EX.NO:	**ARITHMETIC OPERRATIONS ON 8051**
DATE:	

AIM:

To perform arithmetic operations on 8051 micro controller kit.such as Addition ,subtraction, multiplication and division .

APPARATUS REQUIRED:

- Microcontroller kit
- Connecting wire
- Keyboard
- Power supply

ALGORITHM:

Step 1: start

Step 2: Clear display and locate cursor at A,8

Step 3: Move data 1 to Accumulator

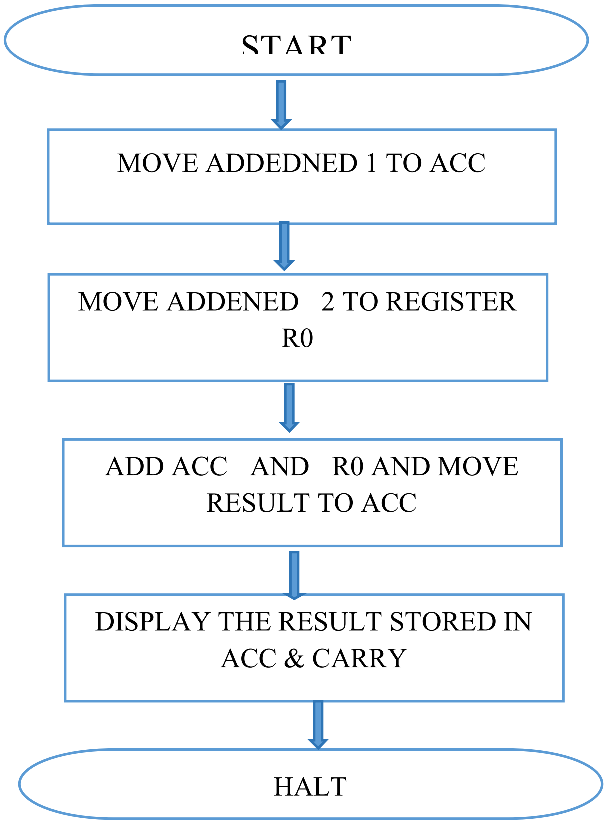
Step 4: Move data 2 to B register

Step 5: AND ,OR that contents of A and B register

Step 6: Display the result in graphical LCD

Step 7: Halt

FLOW CHART:


START
MOVE ADDEDNED 1 TO ACC
MOVE ADDENED 2 TO REGISTER R0
ADD ACC AND R0 AND MOVE RESULT TO ACC
DISPLAY THE RESULT STORED IN ACC & CARRY
HALT

OUTPUT:

Addened 1 in 9500h - 50H

Addened 2 in 9510h - 10H

Result : -60H

ADDITION OF TWO 8-BIT NUMBES:

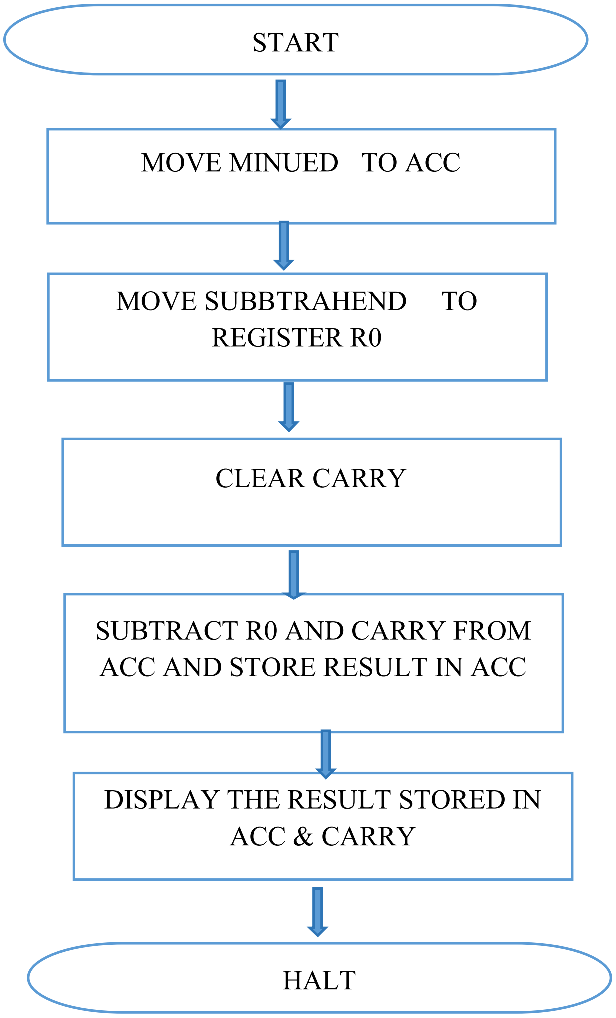
Input: Acc - Addend 1

R0   - Addend 2

Program:

```
8700                          .ORG       OX8700
8700  12    10    00          LCALL      1000 ;Clear Display
8703  90    00    00          MOV        DPTR, #0000
8706  12    10    30          LCALL      1030     ;Locate he cursor
8709  74    50                MOV        A,#50          ;Move Addend 1 to Acc
870B  78    10                MOV        R0,#10         ;Move Addend to R0
870D  28                      ADD        A,R0           ;Add
870D  F8                      MOV        R0,A
870F  74    01                MOV        R0,A
8710  40    01                JC         8714
8713  E4                      CLR        A
8714              ADD_CONT:
8715  12    10    50          LCALL      1050           ;Display   the result
8716  E8                      MOV        A,RO
8717  12    10    50          LCALL      1050           ;Display the result
871B  80    FE                SJMP       871B
```

FLOW CHART:


START
MOVE MINUED TO ACC
MOVE SUBBTRAHEND TO REGISTER R0
CLEAR CARRY
SUBTRACT R0 AND CARRY FROM ACC AND STORE RESULT IN ACC
DISPLAY THE RESULT STORED IN ACC & CARRY
HALT

OUTPUT:

Minuend in 9500h	-	50H
Subtrahend in 9510h	-	10H
Result :	**-**	**40H**

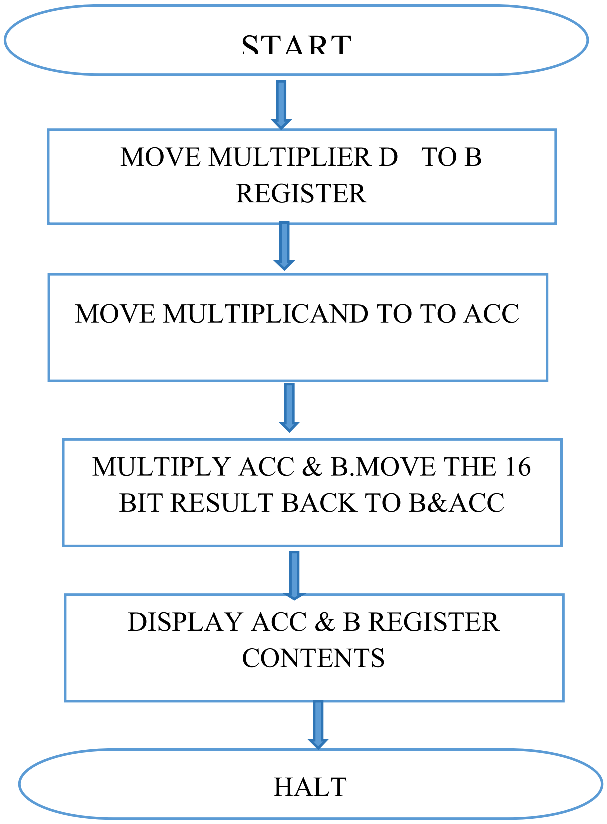
SUBTRACTION OF TWO 8-BIT NUMBERS

```
Input:      Acc   -     Minuend
            R0    -     subtracthend
```

Program:

```
8750                  .     ORG      OX8750
8750  12    10    00        LCALL    1000         ;Clear Display
8753  74    50              MOV      A,#50       ;Move Minuend   to Acc
8755  78    10              MOV      R0,#10      ;Move Subtrahend to R0
8757  C3                    CLR      C           ;Clear carry
8758  98                    SUBB     A,R0        ;Subtract
8759  F8                    MOV      R0,A
875A  74    01              MOV      A,#01
875C  40    01              JC       875F
875E  E4                    CLR      A
875F              SUB_CONT:
875F  12    10    50        LCALL    1050        ;Display   the result
8762  E8                    MOV      A,RO
8763  12    10    50        LCALL    1050        ;Display the result
8766  80    FE              SJMP     8766
```

FLOW CHART:


START
MOVE MULTIPLIER D TO B REGISTER
MOVE MULTIPLICAND TO TO ACC
MULTIPLY ACC & B.MOVE THE 16 BIT RESULT BACK TO B&ACC
DISPLAY ACC & B REGISTER CONTENTS
HALT

OUTPUT:

Multiplicand in 8803H - 50H

Multiplier in 8806H - 10H

Result : - 500H

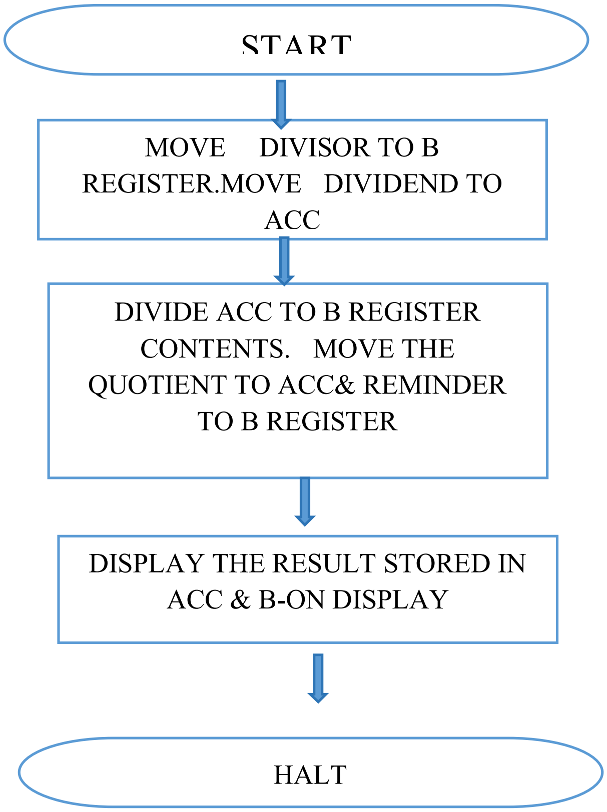
MULTIPLICATION OF TWO 8-BIT NUMBERS

```
Input  :    Acc   -     Multiplicand
            B     -     Multiplier
```

Program :

```
8800                        .ORG      Ox8800
8800  12    10    00        LCALL     1000      ;Clear display
8803  75    F0    50        MOV       B,#50     ;Move multiplier to B-register
8806  74    10              MOV       A,#10     ;Move multiplicant to Acc
8808  A4                    MUL       AB        ;Multipy Acc and B contents
8809  85    F0    83        MOV       DPH,B
880C  F5    82              MOV       DPL,A
880E  12    10    60        LCALL     1060      ;display   the result(DPTR contents)
8811  80    FE              SJMP      8811
```

FLOW CHART:


START
MOVE DIVISOR TO B REGISTER.MOVE DIVIDEND TO ACC
DIVIDE ACC TO B REGISTER CONTENTS. MOVE THE QUOTIENT TO ACC& REMINDER TO B REGISTER
DISPLAY THE RESULT STORED IN ACC & B-ON DISPLAY
HALT

OUTPUT:

Divosor in 8853H - 10H

Dividedend in 8856H -510H

Result : - 0005H

DIVISION OF TWO 8-BIT NUMBERS

Input : Acc - Dividend

B - Divisor

Program:

```
8850                      .ORG     0x8850
8850  12   10   00        LCALL    1000      ;Clear   Display
8853  75   F0   10        MOV      B,#10     ;Move divisor to B register
8856  74   50             MOV      A,#50     ;Move dividend to Acc
8858  84                  DIV      AB        ;Divide Acc by B content
8859  85   F0   83        MOV      DPH,B
885C  F5   82             MOV      DPL,A
885E  12   10   60        LCALL    1060      ;Display the resut
8861  80   FE             SJMP     8861
```

RESULT:

Thus the Arithmetic operation are performed using 8051 kit and the output executed successfully

EX.NO:	
DATE:	**LOGICAL OPERRATIONS ON 8051**

AIM:

To write and perform program for logical AND and OR operation using 8051 Microcontroller Kit.

