

Project 3

Digital multimeter

Kyle Rosenthal | CPE 329-05-Gerfen | 5/18/2017 | Spring 2017

## Link to Video

Part 1: <https://www.youtube.com/watch?v=3XCzjylnndw>

Part 2:<https://www.youtube.com/watch?v=lG7fFXdDqsw>

Part 3\*:<https://www.youtube.com/watch?v=VMEhyfqCo6A>

\*redid triangle wave with new threshold

# Purpose

The purpose of this project was to design a digital multimeter with AC and DC functionalities. This project will enable a greater understanding of frequency and its interaction with the digital world. In addition this project demonstrates a effective terminal interface for VT100 compatible terminals. Furthermore, the ability to go above 1000 Hz and store values allows for much greater usability for the multimeter.

# System Requirements

* The Digital Multimeter (DMM) shall measure voltage.
  + Voltage measurements shall be limited to 0 to 3.3 volts.
  + Voltage measurements shall be limited to 0 to 1000 Hz.
  + Voltage measurements shall be accurate to +/- 1 mv for AC and DC.
  + The DMM shall have a DC setting
    - DC measurements shall average over a 1 ms time period
    - DC measurements of a sinusoidal waveform should be equivalent to the DC offset of the sinusoid
  + The DMM shall have a AC setting
    - AC measurements shall be true-RMS
    - AC measurements shall display the various components.
      * AC voltage measurements shall give the TrueRMS (includes DC offset)
      * AC voltage measurements shall give the CalcRMS (TrueRMS – DC offset)
      * AC voltage measurements shall give the peak-to-peak value.
    - AC measurements shall work for various waveforms
      * Sine waves shall be measurable.
      * Triangular waves shall be measurable
      * Square waves shall be measurable
      * Other periodic waveforms shall be measurable
    - AC measurements shall work for waveforms of various amplitudes and offsets
      * The maximum voltage that shall be measured is 3V
      * The minimum voltage that shall be measured is 0V
      * The minimum peak-to-peak voltage that shall be measured is 0.5V
      * Offset values of up to 2.75V shall be measurable
* The DMM shall measure frequency
  + Frequency measurements shall be limited to 1 to 2000 HZ.
  + Frequency measurements shall be accurate to within 1 Hz.
  + Frequency measurements shall work for various waveforms.
    - Sine waves shall be measurable.
    - Triangular waves shall be measurable.
    - Square waves shall be measurable.
    - Other periodic waves shall be measurable.
* The DMM shall have a terminal-based interface
  + The terminal shall operate at a frequency greater than 9600 baud
  + The terminal shall utilize the VT100 protocol.
    - The terminal shall display all fields in non-changing locations.
  + The terminal shall display AC voltages as described above.
  + The terminal shall display DC voltages as described above.
  + The terminal shall display frequency as described above.
  + The terminal shall organize the presentation of information.
    - AC, DC, and frequency shall be simple to read.
    - The display may use horizontal and vertical lines (borders) to organize the presentation of information.
  + The terminal shall use bar-graphs for voltages being measured.
    - The terminal shall have a bar-graph for “Calc-RMS”.
    - The terminal shall have a bar-graph for DC voltages.
    - The bar graphs shall be shall have delineators, e.g. a scale, indicating the equivalent voltage being measured.
    - The bar graphs shall be a single line of pixels, characters, etc.
    - The bar graphs shall have length that is proportional to the voltage being measured.
    - The bar graphs shall respond in real-time to changes in AC or DC voltage
  + The terminal shall display stored values.
    - The stored values shall be set whenever the system receives a return from the UART.
    - All 5 stored values shall be displayed at the bottom of the terminal interface in a static manner.

# System Specifications

|  |  |  |
| --- | --- | --- |
| Component | Spec | Value |
| MSP432 | Model | MSP432P401R |
|  | Frequency | 24 MHz |
|  | Interrupts | Enabled |
|  | Input Power | 5 V |
| UART | Baud Rate | 115200 |
|  | UART Data Bits | 8 |
|  | UART Mode | 8N1 |

