EMBEDDED SYSTEMS LAB

B.E., VI Semester, Electronics & Communication Engineering

Subject Code	18ECL66	CIE Marks	40
Number of Lecture Hours/Week	01Hr Tutorial (Instructions) + 02 Hours Laboratory = 03	See Marks	60
RBT Levels	L1, L2, L3	Exam Hours	03

CREDITS – 02

Course objectives: 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.

Laboratory Experiments

PART-A: Conduct the following Study experiments to learn ALP using ARM Cortex M3 Registers using an Evaluation board and the required software tool.

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

PART-B: Conduct the following experiments on an ARM CORTEX M3 evaluation board using evaluation version of Embedded 'C' &Keil uVision-4 tool/compiler.

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. Demonstrate the use of an external interrupt to toggle an LED On/Off.
7. Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between.
8. Measure Ambient temperature using a sensor and SPI ADC IC.

COs	Embedded Controller Lab
CO1	Apply the knowledge of ARM Cortex M3instruction set a 32-bit microcontroller, and use the software design tool to analyze programs in Assembly and C language.
CO2	Develop assembly language programs using ARM Cortex M3 for differentapplications and analyze the functionality with appropriate test inputs.
CO3	Use of external devices to interfacewith the I/O ports and internal peripherals of ARM Cortex M3.
CO4	Develop embedded C program for embedded system applications and verify the functionality using software design tool.

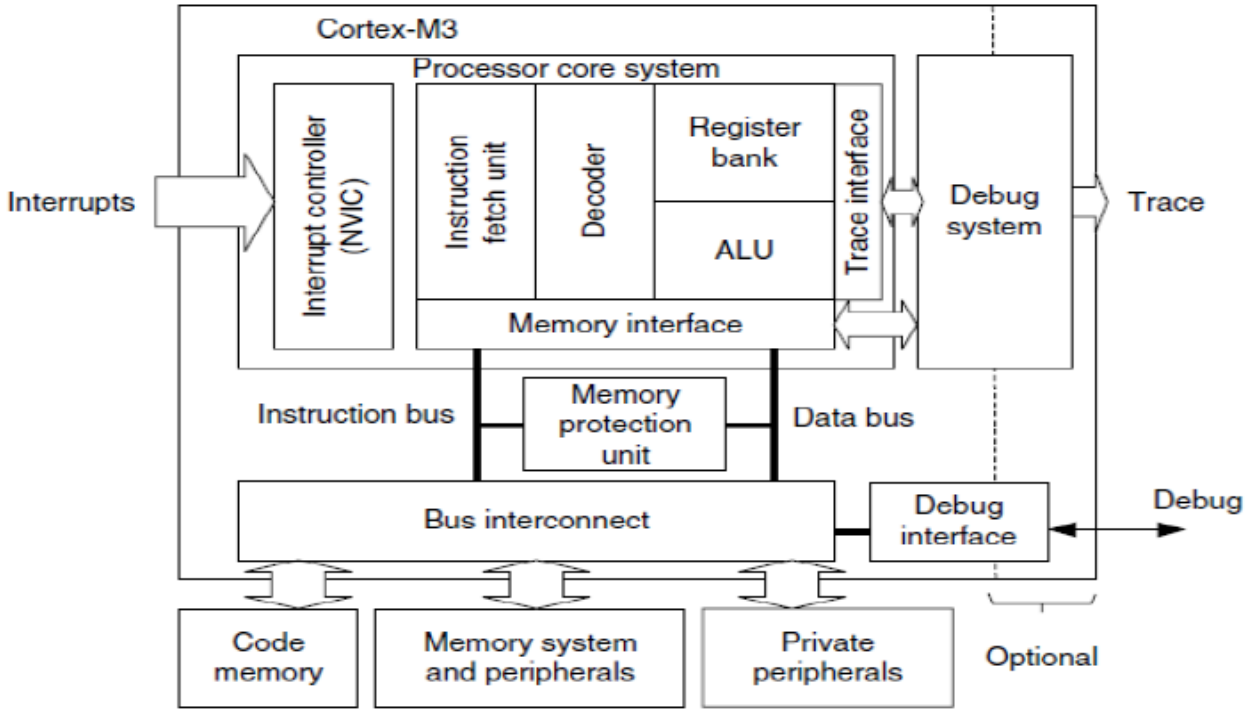
CHAPTER 1

INTRODUCTION TO ARM CORTEX M3 PROCESSOR

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.

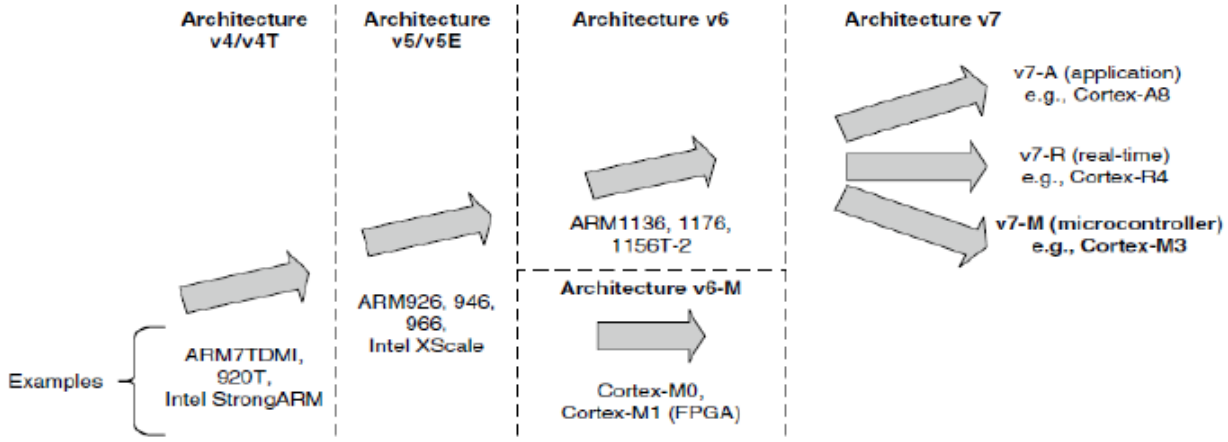
Bit more details on these profiles

***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



ADVANTAGES OF CORTEX M3 PROCESSORS

The Cortex-M3 addresses the requirements for the 32-bit embedded processor market in the following ways:
***Greater performance efficiency*:** allowing more work to be done without increasing the frequency or power requirements
***Low power consumption*:** enabling longer battery life, especially critical in portable products including wireless networking applications
***Enhanced determinism*:** guaranteeing that critical tasks and interrupts are serviced as quickly as possible and in a known number of cycles
***Improved code density*:** ensuring that code fits in even the smallest memory footprints
***Ease of use*:** providing easier programming and debugging for the growing number of 8-bit and 16-bit users migrating to 32 bits
***Lower cost solutions*:** reducing 32-bit-based system costs close to those of legacy 8-bit and 16-bit devices and enabling low-end, 32-bit micro controllers to be priced at less than US$1 for the first time
***Wide choice of development tools*:** from low-cost or free compilers to full-featured development suites from many development tool vendors
Cost savings can be achieved by improving the amount of code reuse across all systems. Because Cortex-M3 processor-based micro controllers can be easily programmed using the C language and are based on a well-established architecture, application code can be ported and reused easily, and reducing development time and testing costs.

APPLICATIONS OF CORTEX M3 PROCESSORS

***Low-cost micro controllers*:** The Cortex-M3 processor is ideally suited for low-cost micro controllers, which are commonly used in consumer products, from toys to electrical appliances. It is a highly competitive market due to the many well-known 8-bit and 16-bit micro controller products on the market. Its lower power, high performance, and ease-of-use advantages enable embedded developers to migrate to 32-bit systems and develop products with the ARM architecture.
***Automotive*:** Another ideal application for the Cortex-M3 processor is in the automotive industry. The Cortex-M3 processor has very high-performance efficiency and low interrupt latency, allowing it to be used in real-time systems. The Cortex-M3 processor supports up to 240 external vectored interrupts, with a built-in interrupt controller with nested interrupt supports and an optional MPU, making it ideal for highly integrated and cost-sensitive automotive applications.
***Data communications*:** The processor's low power and high efficiency, coupled with instructions in Thumb-2 for bit-field manipulation, make the Cortex-M3 ideal for many communications applications, such as Bluetooth and ZigBee.
***Industrial control*:** In industrial control applications, simplicity, fast response, and reliability are key factors. Again, the Cortex-M3 processors interrupt feature, low interrupt latency, and enhanced fault-handling features make it a strong candidate in this area.
***Consumer products*:** In many consumer products, a high-performance microprocessor (or several of them) is used. The Cortex-M3 processor, being a small processor, is highly efficient and low in power and supports an MPU enabling complex software to execute while providing robust memory protection.

MICROCONTROLLER 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 layerAHB(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.

TECHNICAL 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.

Board Specifications of the ARM cortex M3 Microcontroller LPC1768

The LPC1768 is ARM cortex M3 based 32 bit Microcontroller. It has the following features

- 512KB flash memory and 64KB SRAM; In-System Programming (ISP) and In-Application Programming (IAP) capabilities
- It has Single 3.3 V power supply (2.4 V to 3.6 V).
- There are 70 General Purpose I/O (GPIO) pins with configurable pull-up/down resistors, open drain mode, and repeater mode.
- Possesses a 12 bit Analog to Digital converter (ADC) and up to 8 analog channels
- It has 10-bit Digital-to-Analog Converter (DAC) with dedicated conversion timer.
- Easy communication set up with the availability of 12 MHz crystal.
- Possesses a Reset push-button for resetting the controller
- Four general purpose timers/counters are available with a total of 8 capture inputs and 10 compareoutputs.
- Possesses 3 I2C bus interfaces
- There is a serial peripheral interface (SPI) controller with synchronous, serial and full duplex communication.
- Sleep mode, deep sleep mode, power down mode and deep power down mode are the four reduced power modes available.
- Real time clock (RTC) with a separate power domain is also available.
- Presence of an onboard voltage regulator for generating 3.3 volts; Input to this is through the external +5volt DC power supply through a 9 pin DSUB connector.
- Piggy back module is available which contains the LPC1768 controller.
- Presence of one RS232 interface circuit with 9 pin DSUB connector.
- Standard JTAG connector with ARM 2×10 pin layout for programming/debugging with ARM-JTAG
- Possesses DC motor interface for direction and speed control
- Availability of stepper motor interface with direction and speed control
- It has an 8 bit DAC interface.
- There is a 16x2 alphanumeric display for LCD interfacing.
- Presence of a 4x4 matrix keyboard for keyboard interface which is connected to the port lines of the controller.
- Possesses a two digit multiplexed seven segment display interface.
- On board buzzer, relay and LED indication controlled through push button interface circuits are also available.
- It has a 2 channel ADC IC with POT and temperature sensor.
- Standard 26 pin FRC connectors are available to connect to on board interface.

System Specifications

The system specifications are illustrated in table.

Sl. No	System Specifications
1.	Domain: Embedded systems Microprocessors and Microcontrollers
2.	ARM cortex M3 Microcontroller: LPC1768 32-bit RISC microcontroller from NXP founded by Philips.
3.	DC Motor : 5 volts, 100 mA
4.	Desktop computer: Pentium 4, 1 GB RAM, Processor speed 2.5 GHz.
5.	Programming language: Embedded C
6.	Port line: P.124 & P0. 26
7.	Software: Keilμvision-4
8.	Adapter cable, USB cable
9.	In-system Programming (ISP): Flash magic software can be used to download the HEX files to the flash magic of the controller.
10.	Serial communication: RS 232 cross cable connections required for establishing communication between the evaluation board and a display terminal/host computer.
11.	Applications: Rotation of DC motor in clockwise and anticlockwise direction

INTRODUCTION TO KEIL:

One of the important part in making an embedded system is loading the software/program developed into the microcontroller. Usually it is called "burning software" into the controller.

Prerequisite operations with the program that must be done before "burning a program" into a controller, This includes writing the program in assembly language or C language in a text editor like notepad, compiling the program in a compiler and finally generating the hex code from the compiled program.

Earlier people used different softwares/applications for all these 3 tasks

1. Writing was done in a text editor like Notepad/WordPad
2. Compiling was done using a separate software (probably a dedicated compiler for a particular controller like 8051),
3. Converting the assembly code to hex code was done using another software etc.

It takes lot of time and work to do all these separately, especially when the task involves lots of error debugging and reworking on the source code.

KeilMicroVision is a free software which solves many of the pain points for an embedded program developer. This software is an integrated development environment (IDE), which integrated a text editor to write programs, a compiler and it will convert your source code to hex files too.

INTRODUCTION TO FLASH MAGIC:

Flash Magic is Windows software from the Embedded Systems Academy that allows easy access to all the ISP features provided by the devices. These features include

1. Erasing the Flash memory (individual blocks or the whole device)
2. Programming the Flash memory
3. Modifying the Boot Vector and Status Byte
4. Reading Flash memory
5. Performing a blank check on a section of Flash memory
6. Reading the signature bytes
7. Reading and writing the security bits
8. Direct load of a new baud rate (high speed communications)
9. Sending commands to place device in Boot loader mode

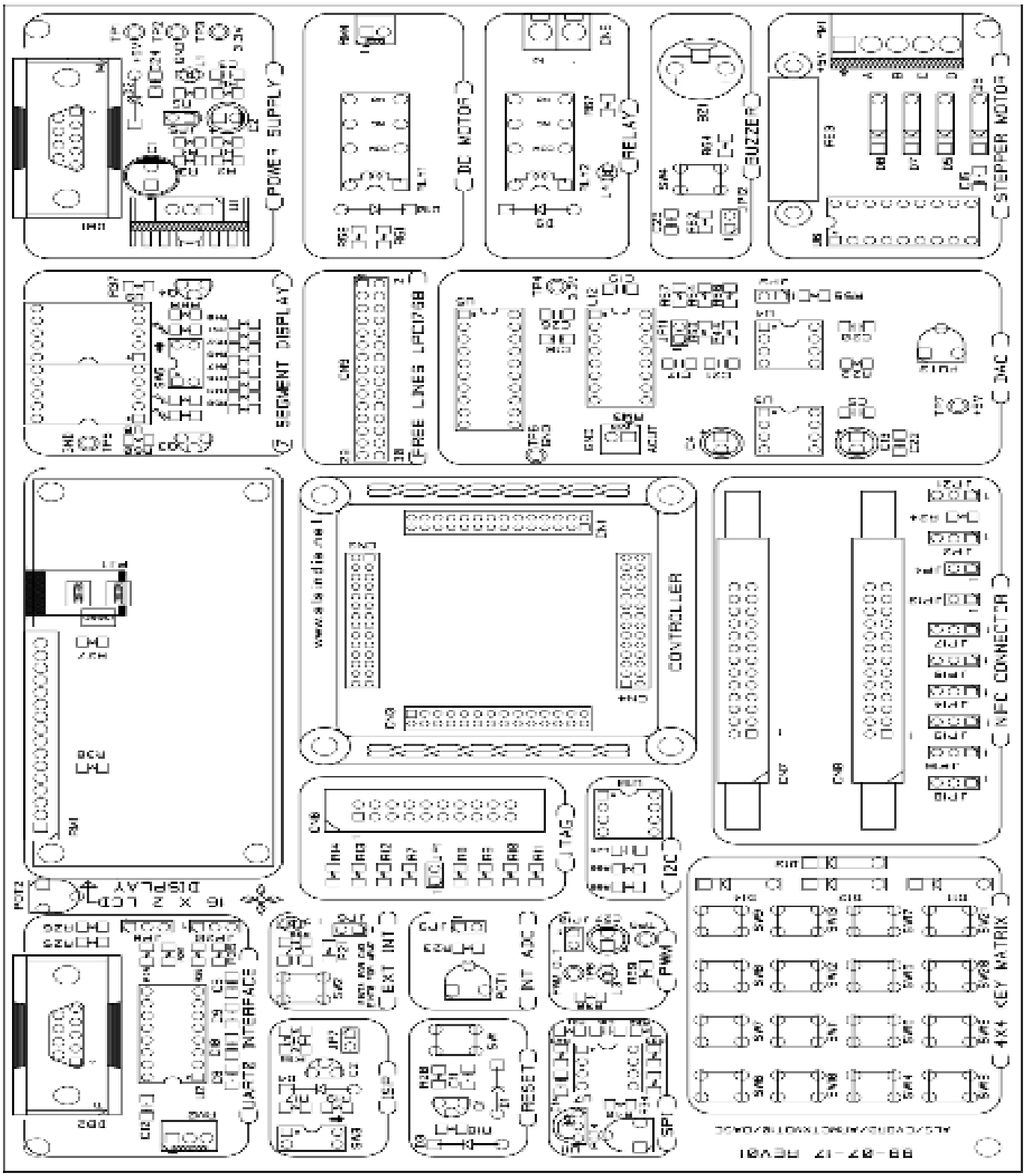
Flash Magic provides a clear and simple user interface to these features. Under Windows, only one application may have access the COM Port at any one time, preventing other applications from using the COM Port.

Flash Magic only obtains access to the selected COM Port when ISP operations are being performed. This means that other applications that need to use the COM Port, such as debugging tools, may be used while Flash Magic is loaded.