APPARATUS REQUIRED:

- 8051 Microcontroller Kit
- Keyboard
- Power supply

ALGORITHM:

Step 1: start

Step 2: Clear display and locate cursor at A,8

Step 3: Move data 1 to Accumulator

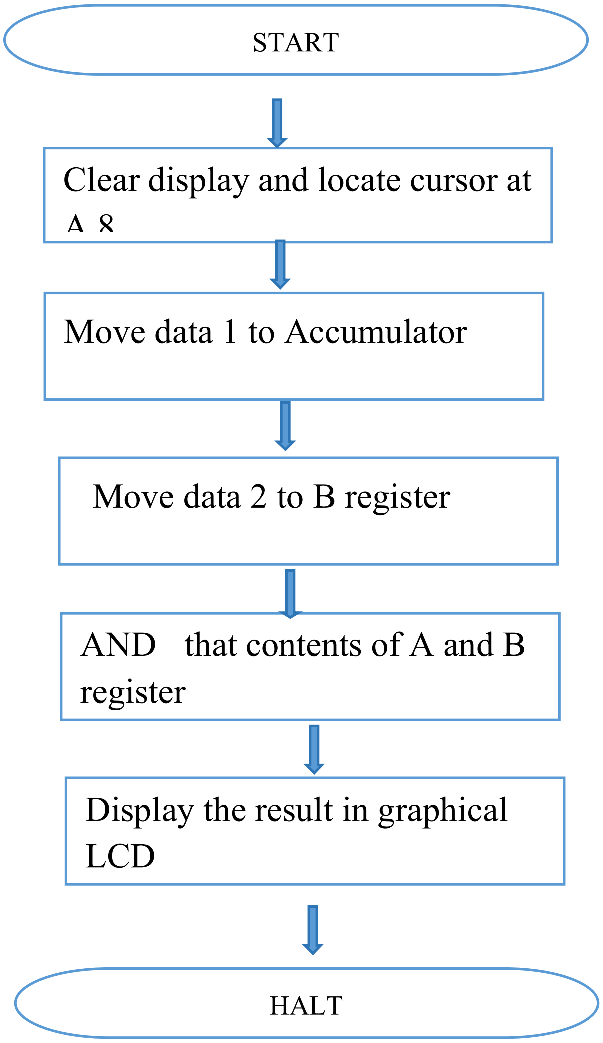
Step 4: Move data 2 to B register

Step 5: AND ,OR that contents of A and B register

Step 6: Display the result in graphical LCD

Step 7: Halt

FLOW CHART:


START
Clear display and locate cursor at 4 8
Move data 1 to Accumulator
Move data 2 to B register
AND that contents of A and B register
Display the result in graphical LCD
HALT

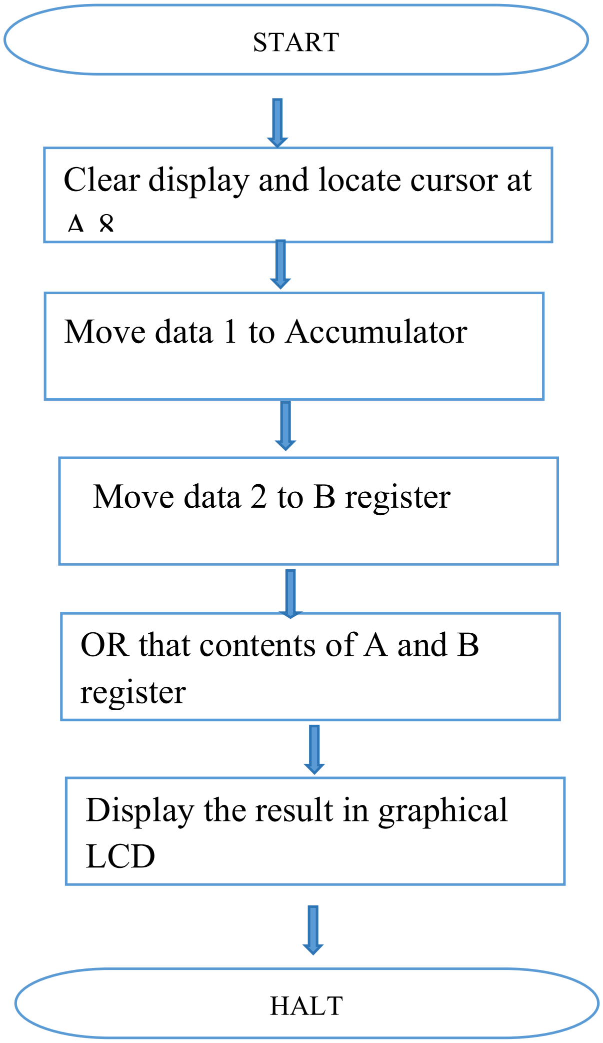
OUTPUT:

INPUT		OUTPUT
ADDRESS	**DATA**	
9159 **915B**	**F0** **EC**	**FC**

PROGRAM:

ADDRESS	PROGRAM	COMMENT
9100	L CALL 1000	Clear display
9103	MOV DPTR, # 408	Locate cusror
9106	L CALL 1030	
9109	MOV A,#FO	Move data 1 to accumulator
9108	MOV B,#EC	Move data 2 to B-register
910E	AND, A,B	AND register A and B
9110	LCALL 1050	Display result
9113	SJMP 9113	halt

FLOW CHART:


START
Clear display and locate cursor at
A 8
Move data 1 to Accumulator
Move data 2 to B register
OR that contents of A and B register
Display the result in graphical LCD
HALT

OUTPUT:

INPUT		OUTPUT
ADDRESS	DATA	
9109	F0	
9108	EC	E0

PROGRAM:

ADDRESS	PROGRAM	COMMENT
9150	L CALL 1000	Clear display
9153	MOV DPTR, # 408	Locate cusror
9156	L CALL 1030	
9159	MOV A,#FO	Move data 1 to accumulator
915B	MOV B,#EC	Move data 2 to B-register
915E	OR, A,B	OR register A and B
9160	LCALL 1050	Display result
9163	SJMP 9163	halt

RESULT:

Thus the logical operations are perfomed and executed the output successfully.

