|  |  |
| --- | --- |
| **Batch No.** | **Seat No.** |
|  |  |

#### Rashtreeya Sikshana Samithi Trust

**R. V. COLLEGE OF ENGINEERING**

**[Autonomous Institution Affiliated to VTU, Belagavi]**

**Department of Computer Science & Engineering**

**Bengaluru-560059**



**Microcontroller & Embedded Systems Lab**

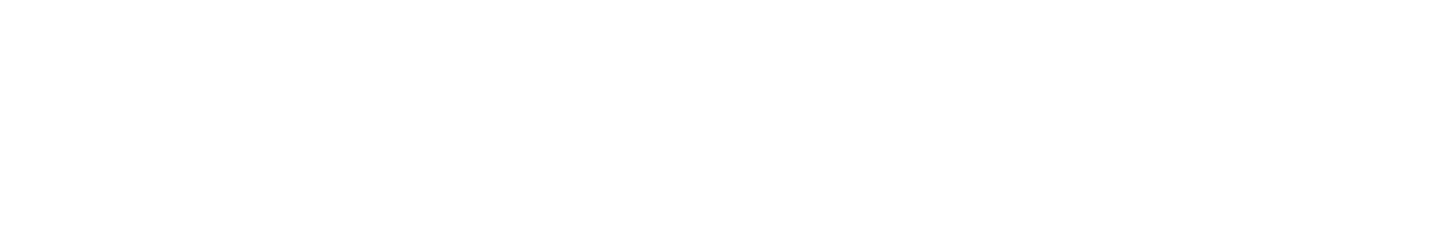
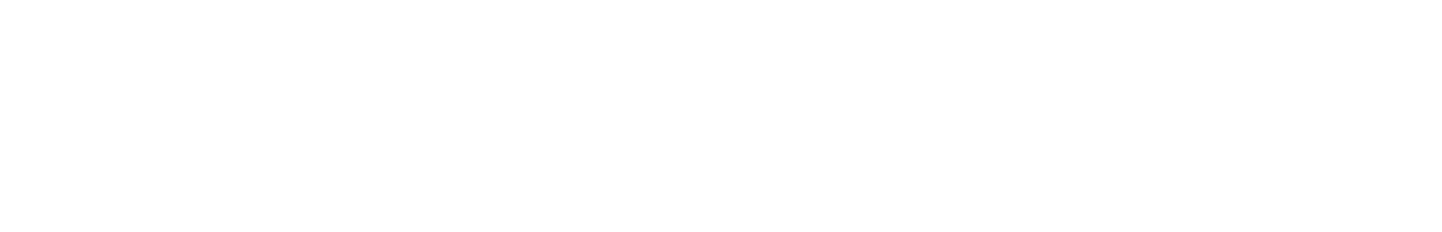
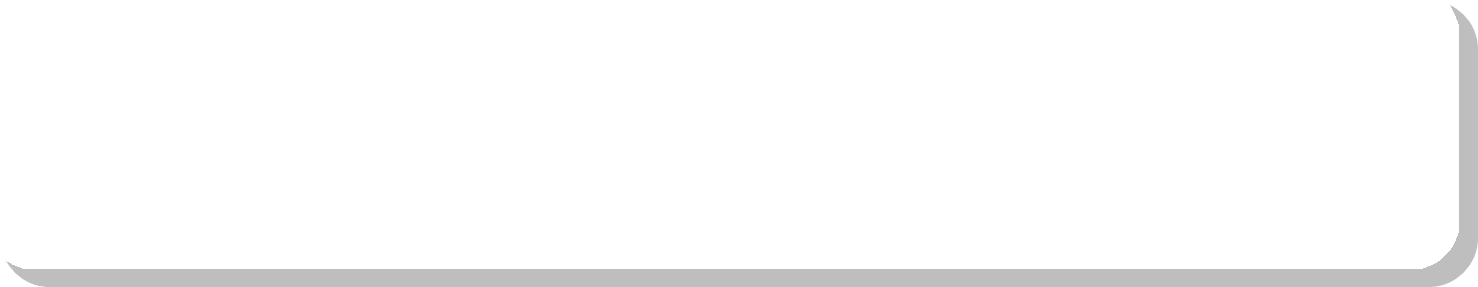
**Subject Code: 18CS44**

**IV SEMESTER B.E.**

**LABORATORY RECORD**

**[Autonomous Scheme 2018]**

**2019-2020**



**Name of the Student: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ USN: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_Section: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Year: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

#### Rashtreeya Sikshana Samithi Trust

**R. V. COLLEGE OF ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



**MICROCONTROLLER & EMBEDDED SYSTEMS LAB**

**(18CS44)**

**2019 - 2020**

**IV SEMESTER B.E LABORATORY RECORD**

**[Autonomous Scheme 2018]**

**R.V. College of Engineering,Bangalore - 59**

***(Autonomous Institution affiliated to VTU, Belagavi)***

## Department of Computer Science & Engineering



LABORATORY CERTIFICATE

This is to certify that Mr. / Ms. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ with USN \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of IV semester has satisfactorily completed the course of experiments in Microcontroller and Embedded Systems Lab [18CS44] prescribed by the department during the year 2019-2020.

|  |  |
| --- | --- |
| **Marks** | |
| **Maximum** | **Obtained** |
| **50** |  |

Signature of the staff in-charge Head of the department

Date:

**R. V. COLLEGE OF ENGINEERING**

**[Autonomous Institution Affiliated to VTU, Belagavi] Department of Computer Science & Engineering Bengaluru-560059**



## VISION

To achieve leadership in the field of Computer Science & Engineering by strengthening fundamentals and facilitating interdisciplinary sustainable research to meet the ever growing needs of the society.

**MISSION**

* To evolve continually as a Centre of excellence in quality education in computers and allied fields.
* To develop state-of-the-art infrastructure and create environment capable for interdisciplinary research and skill enhancement.
* To collaborate with industries and institutions at national and international levels to enhance research in emerging areas.
* To develop professionals having social concern to become leaders in top-notch industries and/or become entrepreneurs with good ethics.

**Program Educational Objectives**

**PEO1:** Develop Graduates capable of applying the principles of mathematics, science, core engineering and Computer Science to solve real-world problems in interdisciplinary domains.

**PEO2:** To develop the ability among graduates to analyze and understand current pedagogical techniques, industry accepted computing practices and state-of-art technology.

**PEO3:** To develop graduates who will exhibit cultural awareness, teamwork with professional ethics, effective communication skills and appropriately apply knowledge of societal impacts of computing technology.

**PEO4:** To prepare graduates with a capability to successfully get employed in the right role / become entrepreneurs to achieve higher career goals or take up higher education in pursuit of lifelong learning.

## Program Outcomes

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

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

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

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

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

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

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

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

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

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

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

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

## Program Specific Outcomes

#### PSO1: System Analysis and Design

The student will:

1. Recognize and understand the dynamic nature of developments in computer architecture, data organization and analytical methods.
2. Learn the applicability of various systems software elements for solving real-world design problems.
3. Identify the various analysis & design methodologies for facilitating development of high quality system software products with focus on performance optimization.
4. Display good team participation, communication, project management and document skills.

#### PSO2: Product Development

The student will:

1. Demonstrate knowledge of the ability to write programs and integrate them resulting in state-of –art hardware/software products in the domains of embedded systems, databases /data analytics, network/web systems and mobile products.
2. Participate in teams for planning and implementing solutions to cater to business – specific requirements displaying good team dynamics and professional ethics.
3. Employee state-of-art methodologies for product development and testing / validation with focus on optimization and quality related aspects

## Course Outcomes

After completing the course, the student will be able to:

**CO1:** Acquire the knowledge of architecture of Microprocessors and Microcontrollers for the different applications.

**CO2:** Develop skill in simple program writing for micro controllers for applications in assembly level language and Embedded C.

**CO3:** Design system configuration for a given application.

**CO4:** Integrate, implement and test the design in applications.

## Do’s and Don’ts in the Laboratory

## Do’s……………...

* Come prepared to the lab with the program logic.
* Use the computers and controller kit for academic purposes only.
* Following the lab exercise cycles as per the instructions given by the department.
* Keep the chairs back to their position before you leave.
* Handle the computer and the kits with care.
* Keep your lab clean.

**Don’ts……………...**

* Coming late to the lab and leaving the lab early.
* Move around in the lab during the lab session.
* Download or install any software onto the computers.
* Tamper system files or try to access the server.
* Write record in lab.
* Change the system assigned to you without the notice of lab staff.
* Carrying CD’s, Floppy’s, Pen Drives and other storage devices into lab.
* Using others login id’s.

**PARTICULARS OF THE EXPERIMENT**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Prog**  **No.** | **Program** | **Page No** | **Marks Split as per rubrics** | | | | | **Marks (10)** |
| **Execution** | | | **Viva** | |
| **2** | **2** | **2** | **2** | **2** |
| 1a | [i]Write Assembly programs, to translate the given code in C to the ARM instruction set. Assume variables are 32bit integers represented in Registers.  [ii]Write an ARM ALP to perform addition and subtraction of two 32bit and 64bit numbers. | 5 |  |  |  |  |  |  |
| 2a | Write an ARM ALP to find smallest and largest of N- 32 bit numbers. | 10 |  |  |  |  |  |  |
| 3a | Write an ARM ALP to compute Average of N-32 bit numbers | 12 |  |  |  |  |  |  |
| 4a | Write an ARM ALP to count the occurrences of given 32-bit number in a list using Linear Search algorithm | 15 |  |  |  |  |  |  |
| 5a | [i]Write an ARM ALP to compute number of 1’s in a given 32-bit number and check the parity of the given number.  [ii] Write an ARM ALP to compute GCD of two given 32-bit numbers. | 17 |  |  |  |  |  |  |
| 6a | Write an ARM ALP to compute the factorial of a given 32-bit number using procedure. | 21 |  |  |  |  |  |  |
| 7a | Write an ARM ALP to sort the given list of 32-bit numbers using Bubble Sort. | 24 |  |  |  |  |  |  |
| 1b | Elevator Interface: Write an Embedded C program to read the elevator switches and simulate elevator up and down movements. | 31 |  |  |  |  |  |  |
| 2b | Seven Segment Display Interface: Write a C program to display messages “FIRE” & “HELP” on 4 digit seven segment display alternately with a suitable delay. | 34 |  |  |  |  |  |  |
| 3b | Stepper Motor Interface: Write an Embedded C program to rotate stepper motor in clockwise direction for “M” steps, anti-clock wise direction for “N” steps. | 39 |  |  |  |  |  |  |
| 4b | DAC Interface : Write an Embedded C program to generate sine , full rectified ,triangular, sawtooth and square waveforms using DAC module | 42 |  |  |  |  |  |  |
| 5b | Matrix Keyboard Interface : Write an mbedded C program to interface 4 X 4 matrix keyboard using lookup table and display the key pressed on the Terminal. | 49 |  |  |  |  |  |  |
| 6b | DC Motor Interface: Write an Embedded C program to generate PWM wave to control speed of DC motor. Control the duty cycle by analog input. | 53 |  |  |  |  |  |  |
| 7b | Character LCD Interface : Write an Embedded C program to display text messages on the multiple lines of the display. | 57 |  |  |  |  |  |  |
| **Total for 140 Marks** | | | | | | | |  |

|  |  |
| --- | --- |
| **LAB INTERNALS** | |
| **RECORD Marks** | / 30 Marks |
| **Mini Project** | / 10 Marks |
| **TEST** | / 10 Marks |
| **TOTAL** | / 50 Marks |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Lab Write-up and Execution Rubrics (Max: 6 marks)** | | | | | |
| **Sl no** | **Criteria** | **Excellent** | **Good** | **Poor** | **Score** |
| 1 | **Understanding of problem and requirements (2 Marks)**  **CO1** | Student exhibits thorough understanding of program requirements and applies ALP / Embedded C for 8051 concepts.  **(2M)** | Student has sufficient understanding of program requirements and applies ALP / Embedded C for 8051 concepts. **(1.5M - 1M)** | Student does not have clear understanding of program requirements and is unable to apply ALP / Embedded C for 8051 concepts.  **(0M)** |  |
| 2 | **Design & Execution (2Marks)**  **CO 2, 3** | Student demonstrates the design & execution of the program with optimized code with all the modifications and test cases handled.  **(2M)** | Student demonstrates the design & execution of the program without optimization of the code and handles only few modifications and few test cases.  **(1.5M - 1M)** | Student has not executed the program.  **(0M)** |  |
| 3 | **Results and Documentation (2Marks)**  **CO 1, 4** | Documentation with appropriate comments and output with observations is covered in manual.  **(2M)** | Documentation with only few comments and only few output cases is covered in manual. **(1.5M - 1M)** | Documentation with no comments and no output cases covered in manual. **(0M)** |  |
| **Viva Voce Rubrics (Max: 4 marks)** | | | | | |
| 1 | **Conceptual Understanding (2 Marks)**  **CO 1** | Explains related architecture & Assembly language programming / Embedded C related concepts involved. **(2M)** | Adequately explains architecture & Assembly language programming / Embedded C related concepts involved. **(1.5-1M)** | Unable to explain the concepts. **(0M)** |  |
| 2 | **Use of appropriate Design Techniques (2 Marks)**  **CO 2, 3** | Insightful explanation of appropriate design techniques for the given problem to derive solution.  **(2M)** | Sufficiently explains the use of appropriate design techniques for the given problem to derive solution. **(1.5M-0.5M)** | Unable to explain the design techniques for the given problem. **(0M)** |  |
| **Total Marks** | | | | |  |
| **Staff Signature:** | |  | | | |

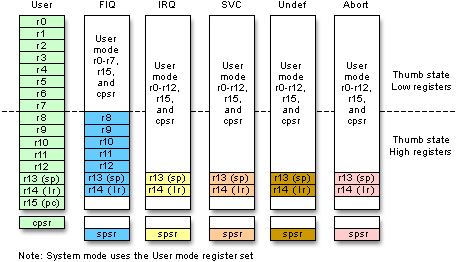
**ARM Assembly Language Programs – Part A**

**ARM Architecture ( Instruction Set Architecture – ISA)**

ARM – stands for “**A**dvanced **R**ISC **M**achine” , ARM based microcontrollers are very popular in 32 bit embedded market, occupying the major share of the market. ARM7 was the first commercial success, used extensively in products like PDAs, IPods, hard disks, set-top boxex, mobile phones etc.

**Operating Modes** : ARM has seven operating modes, i.e it exists in any one of these modes when the processor is running,

1. User mode: simplest mode with least privileges (or also referred as Unprivileged mode), this is the mode under which most applications run, User mode is used for the execution of programs and applications.
2. FIQ Mode (Fast interrupt request): Entered this mode on request from FIQ interrupts
3. IRQ Mode (Interrupt Request): Entered this mode on IRQ request
4. Supervisor: Entered on Reset and when a software interrupt instruction (SWI) is executed, and is generally the mode in which the operating system kernel operates in.
5. Abort: Used to handle memory access violations, Abort is entered after a failed memory access
6. Undef: Used to handle undefined instructions, Undefined is entered if the instruction is not defined (invalid opcodes)
7. System: This is highly privileged mode used by operating systems to manipulate and control the activities of the processor, System mode is a special version of user mode that allows full read write access to the CPSR



**Register Set**: ARM has 37 registers of size 32bits each, they are

1. PC - 1 dedicated program counter
2. CPSR – 1 dedicated program status register, (like flag register of 8086)
3. 5 dedicated saved program status registers (SPSR)
4. 30 general purpose registers

**Overview of ARM Assembly Language Instructions.**

**Data/Register Transfer Instructions**

**Format: MOV REG, REG / IMM**

**MVN REG, REG / IMM**

**Example: MOV R1,R2 ; Move contents of R2 to R1**

**MOV R1,#3 ; Move the immediate 3 to R1**

Shift and Rotate operations can be part of other Register Transfer / Arithmetic / Logic operations. One of the operand can be operated by shift/rotate operations using barrel shifter.

**Example: MOV R1, R0 , LSL #1**

; rotate R0, by operation Logical Left Shift & Put the value to R1, the second operand [R0 with LSL #1] is also called as shifter operand. Register can also be used to indicate the number of bits to be shifted, Ex- MOV R1, R0, LSL R2

Similar logical/rotate options :

LSR #n - Logical Shift Right

LSL #n - Logical Shift Left

ASR #n – Arithmetic Shift Right

ROR #n - Rotate Right

RRX #n - Rotate Right Extended ( i.e with Carry)

(Rotate left ‘n’ bits is equivalent to rotate right by (32-n) bits)

**Updating of Status Flags**

Updation of status flags is possible by appending “S” to the instruction,

MOV**S** R0, #0

**Conditional Execution** : **Suffixing** condition codes is possible for any data processing and branch instructions, if condition code satisfies instruction works, else it is a NOP instruction. There are 15 such condition codes.

Ex: MOV**EQ** R0, #10 ; 10 is moved to R0, if Zero flag is set else it is a NOP

Other Commonly used Condition codes :

EQ Z=1 zero flag set

NE Z=0 zero flag clear

CS C=1 carry flag set

CC C=0 carry flag clear

MI N=1 Sign/Negative flag set (number is MINUS)

PL N=0 Sign/Negative flag is clear(number is POSITIVE)

**ARITHMETIC INSTRUCTIONS**

**Format:**

ADD REG, REG, (REG/IMM)

SUB REG, REG, (REG/IMM)

RSB REG, REG, (REG/IMM)

Ex : ADDS R2,R3,R4 ; R2 🡨 R3 + R4 and update the flags (because of suffix S)

**LOGICAL INSTRUCTIONS**

**Format:**

AND REG, REG, (REG/IMM)

EOR REG, REG, (REG/IMM)

ORR REG, REG, (REG/IMM)

BIC REG, REG, (REG/IMM)

Ex: **ANDS R5, R0 , R1** ; R5 🡨 R0 .AND. R1 (bit AND operation), S – updatde flags

Ex: **ORRS R5, R0 , R1** ; R5 🡨 R0 .OR. R1 (bit AND operation), S – updatde flags

Ex: **EORS R5, R0 , R1** ; R5 🡨 R0 .OR. R1 (bit AND operation), S – updatde flags

* BIC is used to clear selected bits of the Register

Ex: **BIC R5, R0 , R1** ; R5 🡨 R0 .AND. ~R1 (i.e R0. AND. NOT R1)

**COMPARE instructions**

The CPSR register contains four flags Negative(sign flag),Zero,Carry and Overflow flags, which are affected by the execution of following instructions. Flags are also affected by using suffix S to the other data processing instructions.

**Format:**

CMP REG, (REG/IMM)

TST REG, (REG/IMM)

TEQ REG, (REG/IMM)

* CMP R1 , R2 ; pseudo subtraction and updates the flags
* TEQ R1 , R2 ; R1. XOR . R2 pseudo XOR operation and updates the flags
* TST R1 , R2 ; R1. AND . R2 pseudo AND operation and updates the flags

**MULTIPLICATION**

**Format:**

MUL REG , REG, (REG/IMM)

MLA REG , REG, REG, REG

Examples:

**MUL R1 , R2 , R3 - Multiply R1 🡨 R2 x R3**

**MLA R4 , R3 , R2 , R1 - Multiply and Accumulate; R4 🡨(R3 x R2) + R1**

**BRANCH INSTRUCTIONS**

1. **B LOOP ; branch to the address with label LOOP**

**BEQ LOOP ; branch only if Zero Flag is set**

**BNE LOOP ; branch only if Zero Flag is not set (i.e clear)**

**(**used when executing conditional/unconditional branches**)**

1. **BL NEXT ; branch with LINK, copy the PC(address on next instruction i.e PC+4) contents to LR, then branch**

**(**used when calling procedures)

Format of Procedure Calls:

Ex: BL PROC1 ; contents of PC is Copied to LR

**Load & Store Instructions**

Format:

LDR REG, [REG]

LDR REG, [REG,IMM]

LDR REG, [REG,REG]

LDR REG, [REG,REG,SHIFT IMM]

STR REG, [REG]

STR REG, [REG]

STR REG, [REG,IMM]

STR REG, [REG,REG]

STR REG, [REG,REG,SHIFT IMM]

Different addressing modes supported in LOAD/STORE instructions with examples.

1. Register Indirect LDR R1,[R0]
2. Base Offset LDR R1,[R0,#4]
3. Base Index LDR R1,[R0,R2]
4. Based Scaled Index LDR R1,[R0,R2,LSL #2]

**Ex:**

**LDR R1, [R0] ;**contents of memory(32 bit number – 4 bytes) pointed by R0 is loaded into R1

**STR R1, [R0]** ; contents of R1(32bit number-4 bytes) is stored in memory pointed by R0.

##### Experiment No 1A (i):

**AIM: Translate the following code in C to the ARM instruction set. Assume variables are 32bit integers represented in Registers.**

**A = B + C - D**

AREA RESET, CODE

ENTRY

MOV R0,#00 ; A

MOV R1,#NUM1 ; B

MOV R2,#NUM2 ; C

MOV R3,#NUM3; D

ADD R0,R2,R1

SUB R0,R0,R3

STOP B STOP

END

**A = 2 \* A + B**

AREA RESET, CODE

ENTRY

MOV R0,#NUM1 ; A

MOV R1,#NUM2; B

ADD R0,R1,R0,LSL #2

STOP B STOP

END

**Sum of 3X + 4Y + 9Z, where X = 2, Y=3 and Z=4.**

AREA RESET, CODE

ENTRY

MOV R1, #2 ; Let X = 2

MOV R2, #3 ; Let Y = 3

MOV R3, #4 ; Let Z = 4

ADD R1, R1, R1, LSL #1

MOV R2, R2, LSL #2

ADD R3, R3, R3, LSL #3

ADD R1, R1, R2

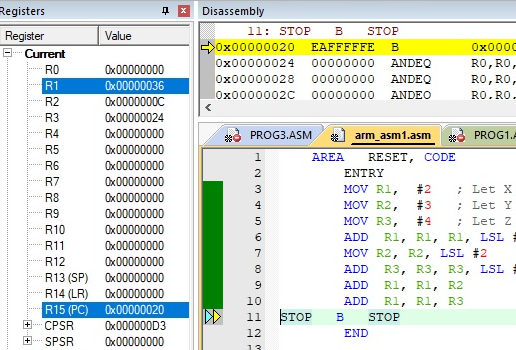
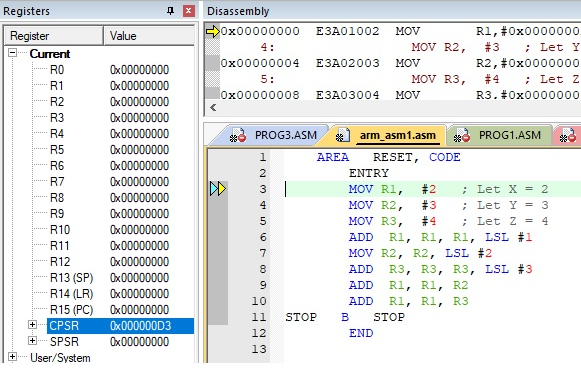
ADD R1, R1, R3

STOP B STOP

END

**Sample Output:**

Before Execution After Execution

****

**Output & Observations:**

**Assignment:**

Write ARM Assembly Language Programs computing the following expressions or program segments.

1. Variables are of type unsigned integers and word sized. Also the variables b, c, d and e are initialized to values 10, 20, 30 and 40 respectively.

a = (b + c) - (d + e)

1. Variables are of type signed integers and half-word sized.Variables b, c, d and e are initialized to values -10, 20, 30 and -40 respectively.

a = (b + c) - (d + e)

1. Variables are of type unsigned integers and word sized. Also the variables b, c, d and e are initialized to values 10, 20, 30 and 40 respectively.

b = a + 256 .   
c = b - 0xfffff  
a = (b + c) - (d + e + 0xffffe)

Write the value in R0 (32 bit number) after the execution of following instructions:

(Note: Write the answer in Hex with caps; Example: **FFFFFF82**)

MOV R0,#1

MVN R0,R0,LSL #3

Write the value in R0 (32 bit number) after the execution of following instructions:

(Note: Write the answer in Hex with caps; Example: **FFFFFF82**)

MOV R0,#1

MVN R0,R0,LSL #1

Write the value in R0 after executing the following instructions:

MOVS R0,#5

MOVEQ R0,#10

Write the value in R0 after executing the following instructions:

MOV R0,#5

MOVEQ R0,#10

Write the value in R0 after executing the following instructions:

(Note: Write the last bits of answer in binary; Example: 11010011)

MOV R0,#00100101B ; number represented in binary form

MOV R1,#00100011B ; number represented in binary form

BIC R0, R0, R1

Write the value in R0 after executing the following instructions:

(Note: Write the last 8 bits of answer in binary; Example: 11010011)

MOV R0,#00100101B ; number represented in binary form

MOV R1,#00100010B ; number represented in binary form

BIC R0, R0, R1

Write the value in R0 (32 bit number) after the execution of following instructions.

