Digital Multimeter

EE 329-01 Microprocessor-based Systems Design

Project #3, Spring 2021

Prepared by: Micah Jeffries

Prepared for: Professor Hummel

Date Submitted: June 4, 2021

I. What is the Digital Multimeter?

The digital multimeter utilizes the analog-to-digital converter (ADC) of the MSP432 microprocessor to measure the DC/AC voltages and the frequency of an input signal. The measurements will include DC offset, RMS, peak-to-peak, and frequency. All measurements are displayed simultaneously so the user does not have to switch modes for DC/AC measurements. The digital multimeter uses RS-232 protocol to communicate with the serial terminal where it will display the voltage and frequency information of the input signal. Realterm was the software application downloaded for the serial terminal in this project report, but other serial terminal applications will work. For the DC offset and RMS value, the digital multimeter also features a character-based bar-graph to display real time changes in both DC and AC voltage.

II. System Specifications

Parameter	Value	Unit
Power Supply Voltage	5	V
Operating Voltage	3.3	V
Baud Rate	115,200	bits / sec
Time Resolution	10,000	samples / sec
Bit Resolution	14	bits
Voltage Resolution	1	mV
Frequency Resolution	1	Hz
Clock Frequency	24	MHz
Operating Temperature	0 to 50	C
Communication Protocol	RS-232	N/A
Input Frequency Range	1 to 1000	Hz

Table 1: System Specifications

III. System Schematic

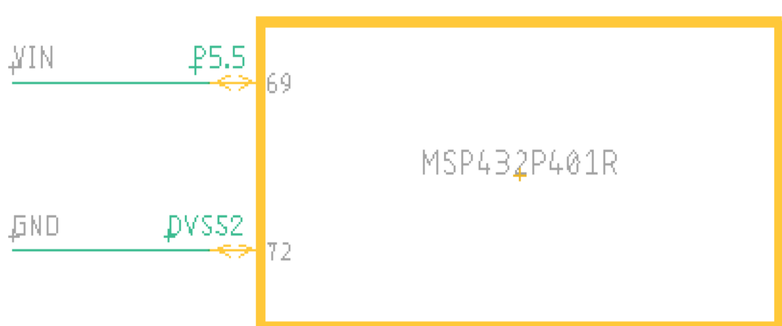

Figure 1: Schematic of Digital Multimeter

IV. Software Architecture

The code starts out by initiating all the built-in hardware subsystems in the MSP432 needed for this project. The clock system is first configured to run the internal clock frequency at 24 MHz. The ADC is configured for a single channel and single conversion where the conversion results are saved in memory register 0 of the ADC. The eUSCI peripheral is configured in UART mode where it will communicate to the serial terminal at a baud rate of 115,200 bps. The main function then enables all the necessary interrupts for the ADC to sample the input signal at an even interval of 100us.

Once all the settings are configured, the code begins sampling the input signal and storing the values in an array for later processing. The ADC samples the input signal 10,000 times which is enough samples to meet the Nyquist requirements of the 1 kHz wave and to capture the entire period of the 1 Hz wave. Once the signal has been sampled 10,000 times, the array is full and ready for processing.

The processing function first calculates the DC offset which is used in the next steps to calculate the frequency. The processing function iterates through the array and counts how many times the input wave crossed the DC offset. Since the signal was sampled at a sample rate of 100us/sample and there are 10,000 samples, the array represents one full second of the input wave. Thus, the frequency can be calculated by incrementing the number of times the signal oscillated between the DC offset. This strategy is essentially equivalent to a software implemented comparator where the reference voltage is the DC offset. The function next calculates all the DC/AC voltages and converts these digital values into mV. Lastly, the processing function will display all the given information on the serial terminal in a neat and organized manner. Refer to figure 2 below for an example of the measurements displayed on the serial terminal.

