|  |  |
| --- | --- |
| **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**



**IOT & Embedded Systems Lab**

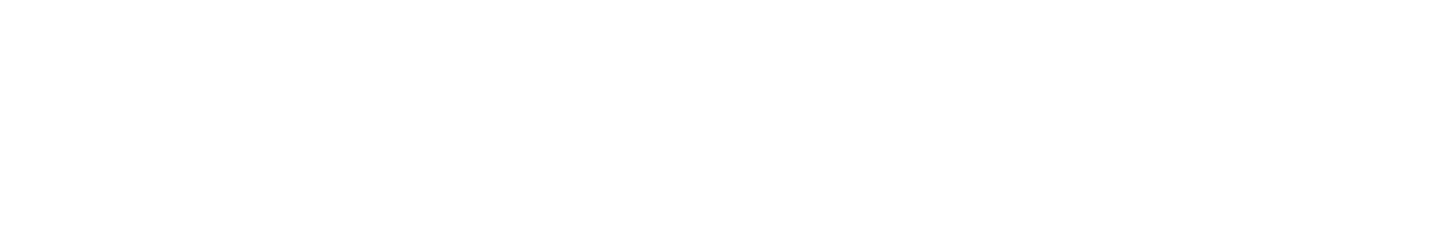
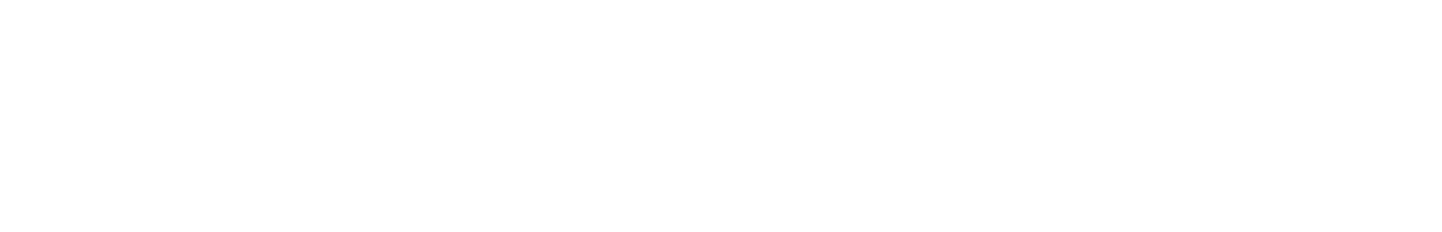
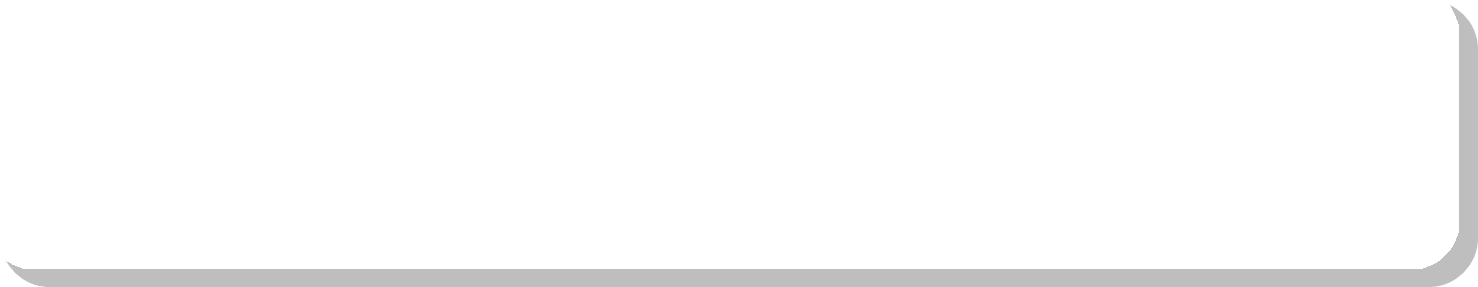
**Subject Code: CS344AI**

**IV SEMESTER B.E.**

**LABORATORY RECORD**

**[Autonomous Scheme 2022]**

**2023-2024**



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

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

#### Rashtreeya Sikshana Samithi Trust

**R. V. COLLEGE OF ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



**IOT & EMBEDDED SYSTEMS LAB**

**(CS344AI)**

**2023 - 2024**

**IV SEMESTER B.E LABORATORY RECORD**

**[Autonomous Scheme 2022]**

**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 IOT and Embedded Systems Lab [CS344AI] prescribed by the department during the year 2023-2024.

|  |  |
| --- | --- |
| **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** |
| 1 | Interface Logic Controller and write Embedded C programs to generate BCD up / down and Ring counters. Input is read from the DIP switch. | 31 |  |  |  |  |  |  |
| 2 | 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 |  |  |  |  |  |  |
| 3 | 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 |  |  |  |  |  |  |
| 4 | DAC Interface : Write an Embedded C program to generate sine , full rectified ,triangular, sawtooth and square waveforms using DAC module | 42 |  |  |  |  |  |  |
| 5 | 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 |  |  |  |  |  |  |
| 6 | 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 |  |  |  |  |  |  |
| 7 | Character LCD Interface : Write an Embedded C program to display text messages on the multiple lines of the display. | 57 |  |  |  |  |  |  |
| **Total for 70 Marks** | | | | | | | |  |

|  |  |
| --- | --- |
| **LAB INTERNALS** | |
| **RECORD Marks** | / 20 Marks |
| **Mini Project** | / 20 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 ARM concepts.  **(2M)** | Student has sufficient understanding of program requirements and applies ALP / Embedded C for ARM concepts. **(1.5M - 1M)** | Student does not have clear understanding of program requirements and is unable to apply ALP / Embedded C for ARM 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:** | |  | | | |

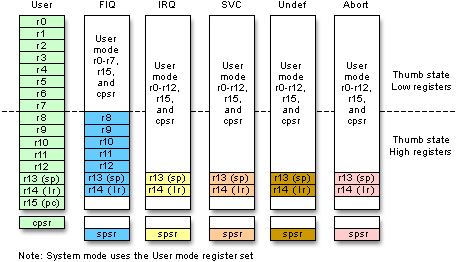
**ARM Assembly Language Programs – Part A**