(Write Ans: Hex with caps,use zero key for typing ‘0’ Ex: **F0FFFF82**)

MOV R0,#0X06

MOV R0,R0,ROR #31

MOV R1,#0x02

ORR R0, R0, R1

Write the value in R0 after executing the following instructions:

(Write Ans: Hex with caps,use zero key for typing ‘0’ Ex: **F0FFFF82**)

MOV R0,#1

MOV R0,R0,LSL #31

MOV R1,R0,ASR #2

Write the value in R0 after executing the following instructions:

(Write Ans: Hex with caps,use zero key for typing ‘0’ Ex: **F0FFFF82**)

MOV R0,#0X0FF

MOV R1,#0X010

CMP R0,R1

MOVCS R2,R0

MOVCC R2,R1

Write the value in R3 after executing the following instructions:

MOVS R1,#1

MOVS R0,#0

MOVEQS R0,#5

MOVEQ R1,#5

MOV R4,R1,LSL R0

MLA R3,R0,R1,R4

##### Experiment No 1A(ii):

**AIM: Write an ARM ALP to perform addition and subtraction of two 32-bit and 64bit numbers.**

MSB:LSB

Value1 = R1 : R2

Value2 = R3 : R4

Result = R5 : R6

**Program:**

**AREA RESET, CODE**

**ENTRY**

LDR R0,=VALUE1

LDR R1,[R0]

LDR R2,[R0,#4]

LDR R0,=VALUE2

LDR R3,[R0]

LDR R4,[R0,#4]

ADDS R6,R2,R4

ADC R5,R1,R3

LDR R0,=RESULT

STR R5,[R0]

STR R6,[R0,#4]

STOP S STOP

VALUE1 **DCD** &BBBBBBBB,&AAAAAAAA

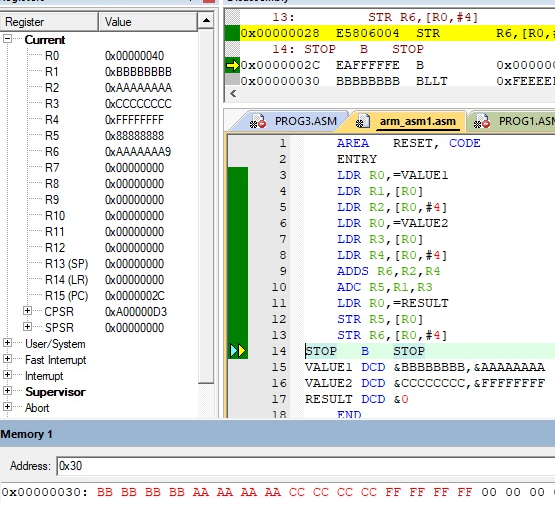
VALUE2 **DCD**  &CCCCCCCC,&FFFFFFFF

**AREA MEMORY, DATA**

**RESULT SPACE 4**

**END**

**Sample Output:**



**Output & Observations:**

Assignment / viva questions:

1. Write the output in the register R2, after execution of the following program

**AREA RESET, CODE, READONLY**

**ENTRY**

**LDR R0,=NUM**

**LDR R1,[R0]**

**MVN R2, #0X 0000 0009**

**AND R1, R2**

**STOP B STOP**

**NUM DCD &FFFFFFFF**

**END**

1. Write the output in the register R4, after execution of the following program

**MOV R0,#2**

**MOV R1,#3**

**MOV R2,#4**

**MLA R4,R0,R1,R2**

1. Program to add two 32 bit numbers

AREA RESET,CODE,READONLY

ENTRY

LDR R0,=NUMS

LDR R1,[R0]

LDR R2,[R0,#4]

LDR R0,=RES

ADD R3,R1,R2

STR R3,[R0]

STOP B STOP

NUMS DCD &10203040,&10203040

##### Experiment No 2A:

##### Algorithm:

1) Initialize first element as smallest [R1] and number of elements *n = n-1*

2) Loop through all the *n* [R4] elements. If the current element is smaller than *the smallest*, then update *smallest*.

**AIM: Write an ARM ALP to find smallest and largest of N- 32 bit numbers.**

**AREA RESET,CODE**

**ENTRY**

LDR R0,=DATA1

LDR R3,=0X40000000 ; memory location for storing answer

MOV R4,#05 ; //N- number of elements

LDR R1,[R0],#04; assume first no. as smaller no & increment R0 by 4

SUB R4,R4,#01 ; compare with n-1 elements

BACK

LDR R2,[R0] ; get next number & compare with small

CMP R1,R2

BLS LESS ; // If R1 < R2 , BRANCH

MOV R1,R2 ; update with new smaller no

LESS

ADD R0, R0,#04 ; increment pointer to next number

SUB R4,R4,#01

CMP R4,#00

BNE BACK

STR R1, [R3] ; // SMALLEST VALUE STORED IN MEMORY LOCATION

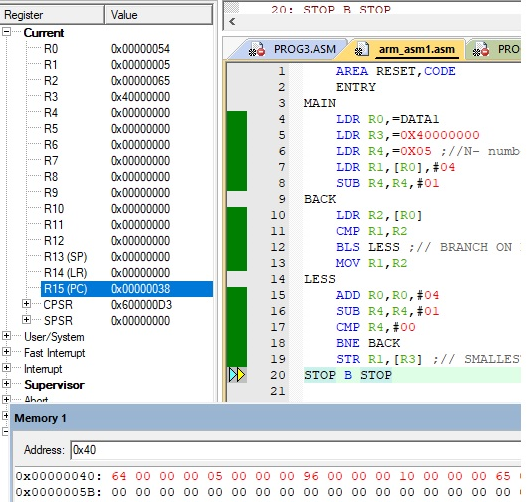
STOP B STOP

**AREA DATA,CODE**

DATA1 DCD &64,&05,&96,&10,&65

**END**

**Sample Output:**



**Output & Observations:**

**Modification/Assignment:**

Design & Develoop an ARM 7 ALP to compute the Maximum and Minimum temperatures, among the set of N 32 bit temperatures stored in the SRAM at the address 0x40000100.

**N EQU 5**

**MIN RN 2**

**MAX RN 3**

**AREA RESET,CODE,READONLY**

**ENTRY**

**LDR R0,=0X40000100 ;memory address of SRAM**

**MOV R1,#N**

**SUB R1,R1,#01**

**LDR MIN,[R0],#04; assume first element,R2-MIN**

**MOV MAX,MIN ; assume first element,R3-MAX**

**BACK**

**LDR R4,[R0],#4;get next element**

**CMP R4,MIN**

**MOVLS MIN,R4**

**CMP R4,MAX**

**MOVHI MAX,R4**

**SUBS R1,#1**

**BNE BACK**

**STOP B STOP**

**END**

NOTE: **feed the numbers in SRAM at run time, In Keil, go to the required memory location, double click on the memory contents, then you can enter the data in hex.**

##### Experiment No 3A:

**AIM: Write an ARM ALP to compute Average of N-32 bit numbers**

##### Algorithm:

1) Initialize *sum = 0*

2) Loop through all the *n* elements. Add the current element to *sum*.

1. Calculate *average = sum divide by n*

**Program:**

**AREA RESET, CODE**

**ENTRY**

MOV R3,#0

MOV R4, #0

LDR R0, =INPUTS

LDR R1, =OUTPUTS

CONT

LDR R2, [R0]

ADD R4, R4, R2

ADD R0, R0, #4

ADD R3, R3, #1

CMP R3, #5

BNE CONT

MOV R2, #5

MOV R3, #0

REPT SUBS R4, R4, R2

ADDPL R3, R3, #1

BPL REPT

ADDMI R4, R4, R2

STR R3, [R1]

**STOP B STOP**

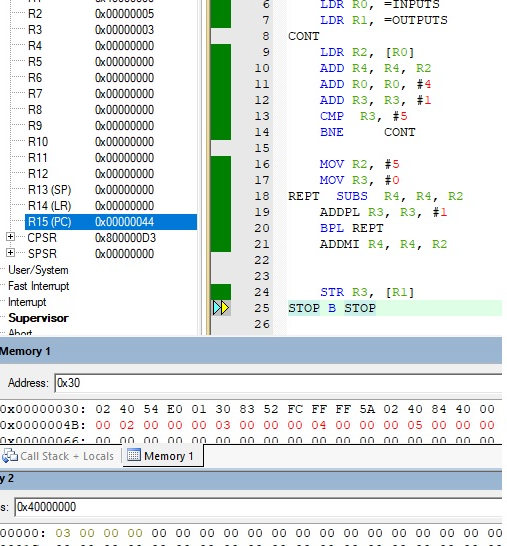
**INPUTS DCD 01,02,03,04,05**

**AREA MEMORY, DATA**

**OUTPUTS SPACE 4**

**END**

**Sample Output:**



**Output & Observations:**

Write ARM Assembly Language Programs equivalent to the following C-code fragments.

1. if ( a < b) {         // a and b are signed integers  
      c = a - b;  
   }
2. if ( a < b) {         // a and b are unsigned integers  
      c = a - b;  
   }
3. if ( a < -1234) {         // a is a signed integer  
      a = 4 \* a ;  
   }  
   else {  
      a = a/4;  
   }
4. if ( a < b ) {         // a, b, c are unsigned integer  
      if ( a < c) {   
         min = a;  
      }  
      else {  
   min = b < c ? b : c;  
   }  
   }  
   else {  
      min = b < c ? b : c  
   }

##### Experiment No 4A:

**AIM: Write an ARM ALP to count the occurrences of the given 32-bit number in a List using Linear Search algorithm**

**Algorithm:**

Linear Search ( array A, key x)

{

for i =0 to n

if A[i] = x then

increment element found count

**}**

**Program:**

**AREA RESET, CODE, READWRITE**

**ENTRY**

LDR R0,=ARR

MOV R1, #0 ; Loop Iterator

MOV R7, #0 ; Number Of occurrences in The Array

MOV R4, #4 ; key

CONT

LDR R3,[R0]

CMP R3, R4

BNE SKIP

ADD R7, R7,#1

SKIP

ADD R0, #4

ADD R1, #1

CMP R1, #10 ; no of elements

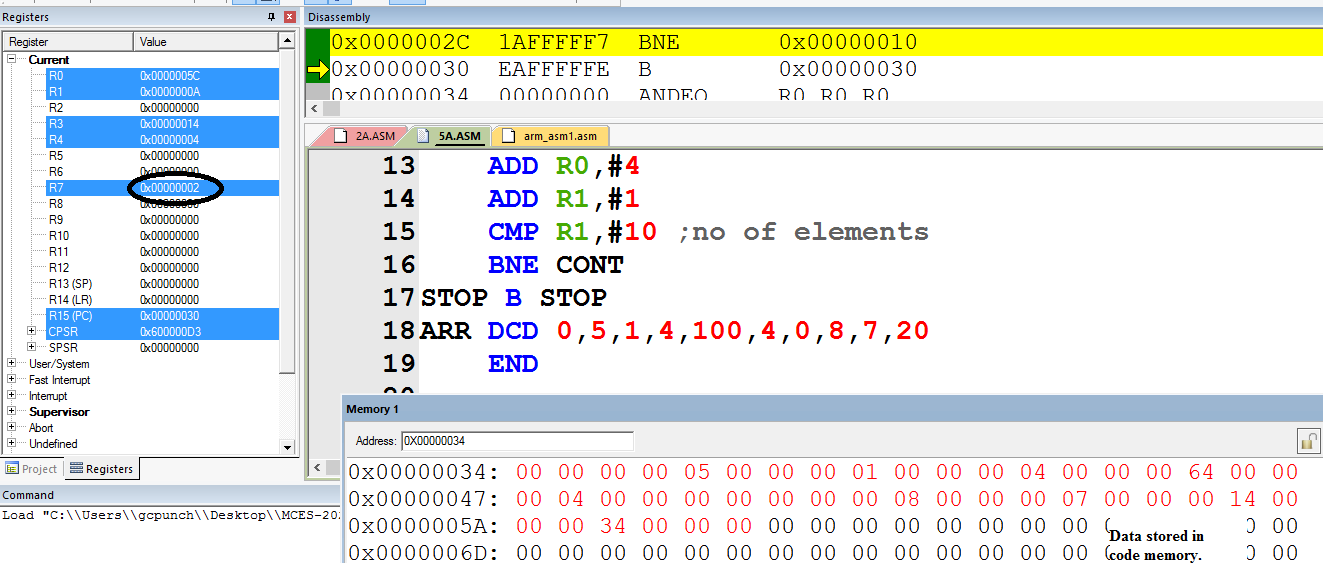
BNE CONT

**STOP B STOP**

ARR DCD 0,5,1,4,100,4,0,8,7,20

**END**

**Sample Output:**



**Output & Observations:**

Assignment /viva questions:

Write ARM Assembly Language Programs equivalent to the following C-code fragments..

1. sqrsum = 0;   
   for( i = 0 ; i < 100; ++i)   
       sqrsum = sqrsum + a[i] \* a[i];
2. convsum = 0;  
   for ( i = 0, j = 99; i <100; ++i, --j )  
         convsum += a[i] \* b[j] ;
3. sum = 0;   
   for( i = 0 ; i < 100; ++i)  
       for( j = 0; j < 100; ++j)  
            sum = sum + a[i, j] ;

##### Experiment No 5A(i):

**AIM: Write an ARM ALP to compute number of 1’s in a given 32 bit number and check the parity of the given number**

**Program:**

AREA RESET,CODE

ENTRY

LDR R0,=DATA1

LDR R1,[R0] ;Stores 32bit number-16 ones

MOV R2,#0 ;loop counter, 32 times(32bits)

MOV R4,#0 ;number of 1's counter

MOV R3,#1

LOOP

MOVS R1,R1,LSR #1

ADDCS R4,R4,#1

ADD R2,#1

CMP R2,#32 ; check for 32 bits

BNE LOOP

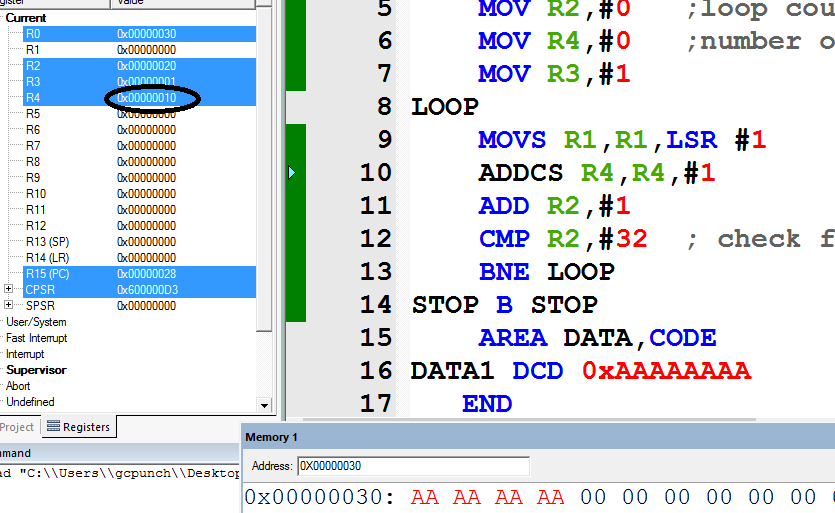
STOP B STOP

AREA DATA,CODE

DATA1 DCD 0xAAAAAAAA

END

**Sample Output:**



**Output & Observations:**

1. Modify the code, to check the parity of the number and store appropriately in any register.
2. Write an ALP to compute the reverse of a 32 bit number, and check the given number is a palindrome or not.

##### Experiment No 5A(ii):

**AIM: Write an ARM ALP to compute GCD of two given 32-bit numbers.**

**Algorithm:**

int gcd(int x,int y){

while(x!=y)

{

if(x>y)

return gcd(x-y,y);

else

return gcd(x,y-x);

}

return x;

}

**Program:**

AREA RESET,CODE

ENTRY

MOV R0,#30 ; test values

MOV R1,#45 ; test values

LOOP CMP R0,R1

BEQ EXIT

BGT COND1

SUB R1,R0

B LOOP

COND1 SUB R0,R1

B LOOP

EXIT LDR R0,=GCD

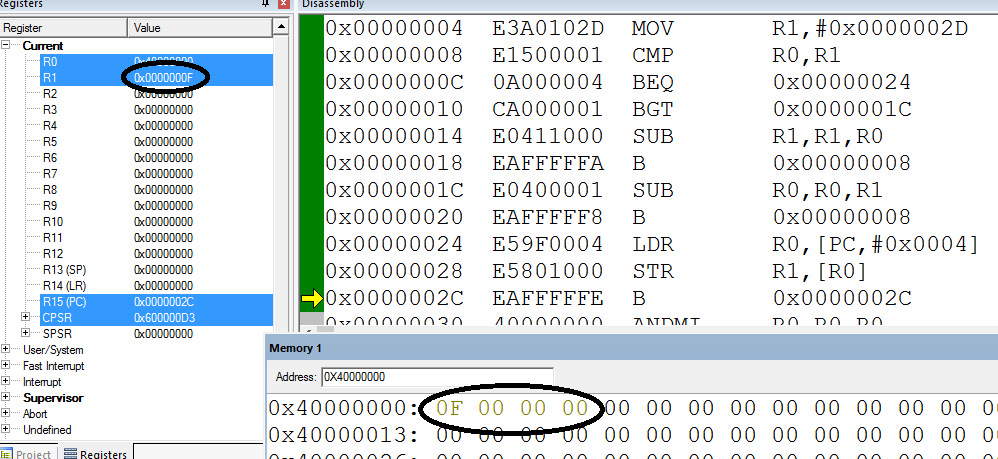
STR R1,[R0]

STOP B STOP

AREA RESULT,DATA

GCD SPACE 4

END

**Sample Output:**

**Output & Observations:**

##### Experiment No 6A:

**AIM: Write an ARM ALP to compute the factorial of a given 32-bit number using procedure.**

**Algorithm:**

fact(int n)

{

Read number n.  
Initialize i =n and fact to 1.  
Repeat while i is not equal to 0.  
{

fact = fact \* i  
i = i -1

}

Return fact

}

**Program:**

AREA RESET,CODE

ENTRY

LDR R0,=INPUT

BL FACT ; CALL SUBROUTINE FACT

LDR R1,=0X40000000 ; RAM area

STR R3,[R1] ; store result in R3 to RAM

STOP B STOP

;subroutine-begin

FACT

LDR R2,[R0] ; get num in R2

CMP R2,#00

BEQ END1

MOV R3,R2 ; result = num

LOOP

SUB R2,#01 ; num = num - 1

CMP R2,#00

MULNE R3,R2,R3 ;result = result \* num

BNE LOOP

MOV PC,LR

END1

MOV R3,#01 ; return R3=1, if num=0

END2

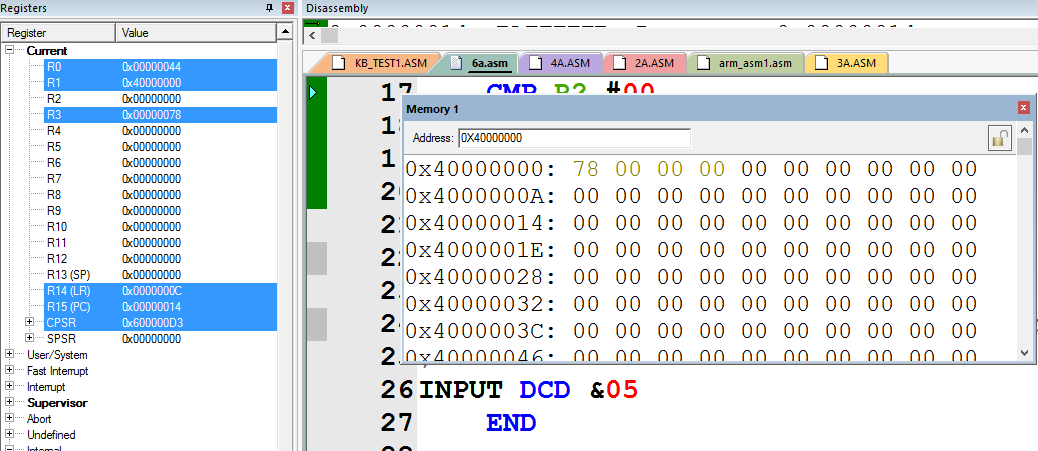
MOV PC,LR ; return from subroutine

;subroutine-end

INPUT DCD &05

END

**Sample Output:**



**Output & Observations:**

**Assignment / viva questions:**

Write an ARM 7 ALP Procedure to convert the following C code. Assume R0 is pointed to array A, and the result is returned through R1. Make suitable assumptions.

int SUM(int A[100],N)

{

int i;

while(i != N) {

sum = sum + A[ i] ;

i++;

}

}

AREA RESET,CODE,READONLY

ENTRY

LDR R0,=NUMS

MOV R4,#5

BL SUM

STOP B STOP

; PROCEDURE TO COMPUTE SUM

; BEGIN

SUM

MOV R1,#0

MOV R2,#0 ; INDEX i

CONT

LDR R3,[R0,R2,LSL #2]

ADD R1,R1,R3

ADD R2,R2,#1

CMP R2,R4

BNE CONT

MOV PC,LR ;BX LR

;END OF PROCEDURE

NUMS DCD 1,2,3,4,5

END

Implement the following C-procedures in ARM. You can assume the parameters are passed either in registers or through the stack.

1. swap(int \*a, int \*b)  
   {  
     int temp;  
     temp = \*a;  
     \*a = \*b;  
     \*b = temp;  
   }
2. Write a sorting routine with signature Sort(int \*data, int dim) where dim gives the dimension of the array 'data'. Use the swap routine you have written earlier in the Sort routine.

##### Experiment No 7A:

**AIM: Write an ARM ALP to sort the given list of 32-bit numbers using Bubble sort.**

##### Algorithm:

##### bubbleSort(list of n elements)

##### {

##### for all n elements

##### n = n-1;

##### for (i=0 to n, i++)

##### if ( A[i] > A[i+1] )

##### temp = A[i]

##### A[i] = A[i+1]

##### A[i+1] = temp

##### }

**Program:**

AREA RESET, CODE

ENTRY

LDR R0, =ARRAY

LDMIA R0, {R1-R10} ;***COPY FROM FLASH TO SRAM,USING REGISTERS***

MOV SP, #0X40000000

STMIA SP, {R1-R10}

MOV R10, #0X40000000 ***; STARTING MEMORY ADDRESS OF ARRAY***

MOV R1, #10 *; TOTAL ELEMENTS*

SUB R1,#1 ***; TOTAL ITERATIONS = N-1***

LOOP1

MOV R2, R1 ***; NO OF COMPARISIONS IN A GIVEN PASS = NO OF PASSES LEFT***

MOV R4, R10 ***; SET R4 = STARTING MEM ADDRESS,FOR EVERY PASS***

LOOP2

LDR R0, [R4] ***; GET FIRST ELEMENT, ARR[i]***

LDR R5, [R4, #4] **;GET NEXT ELEMENT,ARR[i+1]**

CMP R0, R5

BLS SKIP ***;IF ARR[i]<ARR[i+1], skip***

MOV R6, R0 ***;else, SWAP ARR[i] & ARR[i+1]***

MOV R0, R5

MOV R5, R6

STR R0, [R4]

STR R5, [R4,#4]

SKIP

ADD R4, #4

SUBS R2, #1 ; GO TO NEXT COMPARISION

BNE LOOP2

SUBS R1, #1 ; GO TO NEXT ITERATION

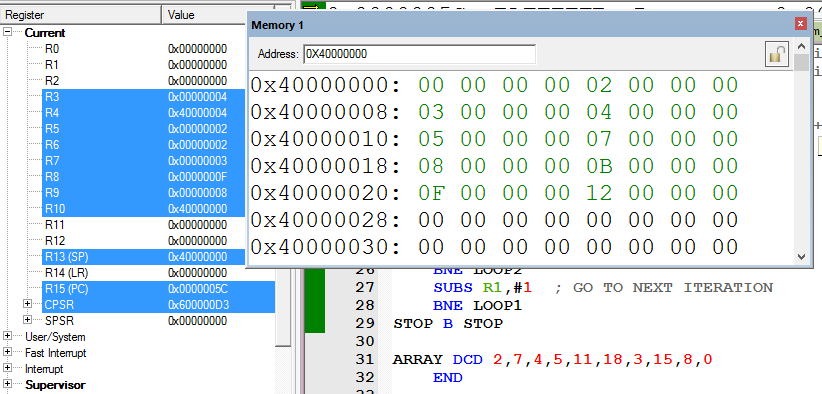
BNE LOOP1

STOP B STOP

ARRAY DCD 2,7,4,5,11,18,3,15,8,0

END

**Sample Output:**



**Output & Observations:**

Modificatin/Assignment:

Set of N 32 bit signed numbers representing temperatures are stored in the code memory at the location TEMP. Correction factor of %2 (divide all the numbers by 2) is to be performed on all the numbers. Develop suitable procedure to perform the same. (Note: the data stored in the code memory can’t be changed.)

