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 Computing - Laboratory

Subject Code: CS344AI

IV SEMESTER B.E.

LABORATORY RECORD

[Autonomous Scheme 2022]

2024-2025

Name of the Student:____________________________ **USN:** ________________

Semester:_______________**Section:**_________________**Year:**________________

Rashtreeya Sikshana Samithi Trust

R. V. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IOT & EMBEDDED COMPUTING LAB

(CS344AI)

2024 - 2025

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 completedthe course of experiments in IOT and Embedded Computing Lab [CS344AI] prescribed by the department during the year 2024-2025.

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.

KNOWLEDGE & ATTITUDE PROFILE

WK1: A systematic, theory-based understanding of the natural sciences applicable to the discipline and awareness of relevant social sciences.

WK2: Conceptually-based mathematics, numerical analysis, data analysis, statistics and formal aspects of computer and information science to support detailed analysis and modelling applicable to the discipline.

WK3: A systematic, theory-based formulation of engineering fundamentals required in the engineering discipline.

WK4: Engineering specialist knowledge that provides theoretical frameworks and bodies of knowledge for the accepted practice areas in the engineering discipline; much is at the forefront of the discipline.

WK5: Knowledge, including efficient resource use, environmental impacts, whole-life cost, re-use of resources, net zero carbon, and similar concepts, that supports engineering design and operations in a practice area.

WK6: Knowledge of engineering practice (technology) in the practice areas in the engineering discipline.

WK7: Knowledge of the role of engineering in society and identified issues in engineering practice in the discipline, such as the professional responsibility of an engineer to public safety and sustainable development.

WK8: Engagement with selected knowledge in the current research literature of the discipline, awareness of the power of critical thinking and creative approaches to evaluate emerging issues.

WK9: Ethics, inclusive behaviour and conduct. Knowledge of professional ethics, responsibilities, and norms of engineering practice. Awareness of the need for diversity by reason of ethnicity, gender, age, physical ability etc. with mutual understanding and respect, and of inclusive attitudes

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: Apply Embedded System and IoT fundamentals and formulate sustainable societal relevant cost-effective solutions.

CO2: Demonstrate the development of software programs using Embedded C, using Microcontrollers and different sensors and peripherals to build embedded system applications .

CO3: Design smart systems using various I/O peripherals, Sensors, embedded protocols like UART,I2C,SPI using modern tools like Keil IDE software for various domains like Healthcare, automation, agriculture, smart cities and others.

CO4: Indulge in developing Novel multi-disciplinary IoT projects using prototype boards, with effective oral & written communication skills and working in teams.

CO5: Engage in Lifelong Learning by investigating and executing real world societal problems using engineering tools – Cross compilers, debuggers and simulators, emerging processor and controller-based hardware platforms, IOT cloud infrastructure & protocols.

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

Part A

Laboratory Experiments using RV-ARM-Board (LPC 2148 ARM Microcontroller) comprises of,

Prog No.	Program	Page No	Marks Split as per rubrics					Marks (10)
			Execution			Viva		
			2	2	2	2	2	
1	Simulator Elevator Interface using switches and LEDs.							
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.							
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.							
4	DAC Interface: Write an Embedded C program to generate sine, full rectified, triangular, sawtooth and square waveforms using DAC module							
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.							
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.							
7	Character LCD Interface: Write an Embedded C program to display text messages on the multiple lines of the display.							
Total for 70 Marks								

PART-B

Design & Develop IOT based Solutions, using RV-IOT-Board, Use ThingSpeak /AWS cloud services, Use Web Application Frameworks like Django/Mobile App using C/C++/ Phython coding and relevant libraries/APIs

Project No.	**Program**	**Page No**	**Marks Split as per rubrics**					**Marks (10)**
			Execution			**Viva**		
			2	**2**	**2**	**2**	**2**	
1	RV-IOT-Board: Practice Project - Data Logger							
2	Project 1: Smart Lighting							
3	Project 2: Intrusion Detection System							
4	Project 3: Smart Parking							
5	Project 4: Weather Monitoring and Reporting Bot							
6	Project 5: Smart Irrigation							
7	Project 6: Forest Fire Detection							
						Total for 70 Marks		

LAB INTERNALS	
RECORD Marks (Part A & Part B)	/ 30 Marks
Mini Projects	/ 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 Embedded C for ARM concepts. **(2M)**	Student has sufficient understanding of program requirements and applies Embedded C for ARM concepts. **(1.5M - 1M)**	Student does not have clear understanding of program requirements and is unable to apply 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 & Embedded C related concepts involved. **(2M)**	Adequately explains architecture & 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:					

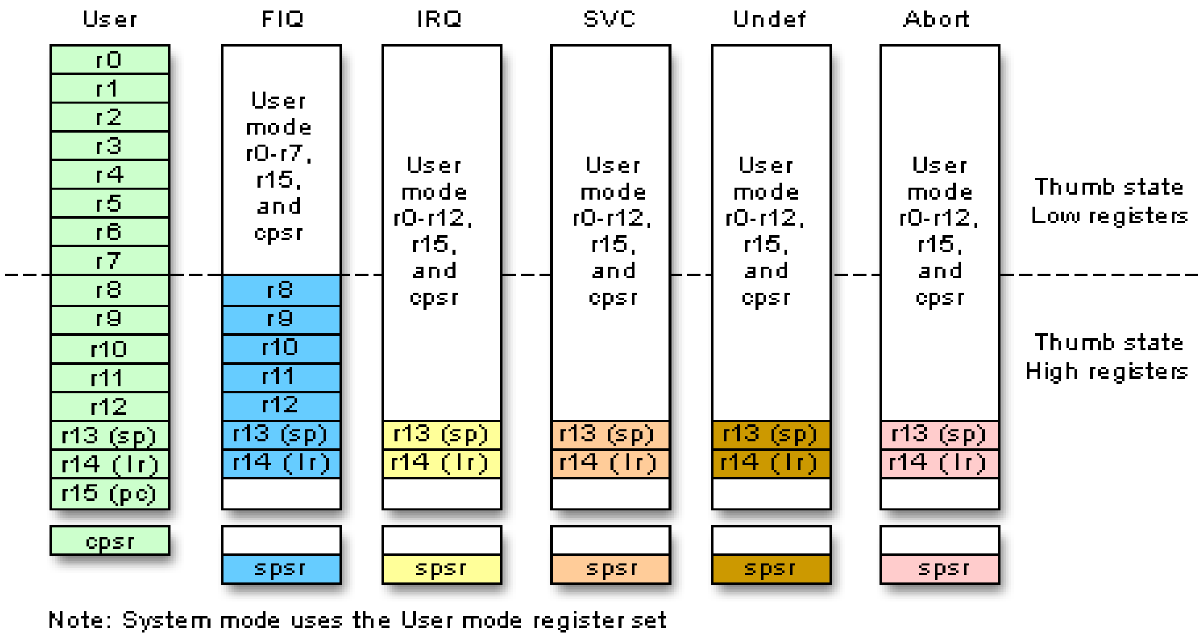
Part A- ARM Fundamentals

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,

i) 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.
ii) FIQ Mode (Fast interrupt request): Entered this mode on request from FIQ interrupts
iii) IRQ Mode (Interrupt Request): Entered this mode on IRQ request
iv) 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.
v) Abort: Used to handle memory access violations, Abort is entered after a failed memory access
vi) Undef: Used to handle undefined instructions, Undefined is entered if the instruction is not defined (invalid opcodes)
vii) 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


User
FIQ
IRQ
SVC
Undef
Abort
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr
User mode r0-r7, r15, and cpsr
User mode r0-r12, r15, and cpsr
spsr
Thumb state Low registers
Thumb state High registers
Note: System mode uses the User mode register set

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

i) PC - 1 dedicated program counter

ii) CPSR – 1 dedicated program status register, (like flag register of 8086)

iii) 5 dedicated saved program status registers (SPSR)

iv) 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 $\leftarrow$ 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 $\leftarrow$ R0 .AND. R1 (bit AND operation), S – updatde flags
Ex: **ORRS R5, R0 , R1** ; R5 $\leftarrow$ R0 .OR. R1 (bit AND operation), S – updatde flags
Ex: **EORS R5, R0 , R1** ; R5 $\leftarrow$ 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 $\leftarrow$ 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 $\leftarrow$ R2 x R3
MLA R4 , R3 , R2 , R1 - Multiply and Accumulate; R4 $\leftarrow$(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**)**
2) **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 1:

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,#0x25 ; B
         MOV  R2,#0x34 ; C
         MOV  R3,#0x72; D

         ADD  R0,R2,R1
         SUB  R0,R0,R3

STOP     B   STOP
         END
```

A = 4* A + B

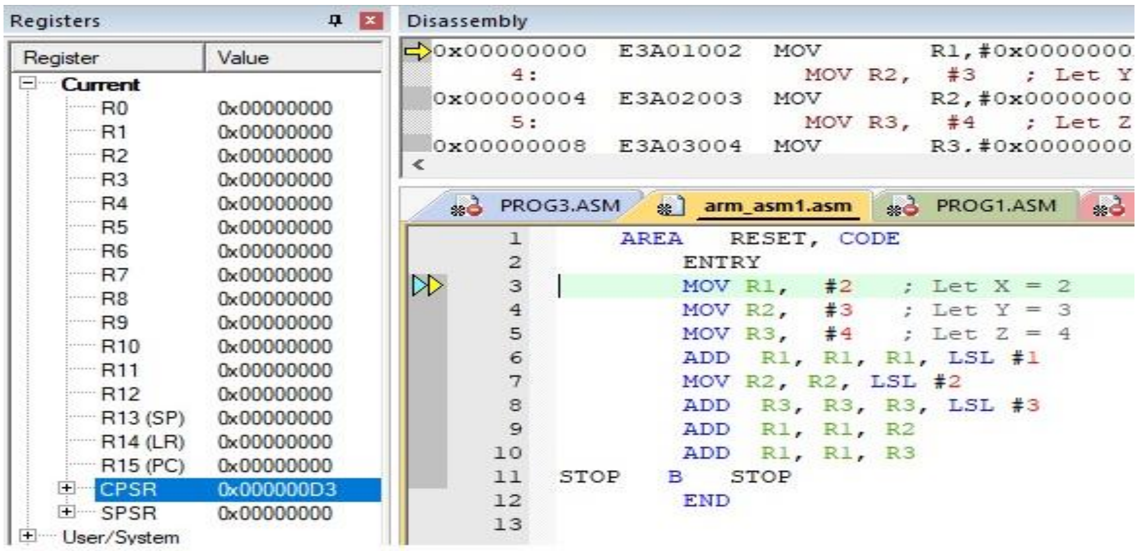
```
    AREA   RESET, CODE
         ENTRY