EX.NO:	**GENERATION OF SQUARE WAVEFORM**
DATE:	

AIM:

To generate a square waveform using 8051 microcontroller kit

APPARATUS REQUIRED:

- 8051 Microcontroller Kit
- Keyboard
- Power supply

ALGORITHM:

Step 1: start

Step 2: Cut DAC port address in DPTR

Step 3: Data set DAC output to 0 level

Step 4: Send data to DAC and off time delay

Step 5: Later ,data to set DAC output to 1 level

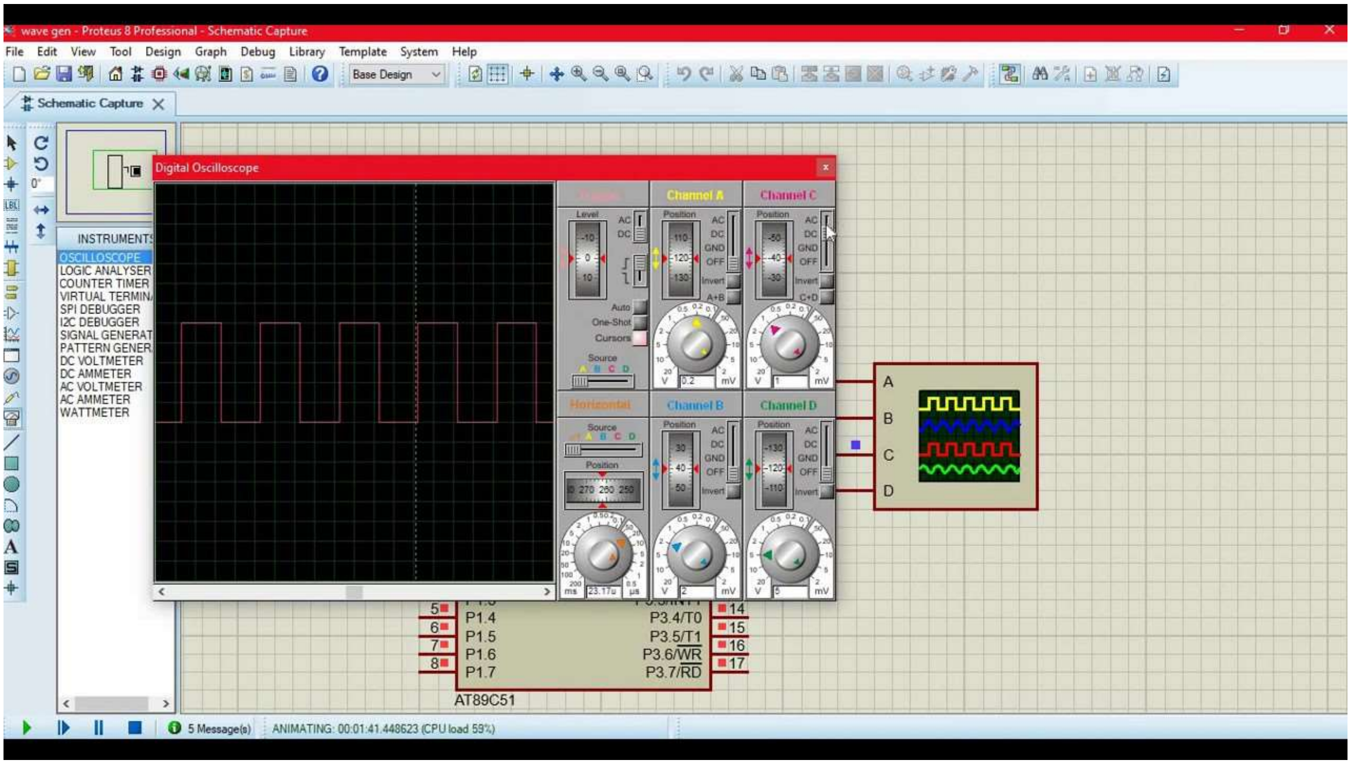
Step 6: Send data DAC & on time delay

Step 7: Halt

OUTPUT:

TYPE	AMPLITUDE	TIME
Square wave form	2*5 volts	2*0.5

WAVEFORM:



wave gen - Proteus 8 Professional - Schematic Capture

5 Message(s)

ANIMATING: 00:01:41.448623 (CPU load 53%)

PROGRAM:

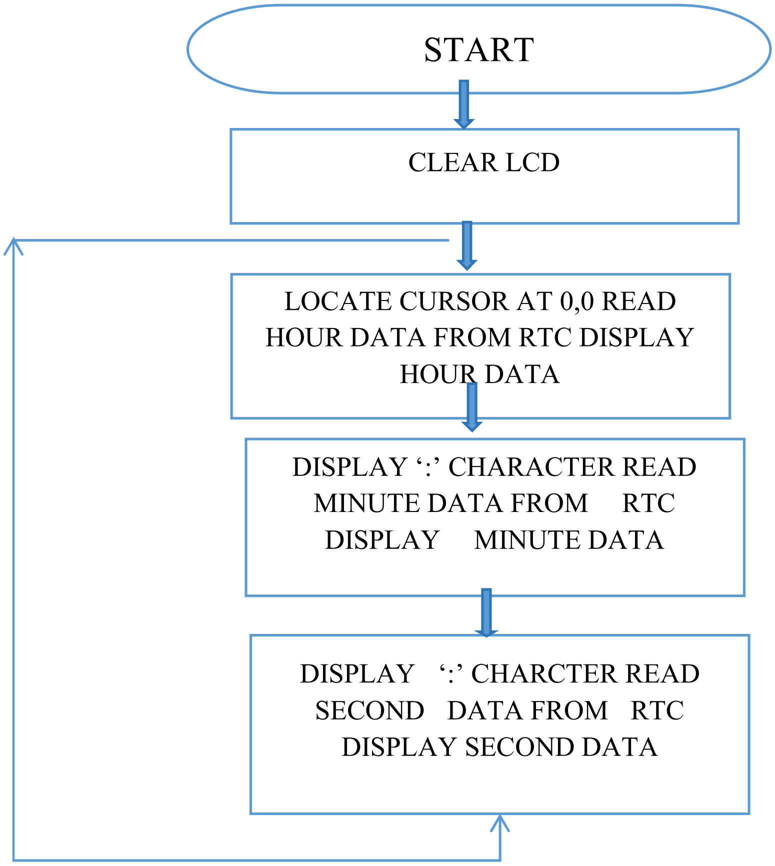
```
                    SQUARE_DAC:
8200  90   FF   CO          MOV      DPTR,#FFCO     ;Get DAC port address in
                                                    DPTR
8203  74   00               MOV      A,#00       ;Data   to set DAC output to 0 level
8205  F0                    MOVX     #DPTR,A     ;Send data to DAC
8206  78   FF               MOVX     R0,#FF      ;off time delay
8208  08   FE               DJNE R0,8208
820A  74   FF               MOV      A,#FF       ;data ot set DAC output to 1 level
820C  F0                    MOVX     #DPTR,A     ;send data to DAC
8200  78   FF               MOV      R0,#FF      ;on time delay
820F  D8   FE               DJNZ R0,820F
8211  80   F0               SJMP 8203
```