Answer:

Assume N=5, and temperatures are stored at TEMP

**AREA RESET,CODE,READONLY**

**ENTRY**

**LDR R0, =TEMP**

**LDMIA R0, {R1-R5}**

**;COPY FROM FLASH TO SRAM,USING REGISTERS**

**MOV SP, #0X40000000**

**STMIA SP, {R1-R5}**

**MOV R0, #0X40000000**

**MOV R1, #5 ; TOTAL ELEMENTS**

**CONT**

**LDR R2,[R0]**

**MOV R2,R2,LSR #1**

**STR R2,[R0]**

**ADD R0,R0,#4**

**SUBS R1,#1**

**BNE CONT**

**STOP B STOP**

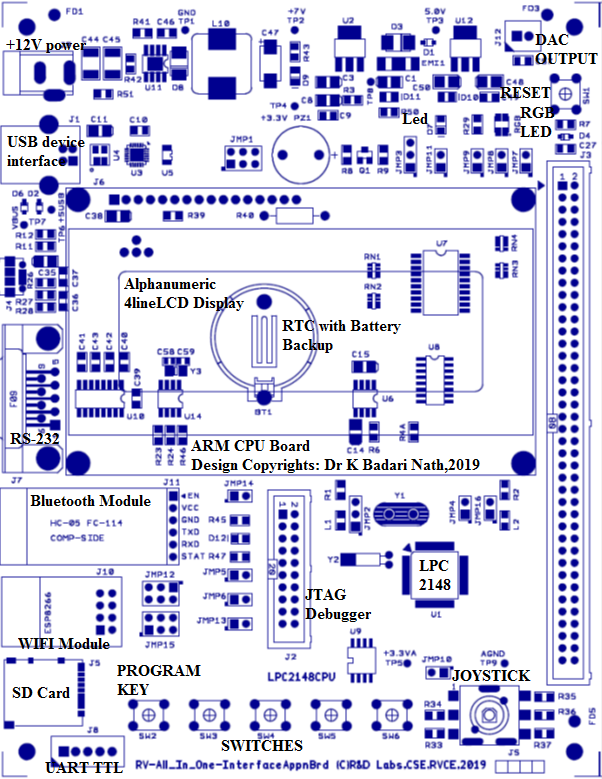
**TEMP DCD 0X30,0X40,0XF0,0X90,0X74**

**END**

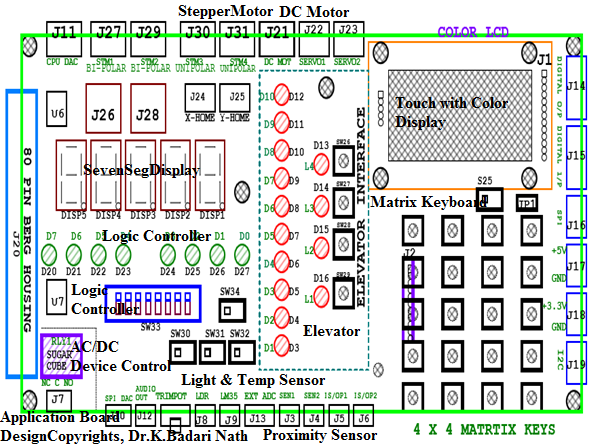
PART – B

Interfacing Programs using ARM LPC 2148

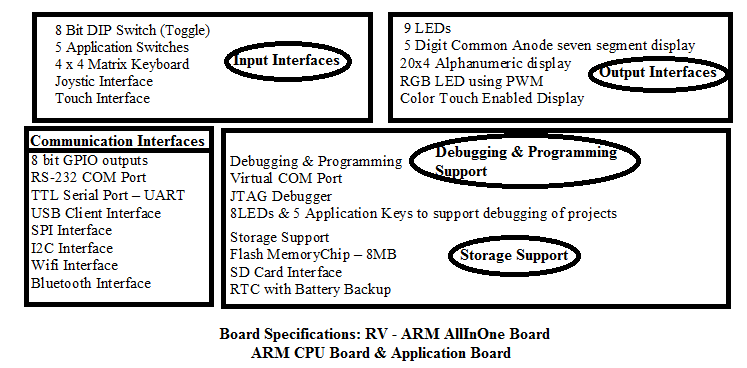
**ARM CPU Board**



**ARM Application Board**

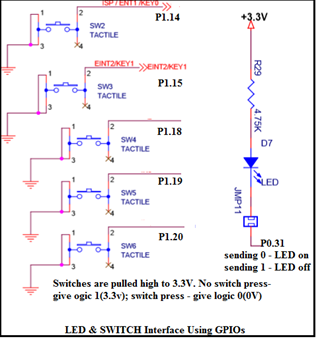


Board Specifications: Power : 12V, 1A



**Sample program : Interfacing LED and Switches**

Interfacing Diagram



*//Sample Program 1: Interfacing LED and Switch to LPC2148 using GPIO pins*

*//P0.31 connected to LED - D7 in CPU board(common anode)*

*//P1.14 connected to Switch - SW2 in CPU board*

#include <lpc214x.h>

#define LED\_OFF (IO0SET = 1U << 31)

#define LED\_ON (IO0CLR = 1U << 31)

#define SW2 (IO0PIN & (1 << 14))

void delay\_ms(unsigned int j);

int main( )

{

IO0DIR = 1U << 31;

IO0SET = 1U << 31;

while(1)

{

if (!(IO0PIN & (1 << 14)))//(if(!SW2 )

{

IO0CLR = 1U << 31; *//LED\_ON*

delay\_ms(250);

IO0SET = 1U << 31; *//LED\_OFF*

delay\_ms(250);

}

}

}

void delay\_ms(unsigned int j)

{

unsigned int x, i;

for(i=0; i<j; i++)

{

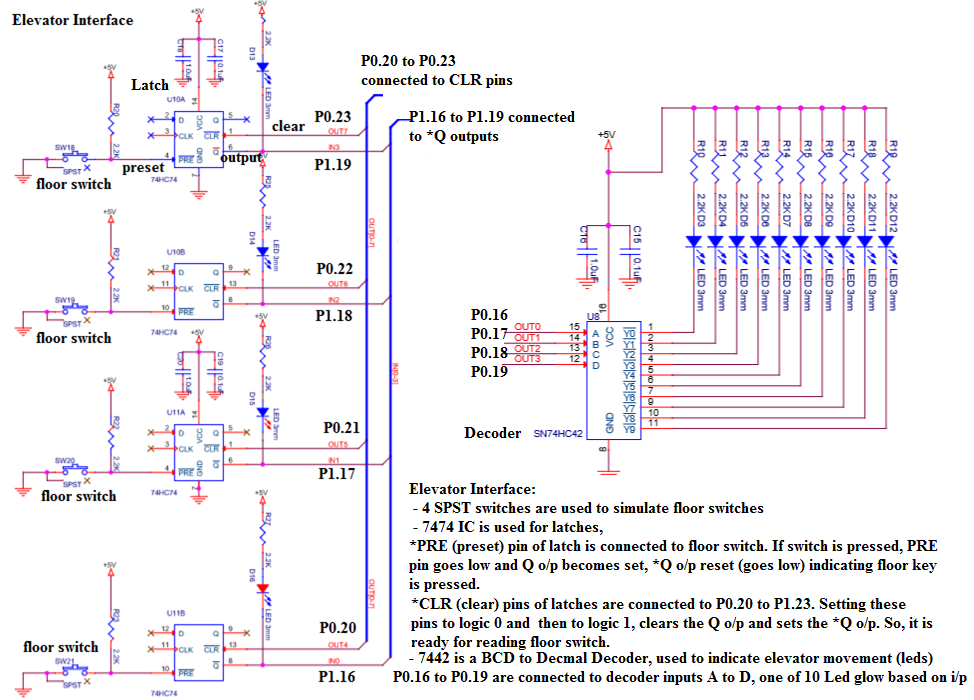
for(x=0; x<10000; x++); */\* loop to generate 1 milisecond delay with CCLK = 60MHz \*/*

}

}

**Program 1B: Elevator Interface: Write an Embedded C program to read the elevator switches and simulate elevator up and down movements**.

**Interfacing Diagram**



*//Elevator Program:*

*// P0.16 - P0.19 are connected to decoder inputs, it makes one of the o/p LEDs 0 to 9 on*

*// P0.20-P0.23 are connected to \*CLR pins of latches: make it '0' and then '1' to clear*

*// elevator keys: \*Q outputs of latches connected to P1.16 TO P1.19*

#include <lpc214x.h>

#define LED\_OFF (IO0SET = 1U << 31)

#define LED\_ON (IO0CLR = 1U << 31)

void delay\_ms(unsigned int j);

void elevator\_run(void);

int main()

{

IO0DIR |= 1U << 31 | 0XFF << 16; *// to set P0.31 & P1.20 to P1.23 as outputs*

IO1DIR |= 1 << 24; *// to set P1.24 as output*

LED\_ON; *// make D7 Led on .. just to indicate the program is running*

elevator\_run( );

while(1);

}

void elevator\_run(void)

{

int i,val;

unsigned int counter;

IO1CLR = 1 << 24; *// enable elevator section in the application board : 0 to enable*

IO0CLR = 0X000F0000; *//to set the elevator led for first floor*

do{

*// clear all the latches \*CLR*

IO0CLR = 0X00F00000;IO0SET = 0X00F00000;

*//waiting for floor key*

do{

counter = (IO1PIN >> 16) & 0X0000000F ; *// wait for any lift/elevator key press*

}while(counter == 0x0F);

if(counter == 0x0e) val=3; *//1110 - floor 1 key pressed*

else if(counter == 0x0d) val=6; *//1101 - floor 2 key pressed*

else if(counter == 0x0b) val=8; *//1011 - floor 3 key pressed*

else if(counter == 0x07) val=10; *//0111- floor 4 key pressed*

*//elevator movement-UP*

for(i=0 ; i<val ; i++)

{

IO0CLR = 0X000F0000;IO0SET = i << 16;

delay\_ms(250);

}

//elevator movement-DN

for(i=val-1;i>=0;i--)

{

IO0CLR = 0X000F0000;IO0SET = i << 16;

delay\_ms(250);

}

} while(1);

}

**Interfacing Circuit working Explanation:**

**Modifications:**

**Ask to allow/not allow intermediate key press when lift is moving.**

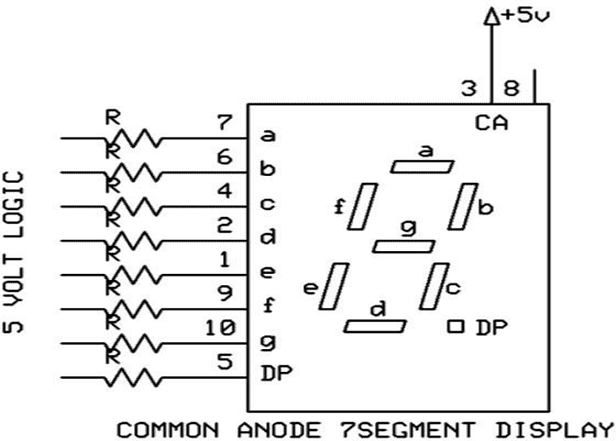
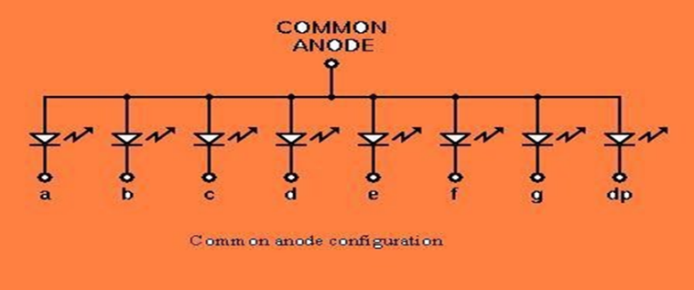
**Modify for only 2 floors**

**Use LED,(indicating Light in lift) when lift is moving**

**Output Observation:**

**Program 2B: Seven Segment Display Interface: Write a C program to display messages “FIRE” & “HELP” on 4 digit seven segment display alternately with a suitable delay.**

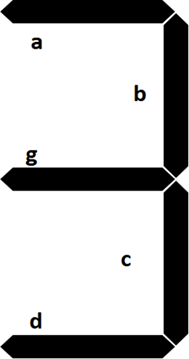
Serial In Parallel Out mode of Shift Register (74HC4094) is used to send 8 bits of data to seven segment display. Seven segment display used is of common anode type i.e. we have to send 0 to make corresponding segment ON and 1 to make it OFF.



To display 3, we have to send following bit pattern,

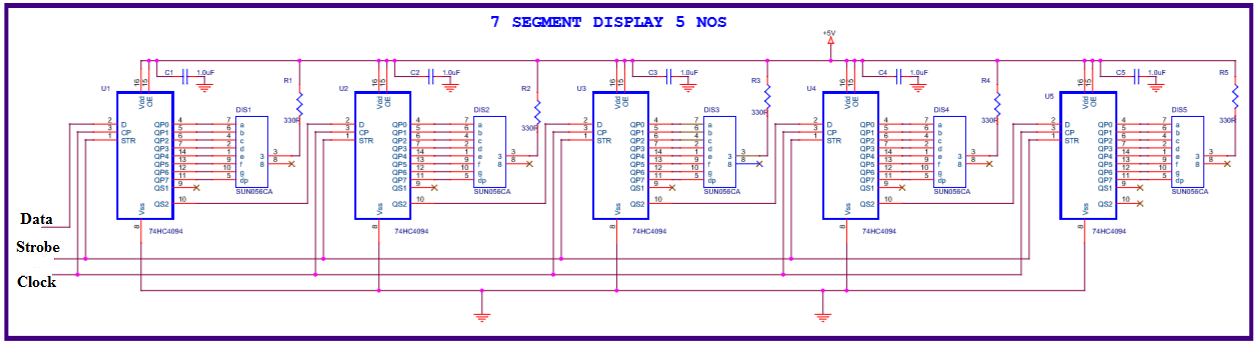
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DP | G | f | e | d | c | b | a |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

This is B0 in hexadecimal. To send B0H we have to start sending the bits from MSB onwards i.e D7 first, D6 next and so on with D0 being the last



Clock pulses are required to clock in the data, 8 clock pulses for one byte of data. As shift registers are cascaded, 8\*4=32 clocks are required to clock in 4 bytes of data. To send “12345”, first we have to send ‘1’, then ‘2’,‘3’ ,’4’ and lastly ‘5’. All the shift registers are cascaded, the data is fed to the shift register using serial in parallel out method. Strobe is used to copy the shifted data to the output pins. STB is generated after shifting is comleted.

Interfacing Diagram



*//Seven Segment Display Program:*

*//P0.19 Data pin of 1st shift register*

*//P0.20 Clock pin of shift registers, make 1 to 0*

*//P0.30 Strobe pin of shift registers: 1 to 0*

#include <lpc214x.h>

#define LED\_OFF (IO0SET = 1U << 31)

#define LED\_ON (IO0CLR = 1U << 31)

#define PLOCK 0x00000400

void delay\_ms(unsigned int j);

void SystemInit(void);

unsigned char getAlphaCode(unsigned char alphachar);

void alphadisp7SEG(char \*buf);

int main()

{

IO0DIR |= 1U << 31 | 1U << 19 | 1U << 20 | 1U << 30 ; *// to set as o/ps*

LED\_ON; *// make D7 Led on .. just indicate the program is running*

SystemInit();

while(1)

{

alphadisp7SEG("fire ");

delay\_ms(500);

alphadisp7SEG("help ");

delay\_ms(500);

}

}

unsigned char getAlphaCode(unsigned char alphachar)

{

switch (alphachar)

{

*// dp g f e d c b a - common anode: 0 segment on, 1 segment off*

case 'f': return 0x8e;

case 'i': return 0xf9;

case 'r': return 0xce;

case 'e':return 0x86; // 1000 0110

case 'h':return 0x89;

case 'l': return 0xc7;

case 'p':return 0x8c;

case ' ': return 0xff;

*//simmilarly add for other digit/characters*

default : break;

}

return 0xff;

}

void alphadisp7SEG(char \*buf)

{

unsigned char i,j;

unsigned char seg7\_data,temp=0;

for(i=0;i<5;i++) *// because only 5 seven segment digits are present*

{

seg7\_data = getAlphaCode(\*(buf+i)); *//instead of this look up table can be used*

*//to shift the segment data(8bits)to the hardware (shift registers) using Data,Clock,Strobe*

for (j=0 ; j<8; j++)

{

//get one bit of data for serial sending

temp = seg7\_data & 0x80; *// shift data from Most significan bit (D7)*

if(temp == 0x80)

IOSET0 |= 1 << 19; *//IOSET0 | 0x00080000;*

else

IOCLR0 |= 1 << 19; *//IOCLR0 | 0x00080000;*

//send one clock pulse

IOSET0 |= 1 << 20; *//IOSET0 | 0x00100000;*

delay\_ms(1);

IOCLR0 |= 1 << 20; *//IOCLR0 | 0x00100000;*

seg7\_data = seg7\_data << 1; *// get next bit into D7 position*

}

}

*// send the strobe signal*

IOSET0 |= 1 << 30; //IOSET0 | 0x40000000;

delay\_ms(1); //nop();

IOCLR0 |= 1 << 30; *//IOCLR0 | 0x40000000;*

return;

}

void SystemInit(void)

{

PLL0CON = 0x01;

PLL0CFG = 0x24;

PLL0FEED = 0xAA;

PLL0FEED = 0x55;

while( !( PLL0STAT & PLOCK ))

{ ; }

PLL0CON = 0x03;

PLL0FEED = 0xAA; *// lock the PLL registers after setting the required PLL*

PLL0FEED = 0x55;

VPBDIV = 0x01; *// PCLK is same as CCLK i.e 60Mhz*

}

void delay\_ms(unsigned int j)

{

unsigned int x,i;

for(i=0;i<j;i++)

{

for(x=0; x<10000; x++);

}

}

// CODE to display an integer number/long integer number

// long int dig\_value;

// unsigned char buf[5];

// sprintf(buf,"%05lu",dig\_value);

// alphadisp7SEG(&buf[0]);

**Interfacing Circuit working Explanation:**

**Modifications:**

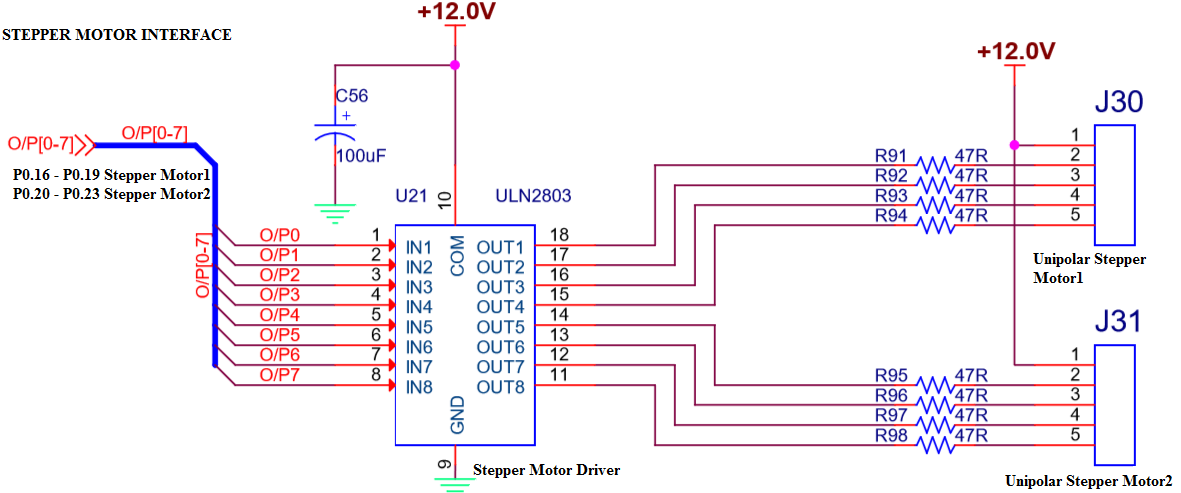
**Ask to display a number on the seven segment display (an integer)**

**Modify the program to rotate the string (which can be more than 5 characters)**

**Output Observation:**

**Program No.3b: Stepper Motor Interface: Write an Embedded C program to rotate stepper motor in clockwise direction for “M” steps, anti-clock wise direction for “N” steps.**

**Interfacing circuit diagram**



* Total number of steps for one revolution = 200 steps (200 teeth shaft)

**Step angle = 360°/200 = 1.8°**

* Use appropriate delay in between consequent steps
* 2Phase, 4winding stepper motor is used, along with driver circuit(ULN 2803) built on the RV All-In- One Card, 12v power is used to drive the stepper motor. Digital input generated by the microcontroller, is used to drive and control the direction and rotation of stepper motors. If it is required to drive bigger/higher torque stepper motors only change is- use MOSFETS or higher power stepper driver ICs to drive motors

*//Stepper Motor Program:*

*//P0.16 to P0.19 are connected to Windings of SMotor*

#include <lpc214x.h>

#define LED\_OFF (IO0SET = 1U << 31)

#define LED\_ON (IO0CLR = 1U << 31)

#define PLOCK 0x00000400

void delay\_ms(unsigned int j);

void SystemInit(void);

int main()

{

unsigned int no\_of\_steps\_clk = 100, no\_of\_steps\_aclk = 100;

IO0DIR |= 1U << 31 | 0x00FF0000 | 1U << 30*; // to set P0.16 to P0.23 as o/ps*

LED\_ON; delay\_ms(500);LED\_OFF; *// make D7 Led on .. just indicate the program is running*

SystemInit( );

do{

IO0CLR = 0X000F0000;IO0SET = 0X00010000;delay\_ms(10);if(--no\_of\_steps\_clk == 0) break;

IO0CLR = 0X000F0000;IO0SET = 0X00020000;delay\_ms(10);if(--no\_of\_steps\_clk == 0) break;

IO0CLR = 0X000F0000;IO0SET = 0X00040000;delay\_ms(10);if(--no\_of\_steps\_clk == 0) break;

IO0CLR = 0X000F0000;IO0SET = 0X00080000;delay\_ms(10);if(--no\_of\_steps\_clk == 0) break;

}while(1);

do{

IO0CLR = 0X000F0000;IO0SET = 0X00080000;delay\_ms(10);if(--no\_of\_steps\_aclk == 0) break;

IO0CLR = 0X000F0000;IO0SET = 0X00040000;delay\_ms(10);if(--no\_of\_steps\_aclk == 0) break;

IO0CLR = 0X000F0000;IO0SET = 0X00020000;delay\_ms(10);if(--no\_of\_steps\_aclk == 0) break;

IO0CLR = 0X000F0000;IO0SET = 0X00010000;delay\_ms(10);if(--no\_of\_steps\_aclk == 0) break;

}while(1);

IO0CLR = 0X00FF0000;

while(1);

}

void delay\_ms(unsigned int j)

{

unsigned int x,i;

for(i=0;i<j;i++)

{

for(x=0; x<10000; x++);

}

}

**Interfacing Circuit working Explanation:**

**Modifications:**

**Ask to rotate the stepper motor to rotate in the given rpm**

**Ask to rotate two stepper motors work simultaneously, in the opposite direction/same direction.**

**Output Observation:**

**Program No.4b: DAC Interface : Write an Embedded C program to generate sine, full rectified sine, Triangular, Sawtooth and Square waveforms using DAC module**



* DAC module of LPC 2148 is 10 bit Digital to Analog converter used to convert 10 bit Digital data to corresponding Analog voltage.
* Digital I/P : **000 to 3FF (0 to 1023)**, corresponding Analog O/P : **0V to 3.3V**
* Resolution = (3.3/1024) ≈ **3.2mili volts**

**Look up Table Creation**: Look up tables are used extensively in embedded systems, to store precomputed digital values, corresponding to analog voltages and used to generate different waveforms using DAC Module. Here the explanation about creating sine table is given.

