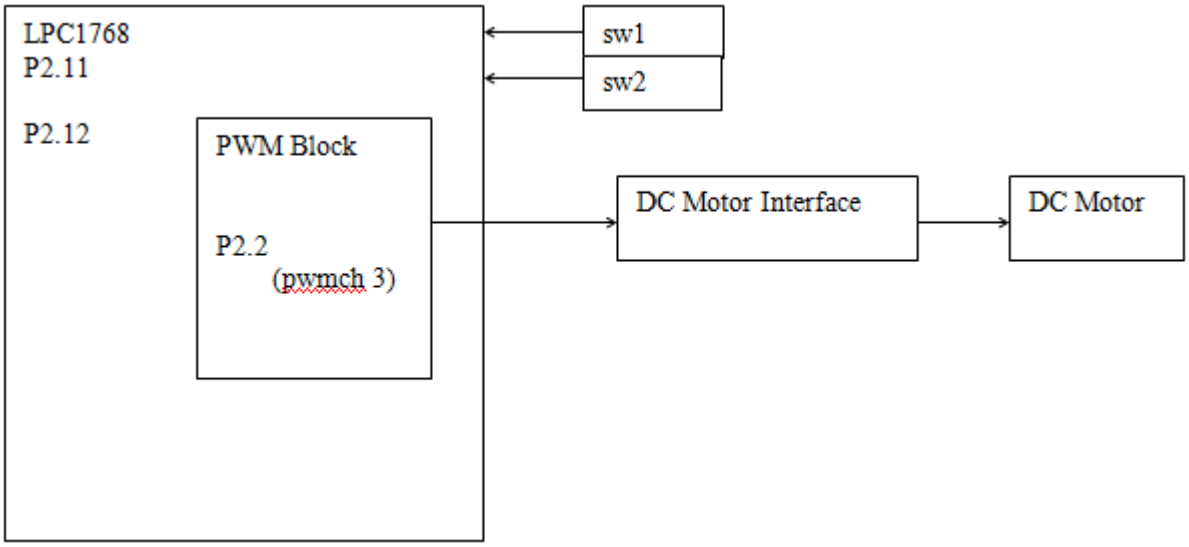
PROGRAM:

```
#include<lpc17xx.h>
void delay(unsigned int k)
{
unsignedinti,j;
for(i=0;i<k;i++)
for(j=0;j<60000;j++);
}
#define SBIT_CNTEN 0
#define SBIT_PWMEN 2
#define SBIT_PWMMR0R 1
```

```
#define SBIT_PWMENA1 9
#define PWM_1 0
int main()
{
int dc;
SystemInit();
LPC_PINCON->PINSEL4=(1<<PWM_1);
LPC_PWM1->TCR=(1<<SBIT_CNTEN)|(1<<SBIT_PWMEN);
LPC_PWM1->PR=0x00;
LPC_PWM1->MCR=(1<<SBIT_PWMMR0R);
LPC_PWM1->MR0=100;
LPC_PWM1->PCR=(1<<SBIT_PWMENA1);
while(1)
{
for(dc=0;dc<100;dc++)
{
LPC_PWM1->MR1=dc;
delay(5);
}
for(dc=100;dc>0;dc--)
{
LPC_PWM1->MR1=dc;
delay(5);
}
```

EXP.NO:8:

Interface and Control a DC Motor.
Connection details:



- Use all the register configuration as used in PWM experiment .
- Use PWM channel 3(P2.2) instead of PWM channel 1(P2.0).

Algorithm:

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. Here, it will be 100.
6. Load the Duty cycles for required PWM3 channel in respective match register MR3.
7. Enable the bits in LER register to load and latch the new match values.
8. Enable the pwm channel 1 in PCR register.

steps to control the speed of DC motor:

After configuring the PWM module,

1. Read the status of the switch 1. For each press, decrease the MR3 by 10. check whether it is greater than 0.Else assume MR3 is always zero for further continuous press of Switch 1.

2. Read the status of the switch 2. For each press, increase the MR3 by 10. Check whether it is less than 100.Else assume MR3 is always 99 for further continuous press of Switch 2.

