Project 3 - Digital Multimeter

Tanvi Kharkar

EE 329-05

Spring Quarter 2021

Professor Hummel

Device Behavior Description

This project designs the MSP432P401R microcontroller in conjunction with a 14-bit analog-to-digital converter (ADC) to act as a digital multimeter. The digital multimeter is capable of measuring DC voltages from 0V to 3.3V as well as AC measurements including peak-to-peak voltage, true RMS voltage, and frequencies from 2Hz to 1kHz for any AC wave. The digital multimeter measurements are readable via a computer terminal.

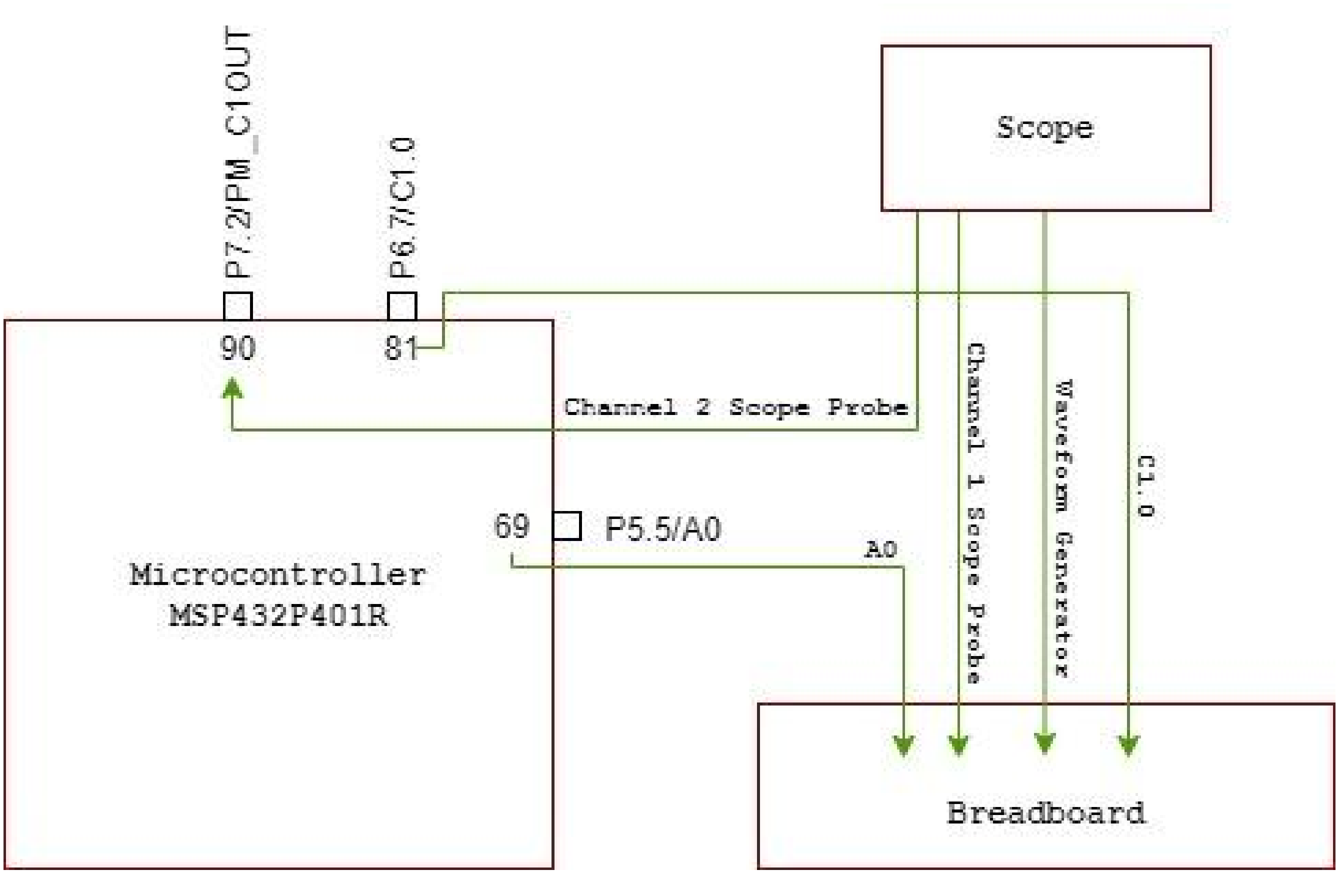
System Specification

Table 1: Digital Multimeter Parameters and Respective Values [1], [2]

System Parameter	Value
ADC Bit Resolution	14-Bit
Measurable Range of Frequencies	2Hz - 1kHz
Measurable Range of Voltages	0V - 3.3V
Clock Frequency	24MHz
Baud Rate	115200

System Schematic

Figure 1: Schematic of Digital Multimeter Hardware Configuration



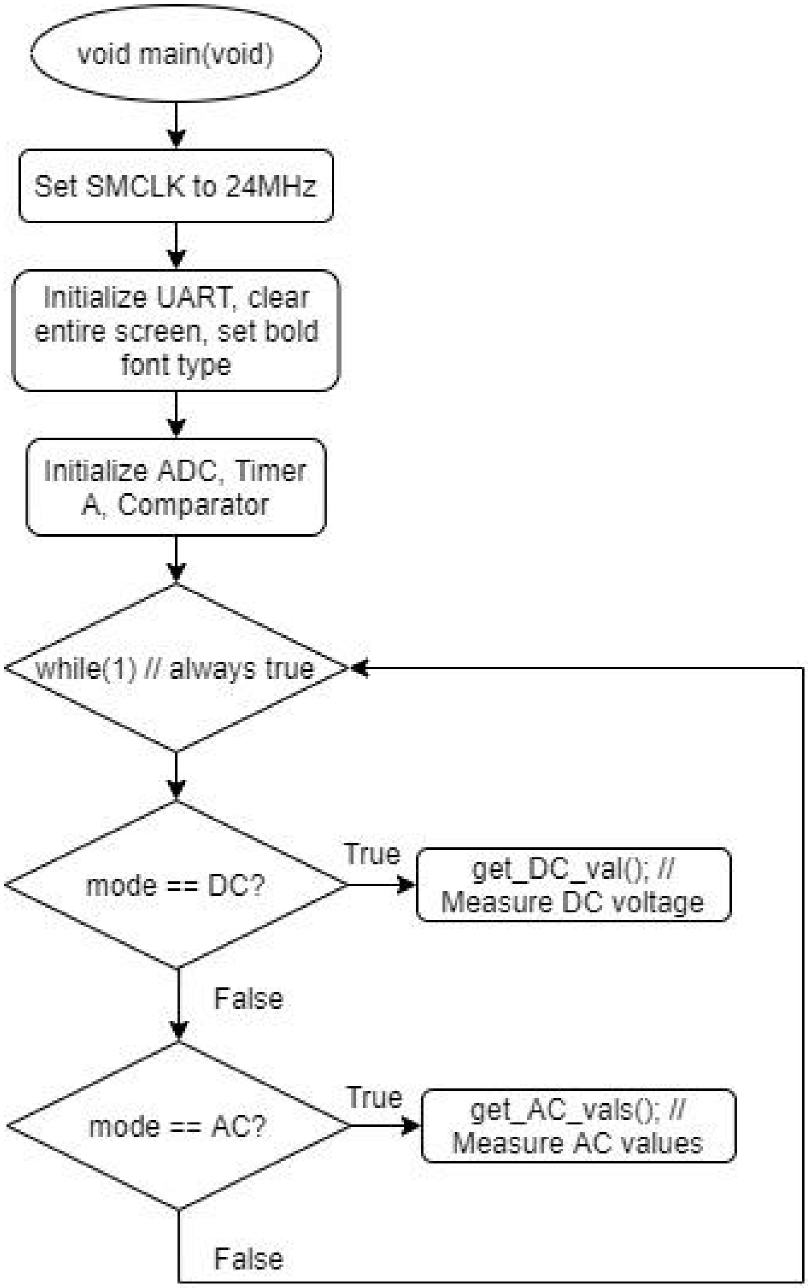
*Waveform Generator, Channel 1 Scope Probe, A0, and C1.0 are all connected to the same breadboard terminal

Software Architecture

The digital multimeter is controlled by a while loop which checks for keyboard presses of either ‘A’ or ‘D’. If ‘A’ is pressed, the digital multimeter will display AC measurements (frequency, peak-to-peak voltage, and true RMS voltage) whereas, if ‘D’ is pressed, the digital multimeter will display DC measurements (DC voltage).