**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]

**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.

##### Sample Program 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

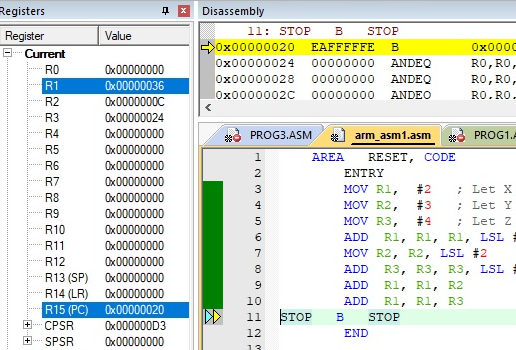
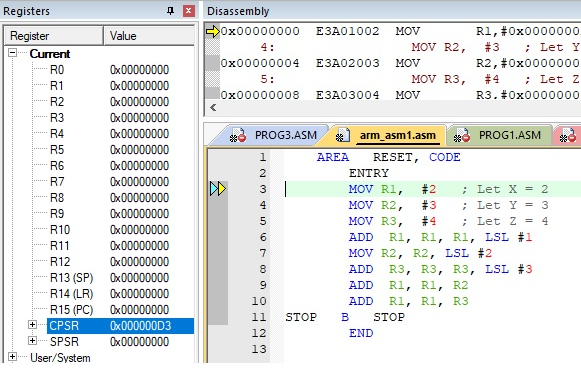
ADD R1, R1, R3

STOP B STOP

END

**Sample Output:**

Before Execution After Execution

****

##### Sample Program 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 R5,R1,R3

ADC R6,R2,R4

LDR R0,=RESULT

STR R5,[R0]

STR R6,[R0,#4]