PROGRAM:

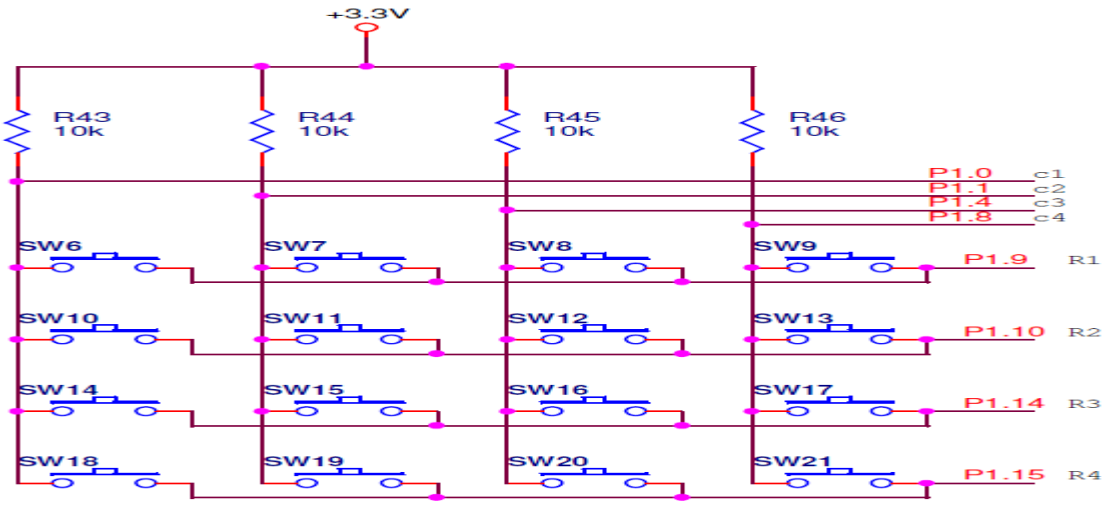
```
#include<lpc17xx.h>
#define cnten 0
#define pwnen 2
#define P2_2        4
#define MROR 1
#define pwnch3 11
#define SW1 11
#define SW2 12
void delay(unsigned int k)
{
unsignedintx,y;
for(x=0;x<k;x++)
for(y=0;y<90000;y++);
}
int main (void)
{
inti=100;
LPC_PINCON->PINSEL4=(1<<P2_2);
LPC_PWM1->TCR=(1<<cnten)|(1<<pwnen) ;
LPC_PWM1->MCR=(1<<MROR);
LPC_PWM1->PCR=(1<<pwnch3);
LPC_PWM1->PR=0x0;
LPC_PWM1->MR0=100;
while(1)
{
LPC_PWM1->MR3=i;
delay(5);
if(!((LPC_GPIO2->FIOPIN>>SW1)&0X1))
{
while(!((LPC_GPIO2->FIOPIN>>SW1)&0X1));
i=i-10;
if(i<=0)
i=0;
}
else if(!((LPC_GPIO2->FIOPIN>>SW2)&0X1))
{
while(!((LPC_GPIO2->FIOPIN>>SW2)&0X1));
i=i+10;
if(i>100)
i=99;
```

```
}
}
}
```

Exp.No: 9:

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

Connection Details:



Program:

```
#include "lpc17xx.h"
#include "lcd.h"

////////////////////////////////////////////
// Matrix Keypad Scanning Routine
//
// COL1 COL2 COL3 COL4
// 0    1    2    3    ROW 1
// 4    5    6    7    ROW 2
// 8    9    A    B    ROW 3
// C    D    E    F    ROW 4
////////////////////////////////////////////

#define COL1            0
#define COL2            1
#define COL3            4
#define COL4            8

#define ROW1                    9
#define ROW2                    10
#define ROW3                    14
#define ROW4                    15

#define COLMASK                         ((1<<COL1) |(1<< COL2) |(1<< COL3) |(1<< COL4))
#define ROWMASK                         ((1<<ROW1) |(1<< ROW2) |(1<< ROW3) |(1<< ROW4))

#define KEY_CTRL_DIR        LPC_GPIO1->FIODIR
#define KEY_CTRL_SET        LPC_GPIO1->FIOSET
```

```
#define KEY_CTRL_CLR       LPC_GPIO1->FIOCLR
#define KEY_CTRL_PIN       LPC_GPIO1->FIOPIN

/////////////// COLUMN WRITE ////////////////////
voidcol_write( unsigned char data )
{
unsignedint temp=0;

temp=(data) & COLMASK;

  KEY_CTRL_CLR |= COLMASK;
  KEY_CTRL_SET |= temp;
}


//////////////////////////////// MAIN  ////////////////////////////////////
int main (void)
{
unsigned char key, i;
unsigned char rval[] = {0x77,0x07,0x0d};
unsigned char keyPadMatrix[] =
{
   '4','8','B','F',
   '3','7','A','E',
   '2','6','0','D',
   '1','5','9','C'
};
SystemInit();
init_lcd();

  KEY_CTRL_DIR |= COLMASK;    //Set COLs as Outputs
  KEY_CTRL_DIR &= ~(ROWMASK); //  Set ROW lines as Inputs

  lcd_putstring16(0,"Press HEX Keys..");// 1st line display
  lcd_putstring16(1,"Key Pressed =   ");// 2nd line display

while (1)
 {
key = 0;
for(i = 0; i< 4; i++ )
    {
       // turn on COL output one by one
                   col_write(rval[i]);

       // read rows - break when key press detected
if (!(KEY_CTRL_PIN & (1<<ROW1)))
break;
```

```
key++;
if (!(KEY_CTRL_PIN & (1<<ROW2)))
break;

key++;
if (!(KEY_CTRL_PIN & (1<<ROW3)))
break;

key++;
                if (!(KEY_CTRL_PIN & (1<<ROW4)))
break;

key++;
   }

        if (key == 0x10)
                lcd_putstring16(1,"Key Pressed =   ");
        else
                {
                        lcd_gotoxy(1,14);
                        lcd_putchar(keyPadMatrix[key]);
                }
 }

}
```