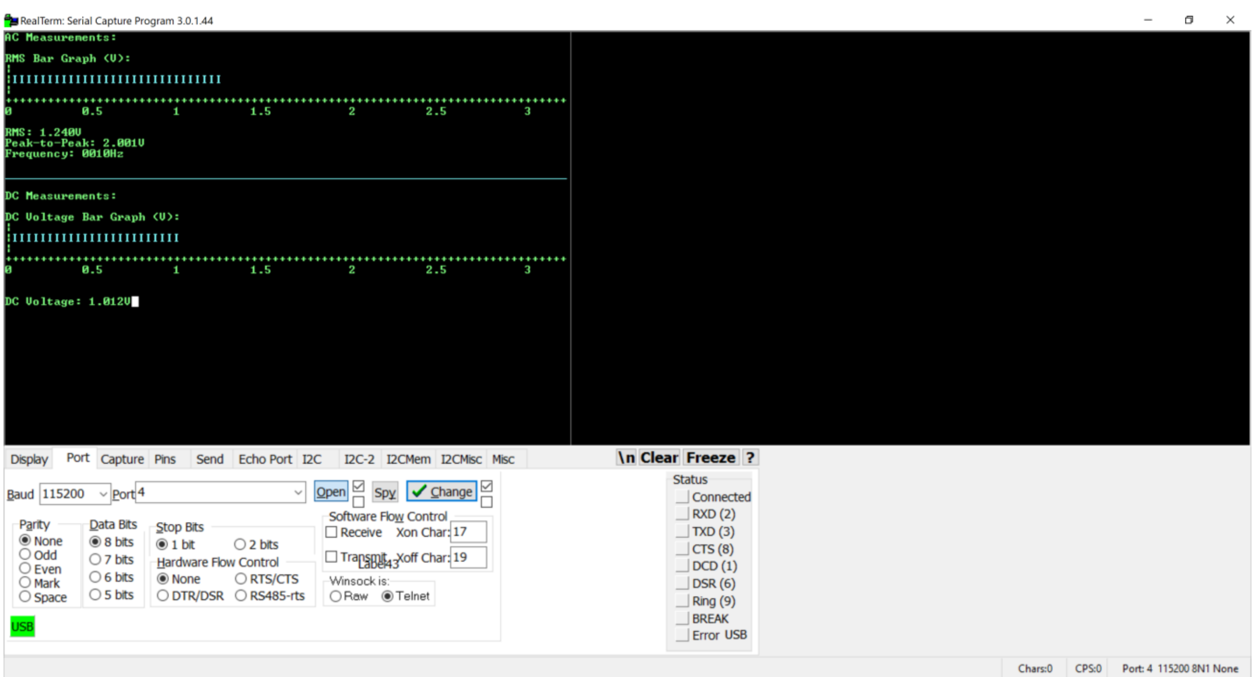

Figure 2: Screen Capture of the Digital Multimeter Terminal Display

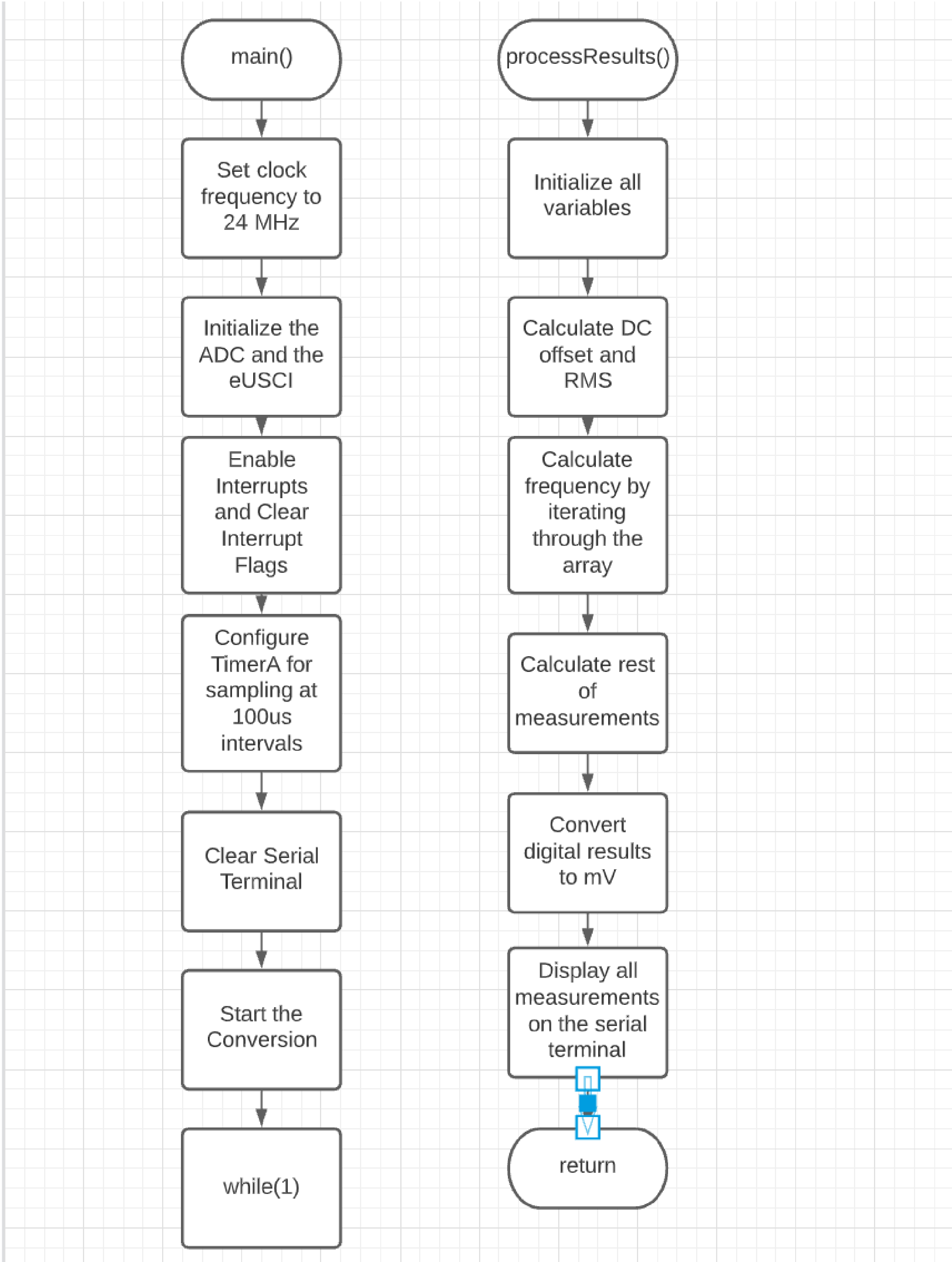

Figure 3: Flowcharts for Main and Array Processing Function

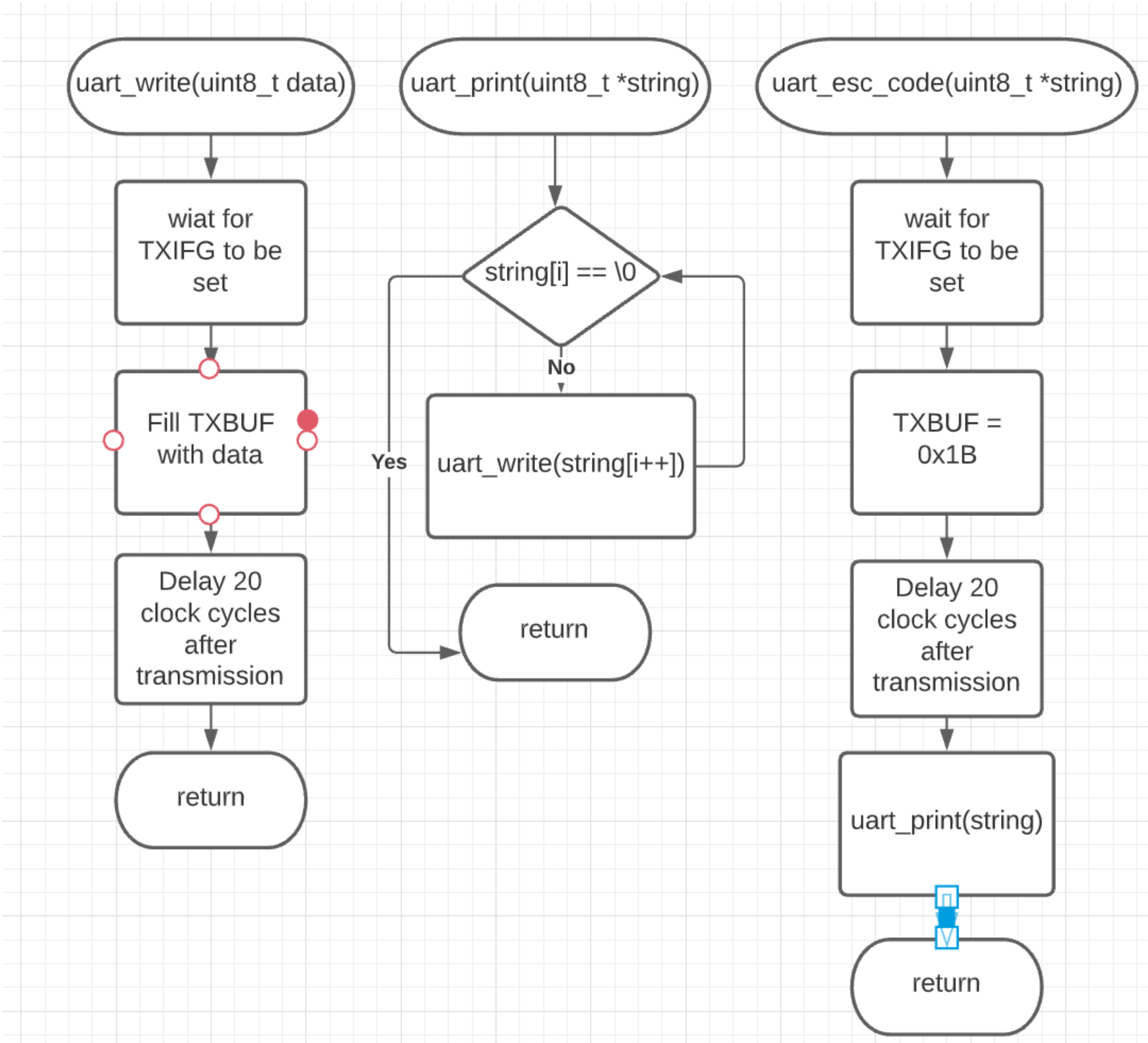

Figure 4: All UART Functions Used for Communicating with Serial Terminal

V. Appendix

A. Code

main.c

```
#include "msp.h"
#include "DCO.h"
#include "ADC.h"
#include "UART.h"
#include <math.h>

int i = 0;
uint16_t array[10000];
uint32_t avg = 0;
uint64_t RMS = 0;

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

    set_DCO(FREQ_24_MHZ);                           // set MCLK to 24 MHz
    ADC14_init();                                   // initialize the ADC
    uart_init();                                    // initialize the eUSCI
peripheral

    __enable_irq();                                 // global interrupt enable
    NVIC->ISER[0] = (1 << ADC14_IRQn);              // enable ADC interrupt
    ADC14->IER0 |= (ADC14_IER0_IE0);                // enable ADC interrupt for
memory location 0
    ADC14->CLRIFGR0 |= (ADC14_IFGR0_IFG0);          // clear interrupt flag for
memory location 0
    ADC14->CLRIFGR0 &= ~(ADC14_IFGR0_IFG0);

    NVIC->ISER[0] = (1 << TA0_0_IRQn);              // enable CCTL0.CCIFG
    TA0CCTL0 |= (TIMER_A_CCTLN_CCIE);               // enable CCIFG interrupt
    TA0CCTL0 &= ~(TIMER_A_CTL_IFG);                 // clear interrupt flag

    TA0CCR0 = PERIOD;                               // configure TimerA to trigger
interrupts every 100 us
    TIMER_A0->CTL = (TIMER_A_CTL_SSEL__SMCLK        // set timerA0 clock source to
SMCLK
                   | TIMER_A_CTL_MC__UP);           // set timerA0 counting mode to
up

    uart_esc_code("[1m\0");                         // turn on bold mode
    uart_esc_code("[32m\0");                        // change text color to green
    uart_esc_code("[2J\0");                         // clear the screen

    ADC14->CTL0 |= (ADC14_CTL0_SC);                 // start the conversion

    while(1);
}

void displayFrequency(uint32_t frequency) {
```

```
    uint8_t digit_1 = 0, digit_2 = 0, digit_3 = 0, digit_4 = 0;
    uint8_t string[7] = {'0', '0', '0', '0', 'H', 'z', '\0'};

    digit_1 = frequency/1000;                               // calculate digits from
frequency value to be sent to serial terminal
    digit_2 = (frequency%1000)/100;
    digit_3 = ((frequency%1000)%100)/10;
    digit_4 = ((frequency%1000)%100)%10;

    string[0] = digit_1 + '0';                              // modify the initiated string
    string[1] = digit_2 + '0';
    string[2] = digit_3 + '0';