STOP S STOP

VALUE1 **DCD** &BBBBBBBB,&AAAAAAAA

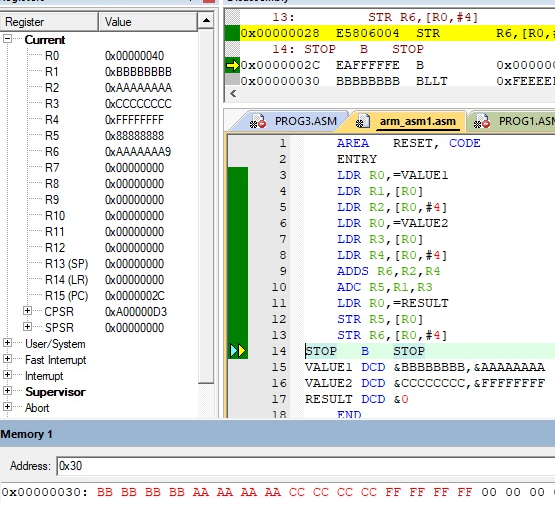
VALUE2 **DCD**  &CCCCCCCC,&FFFFFFFF

**AREA MEMORY, DATA**

**RESULT SPACE 4**

**END**

**Sample Output:**



##### Sample Program 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

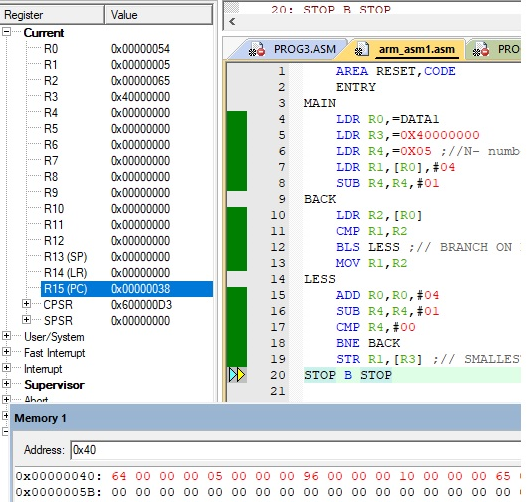
STOP B STOP

**AREA DATA,CODE**

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

**END**

**Sample Output:**



##### Sample Program 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**

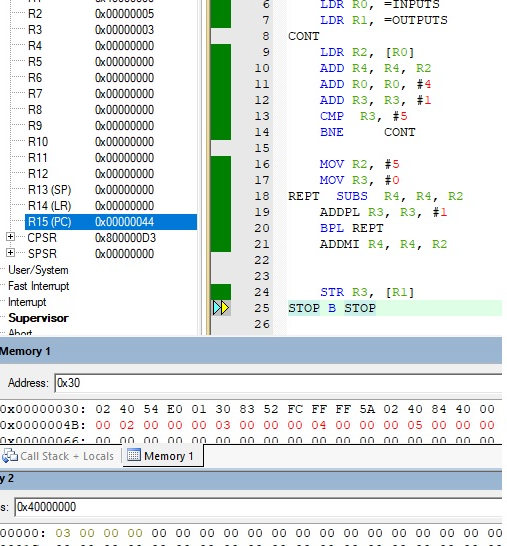
**INPUTS DCD 01,02,03,04,05**

**AREA MEMORY, DATA**

**OUTPUTS SPACE 4**

**END**

**Sample Output:**



##### Sample Program 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

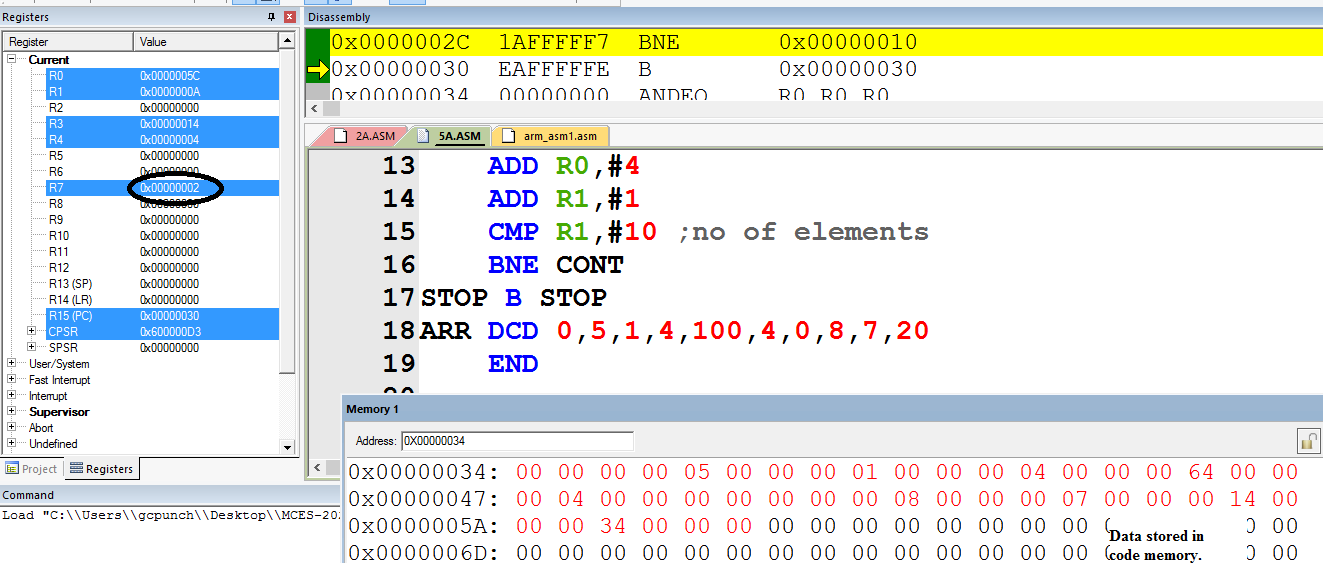
BNE CONT

**STOP B STOP**

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

**END**

**Sample Output:**



##### Sample Program 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

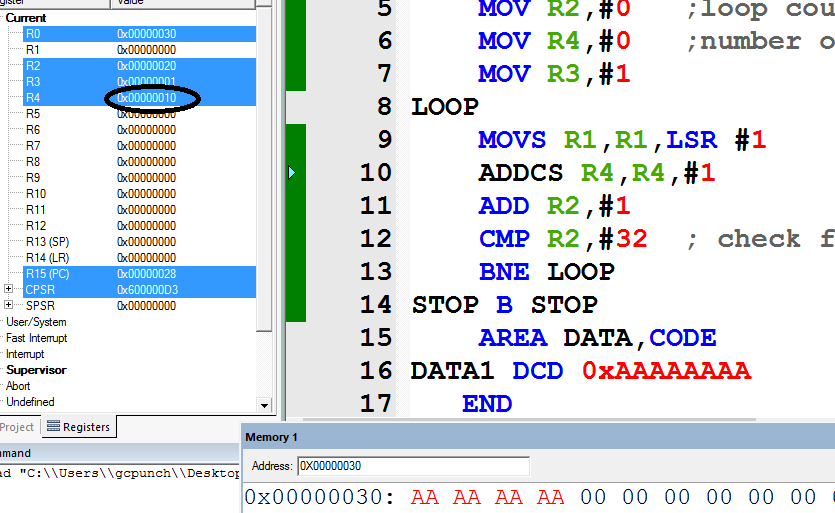
STOP B STOP

AREA DATA,CODE

DATA1 DCD 0xAAAAAAAA

END

**Sample Output:**



##### Sample Program 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

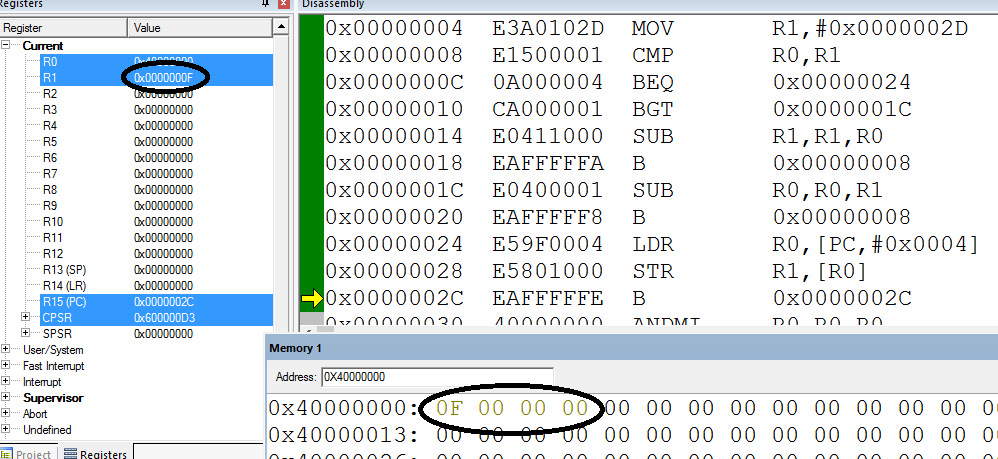
STR R1,[R0]