LCD code:-

```
#include "lpc17xx.h"
#include "lcd.h"
voidLcd_CmdWrite(unsigned char cmd);
voidLcd_DataWrite(unsigned char dat);
#define LCDRS          9
#define LCDRW         10
#define LCDEN         11

#define LCD_D4  19
#define LCD_D5  20
#define LCD_D6  21
#define LCD_D7  22
#define LcdData       LPC_GPIO0->FIOPIN
#define LcdControl    LPC_GPIO0->FIOPIN
#define LcdDataDirn   LPC_GPIO0->FIODIR
#define LcdCtrlDirn   LPC_GPIO0->FIODIR
#define LCD_ctrlMask   ((1<<LCDRS)|(1<<LCDRW)|(1<<LCDEN))
```

```
#define LCD_dataMask   ((1<<LCD_D4)|(1<<LCD_D5)|(1<<LCD_D6)|(1<<LCD_D7))
void delay(unsigned int count)
{
int j=0, i=0;
for (j=0;j<count;j++)
for (i=0;i<30;i++);
}

voidsendNibble(char nibble)
{
LcdData&=~(LCD_dataMask);                 // Clear previous data
LcdData|= (((nibble >>0x00) & 0x01) << LCD_D4);
LcdData|= (((nibble >>0x01) & 0x01) << LCD_D5);
LcdData|= (((nibble >>0x02) & 0x01) << LCD_D6);
LcdData|= (((nibble >>0x03) & 0x01) << LCD_D7);
}

voidLcd_CmdWrite(unsigned char cmd)
{
sendNibble((cmd>> 0x04) & 0x0F);  //Send higher nibble
LcdControl&= ~(1<<LCDRS); // Send LOW pulse on RS pin for selecting Command register
LcdControl&= ~(1<<LCDRW); // Send LOW pulse on RW pin for Write operation
LcdControl |= (1<<LCDEN);  // Generate a High-to-low pulse on EN pin
delay(100);
LcdControl&= ~(1<<LCDEN);

delay(10000);

sendNibble(cmd& 0x0F);            //Send Lower nibble
LcdControl&= ~(1<<LCDRS); // Send LOW pulse on RS pin for selecting Command register
LcdControl&= ~(1<<LCDRW); // Send LOW pulse on RW pin for Write operation
LcdControl |= (1<<LCDEN);  // Generate a High-to-low pulse on EN pin
delay(100);
LcdControl&= ~(1<<LCDEN);

delay(1000);
}


voidLcd_DataWrite(unsigned char dat)
{
sendNibble((dat>> 0x04) & 0x0F);  //Send higher nibble
LcdControl |= (1<<LCDRS);  // Send HIGH pulse on RS pin for selecting data register
LcdControl&= ~(1<<LCDRW); // Send LOW pulse on RW pin for Write operation
LcdControl |= (1<<LCDEN);  // Generate a High-to-low pulse on EN pin
delay(100);
LcdControl&= ~(1<<LCDEN);
```

```
delay(1000);

sendNibble(dat& 0x0F);            //Send Lower nibble
LcdControl |= (1<<LCDRS);  // Send HIGH pulse on RS pin for selecting data register
LcdControl&= ~(1<<LCDRW); // Send LOW pulse on RW pin for Write operation
LcdControl |= (1<<LCDEN);  // Generate a High-to-low pulse on EN pin
delay(100);
LcdControl&= ~(1<<LCDEN);

delay(1000);
}
voidlcd_clear( void)
{
Lcd_CmdWrite( 0x01 );
}
intlcd_gotoxy( unsigned char x, unsigned char y)
{
unsigned char retval = TRUE;

if( (x > 1) && (y > 15) )
   {
retval = FALSE;
   }
else
   {
if( x == 0 ) Lcd_CmdWrite( 0x80 + y );
          else if( x==1 ) Lcd_CmdWrite( 0xC0 + y );
   }
returnretval;
}
voidlcd_putchar( unsigned char c )
{
Lcd_DataWrite( c );
}
voidlcd_putstring( char *string )
{
while(*string != '\0')
  {
lcd_putchar( *string );
string++;
  }
}
void lcd_putstring16( unsigned char line, char *string )
{
unsigned char len = 16;
```

```
lcd_gotoxy( line, 0 );
while(*string != '\0' &&len--)
  {
lcd_putchar( *string );
string++;
  }
}
voidinit_lcd( void )
  {
LcdDataDirn |= LCD_dataMask;  // Configure all the LCD pins as output
LcdCtrlDirn |= LCD_ctrlMask;


                // Initialize Lcd in 4-bit mode
Lcd_CmdWrite(0x03);
delay(2000);
Lcd_CmdWrite(0x03);
delay(1000);
Lcd_CmdWrite(0x03);
delay(100);
Lcd_CmdWrite(0x2);
Lcd_CmdWrite(0x28);
Lcd_CmdWrite(0x0e);
Lcd_CmdWrite(0x06);
Lcd_CmdWrite(0x01);
delay(1);                 // display on

}
```