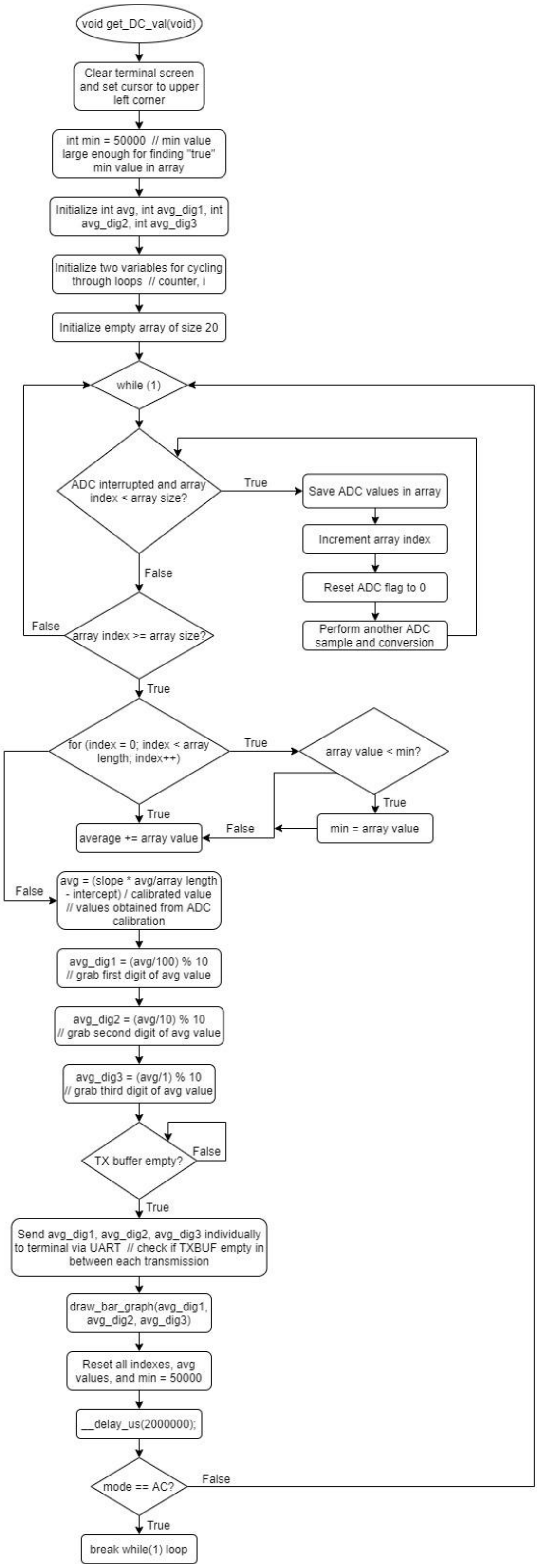
<u>Main Diagram</u>

Figure 2: main flowchart


void main(void)
Set SMCLK to 24MHz
Initialize UART, clear entire screen, set bold font type
Initialize ADC, Timer A, Comparator
while(1) // always true
mode == DC?
True
get_DC_val(); // Measure DC voltage
False
mode == AC?
True
get_AC_vals(); // Measure AC values
False

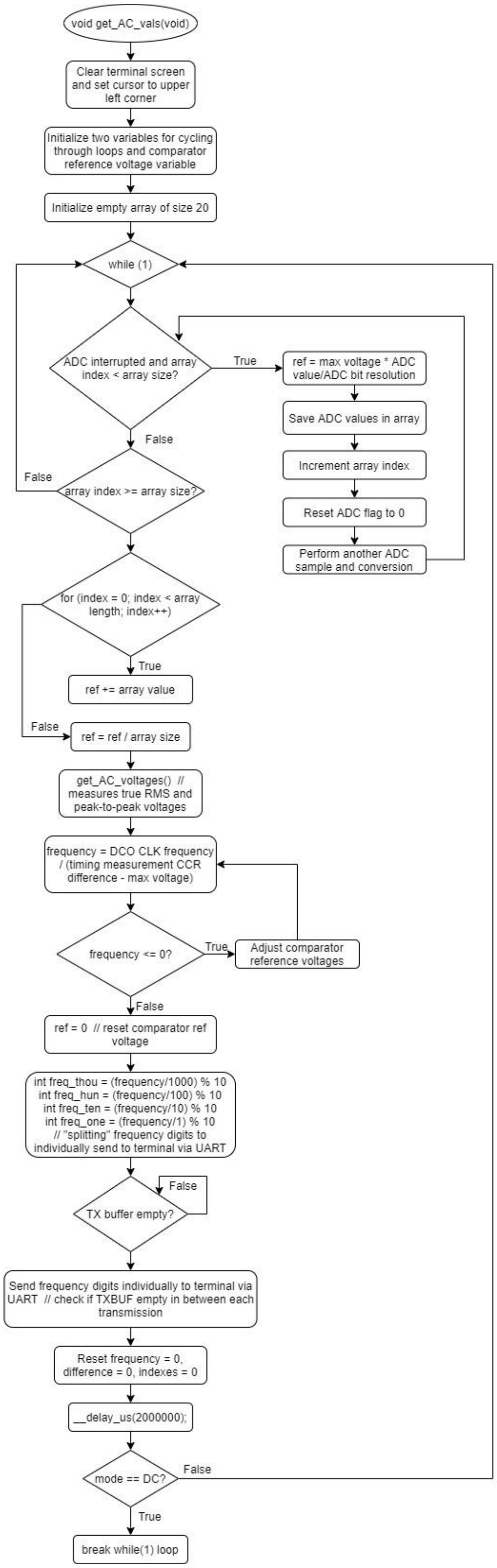
DC Voltage Diagram

Figure 3: get_DC_val flowchart


void get_DC_val(void)
Clear terminal screen and set cursor to upper left corner
int min = 50000 // min value large enough for finding "true" min value in array
Initialize int avg, int avg_dig1, int avg_dig2, int avg_dig3
Initialize two variables for cycling through loops // counter, i
Initialize empty array of size 20
while (1)
ADC interrupted and array index < array size?
True
Save ADC values in array
Increment array index
Reset ADC flag to 0
Perform another ADC sample and conversion
False
False
array index >= array size?
True
for (index = 0; index < array length; index++)
True
array value < min?
True
min = array value
False
True
average += array value
False
avg = (slope * avg/array length - intercept) / calibrated value // values obtained from ADC calibration
avg_dig1 = (avg/100) % 10 // grab first digit of avg value
avg_dig2 = (avg/10) % 10 // grab second digit of avg value
avg_dig3 = (avg/1) % 10 // grab third digit of avg value
TX buffer empty?
False
True
Send avg_dig1, avg_dig2, avg_dig3 individually to terminal via UART // check if TXBUF empty in between each transmission
draw_bar_graph(avg_dig1, avg_dig2, avg_dig3)
Reset all indexes, avg values, and min = 50000
__delay_us(2000000);
mode == AC?
False
True
break while(1) loop

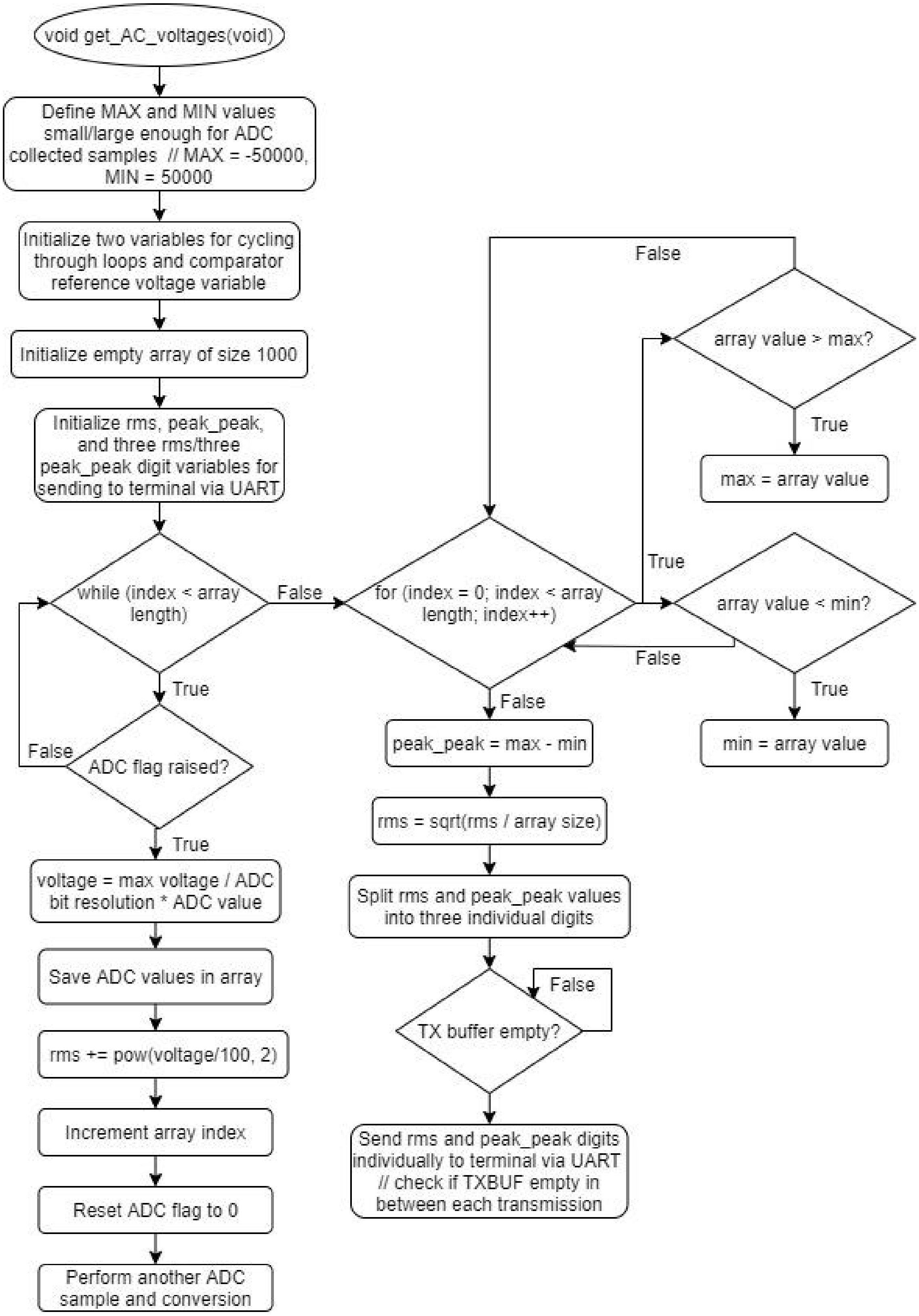
AC Values Diagram

Figure 4: get_AC_vals flowchart


void get_AC_vals(void)
Clear terminal screen and set cursor to upper left corner
Initialize two variables for cycling through loops and comparator reference voltage variable
Initialize empty array of size 20
while (1)
ADC interrupted and array index < array size?
True
ref = max voltage * ADC value/ADC bit resolution
Save ADC values in array
Increment array index
Reset ADC flag to 0
Perform another ADC sample and conversion
False
False
array index >= array size?
for (index = 0; index < array length; index++)
True
ref += array value
False
ref = ref / array size
get_AC_voltages() // measures true RMS and peak-to-peak voltages
frequency = DCO CLK frequency / (timing measurement CCR difference - max voltage)
frequency <= 0?
True
Adjust comparator reference voltages
False
ref = 0 // reset comparator ref voltage
int freq_thou = (frequency/1000) % 10
int freq_hun = (frequency/100) % 10
int freq_ten = (frequency/10) % 10
int freq_one = (frequency/1) % 10
// "splitting" frequency digits to individually send to terminal via UART
False
TX buffer empty?
Send frequency digits individually to terminal via UART // check if TXBUF empty in between each transmission
Reset frequency = 0, difference = 0, indexes = 0
__delay_us(2000000);
mode == DC?
False
True
break while(1) loop