STOP B STOP

AREA RESULT,DATA

GCD SPACE 4

END

**Sample Output:**

##### Sample Program 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

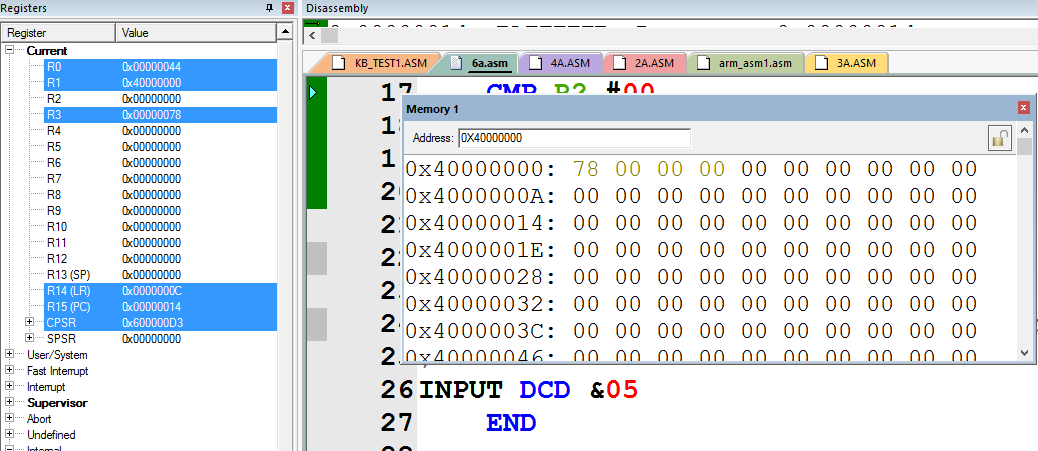
MOV PC,LR ; return from subroutine

;subroutine-end

INPUT DCD &05

END

**Sample Output:**



##### Sample Program 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

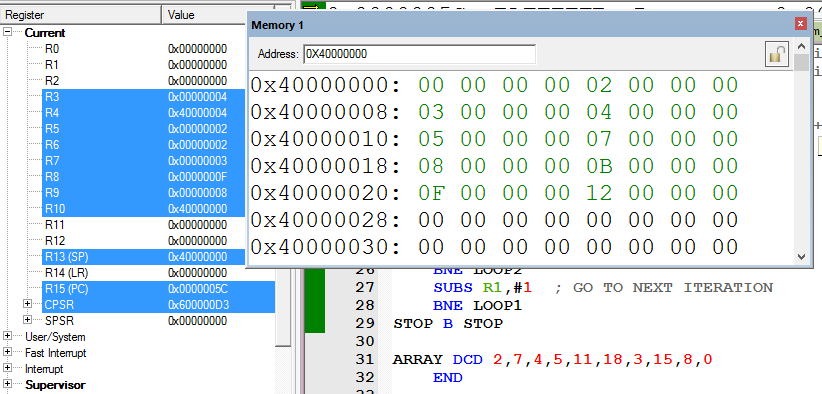
BNE LOOP1

STOP B STOP

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

END

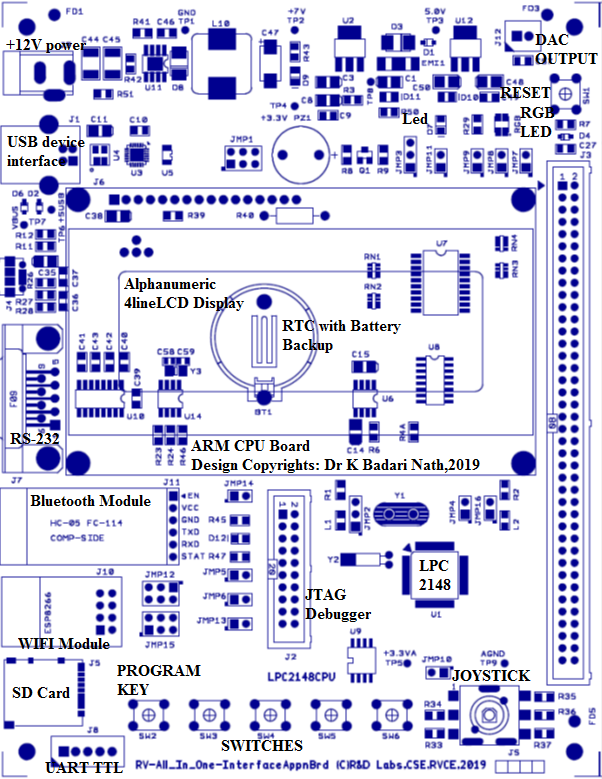
**Sample Output:**



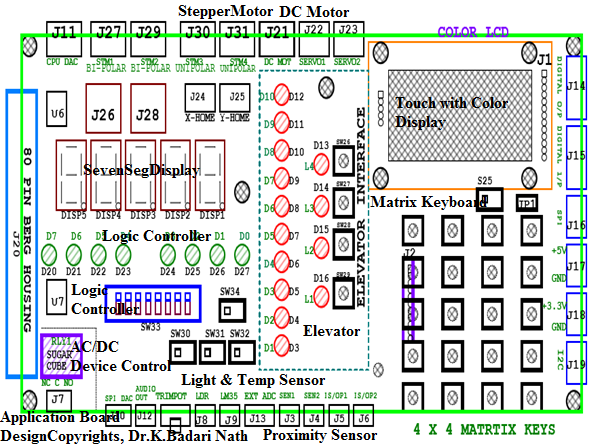
PART – A

Interfacing Programs using ARM LPC 2148

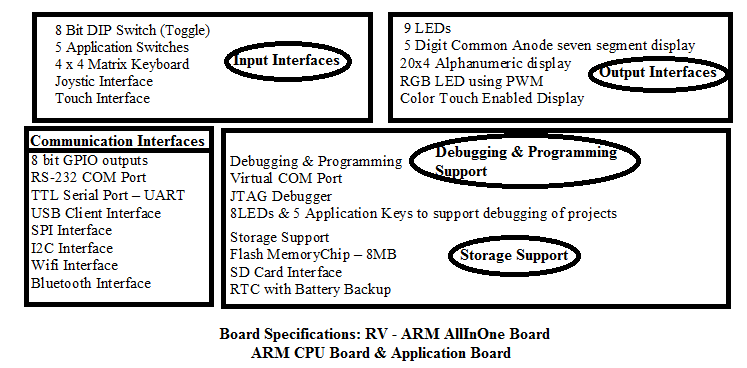
**ARM CPU Board**



**ARM Application Board**

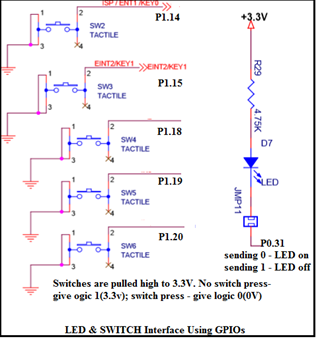


Board Specifications: Power : 12V, 1A



**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++)

{

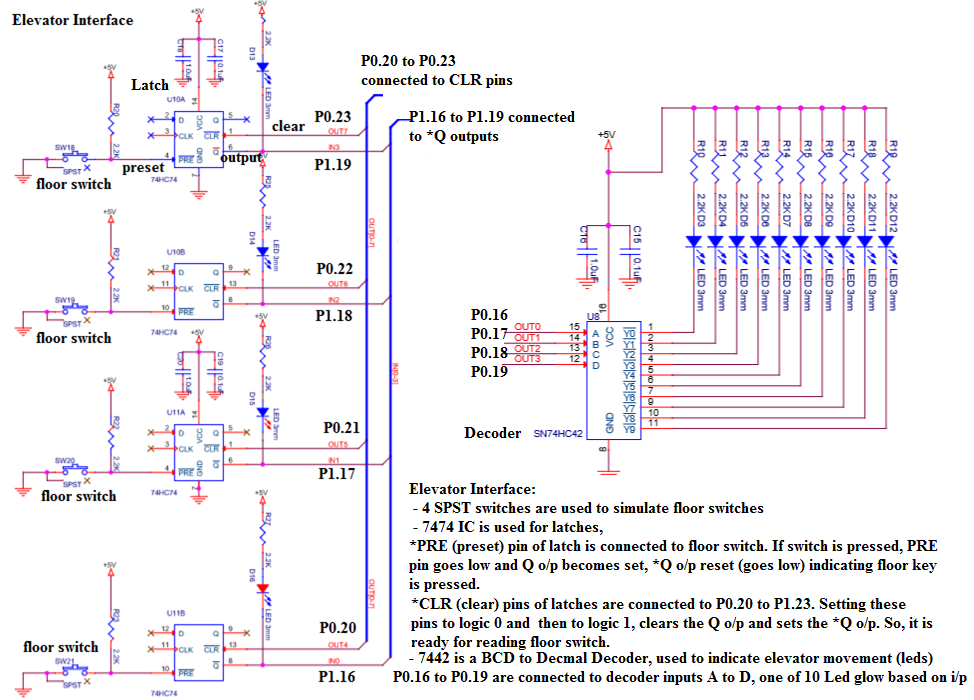
for(x=0; x<10000; x++); */\* loop to generate 1 milisecond delay with CCLK = 60MHz \*/*

}

}

**Program 1: Interface Logic Controller and write Embedded C programs to generate BCD up / down and Ring counters. Input is read from the DIP switch**.

**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 IS\_ON(pin) (IO1PIN & (1U << (pin)))

void delay\_ms(unsigned int x);

void reset\_values(int y);

int contUP = 0;

int contDN = 99;

unsigned int rightSFT = 1U<<7;

unsigned int leftSFT = 1;

const int key0 = 16;

const int key1 = 17;

const int key2 = 18;

const int key3 = 19;

int main()

{

IO0DIR |= 0xFF;

while(1)