    string[3] = digit_4 + '0';

    uart_esc_code("[1B\0");                                 // move cursor down 1
    uart_esc_code("[20D\0");                                // move cursor left 20
    uart_print("Frequency: \0");                            // print frequency to the
terminal
    uart_print(string);

    uart_esc_code("[2B\0");                                 // move cursor down 2
    uart_esc_code("[17D\0");                                // move cursor left 17

}

void displayDCvoltage(uint32_t avg) {

    int j = 0;
    uint8_t digit_1 = 0, digit_2 = 0, digit_3 = 0, digit_4 = 0;
    uint8_t string[7] = {'0', '.', '0', '0', '0', 'V', '\0'};

    digit_1 = avg/1000;                                     // calculate digits from DC
voltage value to be sent to serial terminal
    digit_2 = (avg%1000)/100;
    digit_3 = ((avg%1000)%100)/10;
    digit_4 = ((avg%1000)%100)%10;

    string[0] = digit_1 + '0';                              // modify the initiated string
    string[2] = digit_2 + '0';
    string[3] = digit_3 + '0';
    string[4] = digit_4 + '0';

    uart_esc_code("[H\0");                                  // move cursor to top left
    uart_esc_code("[13B\0");                                // move cursor down 13
    uart_esc_code("[36m\0");                                // change text color to light
blue

uart_print("_________________________________________________________________________
_______\0");

    uart_esc_code("[32m\0");                                // change text color to green
    uart_esc_code("[80D\0");                                // move cursor left 80
    uart_esc_code("[2B\0");                                 // move cursor down 2
    uart_print("DC Measurements: \0");
```

```
    uart_esc_code("[17D\0");                        // move cursor left 17
    uart_esc_code("[2B\0");                         // move cursor down 2
    uart_print("DC Voltage Bar Graph (V): \0");

    uart_esc_code("[26D\0");                        // move cursor left 26
    uart_esc_code("[1B\0");                         // move cursor down 1
    uart_print("|\0");
    uart_esc_code("[1D\0");                         // move cursor left 1
    uart_esc_code("[1B\0");                         // move cursor down 1
    uart_print("|\0");
    uart_esc_code("[1D\0");                         // move cursor left 1
    uart_esc_code("[1B\0");                         // move cursor down 1
    uart_print("|\0");
    uart_esc_code("[1D\0");                         // move cursor left 1
    uart_esc_code("[1B\0");                         // move cursor down 1

