SRH University Heidelberg

Professional Technical Report

Frequency Measurment using MSP430FR4133

Prepared by **Sanjay Sahani(11011726)**

Under guidance of **: Prof.Ing.Vimala Baure**

Contents

Chapter 1

Abstract

Frequency, is to describe the frequency of periodic movement of the physical quantity, easy to transmit, anti-interference ability, many engineering measurements are related to frequency, such as sound frequency, mechanical vibration, measurement speed.

With the advance of the digital process, the traditional frequency-based circuit and timing circuit design of the frequency meter in the speed, accuracy and other aspects of the trend gradually, but was based on ultra-large-scale integrated circuit technology developed from the digital frequency meter to replace it on a large scale. Digital frequency meter can not only solve the problem of low efficiency of analog circuit information processing and transmission, but also improve the accuracy of frequency measurement.

This application report describes a frequency-measuring principle utilizing the ultra low-power microcontroller MSP430FR4133 and the corresponding C language procedure. The combination of two methods for measuring cycle and frequency enables that the procedure has a high measuring accuracy in the high frequency band.

Chapter 2

Introduction

Frequency measurement is a very important application of both counting and timing. Fundamentally, frequency measurement is a measure of how many times something happens within a certain known period, as illustrated in Figure 2.1 . The use can be as diverse as how many counts are received per minute in a counter, how many cycles per second (i.e. Hertz) there are in an electronic or acoustic measurement, or how many wheel revolutions there are per unit of time in a speed measurement. Both a counter and a timer are needed, the timer to measure the reference time and the counter to count the number of events within that time.

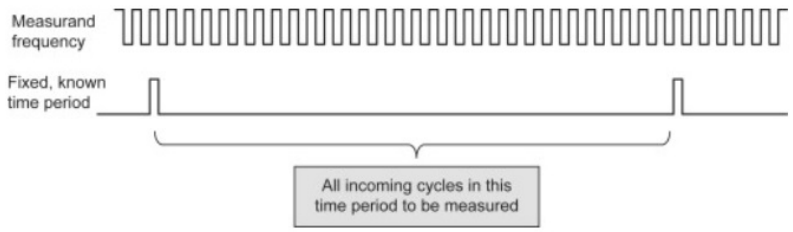

Figure 2.1: Principal for Frequency Measurement

Frequency measurement is required as part of those devices that convert the measured physical quantity into a frequency change, such as a variable reluctance velocity transducer, stroboscopes, vibrating-wire force sensor, resonant wire pressure sensor, turbine flowmeter, Doppler-shift ultrasonic flowmeter, transit-time ultrasonic flow meter, vibrating level sensor, quartz moisture meter, and quartz thermometer. In addition, the output relationship in some forms of a.c. bridge circuits used for measuring inductance and capacitance requires accurate measurement of the bridge excitation frequency.A digital counter/timer is the most common instrument for measuring frequency.

Chapter 3

Project Description

3.1 Project Specification

3.1.1 Objective

- Measurement of frequency of digital signal in range of 1Khz to 100MHz.
- Display the measured frequency on LCD.
- Generate a clock source for other supporting device.

3.1.2 Component Used in MSP430FR4133 Development Kit

- TimerA0
- Clock Sources i.e ACLK,SMCLK
- LCD
- GPIO

3.2 Block Diagram

Figure 3.1 shows the block diagram of our application .TimerAO is used to measure the frequency. Unknown input signal is given to Pin 1.6.At Pin 8.1 the ACLK is taken out to give as sample input to measure the frequency. LCD is connected to Pin 4.1 and 4.2,Measured frequency is displayed in LCD.

Following approach to measure frequency is followed:

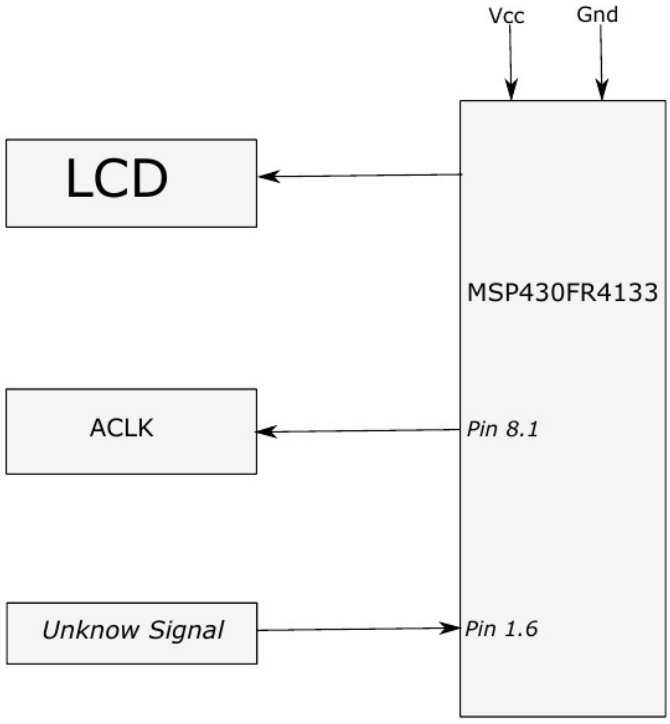

Figure 3.1: Block Diagram for Frequency Measurement

- Configure timer in Capture mode and capture in Rising edge
- CCR2 value is used to capture the count .
- Start the Timer and capture the value of CCR2 in variable Starttime in first interrupt.
- Capture the value of CCR2 and Minus this value from Starttime . This is the no of count in 2 Captures .
- Divide this value By two gives count in one cycle .
- frequency = 1/2*No of count in one cycle .
- As SMCLK is in microsecond frequency = 1000000/ 2*No of count in one cycle.
- Stop the CCR2 interrupt .

3.3 Project Design

```
// Library Declaration

#include "driverlib.h"

// Global Variable Declaration

void main(void)
{
    //Turn of the Watchdog Timer

    // TA0.CCI2A input capture pin, second function

    // Set Pin_0 of Port 1 as output Port

    // Set as ACLK pin , Secondary function

    //Unloack All the pin for IO Opration

   // Define GPIO for LCD and Set its dircection and function

   //Configure Clock parameters and Initialize it

   // Configure LCD parameter and Initialize it

   // Send Initial Massage to be displayed on LCD

   // Configure Timer. Timer Must be configured for following parameter
       // Timer 0
   // Contineous Mode
   // Clock Soucre SMCLK
   // Capture Mode
   // Capture Compare Register
   // Capture on Rising Edge
   // Capture Signal at Pin no 1.6

   // ISR for Timer 0

 while (1)
       {
       // Set Power Mode For MSP430
       // No Opration
       }
 }
```

3.4 Implementation

```
// Configure TimerA_0

Timer_A_initContinuousModeParam param = {0};  // Timer A_0 in continuous
    mode
        param.clockSource = TIMER_A_CLOCKSOURCE_SMCLK; //Select SMCLK for
            the counter
        param.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_DISABLE;
            // Disable Interrupt
        param.timerClear = TIMER_A_DO_CLEAR; //Clear Timer
        param.startTimer = false; // Do not start Timer Yet
        Timer_A_initContinuousMode(TIMER_A0_BASE, &param);

    //Timer A_ in Capture Mode
   Timer_A_initCaptureModeParam param1 = {0};
   // Capture Compare Register 2 is used
        param1.captureRegister = TIMER_A_CAPTURECOMPARE_REGISTER_2 ;
    // Capture on Rising Edge
        param1.captureMode = TIMER_A_CAPTUREMODE_RISING_EDGE ;
    // Set the Pin 1.6 as input signal
        param1.captureInputSelect = TIMER_A_CAPTURE_INPUTSELECT_CCIxA;
        param1.synchronizeCaptureSource = TIMER_A_CAPTURE_SYNCHRONOUS;
    // Capture is Synchronous
        param1.captureInterruptEnable
            =TIMER_A_CAPTURECOMPARE_INTERRUPT_ENABLE;
   // Enable Capture Campare Intrupt
        param1.captureOutputMode =TIMER_A_OUTPUTMODE_OUTBITVALUE;
        Timer_A_initCaptureMode(TIMER_A0_BASE, &param1);
        Timer_A_clearTimerInterrupt(TIMER_A0_BASE);
        Timer_A_clearCaptureCompareInterrupt(TIMER_A0_BASE,
            TIMER_A_CAPTURECOMPARE_REGISTER_2);
    // Start the counter
   Timer_A_startCounter( TIMER_A0_BASE, TIMER_A_CONTINUOUS_MODE);
```

```
//ISR for Timer A_0;
#pragma vector=TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR(void)
{
    switch(__even_in_range( TA0IV, 6 ))
    {
            case 0x00: break;
            case 0x02:break;
            case 0x04:

      switch (captures)
     {
         case 0:
```

```
//Capture the Value of CCR2 at first Capture interrupt
       Starttime = Timer_A_getCaptureCompareCount(TIMER_A0_BASE,
                                    TIMER_A_CAPTURECOMPARE_REGISTER_2);
       captures = 1;
       GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
       break ;

      case 1:
   //Capture the CCR2 value in second Capture interrupt
      Frequency= ((Timer_A_getCaptureCompareCount(TIMER_A0_BASE,
                 TIMER_A_CAPTURECOMPARE_REGISTER_2)-Starttime)/4);
      //Frequency Calculation
   Frequency1 = (1000000/Frequency);
   //Display the Frequency on LCD
        DisplayFrequency (Frequency1);
      captures = 0;

   // Disable the CCR2 interrupt
      Timer_A_disableCaptureCompareInterrupt
          (TIMER_A0_BASE,TIMER_A_CAPTURECOMPARE_REGISTER_2);
      GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
      break;

      default :
            -- captures;
            break;
      }
      break;

   case 0x06:break;
   default: _never_executed();
      }
}
```