{

if(IS\_ON(key0))

{

reset\_values(0);

IO0CLR = 0xFF<<16;

IO0SET |= ((contUP/10)<<4 | (contUP%10))<<16;

contUP++;

if(contUP > 99) contUP = 0;

}

else if(IS\_ON(key1))

{

reset\_values(1);

IO0CLR = 0xFF<<16;

IO0SET |= ((contDN/10)<<4 | (contDN%10)) << 16;

contDN--;

if(contDN < 0) contDN = 99;

}

else if(IS\_ON(key2))

{

reset\_values(2);

IO0CLR = 0xFF<<16;

IO0SET |= leftSFT<<16;

leftSFT<<=1;

if(leftSFT > 1U<<7) leftSFT = 1;

}

else if(IS\_ON(key3))

{

reset\_values(3);

IO0CLR = 0xFF<<16;

IO0SET |= rightSFT<<16;

rightSFT>>=1;

if(rightSFT < 1) rightSFT = 1U<<7;

}

delay\_ms(100);

}

}

void reset\_values(int y)

{

switch(y)

{

case 0: contDN = 99;

rightSFT = 1U<<7;

leftSFT = 1;

break;

case 1: contUP = 0;

rightSFT = 1U<<7;

leftSFT = 1;

break;

case 2: contUP = 0;

contDN = 99;

rightSFT = 1U<<7;

break;

case 3: contUP = 0;

contDN = 99;

leftSFT = 1;

break;

}

}

void delay\_ms(unsigned int ms) {

for(int i = 0; i < ms; i++) {

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

}

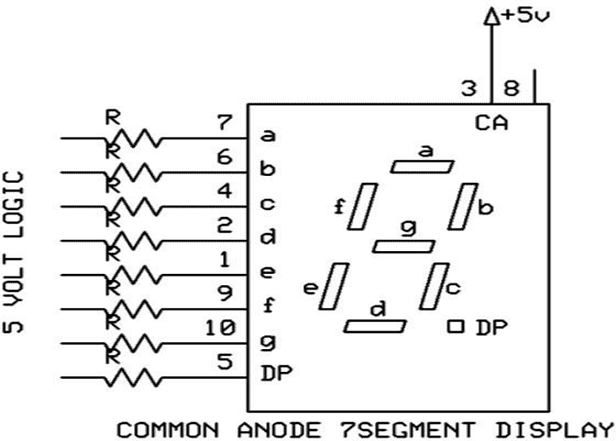
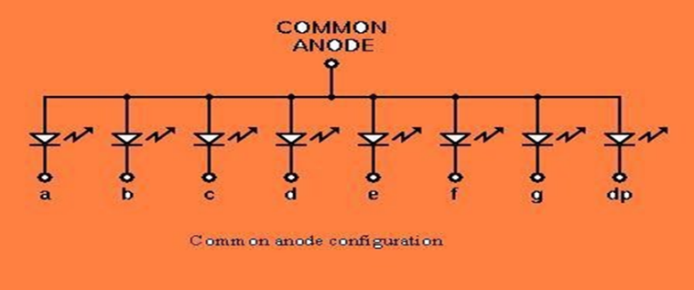
}

**Interfacing Circuit working Explanation:**

**Output Observation:**

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

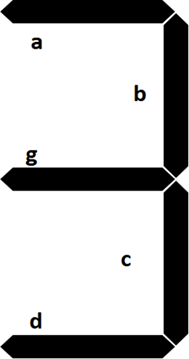
Serial In Parallel Out mode of Shift Register (74HC4094) is used to send 8 bits of data to seven segment display. Seven segment display used is of common anode type i.e. we have to send 0 to make corresponding segment ON and 1 to make it OFF.