uart_print("+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++\0");
    uart_esc_code("[80D\0");                        // move cursor left 80
    uart_esc_code("[1B\0");                         // move cursor down 1
    uart_print("0          0.5          1          1.5           2          2.5
3     \0");

    uart_esc_code("[80D\0");                        // move cursor left 80
    uart_esc_code("[3A\0");                         // move cursor up 3
    uart_esc_code("[1C\0");                         // move cursor right 1
    uart_esc_code("[0K\0");                         // clear the line from the cursor
to the right
    uart_esc_code("[36m\0");                        // change text color to light
blue

    for (j = 0; j < (avg / 40 - 1); j++)            // display bar graph with length
proportional to DC voltage
        uart_print("I\0");

    uart_esc_code("[32m\0");                        // change text color to green
    uart_esc_code("[H\0");                          // move cursor to top left
    uart_esc_code("[25B\0");                        // move cursor down 25
    uart_print("DC Voltage: \0");                   // print DC voltage to the
terminal
    uart_print(string);

}

void displayPK2PK(uint32_t Pk2pk) {

    uint8_t digit_1 = 0, digit_2 = 0, digit_3 = 0, digit_4 = 0;
    uint8_t string[7] = {'0', '.', '0', '0', '0', 'V', '\0'};

    digit_1 = Pk2pk/1000;                           // calculate digits from peak-to-
peak value to be sent to serial terminal
    digit_2 = (Pk2pk%1000)/100;
    digit_3 = ((Pk2pk%1000)%100)/10;
    digit_4 = ((Pk2pk%1000)%100)%10;
```

```
    string[0] = digit_1 + '0';                          // modify the initiated string
    string[2] = digit_2 + '0';
    string[3] = digit_3 + '0';
    string[4] = digit_4 + '0';

    uart_esc_code("[1B\0");                             // move cursor down 1
    uart_esc_code("[11D\0");                            // move cursor left 11
    uart_print("Peak-to-Peak: \0");                     // print peak-to-peak voltage to
the terminal
    uart_print(string);

}

void displayRMS(uint64_t RMS) {

    int j = 0;
    uint8_t digit_1 = 0, digit_2 = 0, digit_3 = 0, digit_4 = 0;
    uint8_t string[7] = {'0', '.', '0', '0', '0', 'V', '\0'};

    digit_1 = RMS/1000;                                 // calculate digits from RMS
value to be sent to serial terminal
    digit_2 = (RMS%1000)/100;
    digit_3 = ((RMS%1000)%100)/10;
    digit_4 = ((RMS%1000)%100)%10;

    string[0] = digit_1 + '0';                          // modify the initiated string
    string[2] = digit_2 + '0';
    string[3] = digit_3 + '0';