SPECIFICATIONS OF ALS-SDA-ARMCTXM3 KIT

- **LPC1768** is **ARM Cortex M3** based Microcontroller with
  - 512KB flash memory and 64KB SRAM In-System Programming (ISP) and In-Application Programming (IAP) capabilities.
  - Single 3.3 V power supply (2.4 V to 3.6 V).
  - 70 General Purpose I/O (GPIO) pins with configurable pull-up/down resistors, open drain mode, and repeater mode.
  - 12-bit Analog-to-Digital Converter (ADC) and up to 8 analog channels.
  - 10-bit Digital-to-Analog Converter (DAC) with dedicated conversion timer.
  - Four general purpose timers/counters, with a total of eight capture inputs and ten compare outputs.
  - Four UART's with fractional baud rate generation, internal FIFO, IrDA.
  - SPI controller with synchronous, serial, full duplex communication.
  - Three enhanced I2C-bus interfaces
  - Four reduced power modes: Sleep, Deep-sleep, Power-down, and Deep power-down.
  - Real-Time Clock (RTC) with a separate power domain.
  - Standard JTAG test/debug interface as well as Serial Wire Debug.
  - Four external interrupt inputs configurable as edge/level sensitive.
- 12MHz Crystal allows easy communication setup
- One on board voltage regulator for generating 3.3V. Input to this will be from External +5V DC Power supply through a 9-pin DSUB connector
- Piggy Back module containing **LPC1768** controller
- Standard JTAG connector with ARM 2×10 pin layout for programming/debugging with ARM-JTAG
- Reset push-button for resetting the controller
- One RS232 interface circuit with 9 pin DSUB connector: this is used by the Boot loader program, to program **LPC1768** Flash memory without external Programmer
- DC motor interface with direction and speed control
- Stepper motor interface with direction and speed control
- 16×2 alphanumeric LCD Display
- On chip ADC interface circuit using AD0.5(P1.31)
- 8-bit DAC interface
- 4x4 Key-Matrix connected to the port lines of the controller
- One External interrupt circuit with LED indication
- Two-digit multiplexed 7-segment display interface
- Interface circuit for on board Buzzer, Relay and Led indication controlled through push button.
- SPI Interface: 2 channel ADC IC with POT and Temperature sensor
- I2C Interface: NVROM IC
- Standard 26-pin FRC connectors to connect to **on-board interface** or some of **ALS standard**

PLACEMENT DIAGRAM



SOFTWARE/FIRMWARE DETAILS

System set up

The system set up was done in Embedded Systems laboratory. The system consists of a Desktop computer, ARM cortex M3 LPC1768 evaluation board and Power supply module. The complete interfacing is illustrated by the photographic view in figure.

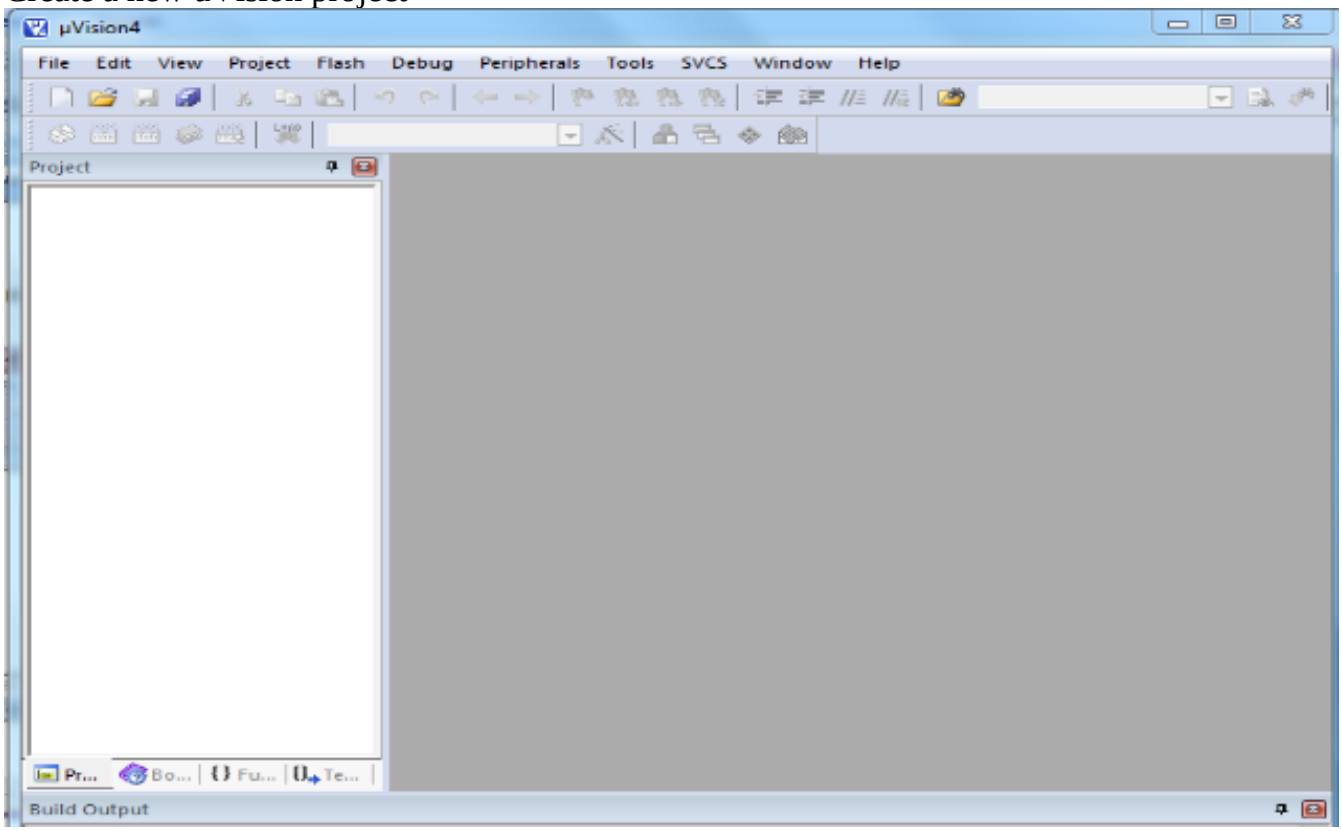
The required hardware connections were done by connecting the adapter cable to the socket and the adapter pin was connected to the ARM cortex M3 LPC1768 evaluation board. Apart from this, the USB cable was connected to the USB port of the desktop computer and the other end of the USB cable was connected to the female connector of the RS232 cable and the male connector of the RS232 cable was connected to the female connector provided on the ARM cortex M3 LPC1768 evaluation board.

Figure - Photographic view of the system

Project Creation in Keil uvision4 IDE

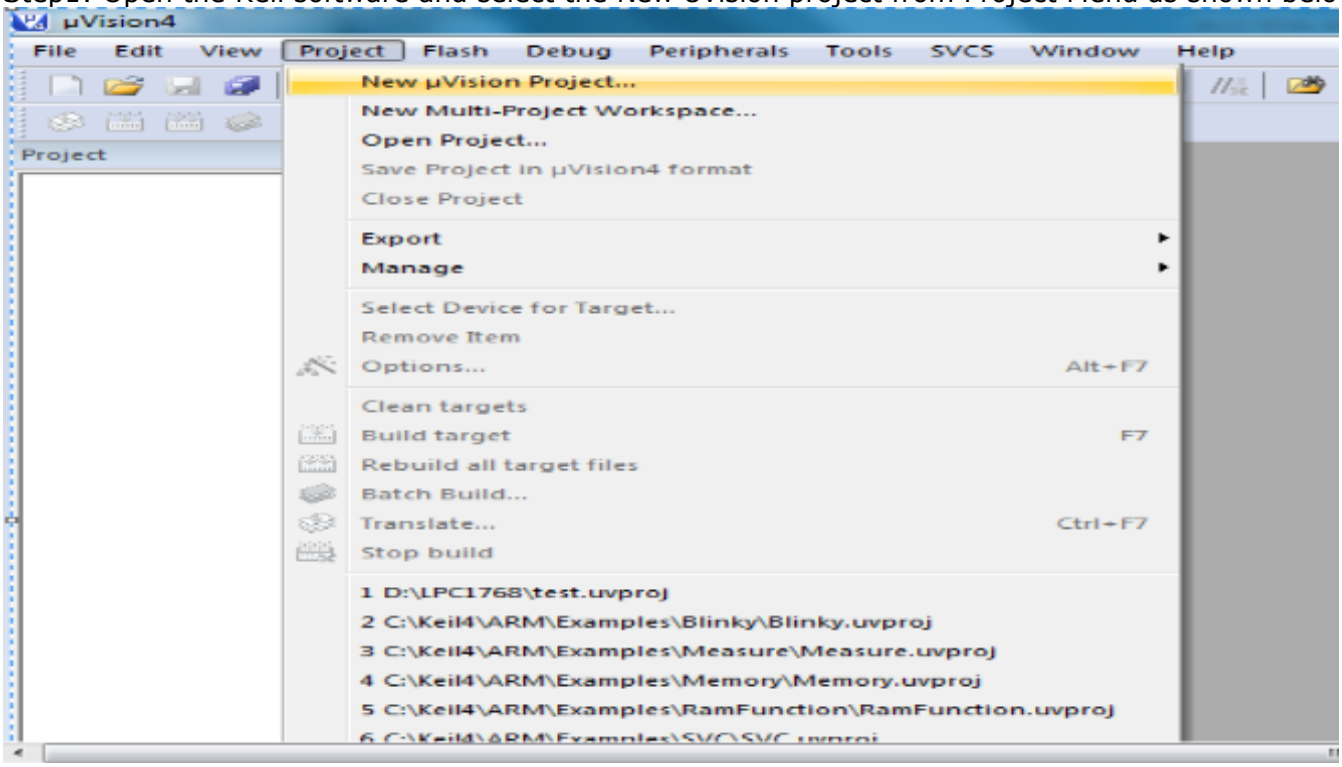
✓ Create a project folder before creating NEW project
✓ Use separate folder for each project
✓ Open Keil uVision4 IDE software by double clicking on “Keil Uvision4” icon

Create a new uVision project

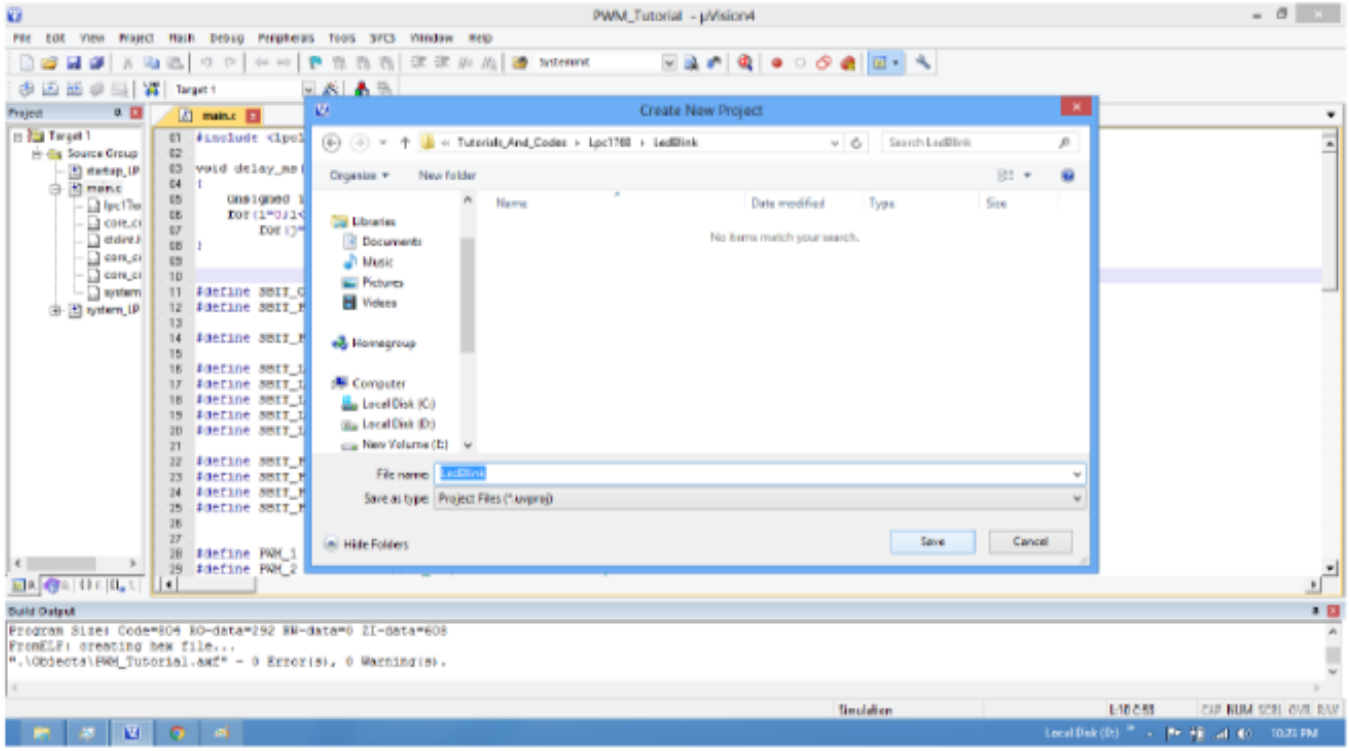


2. EXECUTION STEPS

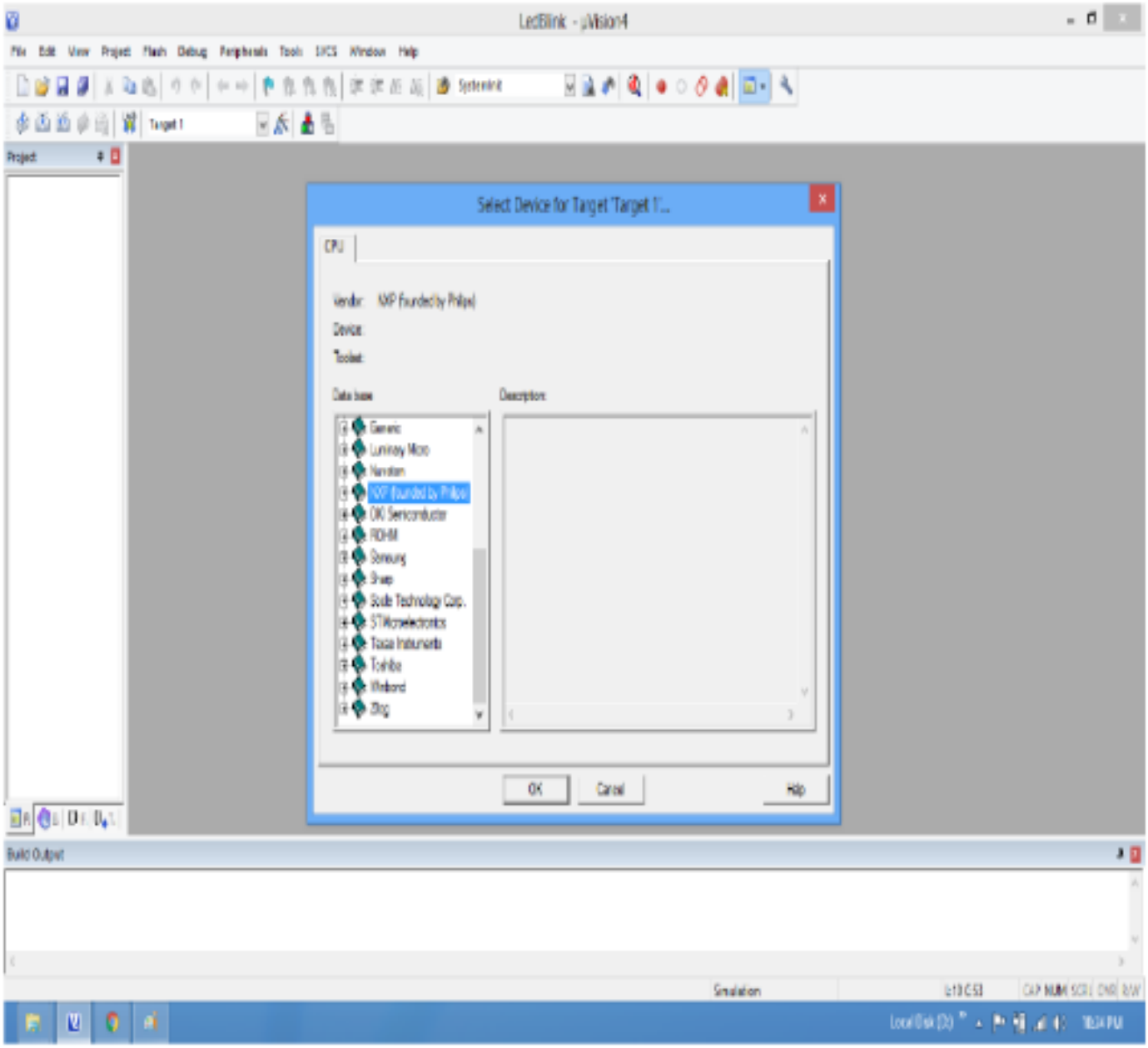
Step1: Open the Keil software and select the New Uvision project from Project Menu as shown below.