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

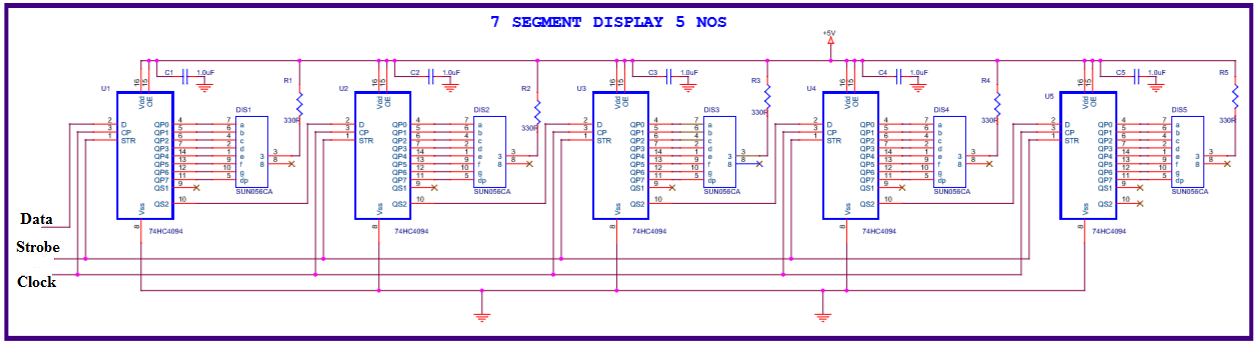
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DP | G | f | e | d | c | b | a |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

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);

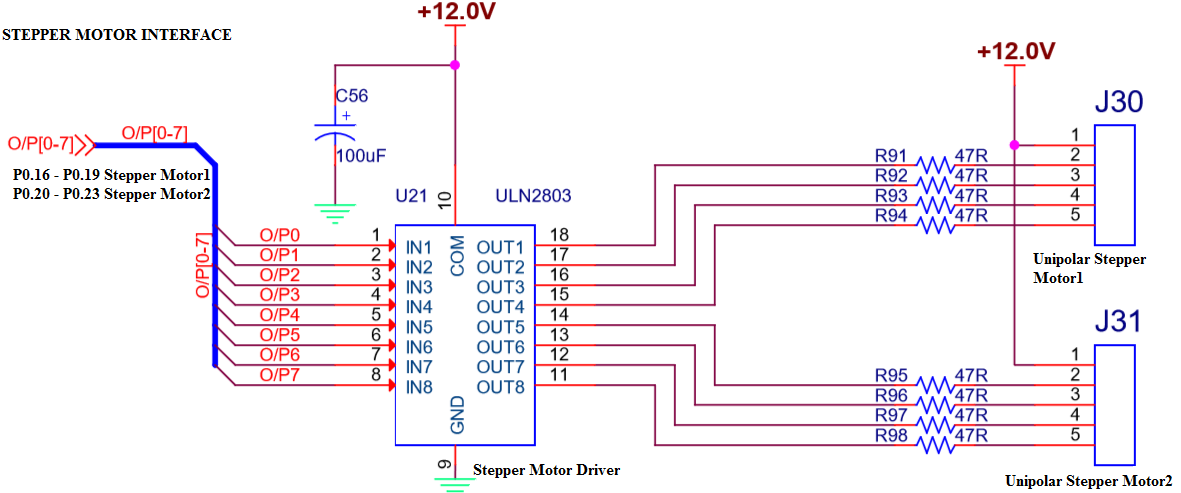
// alphadisp7SEG(&buf[0]);

**Interfacing Circuit working Explanation:**

**Output Observation:**

**Program No.3: 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++);

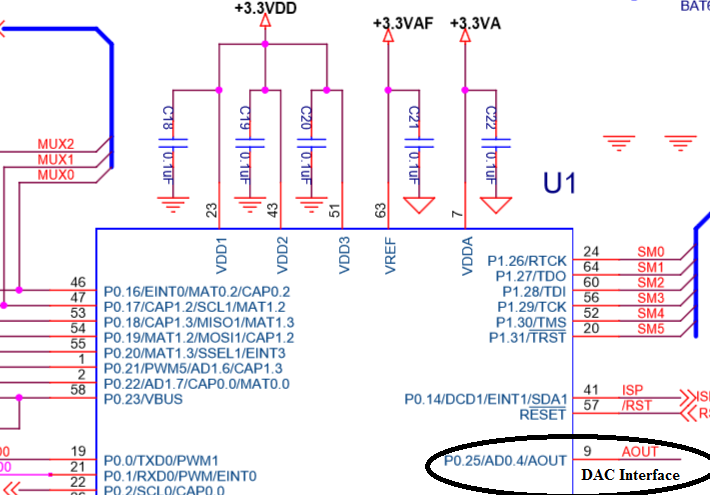
}

}

**Interfacing Circuit working Explanation:**

**Output Observation:**

**Program No.4: 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

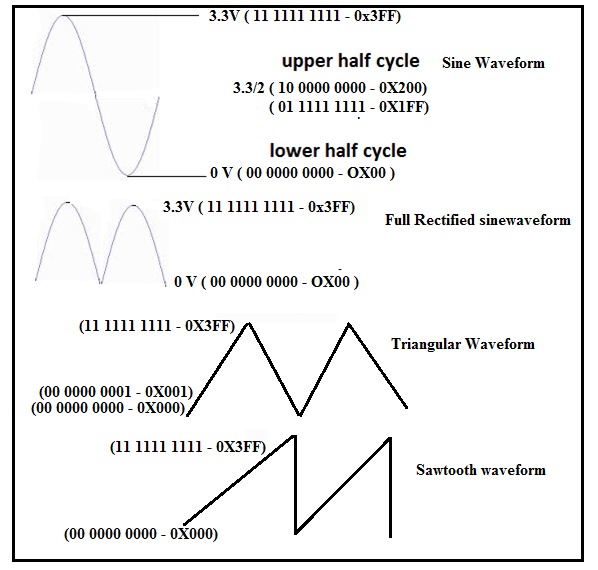
511 x sin 24 = 208 511 x sin 72 = 486

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++);