    string[4] = digit_4 + '0';

    uart_esc_code("[H\0");                              // move cursor to top left
    uart_print("AC Measurements: \0");

    uart_esc_code("[H\0");                              // move cursor to top left
    uart_esc_code("[2B\0");                             // move cursor down 2
    uart_print("RMS Bar Graph (V): \0");

    uart_esc_code("[H\0");                              // move cursor to top left
    uart_esc_code("[3B\0");                             // move cursor down 3
    uart_print("|\0");
    uart_esc_code("[1D\0");                             // move cursor left 1
    uart_esc_code("[1B\0");                             // move cursor down 1
    uart_print("|\0");
    uart_esc_code("[1D\0");                             // move cursor left 1
    uart_esc_code("[1B\0");                             // move cursor down 1
    uart_print("|\0");
    uart_esc_code("[1D\0");                             // move cursor left 1
    uart_esc_code("[1B\0");                             // move cursor down 1

uart_print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++\0");
    uart_esc_code("[80D\0");                            // move cursor left 80
    uart_esc_code("[1B\0");                             // move cursor down 1
```

```
    uart_print("0          0.5          1          1.5          2          2.5          
3     \0");

    uart_esc_code("[H\0");                              // move cursor to top left
    uart_esc_code("[4B\0");                             // move cursor down 4
    uart_esc_code("[1C\0");                             // move cursor right 1
    uart_esc_code("[0K\0");                             // clear the screen from the 
cursor to the right
    uart_esc_code("[36m\0");                            // change the text color to light 
blue

    for (j = 0; j < (RMS / 40 - 1); j++)                // display bar graph with length 
proportional to RMS
        uart_print("I\0");

    uart_esc_code("[32m\0");                            // change the text color to green
    uart_esc_code("[H\0");                              // move cursor to top left
    uart_esc_code("[9B\0");                             // move cursor down 9
    uart_print("RMS: \0");                              // print RMS to the terminal
    uart_print(string);

}

void processResults(void) {

    int j = 0;
    uint32_t frequency = 0, min = 0xFFFFFFFF, max = 0;

    typedef enum {                                      // define voltage states
        POSITIVE,
        NEGATIVE
    } VSTATE;

    VSTATE Vstate = POSITIVE;                           // start in the positive state