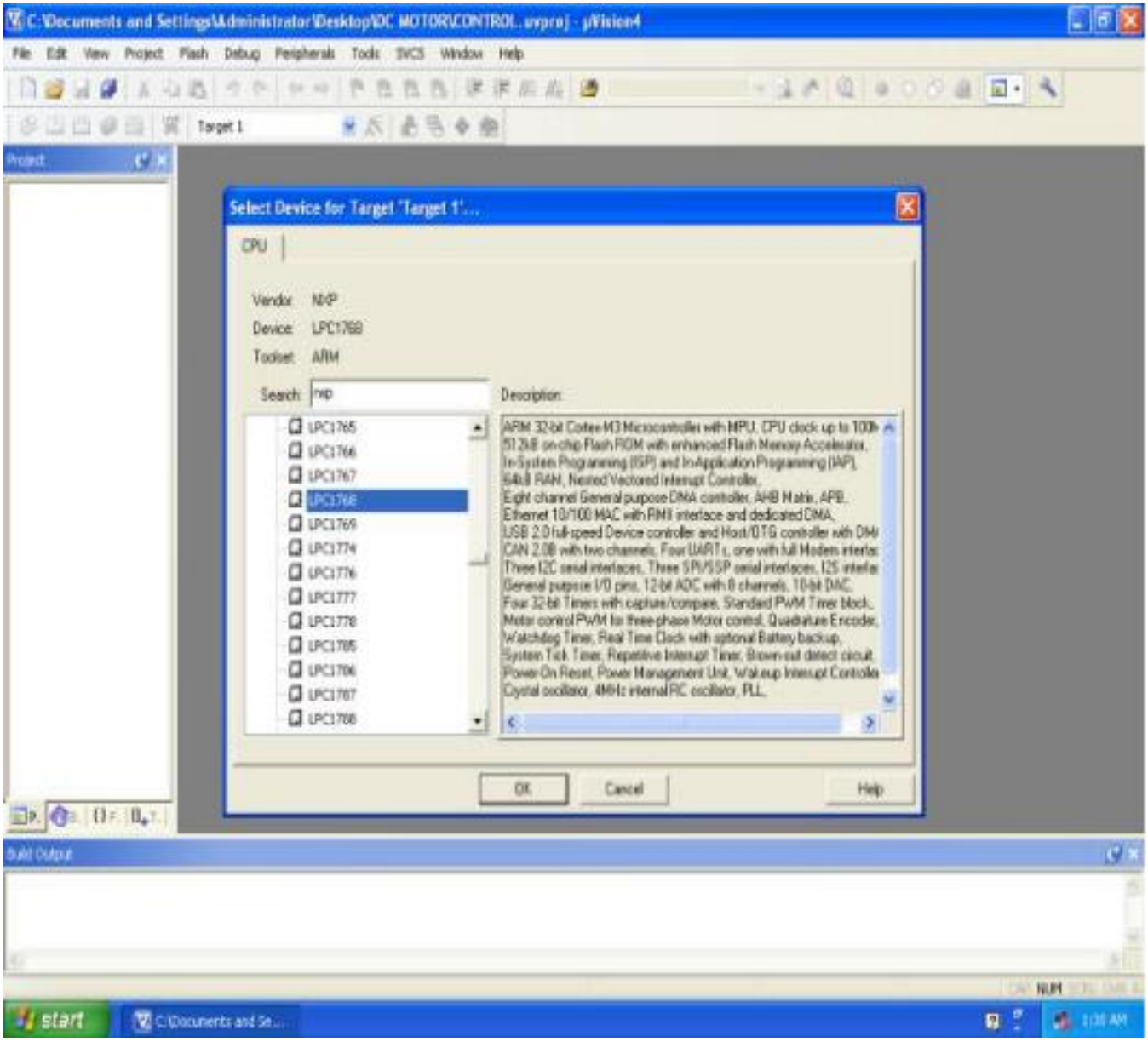
Step2: Browse to your project folder and provide the project name and click on save



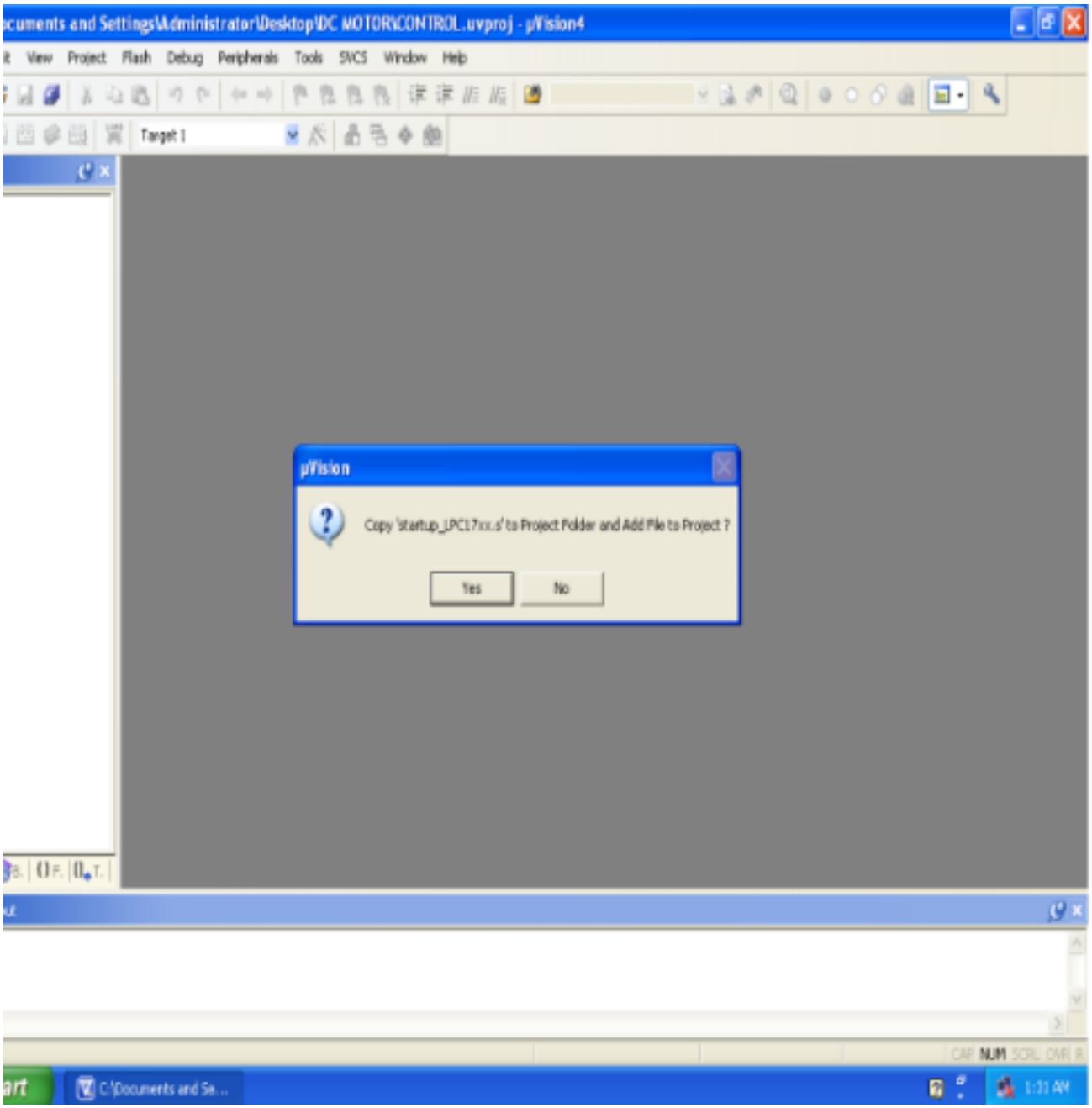
Step3: Once the project is saved a new pop up "Select Device for Target" opens, Select the controller(NXP:LPC1768) and click on OK.



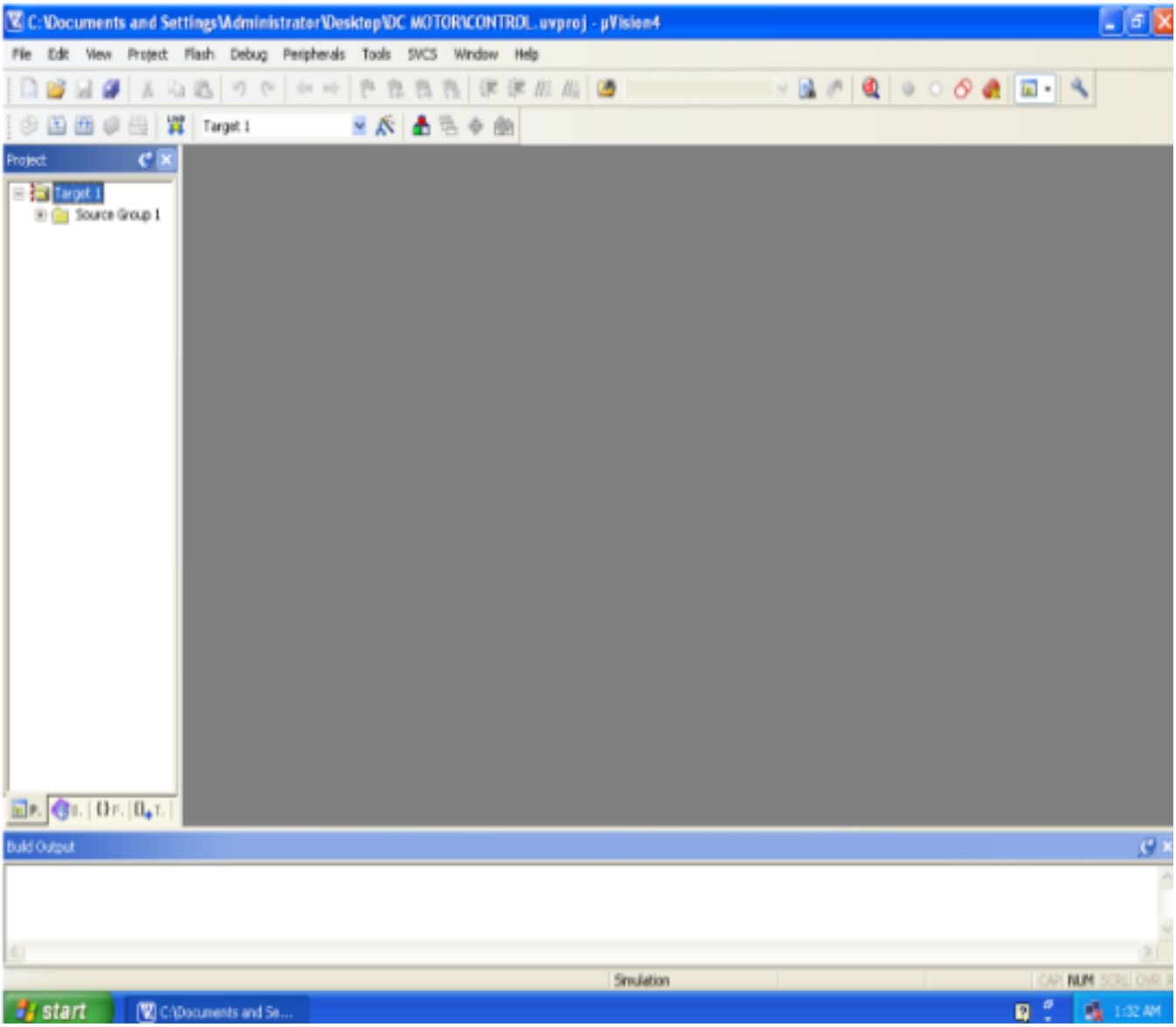
Step4: Select the controller(NXP:LPC1768) and click on OK



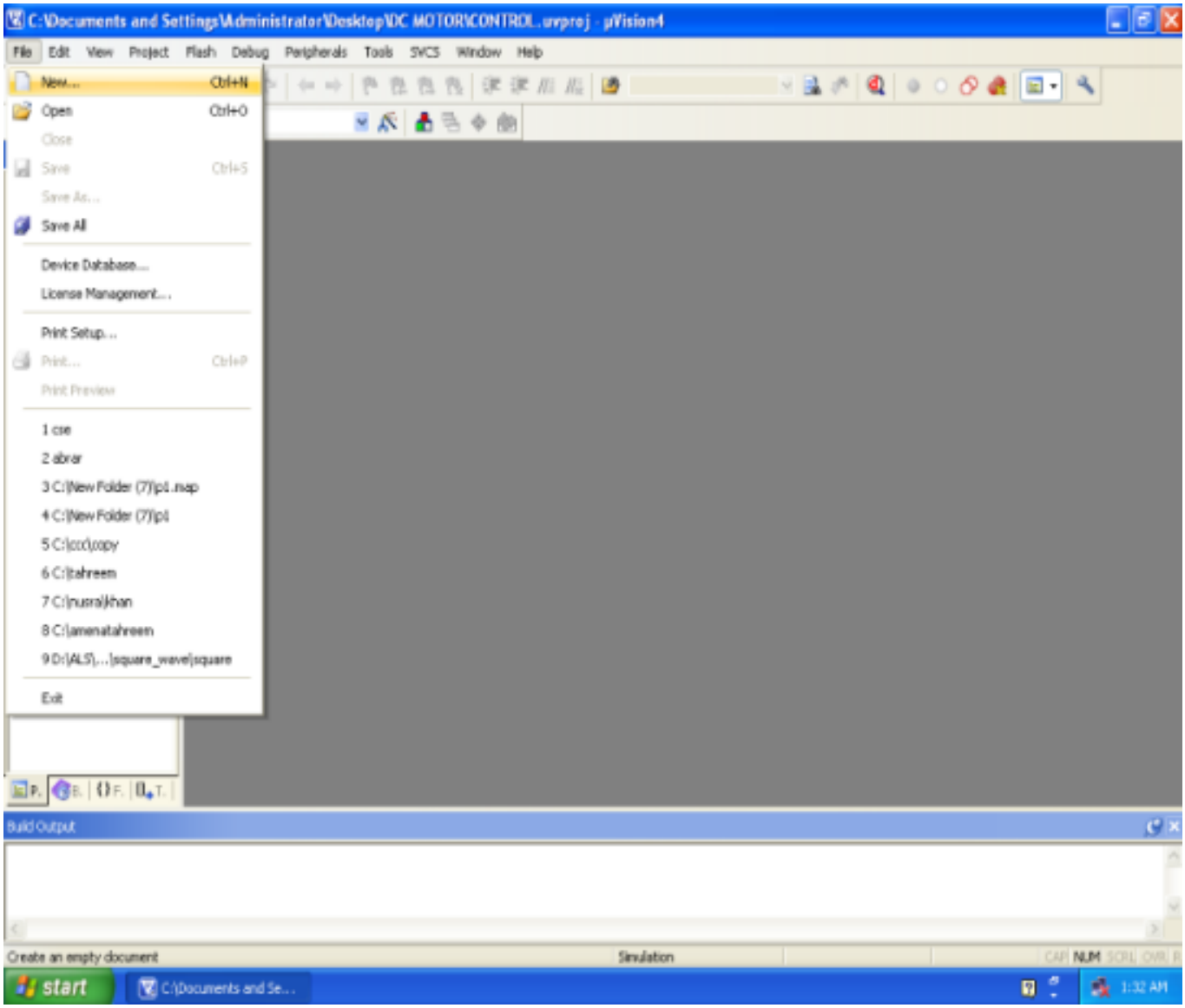
Step5: As LPC1768 needs the startup code, click on Yes option to include the LPC17xx Startup file.