*Table 1 – System Specifications.*

# System Architecture

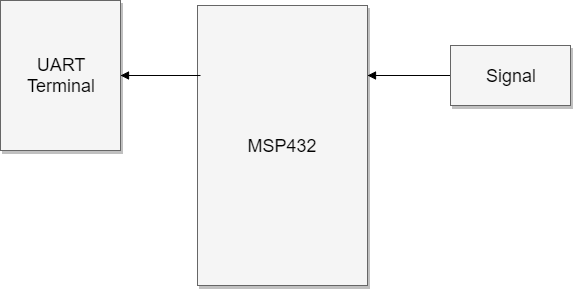


Figure – System Architecture Diagram

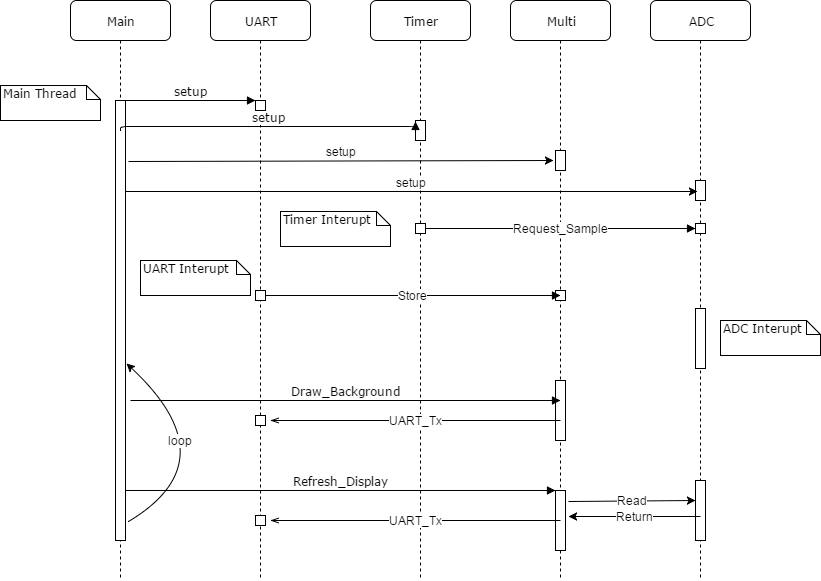


Figure 2 – System Software Diagram

# Component Design

All setting are default settings of MSP432P401R Launchpad running at 3 MHz with interrupts enabled. The five

* main.c
  + main()
    - calls all the setup functions
    - loops and call the display refresh
* uart.c
  + Setup\_UART()
    - Configures the UART
    - Enables interrupt
  + UART0TX(char)
    - Sends a single char
  + UART\_Strign(char \*, int)
    - Sends a string
  + IRQHandler()
    - Call store if a return is received
* adc.c
  + Setup\_ADC()
    - Sets up interrupts and zeros values.
  + RequestSample()
    - Enables the collection of a single adc sample
  + GetRawValue()
    - Gets the raw unconverted UDAC reading for the DC value. This allows more precision but not in a standard unit.
  + GetRawValueAC()
    - Gets the raw unconverted UDAC reading for the AC value. This allows more precision but not in a standard unit.
  + GetFormated\*(char \*)
    - Pastes a formatted in standard units value into the char array.
    - (\*) one of each of the 5 measurement values
      * AC, Vrms true, Vrms calc, V peak to peak, frequency.
  + IRQ\_Handler()
    - Updates the stored values.
* multi.h
  + Setup\_multi()
    - Zeros out the storage values
  + Draw\_Background()
    - Draws the background of the terminal interface.
  + Refresh\_Display()
    - Draws the values for the terminal interface.
  + Store()
    - Stores the displayed values.
* timer.h
  + Timer\_Setup()
    - Enables the timer and turns on the interrupt.
  + IRQ\_Handler()
    - Calls RequestSample from adc.h

## Schematic

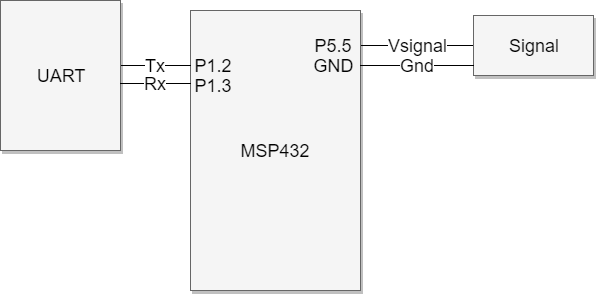


Figure 3 – Schematic Diagram

# Bill of Materials

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Item # | Part # | Supplier | Quantity | Price Ea | Total Price $ |
| MSP432 Launchpad | 1 | MSP432P401R | Digikey | 1 | 13.03 | 13.03 |
| Jumper cables | 2 | 0 | Amazon | 3 | .01 | .03 |
| Breadboard | 5 | 352 | Pololu | 1 | 3.97 | 3.97 |
| Total |  |  |  |  |  | 17.03 |
|  |  |  |  |  |  |  |