         MOV  R0,#0X25 ; A
         MOV  R1,#0X34; 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:

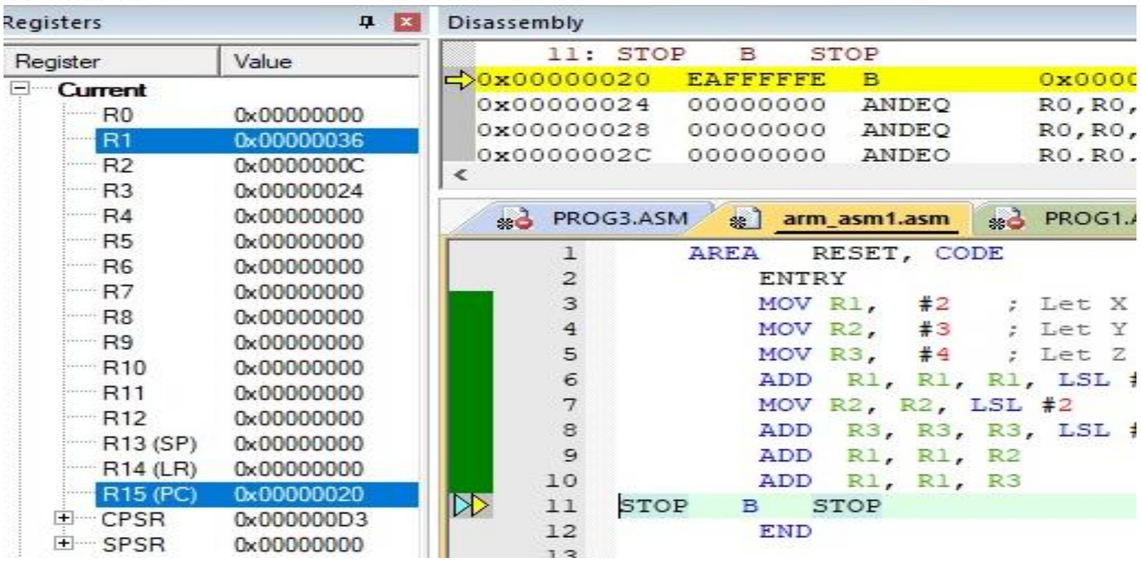
Before Execution After Execution



```
Registers
Register        Value
Current
  R0            0x00000000
  R1            0x00000000
  R2            0x00000000
  R3            0x00000000
  R4            0x00000000
  R5            0x00000000
  R6            0x00000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x00000000
  R14 (LR)      0x00000000
  R15 (PC)      0x00000000
  CPSR          0x000000D3
  SPSR          0x00000000
User/System
```

```
Disassembly
0x00000000  E3A01002  MOV        R1,#0x0000000
     4:                   MOV R2,  #3   ; Let Y
0x00000004  E3A02003  MOV        R2,#0x0000000
     5:                   MOV R3,  #4   ; Let Z
0x00000008  E3A03004  MOV        R3.#0x0000000
```

```
PROG3.ASM   arm_asm1.asm   PROG1.ASM

        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
```



```
Registers
Register        Value
Current
  R0            0x00000000
  R1            0x00000036
  R2            0x0000000C
  R3            0x00000024
  R4            0x00000000
  R5            0x00000000
  R6            0x00000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x00000000
  R14 (LR)      0x00000000
  R15 (PC)      0x00000020
  CPSR          0x000000D3
  SPSR          0x00000000
```

```
Disassembly
    11: STOP    B    STOP
0x00000020  EAFFFFFE  B        0x000C
0x00000024  00000000  ANDEQ    R0,R0,
0x00000028  00000000  ANDEQ    R0,R0,
0x0000002C  00000000  ANDEQ    R0.R0.
```

```
PROG3.ASM   arm_asm1.asm   PROG1.A

        AREA    RESET, CODE
            ENTRY
            MOV R1,    #2    ; Let X
            MOV R2,    #3    ; Let Y
            MOV R3,    #4    ; Let Z
            ADD  R1, R1, R1, LSL #
            MOV  R2, R2, LSL #2
            ADD  R3, R3, R3, LSL #
            ADD  R1, R1, R2
            ADD  R1, R1, R3
STOP    B    STOP
            END
```

Sample Program No 2:

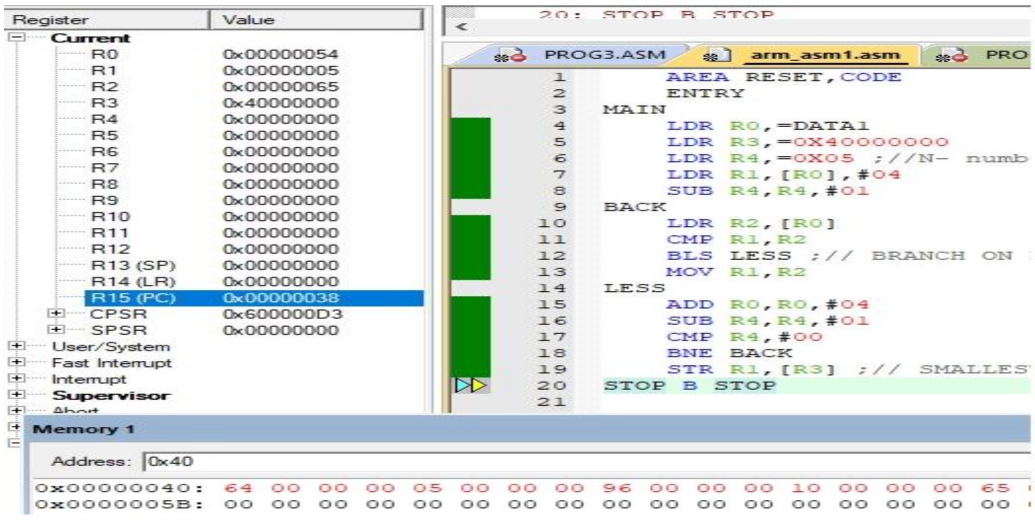
AIM: Write an ARM ALP to find smallest and largest of N- 32 bit numbers.

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*.

```
          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 R2,R1
          MOVLS R1,R2 ; update with new smaller no
          ;ADD R0, R0,#04   ; increment pointer to next number
          SUBS  R4,R4,#01
          ;CMP  R4,#00
          BNE  BACK
          STR   R1, [R3]  ; // SMALLEST VALUE STORED IN MEMORY LOCATION
          STOP B STOP

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

Sample Output:



```
Register          Value
Current
  R0              0x00000054
  R1              0x00000005
  R2              0x00000065
  R3              0x40000000
  R4              0x00000000
  R5              0x00000000
  R6              0x00000000
  R7              0x00000000
  R8              0x00000000
  R9              0x00000000
  R10             0x00000000
  R11             0x00000000
  R12             0x00000000
  R13 (SP)        0x00000000
  R14 (LR)        0x00000000
  R15 (PC)        0x00000038
  CPSR            0x600000D3
  SPSR            0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
```

```
20: STOP B STOP
```

```
        AREA RESET,CODE
        ENTRY
MAIN
        LDR R0,=DATA1
        LDR R3,=0X40000000
        LDR R4,=0X05 ;//N- numb
        LDR R1,[R0],#04
        SUB R4,R4,#01
BACK
        LDR R2,[R0]
        CMP R1,R2
        BLS LESS ;// BRANCH ON
        MOV R1,R2
LESS
        ADD R0,R0,#04
        SUB R4,R4,#01
        CMP R4,#00
        BNE BACK
        STR R1,[R3] ;// SMALLES
STOP B STOP
```

```
Memory 1
Address: 0x40
0x00000040: 64 00 00 00 05 00 00 00 96 00 00 00 10 00 00 00 65
0x0000005B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Sample Program No 3:

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

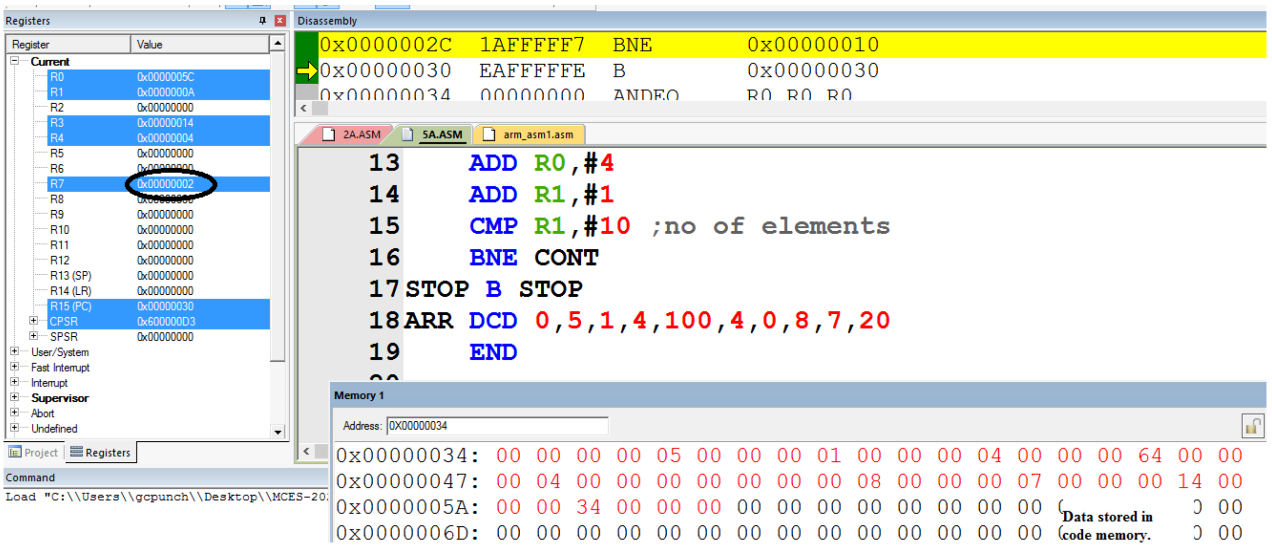
Algorithm:

```
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:



```
Registers
Register        Value
Current
  R0            0x0000005C
  R1            0x0000000A
  R2            0x00000000
  R3            0x00000014
  R4            0x00000004
  R5            0x00000000
  R6            0x00000000
  R7            0x00000002
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x00000000
  R14 (LR)      0x00000000
  R15 (PC)      0x00000030
  CPSR          0x600000D3
  SPSR          0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
Undefined
```

```
Disassembly
0x0000002C  1AFFFFF7  BNE      0x00000010
0x00000030  EAFFFFFE  B        0x00000030
0x00000034  00000000  ANDEQ    R0,R0,R0
```

```
        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
```

```
Memory 1
Address: 0X00000034
0x00000034: 00 00 00 00 05 00 00 00 01 00 00 00 04 00 00 00 64 00 00
0x00000047: 00 04 00 00 00 00 00 00 00 08 00 00 00 07 00 00 00 14 00
0x0000005A: 00 00 34 00 00 00 00 00 00 00 00 00 00 00          0 00
0x0000006D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00          0 00
```

Data stored in code memory.