Exp.No:10:

Measure Ambient temperature using a sensor and SPI ADC IC.

Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device.

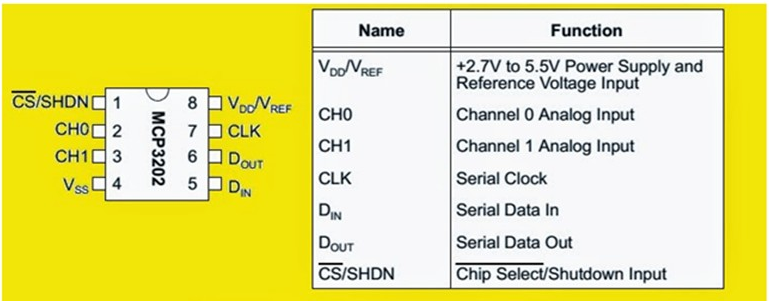
ADC (Analog to Digital Converter):

The Microchip Technology Inc. MCP3202 is a successive approximation 12-bit Analog-to-Digital (A/D) Converter with on-board sample and hold circuitry. The MCP3202 is programmable to provide a single pseudo-differential input pair or dual single-ended inputs. Differential Nonlinearity (DNL) is specified at ±1 LSB, and Integral Nonlinearity (INL) is offered in ±1 LSB (MCP3202-B) and ±2 LSB (MCP3202-C) versions. Communication with the device is done using a simple serial interface compatible with the SPI protocol. The device is capable of conversion rates of up to 100ksps at 5V and 50ksps at 2.7V.The MCP3202 is a Dual Channel 12-Bit A/D Converter with SPI Serial Interface By Microchip . In this tutorial i will interface this ADC using lpc1768 microcontroller using SPI Protocol in mode(0,0) . Maximum clock rate supported by MCP3202 is 1.8 MHz.