Formula for calculation of the sine table entries: **512 + 511 x Sin Ѳ**

(512 Corresponds to 1FFh, i.e. 3.3/2 V, 511 x SIN 90 gives 511, so 512 + 511 = 1023 (for 3.3V). Calculate the digital values to be outputted to DAC for angles in the steps of 6o,

511 x sin 0 = 0 511 x sin 48 = 380

511 x sin 6 = 53 511 x sin 54 = 413

511 x sin 12 = 106 511 x sin 60 = 442

511 x sin 18 = 158 511 x sin 66 = 467

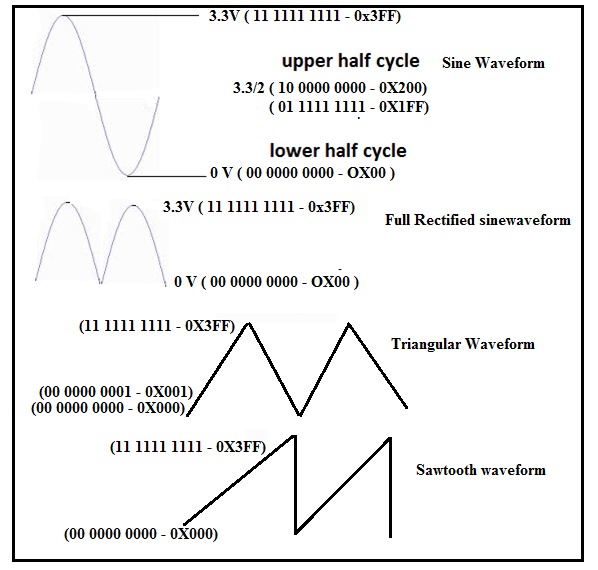
511 x sin 24 = 208 511 x sin 72 = 486

511 x sin 30 = 256 511 x sin 80 = 503

511 x sin 36 = 300 511 x sin 86 = 510

511 x sin 42 = 342 511 x sin 90 = 511

Output the above values in the reverse order to get other portion of the top half cycle, (add 512 for top half cycle, and subtract from 512 for the lower half cycle, refer the table declaration).

****

//Alpha-numeric LCD Interface (4Lines,20characters)

//Connected in 4bit nibble mode

//LCD handshaking:RS->P0.20,EN->P0.25 ,R/W -Gnd

//LCD data:D4,D5,D6,D7 -> P0.16,P0.17,P0.18,P0.19

#include <lpc214x.h>

#include <stdio.h>

#define PLOCK 0x00000400

#define LED\_OFF (IO0SET = 1U << 31)

#define LED\_ON (IO0CLR = 1U << 31)

#define SW2 (IO0PIN & (1 << 14))

#define SW3 (IO0PIN & (1 << 15))

#define SW4 (IO1PIN & (1 << 18))

#define SW5 (IO1PIN & (1 << 19))

#define SW6 (IO1PIN & (1 << 20))

void SystemInit(void);

static void delay\_ms(unsigned int j);//millisecond delay

short int sine\_table[ ] = {512+0,512+53,512+106,512+158,512+208,512+256,512+300,512+342,512+380,512+413,512+442,512+467,512+486,512+503,512+510,512+511, 512+510,512+503,512+486,512+467,512+442,512+413,512+380,512+342,512+300,512+256,512+208,512+158,512+106,512+53,512+0,

512-53,512-106,512-158,512-208,512-256,512-300,512-342,512-380,512-413,512-442,512-467,512-486,512-503,512-510,512-511,

512-510,512-503,512-486,512-467,512-442,512-413,512-380,512-342,512-300,512-256,512-208,512-158,512-106,512-53};

short int sine\_rect\_table[ ] =

{512+0,512+53,512+106,512+158,512+208,512+256,512+300,512+342,512+380,512+413,512+442,512+467,512+486,512+503,512+510,512+511,

512+510,512+503,512+486,512+467,512+442,512+413,512+380,512+342,512+300,512+256,512+208,512+158,512+106,512+53,512+0};

int main()

{

short int value,i=0;

SystemInit();

PINSEL1 |= 0x00080000; /\* P0.25 as DAC output :option 3 - 10 (bits18,19)\*/

IO0DIR |= 1U << 31 | 0x00FF0000 ; // to set P0.16 to P0.23 as o/ps

while(1)

{

if (!SW2) /\* If switch for sine wave is pressed \*/

{

while (i!=60 )

{

value = sine\_table[i++];

DACR = ( (1<<16) | (value<<6) );

delay\_ms(1);

}

i=0;

}

else if (!SW3)

{

while ( i!=30 )

{

value = sine\_rect\_table[i++];

DACR = ( (1<<16) | (value<<6) );

delay\_ms(1);

}

i=0;

}

else if ( !SW4) /\* If switch for triangular wave is pressed \*/

{

value = 0;

while ( value != 1023 )

{

DACR = ( (1<<16) | (value<<6) );

value++;

}

while ( value != 0 )

{

DACR = ( (1<<16) | (value<<6) );

value--;

}

}

else if ( !SW5 ) /\* If switch for sawtooth wave is pressed \*/

{

value = 0;

while ( value != 1023 )

{

DACR = ( (1<<16) | (value<<6) );

value++;

}

}

else if ( !SW6 ) /\* If switch for square wave is pressed \*/

{

value = 1023;

DACR = ( (1<<16) | (value<<6) );

delay\_ms(1);

value = 0;

DACR = ( (1<<16) | (value<<6) );

delay\_ms(1);

}

else /\* If no switch is pressed, 3.3V DC \*/

{

value = 1023;

DACR = ( (1<<16) | (value<<6) );

}

}

}

void SystemInit(void)

{

PLL0CON = 0x01;

PLL0CFG = 0x24;

PLL0FEED = 0xAA;

PLL0FEED = 0x55;

while( !( PLL0STAT & PLOCK ))

{ ; }

PLL0CON = 0x03;

PLL0FEED = 0xAA;

PLL0FEED = 0x55;

}

void delay\_ms(unsigned int j)

{

unsigned int x,i;

for(i=0;i<j;i++)

{

for(x=0; x<10000; x++);

}

}

**Interfacing Circuit working Explanation:**

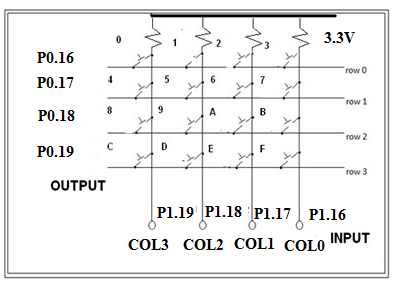
**Modifications:**

**Ask the students to do measurements using oscilloscope.**

**Output Observation:**

**Program No.5b : Matrix Keyboard Interface : Write an embedded C program to interface 4 X 4 matrix keyboard using lookup table and display the key pressed on the Terminal.**

**Interfacing Diagram**



Working method:

* If no key is pressed, we will have on columns 0-3, ‘1111’ on P1.16 to P1.19, as all the inputs are pulled up by pull up resistors.
* If we press any key, let ‘0’ key be pressed, it will short row0 and col0 lines (P0.16 & P1.19), so whatever data (0 or 1) available at row0 (P0.16) is available at col0 (P1.19). Since already columns are pulled high, it is required to apply logic ‘0’ to see change in col0 when the key is pressed.
* To identify which key is pressed,
* Check for a key press in first row by out putting – ‘0111’on row’s, check which column data is changed, if no key press go for next row
* Check for a key press in second row by out putting – ‘1011’on row’s, check which column data is changed, if no key press go for next row
* Check for a key press in third row by out putting – ‘1101’on row’s, check which column data is changed, if no key press go for next row
* Check for a key press in last row by out putting – ‘1110’on row’s, if no key is pressed go for the first row again
* Once the key press is found, use the row number and column number and look up table to convert the key position corresponding to ascii code. Use appropriate delay for debouncing.

*//Matrix 4 x 4 Keyboard*

*//Columns & Rows are pulled to +5v,if dont press key, we receive '1' on columns*

*//Method: Sending '0' to a selected row, checking for '0' on each column*

*//ROWS - ROW0-ROW3 -> P0.16,P0.17,P0.18,P0.19*

*//COLS - COL0-COL3 -> P1.19,P1.18,P1.17,P1.16*

#include <lpc214x.h>

#define PLOCK 0x00000400

#define LED\_OFF (IO0SET = 1U << 31)

#define LED\_ON (IO0CLR = 1U << 31)

#define COL0 (IO1PIN & 1 <<19)

#define COL1 (IO1PIN & 1 <<18)

#define COL2 (IO1PIN & 1 <<17)

#define COL3 (IO1PIN & 1 <<16)

void SystemInit(void);

void delay\_ms(unsigned int j);

void uart\_init(void);

unsigned char lookup\_table[4][4]={ {'0', '1', '2','3'},

{'4', '5', '6','7'},

{'8', '9', 'a','b'},

{'c', 'd', 'e','f'}};

unsigned char rowsel=0,colsel=0;

int main( )

{

SystemInit();

uart\_init();//initialize UART0 port

IO0DIR |= 1U << 31 | 0x00FF0000; // to set P0.16 to P0.23 as o/ps

//make D7 Led on off for testing

LED\_ON; delay\_ms(500);LED\_OFF;delay\_ms(500);

do

{

while(1)

{

*//check for keypress in row0,make row0 '0',row1=row2=row3='1'*

rowsel=0;IO0SET = 0X000F0000;IO0CLR = 1 << 16;

if(COL0==0){colsel=0;break;};if(COL1==0){colsel=1;break;};

if(COL2==0){colsel=2;break;};if(COL3==0){colsel=3;break;};

*//check for keypress in row1,make row1 '0'*

rowsel=1;IO0SET = 0X000F0000;IO0CLR = 1 << 17;

if(COL0==0){colsel=0;break;};if(COL1==0){colsel=1;break;};

if(COL2==0){colsel=2;break;};if(COL3==0){colsel=3;break;};

*//check for keypress in row2,make row2 '0'*

rowsel=2;IO0SET = 0X000F0000;IO0CLR = 1 << 18;//make row2 '0'

if(COL0==0){colsel=0;break;};if(COL1==0){colsel=1;break;};

if(COL2==0){colsel=2;break;};if(COL3==0){colsel=3;break;};

*//check for keypress in row3,make row3 '0'*

rowsel=3;IO0SET = 0X000F0000;IO0CLR = 1 << 19;//make row3 '0'

if(COL0==0){colsel=0;break;};if(COL1==0){colsel=1;break;};

if(COL2==0){colsel=2;break;};if(COL3==0){colsel=3;break;};

};

delay\_ms(50); *//allow for key debouncing*

while(COL0==0 || COL1==0 || COL2==0 || COL3==0);//wait for key release

delay\_ms(50); *//allow for key debouncing*

IO0SET = 0X000F0000; //disable all the rows

U0THR = lookup\_table[rowsel][colsel]; *//send to serial port(check on the terminal)*

}

while(1);

}

void uart\_init(void)

{

*//configurations to use serial port*

PINSEL0 |= 0x00000005; *// P0.0 & P0.1 ARE CONFIGURED AS TXD0 & RXD0*

U0LCR = 0x83; */\* 8 bits, no Parity, 1 Stop bit \*/*

U0DLM = 0; U0DLL = 8; *// 115200 baud rate*

U0LCR = 0x03; */\* DLAB = 0 \*/*

U0FCR = 0x07;  */\* Enable and reset TX and RX FIFO. \*/*

}

void SystemInit(void)

{

PLL0CON = 0x01;

PLL0CFG = 0x24;

PLL0FEED = 0xAA;

PLL0FEED = 0x55;

while( !( PLL0STAT & PLOCK ))

{ ; }

PLL0CON = 0x03;

PLL0FEED = 0xAA; *// lock the PLL registers after setting the required PLL*

PLL0FEED = 0x55;

VPBDIV = 0x01; *// PCLK is same as CCLK i.e 60Mhz*

}

void delay\_ms(unsigned int j)

{

unsigned int x,i;

for(i=0;i<j;i++)

{

for(x=0; x<10000; x++);

}

}

**Interfacing Circuit working Explanation:**

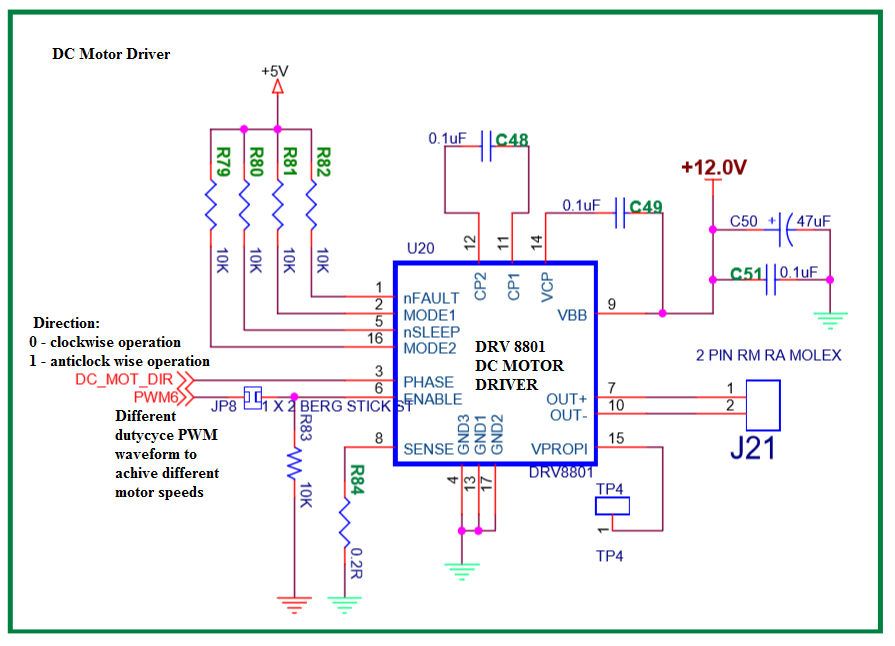
**Modifications:**

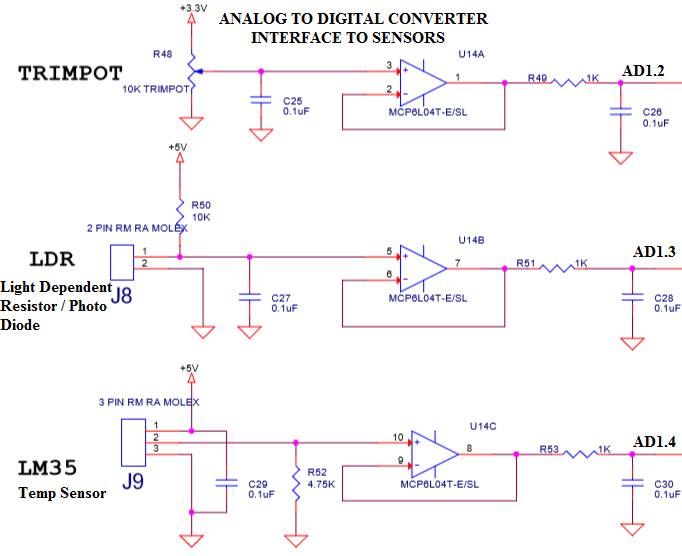
**Ask the students to read 2/3/4 digit number and display on the seven segment display**

**Output Observation:**

**Program No. 6b: DC Motor Interface: Write an Embedded C program to generate PWM wave to control speed of DC motor. Control the duty cycle by analog input fed from potentiometer.**

**Interfacing Diagram**

****



//DC Motor Speed Control

//P0.28 - used for direction control

//P0.9 - used for speed,generated by PWM6

//duty cycle - 0 to 100 controlled by PWM, fed from Potentiameter connected to ADC

#include <lpc214x.h>

#define LED\_OFF (IO0SET = 1U << 31)

#define LED\_ON (IO0CLR = 1U << 31)

#define PLOCK 0x00000400

void delay\_ms(unsigned int j);

void SystemInit(void);

void runDCMotor(int direction,int dutycycle);

unsigned int adc(int no,int ch);

int main()

{

int dig\_val;

IO0DIR |= 1U << 31 | 0x00FF0000 | 1U << 30; *// to set P0.16 to P0.23 as o/ps*

LED\_ON; delay\_ms(500);LED\_OFF; *// make D7 Led on / off for program checking*

SystemInit( );

do{

dig\_val = adc(1,2) / 10;

if(dig\_val > 100) dig\_val =100;

runDCMotor(2,dig\_val); *// run at 10% duty cycle*

}

while(1);

}

void runDCMotor(int direction,int dutycycle)

{

IO0DIR |= 1U << 28; *//set P0.28 as output pin*

PINSEL0 |= 2 << 18; *//select P0.9 as PWM6 (option 2)*

if (direction == 1)

IO0SET = 1 << 28; *//set to 1, to choose anti-clockwise direction*

else

IO0CLR = 1 << 28; *//set to 0, to choose clockwise direction*

PWMPCR = (1 << 14); *// enable PWM6*

PWMMR0 = 1000; *// set PULSE rate to value suitable for DC Motor operation*

PWMMR6 = (1000U\*dutycycle)/100; *// set PULSE period*

PWMTCR = 0x00000009; *// bit D3 = 1 (enable PWM), bit D0=1 (start the timer)*  PWMLER = 0X70; *// load the new values to PWMMR0 and PWMMR6 registers*

}

unsigned int adc(int no,int ch)

{

*// adc(1,4) for temp sensor LM34, digital value will increase as temp increases*

*// adc(1,3) for LDR - digival value will reduce as the light increases*

*// adc(1,2) for trimpot - digital value changes as the pot rotation*

unsigned int val;

PINSEL0 |= 0x0F300000; */\* Select the P0\_13 AD1.4 for ADC function \*/*

*/\* Select the P0\_12 AD1.3 for ADC function \*/*

*/\* Select the P0\_10 AD1.2 for ADC function \*/*

switch (no) *//select adc*

{

case 0: AD0CR=0x00200600|(1<<ch); *//select channel*

AD0CR|=(1<<24); *//start conversion*

while((AD0GDR& (1U<<31))==0);

val=AD0GDR;

break;

case 1: AD1CR=0x00200600|(1<<ch); *//select channel*

AD1CR|=(1<<24); *//start conversion*

while((AD1GDR&(1U<<31))==0);

val=AD1GDR;

break;

}

val=(val >> 6) & 0x03FF; *// bit 6:15 is 10 bit AD value*

return val;

}

void SystemInit(void)

{

PLL0CON = 0x01;

PLL0CFG = 0x24;

PLL0FEED = 0xAA;

PLL0FEED = 0x55;

while( !( PLL0STAT & PLOCK ))

{ ; }

PLL0CON = 0x03;

PLL0FEED = 0xAA; *// lock the PLL registers after setting the required PLL*

PLL0FEED = 0x55;

VPBDIV = 0x01; *// PCLK is same as CCLK i.e 60Mhz*

}

void delay\_ms(unsigned int j)

{

unsigned int x,i;

for(i=0;i<j;i++)

{

for(x=0; x<10000; x++);

}

}

**Interfacing Circuit working Explanation:**

**Modifications:**

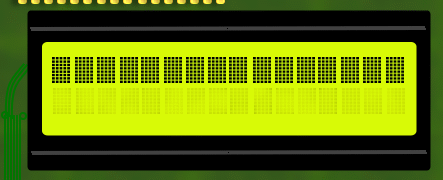
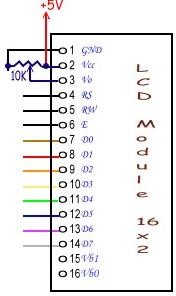
**Ask the student to link motor rotation with reference to temperature/light intensity.**

**Output Observation:**

**Program No.7b: Alpha Numeric LCD Interface - Write an Embedded C program to display text messages on the multiple lines of the display.**

LCD’s are preferred to seven segment displays because of their versatility and capability to house more information.2 line (2 x 6) and 4 line (4x20) is the most popular, low cost character oriented LCD, suitable for understanding the working and programming of LCD. You have seen LCD modules used in many of the electronics devices like coin phone, billing machine and weighing machines. It is a powerful display options for stand-alone systems. Because of low power dissipation, high readability, flexibility for programmers, LCD modules are becoming popular.

LCD consists of DDRAM, CGROM, Shift registers, bit/pixel drivers, refreshing logics and lcd controller. The data to be displayed on lcd, is to be written on to the DDRAM-display data Ram using the ascii format. CGROM-Character generator rom, contains dot/pixel patterns for every character to be displayed (pre programmed). Shift registers are used to convert CGROM parallel data to serial data(serializing), drivers are required to drive (ON/OFF) the bits, refreshing logics are required to hold the display data, as the dots are displayed row by row basis continuously, like in CRT.

LCD provides many control pins, to enable the microcontroller or microprocessor to communicate, whatever the data we write to LCD is of two types, either it is a command to the LCD(to configure) or ASCII code of character to be displayed on LCD (to DDRAM). RS signal is used for this,

**RS (Register Select): 0 or 1**

0 - writing command byte into command register of LCD

1 - writing data (ASCII code) into Data register of LCD

**R/W : (Read/Write) (In our kit, R/W is grounded)**

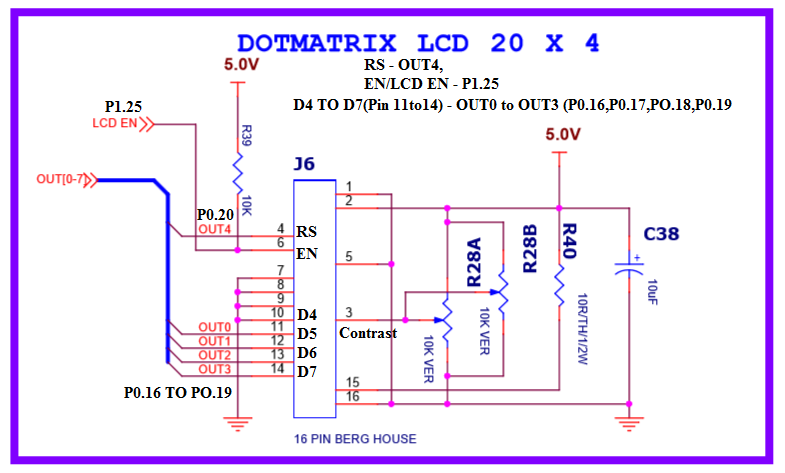
1. Write to LCD (Data/Command)
2. Read from the LCD

**E (Enable) : 1 to 0 pulse**

- Enable is required to perform the writing/reading to LCD, E – ‘1’ (for 450nsec) & then ‘0’ (High to Low Pulse)

**D0-D7** - It is a bidirectional data bus, used to write data/command to LCD or reading status. In 4bit nibble mode, only lines D4 – D7 are used for communication.