```
Command
Load "C:\\Users\\gcpunch\\Desktop\\MCES-20
```

Sample Program No 4:

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

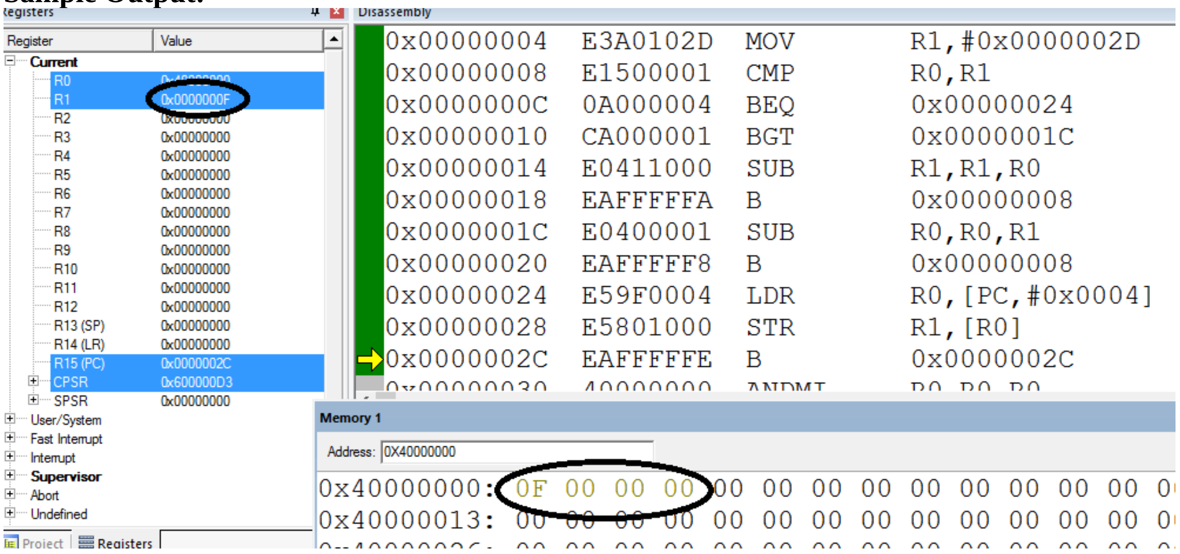
Algorithm:

```
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
        SUBLT R1,R0
        SUBGT R0,R1
        BNE LOOP
EXIT    LDR R0,=GCD
        STR R1,[R0]
STOP    B STOP
        AREA RESULT,DATA
GCD     SPACE 4
        END
```

Sample Output:



```
Registers
Register        Value
Current
  R0            0x40000000
  R1            0x0000000F
  R2            0x00000000
  R3            0x00000000
  R4            0x00000000
  R5            0x00000000
  R6            0x00000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x00000000
  R14 (LR)      0x00000000
  R15 (PC)      0x0000002C
  CPSR          0x600000D3
  SPSR          0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
Undefined
```

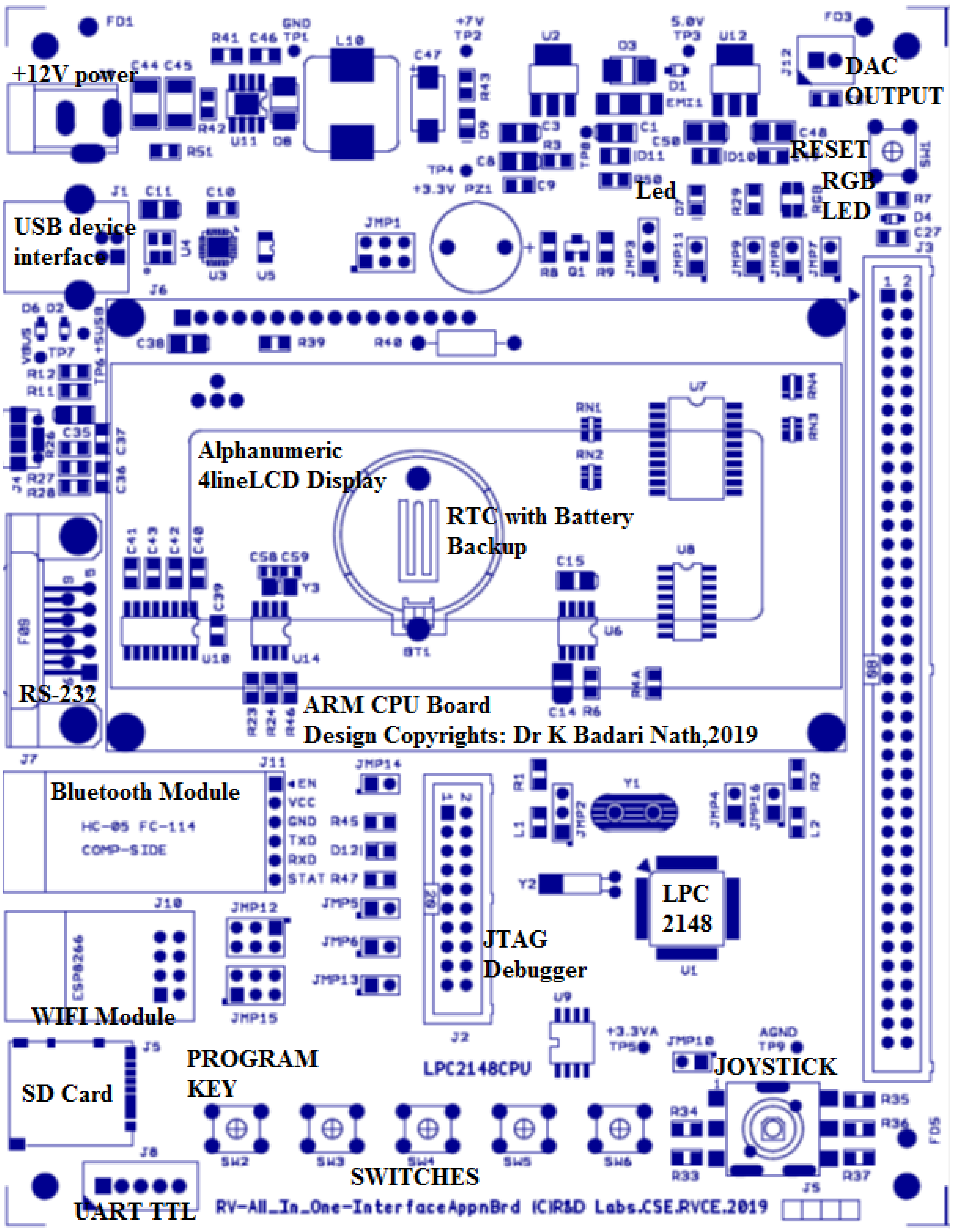
```
Disassembly
0x00000004  E3A0102D  MOV     R1,#0x0000002D
0x00000008  E1500001  CMP     R0,R1
0x0000000C  0A000004  BEQ     0x00000024
0x00000010  CA000001  BGT     0x0000001C
0x00000014  E0411000  SUB     R1,R1,R0
0x00000018  EAFFFFFA  B       0x00000008
0x0000001C  E0400001  SUB     R0,R0,R1
0x00000020  EAFFFFF8  B       0x00000008
0x00000024  E59F0004  LDR     R0,[PC,#0x0004]
0x00000028  E5801000  STR     R1,[R0]
0x0000002C  EAFFFFFE  B       0x0000002C
0x00000030  40000000  ANDMI   R0,R0,R0
```