Fsample= 100 KSPS

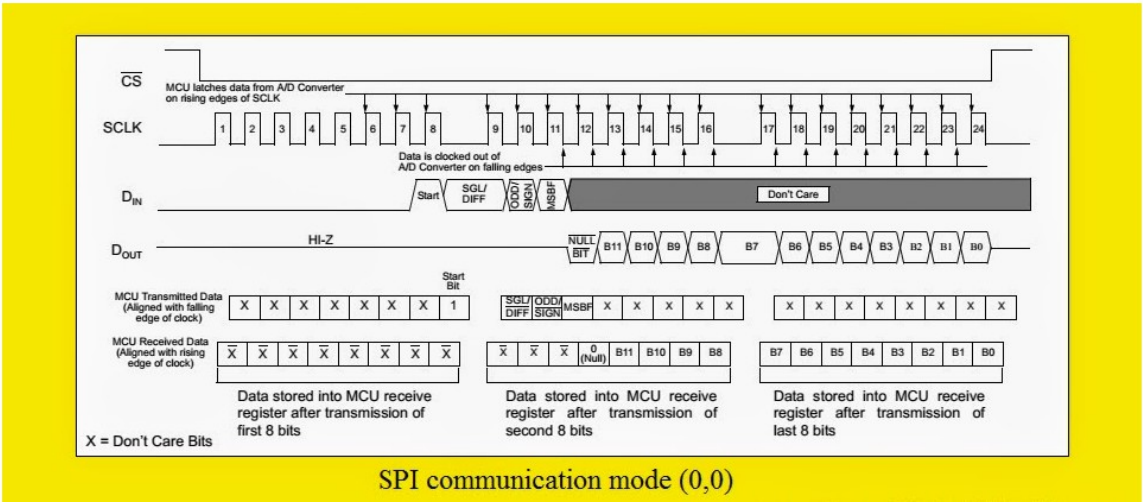
Fclk = 18*Fsample

Configuring SPI Control Register :



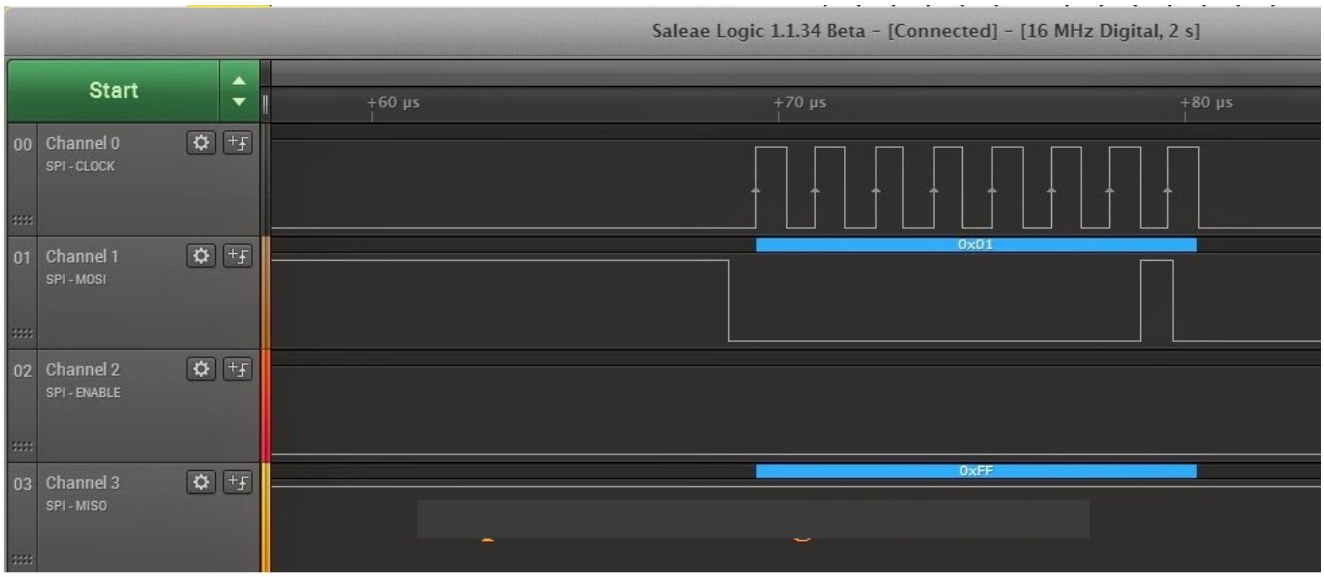
Name	Function
V_{DD}/V_{REF}	+2.7V to 5.5V Power Supply and Reference Voltage Input
CH0	Channel 0 Analog Input
CH1	Channel 1 Analog Input
CLK	Serial Clock
D_{IN}	Serial Data In
D_{OUT}	Serial Data Out
$\overline{CS}$/SHDN	Chip Select/Shutdown Input

Serial Peripheral interface allows high speed synchronous data communication betweenlpc1768microcontrollers.



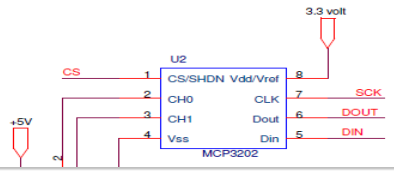
SPI communication mode (0,0)

first we need to send start bit , it is last bit of first byte we are going to send to ADC and then we have to select Configuration modes.there are two modes single ended and pseudo differential mode here choose single ended mode . MSBF bit choses order of format of byte either MSB first or LSB first herechoose MSB bit first format so MSBF=1.ransferingSecondbyte=0xA0forchannel0and Secondbyte=0xE0 for channel 1. ADC will return B11 to B8 of data in the lower nibble of byte so we need to perform some mathematical manipulation. after that we need to send any byte to receive third byte of data.