**Interfacing Diagram**



**LCD Command Table**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** | **Description** |
| Clear display | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Clears Display and returns cursor to home position. |
| Cursor home | 0 | 0 | 0 | 0 | 0 | 0 | 1 | X | Returns cursor to home position. Also returns display being shifted to the original position. |
| Entry mode set | 0 | 0 | 0 | 0 | 0 | 1 | I/D | S | I/D = 0 - cursor is in decrement position. I/D = 1 - cursor is in increment position.  S = 0 - Shift is invisible.  S = 1 - Shift is visible |
| Display ON- OFF  Control | 0 | 0 | 0 | 0 | 1 | D | C | B | D- Display, C- Cursor, B-Blinking cursor  0 - OFF  1 - ON |
| Cursor/ Display Shift | 0 | 0 | 0 | 1 | S/C | R/L | X | X | S/C = 0 - Move cursor. S/C = 1 - Shift display. R/L = 0 - Shift left.  R/L = 1- Shift right. |
| Function Set | 0 | 0 | 1 | DL | N | F | X | X | DL = 0 - 4 bit interface. DL = 1 - 8 bit interface.  N = 0 - 1/8 or 1/11 Duty (1 line).  N = 1 - 1/16 Duty (2 lines).  F = 0 - 5x7 dots.  F = 1 - 5x10 dots. |

**Programming 4 x 20 alphanumeric LCD**

Two steps are involved,

1. Configure the LCD for different parameters/settings, by writing series of commands (command bytes) like
   * Function set command(0x28)
   * Display On command(0x0C)
   * Clear display (0x01)
2. Writing actual string data to LCD, character by character, (by default characters are displayed from line1 first column position, we can issue DDRAM address command - 0x80 + char pos, for first line, 0xc0 + char pos, for second line,0x94+char pos, for third line, 0xD4+char pos, for fourth line).

*//Alpha-numeric LCD Interface (4Lines,20characters)*

*//Connected in 4bit nibble mode*

*//LCD handshaking:RS->P0.20,EN->P0.25 ,R/W -Gnd*

*//LCD data:D4,D5,D6,D7 -> P0.16,P0.17,P0.18,P0.19*

#include <lpc214x.h>

#define PLOCK 0x00000400

#define LED\_OFF (IO0SET = 1U << 31)

#define LED\_ON (IO0CLR = 1U << 31)

#define RS\_ON (IO0SET = 1U << 20)

#define RS\_OFF (IO0CLR = 1U << 20)

#define EN\_ON (IO1SET = 1U << 25)

#define EN\_OFF (IO1CLR = 1U << 25)

void SystemInit(void);

static void delay\_ms(unsigned int j);//millisecond delay

static void delay\_us(unsigned int count);//microsecond delay

static void LCD\_SendCmdSignals(void);

static void LCD\_SendDataSignals(void);

static void LCD\_SendHigherNibble(unsigned char dataByte);

static void LCD\_CmdWrite( unsigned char cmdByte);

static void LCD\_DataWrite( unsigned char dataByte);

static void LCD\_Reset(void);

static void LCD\_Init(void);

void LCD\_DisplayString(const char \*ptr\_stringPointer\_u8);

int main()

{

SystemInit();

IO0DIR |= 1U << 31 | 0x00FF0000 ; *// to set P0.16 to P0.23 as o/ps*

IO1DIR |= 1U << 25; // *to set P1.25 as o/p used for EN*

*// make D7 Led on off for testing*

LED\_ON; delay\_ms(500);LED\_OFF;delay\_ms(500);

LCD\_Reset();

LCD\_Init();

delay\_ms(100);

LCD\_CmdWrite(0x80); LCD\_DisplayString("RV College Of Engrng");

LCD\_CmdWrite(0xc0); LCD\_DisplayString(" Computer Sciene");

LCD\_CmdWrite(0x94); LCD\_DisplayString(" 4th Semester");

LCD\_CmdWrite(0xD4); LCD\_DisplayString(" B Section");

while(1);

}

static void LCD\_CmdWrite( unsigned char cmdByte)

{

LCD\_SendHigherNibble(cmdByte);

LCD\_SendCmdSignals();

cmdByte = cmdByte << 4;

LCD\_SendHigherNibble(cmdByte);

LCD\_SendCmdSignals();

}

static void LCD\_DataWrite( unsigned char dataByte)

{

LCD\_SendHigherNibble(dataByte);

LCD\_SendDataSignals();

dataByte = dataByte << 4;

LCD\_SendHigherNibble(dataByte);

LCD\_SendDataSignals();

}

static void LCD\_Reset(void)

{

/\* LCD reset sequence for 4-bit mode\*/

LCD\_SendHigherNibble(0x30);

LCD\_SendCmdSignals();

delay\_ms(100);

LCD\_SendHigherNibble(0x30);

LCD\_SendCmdSignals();

delay\_us(200);

LCD\_SendHigherNibble(0x30);

LCD\_SendCmdSignals();

delay\_us(200);

LCD\_SendHigherNibble(0x20);

LCD\_SendCmdSignals();

delay\_us(200);

}

static void LCD\_SendHigherNibble(unsigned char dataByte)

{

*//send the D7,6,5,D4(uppernibble) to P0.16 to P0.19*

IO0CLR = 0X000F0000;IO0SET = ((dataByte >>4) & 0x0f) << 16;

}

static void LCD\_SendCmdSignals(void)

{

RS\_OFF; *// RS - 1*

EN\_ON;delay\_us(100);EN\_OFF; *// EN - 1 then 0*

}

static void LCD\_SendDataSignals(void)

{

RS\_ON;// RS - 1

EN\_ON;delay\_us(100);EN\_OFF; // EN - 1 then 0

}

static void LCD\_Init(void)

{

delay\_ms(100);

LCD\_Reset();

LCD\_CmdWrite(0x28u); *//Initialize the LCD for 4-bit 5x7 matrix type*

LCD\_CmdWrite(0x0Eu); *// Display ON cursor ON*

LCD\_CmdWrite(0x01u); *//Clear the LCD*

LCD\_CmdWrite(0x80u); *//go to First line First Position*

}

void LCD\_DisplayString(const char \*ptr\_string)

{

*// Loop through the string and display char by char*

while((\*ptr\_string)!=0)

LCD\_DataWrite(\*ptr\_string++);

}

static void delay\_us(unsigned int count)

{

unsigned int j=0,i=0;

for(j=0;j<count;j++)

{

for(i=0;i<10;i++);

}

}

void SystemInit(void)

{

PLL0CON = 0x01;

PLL0CFG = 0x24;

PLL0FEED = 0xAA;

PLL0FEED = 0x55;

while( !( PLL0STAT & PLOCK ))

{ ; }

PLL0CON = 0x03;

PLL0FEED = 0xAA; *// lock the PLL registers after setting the required PLL*

PLL0FEED = 0x55;

VPBDIV = 0x01; *// PCLK is same as CCLK i.e 60Mhz*

}

void delay\_ms(unsigned int j)

{

unsigned int x,i;

for(i=0;i<j;i++)

{

for(x=0; x<10000; x++);

}

}

**Interfacing Circuit working Explanation:**

**Modifications:**

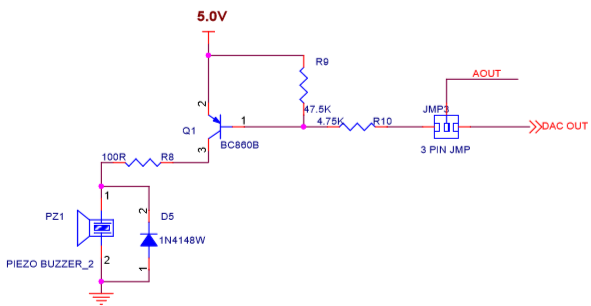
**Ask the student to display a given number on the LCD (instead of string). Also display multiple parameters (string, multiple numbers at different places).**

**Output Observation:**

Mini Project

**(Use Buzzer , Joystick, RGB LED, Logic Controller DIP Switch, Temperature Sensor, LDR, Proximity Sensor, Servomotor along with the earlier learned interfaces to develop the Mini project.)**

**BUZZER INTERFACE & SAMPLE PROGRAM**



//DAC output - P0.25 (option 3:10) , JUM-3:selection for buzzer/CPU-DAC,

//JMP-A10:selection for audio/CPU DAC

//DAC Register - It is a 32 bit register, Bits:D15-D6 holds 10bit digital data; D16 - BIAS bit, set //to 1

PINSEL1 |= 0x00080000; /\* P0.25 as DAC output :option 3 - 10 (bits18,19)\*/

void beep(unsigned int val)

{

//P0.25 used as AOUT - DAC output

//val - 0 to 1023 : 10 bit DAC, P0.25 used as AOUT generates analog output

DACR = ( (1<<16) | (val<<6) );

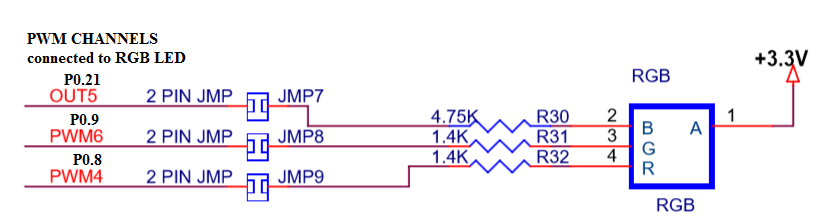
delay\_ms(500);

DACR = ( (1<<16) | (1023<<6) );

delay\_ms(1);

}

**RGB LED INTERFACE & SAMPLE PROGRAM**



void PWM\_RGBLed\_test(void)

{

unsigned int i;

PWM\_Init();

for(i=0;i<=100;i=i+1)

{

PWMMR4 = 100; PWMMR5 = i; PWMMR6 = 100;

PWMLER = 0X70*; // to enable copy to Match registers from shadow regs. 1110000*

delay\_ms(1000);

}

for(i=0;i<=100;i=i+1)

{

PWMMR4 = 100; PWMMR5 = 100; PWMMR6 = i;

PWMLER = 0X70; *// to enable copy to Match registers from shadow regs. 1110000*

delay\_ms(1000);

}

for(i=0;i<=100;i=i+1)

{

PWMMR4 = i PWMMR5 = 100; PWMMR6 = 100;

PWMLER = 0X70*; // to enable copy to Match registers from shadow regs. 1110000*

delay\_ms(1000);

}

}

void PWM\_Init(void)

{

PINSEL0 |= 2 << 18 | 2 << 16 ;

*//SELECT P0.8 PWM4 AND P0.9PWM6 AS 2ND OPTION FOR PWM OPERATION*

PINSEL1 |= 1 << 10; *//SELECT P0.21 PWM5 AS OPTION 1 FOR PWM OPERATION*

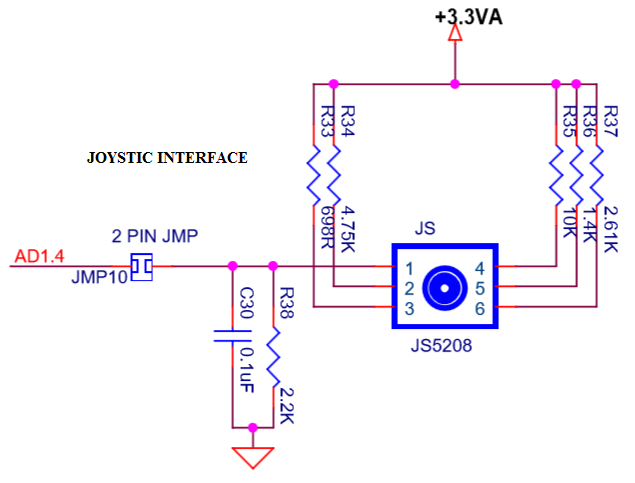
PWMPCR = (1 << 12 | 1 << 13 | 1 << 14); *//Enable PWM4,PWM5 and PWM*

PWMMR0 = 100; *// load the value to MR0 to fix the pulse rate*

PWMTCR = 0x00000009; *// bit D3 = 1 (enable PWM), bit D0=1 (start the timer)*

}

**Joystick Interface**



char Joystick\_position(void)

{

// returns joystic position

// Up , Left , Down, Right & Enter

unsigned int adc\_result=0;

char res =6;

PINSEL0|= 0x0C000000; /\* Select the P0\_13 AD1.4 for ADC function \*/

adc\_result = adc(1,4);

if(adc\_result == 0 && adc\_result < 5) res =6;

else if(adc\_result > 180 && adc\_result < 185) res = 0; // Up

else if(adc\_result > 315 && adc\_result < 325) res = 1; // Left

else if(adc\_result > 460 && adc\_result < 470) res = 2; // Down

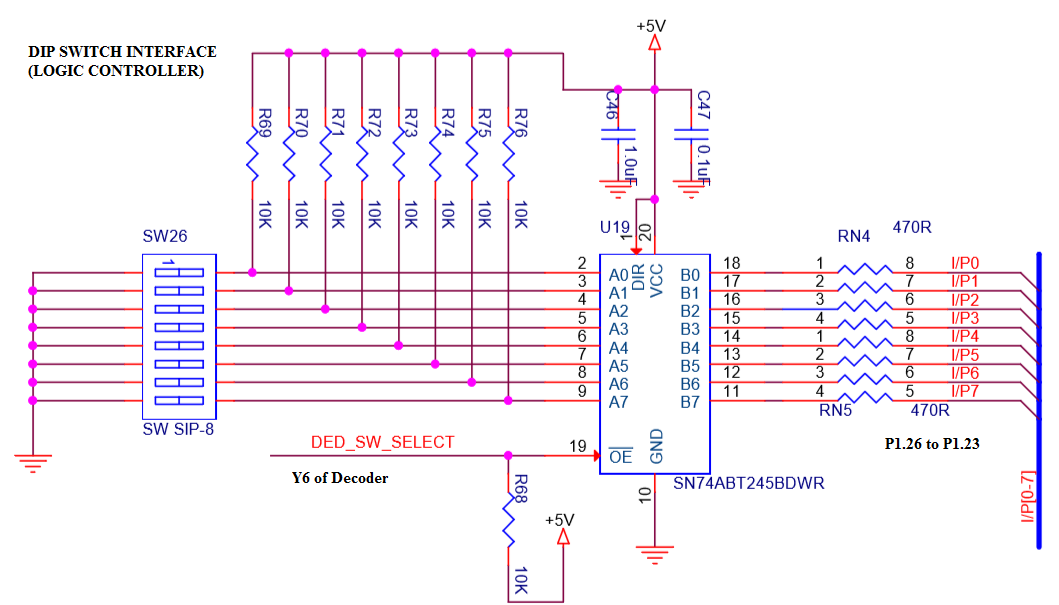
else if(adc\_result > 614 && adc\_result < 624) res = 3; // Right

else if(adc\_result > 770 && adc\_result < 800) res = 4; // Enter

return res;

}

**Logic controller 8 bit DIP Switch Interface.**



unsigned long LC\_readSW(void)

{

unsigned long result;

*//select switch: Mux2:Mux1:Mux0(P0.18,P0.17,P0.16 - 110), Mux select:DEC\_SEL : P0.7 - 0*

IO0SET = 1<<18|1<<17; IO0CLR = (1<<16)|(1<< 7);delay\_ms(2);

*//read the switches: (SW1-8)IN7-0 -> P1.23-P1.16 REFER SW26 in CIRCUIT*

result = (IO1PIN >> 16) & 0X000000FF;delay\_ms(2);

*//disable Mux: DEC\_SEL(P0.7) - 1*

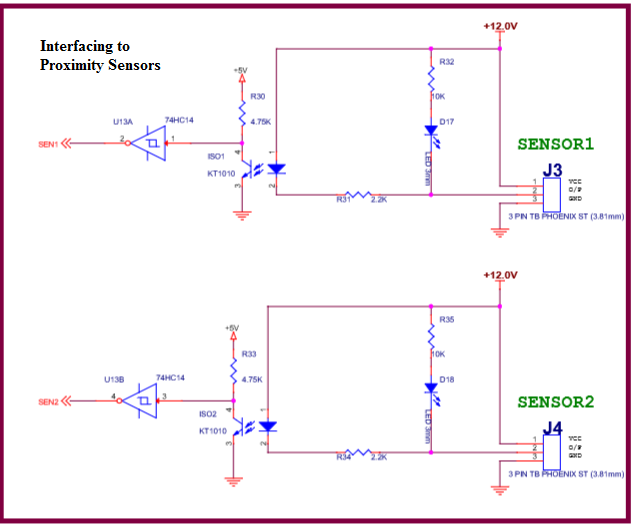
IO0SET = (1U << 7);delay\_ms(2);

*//return 8 bit result, stored in the bits D7-D0*

return result;

}

**Proximity Sensor Interface**

****

int readSensor(int sen\_no)

{

int result=0;

IO1DIR |= 1 << 24;

IO1CLR = 1<< 24; *// enable sensor logic: P1.24 - 0*

switch (sen\_no)

{

case 1: result = IO1PIN & (1<<22); *//P1.22 connected to sensor1*

break;

case 2: result = IO1PIN & (1<<23); *//P1.23 connected to sensor2*

break;

default: result = 0;

};

IO1SET = 1<< 24; *// disable sensor logic: P1.24*

return result;

}

**Appendix: Interfacing Fundammentals using LPC 2148**

Here with we have enclosed,

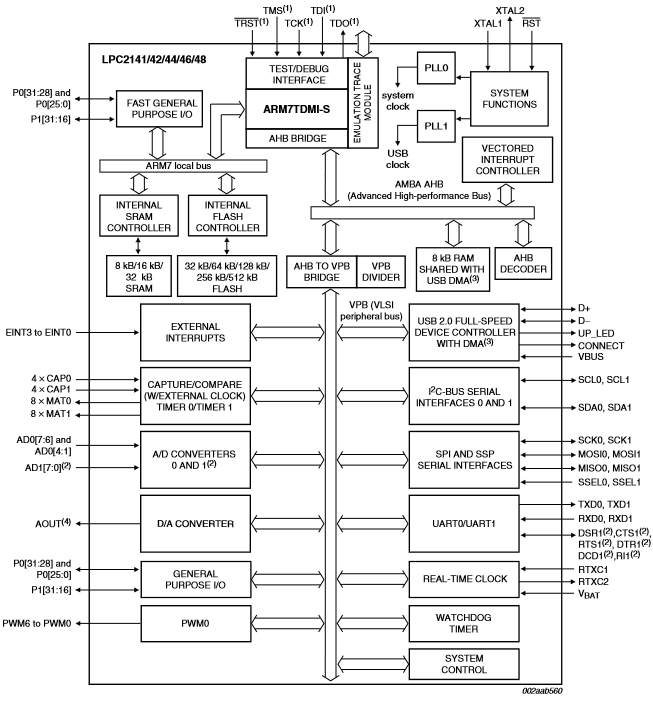
* Summary of pins of LPC 2148,
* GPIO Programming,
* Timers Programming,
* PWM Programming,
* ADC Programming,
* UART Programming,
* Writing ISR Programs

**LPC 2148 - ARM7 core based Microcontroller from NXP**

LPC 2148 is Popular ARM microcontroller from NXP Semiconductors, used my many popular industry embedded products, is based on ARM7TDMI-S ARM 7 core. [It has features like Thumb instructions set (T), JTAG debugger (D), fast multiplier (M) and the embedded ICE (I) [ICE- in circuit emulator, useful while debugging hardware & software, S – synthesizable].

**Features**

1. Uses the core ARM 7TDMI-S in a tiny LQFP64 package, operating at 60 MHz clock speed
2. 8kb-40kb of on-chip static RAM
3. 32kb to 512kb of on-chip flash memory
4. 128 bit memory accelerator enables high-speed 60MHz operation
5. USB 2.0 Full speed device controller with DMA (used as USB device, not as a host)
6. 10 bit ADCs provide a total of 6/14(2148 has 14) analog inputs
7. Single 10-bit DAC provides variable analog output
8. Two 32bit timers/event counters
9. PWM Unit
10. RTC with battery backup facility
11. 2 UARTs
12. I2C interface (2 sets)
13. SPI & SSP interfaces ( supports SD card )
14. Upto 45 GPIO pins
15. Single power supply with POR and BOD circuits, support idle & power-down modes



**Interfacing & Programming GPIO**

LPC 2148 provides 2 ports, of size 32 bits, used to interface LCD, relays, LEDs etc.

Port0 is a 32 bit I/O port, of which 28 pins can be used for digital i/o, P0.31 used only for o/p, pins P0.24,P0.26 & P0.27 are reserved and not available for use.

Port 1 is a 32 bit I/O port, but only P0.16 – P0.31 are available for I/O. Hence totally 45 I/O are possible, but all the pins have alternate functions also, as detailed below.

**Pin Connect Block** : The purpose of this is to configure the GPIO pins to the desired functions. This acts like a multiplexer. Each pin of the chip has a maximum of four functions. To select one specific function for a pin, a multiplexer with two select pins, is necessary. The select pins function is provided by the bits of the PINSEL registers. Three PINSEL registers, (PINSEL0,PINSEL1,PINSEL2) are required, first two are for the Port0. Two bits are required to configure one pin, hence PINSEL0 is used to configure the pins P0.0 to P0.15, PINSEL1 is used to configure P0.16 to P0.31, PINSEL2 is used to configure P1.16 to P1.31.

Example for **Pin P0.5 :**

Bits of PINSEL0 Register Port Pin value of PINSEL bits Function Selected Reset Value

11:10 P0.5 00 GPIO 0

1. MISO0(SPI-0)
2. Match 0.1 (timer 0)
3. AD0.7

To use P0.5 as GPIO, we have to make it 00,

PINSEL0 = 0x0000 0000; By default, on reset, all port pins act as GPIO pins

GPIO

SPIO

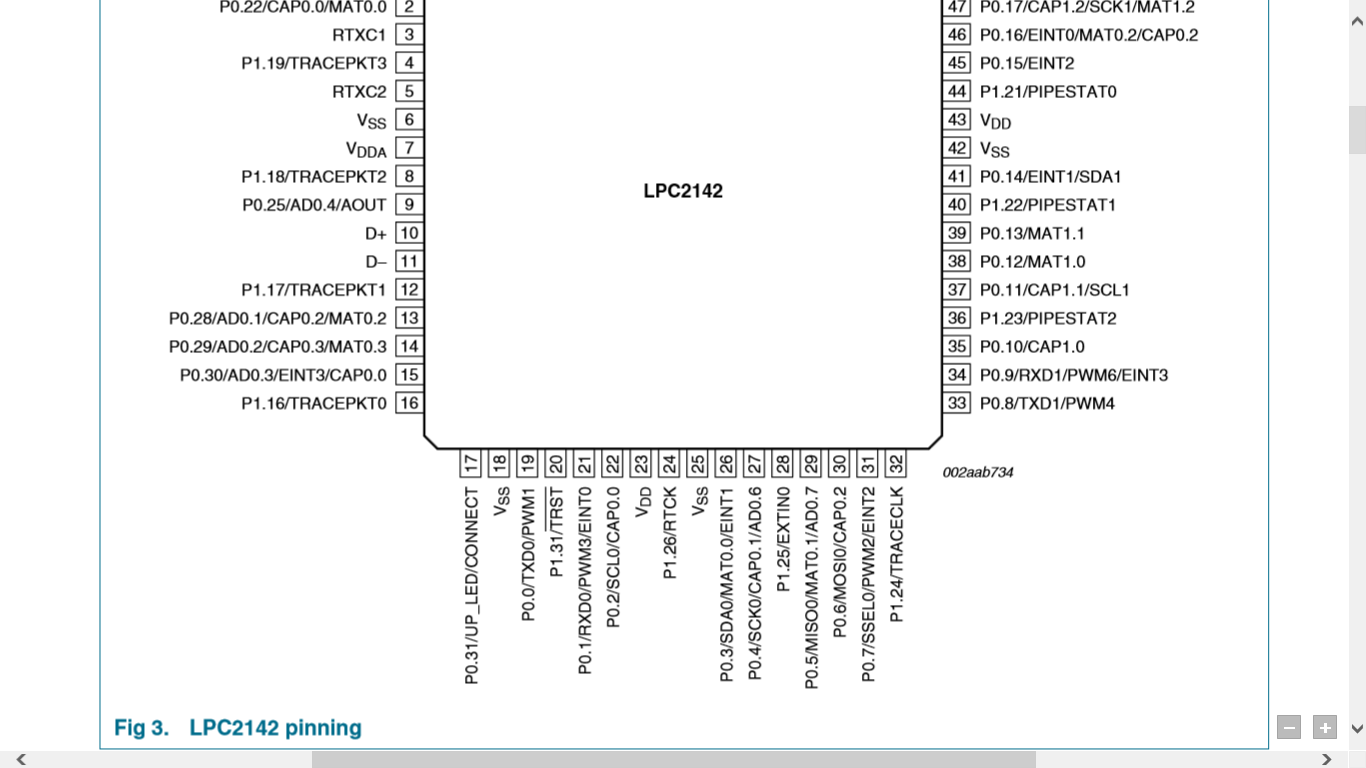
Match0.1

AD0.7

Bits of PINSEL0 Register ( 11:10)

P0.5

PIN Select Mux



**Using GPIO Pins :-** These pins are used for connecting LEDs, Switches, LCD, Relays etc. Following registers are provided to use them,

1. IODIR (IO direction register, IODIR0 for P0 & IODIR1 for P1) – The bit setting of this register configures the pin as input or output, 1 for output, 0 for input
2. IOSET(IO set register, IOSET0 & IOSET1) – This register is used to set the output pins of the chip. To make a pin to be ‘1’, the corresponding bit in the register is to be ‘1’. Writing zeros have no effect
3. IOCLR(IO Clear register, IOCLR0 & IOCLR1) – To make an output pin to have a ‘0’ value, i.e to clear it. The corresponding bit in this register has to be a ‘1’. Writing zeros have no effect.
4. IOPIN (IO Pin register, IOPIN0 & IOPIN1): From this register the value of the corresponding pin can be read, irrespective of whether the pin is an input or output pin.