RESULT:

Thus the program to generate square waveform using 8051 microcontroller was Perfomed successfully.

FLOW CHART:

FLOW CHART:


START
CLEAR LCD
LOCATE CURSOR AT 0,0 READ HOUR DATA FROM RTC DISPLAY HOUR DATA
DISPLAY ‘:’ CHARACTER READ MINUTE DATA FROM RTC DISPLAY MINUTE DATA
DISPLAY ‘:’ CHARCTER READ SECOND DATA FROM RTC DISPLAY SECOND DATA

EX.NO:	**DESIGN OF DIGITAL CLOCK USING TIMERS/COUNTERS**
DATE:	

AIM:

To design a digital clock using Times/counters in 8051 microcontroller.

APPARATUS REQUIRED:

- 8051 Microcontroller Kit
- Keyboard
- Power supply

ALGORITHM:

Step 1: start

Step 2: Read the hours data from RTC & display

Step 3: Display characters

Step 4: Read minutes data from RTC

Step 5: Display Accumualator and Display character ‘:’

Step 6: Read the seconds data from RTC and dislay the Accumulator

Step 7: Halt

PROGRAM:

DIGITAL_CLOCK

```
8C00                         .ORG      0x8C00
8C00  12    10    00         LCALL     1000
8C03              RPT_PROCESS_RTC DEMO
8C03  D2    00               SETB      00
8C05  C2    01               CLR       02
```

OUTPUT:

```
HH:MM:SS
```

```
8C09  90    00    00          MOV       DPTR,#0000
8C0C 12     10    30          LCALL     10A0
8C0F  74    04                MOV       A,#04
8C11  12    10    A0          LCALL     10A0      ;Reads hours data from RTC
8C14  12    10    50          LCALL     1050      ;Display
8C17  74    3A                MOV       A,#3A
8C19  12    10    40          LCALL     1040      ;Display character ‘:’
8C1C 74     03                MOV       A,#03
8C1E 12     10    A0          LCALL     10A0      ;Read minutes data from RTC
8C21  12    10    50          LCALL     1050      ;Display Accuulator
8C24  74    3A                MOV       A,#3A
8C26  12    10    40          LCALL     1040      ;Display character ‘:’
8C29  74    02                MOV       A,#02
8C2B 12     10    A0          LCALL     10A0      ;Read seconds from RTC
8C2E 12     10    50          LCALL     1050      ;Display Accumulator
8C31  12    10    36          LCALL     8C36
8C34  80    CD                SJMP 8C03
8C36              DISPLAY_DELAY
8C36  C0    83                PUSHDPH
8C36  C0    82                PUSHDPL


8C3A C0     E0                PUSHACC
8C3C 90     80    00          MOV       DPTR,#8000
8C3F              WAIT_DISPLAY_DELAY
8C3F  A3                      INC       DPTR
8C40  E5    83                MOV       A,DPH
8C42  45    82                ORL       A,DPL
```

```
8C44  70      F9          JNZ       8C3F
8C46  D0      E0          POP       ACC
8C48  D0      82          POP       DPL
8C4A D0       83          POP       DPH
8C4C 22                   RET
```

RESULT:

Thus the program to generate digital clock using timers/counters uisng 8051 microcontroller was perfomed successfully

EX.NO:	**INTERFACING ADC AND DAC**
DATE:	

AIM:

To develop a C-Language program for reading an on-chip ADC, convert into decimal and to display it in PC and to generate a square wave depending on this ADC reading. The ADC input is connected to any analog sensor/ on board potentiometer.

APPARATUS REQUIRED:

- Personal computer
- VSK -SCM devolopment board
- IAR IDE software
- Flash Loader Demostrator
- CRO

ALGORITHM:

Step 1: Write the C program for the given task and execute using IAR

Step 2: Follow the steps 1 of How to create a New project

Step 3: Type the below code and save it with the name (***anyname.c***)