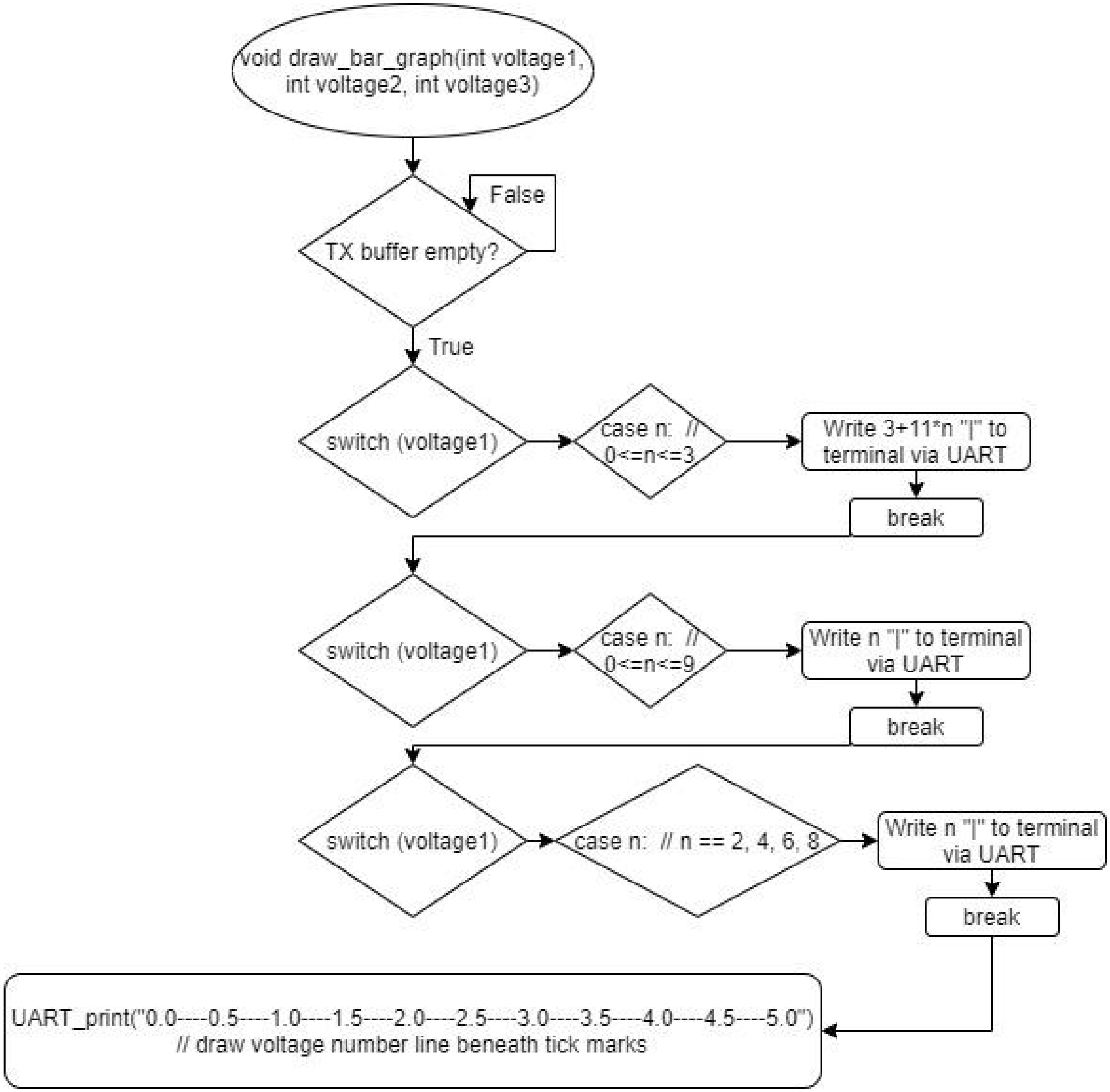
AC Voltages Diagram

Figure 5: get_AC_voltages flowchart


void get_AC_voltages(void)
Define MAX and MIN values small/large enough for ADC collected samples // MAX = -50000, MIN = 50000
Initialize two variables for cycling through loops and comparator reference voltage variable
Initialize empty array of size 1000
Initialize rms, peak_peak, and three rms/three peak_peak digit variables for sending to terminal via UART
while (index < array length)
True
ADC flag raised?
False
True
voltage = max voltage / ADC bit resolution * ADC value
Save ADC values in array
rms += pow(voltage/100, 2)
Increment array index
Reset ADC flag to 0
Perform another ADC sample and conversion
False
for (index = 0; index < array length; index++)
False
peak_peak = max - min
rms = sqrt(rms / array size)
Split rms and peak_peak values into three individual digits
TX buffer empty?
False
Send rms and peak_peak digits individually to terminal via UART // check if TXBUF empty in between each transmission
True
False
array value > max?
True
max = array value
array value < min?
False
True
min = array value

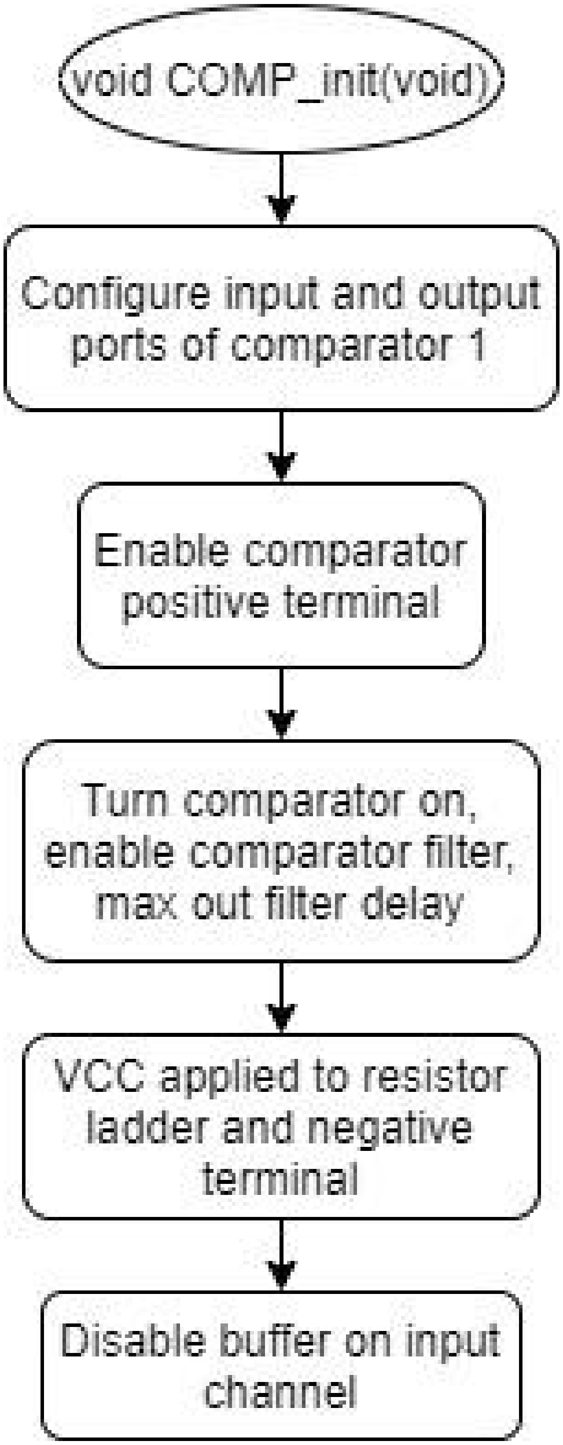
Bar Graph Diagram

Figure 6: draw_bar_graph flowchart


void draw_bar_graph(int voltage1, int voltage2, int voltage3)
TX buffer empty?
False
True
switch (voltage1)
case n: // 0<=n<=3
Write 3+11*n "|" to terminal via UART
break
switch (voltage1)
case n: // 0<=n<=9
Write n "|" to terminal via UART
break
switch (voltage1)
case n: // n == 2, 4, 6, 8
Write n "|" to terminal via UART
break
UART_print("0.0----0.5----1.0----1.5----2.0----2.5----3.0----3.5----4.0----4.5----5.0")
// draw voltage number line beneath tick marks