Pin Assignment with LPC1768:

	SPI - ADC	LPC1768 Lines	SPI - ADC
MCP 3202	CS	P0.28	
	CLK	P0.27	
	Dout	P3.26	

	Din	P3.25	

ALGORITHM:

PROGRAM:

```
#include <LPC17xx.H>
#include <stdint.h>
#include <stdio.h>
#include "delay.h"
#include "spi_manul.h"
#include "lcd.h"
#define pulse_val 2

main()
{
unsignedintspi_rsv=0;
float vin;
charbuf[20];
SystemInit ();
lcd_init();
lcd_str("SPI 3202-b");
delay(60000);
delay(60000);
while(1)
   {
lcd_clr();
lcd_cmd(0x80);
spi_rsv = spi_data1(15);
vin = ( ( spi_rsv& 0xfff ) * (3.3) ) / 4096  ;
sprintf(buf,"Temp: %0.2f degC",(vin*100) );
lcd_str(buf);
delay(50000);
delay(50000);
   }
}
```

SPI ADC data fetching program:-

```
#include <LPC17xx.H>
#include "delay.h"
#define pulse_val 2
#define CLK 1<<27
#define          CS       1<<28
#define DDOUT 26
#define          DOUT  1<<25
#define          DIN       1<<26
#define spi_stst 0
unsignedintspi_data(char sel)
{
charclks = 4;
  LPC_GPIO0->FIODIR |= CS|CLK;
  LPC_GPIO3->FIODIR = DOUT;

  LPC_GPIO0->FIOSET = CS|CLK;
  LPC_GPIO3->FIOCLR       = DOUT;
nop_delay(100);
#if spi_stst
if ( LPC_GPIO3->FIOPIN & DIN )
  {
return 'P';
  }
#endif
   LPC_GPIO0->FIOCLR = CS;
          nop_delay(pulse_val);
while(clks)
   {
           LPC_GPIO0->FIOCLR = CLK;
          nop_delay(pulse_val);

           LPC_GPIO3->FIOPIN = (sel& 1) << DDOUT;
          sel = sel>> 1;

           LPC_GPIO0->FIOSET = CLK;
          nop_delay(pulse_val);

          clks--;
   }
    LPC_GPIO0->FIOCLR = CLK;
          nop_delay(pulse_val);
if (!( LPC_GPIO3->FIOPIN & DIN ) )
  {
return 'U';
  }
                    clks = 12;
```

```
while(clks)
   {
clks--;
            LPC_GPIO0->FIOCLR = CLK;
           nop_delay(pulse_val);
            LPC_GPIO0->FIOSET = CLK;
           nop_delay(pulse_val);
   }
           nop_delay(pulse_val);
if (!( LPC_GPIO3->FIOPIN & DIN ) )
  {
return 'U';
  }
return 'Z';
}
unsignedint spi_data1(char sel)
{
unsignedintspi_reg=0;
charclks = 12;
  LPC_GPIO0->FIODIR |= CS|CLK;
  LPC_GPIO3->FIODIR = DOUT;

  LPC_GPIO0->FIOSET = CS|CLK;
  LPC_GPIO3->FIOSET        = DOUT;
  LPC_GPIO3->FIOPIN        = DIN;

nop_delay(100);

  LPC_GPIO0->FIOCLR = CS;
  //start condi

  LPC_GPIO0->FIOCLR = CLK;
  LPC_GPIO3->FIOSET        = DOUT;
nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
nop_delay(5);

  //single mode
  LPC_GPIO0->FIOCLR = CLK;
  LPC_GPIO3->FIOSET        = DOUT;

nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
nop_delay(5);
  //chanl 1
  LPC_GPIO0->FIOCLR = CLK;
  LPC_GPIO3->FIOSET        = DOUT;
```

```
nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
nop_delay(5);
  //msb first
  LPC_GPIO0->FIOCLR = CLK;
  LPC_GPIO3->FIOSET        = DOUT;

nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
nop_delay(5);
  //smpling
//  LPC_GPIO0->FIOCLR = CLK;
//  nop_delay(pulse_val);
//  LPC_GPIO0->FIOSET = CLK;
//  nop_delay(2);

  //null bit
  LPC_GPIO0->FIOCLR = CLK;
nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
//  while ( ( LPC_GPIO3->FIOPIN & DIN ) == DIN );
// if( !( LPC_GPIO3->FIOPIN & DIN ) );
//  {
//      return 'U';
//  }
nop_delay(5);
           clks = 12;
           while(clks)
            {
              LPC_GPIO0->FIOCLR = CLK;
nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
if( ( LPC_GPIO3->FIOPIN & DIN ) )
  {
spi_reg |= 1<<(clks-1);
  }
else
  {
          spi_reg = spi_reg;
  }
clks--;
nop_delay(5);
            }
nop_delay(1);
returnspi_reg;
}
```