}

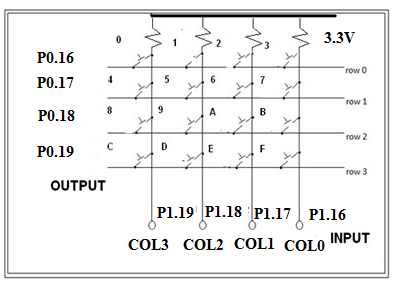
}

**Interfacing Circuit working Explanation:**

**Output Observation:**

**Program No.5: 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++);

}

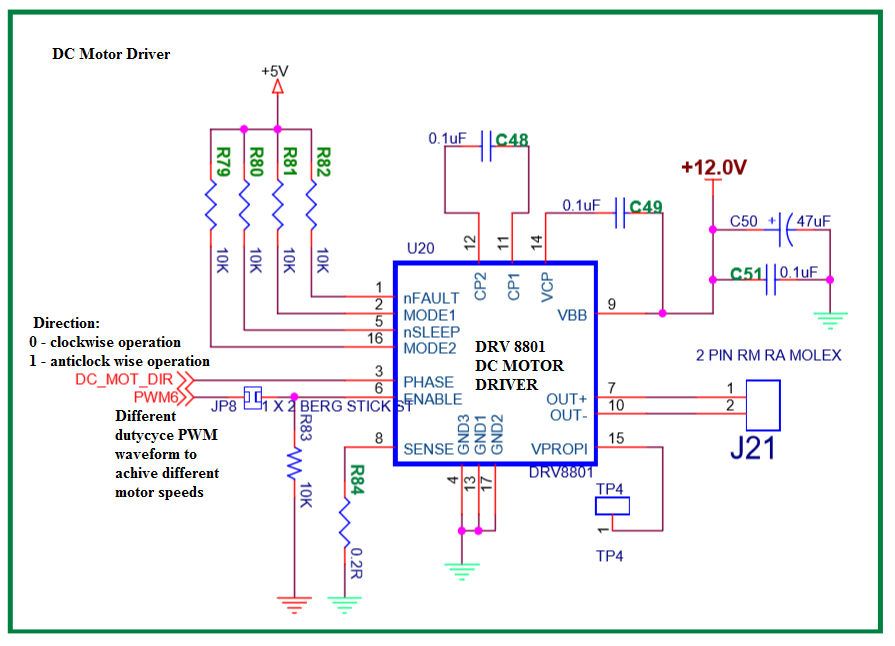
}

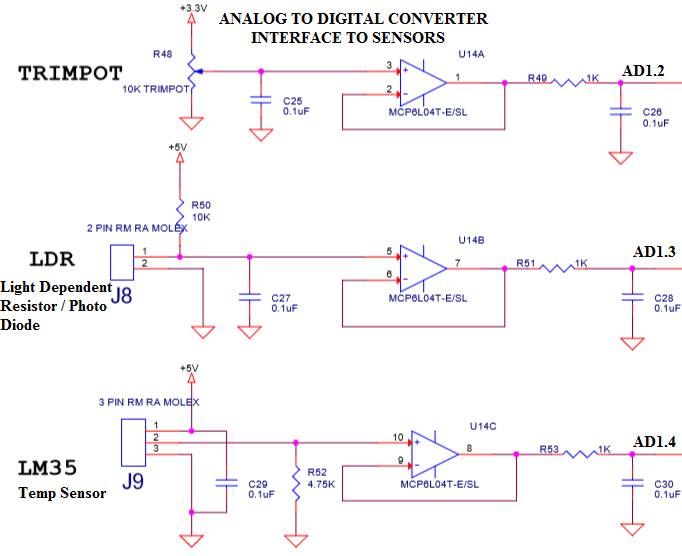
**Interfacing Circuit working Explanation:**

**Output Observation:**

**Program No. 6: 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++);

}

}

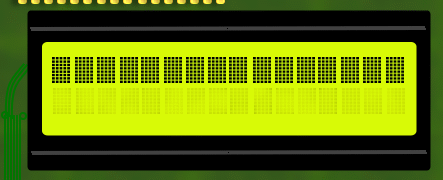
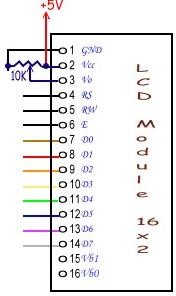
**Interfacing Circuit working Explanation:**

**Output Observation:**

**Program No.7: 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)**

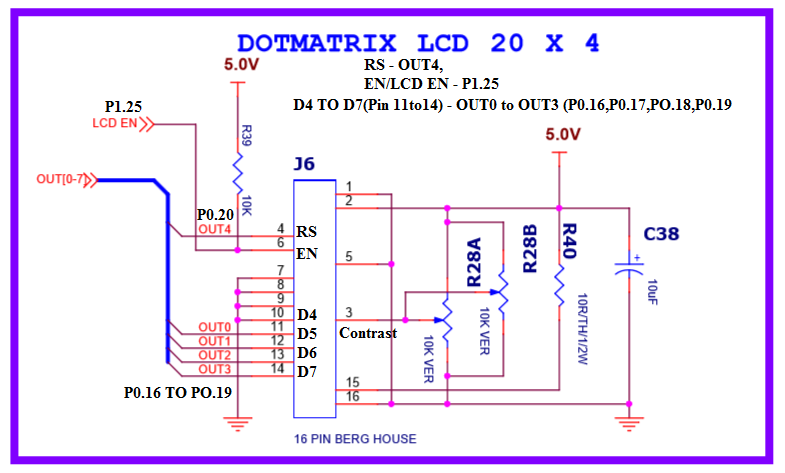
1. Write to LCD (Data/Command)
2. Read from the LCD