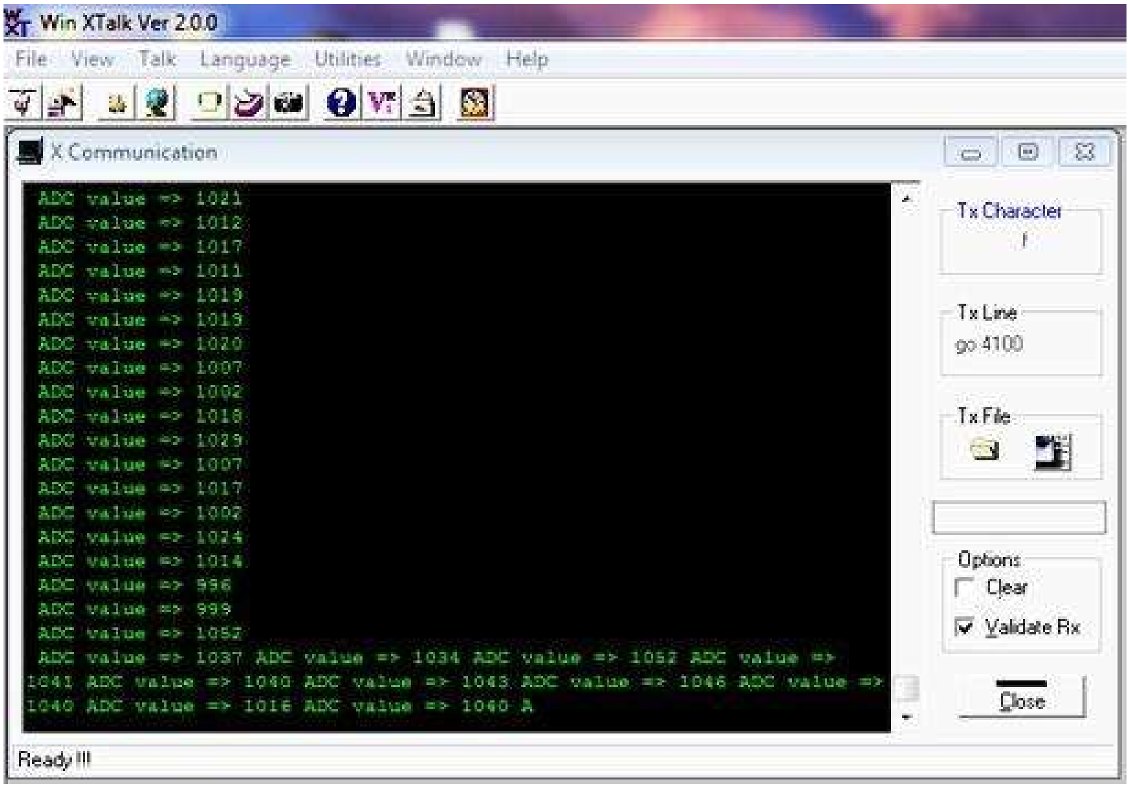
Step 4: . Follow the procedures in ***How to Download a Code to Our Controller to download your code***

Step 5: Follow the steps 2 to 6 of How to create a New Project to add the necessary file, compile and build the program

Step 7: Halt

OUTPUT:

ADC:



```
 ADC value => 1021
 ADC value => 1012
 ADC value => 1017
 ADC value => 1011
 ADC value => 1019
 ADC value => 1019
 ADC value => 1020
 ADC value => 1007
 ADC value => 1002
 ADC value => 1018
 ADC value => 1029
 ADC value => 1007
 ADC value => 1017
 ADC value => 1002
 ADC value => 1024
 ADC value => 1014
 ADC value => 996
 ADC value => 999
 ADC value => 1052
 ADC value => 1037 ADC value => 1034 ADC value => 1052 ADC value =>
1041 ADC value => 1040 ADC value => 1043 ADC value => 1046 ADC value =>
1040 ADC value => 1016 ADC value => 1040 A
```

Tx Character: !

Tx Line: go 4100

□ Clear

☑ Validate Rx

Ready !!!

PROGRAM: ADC

```
#include "stm32f4xx.h"
#include <stdio.h>
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_adc.h"
int ConvertedValue = 0;
//Converted value read from ADC1 void adc_configure()
{
//Clock configuration
RCC->APB2ENR |= 1<<10;
//The ADC3 is connected the APB2 peripheral bus
thus we will use its clock source
RCC->AHB1ENR |= 1<<0;
//Clock for the ADC port!! Do not forget about this
one ;)
//Analog pin configuration
GPIOA->MODER |=0x0000000F; //analog mode PA1,PA2
GPIOA->OSPEEDR = 0xFFFFFFFF; GPIOA->PUPDR = 0x00000000;
/* ADC configuration */
ADC3->CR1 = 0x00000000;
//scan mode disable,12-bit resolution.
ADC3->CR2 = 0x00000002;
//data right alignment, continuous conversion
mode. ADC3->SQR1 = 0x00000000; //single mode conversion
ADC3->CR2 |= 0x00000001;
//ADC enable
```

```
ADC3->SMPR2 = 0x00000030;
//ADC3 channel-1 with 144 cycles
ADC3->SQR3 = 0x00000001
;//rank1 to ADC3 channel-1
}
int adc_convert()
{
ADC_ SoftwareStartConv(ADC3);
//Start the conversion
while(!ADC_GetFlagStatus(ADC3, ADC_FLAG_EOC));
//Processing the conversion
return ADC_GetConversionValue(ADC3);
//Return the converted data
}
void USART2_config()
{
RCC->AHB1ENR |= 1 << 0;
//clock to portA
RCC->APB1ENR |= 1 <<17; //clock to USART2
GPIOA->MODER |= 0x000000A0;

//alternate function mode(PA2,PA3)
GPIOA->AFR[0] |= 0x00007700;
//USART2 AF
USART2->BRR = 0x16D;
//115200 baud rate
USART2->CR3 = 0x0000;
```

```
USART2->CR2 = 0x000;
USART2->CR1 = 0x200C;
}
int main(void){
USART2_config();
adc_configure();
//Start configuration
while(1)
{
ConvertedValue = adc_convert();
//Read the ADC converted value
printf("\n ADC value => %d",ConvertedValue);
//print the ADC value
}}
int putchar(int data)
{
USART2->DR = (data & 0x01FF);
/* Loop until the end of transmission */
while((USART2->SR & 0x40) ==0)
{}
return data;
```