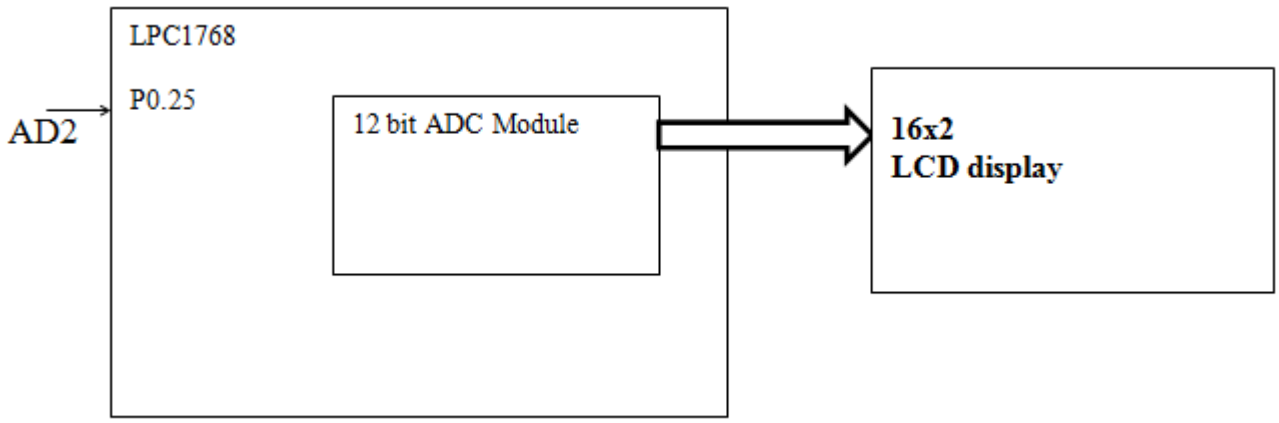
LCD display program:-

```
#include <LPC17xx.H>
#include "delay.h"
#define RRW (7<<9)
#define DATA_L (15<<19)
voidlcd_pin(void)
{
  LPC_GPIO0->FIODIR |= RRW|DATA_L;
}

voidlcd_cmd(unsigned char cmd)
{

        LPC_GPIO0->FIOPIN =( ( ( cmd& 0xf0 )>> 4) << 19 )| (1<<11);
        delay(200);
        LPC_GPIO0->FIOPIN =( ( ( cmd& 0xf0 )>> 4) << 19 );

        LPC_GPIO0->FIOCLR |= RRW|DATA_L;
        delay(10);

        LPC_GPIO0->FIOPIN = ( ( ( cmd& 0xf ) ) << 19 )| (1<<11);
        delay(200);
        LPC_GPIO0->FIOPIN = ( ( ( cmd& 0xf ) ) << 19 );

}

voidlcd_data(unsigned char cmd)
{
LPC_GPIO0->FIOPIN =(1<<9)|( ( ( cmd& 0xf0 )>> 4) << 19 )| (1<<11);
        delay(200);
        LPC_GPIO0->FIOPIN =( ( ( cmd& 0xf0 )>> 4) << 19 );

        LPC_GPIO0->FIOCLR |= RRW|DATA_L;
        delay(10);

        LPC_GPIO0->FIOPIN = (1<<9)|( ( ( cmd& 0xf ) ) << 19 )| (1<<11);
        delay(200);
        LPC_GPIO0->FIOPIN = ( ( ( cmd& 0xf ) ) << 19 );
}
voidlcd_init(void)
{
lcd_pin();
lcd_cmd(0x03);
delay(3000);
lcd_cmd(0x03);
```

```
delay(1000);
lcd_cmd(0x03);
delay(100);
lcd_cmd(0x2);
lcd_cmd(0x28);
lcd_cmd(0x0e);
lcd_cmd(0x06);
lcd_cmd(0x01);
delay(1);
}

voidlcd_str(char *lstr)
{
while(*lstr)
  {
lcd_data(*lstr);
        lstr++;
  }
}

voidlcd_clr(void)
{
        lcd_cmd(0x03);
delay(10);
lcd_cmd(0x2);
lcd_cmd(0x28);
lcd_cmd(0x0e);
lcd_cmd(0x06);
lcd_cmd(0x01);
delay(10);
}
```