*Table 2 – Bill of Materials.*

# System Integration

The beginning of this system integration was creating a terminal interface. This means helper functions were needed for the UART. The ability to use the UART with a string at a time greatly improved the ability to design a clean UI. The next step was measuring DC values while ignoring AC fluctuations. This means designing a system that could average away those fluctuations but could also adapt for new waveforms. A pipeline was used with a pointer to the current node in the pipe. This pipeline was increases and balanced with the ADC rate to create a 4000 measurement wide pipeline that is filled once per second.

The resulting pipeline enables a quick access for computing RMS values. The more difficult part was measuring the frequency. A up and down counter are used that measure the rise and fall times and use the two together to measure the frequency. This is a nice solution because it scales with higher sample rates, wider frequency ranges, or longer pipelines.

A weird bug that turned up was with calculating the RMS values. It turned out the pipeline filled up beyond the limits of an int and was causing unpredictable behavior. This was resolved by switching it to a double as well as multiplying it to get better resolution when taking the square root.

# Conclusion

In conclusion, the multimeter was success, despite some difficulties along the way. Designing a mechanism to measure the samples and keep track of the values was a difficult task. *It is highly recommended to whiteboard out possible solutions for measuring the frequency. It is important to keep track of which operation cost more time because by limiting those it will allow for greater sampling rates or just extra time when you need it later. Furthermore, when doing conversions keep track of the values to ensure that it does not go outside of the bound for that number type for that could cause otherwise hard to see bugs.* While trying to debug the two interrupts canceling each other out, lots of interesting and worthwhile information was discovered about timers and interrupts that will help in future endeavors. There was an attempt to add resistance measurement to this project but the results were unstable at best and as a result, measuring resistance was sidelined for other improvements.

# Appendices

## Referances

* CPE 329 - Project 2 – Function Genorator v0.02 - S2017
* [MSP432 - Technical Reference Manual File](http://www.ti.com/general/docs/litabsmultiplefilelist.tsp?literatureNumber=slau356f)
* [MCP4921 – Spec Sheet](http://ww1.microchip.com/downloads/en/devicedoc/21897b.pdf)
* Schematic created with: http://www.draw.io/

## Code

### main.c

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// MSP432 main.c

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "msp.h"

#include "adc.h"

#include "uart.h"

#include "timer.h"

#include "multi.h"

void main(void)

{

WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer

// DCO = 3 MHz, SMCLK and MCLK = DCO

CS->KEY = CS\_KEY\_VAL;

CS->CTL0 = 0;

CS->CTL0 = CS\_CTL0\_DCORSEL\_1; // DCO = 3 MHz

CS->CTL1 = CS\_CTL1\_SELA\_2 | CS\_CTL1\_SELS\_3 | CS\_CTL1\_SELM\_3;

CS->KEY = 0;

\_\_enable\_irq();

P3->DIR |= BIT2;

Setup\_UART();

Setup\_Multi();

Setup\_ADC(33,0);

Timer\_Setup();

\_\_enable\_irq();

int i = 0;

Draw\_Background();

while (1) {

//ADC\_RequestNextSample();

if (i%4 == 0) {

Refresh\_Display();

}

i++;

if (i%1000 == 0) {

Draw\_Background();

}

}

}

### uart.h

/\*

\* uart.h

\*

\* Created on: May 8, 2017

\* Author: kmrosent

\*/

#ifndef UART\_H\_

#define UART\_H\_

int statusFlag;

int val;

void Setup\_UART();

unsigned char UART0Rx(void);

unsigned char UART0Tx(unsigned char c);

void UART\_String(char \* str, int len, int newline);

void EUSCIA0\_IRQHandler(void);

#endif /\* UART\_H\_ \*/

### uart.c

/\*

\* uart.c

\*

\* Created on: May 8, 2017

\* Author: kmrosent

\*/

#include "uart.h"

#include "multi.h"

#include "msp.h"

/\*\* sets up the uart \*/

void Setup\_UART() {

val = 0;

statusFlag = 0;

EUSCI\_A0->CTLW0 |= EUSCI\_A\_CTLW0\_SWRST;

EUSCI\_A0->MCTLW = 0;

EUSCI\_A0->CTLW0 = EUSCI\_A\_CTLW0\_SSEL1 + EUSCI\_A\_CTLW0\_SWRST;