```
Memory 1
Address: 0X40000000
0x40000000: 0F 00 00 00 00 00 00 00 00 00 00 00 00 0
0x40000013: 00 00 00 00 00 00 00 00 00 00 00 00 00 0
```

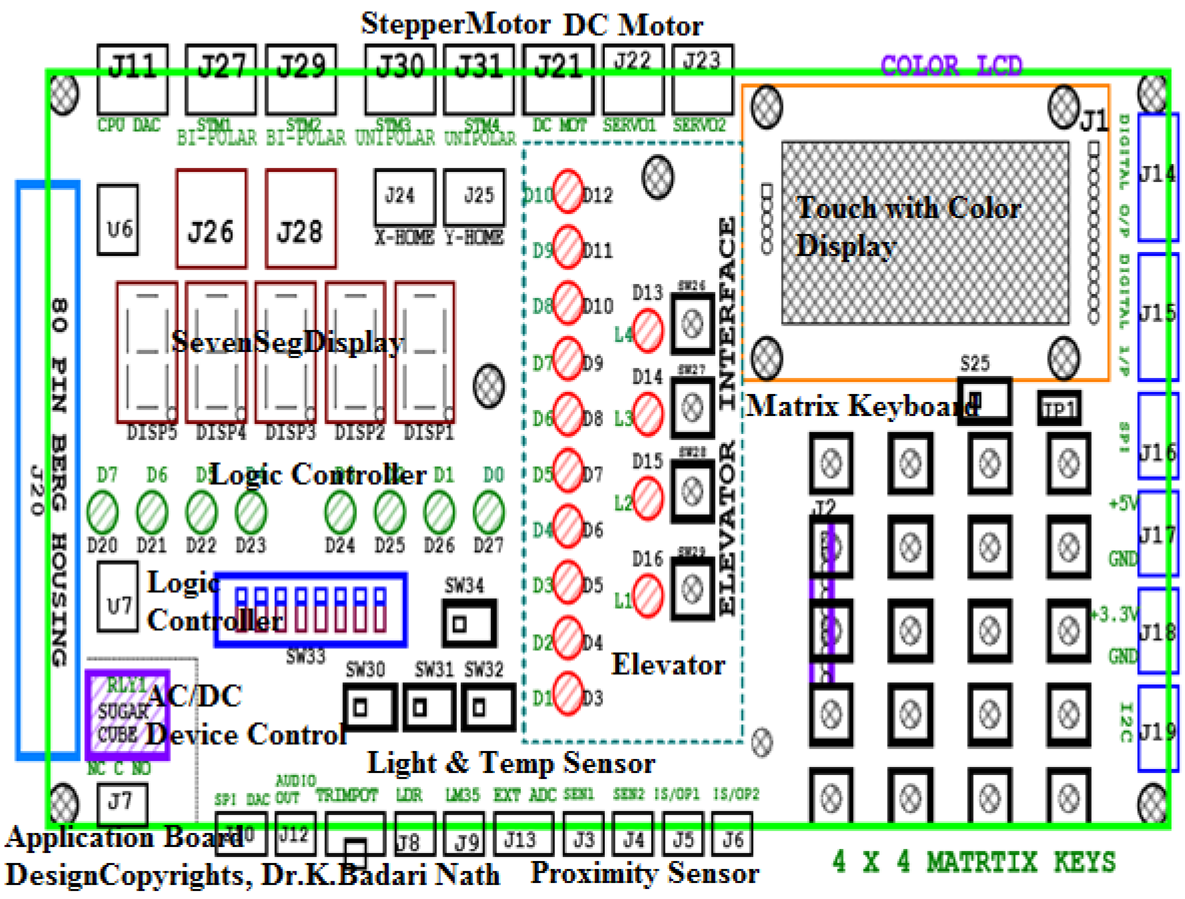
PART – A

Interfacing Programs using ARM LPC 2148

ARM CPU Board



ARM Application Board



Board Specifications: Power : 12V, 1A

Input Interfaces

8 Bit DIP Switch (Toggle)
5 Application Switches
4 x 4 Matrix Keyboard
Joystic Interface
Touch Interface

Output Interfaces

9 LEDs
5 Digit Common Anode seven segment display
20x4 Alphanumeric display
RGB LED using PWM
Color Touch Enabled Display

Communication Interfaces

8 bit GPIO outputs
RS-232 COM Port
TTL Serial Port – UART
USB Client Interface
SPI Interface
I2C Interface
Wifi Interface
Bluetooth Interface

Debugging & Programming Support

Debugging & Programming
Virtual COM Port
JTAG Debugger
8LEDs & 5 Application Keys to support debugging of projects

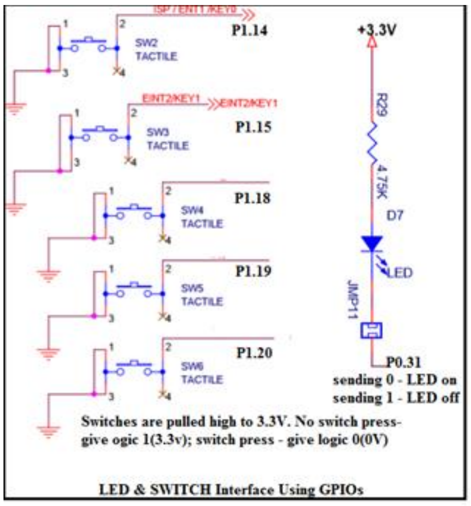
Storage Support

Storage Support
Flash MemoryChip – 8MB
SD Card Interface
RTC with Battery Backup

Board Specifications: RV - ARM AllInOne Board
ARM CPU Board & Application Board

Sample program : Interfacing LED and Switches

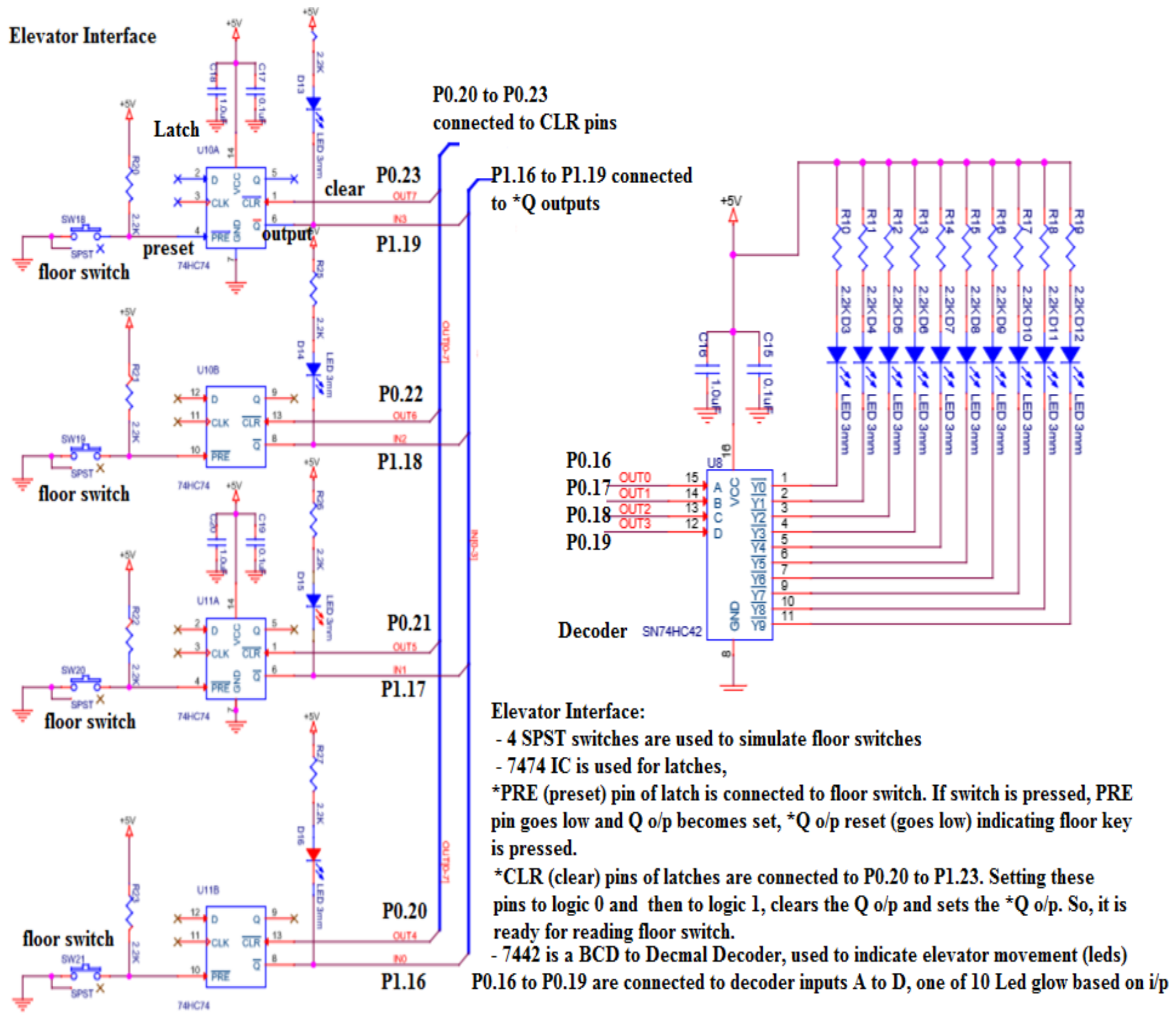
Interfacing Diagram



```
 //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: Elevator Interface: Write an Embedded C program to read the elevator switches and simulate elevator up and down movements.

Interfacing Diagram



Elevator Interface:
- 4 SPST switches are used to simulate floor switches
- 7474 IC is used for latches,

*PRE (preset) pin of latch is connected to floor switch. If switch is pressed, PRE pin goes low and Q o/p becomes set, *Q o/p reset (goes low) indicating floor key is pressed.

*CLR (clear) pins of latches are connected to P0.20 to P1.23. Setting these pins to logic 0 and then to logic 1, clears the Q o/p and sets the *Q o/p. So, it is ready for reading floor switch.

- 7442 is a BCD to Decmal Decoder, used to indicate elevator movement (leds)

P0.16 to P0.19 are connected to decoder inputs A to D, one of 10 Led glow based on i/p

//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*

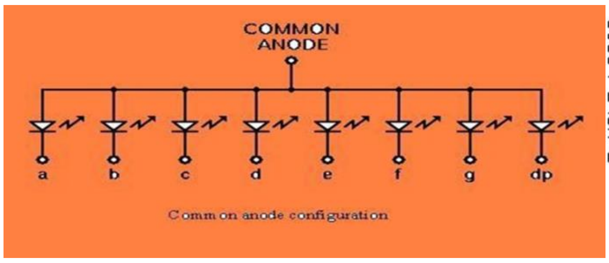
```
#include <lpc214x.h>
#define LED_OFF (IO0SET = 1U << 31)
#define LED_ON (IO0CLR = 1U << 31)
void delay_ms(unsigned int j);
void elevator_run(void);
int main()
{
        IO0DIR |= 1U << 31 | 0XFF << 16; // to set P0.31 & P1.20 to P1.23 as outputs
        IO1DIR |= 1 << 24; // to set P1.24 as output
        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 grnd 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);
}
```

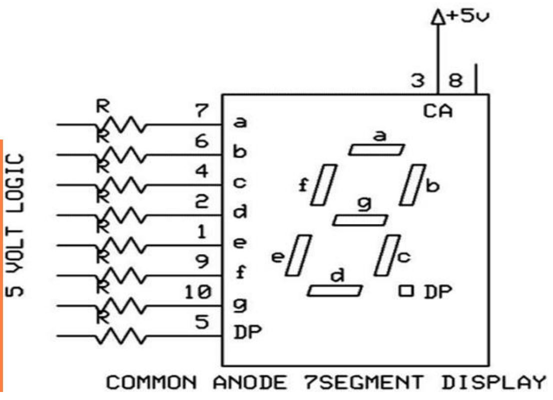
Interfacing Circuit working Explanation:

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.