    avg = avg / 10000;                                  // calculate the DC offset
    RMS = sqrt(RMS / 10000);                            // calculate the RMS
    RMS = 0.2 * RMS - 20;                               // convert digital RMS to voltage 
RMS

    for (j = 0; j < 10000; j++) {                       // calculate the frequency (count 
how many times the voltage crosses the DC voltage in one sec)
        switch(Vstate) {
            case POSITIVE:
                if (array[j] < avg - 500) {
                    Vstate = NEGATIVE;
                }
                break;

            case NEGATIVE:
                if (array[j] > avg + 500) {
                    frequency += 1;
                    Vstate = POSITIVE;
                }
                break;
```

```
            default:
                Vstate = POSITIVE;
        }

        if (array[j] > max)                                 // find the max value
            max = array[j];

        if (array[j] < min)                                 // find the min value
            min = array[j];
    }

    avg = avg * 0.2 - 20;                                   // convert digital avg to voltage
avg
    max = max * 0.2 - 20;                                   // convert digital max to voltage
max
    min = min * 0.2 - 20;                                   // convert digital min to voltage
min

    displayRMS(RMS);                                        // display the RMS on the
terminal
    displayPK2PK(max - min);                                // display the peak-to-peak
voltage on the terminal
    displayFrequency(frequency);                            // display the frequency on the
terminal
    displayDCvoltage(avg);                                  // display the DC voltage on the
terminal

    avg = RMS = 0;                                          // reset avg and RMS
}

void TA0_0_IRQHandler (void) {

    TA0CCTL0 &= ~(TIMER_A_CTL_IFG);                         // clear interrupt flag
    ADC14->CTL0 |= (ADC14_CTL0_SC);                         // start the conversion

}

void ADC14_IRQHandler(void) {

    if (i == 10000) {                                       // is the array ready for
processing?
        i = 0;
        processResults();
    }

    array[i++] = ADC14->MEM[0];                             // store the converted value in
the array for further processing
    avg += ADC14->MEM[0];                                   // increment DC voltage for later
processing
    RMS += ADC14->MEM[0] * ADC14->MEM[0];                   // increment RMS for later
processing
}
```