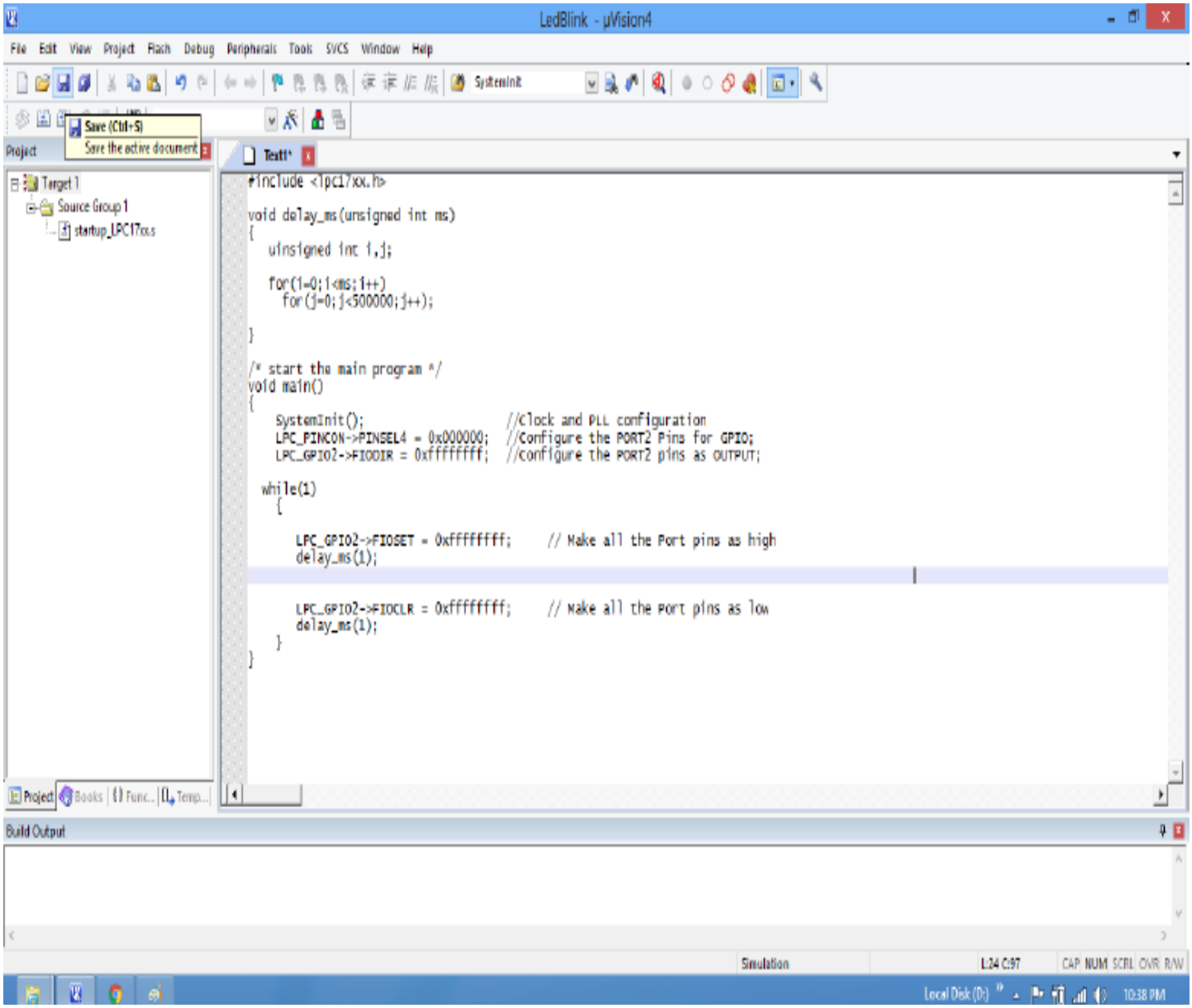
Step6: Create a new file to write the program.



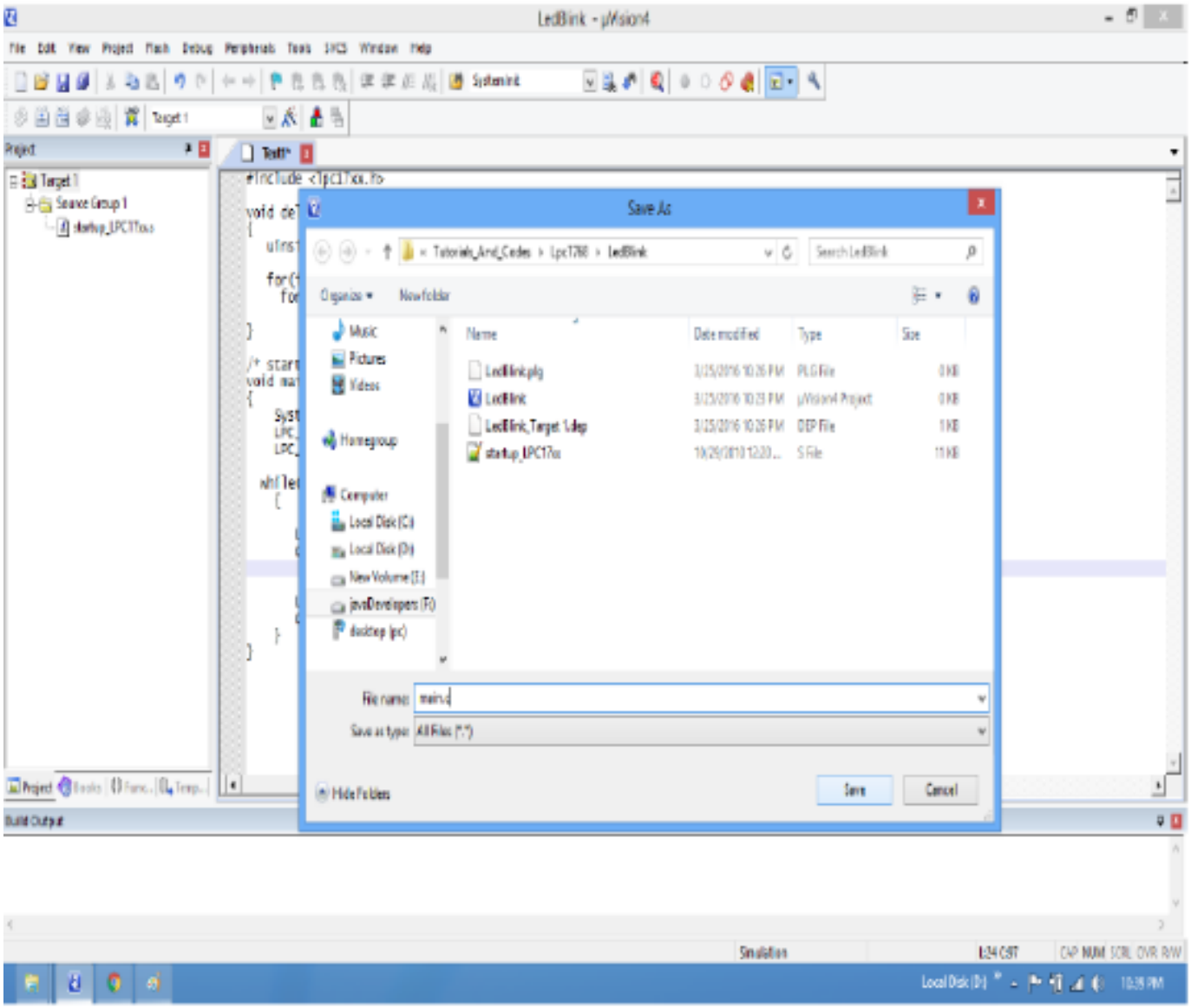
Step6: Create a new file to write the program.



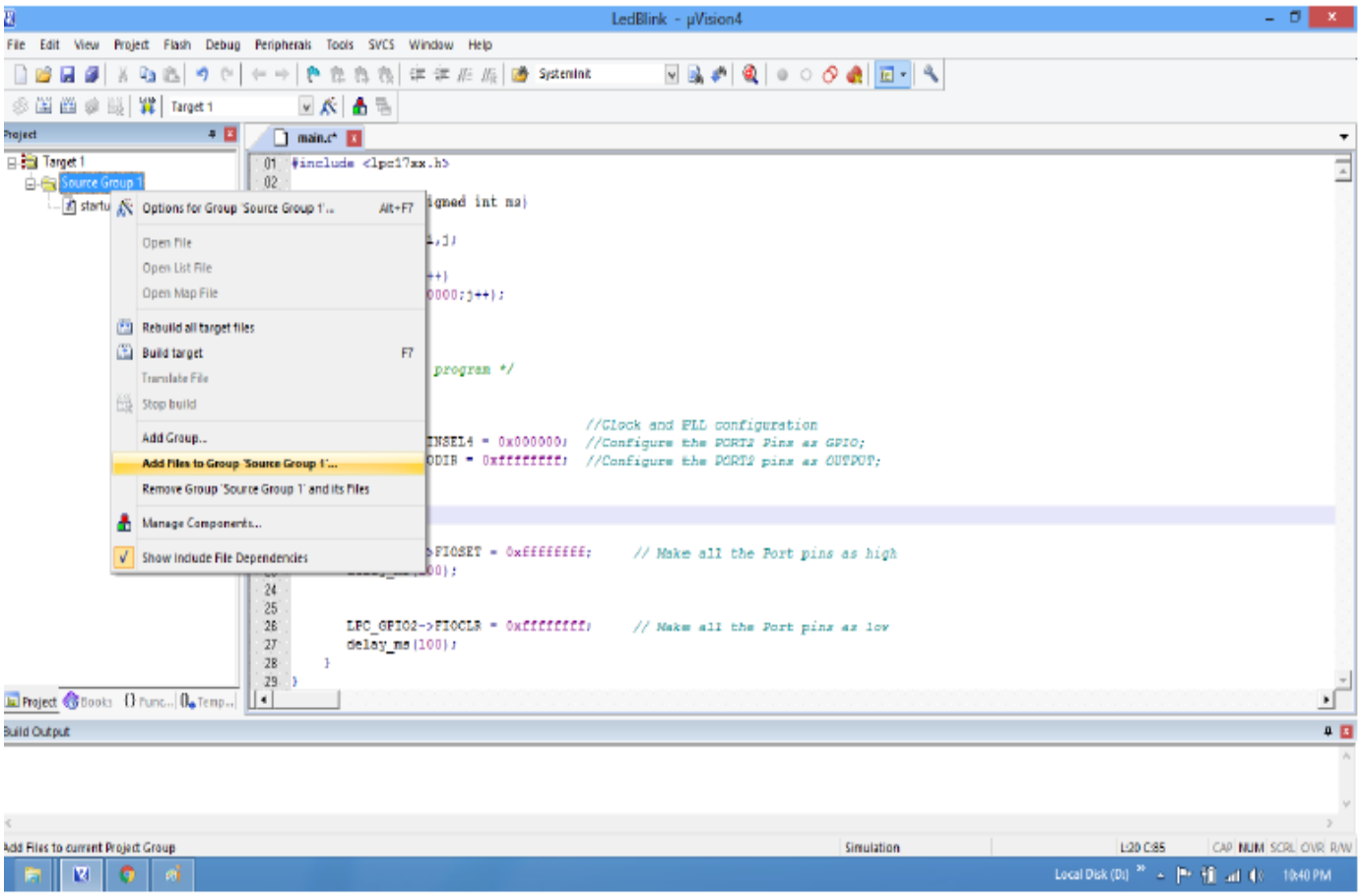
Step7: Type the code or Copy paste the below code snippet.



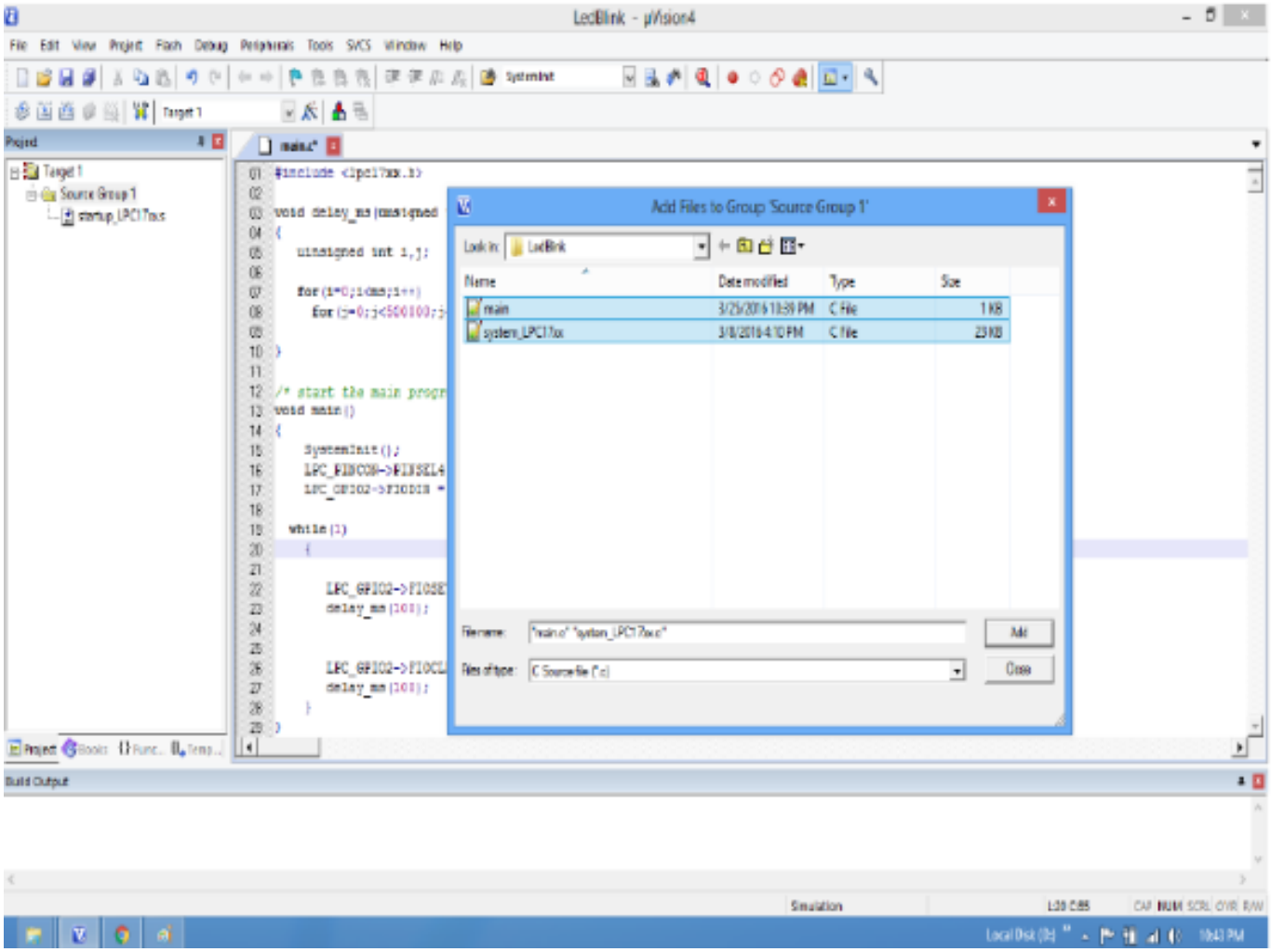
Step8: After typing the code save the file as main.c.



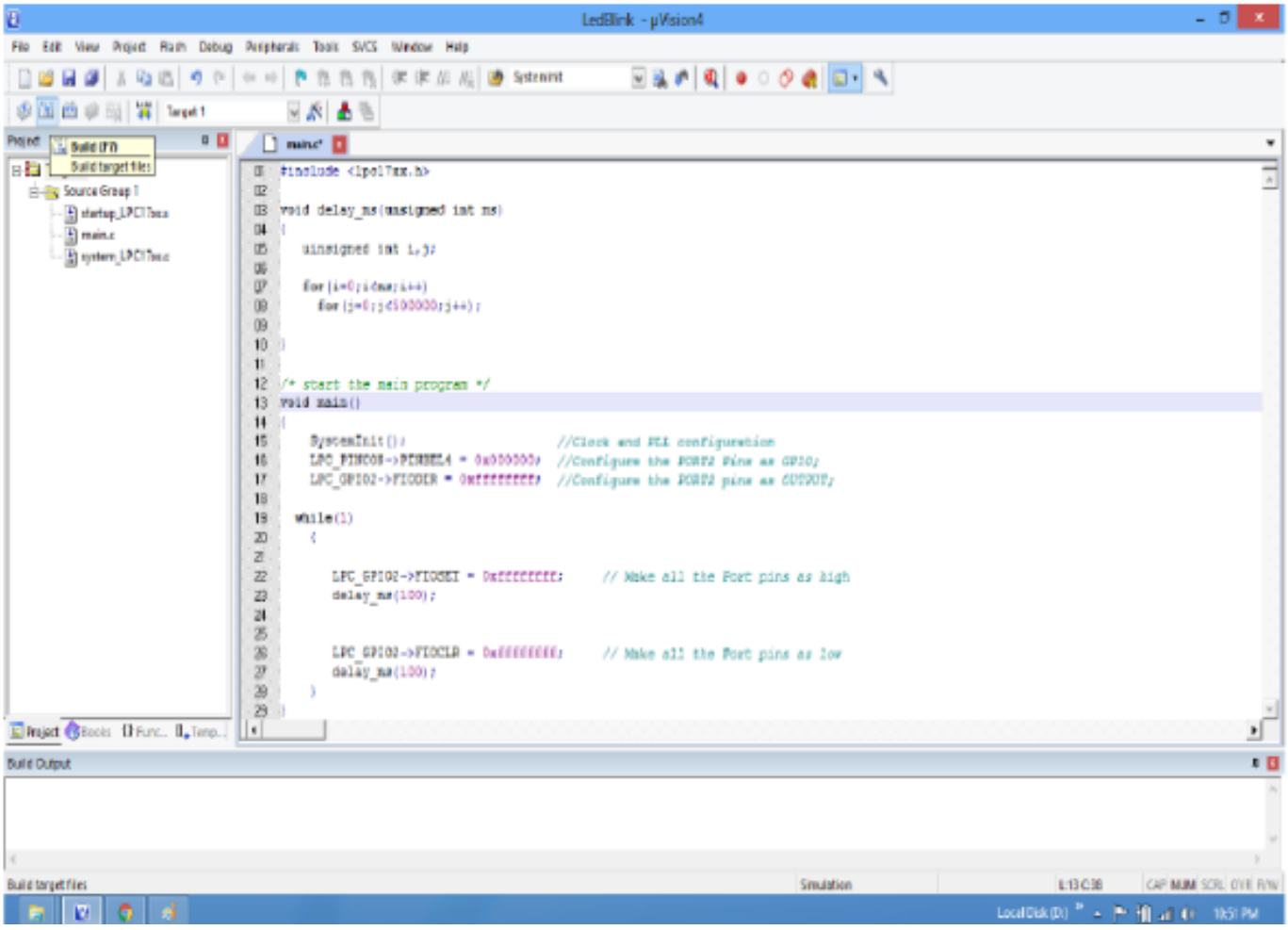
Step9: Add the recently saved file to the project.