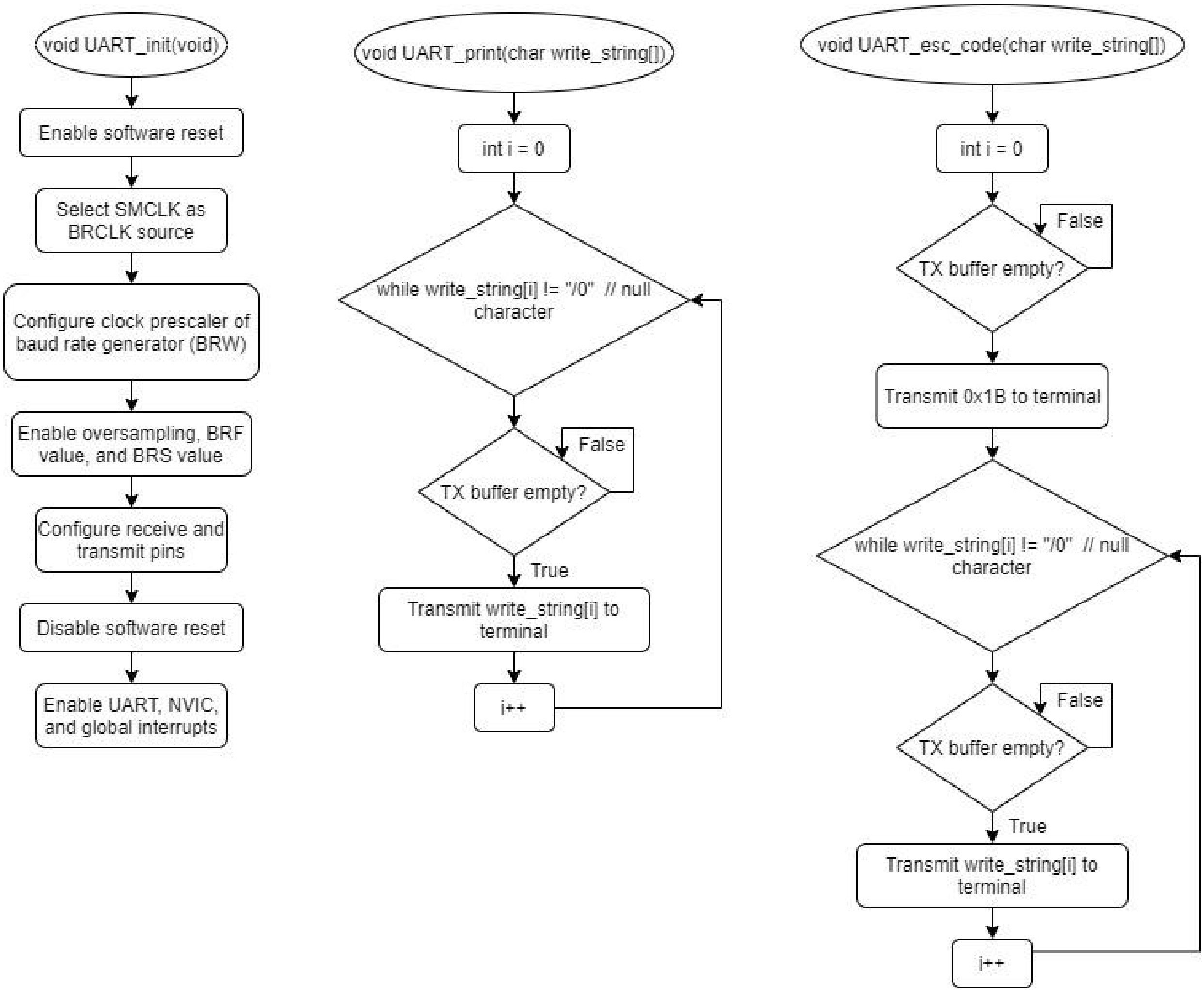
Comparator Diagram

Figure 7: COMP_init flowchart


void COMP_init(void)
Configure input and output ports of comparator 1
Enable comparator positive terminal
Turn comparator on, enable comparator filter, max out filter delay
VCC applied to resistor ladder and negative terminal
Disable buffer on input channel

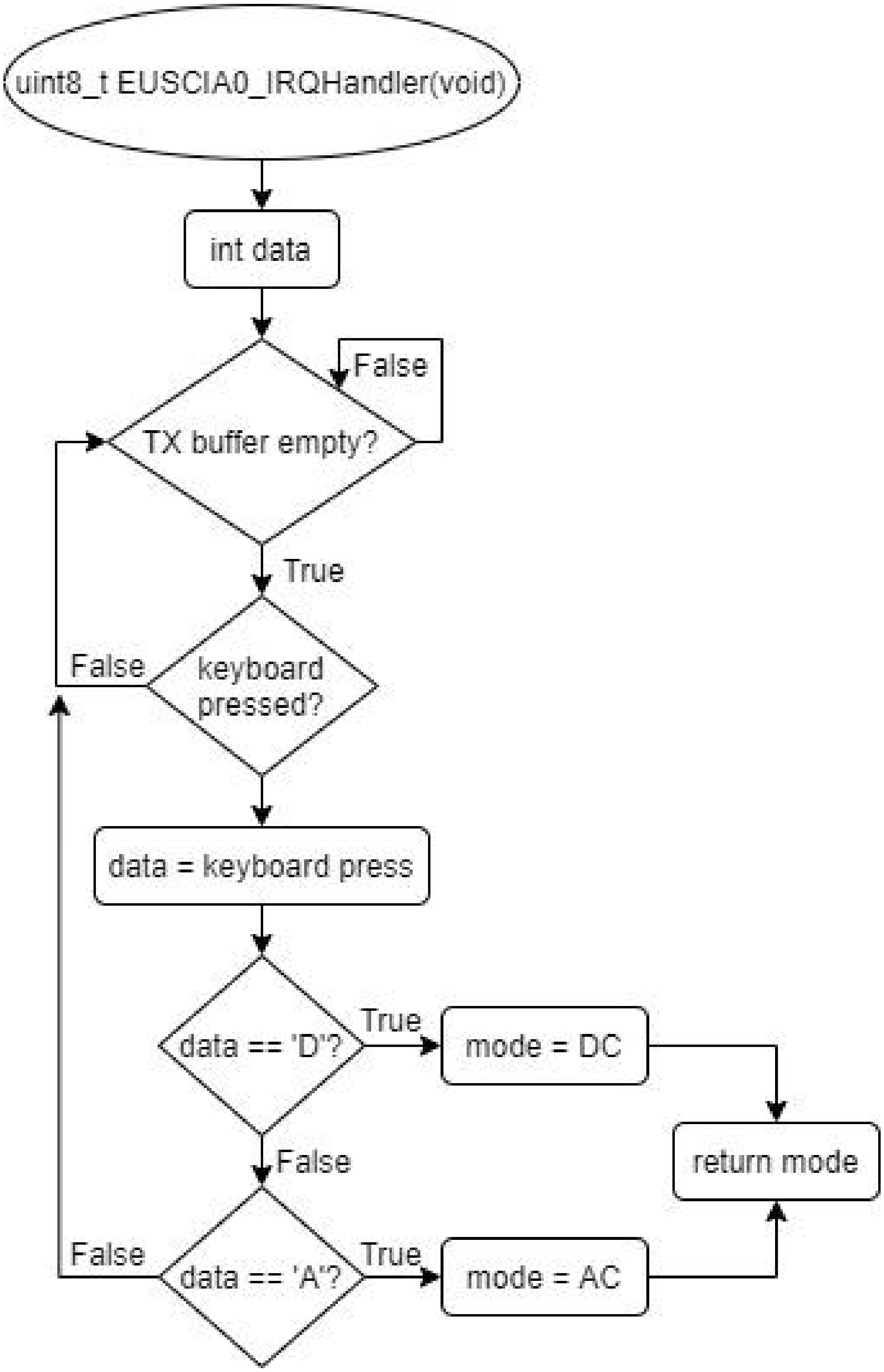
UART Diagrams

Figure 8: UART_init, UART_print, UART_esc_code flowcharts


void UART_init(void)
Enable software reset
Select SMCLK as BRCLK source
Configure clock prescaler of baud rate generator (BRW)
Enable oversampling, BRF value, and BRS value
Configure receive and transmit pins
Disable software reset
Enable UART, NVIC, and global interrupts
void UART_print(char write_string[])
int i = 0
while write_string[i] != "/0" // null character
TX buffer empty?
False
True
Transmit write_string[i] to terminal
i++
void UART_esc_code(char write_string[])
int i = 0
TX buffer empty?
False
Transmit 0x1B to terminal
while write_string[i] != "/0" // null character
TX buffer empty?
False
True
Transmit write_string[i] to terminal
i++