Exp.No:11: (Beyond syllabus)

Interface 12 bit internal ADC to convert the analog to digital and display the same on LCD.
Connection Details:



ALGORITHM:

Steps for Configuring ADC

1. Configure the GPIO pin for ADC function using PINSEL register.
2. Enable the CLock to ADC module.
3. Deselect all the channels and Power on the internal ADC module by setting ADCR.PDN bit.
4. Select the Particular channel for A/D conversion by setting the corresponding bits in ADCR.SEL
5. Set the ADCR.START bit for starting the A/D conversion for selected channel.
6. Wait for the conversion to complete, ADGR.DONE bit will be set once conversion is over.
7. Read the 12-bit A/D value from ADGR.RESULT.
8. Use it for further processing or just display on LCD.

PROGRAM:

```
#include "lpc17xx.h"
#include "lcd.h"
#define VREF       3.3 //Reference Voltage at VREFP pin, given VREFN = 0V(GND)
#define ADC_CLK_EN (1<<12)
#define SEL_AD0_2  (1<<2) //Select Channel AD0.2
#define CLKDIV     1 //ADC clock-divider (ADC_CLOCK=PCLK/CLKDIV+1) = 12.5Mhz
@ 25Mhz PCLK
#define PWRUP      (1<<21) //setting it to 0 will power it down
#define START_CNV  (1<<24) //001 for starting the conversion immediately
#define ADC_DONE   (1U<<31) //define it as unsigned value or compiler will throw #61-D
warning
#define ADCR_SETUP_SCM ((CLKDIV<<8) | PWRUP)
//////////   Init ADC0 CH2  /////////////////
Init_ADC()
{
  // Convert Port pin 0.25 to function as AD0.2
   LPC_SC->PCONP |= ADC_CLK_EN; //Enable ADC clock
        LPC_ADC->ADCR =  ADCR_SETUP_SCM | SEL_AD0_2;
        LPC_PINCON->PINSEL1 |= (1<<18) ; //select AD0.2 for P0.25
}

//////////   READ ADC0 CH:2  /////////////////
unsignedintRead_ADC()
{
unsignedinti=0;
LPC_ADC->ADCR |= START_CNV; //Start new Conversion
                while((LPC_ADC->ADDR2 & ADC_DONE) == 0); //Wait untill conversion is
finished
                i = (LPC_ADC->ADDR2>>4) & 0xFFF; //12 bit Mask to extract result

}
//////////   DISPLAY ADC VALUE /////////////////
Display_ADC()
{
unsignedintadc_value = 0;
charbuf[4] = {5};
float voltage = 0.0;
        adc_value = Read_ADC();
        sprintf((char *)buf, "%3d", adc_value);         // display 3 decima place
```

```
        lcd_putstring16(0,"ADC VAL = 000   "); //1st line display
        lcd_putstring16(1,"Voltage   00 V"); //2nd line display
        lcd_gotoxy(0,10);
        lcd_putstring(buf);
        voltage = (adc_value * 3.3) / 4095 ;
        lcd_gotoxy(1,8);
        sprintf(buf, "%3.2f", voltage);
        lcd_putstring(buf);
}
//////////   MAIN /////////////////
int main (void)
{
init_lcd();
Init_ADC();
  lcd_putstring16(0,"** MICROLAB **");
  lcd_putstring16(1,"** INSTRUMENTS    **");
delay(60000);
delay(60000);
delay(60000);
  lcd_putstring16(0,"ADC Value.. ");
  lcd_putstring16(1,"voltage.....");
while(1)
  {
        Display_ADC();
        delay(100000);
  }
}
```

DEPARTMENT VISION & MISSION

VISION

To become a pioneer in developing competent professionals with societal and ethical values through transformational learning and interdisciplinary research in the field of Electronics and Communication Engineering.

MISSION

The department of Electronics and Communication is committed to:

M1: Offer quality technical education through experiential learning to produce competent engineering professionals.

M2: Encourage a culture of innovation and multidisciplinary research in collaboration with industries/universities.

M3: Develop interpersonal, intrapersonal, entrepreneurial and communication skills among students to enhance their employability.

M4: Create a congenial environment for the faculty and students to achieve their desired goals and to serve society by upholding ethical values.