LedBlink - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
SystemInit
Target 1
Project
Target 1
Source Group 1
main.c*
#include <lpc17xx.h>
Options for Group 'Source Group 1'... Alt+F7
Open File
Open List File
Open Map File
Rebuild all target files
Build target F7
Translate File
Stop build
Add Group...
Add Files to Group 'Source Group 1'...
Remove Group 'Source Group 1' and its Files
Manage Components...
Show Include File Dependencies
//Clock and PLL configuration
//Configure the PORT2 Pins as GPIO;
//Configure the PORT2 pins as OUTPUT;
program */
FIOSET = 0xFFFFFFFF; // Make all the Port pins as high
LPC_GPIO2->FIOCLR = 0xFFFFFFFF; // Make all the Port pins as low
delay_ms(100);
Project Books Func... Temp...
Build Output
Add Files to current Project Group
Simulation
L20 C85
CAP NUM SCRL OVR R/W
Local Disk (D:)
10:40 PM

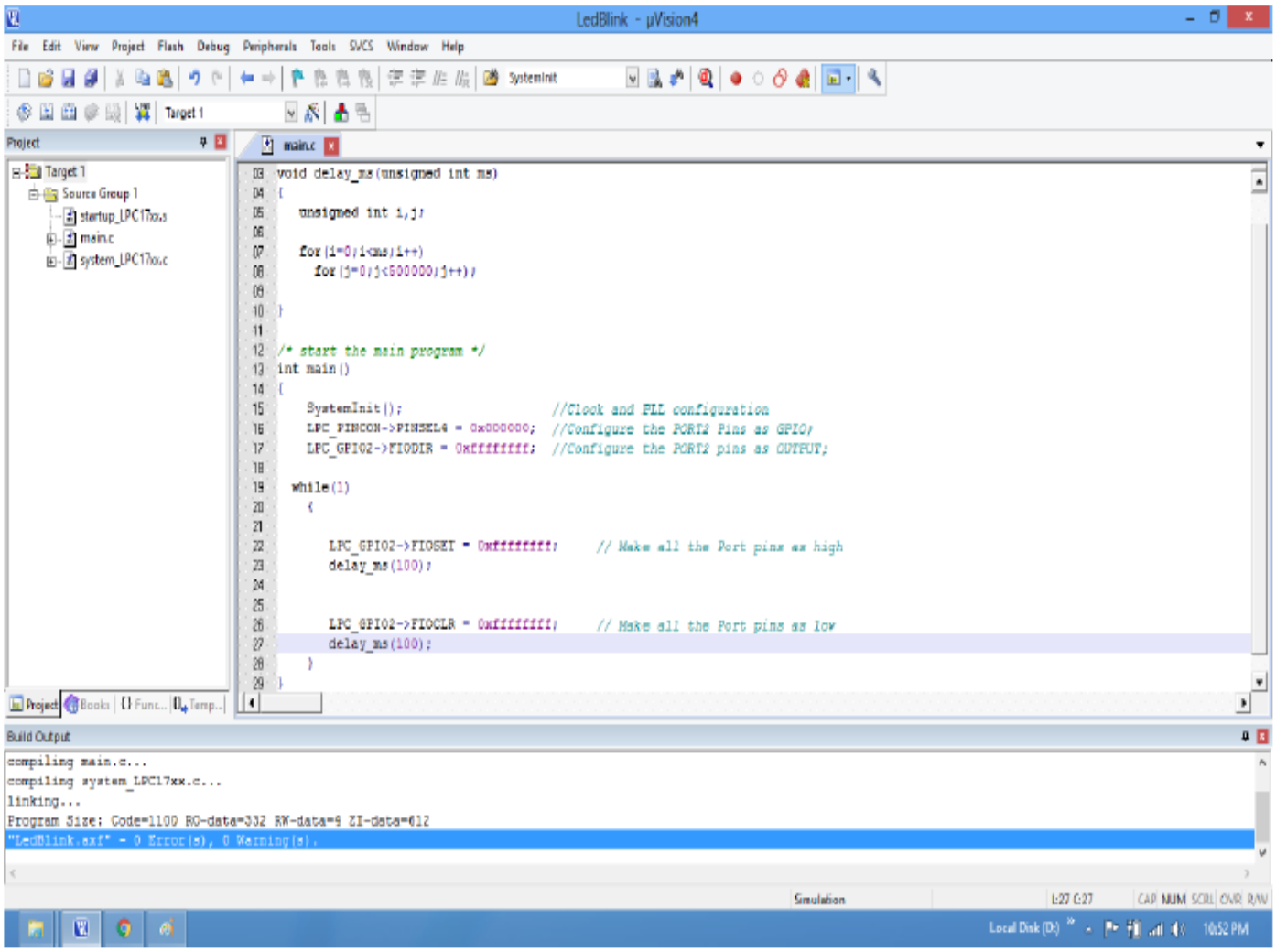
Step10: Add the main.c along with system_LPC17xx.c.



Step11: Build the project and fix the compiler errors/warnings if any.

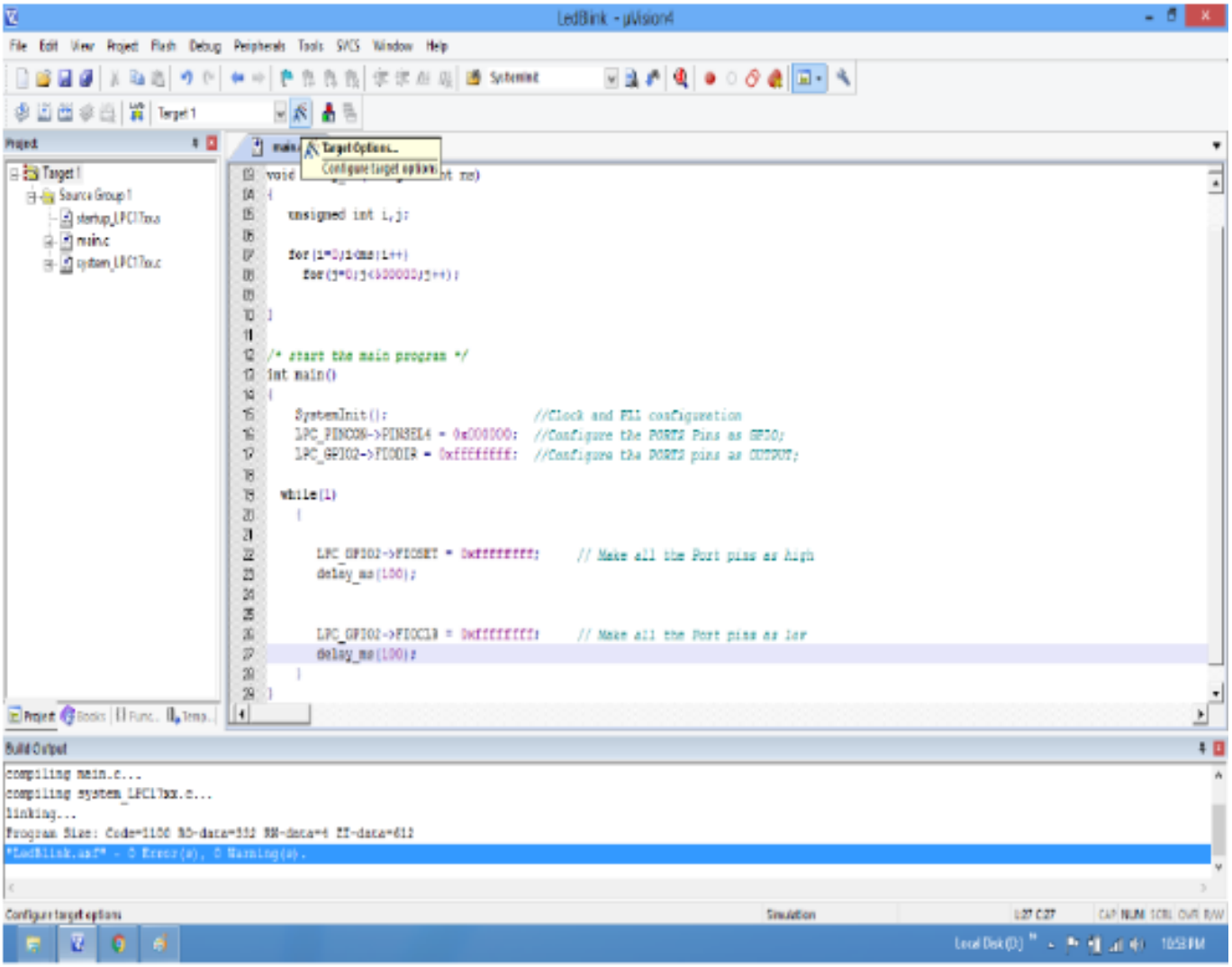


Step12: Code is compiled with no errors. The **.hex** file is still not generated.



Enable Hex File Generation

Step13: Click on Target Options to select the option for generating .hex file.

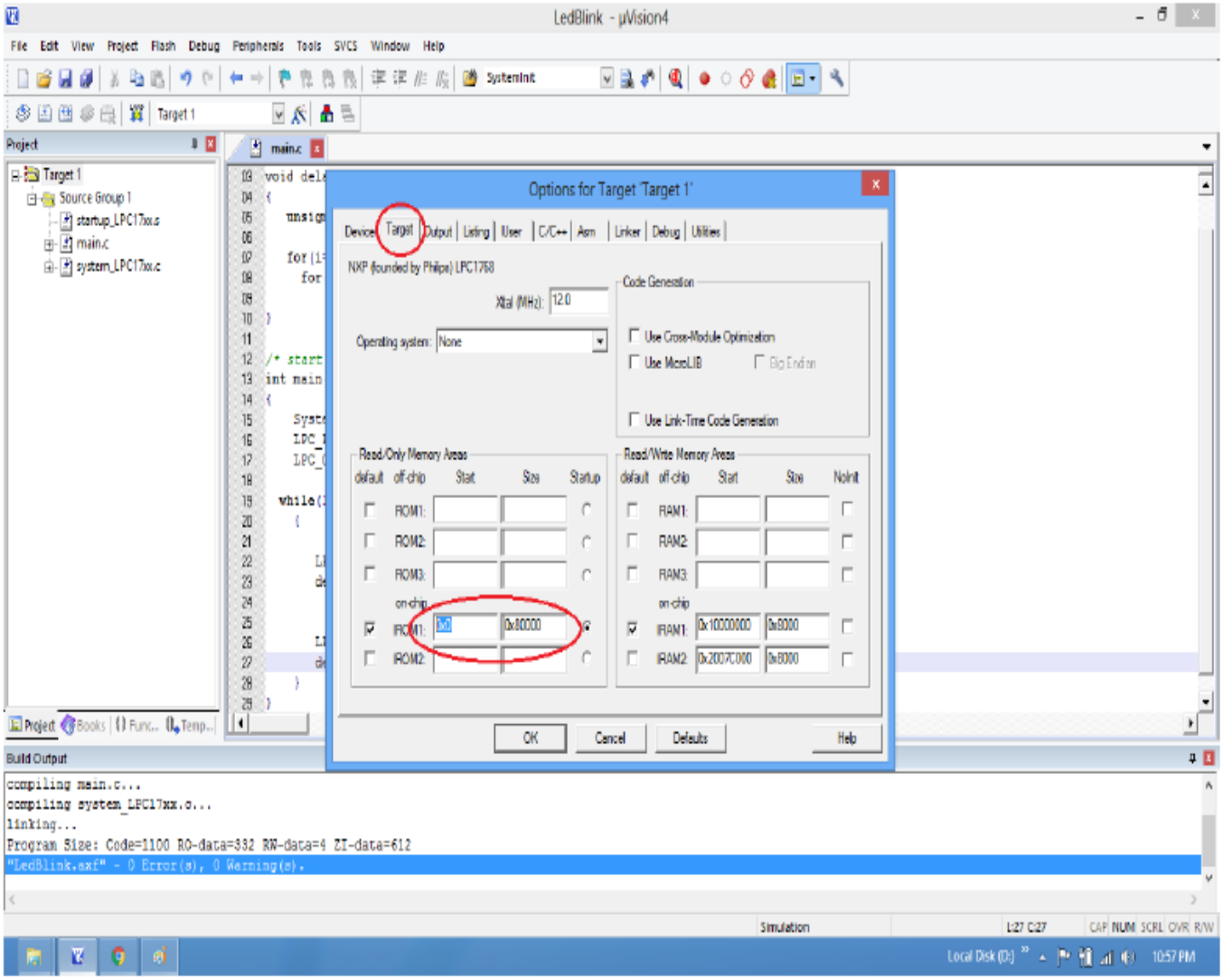


Step14: In Project Window Right click "TARGET1" and select "options for target „TARGET1". Then go to option "Target" in that