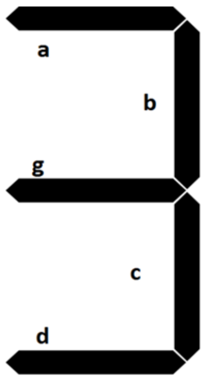




To display 3, we have to send following bit pattern,

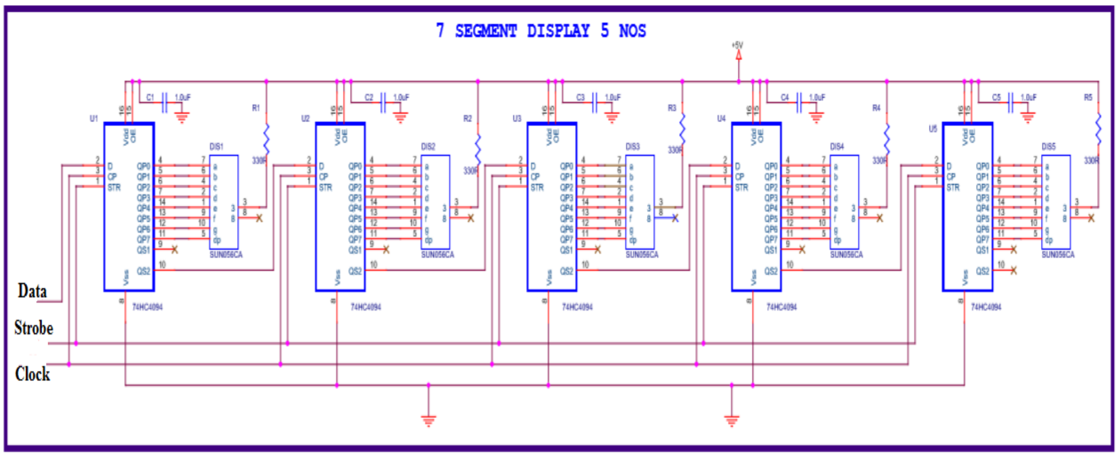
DP	G	f	E	d	c	b	a
1	0	1	1	0	0	0	0

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);
             }

}
```

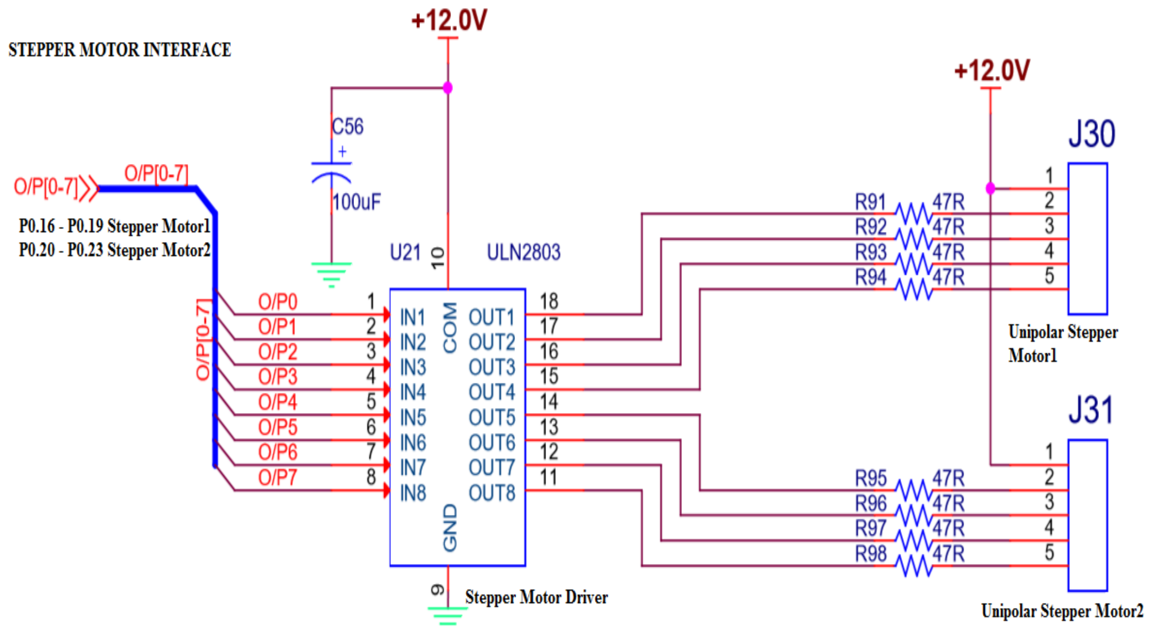
```
unsigned char getAlphaCode(unsigned char alphachar)
{
  switch (alphachar)
        {
        // dp g f e d c b a - common anode: 0 segment on, 1 segment off
        case 'f':return 0x8e;
        case 'i':return 0xf9;
        case 'r':return 0xce;
        case 'e':return 0x86;  // 1000 0110
        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°

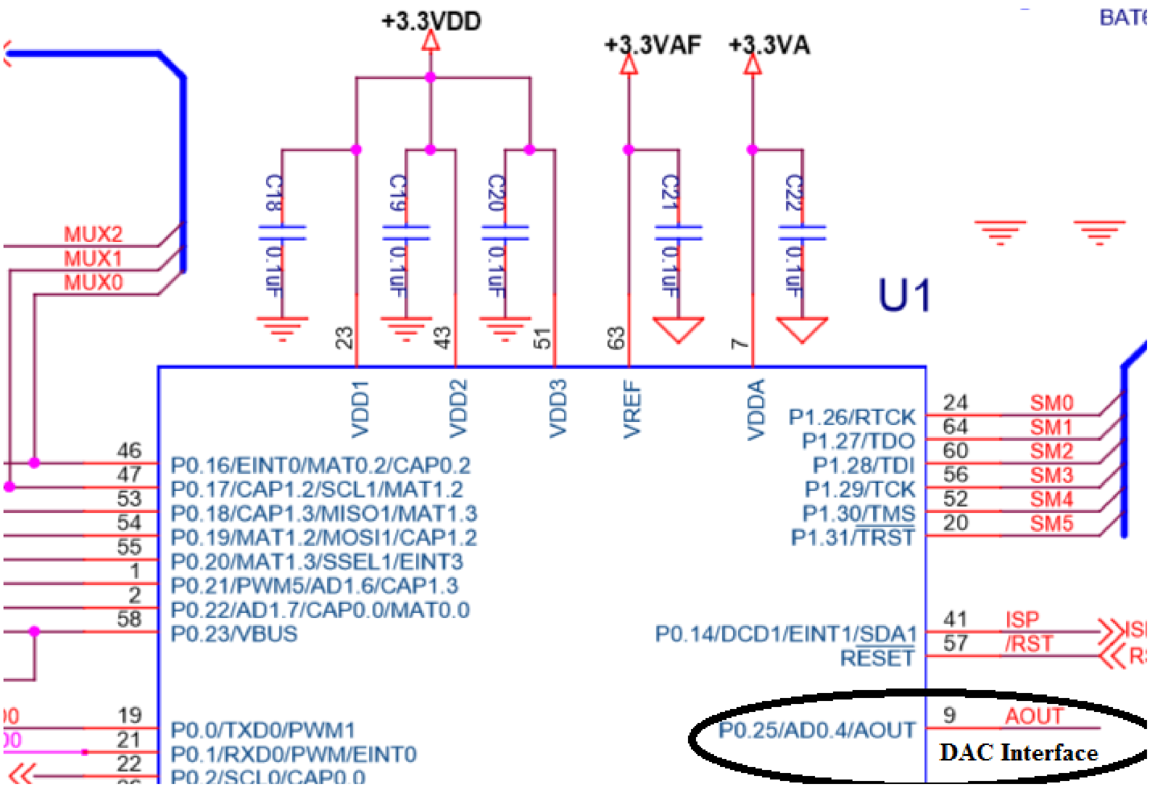
- Use appropriate delay in between consequent steps
- 2Phase, 4winding stepper motor is used, along with driver circuit(ULN 2803) built on the RV All-In- One Card, 12v power is used to drive the stepper motor. Digital input generated by the microcontroller, is used to drive and control the direction 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



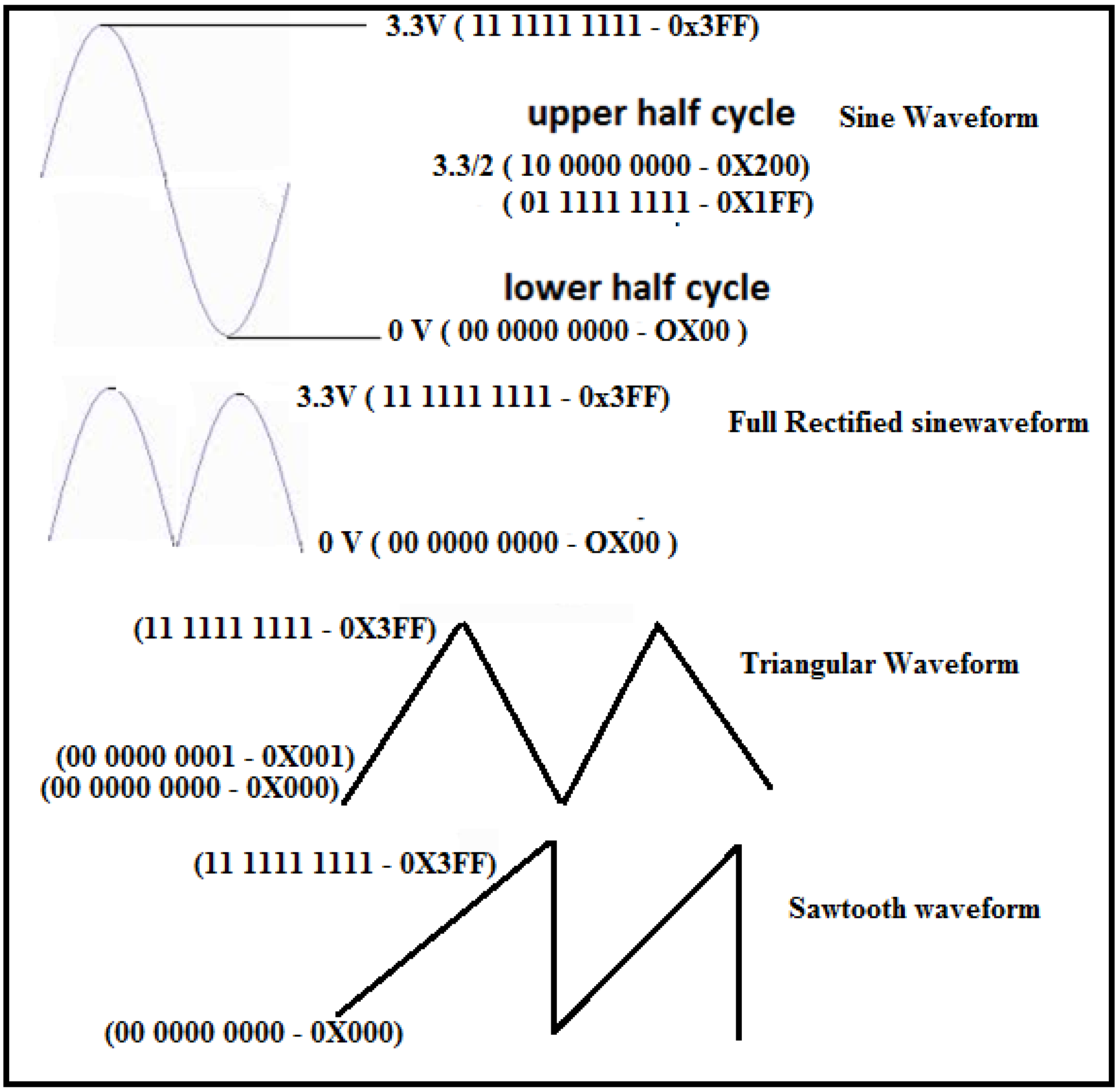
- DAC module of LPC 2148 is 10 bit Digital to Analog converter used to convert 10 bit Digital data to corresponding Analog voltage.
- Digital I/P : **000 to 3FF (0 to 1023)**, corresponding Analog O/P : **0V to 3.3V**
- Resolution = (3.3/1024) ≈ **3.2mili volts**