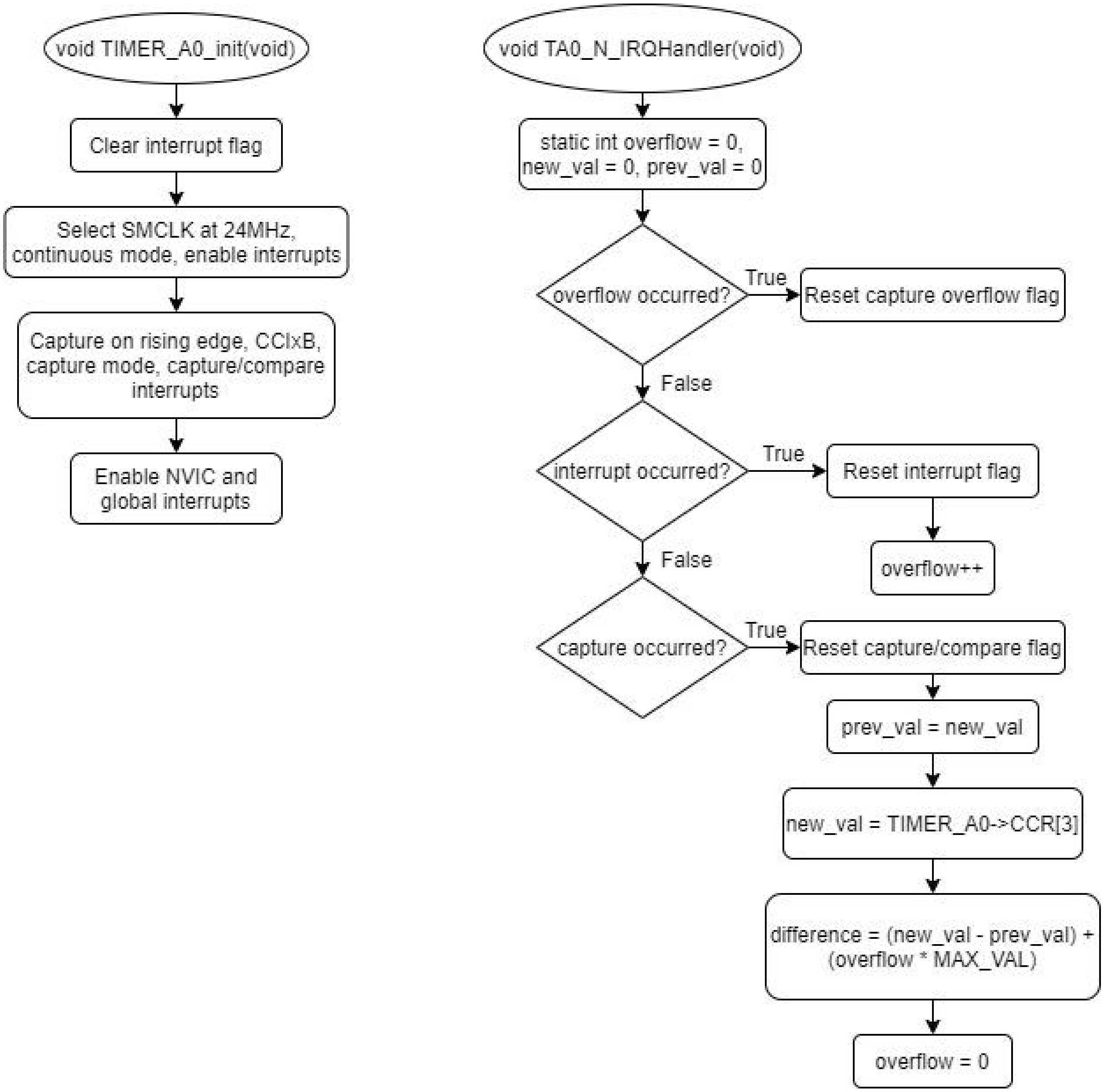
EUSCI ISR Diagram

Figure 9: EUSCIA0_IRQHandler flowchart



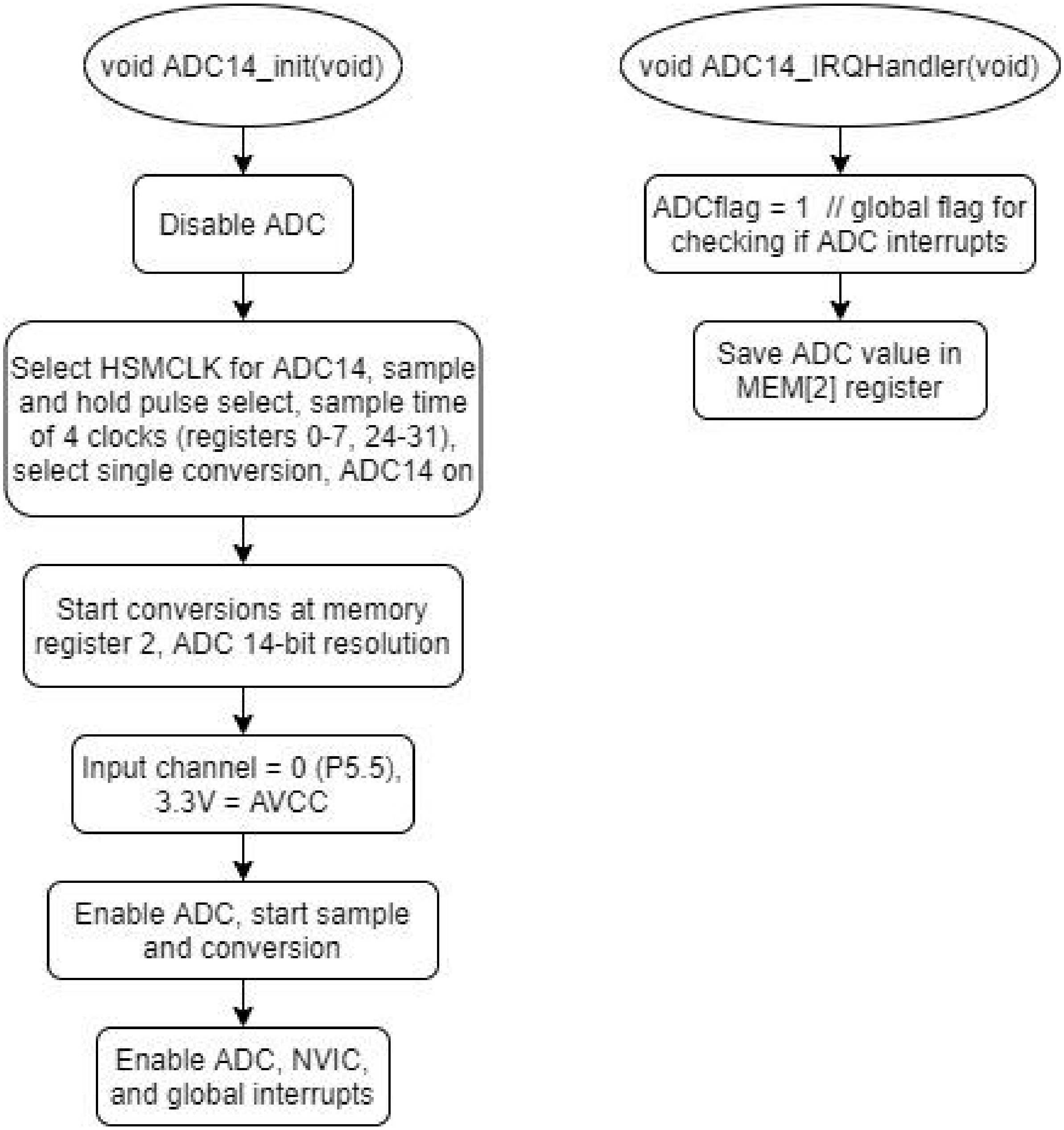
Timer A Diagrams

Figure 10: TIMER_A0_init, TA0_N_IRQHandler flowcharts


void TIMER_A0_init(void)
Clear interrupt flag
Select SMCLK at 24MHz, continuous mode, enable interrupts
Capture on rising edge, CCIxB, capture mode, capture/compare interrupts
Enable NVIC and global interrupts
void TA0_N_IRQHandler(void)
static int overflow = 0, new_val = 0, prev_val = 0
overflow occurred?
True
Reset capture overflow flag
False
interrupt occurred?
True
Reset interrupt flag
overflow++
False
capture occurred?
True
Reset capture/compare flag
prev_val = new_val
new_val = TIMER_A0->CCR[3]
difference = (new_val - prev_val) + (overflow * MAX_VAL)
overflow = 0

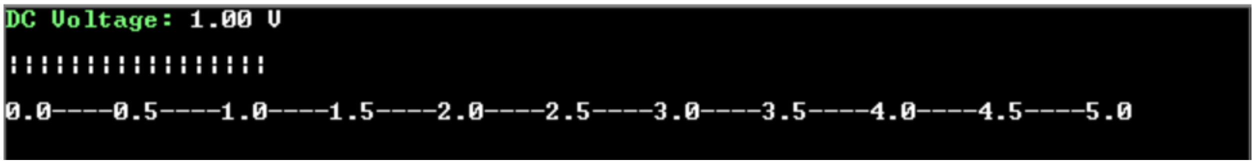
ADC Diagrams

Figure 11: ADC14_init, ADC14_IRQHandler flowcharts


void ADC14_init(void)
Disable ADC
Select HSMCLK for ADC14, sample and hold pulse select, sample time of 4 clocks (registers 0-7, 24-31), select single conversion, ADC14 on
Start conversions at memory register 2, ADC 14-bit resolution
Input channel = 0 (P5.5), 3.3V = AVCC
Enable ADC, start sample and conversion
Enable ADC, NVIC, and global interrupts
void ADC14_IRQHandler(void)
ADCflag = 1 // global flag for checking if ADC interrupts
Save ADC value in MEM[2] register

Sample DC Measurement Terminal Display

Figure 12: DC Measurement of 1V DC Offset



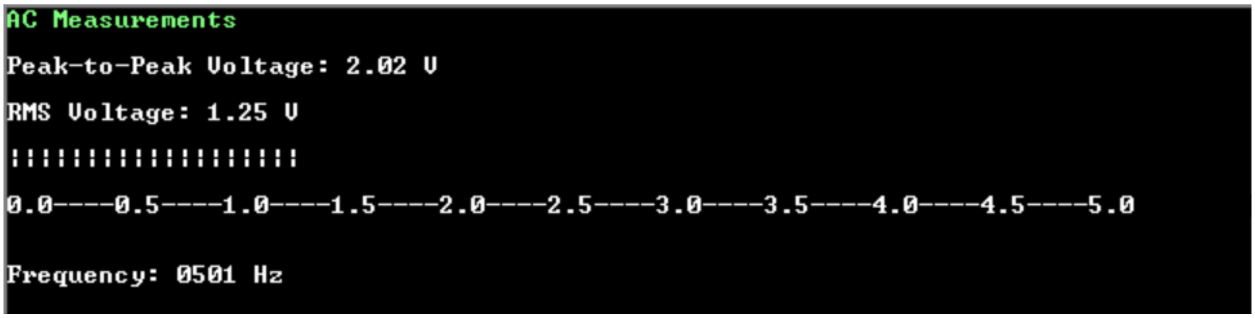
```
DC Voltage: 1.00 V

!!!!!!!!!!!!!!!!!

0.0----0.5----1.0----1.5----2.0----2.5----3.0----3.5----4.0----4.5----5.0
```