3.5 Analysis

3.5.1 Power Analysis

Figure 3.2 shows the power analysis of application. Function is used to set the Low power mode of MS430FR4133.

```
__bis_SR_register(LPM1_bits + GIE)
```

Figure 3.3 shows the Energy analysis and 3.4 shows the Summary of Power consumption of application of application .

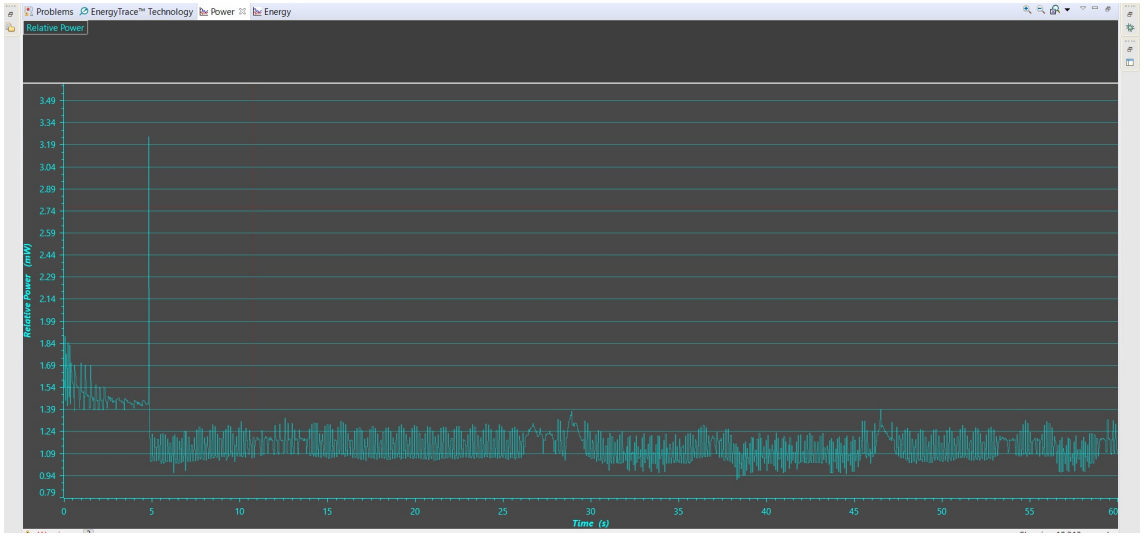

Figure 3.2: Power analysis for this application

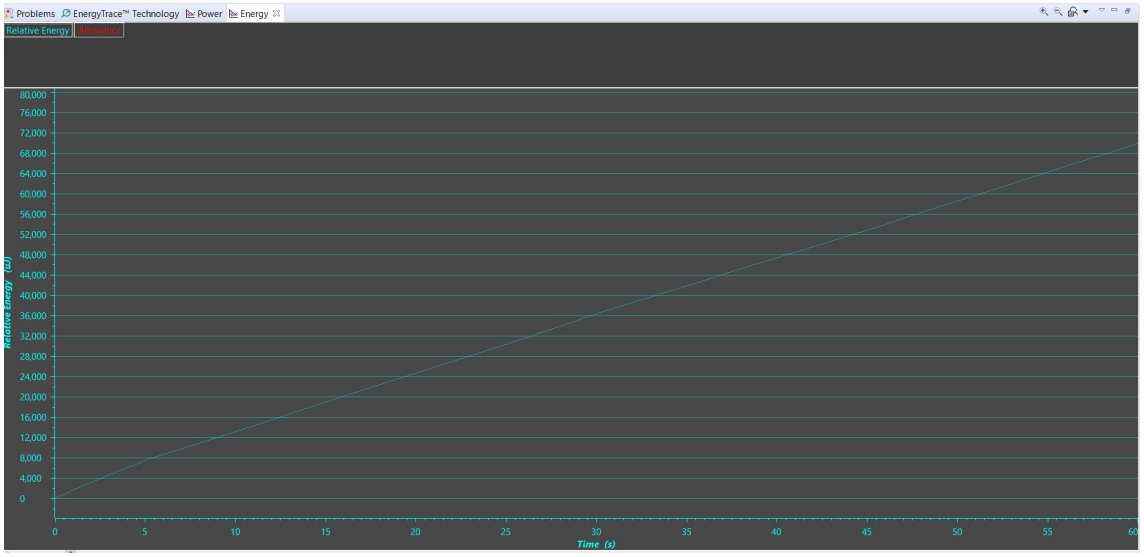

Figure 3.3: Energy analysis for this application

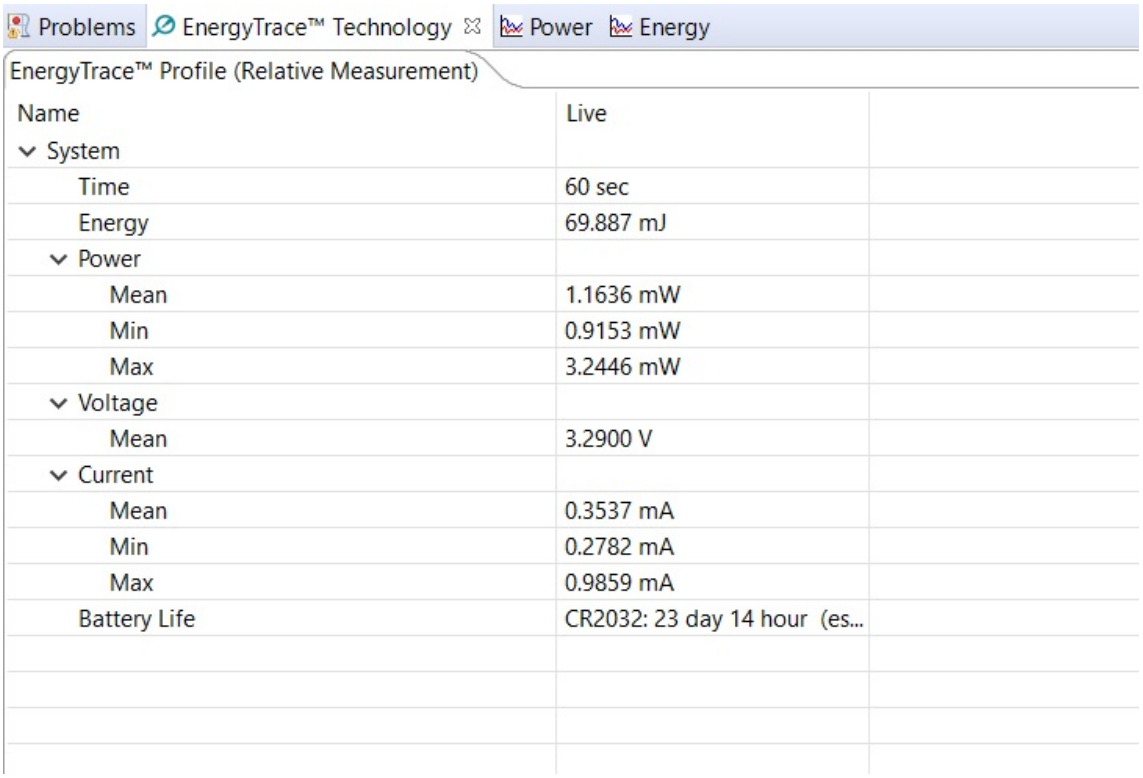

Figure 3.4: Summary of Power consumption for this application

Chapter 4

Improvements

- This method can be used only to measure frequency upto 500K Hz as Using SMCLK as counter clock. Use of external clock can add up the range .
- Use of External clock source also gives us freedom to set controller is LPM mode 3 so as to reduction in power consumption.
- Operating range of this controller is 16 MHz .controller with higher frequency will add up the range of frequency measurement.
- Only Frequency of the signal is calculate. Other Frequency Parameters can be calculated .
- Better LCD can also be used such as 2X16 alphanumerical so as to display more clear massage.