**Programming TIMER Unit of LPC 2148**

Provides two 32 bit timer/counters - Timer0 & Timer1. When used as Timer PCLK is used for its counting, when used as a counter it uses external clock source for its counting.

Timer control Register

MATCH CONTROL REG

MATCH REGISTER

Timer count register

Prescaler

PCLK

control

stop

reset

interrupt

It has following registers for its working,

TC – Timer counter , 32 bits, counts from 00000000h to FFFFFFFFH

TCR- Timer control register, 8 bits, used to enable/disable timer

D7 D6 D5 D4 D3 D2 D1 D0

R E

E =1 to enable the timer , R=1 to reset the TC to zero

MR0,MR1,MR2,MR3 – 4 Match Registers, 32 bits, whose values are compared with TC value, on match certain action is performed based on the MCR (match control register)

MCR – Match Control Register, 16 bits, 3 bits for each Match register

D15 . . . D5 D4 D3 D2 D1 D0

S R I S R I

MR1 MR0

The lowest three bits are for controlling the operations related to the Match register 0, next three for MR1,MR2 and MR3, in that order. The meaning of the bits is explained below,

I - When ‘1’ , an interrupt is activated when match occurs, ‘0’ interrupt is disabled

R- When ‘1’, the timer count register is reset when match occurs

S- When ‘1’ , the timer count and the pre-scale counter will be stopped when match occurs, also timer is disabled.

**Example**: Generate the square wave of frequency 1KHz using the timer, on P1.16pin

Steps-

1. Load a number in the match register, Let us assume PCLK = 15 MHz (CCLK -60MHz,%by 4 using VPB register setting),

count = Time period of required output(Td) / time period of input frequency(T),

Td = 1/1KHz = 1 msec, half of it is 0.5msec; T = 1/15MHz = 0.067µsec

= 0.5 msec/0.067µsec = 7462

1. Load the MCR for stopping the timer on match & disable the interrupt
2. Start the timer, by enabling the ‘E’ bit in TCR
3. Now TC starts counting, when it matches with the MR value, it stops counting
4. Stop the timer

#include <LPC214x.h>

void delay(void);

int main(void)

{

T0MR0 = 7462; //use the Timer0 and load the MR0 with count

T0MCR = 0X0004; // 0000….100 – Stop the timer, after match

I0DIR1 = 0X00010000; //make P1.16 as output

While(1)

{

I0SET1 = 1 << 16; //set P1.16 to 1

delay();

I0CLR1 = 1 <<16; //clear P1.16 to 0

delay();

}

}

Void delay(void)

{

T0TCR = 1; //start the timer

While (!(T0TC == T0MR0));

T0TCR = 2; // reset the counter and stop the timer

T0TC = 0;

}

**Programming with PULSE WIDTH MODULATION UNIT**

Duty cycle = [Ton / ( Ton + Toff) ] x 100 in%

Ton Toff

PWM output is basically a waveform, whose ON time & Period (On+Off) can be varied, i.e we can generate the waveform with programmable Pulse rate (period of one cycle- On + Off time) & Pulse period(On time) and hence achieving different duty cycles ( ratio of ON time to ON+OFF time).

LPC 2148 PWM unit provides 6 PWM channels, PWM 1to 6, which can be connected to motor driver circuits, to control their speed. The registers of PWM unit are similar to timer,

**PWMTC** -PWM timer counter , 32 bits, similar to timer count register, counts up using PCLK

**PWMTCR** -PWM timer control register, similar to timer control register,used to start/stop the timer, reset the count, enable/disable PWM, ( if PWM is disabled, it acts like timer)

**MR0** -Match Register 0, its value decides pulse rate, i.e period T (ON+OFF)

**MR1 to MR6** – 32 bit Match registers, used to fix the ON time of PWM waveforms, the values of these registers are continuously compared to the count in the timer register, and generates the corresponding PWM waveforms on channels PWM1 to PWM6 , so by changing the values in the MR1-MR6 we can produce different duty cycles.

**PWMPCR** – PWM control register, used to enable and configure the PWM channels as

a) single edge

b) double edge.

In single edge the output is always high, we are only controlling the falling edge ( that happens whenever count matches to the match registers).

In double edge , we are controlling both the rising and falling edges of the pulse (i.e why we require two match registers for one channel, first match register controls the rising edge, other one controls the falling edge )

PWMPCR - D15 D14 D13……D9.. D6 D5 …..D2 D1 D0

Bits D2 to D6 used to select single edge/double edge for channels PWM2 to PWM6

Bits D9 to D14 used to enable the channels PWM1 to PWM6.

**Example: Generate the PWM waveform of 25% duty cycle**

#include <LPC214x.h>

void PWM\_Init(void)

{

//P0.1 pin has second alternate function as PWM3 channel, so using PISEL0 register

PINSEL0 |= 0x0000 0008; // Select P0.1 as PWM output , bits D2 & D3 are for P0.1

//Configure PWM channel 3 as single edge type and enable the channel

PWMPCR = 0x0000 0800; //bit D3 to select single edge(make it 0), bit D11 is for

//enabling PWM3 channel(make it 1)

//load the value to MR0 to fix the pulse rate

PWMMR0 = 2000; // any other value, I could have taken

// enable PWM unit of LPC2148 and start the timer

PWMTCR = 0x0000 0009; // bit D3 = 1 (enable PWM), bit D0=1 (start the timer)

}

int main()

{

PWM\_Init();

While(1)

{

PWMMR3 = 500; // value which decides pulse ON time, ¼ of 2000

PWMLER = 0X08;

// PWM Latch enable register, 8bit register - bit D0-D6 are for MR0- MR6,

// usually what we write to match registers will go to shadow registers, while timer is // running when the match event occurs, then it will be copied to match registers, // // only if the corresponding bits in the PWMLER is enabled

}

}

**Analog Signal Interfacing using LPC 2148**

LPC 2148 provides two Analog channels: ADC0 & ADC1. ADC0 provides 6 inputs, ADC1 provides 8 inputs. It is of 10 bit successive Approximation type, it can generate 400000 samples/sec

ADC Channel 0 Inputs:

AD0.1 (P0.28) ,AD0.2 (P0.29),AD0.3 (P0.30),AD0.4 (P0.25),AD0.6 (P0.4),AD0.7 (P0.5)

ADC Control Register (32 bit Register) :

Bits - 7 to 1: set corresponding bit to 1 to select channel

Bits - 15 to 8: (CLKDIV)- PCLK is divdided by this value to get ADC clock :

Bits – 26 to 24 : 001 Start Conversion Now

ADC Global Data Register (32 bit Register)

This register contains the ADC's DONE bit and the result of the most recent A/D conversion.

Bit 5:0 - Reserved

Bits 15:6 - Result ,when DONE bit is set to 1, this field contains 10 bit ADC result 0 to 1023, correspond to 0V to 3V ( Vref )

Bit 31 - DONE, This bit set to 1, when conversion completes, It is cleared when this register is read and when the AD0CR is written

Program Steps to Read Analog Input :

1. Select the analog input pin (1 to 6 of ADC0 or 0 to 7 of ADC1) and configure the pin as Analog input using PINSEL register
2. Write to ADC control Register:

* Make ADC operational (set the Bit 21)
* Select the channel(set any one bit of Bit0 to Bit7 – for AD0 to AD7)
* Issue SOC signal (set 001 on bits : 26 25 24)

Example: select AD0.1

AD0CR = (1 << 1) | (1 << 21) | (1<<24) ; // assuming other bits are zero

1. Check for the conversion to complete, by reading Bit31 of GDR

Example: for ADC0

while( (AD0GDR & (unsigned long) 1 << 31) == 0);

1. Read the Digital output from GDR, after aligning the result to LSB (Bit0) and masking other bits to Zero

Example:

i = (AD0GDR >> 6 ) & 0x3FF ;

Example Program :

LDR(Light Dependant Resistor ) / PhotoDiode is connected to P0.28 and LED is connected to P0.21, Read the Light intensity. Make the LED on whenever there is less/no light.

#include <lpc214x.h>

#define LED\_ON IO0SET = 1 << 21

#define LED\_OFF IO0CLR = 1 << 21

int main()

{

unsigned int i;

IO0DIR = (1 << 21) ; // P0.21-o/p

PINSEL1 = 1 << 24; // P0.28 AS AD0.1(01)

LED\_ON;

do

{

AD0CR= (1 << 1 | 1 << 21 | 1 << 24) ;

While ( (AD0GDR & (unsigned long) 1 << 31) == 0);

i = (AD0GDR >> 6 ) & 0x3FF ;

if (i > 100)

LED\_OFF;

else

LED\_ON;

}

while(1);

}

**Serial Port Programming….**

UART Registers :The below table shows the registers associated with LPC2148 UART.

|  |  |
| --- | --- |
| **Register** | **Description** |
| UxRBR | Contains the recently received Data |
| UxTHR | Contains the data to be transmitted |
| UxFCR | FIFO Control Register |
| UxLCR | Controls the UART frame formatting(Number of Data Bits, Stop bits) |
| UxDLL | Least Significant Byte of the UART baud rate generator value. |
| UxDLM | Most Significant Byte of the UART baud rate generator value. |

UxLSR Line status register, used to check transmitter is free / Receiver has data

**UxFCR ( FIFO Control Register )**

LPC2148 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.

**Bit 0 – FIFO:**  
This bit is used to enable/disable the FIFO for the data 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 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)

**UxLCR ( Line Control Register )**

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

**Bit 1:0 – WLS : WordLenghtSelect**  
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  
0-- Enable access to divisor latch

**Example:**

U0LCR = (0x03<<0) | (1<<7); 8bit data, 1Stop bit, No parity

**UxLSR (Line Status Register)**

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

**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.

**Baudrate Calculation**

There are two PLL Modules in LPC214X series of MCUs: PLL0 and PLL1. PLL0 is used to generate the system clock and PLL1 is used to generate clock signal for USB. Both PLL0 and PLL1 accept the input clock signal in the range of 10 MHz to 24 MHz but the output of PLL0 is multiplied to 10 MHz to 60 MHz whereas the output of PLL1 is fixed at 48 MHz for USB clock

The value to be loaded into PLL0CFG, to get CCLK of 60MHz using 12MHz Crystal Frequency (FOSC), is **0x24.**

The clock signal from the PLL is given to CPU as its clock. The PLL signal must also be supplied to other on – chip peripherals. By default, the peripheral clock (PCLK) runs at a quarter speed of the CCLK (i.e 15MHz if CCLK is 60MHz). The relation between the CPU clock and peripheral clock can be configured by using a special register called “VPBDIV” (by default VPBDIV = 0x00).

(for more information on PLL- <https://www.electronicshub.org/arm-pll-tutorial/> )

[pclk = cclk / 4, cclk = 60MHz, for FOSC = 12MHz (default values)];

LPC2148 generates the baud rate depending on the values of DLM, DLL (Divisor Latch MSB & LSB bytes)

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

for DivAddVal/MulVal == 0

Using the above parameters , DLL/DLM is calculated as below.  
**(256 \* DLL + DLM) = PCLK / (16\* Baudrate).**

Example Program to compute DLL & DLM for given baudrate:

unsigned long var\_RegValue\_u32 = ( pclk / (16 \* baudrate ));

U0DLL = var\_RegValue\_u32 & 0xFF;

U0DLM = (var\_RegValue\_u32 >> 0x08) & 0xFF;

**Steps for Configuring UART0**

Below are the steps for configuring the UART0.

1. Configure the GPIO pin for UART0 function using PINSEL register.
2. Configure the FCR for enabling the FIXO and Reste 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.
5. Update the DLM,DLL with the calculated values.
6. 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. Use the following registers to receive/transmit the data.

|  |  |
| --- | --- |
| UxRBR | Contains the recently received Data |
| UxTHR | Contains the data to be transmitted |

**Example Program to transmit the string to PC**

#include <lpc214x.h>

int main()

{

unsigned char i=0,ch,msg[]={"RVCE-CSE"};

**//Configure P0.0/P0.1 as RX0 and TX0**

PINSEL0 = (1 << 0)|(1 << 2); //P0.0-TxD0,P0.1-RxD0,2ndOption:01

**// Enable FIFO and reset Rx/Tx FIFO buffers**

U0FCR = (1<<0) | (1<<1) | (1<<2);

**// 8bit data, 1Stop bit, No parity**

U0LCR = (0x03<<0) | (1<<7);

//**PCLK = CCLK / 4, CCLK = 60MHz, for FOSC = 12MHz (default)**

U0DLL = 97;

U0DLM = 0;

**// Clear DLAB after setting DLL,DLM**

U0LCR = (0x03 << 0);

// **Program to send the string**

while((ch=msg[i++])!= '\0')

{

**// Wait for Previous transmission**

while((U0LSR & (0x01<<5))== 0x00);

**// Load the data to be transmitted**

U0THR= ch;

}

while(1);

}

**VECTORED INTERRUPT CONTROLLER (VIC) & Writing ISRs**

VIC present inside the LPC 2148, manages all the interrupts generated from the ARM core, peripherals present inside the IC and also external interrupts. Each peripheral device has one interrupt line connected to the VIC (but inside the peripheral, several interrupt flags are present representing the different events for the interrupt generation).

VIC receives total 32 interrupt requests from different sources like UART, PWM unit, Timers, USB, EINT0-3 etc and categorizes them as,

1. FIQ (Fast interrupt request, has the highest priority, serviced first by FIQ handler. If more than one FIQ is occurred, FIQ handler must read the contents of FIQ status register to check the source and initiate the corresponding service routine)
2. IRQ –Medium priority, Vectored interrupt request ( 16 of the total 32 interrupts can be assigned to IRQ, referred as IRQ slots, 0 to 15, with slot0 having the highest priority, every slot is provided with two registers, namely VICVectAddr and VICVectCntl to specify the slots interrupt vector and enable the slot, since it is vectored. Programmer has to map the peripheral i.e interrupt source to the slot, using the above registers.
3. Non-vectored IRQ – Lower priority ( interrupt vector, in this case is common for all non-vectored IRQs, and is to loaded into the register VICDefVectAddr )

The VIC generates FIQ signal combining all the FIQ interrupt requests and combines the requests from all the vectored and non-vectored IRQ’s to produce IRQ signal to the ARM processor.

FIQ is serviced first, if no FIQ then IRQ related to Vectored interrupts is serviced, based on the priority assignment (IRQ0 highest, IRQ15 lowest), if no vectored interrupt request, then non-vectored IRQ is generated.

VICIntEnable : register used to Enable/ disable interrupts coming from different sources

Ex: to enable timer0 interrupt - VICIntEnable = 0x0000 0010 ; bit D4 corresponds to Timer0

VICIntSelect : register used to classify interrupt as FIQ or IRQ , by default all are IRQ

VICVectCntl0-15 : Vector Control registers used to map peripheral to IRQ slot and enable it, let us map timer0 to slot 4 VICCntl4 = 0x0000 0024

(D31…………………D5 .D4 D3 D2 D1 D0) D4 to D0 used to indicate the peripheral,

1 0 0 1 0 0 D5 to enable the channel

Note: every peripheral is given some position, in the list of 32 source of interrupts, for Timer0 it is 4, refer data sheet.

VICVectAddr0 – 15: Vector Address Registers, used to hold interrupt vector corresponding to slot,

VICVectAddr4 = (unsigned long) Timer0\_isr; // Timer0\_isr is the isr function

**Example :** Using timer0 and interrupts (ISR) generate the waveform on P1.16

#include <LPC2148x.h>

unsigned int x=0;

\_\_irq void Timer0\_ISR(void)

{

x = x ^ 1;

if (x)

I0SET1 = 1 << 16; //P1.16 = 1

else

I0CLR1 = 1 <<16; // P1.16 = 0

T0IR = 0x01; // clear match0 interrupt, and get ready for the next interrupt

VICVectAddr = 0x00000000 ; //End of interrupt

}

int main(void)

{

I0DIR1 = 0x0001 0000; //set P1.16 as output

//disable timer and load timer and interrupt related reg values

T0TCR = 0x00;

T0MCR= 0x0003; // Enable Interrupt and reset timer after match

T0TC = 0xFF; // some count is loaded

//load interrupt related registers , assigning Timer0 to IRQ slot 4

VICVectAdd4 = (unsigned long)Timer0\_ISR; // set the timer ISR vector address

VICVectCntl4 = 0x0000024; // set the channel

VICIntEnable = 0x0000010; // enable the timer0 interrupt

//start the timer

T0TCR = 0x01;

while(1)

{

//do other works

}; // now timer interrupt is serviced automatically using the ISR

}

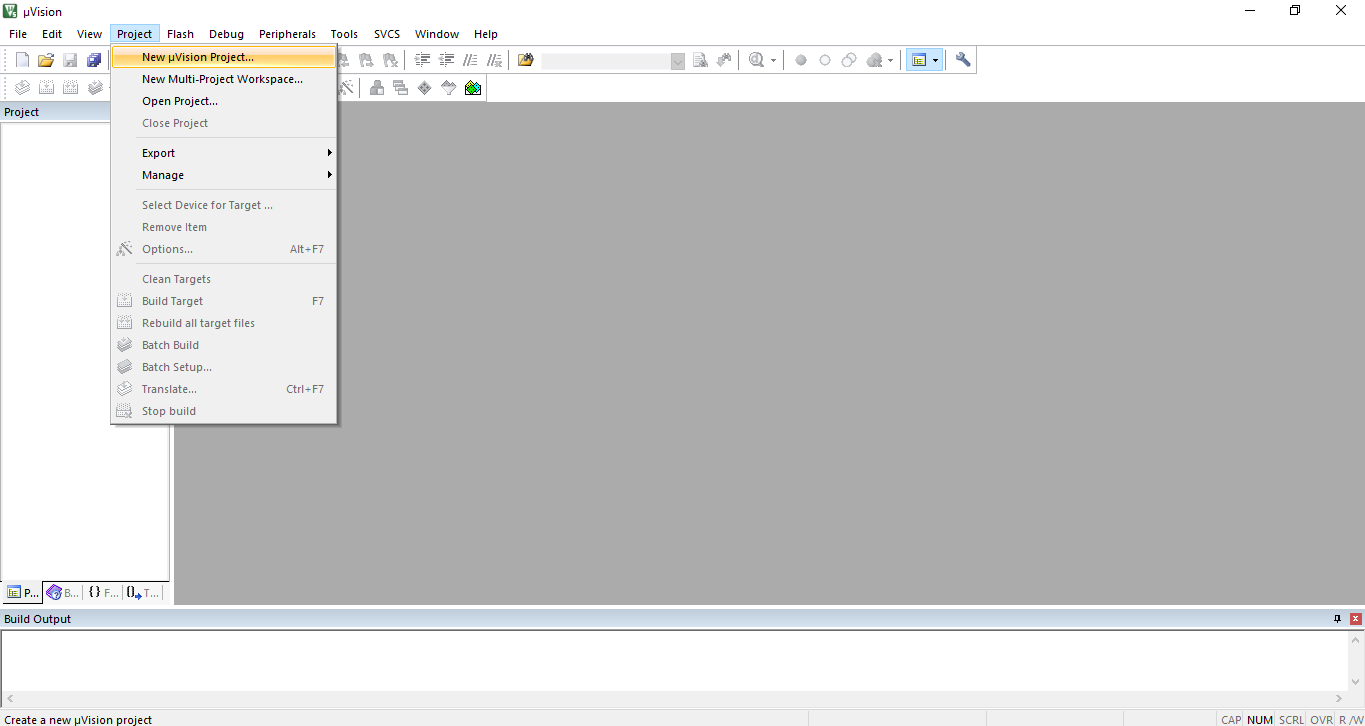
**Appendix 2 : Creating Keil Project**

**INSTALLING KEIL VERSION 5**

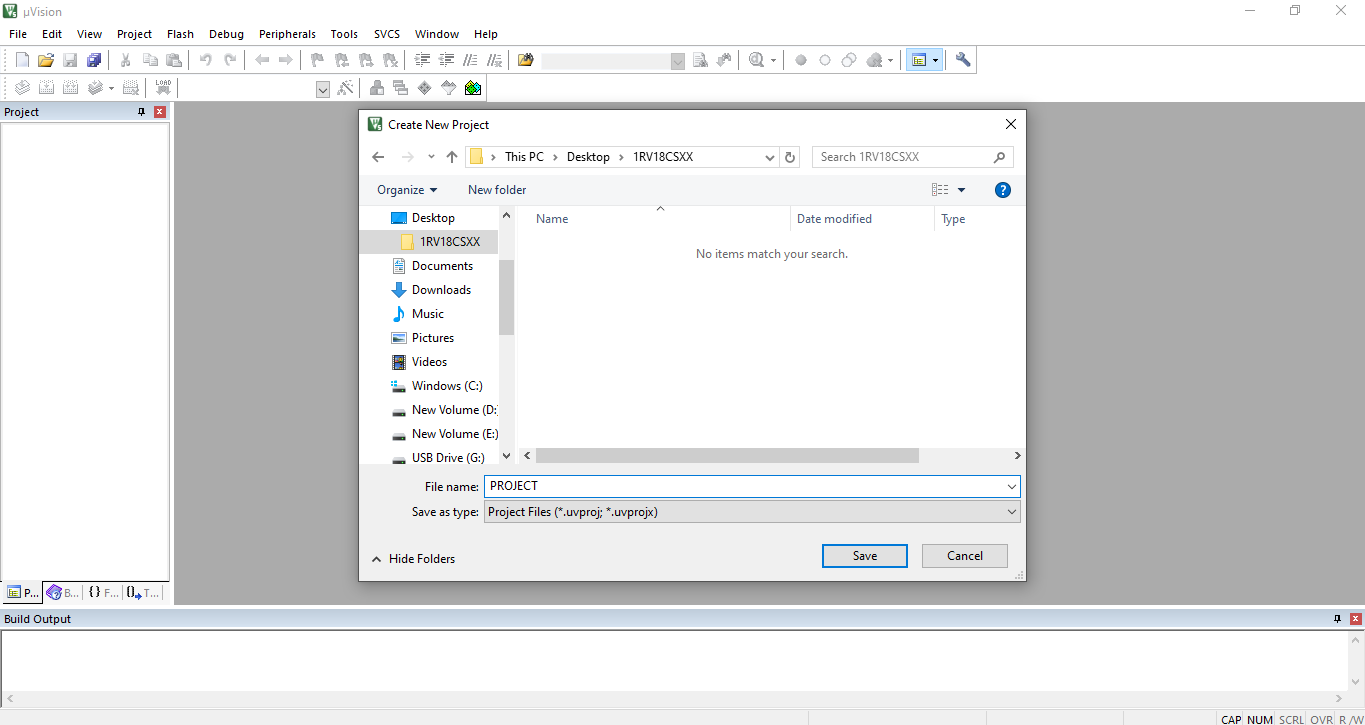
1. **First install mdk528a and then install the patch MDK79525 (these files can be downloaded form keil.com website). Mdk528a is an ARM compiler and MDK79525 is required to compile for LPC2148.**
2. **Keil microvision 5 is an IDE supporting editor, assembler, compiler, simulator and debugger.**