Sample AC Measurements Terminal Display

Figure 13: AC Measurements of 500Hz, 2V$_{peak\text{-}to\text{-}peak}$, 1V DC Offset Sine Wave



```
AC Measurements

Peak-to-Peak Voltage: 2.02 V

RMS Voltage: 1.25 V

!!!!!!!!!!!!!!!!!!!

0.0----0.5----1.0----1.5----2.0----2.5----3.0----3.5----4.0----4.5----5.0

Frequency: 0501 Hz
```

Appendices

main.c

```
--------------------------------------------------------------------------------------------------------------
#include "msp.h"
#include "DCO.h"
#include "UART.h"
#include "ADC.h"
#include "math.h"

#define CPU_FREQ 3000000    // Microcontroller frequency
#define SCLK_FREQ 24000000
#define WRITE_DELAY 2000000
#define __delay_us(t_us) (__delay_cycles((((uint64_t)t_us)*CPU_FREQ) / 1000000))

#define DC 1    // DC or AC depending on measurements to be taken
#define AC 2

#define C1_IN_PORT P6   // Initializing comparator ports
#define C1_IN BIT7      // Input port: P6.7
#define C1_OUT_PORT P7  // Output port: P7.2
#define C1_OUT BIT2

#define MAX_VAL 0xFFFF
#define MAX_VOLTAGE 33000
#define ADC_RES 16384   // Max resolution of ADC 14-bit: 2^14 = 16384
#define REF_CONV 31
#define FREQ_CONV 300
#define ANALOG_ARRAY_LENGTH 1000

int val; // Global for saving MEM[2] values
uint8_t ADCflag = 0;    // Global flag for checking when to populate ADC array
uint8_t mode;   // Mode controlled by user (keyboard press) for deciding whether to take AC or DC measurements
int difference = 0; // Global for calculating difference between CCR Timer A measurements

void COMP_init(void);   // Initializing functions
void TIMER_A0_init(void);
void draw_bar_graph(int voltage1, int voltage2, int voltage3);
void TA0_N_IRQHandler(void);
void ADC14_IRQHandler(void);
uint8_t EUSCIA0_IRQHandler(void);
void get_DC_val(void);
void get_AC_vals(void);
void get_AC_voltages(void);


void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

    set_DCO(FREQ_24_MHz);
    UART_init();    // Initialize UART
    UART_esc_code(CLEARSCREEN); // Clear terminal screen
    UART_esc_code(BOLD);    // All terminal text is bold print for ease of reading
    ADC14_init();   // Initialize ADC
    TIMER_A0_init();    // Initialize Timer A
    COMP_init();    // Initialize Comparator E1

    while(1)
    {
        if (mode == DC) // If key 'D' pressed on keyboard, take DC measurements
        {
            UART_esc_code(TOPLEFT); // Moves cursor to the top left of the screen, essentially "rewriting" screen
            get_DC_val();   // Calculates DC voltage value
        }

        else if (mode == AC) // If key 'A' pressed on keyboard, take AC measurements
        {
            UART_esc_code(TOPLEFT);
            get_AC_vals();  // Calculates AC true RMS voltage, peak-to-peak voltage, and frequency
        }
    }
}


void get_DC_val(void)
{
    UART_print(CLEARSCREEN);
    UART_print(TOPLEFT);

    int min = DEF_MIN;
    int avg = 0;
    int avg_dig1, avg_dig2, avg_dig3;   // Variables for individually sending average DC voltage value to terminal via
UART
    uint8_t counter, i = 0;    // Variables for cycling through loops
    int digital_conversion[ARRAY_LENGTH] = EMPTY_ARRAY;  // Initialize empty array for saving ADC values


    while(1)    // Infinite loop to check for ADC global flag
    {
        if (ADCflag == 1 && i < ARRAY_LENGTH)   // ADC interrupt triggered
        {
            digital_conversion[i] = val;    // Save ADC value in array
            i++;    // Increment array index
            ADCflag = 0;    // Reset ADC interrupt flag
            ADC14->CTL0 |= (ADC14_CTL0_SC);  // Perform another sample and conversion
        }

        else if (i >= ARRAY_LENGTH)    // Array is filled with samples
        {
            for (counter = 0; counter < ARRAY_LENGTH; counter++)
            {
                if (digital_conversion[counter] < min)  // Find min value in array
                {
                    min = digital_conversion[counter];
                }

                avg += digital_conversion[counter]; // Add all values in array to calculate average
            }

            avg = (SLOPE * (avg/ARRAY_LENGTH) - INTERCEPT) / CAL; // Calculate calibrated average of array
            min = (SLOPE * min - INTERCEPT) / CAL;  // Calculate calibrated min of array

            if (min <= 0)   // To avoid negative integers
                avg = ZERO;

            else if (min > 0 && min <= 69)  // Add 3 for 0V to 69mV, calibration
                avg += PLUS3;

            else if (min >= 70 && min <= 169)   // Add 2 for 70mV to 169mV, calibration
                avg += PLUS2;
            else if (min > 325) // Max voltage = 3.3V
                avg = MAX;

            avg_dig1 = (avg / 100) % 10;    // "Splitting" average voltage into three individual values...
            avg_dig2 = (avg / 10) % 10;     // ...to send to terminal via UART
            avg_dig3 = (avg / 1) % 10;

            while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));    // Wait for TXBUF to be empty in order to transmit...
                                                            // ... to terminal via UART
            UART_esc_code(LEFT);    // Moves cursor to the left of the terminal to keep text level
            UART_esc_code(GREEN);   // Header title is green
            UART_print("DC Voltage: ");
            UART_esc_code(WHITE);   // Change font color to white
            while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));
            EUSCI_A0->TXBUF = avg_dig1 + '0';   // Print digits one by one
            UART_print(".");
            while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));
            EUSCI_A0->TXBUF = avg_dig2 + '0';
            while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));
            EUSCI_A0->TXBUF = avg_dig3 + '0';
            while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));
            UART_print(" V");
            UART_esc_code(LEFT);
            UART_esc_code(NEWLINE); // Skips a line on terminal
            UART_esc_code(NEWLINE);

            draw_bar_graph(avg_dig1, avg_dig2, avg_dig3);   // Function for drawing voltage bar graph

            i = 0;  // Reset array index
            min = DEF_MIN;
            avg = 0;
            avg_dig1 = 0;
            avg_dig2 = 0;
            avg_dig3 = 0;
            ADCflag = 1;    // Raise interrupt flag
            __delay_us(WRITE_DELAY);   // Delay for allowing enough time for terminal to display all values
            UART_esc_code(CLEARSCREEN);
            UART_esc_code(TOPLEFT);
        }
        if (mode == AC) // If 'A' press detected, switch to AC measurements mode
            break;
    }
}


void get_AC_vals(void)
{
    UART_print(CLEARSCREEN);
    UART_print(TOPLEFT);

    uint8_t i = 0, counter = 0;    // Variables for cycling through loops
    int analog_conversion[ARRAY_LENGTH] = EMPTY_ARRAY;  // Initialize empty array for saving MEM[2] values
    int REF = 0, frequency = 0, freq_thou = 0, freq_hun = 0, freq_ten = 0, freq_one = 0; // REF = reference voltage
value

    while(1)
    {
        if (ADCflag == 1 && i < ARRAY_LENGTH)
        {
            REF = REF_CONV * val/ADC_RES;   // Converts ADC value to equivalent AC voltage
            analog_conversion[i] = REF; // Save REF value in array
            i++;    // Increment array index
            ADCflag = 0;    // Reset interrupt flag
            ADC14->CTL0 |= (ADC14_CTL0_SC);  // Perform another sample and conversion
        }

        else if (i >= ARRAY_LENGTH) // Array is filled with samples
        {
            for (counter = 0; counter < ARRAY_LENGTH; counter++)   // Add all REF values in array
            {
                REF += analog_conversion[counter];
            }

            REF = REF/ARRAY_LENGTH; // Divide all REF values by length of array to find average ref voltage value

            get_AC_voltages();  // Calculates true RMS and peak-to-peak AC voltages

            frequency = SCLK_FREQ/(difference - FREQ_CONV);   // Converts ADC reading to equivalent AC frequency
using...
                                                        // ... difference between timing measurements captured by Timer
A

            if (frequency <= 0)     // Different reference voltage value implementations to test if frequency <= 0
            {
                COMP_E1->CTL2 |= ((REF+1) << COMP_E_CTL2_REF0_OFS
                              |   (REF) << COMP_E_CTL2_REF1_OFS);
            }

            frequency = SCLK_FREQ/(difference - FREQ_CONV);  // Check frequency in between each ref voltage
implementation

            if (frequency <= 0)
            {
                COMP_E1->CTL2 |= ((REF-1) << COMP_E_CTL2_REF0_OFS
                              |   (REF-1) << COMP_E_CTL2_REF1_OFS);
            }

            frequency = SCLK_FREQ/(difference - FREQ_CONV);

            if (frequency <= 0)
            {
                COMP_E1->CTL2 |= ((REF+1) << COMP_E_CTL2_REF0_OFS
                              |   (REF+1) << COMP_E_CTL2_REF1_OFS);
            }

            frequency = SCLK_FREQ/(difference - FREQ_CONV);

            if (frequency <= 0)
            {
                COMP_E1->CTL2 |= ((REF) << COMP_E_CTL2_REF0_OFS
                              |   (REF-1) << COMP_E_CTL2_REF1_OFS);
            }

            frequency = SCLK_FREQ/(difference - FREQ_CONV);

            if (frequency <= 0)
            {
                COMP_E1->CTL2 |= ((REF+1) << COMP_E_CTL2_REF0_OFS
                              |   (REF-1) << COMP_E_CTL2_REF1_OFS);
            }

            REF = 0;    // Reset reference voltage value
            frequency = SCLK_FREQ/(difference - FREQ_CONV);   // Recalculate final frequency value
            if (frequency < 0)  // To avoid negative frequencies
```

```
                frequency = 0;

            freq_thou = (frequency/1000) % 10;  // "Splitting" frequency into four individual values to send to...
            freq_hun = (frequency/100) % 10;    // ... terminal via UART
            freq_ten = (frequency/10) % 10;
            freq_one = (frequency/1) % 10;

            while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));  // Wait for TXBUF to be empty in order to transmit to...
                                                            // ...terminal via UART
            UART_print("Frequency: ");