Chapter 5

References

- MSP430 Microcontroller Basics by John H Davie
- https://pdfs.semanticscholar.org/ff7d/7f52953fe22cab15ac38c5486c64a9f53901.pdf
- https://www.sciencedirect.com/topics/engineering/measurement-frequency
- http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview.html

Chapter 6

Appendices

6.1 Source Code

6.1.1 main.h

```
#ifndef MAIN_H_
#define MAIN_H_
#include <msp430.h>
#include <driverlib.h>
void Timer_init();
#endif
```

6.1.2 main.c

```
#include "LCD.h"
#include "string.h"
#include "driverlib.h"
#include "main.h"
#include "msp430fr4133.h"
volatile unsigned int captures = 0;
volatile unsigned int Starttime = 0;
volatile unsigned int Frequency = 0,Frequency1 = 0;

void main(void)
{
    WDT_A_hold(WDT_A_BASE);
    // TA0.CCI2A input capture pin, second function
    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1, GPIO_PIN6,
        GPIO_PRIMARY_MODULE_FUNCTION);

    GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
```

```
    //GPIO_setAsOutputPin(GPIO_PORT_P4, GPIO_PIN0);

    // Set as ACLK pin, second function
    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P8, GPIO_PIN1,
        GPIO_PRIMARY_MODULE_FUNCTION);

    PMM_unlockLPM5();
   Inti_GPIO();
    Init_Clock();
    Init_LCD();
    displayScrollText("FREQUENCY MEASURMENT");
    Timer_init();

 while (1)
       {
       __bis_SR_register(LPM1_bits + GIE);
       __no_operation();
       }

}
void Timer_init()
{
       Timer_A_initContinuousModeParam param = {0};
       param.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
       param.timerInterruptEnable_TAIE = TIMER_A_TAIE_INTERRUPT_DISABLE;
       param.timerClear = TIMER_A_DO_CLEAR;
       param.startTimer = false;
       Timer_A_initContinuousMode(TIMER_A0_BASE, &param);

   Timer_A_initCaptureModeParam param1 = {0};
       param1.captureRegister = TIMER_A_CAPTURECOMPARE_REGISTER_2 ;
       param1.captureMode = TIMER_A_CAPTUREMODE_RISING_EDGE ;
       param1.captureInputSelect = TIMER_A_CAPTURE_INPUTSELECT_CCIxA;
       param1.synchronizeCaptureSource = TIMER_A_CAPTURE_SYNCHRONOUS;
       param1.captureInterruptEnable
           =TIMER_A_CAPTURECOMPARE_INTERRUPT_ENABLE;
       param1.captureOutputMode =TIMER_A_OUTPUTMODE_OUTBITVALUE;
       Timer_A_initCaptureMode(TIMER_A0_BASE, &param1);
       Timer_A_clearTimerInterrupt(TIMER_A0_BASE);
       Timer_A_clearCaptureCompareInterrupt(TIMER_A0_BASE,
           TIMER_A_CAPTURECOMPARE_REGISTER_2);

      Timer_A_startCounter( TIMER_A0_BASE, TIMER_A_CONTINUOUS_MODE);
}


#pragma vector=TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR(void)
{
```

```
    switch(__even_in_range( TA0IV, 6 ))
    {
            case 0x00: break;
            case 0x02:break;
            case 0x04:

            switch (captures)
      {
          case 0:
          Starttime = Timer_A_getCaptureCompareCount(TIMER_A0_BASE,
                                        TIMER_A_CAPTURECOMPARE_REGISTER_2);
          captures = 1;
          GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
          break ;

     case 1:
     Frequency= ((Timer_A_getCaptureCompareCount(TIMER_A0_BASE,
                   TIMER_A_CAPTURECOMPARE_REGISTER_2)-Starttime)/4);
      Frequency1 = (1000000/Frequency);
      DisplayFrequency (Frequency1);
     captures = 0;
     Timer_A_disableCaptureCompareInterrupt
        (TIMER_A0_BASE,TIMER_A_CAPTURECOMPARE_REGISTER_2);
     GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
     break;

      default :
                -- captures;
               break;
    }
   break;

 case 0x06:break;
  default: _never_executed();
      }
}
```