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 Θ**
(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 6^o,

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+25
6,512+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+25
6,512+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) );
          }
     }
 }
```

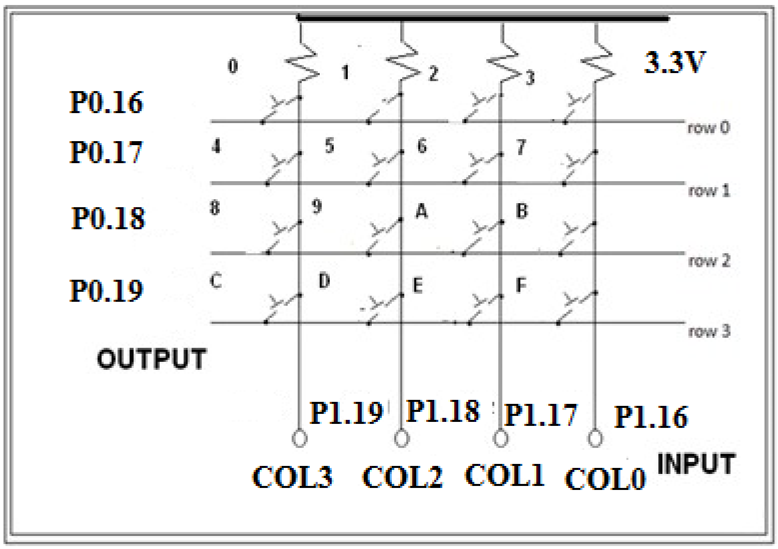
```
void SystemInit(void)
{
   PLL0CON = 0x01;
   PLL0CFG = 0x24;
   PLL0FEED = 0xAA;
   PLL0FEED = 0x55;
   while( !( PLL0STAT & PLOCK ))
   {   ;   }
   PLL0CON = 0x03;
   PLL0FEED = 0xAA;
   PLL0FEED = 0x55;
}
void delay_ms(unsigned int j)
{
  unsigned int x,i;
  for(i=0;i<j;i++)
   {
      for(x=0; 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.

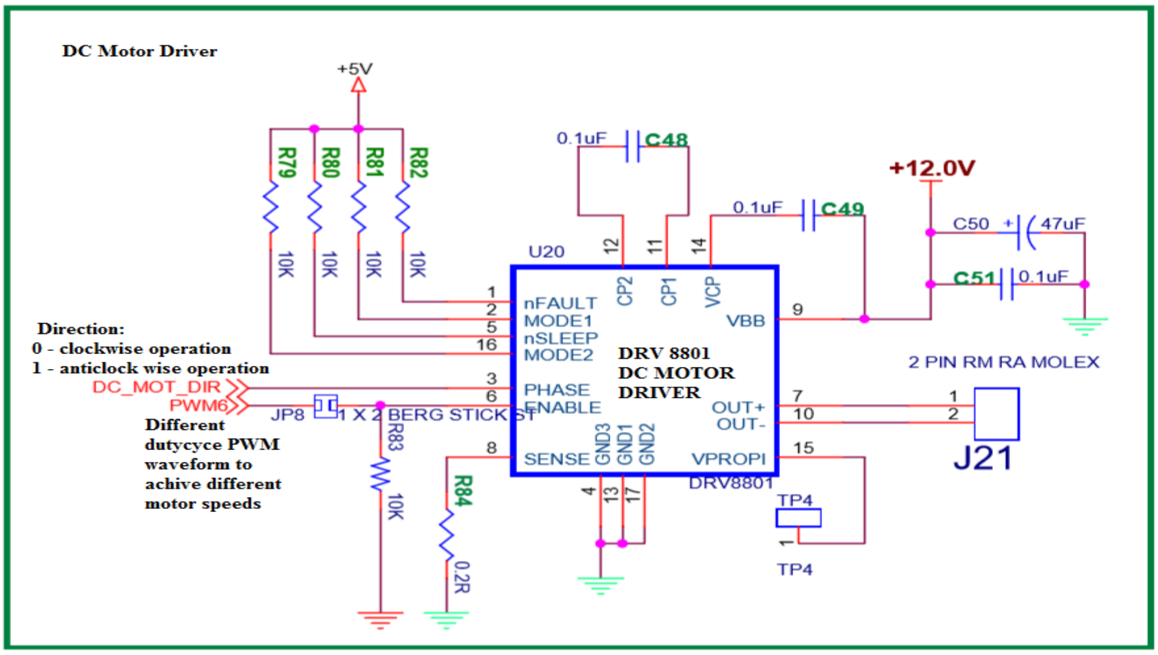
```
//Matrix 4 x 4 Keyboard
//Columns & Rows are pulled to +5v,if dont press key, we receive '1' on columns
//Method: Sending '0' to a selected row, checking for '0' on each column
//ROWS - ROW0-ROW3 -> P0.16,P0.17,P0.18,P0.19
//COLS - COL0-COL3 -> P1.19,P1.18,P1.17,P1.16
#include <lpc214x.h>
#define 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++);
   }
 }
```

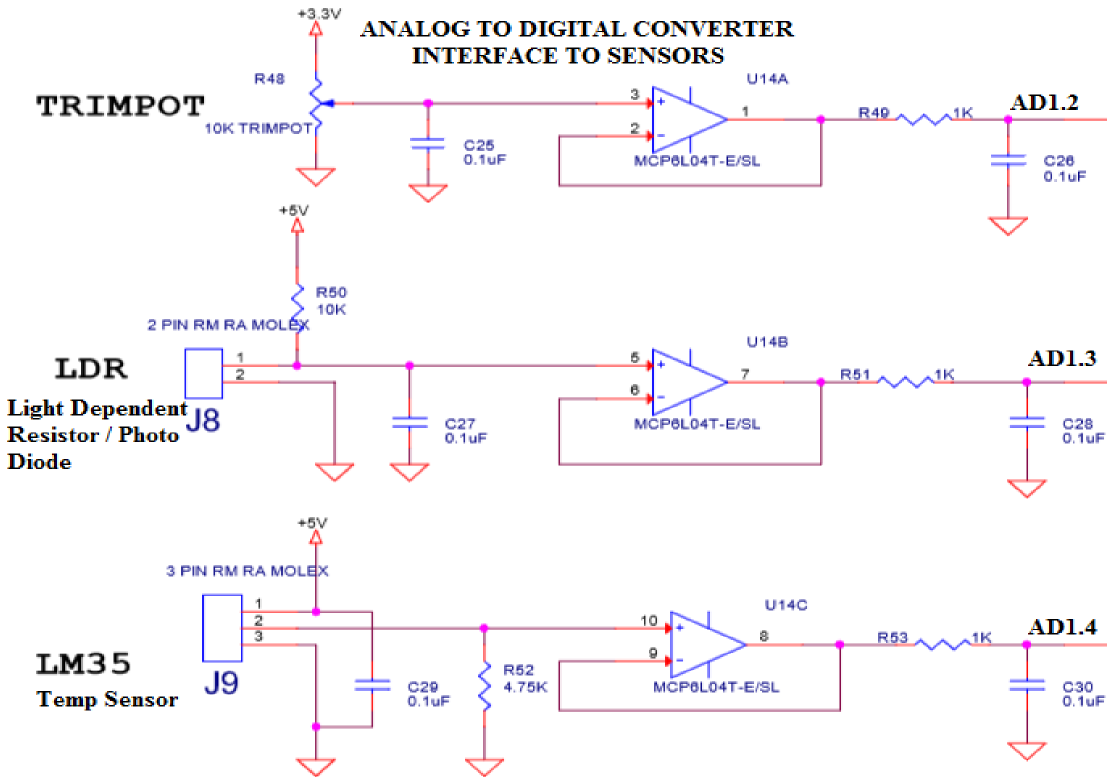
Interfacing Circuit working Explanation:

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:

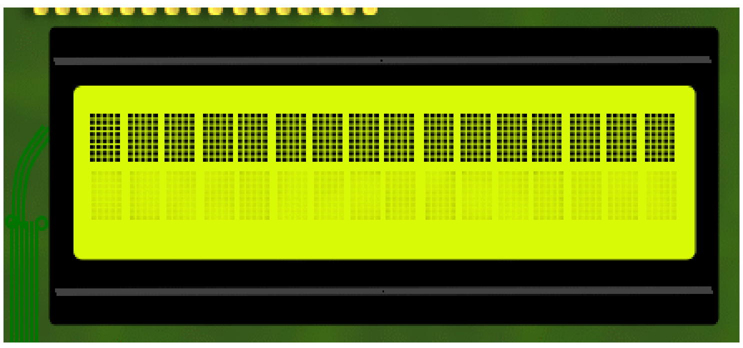
Output Observation:

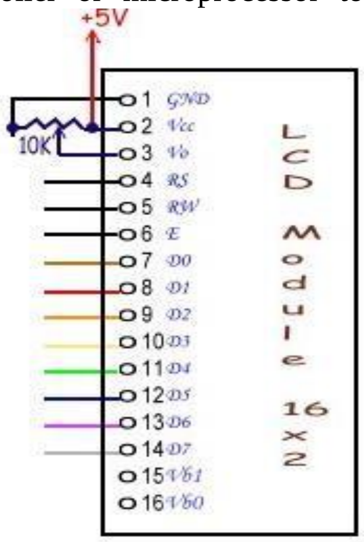
Program No.7: Alpha Numeric LCD Interface - Write an Embedded C program to display text messages on the multiple lines of the display.