**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++);

}

}

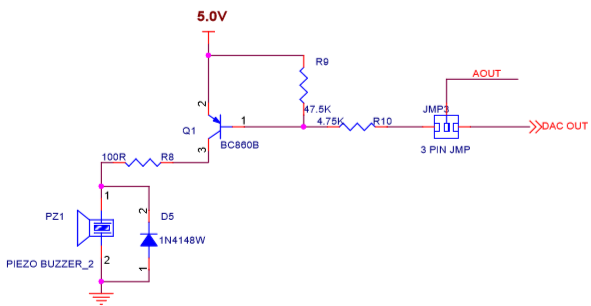
**Interfacing Circuit working Explanation:**

**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) );

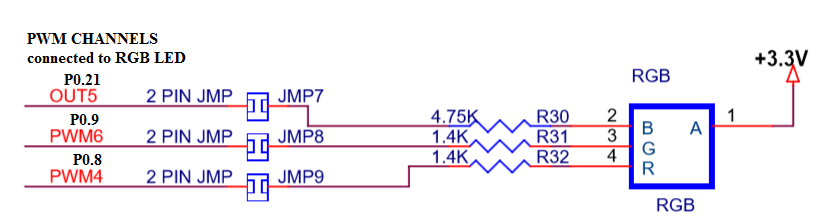
delay\_ms(500);

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*

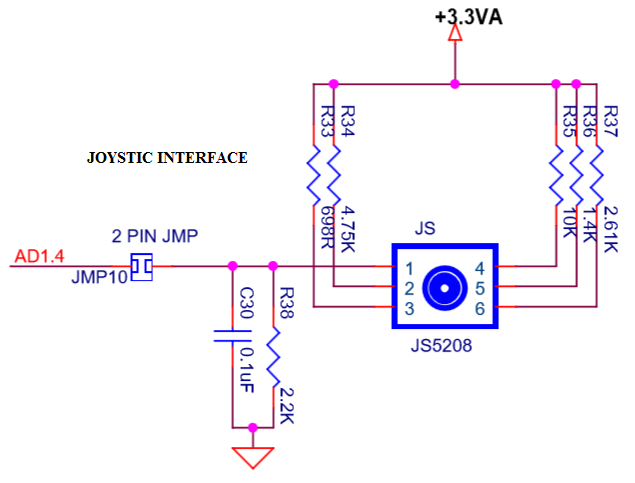
PWMPCR = (1 << 12 | 1 << 13 | 1 << 14); *//Enable PWM4,PWM5 and PWM*

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

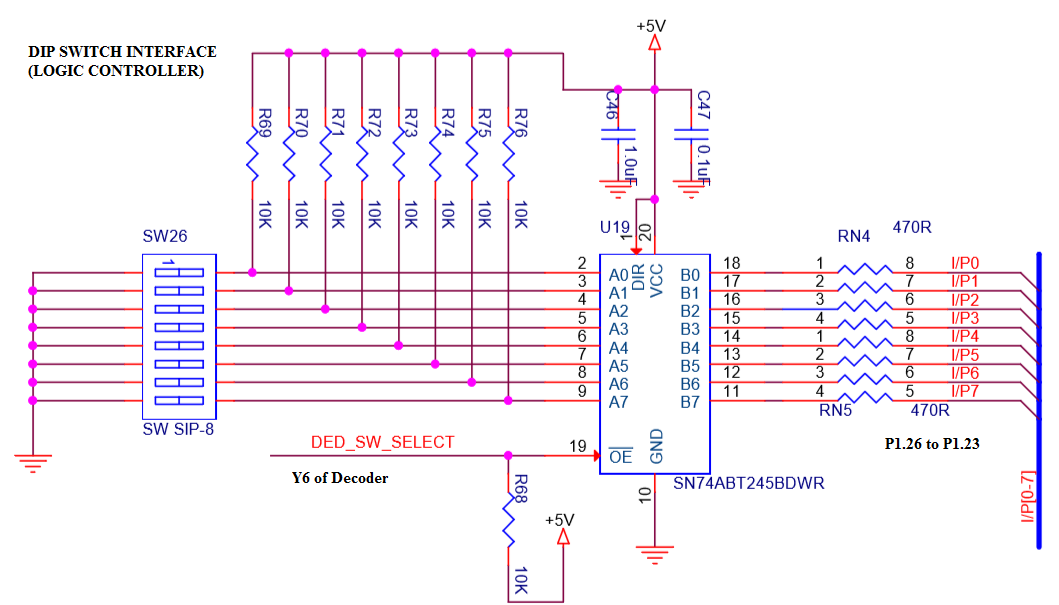
else if(adc\_result > 614 && adc\_result < 624) res = 3; // Right

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*

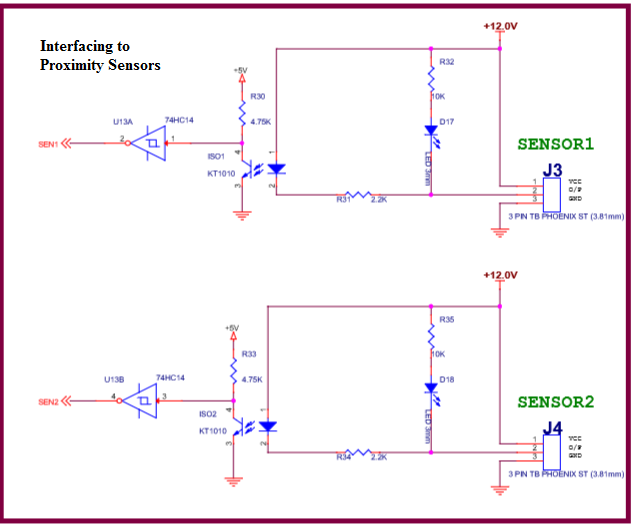
IO0SET = (1U << 7);delay\_ms(2);

*//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;

}