**CREATING A PROJECT**

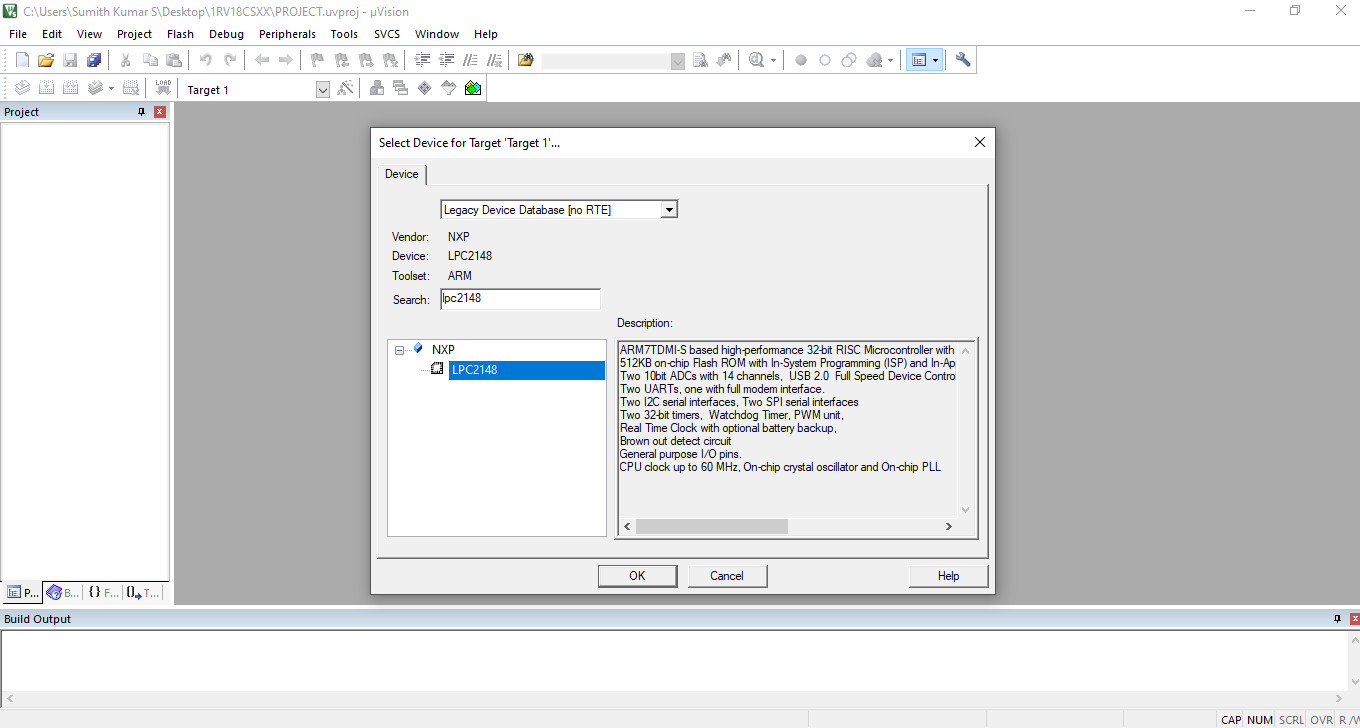
* **Create a folder with your USN on the Desktop.**
* **Open Keil microVision 5.**
* **If there are any projects already open, go to projects tab on the top and close it.**

****

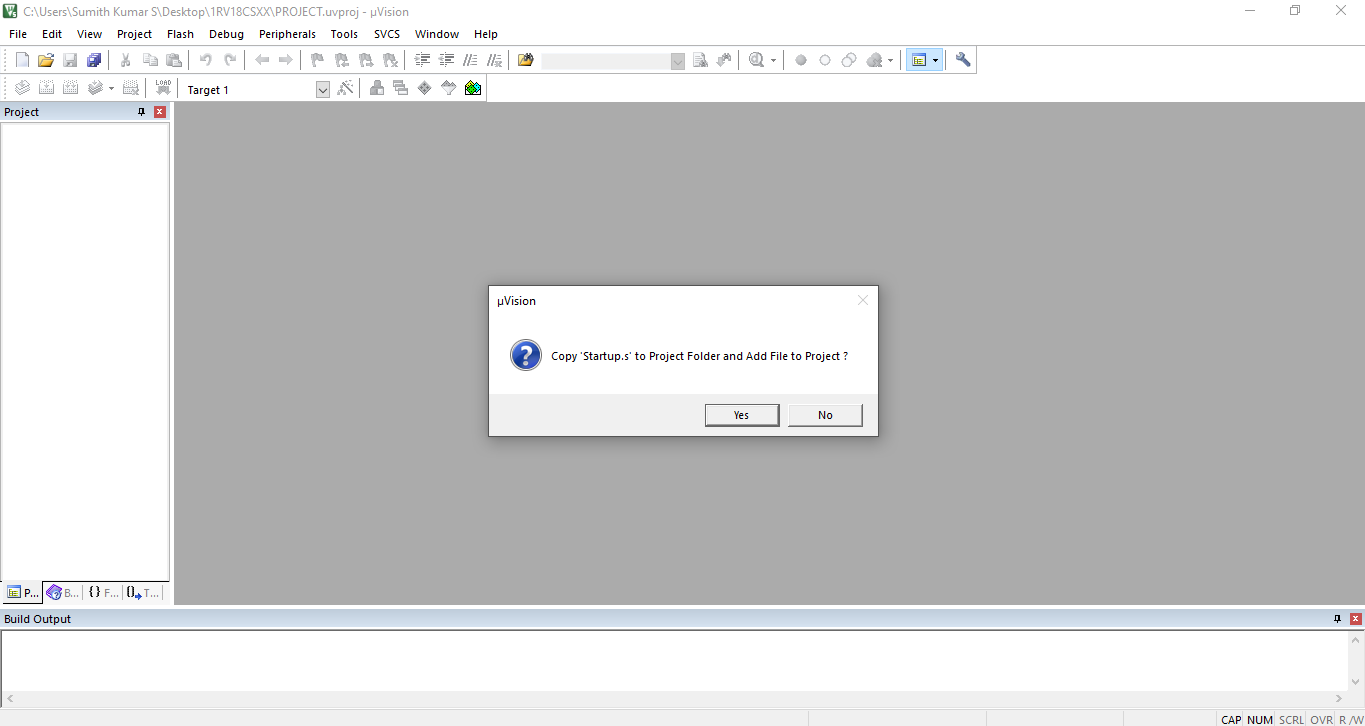
* **Go to Project->new microvision Project.**

****

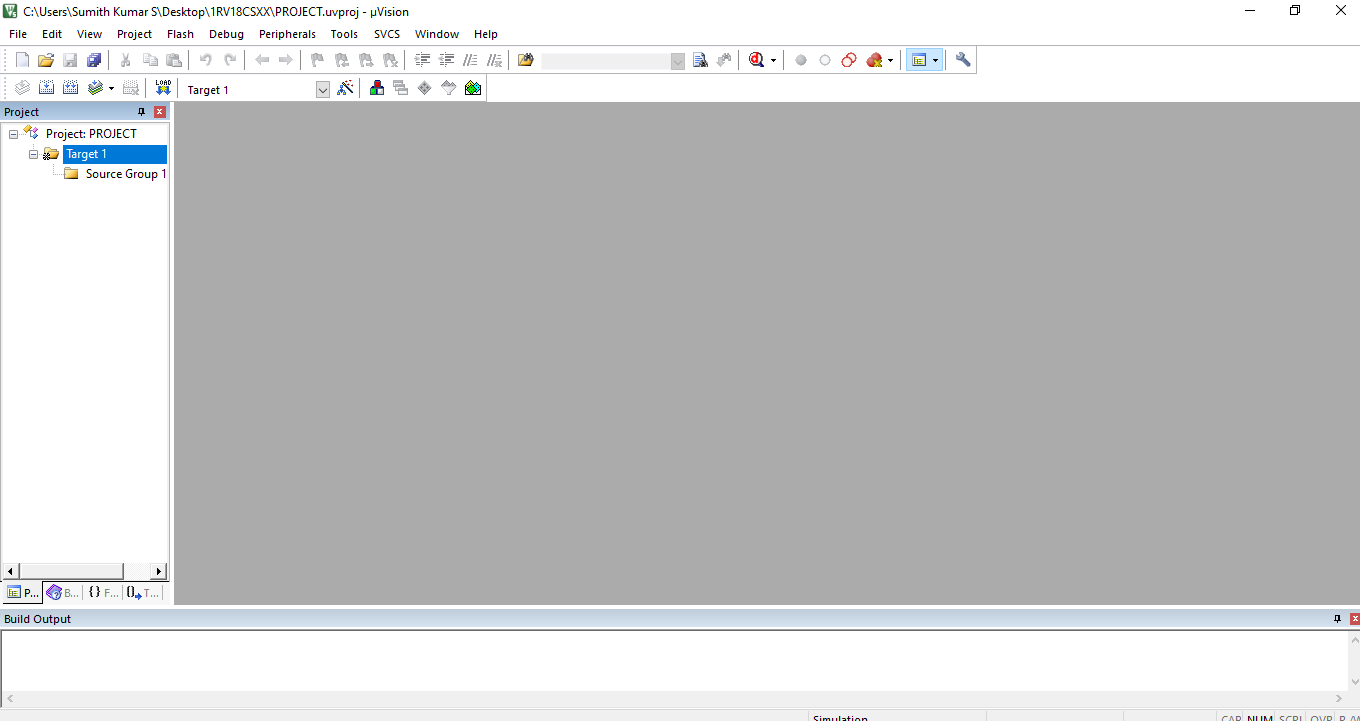
* **Give your project a name and save it in the folder created earlier.**

****

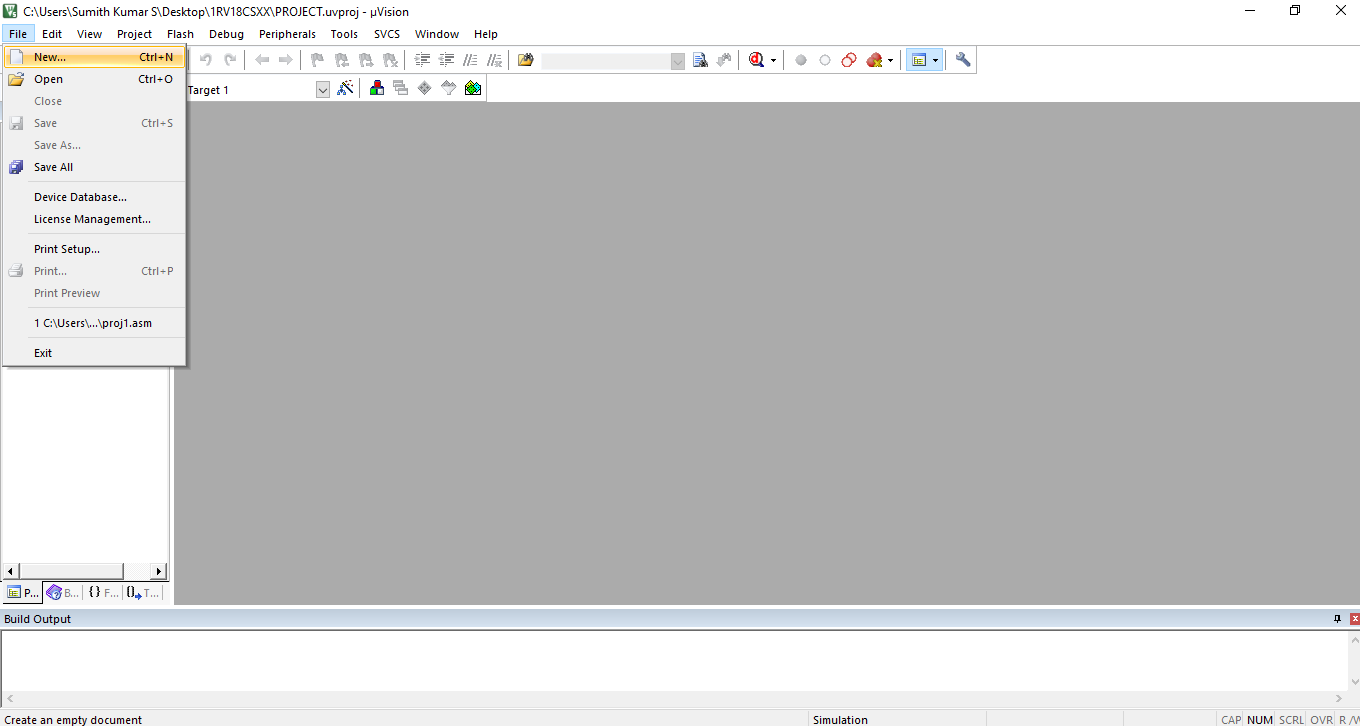
* **Select Device as ‘Legacy Device Database’ and search for ‘LPC2148’ and select it.**

****

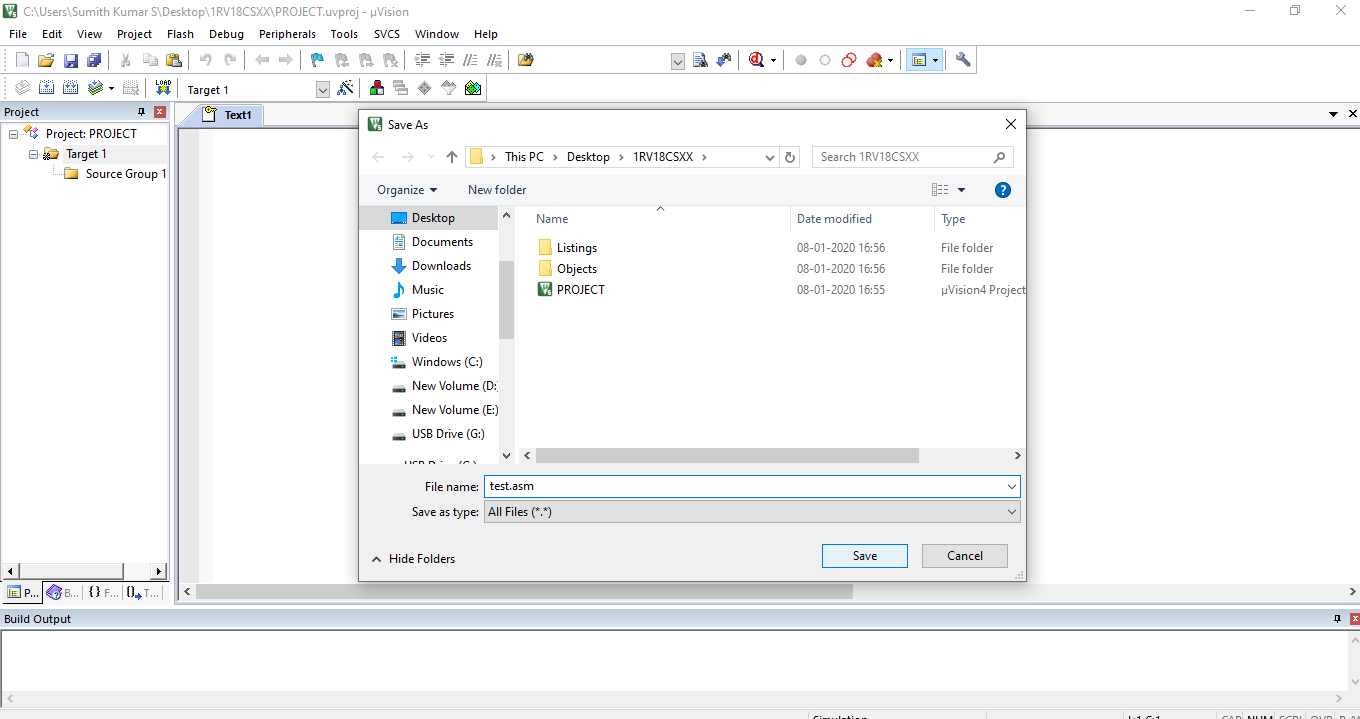
* **For the above window select No and proceed. For C project, say Yes.**

****

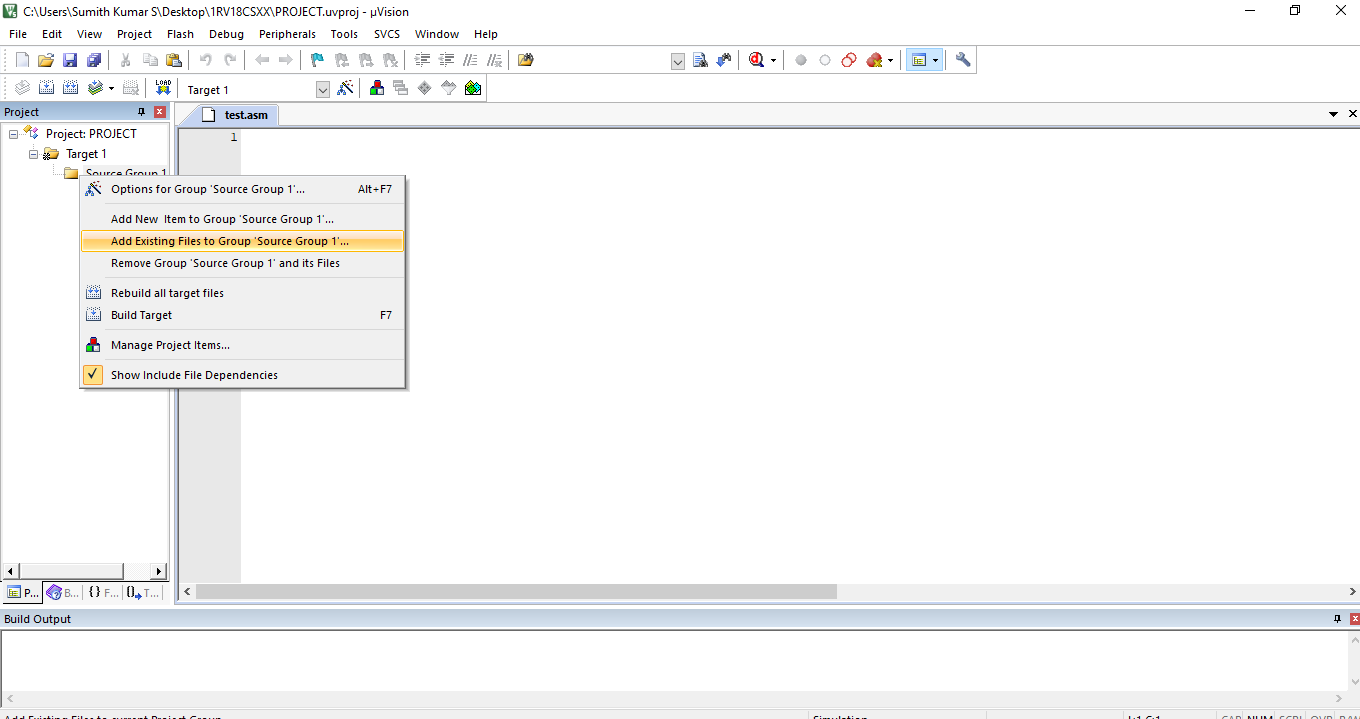
**We can see that the project is created**

****

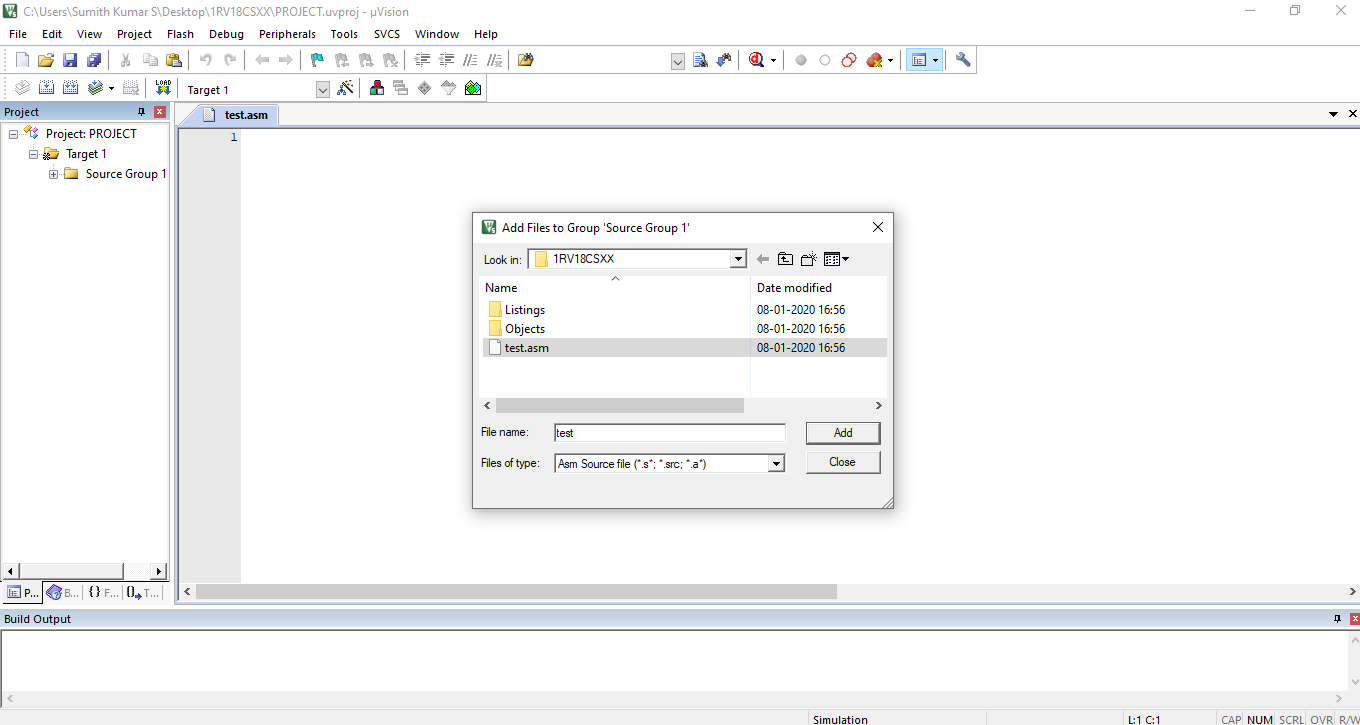
* **Click on File->New and create a file.**

****

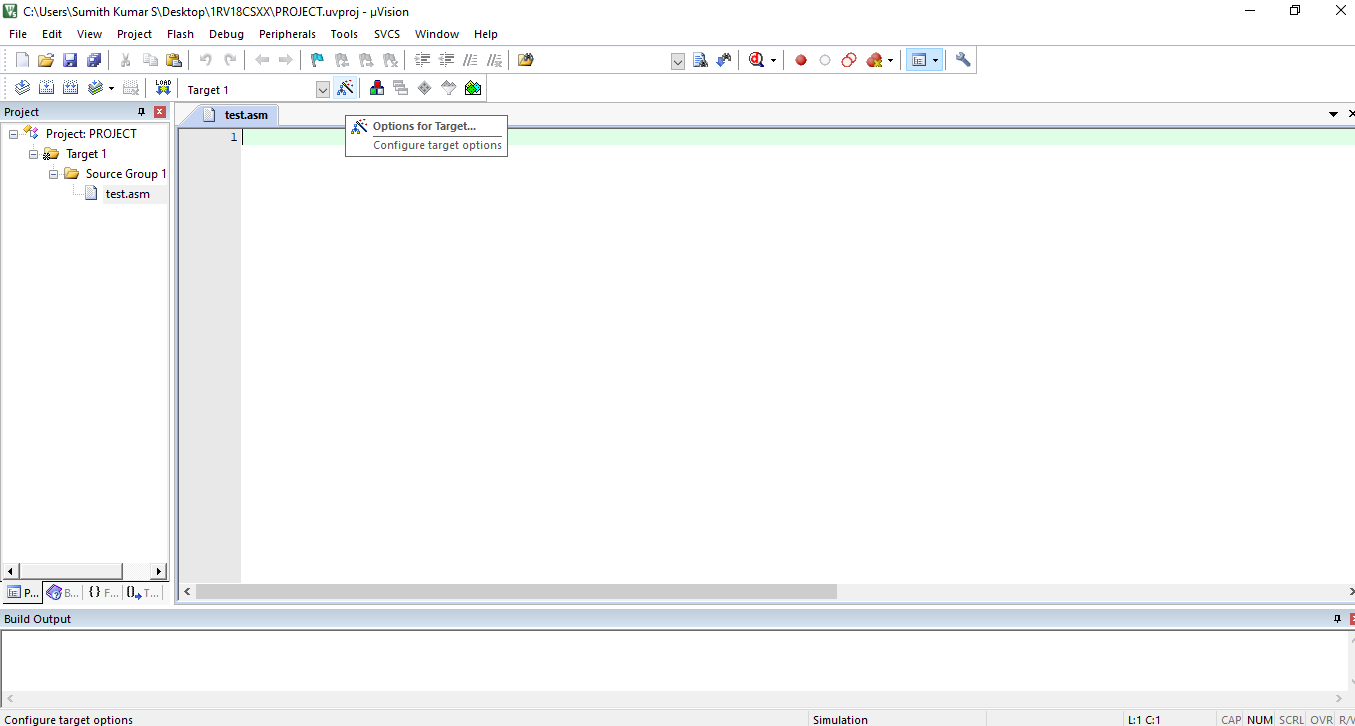
* **Immediately save the file created with .asm extension.**