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

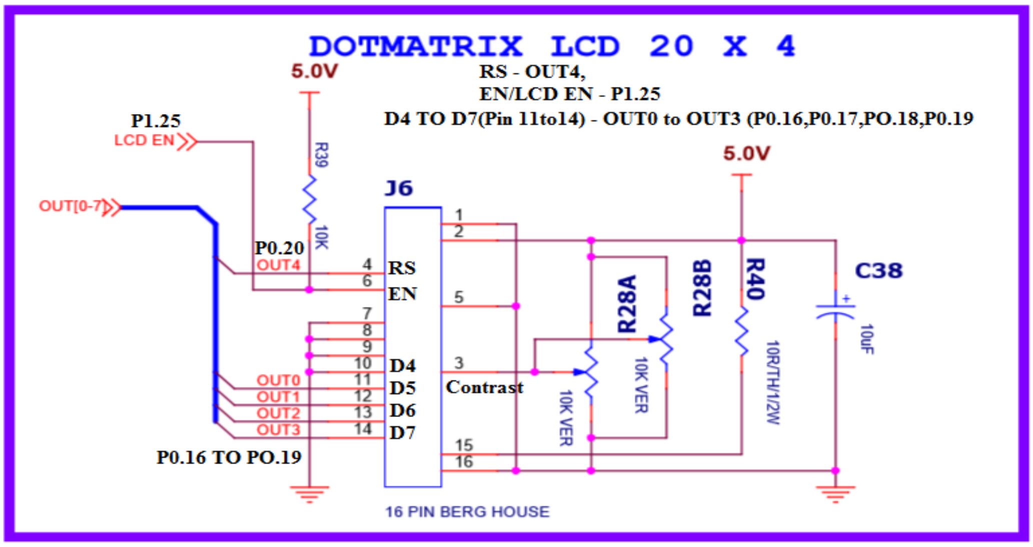




communicate, whatever the data we write to LCD is of two types, either it is a command to the LCD(to configure) or ASCII code of character to be displayed on LCD (to DDRAM). RS signal is used for this,

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)
  0- Write to LCD (Data/Command)
  1- 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:

PART - B

Internet Of Things

The Internet of Things (IoT) is a network where objects, equipped with sensors and other technologies, connect to the internet to exchange data and communicate. Its goal is to enhance intelligence, efficiency, and decision-making by enabling devices to collect and share data.

Key components of IoT include:

- Devices/Objects: Physical entities embedded with sensors, actuators, and technologies like LDR sensors, Potentiometers, and RTC modules for data collection and transmission.

- Connectivity: IoT devices utilize various communication protocols such as cellular networks, Bluetooth, Wi-Fi, and Zigbee to connect to the internet, enabling interaction between devices and the cloud.

- Data Processing: Data collected from sensors is processed locally on chips or in the cloud using algorithms to extract valuable insights, essential for decision-making and device actions.

- Cloud Computing: Cloud platforms store and process the vast amounts of data generated by IoT devices, offering scalability, flexibility, and accessibility for applications and services.

- Applications: IoT applications span across industries like smart homes, healthcare, agriculture, manufacturing, and transportation, revolutionizing various sectors.

- Security: Securing IoT devices and the data they generate is crucial, involving measures like encryption, authentication, and secure update mechanisms to counter cyber threats.

IoT has the potential to revolutionize human interactions with the world, creating more connected, automated, and efficient systems. However, it also poses challenges related to privacy, security, and data management. As technology advances, IoT's impact and utilization are expected to grow across different societal and industrial domains.

IOT Cloud : Working of ThingSpeak IOT Cloud

ThingSpeak functions as an open-source platform for the Internet of Things (IoT), allowing users to collect, analyze, and display data from their IoT devices. It simplifies the process of storing and accessing data from connected devices and offers tools for real-time analysis and presentation.

Steps for Using ThingSpeak Cloud

- Channel Creation: In ThingSpeak, a "channel" serves as a crucial component. It acts as a virtual storage space where users can organize and retain data from their IoT devices. Upon creating a channel, users specify parameters such as field numbers, names, and data formats.

- API Key Acquisition: Each ThingSpeak channel is associated with API keys. These keys are essential for authenticating and authorizing access to the channel. Users can find their API keys in their ThingSpeak account settings.

- IoT Device Configuration: IoT devices need to be configured to send data to the designated ThingSpeak channel. This typically involves programming the devices to utilize the ThingSpeak API and providing the correct API key for the assigned channel. Devices can use various communication protocols like HTTP, MQTT, or others supported by ThingSpeak.

- Data Transmission to ThingSpeak: IoT devices regularly send data to the designated ThingSpeak channel using the API key. The data is organized according to the predefined fields within the channel, with each field representing a distinct parameter or sensor reading.

- Data Storage: ThingSpeak stores the incoming data within the channel, making it available for historical analysis. Users can set up the platform to retain data for a specified period.

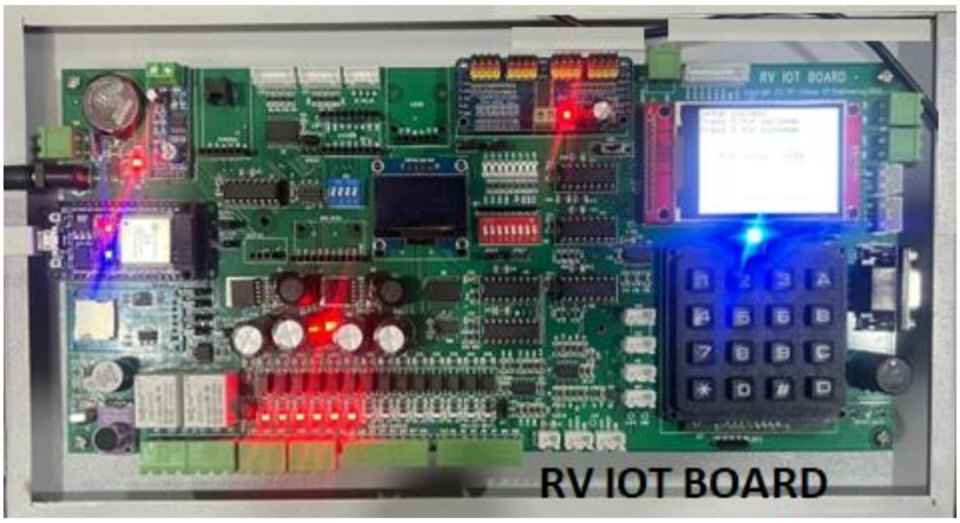
- Analysis and Presentation of Data: ThingSpeak provides integrated tools for users to analyze and present the collected data. Users can create MATLAB-like scripts, known as "ThingSpeak MATLAB Analysis," to perform custom analyses on the data. Additionally, the platform supports the creation of charts, graphs, and maps to visualize data trends.

- Response to Data with ThingSpeak React: Users can define predefined actions or alerts based on the received data. ThingSpeak React allows users to trigger actions when specific conditions are met, enabling a level of automation.

The RV IoT Board

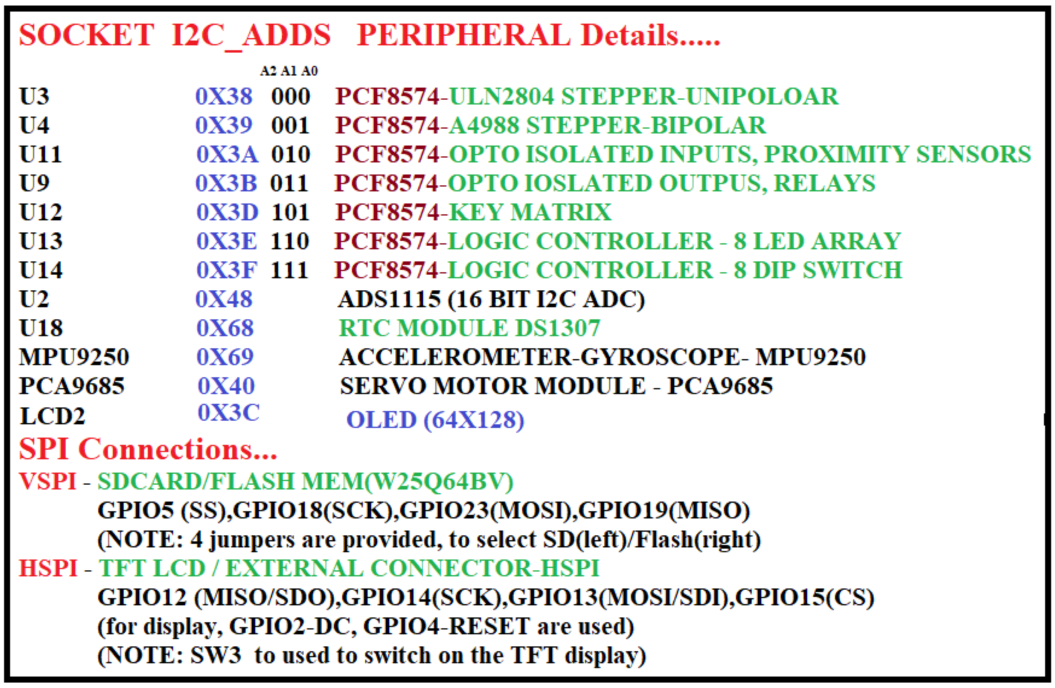
This unique and indigenous board, meticulously crafted and developed by the Department of Computer Science and Engineering at RV College of Engineering, offers a seamless platform for the creation of sophisticated IoT applications tailored for industrial use. At its core lies the ESP-32 Development Board, a microcontroller renowned for its capability to interconnect various sensors integrated onto the chip.

An All-In-One IoT board offers benefits such as a simplified and convenient development platform by integrating various Hardware components, reducing complexity and saving time. It reduces the physical footprint, accelerates Prototyping, and ensures compatibility and reliability of integrated parts. These boards are power-efficient, cost-effective in the long run, and aid in easier debugging and troubleshooting. They facilitate rapid iteration, making them ideal for educational purposes and streamlined deployment, catering to a range of IoT project's needs while considering convenience and integration.



All-in-one IoT boards find diverse applications across industries, including Home automation, Wearables, Environmental monitoring, Industrial automation, Agriculture, Smart cities, Healthcare, Retail, Energy management, Education, and Prototyping. These integrated boards facilitate rapid prototyping of smart devices, systems, and solutions for purposes like real-time data collection, remote monitoring, energy efficiency, and personalized control, making them valuable tools for creating efficient, connected, and innovative IoT applications in various domains.