ADC.h

```c
#ifndef ADC_H_
#define ADC_H_

#define ANALOG_PORT P5
#define ANALOG_IN BIT5

void ADC14_init(void);

#endif /* ADC_H_ */
```

ADC.c

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

void ADC14_init(void) {

    ADC14->CTL0 &= ~(ADC14_CTL0_ENC             // disable the enable and start
options
                   | ADC14_CTL0_SC);
    ADC14->CTL0 |= (ADC14_CTL0_SSEL__SMCLK      // configure settings for ADC
                  | ADC14_CTL0_CONSEQ_0
                  | ADC14_CTL0_SHS_0
                  | ADC14_CTL0_SHP
                  | ADC14_CTL0_SHT0_0
                  | ADC14_CTL0_SHT1_0
                  | ADC14_CTL0_ON);

    ADC14->CTL1 |= (ADC14_CTL1_RES_3);          // 14 bit resolution
    ADC14->MCTL[0] |= (ADC14_MCTLN_INCH_0       // analog pin and voltage reference
select
                     | ADC14_MCTLN_VRSEL_0);

    ANALOG_PORT->SEL0 |= (ANALOG_IN);           // configure the analog port
    ANALOG_PORT->SEL1 |= (ANALOG_IN);

    ADC14->CTL0 |= (ADC14_CTL0_ENC);            // enable the ADC

}
```

DCO.h

```c
#ifndef DCO_H_
#define DCO_H_

#define CPU_FREQ 24000000
#define __delay_us(t_us) (__delay_cycles((((uint64_t)t_us)*CPU_FREQ) / 1000000))

#define FREQ_15_MHZ 1500000
#define FREQ_3_MHZ 3000000
```

```c
#define FREQ_6_MHZ 6000000
#define FREQ_12_MHZ 12000000
#define FREQ_24_MHZ 24000000

#define PERIOD 2400

void set_DCO(uint32_t frequency);