            while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
            EUSCI_A0 -> TXBUF = freq_thou + '0';    // Prints digits one by one
            while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
            EUSCI_A0 -> TXBUF = freq_hun + '0';
            while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
            EUSCI_A0 -> TXBUF = freq_ten + '0';
            while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
            EUSCI_A0 -> TXBUF = freq_one + '0';
            while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
            UART_print(" Hz");
            UART_esc_code(NEWLINE);
            UART_esc_code(NEWLINE);

            difference = 0;
            frequency = 0;
            i = 0;
            __delay_us(WRITE_DELAY);
            UART_esc_code(CLEARSCREEN);
            UART_esc_code(TOPLEFT);
        }
        if (mode == DC) // If 'D' press detected, switch to DC measurements mode
            break;
    }
}


void get_AC_voltages(void)
{
    int max = DEF_MAX;   // Default values for maximum and minimum of array
    int min = DEF_MIN;
    int rms = 0, new_val = 0;
    uint8_t rms_dig1, rms_dig2, rms_dig3;   // Variables for sending individual RMS digits to terminal via UART
    int peak_peak = 0;
    uint8_t peak_dig1, peak_dig2, peak_dig3;
    int analog_vals[ANALOG_ARRAY_LENGTH] = 0;  // Initialize empty array for saving ADC values
    uint16_t i = 0, counter = 0;     // Variables for cycling through arrays

    while(i < ANALOG_ARRAY_LENGTH)  // Keep running until all array values are filled
    {
        if (ADCflag == 1)
        {
            new_val = MAX_VOLTAGE/ADC_RES * val;    // Converts ADC value to equivalent AC voltage
            analog_vals[i] = new_val;   // Save converted voltage value in array
            rms += pow(new_val/100, 2); // Adding square of all samples to calculate true RMS voltage
            i++;    // Increment array index
            ADCflag = 0;    // Reset interrupt flag
            ADC14->CTL0 |= (ADC14_CTL0_SC);  // Perform another sample and conversion
        }
    }

    if (i >= ANALOG_ARRAY_LENGTH)    // Array is filled
    {
```

```c
    for (counter = 0; counter < ANALOG_ARRAY_LENGTH; counter++)
    {
        if (analog_vals[counter] > max)  // Find max value in array
        {
            max = analog_vals[counter];
        }

        if (analog_vals[counter] < min) // Find min value in array
        {
            min = analog_vals[counter];
        }
    }
}
    peak_peak = max - min;  // Calculate peak-to-peak voltage as max voltage - min voltage
    rms = sqrt(rms/1000);   // Calculate final true RMS voltage value
    rms_dig1 = (rms/100) % 10;  // "Splitting" RMS voltage into individual values to send to terminal...
    rms_dig2 = (rms/10) % 10;   // ... via UART
    rms_dig3 = (rms/1) % 10;
    peak_dig1 = (peak_peak/10000) % 10;
    peak_dig2 = (peak_peak/1000) % 10;
    peak_dig3 = (peak_peak/100) % 10;

    UART_esc_code(GREEN);   // Header text is green
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));  // Wait for TXBUF to be empty before writing to terminal
    UART_print("AC Measurements");
    UART_esc_code(WHITE);   // Change font color to white
    UART_esc_code(LEFT);    // Move cursor to the left of the screen to create level text lines
    UART_esc_code(NEWLINE); // Skip a line on the terminal
    UART_esc_code(NEWLINE);
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    UART_print("Peak-to-Peak Voltage: ");
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    EUSCI_A0 -> TXBUF = peak_dig1 + '0';    // Individually send peak-to-peak voltage value digits
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    UART_print(".");
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    EUSCI_A0 -> TXBUF = peak_dig2 + '0';
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    EUSCI_A0 -> TXBUF = peak_dig3 + '0';
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    UART_print(" V");
    UART_esc_code(NEWLINE);
    UART_esc_code(NEWLINE);
    UART_esc_code(LEFT);
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    UART_print("RMS Voltage: ");
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    EUSCI_A0 -> TXBUF = rms_dig1 + '0';    // Individually send true RMS voltage value digits
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    UART_print(".");
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    EUSCI_A0 -> TXBUF = rms_dig2 + '0';
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    EUSCI_A0 -> TXBUF = rms_dig3 + '0';
    while(!(EUSCI_A0 -> IFG & EUSCI_A_IFG_TXIFG));
    UART_print(" V");
    UART_esc_code(LEFT);
    UART_esc_code(NEWLINE);
    UART_esc_code(NEWLINE);
    draw_bar_graph(rms_dig1, rms_dig2, rms_dig3);   // Function for drawing voltage bar graph
    UART_esc_code(NEWLINE);
    UART_esc_code(NEWLINE);
```

```c
        UART_esc_code(NEWLINE);
        UART_esc_code(LEFT);
        UART_esc_code(CLEARDOWN);   // Clear screen from cursor down
}


void TIMER_A0_init(void)
{
    TIMER_A0->CCTL[3] &= ~TIMER_A_CCTLN_CCIFG;  // Clear interrupt flag

    TIMER_A0->CTL = (TIMER_A_CTL_SSEL__SMCLK    // Select SMCLK at 24MHz
                  |  TIMER_A_CTL_MC__CONTINUOUS // Select CONTINUOUS mode
                  |  TIMER_A_CTL_IE);           // Enable interrupts

    TIMER_A0->CCTL[3] = (TIMER_A_CCTLN_CM_1     // Capture on rising edge
                      |  TIMER_A_CCTLN_CCIS_1   // Select CCIxB
                      |  TIMER_A_CCTLN_CAP      // Capture mode
                      |  TIMER_A_CCTLN_CCIE);   // Capture/compare interrupt

    NVIC->ISER[0] = 1 << (TA0_N_IRQn);
    __enable_irq();
}


void COMP_init(void)
{
    C1_IN_PORT->SEL0 |= C1_IN;  // C1.0 = P6.7
    C1_IN_PORT->SEL1 |= C1_IN;

    C1_OUT_PORT->SEL0 |= C1_OUT;    // C1OUT = P7.2
    C1_OUT_PORT->SEL1 &= ~C1_OUT;
    C1_OUT_PORT->DIR |= C1_OUT;

    COMP_E1->CTL0 = 0;  // Initialize registers to 0 to make sure the correct settings are implemented
    COMP_E1->CTL1 = 0;
    COMP_E1->CTL2 = 0;

    COMP_E1->CTL0 = (COMP_E_CTL0_IPEN     // Enable comparator positive terminal
                  |  COMP_E_CTL0_IPSEL_0);// Positive terminal = Ch0, C1.0 = P6.7

    COMP_E1->CTL1 = (COMP_E_CTL1_ON       // Turn comparator on
                  |  COMP_E_CTL1_F        // Enable comparator filter
                  |  COMP_E_CTL1_FDLY_3); // Max out filter delay (~3000ns)

    COMP_E1->CTL2 = (COMP_E_CTL2_RS_1     // VCC applied to resistor ladder
                  |  COMP_E_CTL2_RSEL);   // VCC applied to negative terminal

    COMP_E1->CTL3 = COMP_E_CTL3_PD0;      // Disable buffer on Port Ch0
}


void TA0_N_IRQHandler(void)
{
    static int overflow = 0, new_val = 0, prev_val = 0;
    if (TIMER_A0->CCTL[3] & TIMER_A_CCTLN_COV)  // If overflow occurs
    {
        TIMER_A0->CCTL[3] &= ~TIMER_A_CCTLN_COV;    // Reset capture overflow
    }

    if (TIMER_A0->CTL & TIMER_A_CTL_IFG)    // If interrupt occurs
    {
        TIMER_A0->CTL &= ~TIMER_A_CTL_IFG;  // Clear interrupt flag
        overflow++; // Increment overflow for frequency calculations
    }

    if (TIMER_A0->CCTL[3] & TIMER_A_CCTLN_CCIFG)    // If capture occurs
    {
        TIMER_A0->CCTL[3] &= ~TIMER_A_CCTLN_CCIFG;   // Reset CC flag
        prev_val = new_val; // Store first value in CC register
        new_val = TIMER_A0->CCR[3]; // Store second value in CC register
        difference = (new_val - prev_val) + (overflow * MAX_VAL);   // Calculate difference for freq calculation
        overflow = 0;   // Reset overflow for next freq calculation
    }
}


void ADC14_IRQHandler(void)
{
    ADCflag = 1;    // Raise ADC interrupt flag
    val = ADC14->MEM[2];  // Store value in memory register 2 in global array
}


void draw_bar_graph(int voltage1, int voltage2, int voltage3)
{
    while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));    // Wait for TXBUF to be empty
    switch (voltage1)   // Switch cases to decide how many "|" to print depending on...
    {                   // ... voltage value
        case 0:
            UART_print("|||");
            break;

        case 1:
            UART_print("|||||||||||||||||");
            break;

        case 2:
            UART_print("|||||||||||||||||||||||||||||||");
            break;

        case 3:
            UART_print("|||||||||||||||||||||||||||||||||||||||||||||");
            break;
    }

    while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));
    switch (voltage2)
    {
        case 0:
            UART_print("");
            break;
        case 1:
            UART_print("|");
            break;
        case 2:
            UART_print("||");
            break;
        case 3:
            UART_print("|||");
            break;
        case 4:
            UART_print("||||");
            break;
        case 5:
            UART_print("|||||||");
            break;
        case 6:
            UART_print("||||||||");
            break;
        case 7:
            UART_print("|||||||||");
            break;
        case 8:
            UART_print("||||||||||");
            break;
        case 9:
            UART_print("|||||||||||");
            break;
    }

    while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));
    switch (voltage3)
    {
        case 2:
            UART_print("|");
            break;
        case 4:
            UART_print("||");
            break;
        case 6:
            UART_print("|||");
            break;
        case 8:
            UART_print("||||");
            break;
    }

    UART_esc_code(NEWLINE);
    UART_esc_code(NEWLINE);
    UART_esc_code(LEFT);
    while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));
    UART_print("0.0----0.5----1.0----1.5----2.0----2.5----3.0----3.5----4.0----4.5----5.0");    // Draw number line
}

uint8_t EUSCIA0_IRQHandler(void)
{
    int data;
    while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));
    if (EUSCI_A0->IFG & EUSCI_A_IFG_RXIFG)  // Check if computer keyboard pressed
    {
        data = EUSCI_A0->RXBUF; // Receive data from keyboard press

        if (data == 'D')
        {
            mode = DC;   // DC mode measurements terminal display
        }

        else if (data == 'A')
        {
            mode = AC;   // AC mode measurements terminal display
        }
    }
    return mode;
}
```