DAC

```
#include "stm32f4xx.h"

#include "stm32f4xx_dac.h"

#include "stm32f4xx_gpio.h"

#include "stm32f4xx_rcc.h"

#include "stm32f4xx_tim.h"

#include "stm32f4xx_syscfg.h"

#define sine_wave_gk

//#define triangular_wave_gk

//#define square_wave_gk

//#define sawtooth_wave_gk

unsigned

intsine_wave[200]={2048,2112,2176,2240,2304,2368,2431,2494,2557,2619,2680,2741,2801,2
860,29

19,2977,3034,3090,3144,3198,3251,3302,3352,3401,3449,3495,3540,3583,3625,3665,3704,37
41,377

6,3809,3841,3871,3900,3926,3951,3973,3994,4013,4030,4045,4058,4069,4078,4085,4090,409
3,4095

,4093,4090,4085,4078,4069,4058,4045,4030,4013,3994,3973,3951,3926,3900,3871,3841,3809
,3776,

3741,3704,3665,3625,3583,3540,3495,3449,3401,3352,3302,3251,3198,3144,3090,3034,2977,
2919,2

860,2801,2741,2680,2619,2557,2494,2431,2368,2304,2240,2176,2112,2048,1984,1920,1856,1
792,17

28,1665,1602,1539,1477,1416,1355,1295,1236,1177,1119,1062,1006,952,898,845,794,744,69
5,647,6

01,556,513,471,431,392,355,320,287,255,225,196,170,145,123,102,83,66,51,38,27,18,11,6,3,1
,3,6,11

,18,27,38,51,66,83,102,123,145,170,196,225,255,287,320,355,392,431,471,513,556,601,647,6
95,744,
```

```
794,845,898,952,1006,1062,1119,1177,1236,1295,1355,1416,1477,1539,1602,1665,1728,1792
,1856,

1920,1984};

int main(void)

{

static int i;

int j;

GPIO_InitTypeDef

GPIO_InitStructure;/* DMA1 clock enable */

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);

/* GPIOA clock enable (to be used with DAC) */

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

/* DAC Periph clock enable */

RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);

/* DAC channel 1 & 2 (DAC_OUT1 = PA.4)(DAC_OUT2 = PA.5) configuration */

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;

GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

GPIO_Init(GPIOA, &GPIO_InitStructure);

//TIM6_Config();

while (1)

{j=0;

#if defined sine_wave_gk

for(i=0;i<200;i++)

{

DAC->DHR12R2 = sine_wave[i];

DAC_Cmd(DAC_Channel_2, ENABLE);

}
```

```
#else if defined triangular_wave_gk
for(i=0;i<=4095;i++)
{
DAC->DHR12R2 = i;
DAC_Cmd(DAC_Channel_2, ENABLE);
}
for(i=4095;i>=0;i--)
{
DAC->DHR12R2 = i;
DAC_Cmd(DAC_Channel_2, ENABLE);
}
#else if defined square_wave_gk
while((j++) < 100000)
DAC->DHR12R2 = 0xFFF;
while((j--) > 0)
DAC->DHR12R2 = 0x000;
DAC_Cmd(DAC_Channel_2, ENABLE);
#else if defined sawtooth_wave_gk
for(i=0;i<4095;i++)
{
}
#endif
DAC->DHR12R2 = i;
DAC_Cmd(DAC_Channel_2, ENABLE);
//DAC_Ch2_TriangleConfig();
```

RESULT:

Thus, the C program is written and executed by verifying the output using ARM processor.

EX.NO:	INTERFACING LED ,LCD AND KEYBOARD
DATE:	

AIM:

To implemnet interfacing LED , LCD and KEYBOARD using ARMv7 CPUlater

APPARATUS REQUIRED:

- Personal computer
- CPU-later

ALGORITHM:

Step 1:Open the CPULator website.

Step 2:write the code

Step 3:Compile and build the program by following the steps provided in CPULator.

Step 4:Run the simulation,

PROGRAM:

LED

```
#include <stdint.h>
// Define the memory-mapped I/O address for the LED
#define LED_BASE 0xFF200000   // Replace with your actual LED base address
#define DELAY_COUNT 1000000   // Adjust for appropriate delay


// Function to create a delay
void delay(volatile uint32_t count) {
    while (count--) {
        // Do nothing, just wait
    }
}
```

```
int main() {
    // Pointer to the LED register
    volatile uint32_t *led = (volatile uint32_t *)LED_BASE;

    while (1) {
        // Turn LED on
        *led = 0xFF;
        delay(DELAY_COUNT);

        // Turn LED off
        *led = 0x00;
        delay(DELAY_COUNT);
    }

    return 0;
}
```

LCD:

```
#include <stdint.h>
// Define the memory-mapped I/O address for the LCD
#define LCD_BASE 0xFF210000   // Replace with your actual LCD base address
#define DELAY_COUNT 1000000   // Adjust for appropriate delay
// Function to create a delay
void delay(volatile uint32_t count) {
    while (count--) {
        // Do nothing, just wait
    }
}
```

```
// Function to write a command to the LCD
void lcd_write_command(uint8_t command) {
    volatile uint32_t *lcd = (volatile uint32_t *)LCD_BASE;
    *lcd = (command & 0xFF) | 0x100; // Command mode
    delay(DELAY_COUNT);
}
// Function to write data to the LCD
void lcd_write_data(uint8_t data) {
    volatile uint32_t *lcd = (volatile uint32_t *)LCD_BASE;
    *lcd = (data & 0xFF) | 0x200; // Data mode
    delay(DELAY_COUNT);
}
// Function to initialize the LCD
void lcd_init() {
    lcd_write_command(0x38); // Function set
    lcd_write_command(0x0C); // Display ON
    lcd_write_command(0x01); // Clear display
    delay(DELAY_COUNT);
}
// Function to write a string to the LCD
void lcd_write_string(const char *str) {
    while (*str) {
        lcd_write_data(*str++);
    }
}
int main() {
    // Initialize the LCD
    lcd_init();

    // Write a string to the LCD
```

```
    lcd_write_string("Hello, World!");
   while (1) {
        // Infinite loop to keep the program running
    }
    return 0;
}
```