#endif /* DCO_H_ */
```

DCO.c

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

void set_DCO(uint32_t frequency){

    CS->KEY = CS_KEY_VAL; // unlock CS registers

    switch (frequency) {
        case FREQ_15_MHZ:
            CS->CTL0 = (CS_CTL0_DCORSEL_0); // set DCO to 1.5 MHz
            break;

        case FREQ_3_MHZ:
            CS->CTL0 = (CS_CTL0_DCORSEL_1);   // set DCO to 3 MHz
            break;

        case FREQ_6_MHZ:
            CS->CTL0 = (CS_CTL0_DCORSEL_2);   // set DCO to 6 MHz
            break;

        case FREQ_12_MHZ:
            CS->CTL0 = (CS_CTL0_DCORSEL_3);  // set DCO to 12 MHz
            break;

        case FREQ_24_MHZ:
            CS->CTL0 = (CS_CTL0_DCORSEL_4);  // set DCO to 24 MHz
            break;

        default: break;
    }

    CS->CTL1 = (CS_CTL1_DIVM__1 |       // MCLK / 1
                CS_CTL1_SELS__DCOCLK | // SMCLK / HSMCLK using DCO
                CS_CTL1_SELM__DCOCLK); // MCLK using DCO

    CS->KEY = 0; // lock CS registers

}
```

UART.h

```
ifndef UART_H_
#define UART_H_

#define UART_PORT P1

#define UART_RXD BIT2
#define UART_TXD BIT3

void uart_init(void);
void uart_write(uint8_t uart_data);
void uart_print(uint8_t *string);
void uart_esc_code(uint8_t *string);

#endif /* UART_H_ */
```

UART.c

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

void uart_init(void) {
    EUSCI_A0->CTLW0 |= EUSCI_A_CTLW0_SWRST;             // put the eUSCI into
sowftware reset
    EUSCI_A0->CTLW0 = (EUSCI_A_CTLW0_MODE_0
                     | EUSCI_A_CTLW0_SSEL__SMCLK
                     | EUSCI_A_CTLW0_SWRST);

    EUSCI_A0->BRW = 13;                                 // clock divider at 13
    EUSCI_A0->MCTLW |= EUSCI_A_MCTLW_OS16;              // enable oversampling
    EUSCI_A0->MCTLW |= ((0 << EUSCI_A_MCTLW_BRF_OFS)    // Set first modulation stage
                    & EUSCI_A_MCTLW_BRF_MASK);
    EUSCI_A0->MCTLW |= ((0x25 << EUSCI_A_MCTLW_BRS_OFS) // Set second modulation
stage
                    & EUSCI_A_MCTLW_BRS_MASK);

    UART_PORT->SEL0 |= (UART_RXD | UART_TXD);           // configure UART pins
    UART_PORT->SEL1 &= ~(UART_RXD | UART_TXD);

    EUSCI_A0->CTLW0 &= ~(EUSCI_A_CTLW0_SWRST);          // clear software reset
}

void uart_write(uint8_t uart_data) {

    while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));    // wait for TXIFG to be set
(TXBUF is empty)
    EUSCI_A0->TXBUF = uart_data;

    __delay_cycles(20);                         // delay after transmission
}

void uart_print(uint8_t *string) {
```

```c
    int i = 0;

    while (string[i] != '\0') {
        uart_write(string[i++]);
    }

}

void uart_esc_code(uint8_t *string) {

    while(!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG));     // wait for TXIFG to be set
(TXBUF is empty)
    EUSCI_A0->TXBUF = 0x1B;

    __delay_cycles(20);                         // delay after transmission

    uart_print(string);
}
```

B. References

[1] Texas Instruments, Dallas TX, United States, *MSP432P4xx SimpleLink™ Microcontrollers Technical Reference Manual*, Accessed: June 2, 2021. [Online]. Available: https://www.ti.com/lit/ug/slau356i/slau356i.pdf

[2] Texas Instruments, Dallas TX, United States, *MSP432P401R, MSP432P401M SimpleLink™ Mixed Signals Microcontrollers datasheet,* Accessed: June 2, 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/msp432p401r.pdf?ts=1622665321421&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP432P401R%253FkeyMatch%253DMSP432P401R%2526tisearch%253Dsearch-everything%2526usecase%253DGPN