****

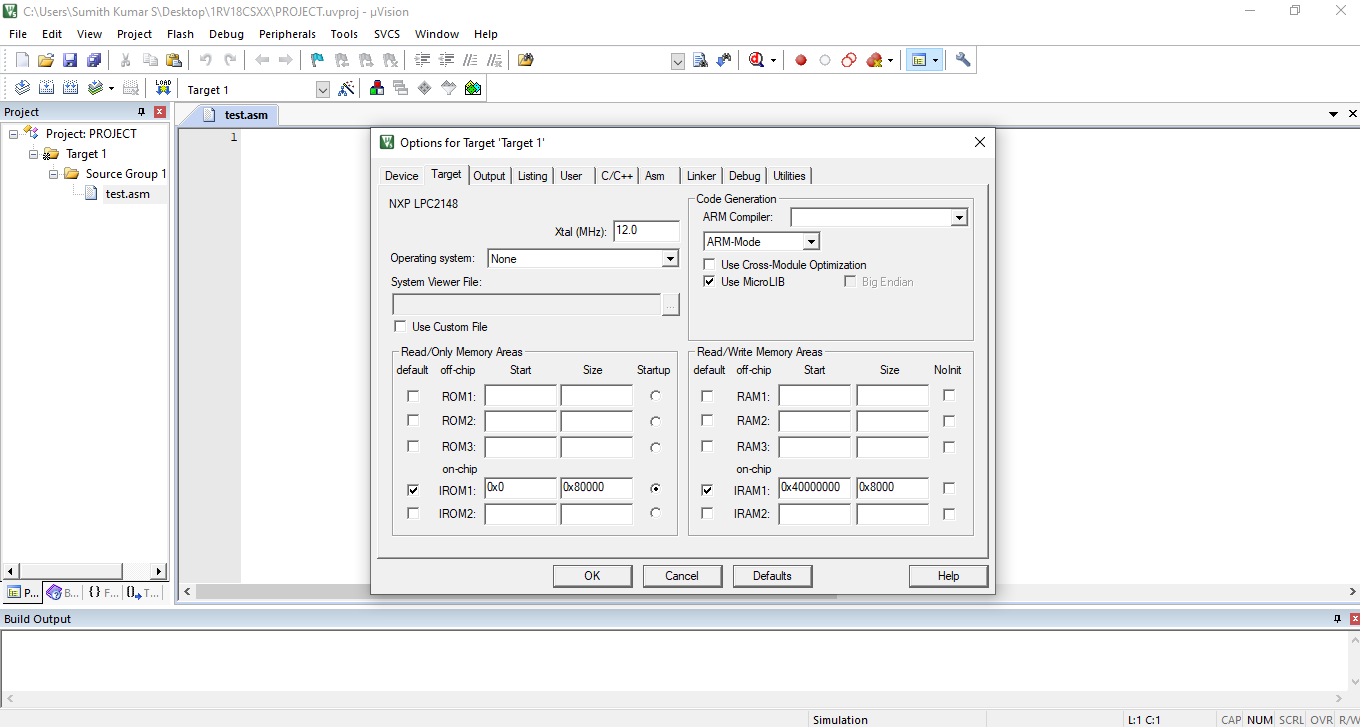
* **Right click on ‘Source Group’ and select ‘Add Existing Files to Group’.**

****

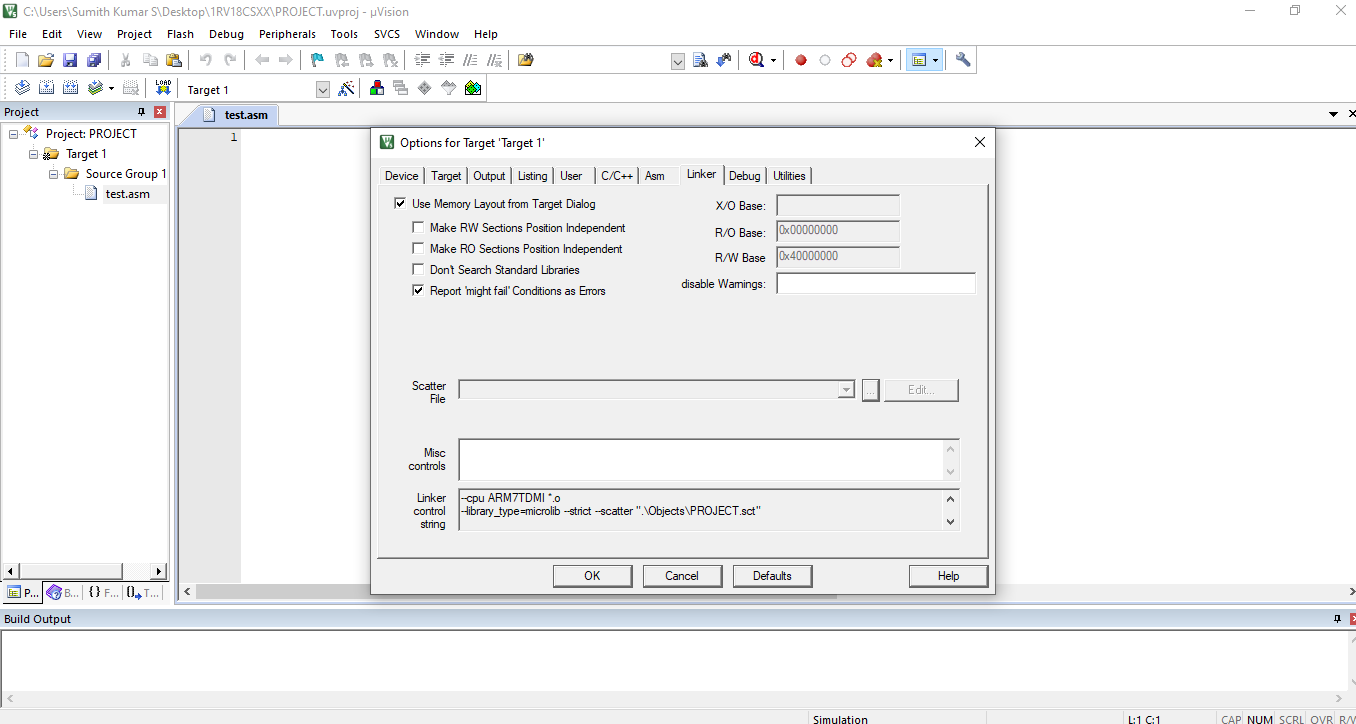
* **See that the ‘Files of type’ is Asm Source file and select the file which was created earlier.**

****

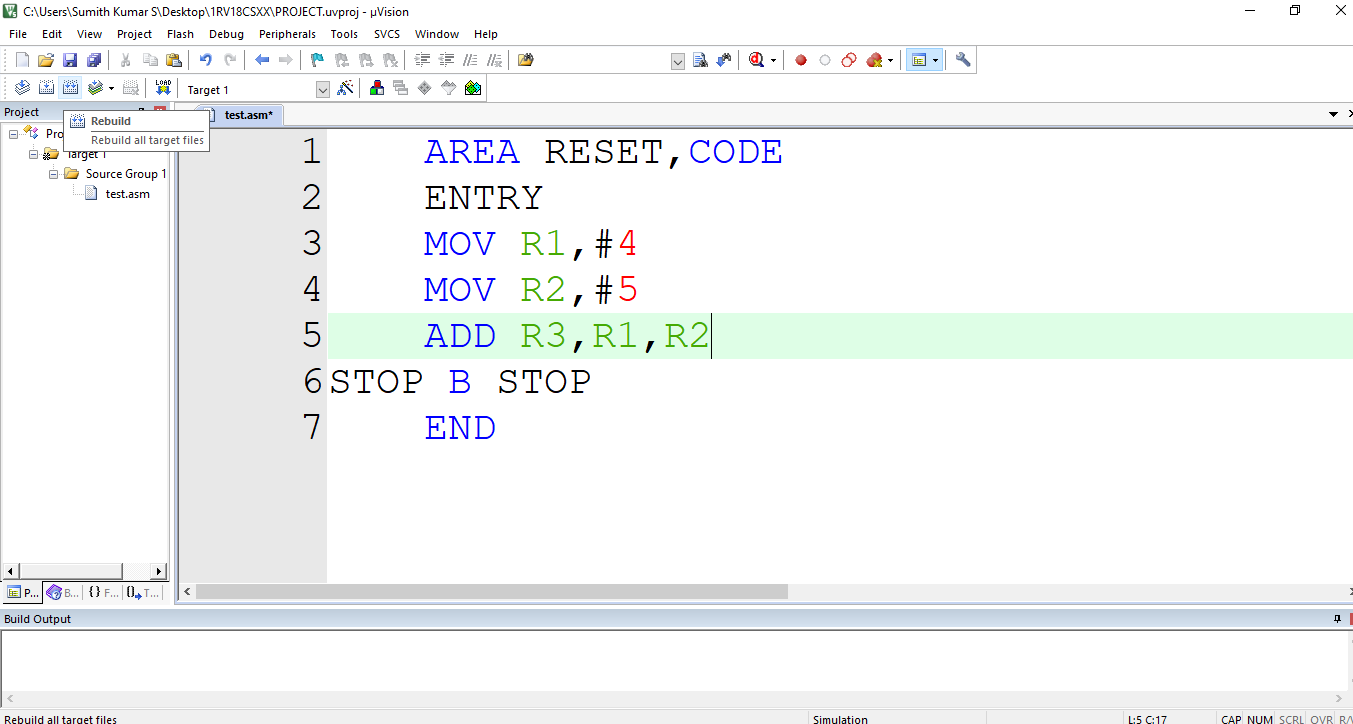
* **We can see that the file test.asm is added to the ‘Source Group’.**
* **Next click on ‘Options for target’ as shown above.**

****

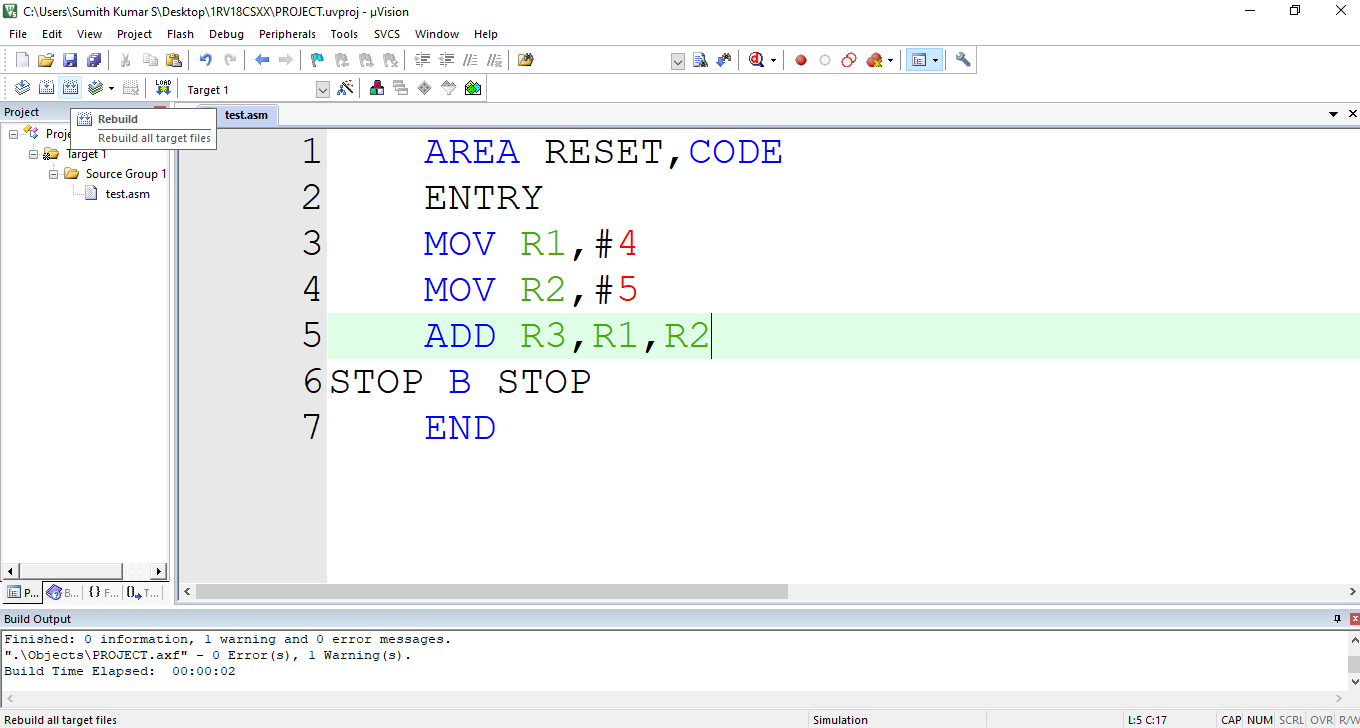
* **Under the Target section, check the Use MicroLIB which is present in ‘Code Generation’.**

****

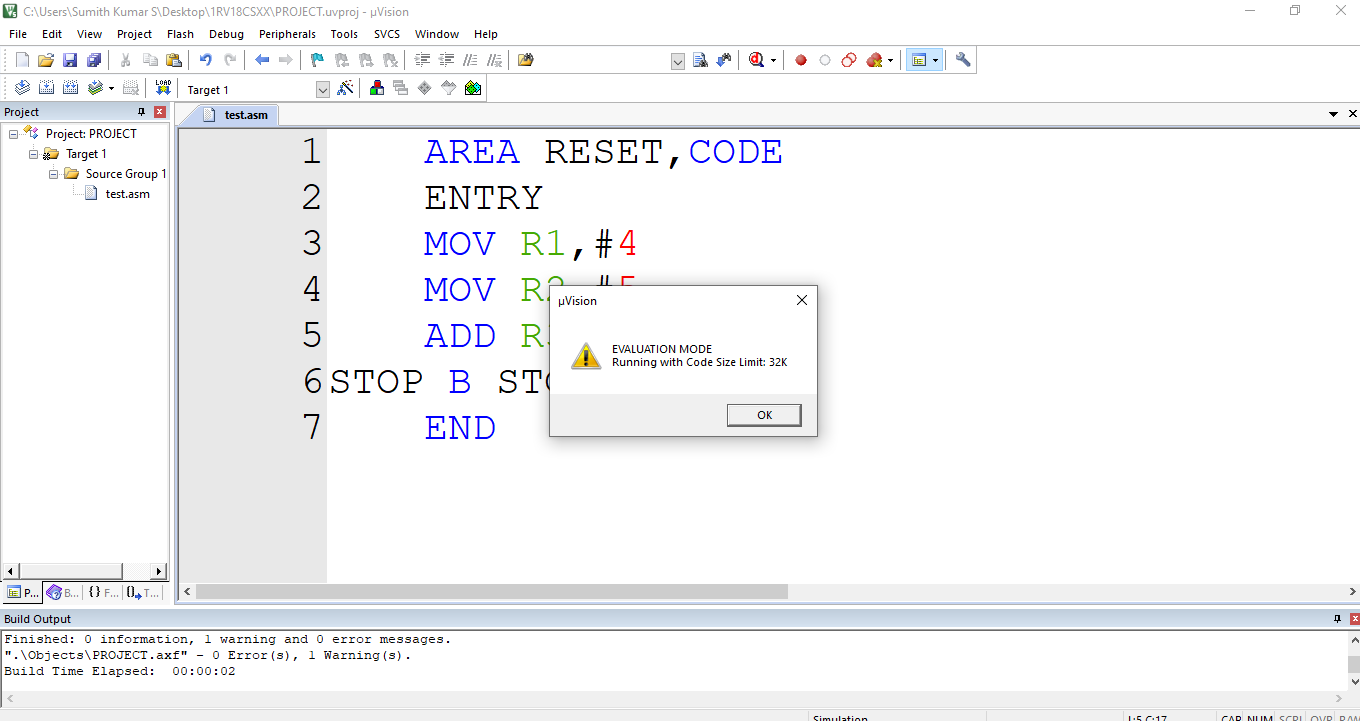
* **Under the Linker section, check the ‘Use Memory Layout from Target Dialog ‘ and click on OK and close it.**

****

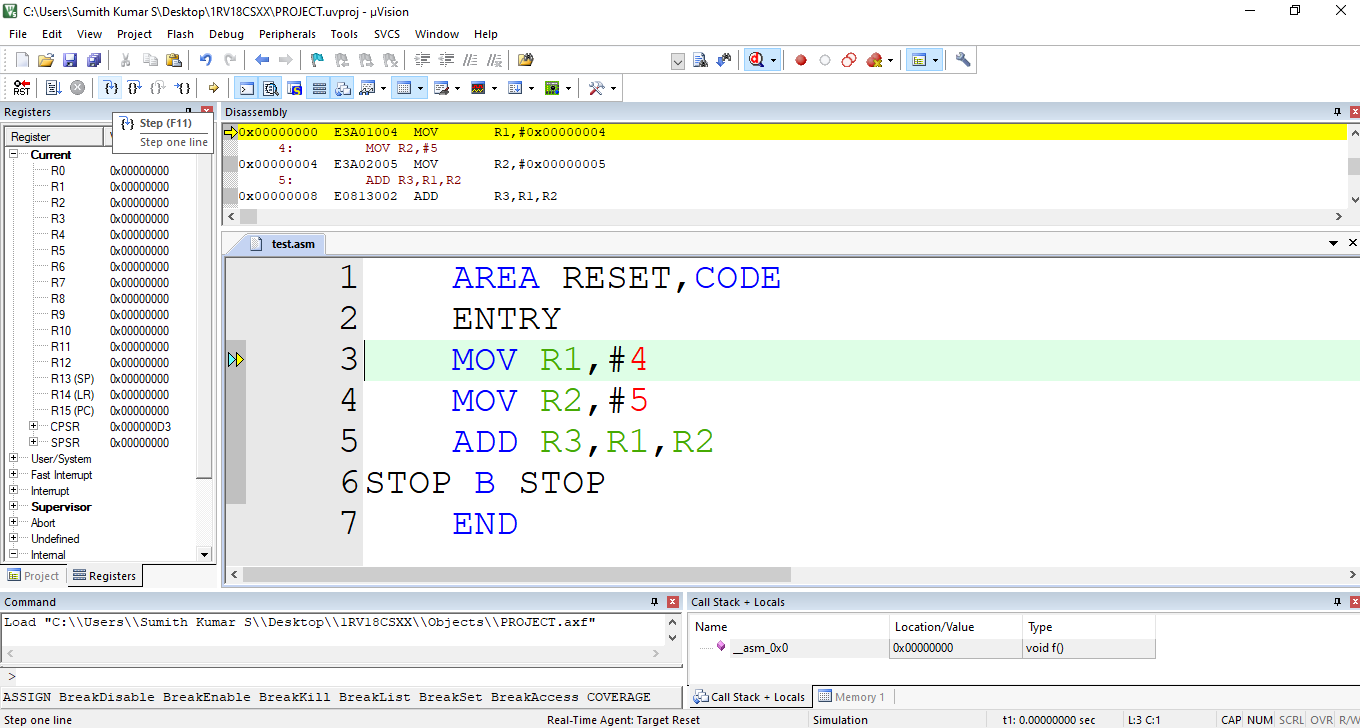
* **Now we can start writing Assembly Language Programs.**
* **Above is a sample program to add 2 integers and store the sum in register R3.**

****

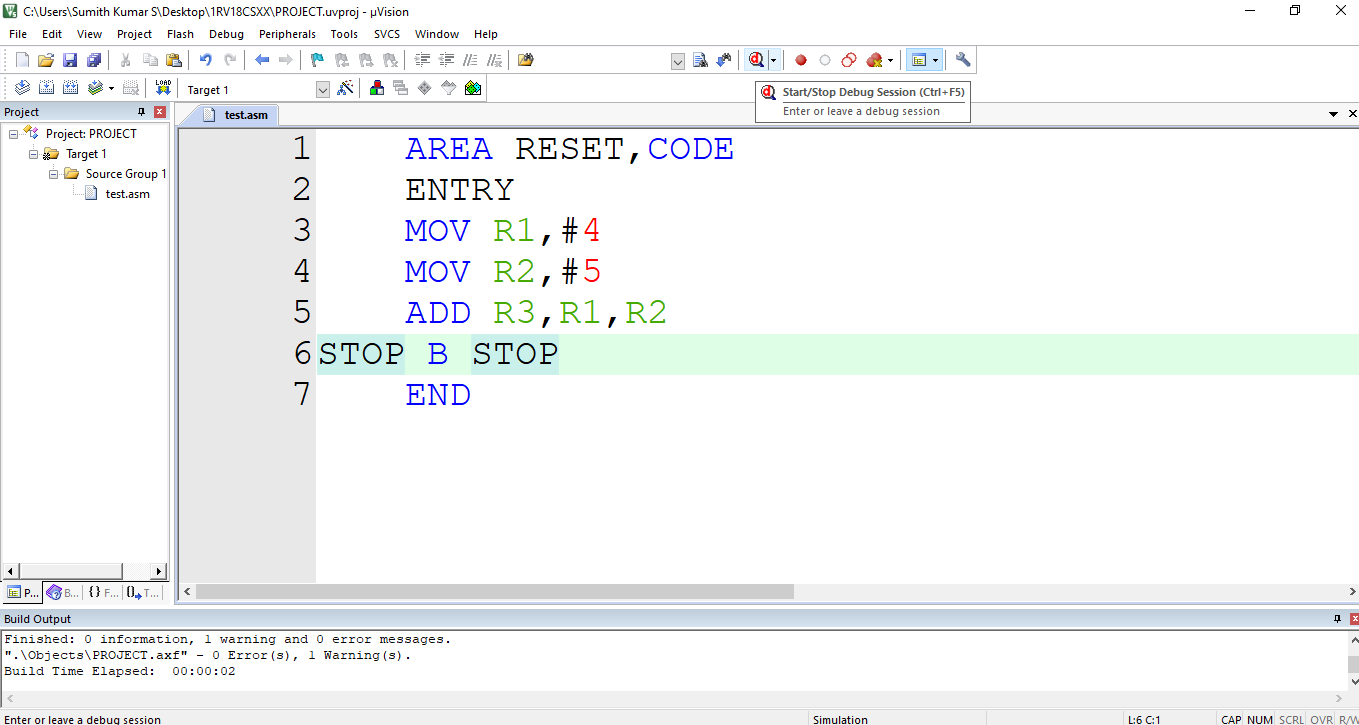
* **After write the ALP, in order to build it, select the build option as shown above. We can build only one file or all the files at once. Make sure that, one ASM file with ENTRY is only active in the project, else remove other files and keep one.**

****

* **We can execute programs of size only less than 32K. Select OK for the above window.**

****

* **To run program step by step select the option shown above and observe the content of the registers.**

****

* **If you need to make any change to the code, select the debugger and modify the code and build the code and then execute.**