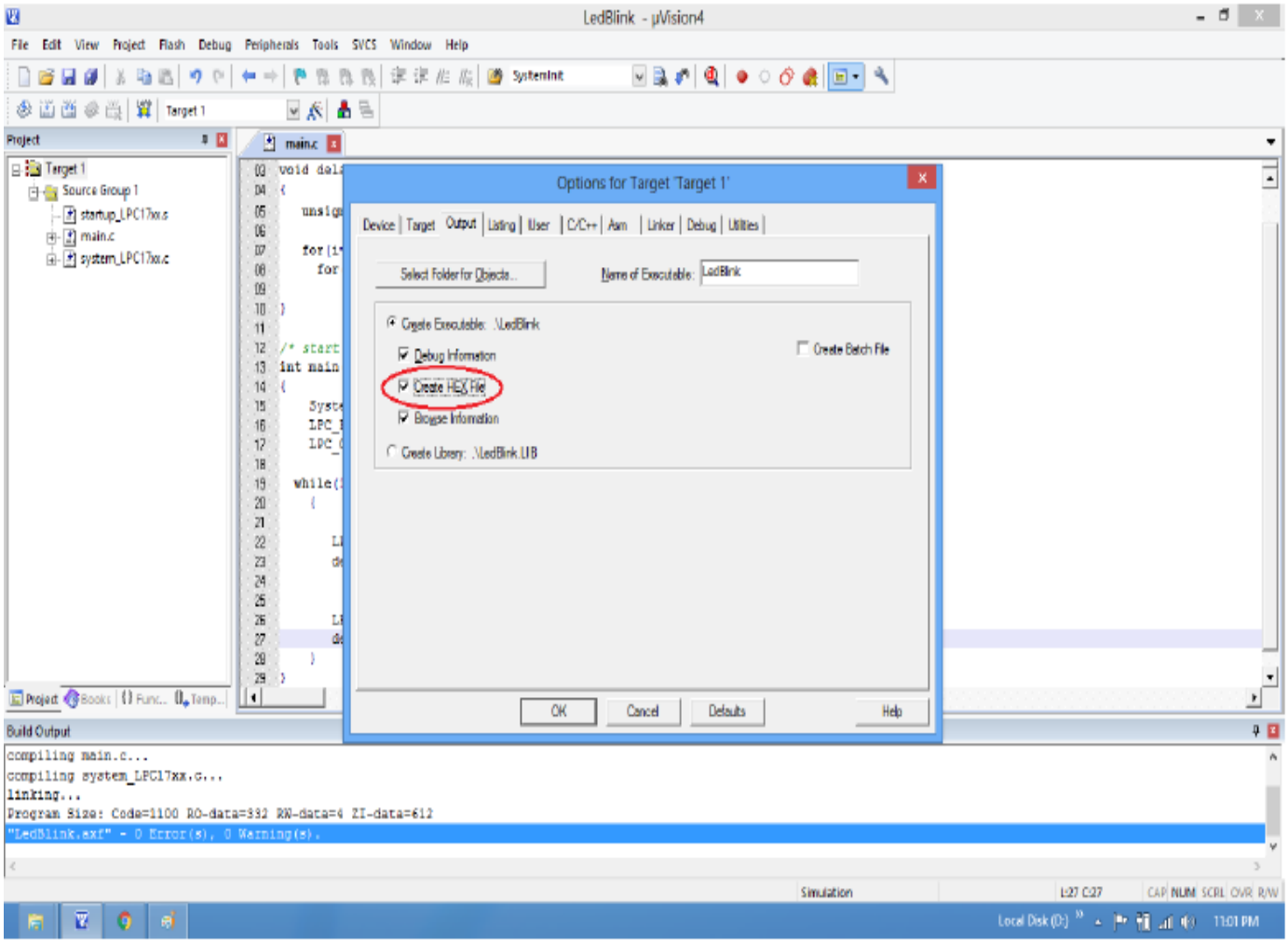
1. Xtal 12.0MHz and Use MicroLIB

2. Set IROM1 start address as 0x0000.

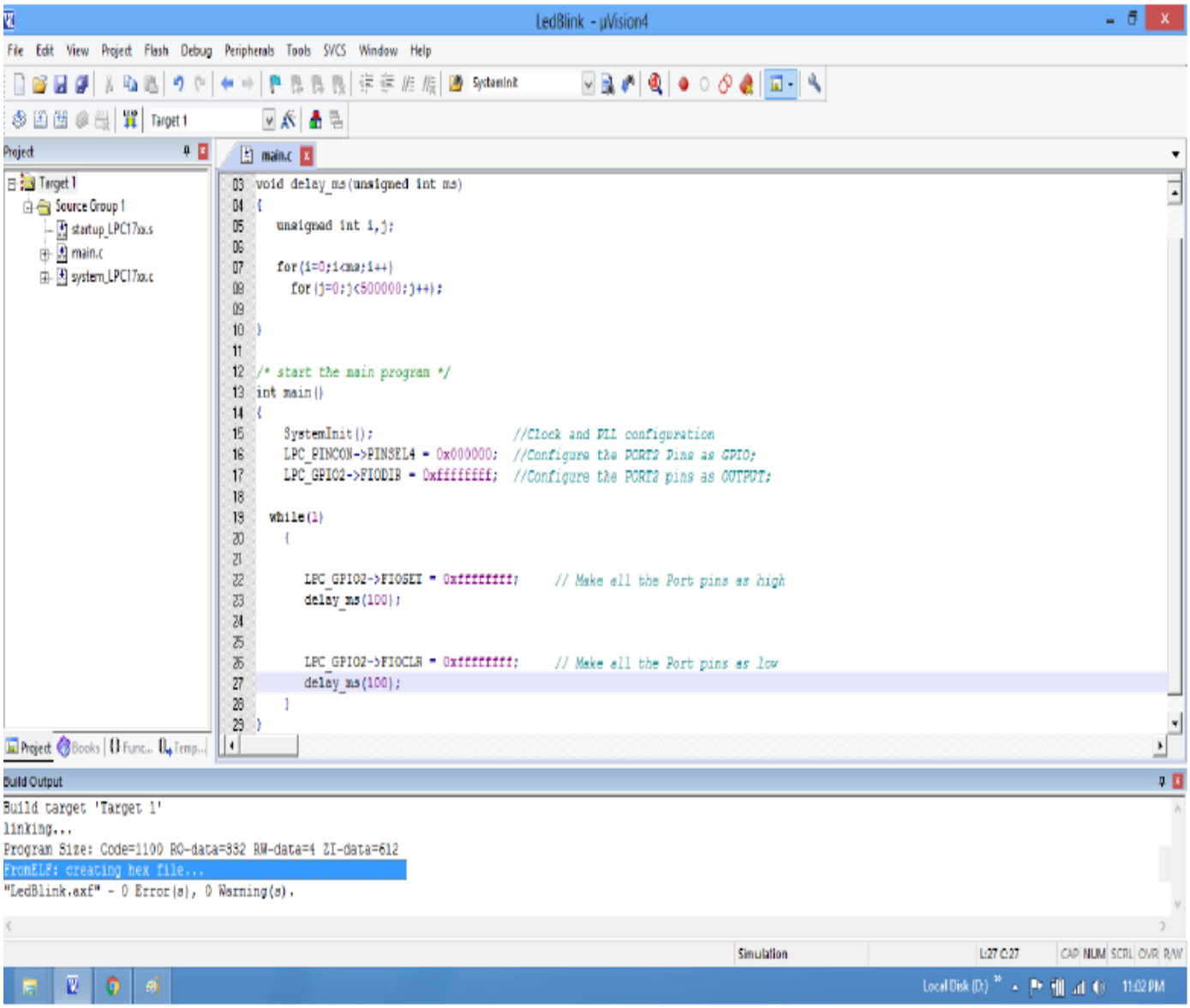
3. Select IRAM1 (starting 0×10000000 size 0×8000)



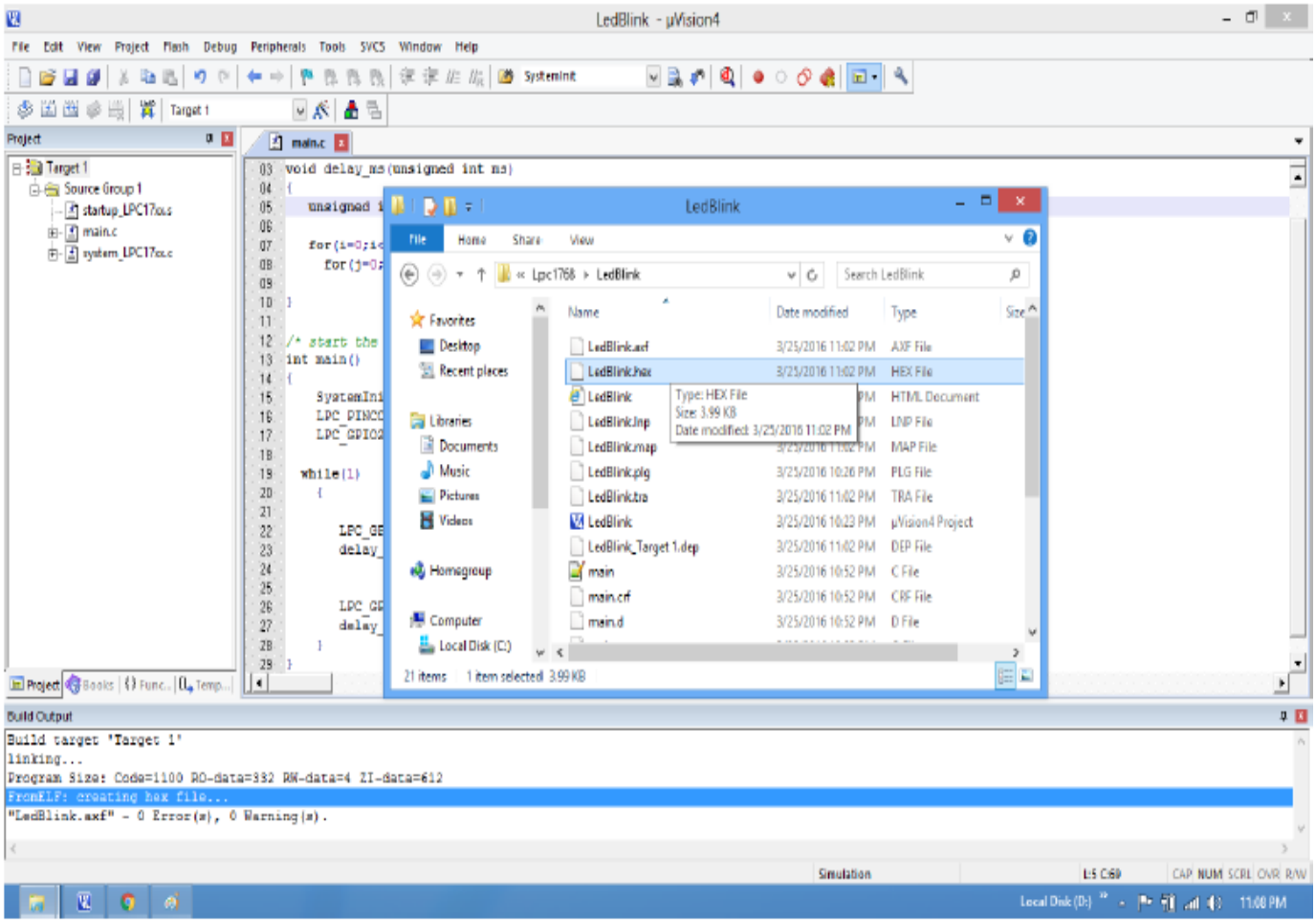
Step15: Enable the option to generate the .hex file



Step16: .Hex file is generated after a rebuild.



Step17: Check the project folder for the generated .hex file.



Now .hex file with project name will be generated.

Working with Flash Magic Software:-

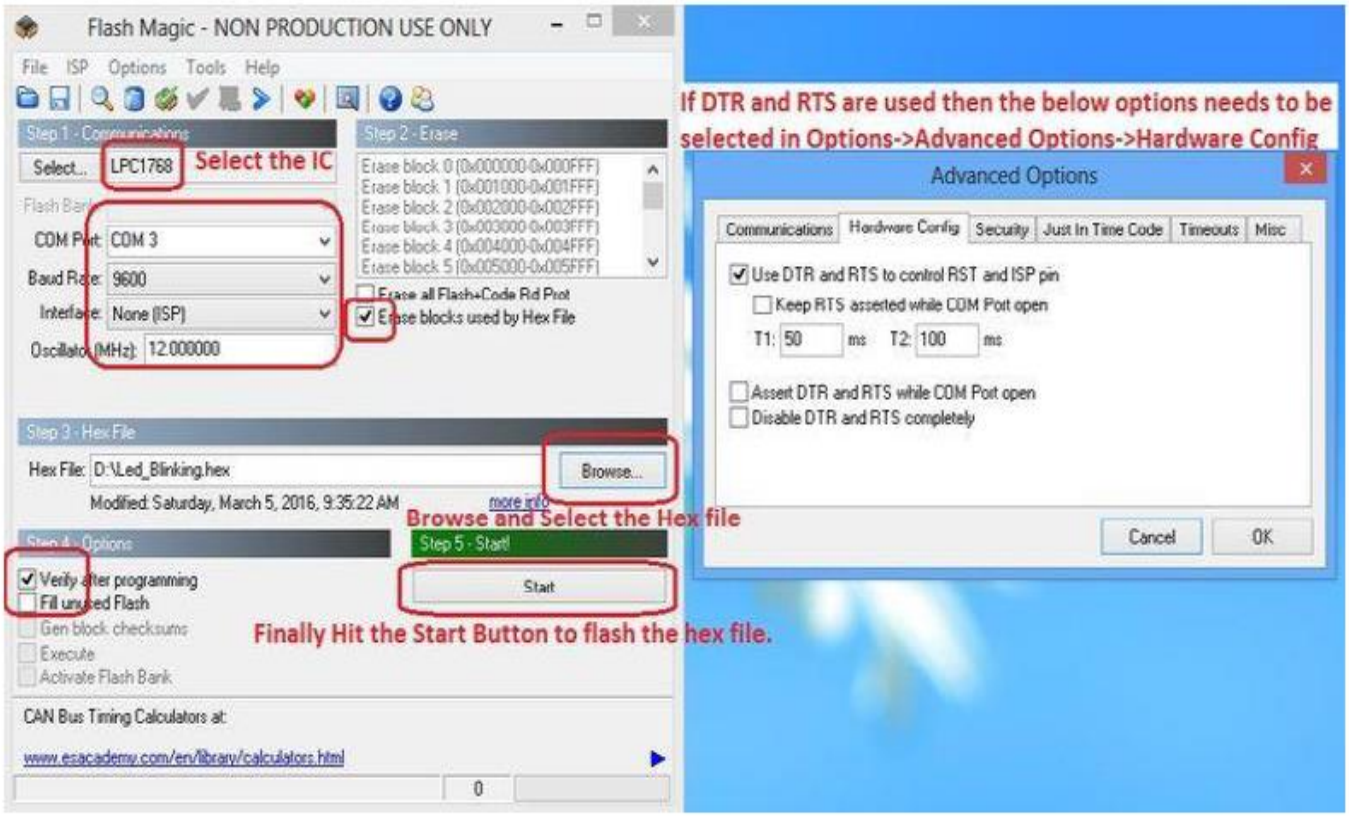
STEPS TO BE FOLLOWED IN FLASH MAGIC 6.01 TOOLS

Connect 9 pin DSUB USB-to-SERIAL cable from PC to DB2 at the board

On the 2 way dip switch SW3 and Short jumper JP7

Now open the flash magic software, some settings in FLASH MAGIC and follow the below steps.

1. Select the IC from Select Menu (LPC1768).
2. Select the COM Port. Check the device manager for detected Com port.
3. Select Baud rate from 9600
4. Select None ISP Option.
5. Oscillator Frequency 12.000000(12 MHz).
6. Check the Erase blocks used by Hex file option
7. Browse and Select the hex file.
8. Check the Verify After Programming Option.
9. If DTR and RTS are used then go to Options->Advanced Options-> Hardware Config and select the Use DTR and RTS Option.
10. Hit the Start Button to flash the hex file.
11. Once the hex file is flashed, Reset the board. Now the controller should run your application code.



QUICK REFERENCE

PORT LINE DETAILS – Used for on board interfaces

Sl No	Program Name	Port Line
1	LCD	P2.4-P2.9
2	7 SEG DISPLAY	P0.4-0.11 & P0.19-P0.20
3	STEPPER MOTOR	P2.0-P2.3
4	DC MOTOR	P1.24 & P0.26
5	RELAY	P0.25
6	BUZZER	P0.24
7	EXT INERRUPT 3	P2.13
8	DAC0800	P0.4 – P0.11
9	PWM	P3.25
10	INTERNAL ADC	P1.31
11	4X4 KEY MATRIX	P1.16-P1.23
12	SPI	P0.17-P0.20
13	SW4	P2.11
14	L2	P2.12

PROGRAM 1:
Display "Hello world" message using UART

UART module and Registers: LPC1768 has 4-UARTs numbering 0-3, similarly pins are marked as RxD0-RxD3 and TxD0-TxD3. As the LPC1768 pins are multiplexed for multiple functionalities, first they have to be configured as UART pins.

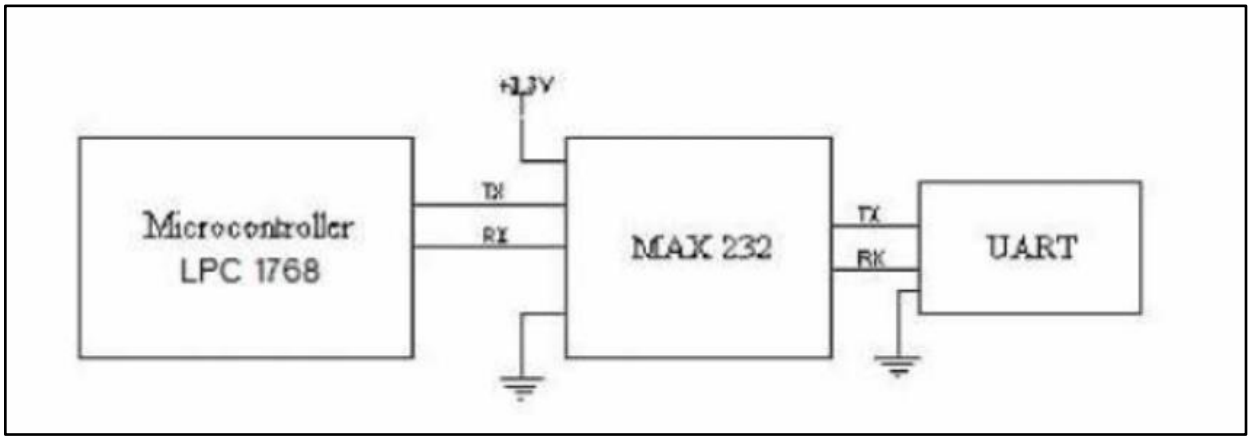


Figure: Usage of internal UART

```
#include   <LPC17xx.h>
void delay(unsigned int r1);
void UART0_Init(void);
unsignedinti ;
unsigned char *ptr, arr[] = "Hello World\r";
#define THR_EMPTY 0x20                                          //U0THR is Empty
int main(void)
{
        UART0_Init();
        while(1)
        {
                ptr = arr;
                while ( *ptr != '\0')
                {
                        while ((LPC_UART0->LSR & THR_EMPTY) != THR_EMPTY) ;
                        // if LSR 5th bit = 0 wait for data
                        // if LSR 5th bit=1 THR has valid data to be transmitted.
                        //Is U0THR is EMPTY??
                        LPC_UART0->THR = *ptr++;
                }
                for(i=0;i<=60000;i++);              // delay
        }
}
```

```
void UART0_Init(void)
{
        LPC_SC->PCONP |= 0x00000008;                    //UART0 peripheral enable
        LPC_PINCON->PINSEL0 = 0x00000050;               //P0.2 used as TxD0, P0.3 used as RxD0
        LPC_UART0->LCR = 0x00000083;                    //enable divisor latch, parity disable,
                                                        //1 stop bit, 8 bit word length
        LPC_UART0->DLM = 0X00;
        LPC_UART0->DLL = 0x1A;                          //select baud rate 9600 bps for 4 MHz
        LPC_UART0->LCR = 0X00000003;                    //Disable divisor latch
        LPC_UART0->FCR = 0x07;                  //FIFO enable, RX FIFO reset, TX FIFO reset
}
```

RESULT:LPC1768 ARM microcontroller is programmed to interface microcontroller and PC using UART. A message “Hello world” is transmitted from the microcontroller using UART interface. The message is received in the PC and displayed in a command window.

PROGRAM 2:

Interface and control a DC motor

A DC motor is an Electrical machine which converts Electrical energy into Mechanical energy. It works on the principle that whenever a current carrying conductor is placed in a magnetic field, it experiences a mechanical force.