6.1.3 LCD.h

```
#include <msp430fr4133.h>

#ifndef LCD_H_
#define LCD_H_


#define LCDMEMW ((int*)LCDMEM)

#ifdef __IAR_SYSTEMS_ICC__
```

```
#define LCDBMEMW ((int*)&LCDM32)
#else
#define LCDBMEMW ((int*)LCDBMEM)
#endif

#define pos1 4 /* Digit A1 - L4 */
#define pos2 6 /* Digit A2 - L6 */
#define pos3 8 /* Digit A3 - L8 */
#define pos4 10 /* Digit A4 - L10 */
#define pos5 2 /* Digit A5 - L2 */
#define pos6 18 /* Digit A6 - L18 */

void Inti_GPIO();
void Init_Clock();
void Init_LCD();
void clearLCD();
void displayScrollText(char *msg);
void showChar(char c, int position);
void DisplayFrequency(unsigned int count);
unsigned int frequency;

extern const char digit[10][2];
extern const char alphabetBig[26][2];

#endif
```

6.1.4 LCD.c

```
#include "LCD.h"
#include "string.h"
#include "driverlib.h"
#include "main.h"

volatile unsigned char * mode = &BAKMEM4_L;
volatile unsigned int * unit = &BAKMEM0_L;
volatile unsigned int * ten = &BAKMEM0_H;
volatile unsigned int * hun = &BAKMEM1_L;
volatile unsigned int * thou = &BAKMEM1_H;
volatile unsigned int * tthou = &BAKMEM1_L;

const char digit[10][2] =
{
    {0xFC, 0x28}, /* "0" LCD segments a+b+c+d+e+f+k+q */
    {0x60, 0x20}, /* "1" */
    {0xDB, 0x00}, /* "2" */
    {0xF3, 0x00}, /* "3" */
    {0x67, 0x00}, /* "4" */
    {0xB7, 0x00}, /* "5" */
```

```
    {0xBF, 0x00}, /* "6" */
    {0xE4, 0x00}, /* "7" */
    {0xFF, 0x00}, /* "8" */
    {0xF7, 0x00}  /* "9" */
};

const char alphabetBig[26][2] =
{
    {0xEF, 0x00}, /* "A" LCD segments a+b+c+e+f+g+m */
    {0xF1, 0x50}, /* "B" */
    {0x9C, 0x00}, /* "C" */
    {0xF0, 0x50}, /* "D" */
    {0x9F, 0x00}, /* "E" */
    {0x8F, 0x00}, /* "F" */
    {0xBD, 0x00}, /* "G" */
    {0x6F, 0x00}, /* "H" */
    {0x90, 0x50}, /* "I" */
    {0x78, 0x00}, /* "J" */
    {0x0E, 0x22}, /* "K" */
    {0x1C, 0x00}, /* "L" */
    {0x6C, 0xA0}, /* "M" */
    {0x6C, 0x82}, /* "N" */
    {0xFC, 0x00}, /* "O" */
    {0xCF, 0x00}, /* "P" */
    {0xFC, 0x02}, /* "Q" */
    {0xCF, 0x02}, /* "R" */
    {0xB7, 0x00}, /* "S" */
    {0x80, 0x50}, /* "T" */
    {0x7C, 0x00}, /* "U" */
    {0x0C, 0x28}, /* "V" */
    {0x6C, 0x0A}, /* "W" */
    {0x00, 0xAA}, /* "X" */
    {0x00, 0xB0}, /* "Y" */
    {0x90, 0x28}  /* "Z" */
};
void Inti_GPIO()
{
    //Port select XT1
        GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P4,GPIO_PIN1
            + GPIO_PIN2,GPIO_PRIMARY_MODULE_FUNCTION);
}

void Init_Clock()
{
    //Set external frequency for XT1
       CS_setExternalClockSource(32768);

      //Select XT1 as the clock source for ACLK with no frequency divider
       CS_initClockSignal(CS_ACLK, CS_XT1CLK_SELECT, CS_CLOCK_DIVIDER_1);
```

```
        //Start XT1 with no time out
        CS_turnOnXT1(CS_XT1_DRIVE_0);

        //clear all OSC fault flag
        CS_clearAllOscFlagsWithTimeout(1000);
}
void Init_LCD()
{
    // L0~L26 & L36~L39 pins selected
    LCD_E_setPinAsLCDFunctionEx(LCD_E_BASE, LCD_E_SEGMENT_LINE_0,
        LCD_E_SEGMENT_LINE_26);
    LCD_E_setPinAsLCDFunctionEx(LCD_E_BASE, LCD_E_SEGMENT_LINE_36,