DCO.h

```c
// DCO Code

#ifndef DCO_H_
#define DCO_H_

#define CS_KEY_VAL 0x695A

#define CYCLES 3000000
#define FREQ_1_5_MHz CS_CTL0_DCORSEL_0
#define FREQ_3_MHz CS_CTL0_DCORSEL_1
#define FREQ_6_MHz CS_CTL0_DCORSEL_2
#define FREQ_12_MHz CS_CTL0_DCORSEL_3
#define FREQ_24_MHz CS_CTL0_DCORSEL_4

// function prototypes
void set_DCO(int);


#endif /* DCO_H_ */
```

DCO.c

```c
#include "msp.h"
#include "DCO.h"

void set_DCO(int frequency)
{

    CS->KEY = CS_KEY_VAL; // Unlock CS Registers
    CS->CTL0 = 0;


    if(frequency == FREQ_1_5_MHz)
        {
            CS->CTL0 = CS_CTL0_DCORSEL_0; // Set DCO to 1.5MHz

        }
    else if(frequency == FREQ_3_MHz)
        {
            CS->CTL0 = CS_CTL0_DCORSEL_1; // Set DCO to 3MHz

        }
    else if(frequency == FREQ_6_MHz)
        {
            CS->CTL0 = CS_CTL0_DCORSEL_2; // Set DCO to 6MHz

        }
    else if(frequency == FREQ_12_MHz)
        {
            CS->CTL0 = CS_CTL0_DCORSEL_3; // Set DCO to 12MHz

        }
    else if(frequency == FREQ_24_MHz)
        {
            CS->CTL0 = CS_CTL0_DCORSEL_4; // Set DCO to 24MHz

        }

    CS->CTL1 |= (CS_CTL1_SELS__DCOCLK) ;    // Write DCO to SMCLK
    CS->KEY = 0;    // Lock CS Registers
}
```

UART.h

```
// UART Code

#ifndef UART_H_
#define UART_H_

#define BRW_VAL 13  // Calculated values for SMCLK = 24MHz and 115200 baud rate
#define BRF_VAL 0
#define BRS_VAL 0x25

#define PUART P1    // UART port: P1
#define RX BIT2     // RX: P1.2
#define TX BIT3     // TX: P1.3
#define NULL '\0'   // Null character to indicate end of phrase
#define ESC 0x1B    // Escape code to enable escape commands
#define BOLD "[1m"  // Bold font
#define TOPLEFT "[H" // Moves cursor to the top left of the screen
#define GREEN "[32m" // Green font
#define LEFT "[100D" // Make value large enough such that cursor will always...
                     // ...move to the very left of the terminal screen
#define NEWLINE "[1B" // Skips line on terminal
#define WHITE "[37m"  // White font
#define CLEARSCREEN "[2J"   // Clears the entire terminal screen
#define CLEARDOWN "[0J" // Clears the terminal screen from the cursor down

void UART_init(void);   // Define UART.c functions
void UART_print(char write_string[]);
void UART_esc_code(char write_string[]);

#endif /* UART_H_ */
```

UART.c

```
--------------------------------------------------------------------------------------------------------------
#include "msp.h"
#include "DCO.h"
#include "UART.h"

void UART_init(void)
{
    EUSCI_A0->CTLW0 |= EUSCI_A_CTLW0_SWRST;  // Initialize software reset

    EUSCI_A0->CTLW0 = (EUSCI_A_CTLW0_SWRST | EUSCI_A_CTLW0_SSEL__SMCLK);    // Select SMCLK as BRCLK source
    EUSCI_A0->BRW = BRW_VAL;    // Configure clock prescaler of baud rate generator
    EUSCI_A0->MCTLW = (EUSCI_A_MCTLW_OS16   // Enable oversampling
                    | (BRF_VAL<<EUSCI_A_MCTLW_BRF_OFS)   // 1st modulation
                    | (BRS_VAL<<EUSCI_A_MCTLW_BRS_OFS)); // 2nd modulation

    PUART->SEL0 |= (RX|TX); // Configure UART pins
    PUART->SEL1 &= ~(RX|TX);

    EUSCI_A0->CTLW0 &= ~EUSCI_A_CTLW0_SWRST; // Clear software reset

    EUSCI_A0->IE |= EUSCI_A_IE_RXIE;    // Enable RX to trigger interrupt
    NVIC->ISER[0] = 1 << (EUSCIA0_IRQn);    // Enable UART interrupt
    __enable_irq();     // Enable global interrupts
}


void UART_print(char write_string[])
{
    int i = 0;  // Integer for checking characters in string
    while (write_string[i] != NULL)  // Write string until null character reached
    {
        while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));
        EUSCI_A0->TXBUF = write_string[i];
        i++;
    }
}


void UART_esc_code(char write_string[])
{
    int i = 0;  // Integer for checking characters in string
    while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));
    EUSCI_A0->TXBUF = 0x1B; // Send '0x1B' to enable escape commands
    while (write_string[i] != NULL)  // Write string until null character reached
    {
        while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));
        EUSCI_A0->TXBUF = write_string[i];
        i++;
    }
}
```

ADC.h

```
// ADC Code

#ifndef ADC_H_
#define ADC_H_


#define PORT_ADC P5     // Analog pin = P5.5
#define ANALOG_PIN BIT5
#define MEM_REG 2   // Memory register 2 used for ADC14
#define DEF_MAX -50000  // Max comparator value in for loop to calculate max of array
#define DEF_MIN 50000   // Min comparator value in for loop to calculate min of array
#define ARRAY_LENGTH 20 // 20 samples in array
#define EMPTY_ARRAY {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0} // Initialize an empty array
#define SLOPE 203   // Calculated from array values
#define INTERCEPT 75031 // Calculated from array values
#define CAL 10000   // Value used to calibrate voltages
#define ZERO 0
#define MAX 330
#define PLUS2 2  // Calibration value
#define PLUS3 3  // Calibration value

void ADC14_init(void);

#endif /* ADC_H_ */
```

ADC.c

```c
#include "msp.h"
#include "DCO.h"
#include "UART.h"
#include "ADC.h"


void ADC14_init(void)
{
    ADC14->CTL0 &= ~ADC14_CTL0_ENC;  // ADC14 disabled

    ADC14->CTL0 = ADC14_CTL0_SSEL__HSMCLK    // Select HSMCLK for ADC14
                | ADC14_CTL0_SHP    // Sample and hold pulse mode select
                | ADC14_CTL0_SHT0_0     // Sample time of 4 clocks, registers 0-7, 24-31
                | ADC14_CTL0_CONSEQ_0 // Select single conversion
                | ADC14_CTL0_ON;    // ADC14 on

    ADC14->CTL1 = (MEM_REG<<ADC14_CTL1_CSTARTADD_OFS)   // Start conversions at memory register 2;
                | ADC14_CTL1_RES__14BIT  // ADC 14-bit resolution
                | ADC14_CTL1_BATMAP;

    ADC14->MCTL[2] = ADC14_MCTLN_INCH_0     // Input channel = 0, P5.5
                   | ADC14_MCTLN_VRSEL_0;   // Select 3.3V = AVCC

    ADC14->CTL0 |= ADC14_CTL0_ENC | ADC14_CTL0_SC;   // ADC enabled, start sample and conversion

    PORT_ADC->SEL0 |= ANALOG_PIN;  // Configure analog pin, P5.5
    PORT_ADC->SEL1 |= ANALOG_PIN;

    ADC14->IER0 = ADC14_IER0_IE2; // Enable interrupt for memory control register 2
    NVIC->ISER[0] = 1 << (ADC14_IRQn);
    __enable_irq();
}
```

External References

[1] *MSP432P4xx SimpleLink™ Microcontrollers Technical Reference Manual*, Texas Instruments, Mar. 2015 [Revised June 2019].

[2] Texas Instruments, “MSP432P401R, MSP432P401M SimpleLink™ Mixed-Signal Microcontrollers,” MSP432P401R datasheet, Mar. 2015 [Revised June 2019].