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.

```
/*********************************************************************************
DC MOTOR CONTROL USING ON CHIP PWM
Controller: LPC1768
Description:
Port line P1.24 is used for the PWM feature. When T0TC matches the MR0 counts interrupt is generated
and Duty cycle will be changed. Depends upon the Duty cycle Motor speed also changes.
PWM 1.5 is used. Match register 0 is used for count purpose.
PWM 1 match register 5 is incremented or decremented at each interrupted
*********************************************************************************/
SPEED CONTROL OF DC MOTOR

#include <lpc17xx.h>
voidpwm_init(void);
void PWM1_IRQHandler(void);
unsigned long int i;
unsigned char flag,flag1;
int main(void)
{
        SystemInit();
        SystemCoreClockUpdate();
        pwm_init();
        while(1){
                for(i=0;i<=1000;i++);                   // delay
        }//end of while
}//end of main

voidpwm_init(void)
{
        LPC_SC->PCONP = (1<<6);                 //PWM1 is powered
        LPC_PINCON->PINSEL3 = 0x00020000;   //pwm1.5 is selected for the pin P1.24
        LPC_PWM1->PR  = 0x00000000;             //Count frequency : Fpclk
```

```
        LPC_PWM1->PCR = 0x0002000;   //select PWM5 single edge and PWM5 output is enabled
        LPC_PWM1->MCR = 0x00000003;          //Reset and interrupt on PWMMR0
        LPC_PWM1->MR0 = 60000;               //setup match register 0 count
        LPC_PWM1->MR5 = 0x00000100;          //setup match register MR5
        LPC_PWM1->LER = 0x000000FF;          //enable shadow copy register
        LPC_PWM1->TCR = 0x00000002;          //RESET COUNTER AND PRESCALER
        LPC_PWM1->TCR = 0x00000009;          //enable PWM and counter
        NVIC_EnableIRQ (PWM1_IRQn);
return;
}
void PWM1_IRQHandler(void)
{
        LPC_PWM1->IR = 0xff;                            //clear the interrupts
        if(flag == 0x00)
        {
        LPC_PWM1->MR5 += 100;
        LPC_PWM1->LER = 0x000000FF;
                if(LPC_PWM1->MR5 >= 57000)     //for 10% of duty cycle MR5=3000,
        //for 50%, MR5=15000, for 90% MR5= 27000                                        {
                        flag1 = 0xff;
                        flag = 0xff;
                        LPC_PWM1->LER = 0x000000fF;
                 }
                for(i=0;i<8000;i++);
        }
else if(flag1 == 0xff)
   {
                        LPC_PWM1->MR5 -= 100;
                        LPC_PWM1->LER = 0x000000FF;
                        if(LPC_PWM1->MR5 <= 0x300)              //1% of duty cycle MR5=300
                        {
                                flag  = 0x00;
                                flag1 = 0x00;
                                LPC_PWM1->LER = 0X000000fF;
                        }
                for(i=0;i<8000;i++);
         }
}
```

DIRECTION CONTROL OF DC MOTOR

```
/*****************************************************************************
Description:
Direction of the DCM is controlled in this software by alternatively interchanging the supply with the help of Relay.
Port lines: P1.24 and P0.26.
Port line P1.24 used for the PWM feature. When T0TC matches the MR0 counts interrupt is generated and duty cycle will be changed. Depends upon the Duty cycle Motor speed also changes.
PWM 1.5 is used. Match register 0 is used for count purpose.
PWM 1 match register is incremented or decremented at each interrupted.
*****************************************************************************/
#include  <LPC17xx.H>
voidClock_Wise(void);
voidAClock_Wise(void);
unsigned long i;
int main(void)
{
        LPC_PINCON->PINSEL1 = 0x00000000;   //P0.26 GPIO, P0.26 controls direction
        LPC_PINCON->PINSEL3 = 0x00000000;   //P1.24 GPIO
        LPC_GPIO0->FIODIR |= 0x04000000;    //P0.26 output
        LPC_GPIO1->FIODIR |= 0x01000000;    //P1.24 output
        while(1)
        {
                Clock_Wise();
                for(i=0;i<300000;i++);
                AClock_Wise();
                for(i=0;i<300000;i++);
        }               //end while(1)
}                       //end main

voidClock_Wise(void)
{
        LPC_GPIO1->FIOCLR = 0x01000000;     //P1.24 Kept low to off DCM
        for(i=0;i<10000;i++);               //delay to compensate inertia
        LPC_GPIO0->FIOSET = 0x04000000;     //coil is on
        LPC_GPIO1->FIOSET = 0x01000000;     //motor in on
}                                           //end void Clock_Wise(void)
```

```
voidAClock_Wise(void)
{
        LPC_GPIO1->FIOCLR = 0x01000000;     //P1.24 Kept low to off DCM
        for(i=0;i<10000;i++);               //delay to componsate inertia
        LPC_GPIO0->FIOCLR = 0x04000000;     //coil is off
        LPC_GPIO1->FIOSET = 0x01000000;     //Motor is on
}                                           //end void AClock_Wise(void)
```

RESULT: LPC1768 ARM microcontroller is programmed to interface and control a DC motor.

PROGRAM 3:

Interface a stepper motor and rotate it in clockwise and anti-clockwise direction

A stepper motor (also referred to as step or stepping motor) is an electromechanical device achieving mechanical movements through conversion of electrical pulses. Stepping motors can be viewed as electric motors without commutators. Typically, all winding in the motor are part of the stator, and the rotor is either a permanent magnet or, in the case of variable reluctance motors, a toothed block of some magnetically soft material. All of the commutation must be handled externally by the motor controller, and typically, the motors and controllers are designed so that the motor may be held in any fixed position as well as being rotated one way or the other.

Stepper motors are driven by digital pulses rather than by a continuous applied voltage. Unlike conventional electric motors which rotate continuously, stepper motors rotate or step in fixed angular increments. A stepper motor is most commonly used for position control. With a stepper motor/driver/controller system design, it is assumed the stepper motor will follow digital instructions. Most steppers can be stepped at audio frequencies, allowing them to spin quite quickly, and with an appropriate controller, they may be started and stopped at controlled orientations.

In this lab, we'll use a so called "five wire stepper" shown in the figure below. This 28BYJ48 stepper motor is driven via a driver board that contains 4 Darlington drivers (ULN2003) and 4 LEDs.
PM1 – it's a 5 pin straight male power mate.

A stepper motor direction is controlled by shifting the voltage across the coils. Port lines: P2.0 to P2.3.

```
/*****************************************************************************
Stepper motor Direction control
Controller: LPC1768
Description: A stepper motor direction is controlled by shifting the voltage across the coils.
Port lines: P2.0 to P2.3.
*****************************************************************************/
```

0	0	0	0	0	0	0	0

FIODIR By default all bits are zero

Direction: 0001 1
0010 2

0100 4
1000 8

PINSEL: Each Port has 2 PINSEL Registers

Port 2 [PINSEL4 P2.0 –P2.15
[PINSEL5 P2.16 – P2.31
For this program, we use PINSEL4 (Since P2.0 – P2.3 is required)
By default, PINSEL4 is 0xFFFFFFFF

```
#include  <LPC17xx.H>
voidclock_wise(void);
voidanti_clock_wise(void);
unsigned long int var1,var2;
unsignedinti=0,j=0,k=0;
int main(void)
{
        LPC_PINCON->PINSEL4 = 0x00000000;          //P2.0 to P2.3 GPIO
        LPC_GPIO2->FIODIR = 0x0000000F;            //P2.0 to P2.3 output
        while(1)
        {
                for(j=0;j<50;j++)                  //50 times in Clock wise Rotation
                clock_wise();
                for(k=0;k<65000;k++);              //Delay to show  Anti-clock Rotation
                for(j=0;j<50;j++)                  /50 times in  Anti-clock wise Rotation
                anti_clock_wise();
                for(k=0;k<65000;k++);              //Delay to show clock Rotation
        }
                        //End of while(1)
}                               //End of main
voidclock_wise(void)
{
  var1 = 0x00000001;                    //For Clockwise
for(i=0;i<=3;i++)                       //for A B C D Stepping
        {
```

```
          LPC_GPIO2->FIOCLR =  0X0000000F;
          LPC_GPIO2->FIOSET =  var1;
         var1 = var1<<1;                          //For Clockwise
for(k=0;k<3000;k++);                        //for step speed variation
  }
}
voidanti_clock_wise(void)
{
var1 = 0x0000008;                           //For Anticlockwise
for(i=0;i<=3;i++)                           //for A B C D Stepping
   {
        LPC_GPIO2->FIOCLR =  0X0000000F;
        LPC_GPIO2->FIOSET =  var1;
        var1 = var1>>1;                     //For Anticlockwise
for(k=0;k<3000;k++);                        //for step speed variation
   }
}
```

RESULT:LPC1768 ARM microcontroller is programmed to interface a stepper motor. Stepper motor is rotated in both clockwise and anti-clockwise direction. The direction of rotation in the motor is changed after a certain delay.

PROGRAM 5:

Interface a DAC and generate Triangular and Square waveforms Square Waveform

```
/*****************************************************************************
External DAC interface (Square Wave)
Controller: LPC1768
Description: This example explains about how Square Wave is generated.
P0.4 to P0.11 are used to get the Digital values.
*****************************************************************************/
```

PINSEL0

0000	0000	0000	0000	0000	0000	0000	0000
0	0	0	0	0	0	0	0

GPIO0->FIODIR

0000	0000	0000	0000	0000	1111	1111	0000
0	0	0	0	0	F	F	0

FIOMASK

1111	1111	1111	1111	1111	0000	0000	1111
0	0	0	0	0	F	F	0

```
#include <LPC17xx.H>
void delay(void);
int main ()
{
LPC_PINCON->PINSEL0 = 0x00000000;   // Configure P0.4 to P0.11 as GPIO
 LPC_GPIO0->FIODIR   = 0x00000FF0 ;     // Configure P0.4 to P0.11 as Output
LPC_GPIO0->FIOMASK = 0XFFFFF00F;
while(1)
{
   LPC_GPIO0->FIOSET  = 0x00000FF0 ;
delay();
   LPC_GPIO0->FIOCLR  = 0x00000FF0 ;
delay();
   }
}

void delay(void)
{
        unsignedinti=0;
        for(i=0;i<=9500;i++);
}
```

Triangular Waveform

```
/*********************************************************************************
External DAC interface (Triangular Waveform)
Controller: LPC1768
Description: This example explains about how Triangular Wave is generated.
P0.4 to P0.11 are used to get the Digital values.
*********************************************************************************/
#include <LPC17xx.H>
int main ()
{
        unsigned long int temp=0x00000000;
        unsignedinti=0;
        LPC_PINCON->PINSEL0 = 0x00000000;          // Configure P0.4 to P0.11 as GPIO
        LPC_GPIO0->FIODIR   = 0x00000FF0 ;         // Configure P0.4 to P0.11 as Outputs
        LPC_GPIO0->FIOMASK  = 0XFFFFF00F;
        while(1)
        {
                for(i=0;i!=0xFF;i++)                    /*output 0 to FE*/
                {
                        temp=i;
                        temp = temp << 4;
                        LPC_GPIO0->FIOPIN = temp;
                 }
                for(i=0xFF; i!=0;i--)         /* output FF to 1*/
                {
                        temp=i;
                        temp = temp << 4;
                        LPC_GPIO0->FIOPIN = temp;
                  }
        }//End of while(1)
}//End of main()
```

RESULT:LPC1768 - ARM CORTEX M3 microcontroller is programmed to generate waveforms using DAC. DAC is interfaced with the microcontroller and the output is observed on CRO, CRO is connected to the output pin of DAC. The digital value is converted into analog value using DAC and displayed on CRO. The amplitude and frequency of generated triangular wave are _____ Volts and _____ Hertz, respectively. The amplitude and frequency of generated square wave are _____ Volts and _____ Hertz, respectively.

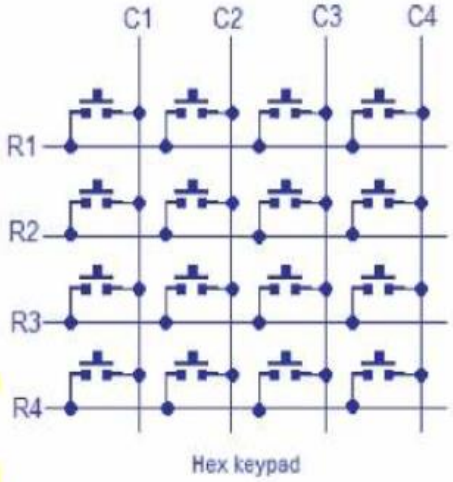
PROGRAM 5:
Interface a 4x4 keyboard and display the key code on an LCD

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	R1	R2	R3	R4	C1	C2	C3	C4
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



Hex keypad

```
#include <LPC17xx.h>
#include "lcd.h"
void scan(void);
unsigned char col,row,var,flag,key,*ptr;
unsigned long int i,var1,temp,temp3;
unsigned char SCAN_CODE[16] = {0x1E,0x1D,0x1B,0x17,
                               0x2E,0x2D,0x2B,0x27,
                               0x4E,0x4D,0x4B,0x47,
                               0x8E,0x8D,0x8B,0x87};

unsigned char ASCII_CODE[16] =  {'0','1','2','3',
                                 '4','5','6','7',
                                 '8','9','A','B',
                                 'C','D','E','F'};

int main(void)
{
        LPC_PINCON->PINSEL3 = 0x00000000;          //P1.20 to P1.23 MADE GPIO
        LPC_PINCON->PINSEL0 = 0x00000000;          //P0.15 as GPIO
        LPC_PINCON->PINSEL1 = 0x00000000;          //P0.16 t0 P0.18 made GPIO
        LPC_GPIO0->FIODIR &= ~0x00078000;      //made INput P0.15 to P0.18 (cols)
        LPC_GPIO1->FIODIR |= 0x00F00000;       //made output P1.20 to P1.23 (rows)
        LPC_GPIO1->FIOSET = 0x00F00000;
        lcd_init();

        while(1)
        {
                while(1)
                {
                        for(row=1;row<5;row++)
                        {
                                if(row == 1)
                                var1 = 0x00100000;
                                else if(row == 2)
                                var1 = 0x00200000;
                                else if(row == 3)
                                var1 = 0x00400000;
                                else if(row == 4)
                                var1 = 0x00800000;
```

```
                        temp = var1;
                        LPC_GPIO1->FIOSET = 0x00F00000;
                        LPC_GPIO1->FIOCLR = var1;
                        flag = 0;
                        scan();
                        if(flag == 1)
                        break;

                }
                                //end for(row=1;row<5;row++)
                if(flag == 1)
                break;

        }
                                        //2nd while(1)
        for(i=0;i<16;i++)
        {
                if(key == SCAN_CODE[i])
                {
                        key = ASCII_CODE[i];
                        break;
                }
                        //end if(key == SCAN_CODE[i])
        }
                                //end for(i=0;i<16;i++)
        temp1 = 0x80;
        lcd_com();
        delay_lcd(800);
        lcd_puts(&key);

    }
                        //end while 1
}
                        //end main
void scan(void)
{
    unsigned long temp3;
    temp3 = LPC_GPIO0->FIOPIN;
    temp3 &= 0x00078000;
    if(temp3 != 0x00078000)
```

```
        {
                        flag = 1;
                        temp3 >>= 15;                   //Shifted to come at LN of byte
                        temp>>= 16;                     //shifted to come at HN of byte
                        key = temp3|temp;
        }
                                //1st if(temp3 != 0x00000000)
}
                                //end scan
```

RESULT: LPC1768 ARM microcontroller is programmed to interface a 4x4 keyboard. The key press is captured through the interface and the value of pressed key is computed in the program. Then, the key value or code is displayed on an LCD screen.