        LCD_E_SEGMENT_LINE_39);

    LCD_E_initParam initParams = LCD_E_INIT_PARAM;
    initParams.clockDivider = LCD_E_CLOCKDIVIDER_3;
    initParams.muxRate = LCD_E_4_MUX;
    initParams.segments = LCD_E_SEGMENTS_ENABLED;

    LCD_E_init(LCD_E_BASE, &initParams);

   LCD_E_setVLCDSource(LCD_E_BASE, LCD_E_INTERNAL_REFERENCE_VOLTAGE,
       LCD_E_EXTERNAL_SUPPLY_VOLTAGE);
   LCD_E_setVLCDVoltage(LCD_E_BASE, LCD_E_REFERENCE_VOLTAGE_2_96V);
   LCD_E_enableChargePump(LCD_E_BASE);
   LCD_E_setChargePumpFreq(LCD_E_BASE, LCD_E_CHARGEPUMP_FREQ_16);
   LCD_E_clearAllMemory(LCD_E_BASE);

    // L0 = COM0, L1 = COM1, L2 = COM2, L3 = COM3
    LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_0,
        LCD_E_MEMORY_COM0);
    LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_1,
        LCD_E_MEMORY_COM1);
    LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_2,
        LCD_E_MEMORY_COM2);
    LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_3,
        LCD_E_MEMORY_COM3);

    LCD_E_selectDisplayMemory(LCD_E_BASE, LCD_E_DISPLAYSOURCE_MEMORY);

    LCD_E_on(LCD_E_BASE);
}


void displayScrollText(char *msg)
{
    int length = strlen(msg);
    int oldmode = *mode;
```

```
    int i;
    int s = 5;
    char buffer[6] = "    ";


    for (i=0; i<length+7; i++)
    {
        if (*mode != oldmode)
            break;
        int t;
        for (t=0; t<6; t++)
            buffer[t] = ' ';
        int j;
        for (j=0; j<length; j++)
        {
            if (((s+j) >= 0) && ((s+j) < 6))
                buffer[s+j] = msg[j];
        }
        s--;

        showChar(buffer[0], pos1);
        showChar(buffer[1], pos2);
        showChar(buffer[2], pos3);
        showChar(buffer[3], pos4);
        showChar(buffer[4], pos5);
        showChar(buffer[5], pos6);

        __delay_cycles(200000);
}
}

void showChar(char c, int position)
{
    if (c == ' ')
    {

        LCDMEMW[position/2] = 0;
    }
    else if (c >= '0' && c <= '9')
    {

        LCDMEMW[position/2] = digit[c-48][0] | (digit[c-48][1] << 8);
    }
    else if (c >= 'A' && c <= 'Z')
    {

        LCDMEMW[position/2] = alphabetBig[c-65][0] | (alphabetBig[c-65][1]
            << 8);
    }
```

```
    else
    {

        LCDMEMW[position/2] = 0xFFFF;
    }

}
void DisplayFrequency(unsigned int count)
{

     unsigned int count1,count2,count3,count4;

      unit= &count;
      count1=*unit/10;
      ten=&count1;
      count2=*ten/10;
      hun=&count2;
      count3=*hun/10;
      thou =&count3;
      count4=*thou/10;
      tthou=&count4;

         showChar('H' ,pos6);
        showChar((*unit) % 10+'0',pos5);
        showChar((*ten) % 10 +'0' ,pos4);
        showChar((*hun) % 10 +'0' ,pos3);
        showChar((*thou) % 10 +'0' ,pos2);
        showChar((*tthou) % 10 +'0' ,pos1);
}

void clearLCD()
{
    LCDMEMW[pos1/2] = 0;
    LCDMEMW[pos2/2] = 0;
    LCDMEMW[pos3/2] = 0;
    LCDMEMW[pos4/2] = 0;
    LCDMEMW[pos5/2] = 0;
    LCDMEMW[pos6/2] = 0;
    LCDMEM[12] = LCDMEM[13] = 0;
}
```

List of Figures