The IoT board under consideration is built around an ESP32 core-based microcontroller, featuring peripherals like an SD card interface, TFT display, Audio modules, Keypad interface, I2S communication, Microphone input, Motor control interfaces, and an array of sensors, enhancing its versatility and functionality. It provides communication channels such as I2C, SPI, UART which enhance the usability of the All-in-one IoT board.



SOCKET	I2C_ADDS	A2 A1 A0	PERIPHERAL Details.....
U3	0X38	000	PCF8574-ULN2804 STEPPER-UNIPOLOAR
U4	0X39	001	PCF8574-A4988 STEPPER-BIPOLAR
U11	0X3A	010	PCF8574-OPTO ISOLATED INPUTS, PROXIMITY SENSORS
U9	0X3B	011	PCF8574-OPTO IOSLATED OUTPUS, RELAYS
U12	0X3D	101	PCF8574-KEY MATRIX
U13	0X3E	110	PCF8574-LOGIC CONTROLLER - 8 LED ARRAY
U14	0X3F	111	PCF8574-LOGIC CONTROLLER - 8 DIP SWITCH
U2	0X48		ADS1115 (16 BIT I2C ADC)
U18	0X68		RTC MODULE DS1307
MPU9250	0X69		ACCELEROMETER-GYROSCOPE- MPU9250
PCA9685	0X40		SERVO MOTOR MODULE - PCA9685
LCD2	0X3C		OLED (64X128)

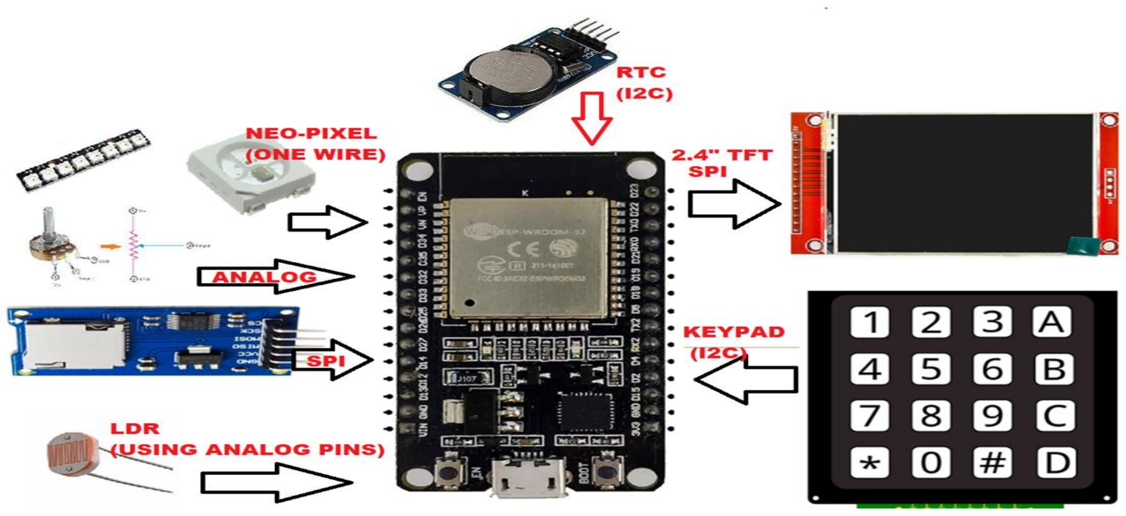
SPI Connections...

VSPI - SDCARD/FLASH MEM(W25Q64BV)
GPIO5 (SS),GPIO18(SCK),GPIO23(MOSI),GPIO19(MISO)
(NOTE: 4 jumpers are provided, to select SD(left)/Flash(right)

HSPI - TFT LCD / EXTERNAL CONNECTOR-HSPI
GPIO12 (MISO/SDO),GPIO14(SCK),GPIO13(MOSI/SDI),GPIO15(CS)
(for display, GPIO2-DC, GPIO4-RESET are used)
(NOTE: SW3 to used to switch on the TFT display)

Sample Data Logger Application Using RV-IOT-Board

The Detailed block diagram of the system as shown in the below figure, indicates the peripherals used in the application.



Features of the System: The sample application serves as a prototype for IoT systems, demonstrating the seamless integration of multiple sensors and peripherals with the ESP32 microcontroller board.

Display of Sensor Values and RTC Information on TFT Display:

- Application featured a TFT display that provided real-time updates of sensor values, including LDR and Potentiometer readings.
- The display showcased RTC information, displaying the current date, time, and day of the week based on the Real-Time Clock (RTC) module.
- NeoPixel can be used for status indications.

Real-time Data Streams from Sensors to ThingSpeak Cloud

- Application facilitated the seamless transmission of real-time sensor data streams, including readings from the Light Dependent Resistor (LDR) and Potentiometer, to the ThingSpeak cloud platform, over WiFi.
- This ensures that the system could continuously monitor environmental conditions and provide timely insights for analysis and decision-making.

```
// Include the libraries
#include <WiFi.h>
#include "ThingSpeak.h"
#include <SPI.h>
#include <TFT_eSPI.h>
#include <Wire.h>
#include "RTClib.h"
#include <Adafruit_NeoPixel.h>
//Define Pin numbers for LDR, Microphone/Potentiameter
const int ldrPin = 39; // Analog pin connected to the LDR output
const int micPin = 36; // Analog pin connected to the microphone output
int  ldrValue = 0, micValue = 0, counter = 0;
// Define WiFi and ThingSpeak Cloud Constants
#define SECRET_SSID "xxxxxxxx"
#define SECRET_PASS "yyyyyyyy”
#define SECRET_CH_ID xxxxxxx
#define SECRET_WRITE_APIKEY "wwwwwwwwwwww"
#define PIN_WS2812B 17  ; Pin No used to connect NeoPixel
// Create Objects of TFT display, WiFi, RTC and NeoPixel
TFT_eSPI tft = TFT_eSPI();
RTC_DS1307 rtc;
Adafruit_NeoPixel ws2812b(1, PIN_WS2812B, NEO_GRB +NEO_KHZ800);
WiFiClient client;
char ssid[] = SECRET_SSID;
char pass[] = SECRET_PASS;
unsigned long myChannelNumber = SECRET_CH_ID;
const char * myWriteAPIKey = SECRET_WRITE_APIKEY;
```

// Defining setup function

```
void setup() {
pinMode(ldrPin, INPUT); pinMode(micPin, INPUT);//Configure Input Pins
Serial.begin(115200); //Initialize Serial Port
   while (!Serial) {; }// wait for serial port to connect.
Wire.begin(); WiFi.mode(WIFI_STA); // Initialize I2C,Wifi
ThingSpeak.begin(client); // Initialize ThingSpeak
tft.init(); tft.setRotation(3); Serial.println("Connecting to TFT...");
tft.fillScreen(TFT_WHITE);tft.setTextColor(TFT_BLACK);tft.setTextSize(2);
if (! rtc.begin()) {
  Serial.println("Couldn't find RTC"); while (1);
}
if (!rtc.isrunning()) {
  Serial.println("RTC is NOT running, let's set the time!");
  rtc.adjust(DateTime(F(_DATE), F(TIME_)));
}
rtc.writeSqwPinMode(DS1307_SquareWave1HZ);
ws2812b.begin();ws2812b.setBrightness(100);
}
```

```
void loop() {  // Connect or reconnect to WiFi
// Connect to the Wifi
if (WiFi.status() != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(SECRET_SSID);
    while (WiFi.status() != WL_CONNECTED) {
        WiFi.begin(ssid, pass);
        Serial.print(".");
        delay(5000);
     }
    Serial.println("\nConnected.");
}
// Read Date & Time from RTC and Update on the TFT
DateTime now = rtc.now();
tft.setTextSize(2); tft.setCursor(2, 4); tft.print(" ");tft.setCursor(2, 4); tft.print(now.hour());
tft.print(":");tft.print(now.minute()); tft.print(":"); tft.print(now.second());
tft.setCursor(2,7); tft.print(" "); tft.print(now.day()); tft.print("-");
tft.print(now.month()); tft.print("-");tft.print(now.year());
// Capture the Values from the Sensors and Display on the TFT
ldrValue = analogRead(ldrPin);
micValue = analogRead(micPin);
tft.setCursor(50, 100); tft.setTextSize(2); tft.print("LDR:"); tft.print(ldrValue);
tft.setCursor(50, 150); tft.print("Mic:");tft.print(micValue);
// Show the Alarming Values using NeoPixel
ws2812b.clear();
if (ldrValue > 2000) {   // If LDR value exceeds…show a change in color in NeoPixel
   ws2812b.setPixelColor(0, ws2812b.Color(255, 0, 0));
   ws2812b.show();
}
else {
   ws2812b.setPixelColor(0, ws2812b.Color(0, 255, 0));
   ws2812b.show();
}
//Update the Cloud as per the time constraints of ThingSpeak Cloud
if (counter % 80 == 0) {  // after a certain time elapse… Update the cloud
    ThingSpeak.setField(1, ldrValue);
    ThingSpeak.setField(2, micValue);

    int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    if (x == 200)     Serial.println("Channel update successful.");
    else   Serial.println("Problem updating channel. HTTP error code " + String(x));
}
counter++;  delay(500);
// Clear the Display and Repeat the operations
tft.fillScreen(TFT_WHITE); tft.setTextColor(TFT_BLACK);tft.setTextSize(2);
}
```

IOT Design Methodology

Follow the following IOT Design Methodology steps, in documenting every project.

Step 1: Purpose & Requirements Specification

Step 2: Process Specification

Step 3: Domain Model Specification

Step 4: Information Model Specification

Step 5: Service Specifications

Step 6: IOT Level Specification

Step 7: Functional View Specification

Step 8: Operational View Specification

Step 9: Device & Component Integration

Step 10 : Application Development

Use the appropriate modules present in the RV IOT Board to realize the following projects.

IOT Projects
Project1: Smart Lighting
Project2: Intrusion Detection System
Project3: Smart Parking
Project4: Weather Monitoring and Weather Reporting Bot
Project5: Smart Irrigation
Project6: Forest Fire Detection

Mini Project 2 : Based on the projects executed, propose any innovative IOT prototype, using the RV-IOT-Board with any cloud platform and Web/Mobile Application.

Project 1: Smart Lighting

Project 2: Intrusion Detection System

Project 3: Smart Parking

Project 4: Weather Monitoring and Weather Reporting Bot

Project 5: Smart Irrigation

Project 6: Forest Fire Detection