PROGRAM 6:

Demonstrate the use of an external interrupt to toggle an LED On/ Off

```
#include  <LPC17xx.h>
void EINT3_IRQHandler(void);
unsigned char int3_flag=0;
unsignedint z;
int main(void)
{
LPC_PINCON->PINSEL4 = 0x04000000;          //P2.13 as EINT3
LPC_GPIO2->FIODIR = 0x00001000;            //P2.12 is assigned output
LPC_GPIO2->FIOSET = 0x00001000;            //Initial LED is kept on
LPC_SC->EXTINT = 0x00000008;  //writing 1 clears the interrupt, get set if there is interrupt
LPC_SC->EXTMODE = 0x00000008;//EINT3 is initiated as edge sensitive, 0 for level sensitive
LPC_SC->EXTPOLAR = 0x00000000;       //EINT3 is falling edge sensitive, 1 for rising edge

            //above registers, bit0-EINT0, bit1-EINT1, bit2-EINT2,bit3-EINT3
NVIC_EnableIRQ(EINT3_IRQn);                          //core_cm3.h
while(1) ;
}
void EINT3_IRQHandler(void)
{
        LPC_SC->EXTINT = 0x00000008;                          //cleares the interrupt
        if(int3_flag == 0x00)                                 //when flag is '0' off the LED
                {
                        LPC_GPIO2->FIOCLR = 0x00001000;
                        for(z=0;z<=300000;z++);
                        int3_flag = 0xff;
                }
                else                                          //when flag is FF on the LED
                {
                        LPC_GPIO2->FIOSET = 0x00001000;
                        for(z=0;z<=300000;z++);
                        int3_flag = 0;
                }
}
```

RESULT: LPC1768 ARM microcontroller is programmed to interface an external interrupt to toggle an LED On/ Off. The external interrupt is generated using a push button. When the push button is pressed, the LED which is in ‘Off’ state will turn ‘On’ and vice-versa

PROGRAM 7:

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

```
/*****************************************************************************
SEVEN SEGMENT DISPLAY
Controller: LPC1768
Description: DISPLAYS ARE CONNECTED IN COMMON CATHODE MODE
Port0 Connected to data lines of all 7 segment displays
*****************************************************************************
        a
         ----
        f|  g |b
         |----|
        e|    |c
         ----  . dot
        d
        a = P0.04
        b = P0.05
        c = P0.06
        d = P0.07
        e = P0.08
        f = P0.09
        g = P0.10
        dot = P0.11
Select lines for two 7 Segments
DIS1 P0.19
DIS2 P0.20
*****************************************************************************/
```

GPIO0

20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0

1	8	0	f	f	0

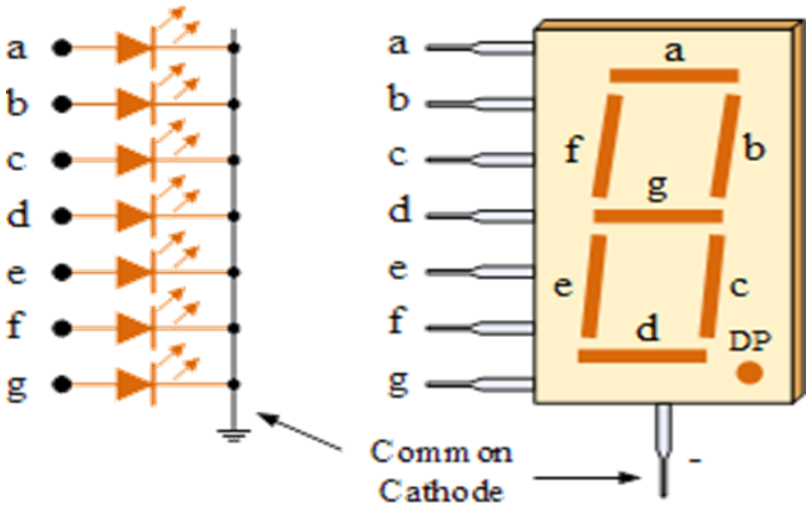


Figure: common cathode display (to glow particular segment pass logic 1)

	h	g	f	e	d	c	b	a	hex value
0	0	0	1	1	1	1	1	1	3F
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	0	1	1	1	1	6F
A	0	1	1	1	0	1	1	1	77
b	0	1	1	1	1	1	0	0	7C
C	0	0	1	1	1	0	0	1	39
d	0	1	0	1	1	1	1	0	5E
E	0	1	1	1	1	0	0	1	79
F	0	1	1	1	0	0	0	1	71

Table: hexadecimal equivalent values for each digit to be displayed

```
#include  <LPC17xx.h>
Unsigned intdelay,  Switchcount=0,j;

unsigned intDisp[16] = {0x000003f0, 0x00000060, 0x000005b0, 0x000004f0, 0x00000660, 0x000006d0,0x000007d0, 0x00000070, 0x000007f0, 0x000006f0, 0x00000770,0x000007c0, 0x00000390, 0x000005e0, 0x00000790, 0x00000710 };

#define ALLDISP  0x00180000                                    //Select all display
#define DATAPORT 0x00000ff0          //P0.4 to P0.11 : Data lines connected to drive Seven Segments
int main (void)
{
        LPC_PINCON->PINSEL0 = 0x00000000;
        LPC_PINCON->PINSEL1 = 0x00000000;
        LPC_GPIO0->FIODIR =       0x00180ff0;

        while(1)
        {
        LPC_GPIO0->FIOSET |= 0x00100000;
        LPC_GPIO0->FIOCLR =0x00000ff0;            // clear the data lines to 7-segment displays
        LPC_GPIO0->FIOSET = Disp[Switchcount];  // get the 7-segment display value from the array
                        for(j=0;j<3;j++)
                        for(delay=0;delay<30000;delay++);                        // delay
                        Switchcount++;
                        if(Switchcount == 0x10)        // 0 to F has been displayed ? go back to 0
                        {
                                Switchcount = 0;
                                  LPC_GPIO0->FIOCLR  =  0x00180ff0;
                         }
        }
}
```

RESULT: LPC1768 ARM microcontroller is programmed to display hexadecimal digits on 7-segment LED. The equivalent code for displaying hexadecimal digits is stored in the program. The code is provided as input to the LED to display an hexadecimal digit. The displayed digit is changed after a delay and all the hexadecimal digits are displayed repetitively.

PROGRAM 8:

Measure Ambient temperature using a sensor and SPI ADC IC

```
#include  <LPC17xx.h>
#include  <stdio.h>
#include "UART0.h"
#include "SPI.h"
#define VREF            5280
#define FULLSCALE 4095
#define SSEL            0x00010000

unsigned char temp1=0x00;
unsignedint up4bits,low8bits;
unsigned char *ptr,*ptr1,adc_val[8],temp_float[12];
unsigned char temp_op[]= "TEMP('C)=";
floatanalog_val,temp_valK,temp_valC;
volatile unsigned long i=0,j = 0;
unsignedintavg;

int main(void)
{
LPC_GPIO0->FIODIR |= SSEL;                  //CHIPSELECT LINE MADE AS O/P P0.7
LPC_GPIO0->FIOSET = SSEL;   //if device is powered up with CS low, set high before comm
UART0_Init();                                                   //Initializes UART0
SPI_Init();                                             //Initializes SPI
spi_flag=0;
while(1)
        {
                        for(i=0;i<8;i++)
                        {
                        LPC_GPIO0->FIOCLR = SSEL;                       //make SS low
                        for(j=0;j<1000;j++);
                        LPC_SPI->SPDR = 0x01;                           //Send the start bit
        while(spi_flag == 0);
        spi_flag = 0;
        LPC_SPI->SPDR = 0xC0;     //Select channel in single ended mode & MSB first format
        while(spi_flag == 0);
        spi_flag = 0;
        up4bits = temp;                                         //temp has SPI data register value
        LPC_SPI->SPDR = 0x00;
        while(spi_flag == 0);
```

```
        spi_flag = 0;
                low8bits = temp;                                    //SPI Data Register.
        up4bits = (up4bits & 0x0f)<<8;
        low8bits|=up4bits;                                  //combine the lower bits and upper bits
        avg += low8bits;
        LPC_GPIO0->FIOSET = SSEL;                                   //make SS high
        for(j=0;j<1000;j++);
                                }

        avg>>= 3;
        analog_val = (float) ( avg * (float)VREF)/(float)FULLSCALE ;
        temp_valK = (float)( (analog_val * 273.14)/(2731.4));       //temperature in deg K
        temp_valC      = temp_valK - 273.14;                        //temperature in deg C
        sprintf(temp_float,"%3.2f",temp_valC);
        avg = 0x00;
        ptr = temp_op;

while(*ptr!='\0')
      {
        LPC_UART0->THR = *ptr++;
                while(tx0_flag == 0);
                tx0_flag= 0;
                for(i=0;i<200;i++);
                                }
        ptr1 = temp_float;
while(*ptr1!='\0')
      {
        LPC_UART0->THR = *ptr1++;
        while(tx0_flag == 0);
                tx0_flag= 0;
        for(i=0;i<200;i++);
                                }
for(j=0;j<65000;j++); //delay for display // for 1 sec value 400000//for 10000 counts 25msec.
LPC_UART0->THR = 0x0D;                                              //new line
        while(tx0_flag == 0);
        tx0_flag= 0;
  }
}
```

SPI INTERFACE

```
#include <LPC17xx.h>
#include "SPI.h"
unsigned char spi_flag = 0, temp=0;
voidSPI_Init(void)
{
        LPC_PINCON->PINSEL0 |= 0xC0000000;  //P0.15 as SCK
        LPC_PINCON->PINSEL1 = 0x0000003C;   //select MISO-P0.17,MOSI-P0.18
        LPC_SPI->SPCCR = 0x1E; //  SPI CLOCK SELECTED AS 100KHZ
        LPC_SPI->SPCR = 0xA0; //8 bit data, actve high clk, master SPI mode,SPIInt enable
                // Master mode and SCK line is active high
        LPC_SPI->SPINT  = 0x01; //clear the interrupt flag
        NVIC_EnableIRQ(SPI_IRQn);
}

voidSPI_IRQHandler(void)
{
spi_flag = 1;
temp = LPC_SPI->SPSR;   // To clear SPIF bit we have to read status register.
temp = LPC_SPI->SPDR;   // Then read the data register(optional)
   LPC_SPI->SPINT = 0x01;  // To clear the SPI interrupt
}
```

UART INTERFACE

```
#include <LPC17xx.h>
#include "uart0.h"
unsigned char recv_buf[50], recv_data=0, recv_index=0;
unsigned char rx0_flag=0, tx0_flag=0;
void UART0_Init(void)
{
        LPC_SC->PCONP |= 0x00000008;   //UART0 peripheral enable
        LPC_PINCON->PINSEL0 &= ~0x000000F0;
        LPC_PINCON->PINSEL0 |= 0x00000050;
        LPC_UART0->LCR = 0x00000083; //enable divisor latch, parity disable, 1 stop bit, 8bit word
length
        LPC_UART0->DLM = 0X00;
        LPC_UART0->DLL = 0x1A;          //select baud rate 9600 bps
        LPC_UART0->LCR = 0X00000003;
        LPC_UART0->FCR = 0x07;
        LPC_UART0->IER = 0X03;                  //select Transmit and receive interrupt
        NVIC_EnableIRQ(UART0_IRQn);             //Assigning channel
}
void UART0_IRQHandler(void)
{
        unsigned long Int_Stat;
        Int_Stat = LPC_UART0->IIR;          //reading the data from interrupt identification register
        Int_Stat = Int_Stat& 0x06;              //masking other than txmitint&rcve data indicator
        if((Int_Stat& 0x02)== 0x02)  //transmit interrupt
                tx0_flag = 0xff;
        else if( (Int_Stat& 0x04) == 0x04)  //recve data availble
        {
                recv_data = LPC_UART0->RBR;
                rx0_flag = 0xff;
        }
}
```

Result:LPC1768 ARM microcontroller is programmed to record the room's ambient temperature using temperature sensor. The temperature obtained from the sensor is converted into a digital value using SPI ADC.

PART A:

ALP to multiply two 16 bit binary numbers.

Program:

```
;/* PROGRAM TO MULTIPLY TWO 16BIT NUMBERS
                */
;/*      VALUE1:     1900H (6400)          (IN R1)
                    */
        ;/*     VALUE2:     0C80H(3200)           (IN R2)
                                */
        ;/*     RESULT:     1388000H(20480000)(IN R3)
                                */
        ;/* SET A BREAKPOINT AT NOP INSTRUCTION,RUN THE PROGRAM & CHECK THE
        RESULT      */
        ;/* PROGRAM WRITTEN BY ALS R&D TEAM BENGALURU DATE:08/08/2011
                    */

        AREA  MULTIPLY , CODE, READONLY
                    EXPORT __main
        __main
        ENTRY                           ;Mark first instruction to execute

        START

              MOV r1,#6400              ; STORE FIRST NUMBER IN R0
              MOV r2,#3200              ; STORE SECOND NUMBER IN R1
              MUL r3,r1,r2              ; MULTIPLICATION

        NOP
        NOP
        NOP

        END                         ;Mark end of file
```

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

```
;/* PROGRAM TO FIND THE SUM OF FIRST 10 INTEGER NUMBER
            */
;/* In this example we have taken n=10
    */
;/* Check the result in R0/R3 register =37H (55)                                    */
;/* SET A BREAKPOINT AT NOP INSTRUCTION,RUN THE PROGRAM & CHECK THE
RESULT   */
;/* PROGRAM WRITTEN BY ALS R&D TEAM BENGALURU      DATE:08/08/2011
            */

AREA  ADD1to10, CODE, READONLY
    EXPORT __main
__main
            MOV r0, #10                 ; STORE SUM NUMBER IN R0
            MOV r1,r0                   ; MOVE THE SAME NUMBER IN R1

rpt
            SUBS r1, r1, #1                         ; SUBTRACTION
            CMP r1, #0                  ; COMPARISON
            BEQ STOP
            ADD r3,r0,r1;           ; ADDITION
            MOV   r0,r3                             ; Result
            BNE rpt                 ; BRANCH TO THE LOOP IF NOT EQUAL
STOP

            NOP
            NOP
            NOP

    END                             ;Mark end of file
```

3. ALP to find the number of 0's and 1's in a 32 bit data

```
;/* PROGRAM TO COUNT THE NUMBER OF ONES & ZEROS IN TWO CONSECUTIVE
MEMORY LOCATIONS*/
;/*  WE TOOK TWO NUMBERS i.e. 0X11111111,0XAA55AA55      (R0)
                    */
;/*  CHECK THE RESULT IN R2 FOR ONES & R3 FOR ZEROS
                          */
;/* SET A BREAKPOINT AT NOP INSTRUCTION,RUN THE PROGRAM & CHECK THE
RESULT                    */
;/* PROGRAM WRITTEN BY ALS R&D TEAM BENGALURU       DATE:08/08/2011
                          */

AREA  ONEZERO , CODE, READONLY
    EXPORT __main
__main
ENTRY                           ;Mark first instruction to execute

START

            MOV R2,#0                       ; COUNTER FOR ONES
            MOV R3,#0                       ; COUNTER FOR ZEROS
            MOV R7,#2                       ; COUNTER TO GET TWO WORDS
            LDR R6,=VALUE                   ; LOADS THE ADDRESS OF VALUE

LOOP        MOV R1,#32                      ; 32 BITS COUNTER
       LDR R0,[R6],#4                  ; GET THE 32 BIT VALUE

LOOP0       MOVS R0,R0,ROR #1               ; RIGHT SHIFT TO CHECK CARRY BIT (1's/0's)
            BHI ONES                        ; IF CARRY BIT IS 1 GOTO ONES BRANCH
OTHERWISE NEXT

ZEROS       ADD R3,R3,#1                    ; IF CARRY BIT IS 0 THEN INCREMENT THE
COUNTER BY 1(R3)
            B LOOP1                                 ; BRANCH TO LOOP1

ONES        ADD R2,R2,#1                    ; IF CARRY BIT IS 1 THEN INCREMENT THE
COUNTER BY 1(R2)

LOOP1       SUBS R1,R1,#1                   ; COUNTER VALUE DECREMENTED BY 1
            BNE LOOP0                       ; IF NOT EQUAL GOTO TO LOOP0 CHECKS
32BIT
```

```
        SUBS R7,R7,#1                ; COUNTER VALUE DECREMENTED BY 1
        CMP R7,#0                    ; COMPARE COUNTER R7 TO 0
        BNE LOOP                     ; IF NOT EQUAL GOTO TO LOOP

    NOP
    NOP
    NOP


VALUE DCD 0X11111111,0XAA55AA55;   TWO VALUES IN AN ARRAY

    END                  ; Mark end of file
```

4. ALP to find determine whether the given 16 bit is even or odd

```
AREA  ODD_EVEN , CODE, READONLY
    EXPORT __main
__main
ENTRY                                   ;Mark first instruction to execute

START
        LDR R1,=VALUE           ; LOAD R1 WITH ADDRESS OF VALUE
        MOV R3,#0               ; move 0 to r3
        MOV R2,#0               ; move 0 to r2
        MOV R4,#0               ; move 0 to R4
        MOV R5,#4               ; move 4 to R5 (TO COUNT NUMBER OF VALUES)

LOOP0   LDR R2,[R1],#2          ; move the value of MEM LOCN POINTED BY r1 to r2
      MOVS R2,R2,ROR #1     ; RIGHT SHIFT TO CHECK CARRY BIT (1's/0's)
        ADDCS R3,R3,#1          ; IF CARRY BIT IS 1 add r3 with 1 (TO COUNT ODD
NUMBERS)
        ADDCC R4,R4,#1          ; IF CARRY BIT IS 0 add r4 with 1 (TO COUNT EVEN
NUMBERS)
        ADD R1,R1,#2
        SUBS R5,R5,#1
        BNE LOOP0

        MOV  R6,R3
        MOV R7,R4
        NOP
        NOP

VALUE   DCD   0x1234,0x8D73,0x1111,0xAAAA;


    END
```

5. ALP to write data to RAM

```
AREA  ODD_EVEN , CODE, READONLY
```

```
    EXPORT __main
__main
ENTRY                                   ;Mark first instruction to execute

START
     LDR R1,VALUE     ;load the adress to R1

          MOV R3,#00              ;intitialise R3 and R4 by moving 00
          MOV R4,#00
          MOV R5,#4               ;INITIALISE COUNTER R5 BY MOVING 4
      LDR R2,=ARRAY          ;load address of array to R2

LOOP1     LDR R3,[R2]             ;load the value from mem locn pointed by R2 to R3
          STR R3,[R1]             ;store the R3 value to mem locn pointed by R1

          ADD R2,R2,#4            ;increament R2 + 4 times
          ADD R1,R1,#5            ;increament R1 + 5 times
          SUBS R5,R5,#1
          BNE LOOP1

ARRAY  DCD 0x87A23579,0x8235CC80,0x35254565,0x72889544;
VALUE  DCD 0x10000000;              ADDRESS OF RAM i.e 0X10000000 ASSIGNED TO
VARIABLE 'VALUE'

  END
```