KEYBOARD

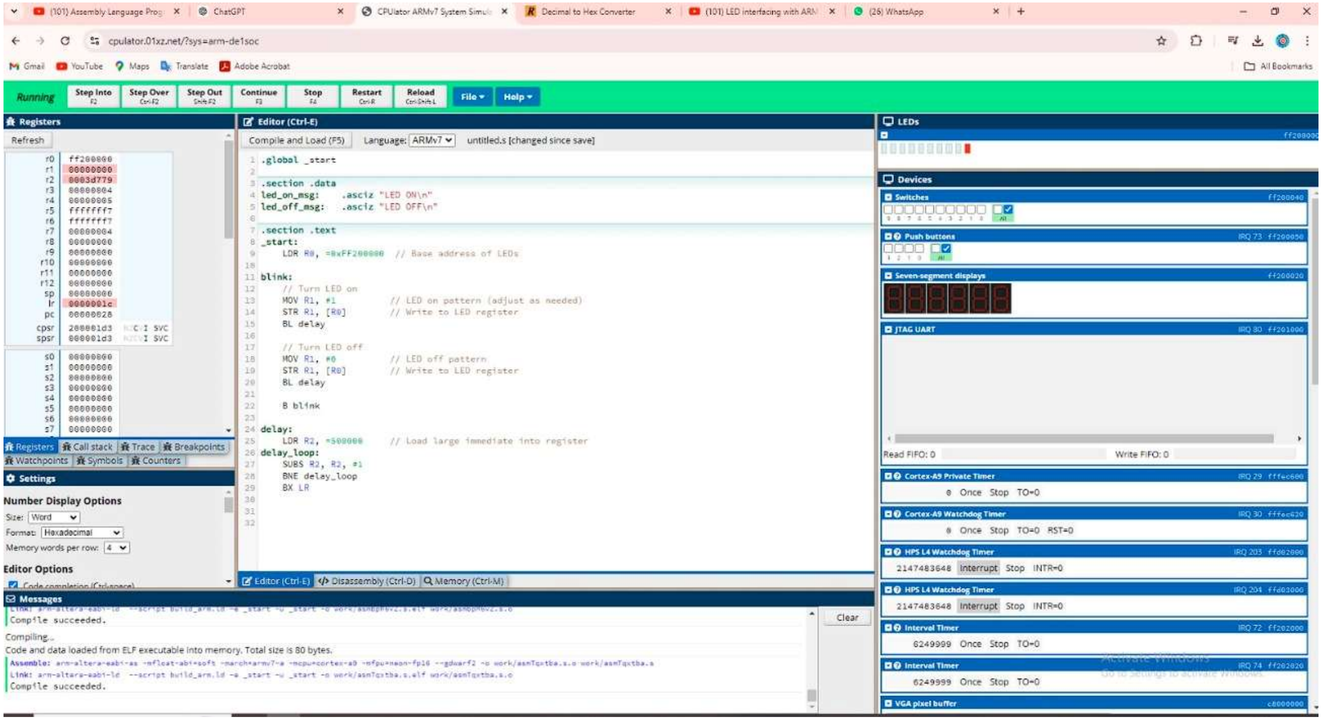
```
#include <stdint.h>
#include <stdio.h>
// Define the memory-mapped I/O address for the keyboard
#define KEYBOARD_BASE 0xFF220000   // Replace with your actual keyboard base address
#define DELAY_COUNT 1000000   // Adjust for appropriate delay
// Function to create a delay
void delay(volatile uint32_t count) {
    while (count--) {
        // Do nothing, just wait
    }
}

// Function to read a key from the keyboard
uint8_t keyboard_read_key() {
    volatile uint32_t *keyboard = (volatile uint32_t *)KEYBOARD_BASE;
    return (uint8_t)(*keyboard & 0xFF); // Read key press
}
// Function to initialize the keyboard
void keyboard_init() {
    // If any initialization is needed, it can be added here
}
```

```
// Function to decode a key press (simplified)
char decode_key(uint8_t scancode) {
    switch (scancode) {
        case 0x1C: return 'A';
        case 0x32: return 'B';
        case 0x21: return 'C';
        case 0x23: return 'D';
        case 0x24: return 'E';
        // Add more scancode mappings as needed
        default: return '?'; // Unknown key
    }
}
int main() {
    // Initialize the keyboard
    keyboard_init();
    while (1) {
        // Read a key from the keyboard
        uint8_t scancode = keyboard_read_key();
        char key = decode_key(scancode);
        // Print the key to the console
        if (key != '?') {
            printf("Key pressed: %c\n", key);
        }
       delay(DELAY_COUNT);
    }
    return 0;
}
```

OUTPUT:

OUTPUT:

LCD, LED and KEYBOARD:



```
.global _start

.section .data
led_on_msg:    .asciz "LED ON\n"
led_off_msg:   .asciz "LED OFF\n"

.section .text
_start:
    LDR R0, =0xFF200000  // Base address of LEDs

blink:
    // Turn LED on
    MOV R1, #1          // LED on pattern (adjust as needed)
    STR R1, [R0]        // Write to LED register
    BL delay

    // Turn LED off
    MOV R1, #0          // LED off pattern
    STR R1, [R0]        // Write to LED register
    BL delay

    B blink

delay:
    LDR R2, =500000     // Load large immediate into register
delay_loop:
    SUBS R2, R2, #1
    BNE delay_loop
    BX LR
```

```
  r0  ff200000
  r1  00000000
  r2  0003d779
  r3  00000004
  r4  00000005
  r5  fffffff7
  r6  fffffff7
  r7  00000004
  r8  00000000
  r9  00000000
 r10  00000000
 r11  00000000
 r12  00000000
  sp  00000000
  lr  0000001c
  pc  00000028
cpsr  200001d3  C I SVC
spsr  000001d3    I SVC

  s0  00000000
  s1  00000000
  s2  00000000
  s3  00000000
  s4  00000000
  s5  00000000
  s6  00000000
  s7  00000000
```

Number Display Options

Size: Word

Format: Hexadecimal

Memory words per row: 4

Editor Options

☑ Code completion (Ctrl-space)

```
Compile succeeded.
Compiling...
Code and data loaded from ELF executable into memory. Total size is 80 bytes.
Assemble: arm-altera-eabi-as -mfloat-abi=soft -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 --gdwarf2 -o work/asmTqxtba.s.o work/asmTqxtba.s
Link: arm-altera-eabi-ld  --script build_arm.ld -e _start -u _start -o work/asmTqxtba.s.elf work/asmTqxtba.s.o
Compile succeeded.
```

```
Cortex-A9 Private Timer       0  Once  Stop  TO=0
Cortex-A9 Watchdog Timer      0  Once  Stop  TO=0  RST=0
HPS L4 Watchdog Timer  2147483648  Interrupt  Stop  INTR=0
HPS L4 Watchdog Timer  2147483648  Interrupt  Stop  INTR=0
Interval Timer          6249999  Once  Stop  TO=0
Interval Timer          6249999  Once  Stop  TO=0
```

RESULT:

Thus the LCD,LED and KEYBOARD are interfaced with ARMv7 in CPU-lator has been executed successfully.