EUSCI\_A0->BRW = 26;

P1SEL0 |= (BIT2 + BIT3);

P1SEL1 &= ~(BIT2 + BIT3);

EUSCI\_A0->CTLW0 &= ~EUSCI\_A\_CTLW0\_SWRST;

EUSCI\_A0->IFG |= EUSCI\_A\_IFG\_RXIFG;

EUSCI\_A0->IE |= EUSCI\_A\_IE\_RXIE;

//NVIC\_SetPriority(EUSCIA0\_IRQn, 4);

NVIC\_EnableIRQ(EUSCIA0\_IRQn);

}

/\* read a character from UART0 \*/

unsigned char UART0Rx(void) {

char c;

while(!(EUSCI\_A0->IFG & 0x01)) ;

c = EUSCI\_A0->RXBUF;

return c;

}

/\* write a character to UART \*/

unsigned char UART0Tx(unsigned char c) {

while(!(EUSCI\_A0->IFG&0x02)) ;

EUSCI\_A0->TXBUF = c;

return c;

}

/\* write a string to UART \*/

void UART\_String(char \* str, int len, int newline) {

int i;

for (i = 0; i < len; i++) {

UART0Tx(str[i]);

}

if (newline == 1) {

UART0Tx('\033');

UART0Tx('E');

}

}

/\*\*

\* Handle interrupt and if ‘return’ calls Store()

\*/

void EUSCIA0\_IRQHandler(void) {

char c = EUSCI\_A0->RXBUF;

if (c == '\r') {

Store();

}

//while(!(EUSCI\_A0->IFG & 0x02)) {}

//EUSCI\_A0->TXBUF = c;

}

### adc.h

/\*

\* adc.h

\*

\* Created on: May 10, 2017

\* Author: kmrosent

\*/

#ifndef ADC\_H\_

#define ADC\_H\_

#define F\_ADC\_READ\_ME 0

#define F\_ADC\_REQUEST 1

#define F\_ADC\_NO\_OP 2

#define CAL 79 / 428 + 30

#define THRESH 100

#define SAMPLES 4000

#define NUM\_FREQ 20

#define RESISTOR 194000

int vL, vH, adcflag;

unsigned int lastRead[SAMPLES];

unsigned int pos;

unsigned int avg;

unsigned long long rms;

int freq[20];

int freqAvg;

int freqPos;

int upCount;

int dCount;

int max[2];

int min[2];

void Setup\_ADC(int v\_h, int v\_l);

void ADC\_RequestNextSample();

void ADC14\_IRQHandler();

int ADC\_CheckReady();

unsigned int ADC\_GetRawValue();

unsigned int ADC\_GetRawValueAC();

unsigned int ADC\_GetRawValueOhm();

void ADC\_GetFormatedDC(char\* value);

void ADC\_GetFormatedAC(char\* value);

void ADC\_GetFormatedAC\_Calc(char\* value);

void ADC\_GetFormatedVpp(char\* value);

void ADC\_GetFormatedFreq(char\* value);

void ADC\_GetFormatedOhm(char\* value);

#endif /\* ADC\_H\_ \*/

### adc.c

/\*

\* adc.c

\*

\* Created on: May 10, 2017

\* Author: kmrosent

\*/

#include "adc.h"

#include "msp.h"

#include <math.h>

/\* sets up the adc and zeros pipes \*/

void Setup\_ADC(int v\_h, int v\_l) {

int i;

vL = v\_l;

vH = v\_h;

adcflag = F\_ADC\_NO\_OP;

pos = 0;

avg = 0;

upCount = 0;

dCount = 0;

freqAvg = 0;

freqPos = 0;

for (i = 0; i < NUM\_FREQ; i++) {

freq[i] = 0;

}

for (i = 0; i < SAMPLES; i++) {

lastRead[i] = 0;

}

P5->SEL0 |= BIT5;

P5->SEL1 |= BIT5;

//sample speed, sample and hold = 16, on

ADC14->CTL0 = ADC14\_CTL0\_SHT0\_2 | ADC14\_CTL0\_SHP | ADC14\_CTL0\_ON;

//sample res = 14 bit

ADC14->CTL1 = ADC14\_CTL1\_RES\_3;

ADC14->MCTL[0] |= ADC14\_MCTLN\_INCH\_0;

// int enable

ADC14->IER0 |= ADC14\_IER0\_IE0;

NVIC\_EnableIRQ(ADC14\_IRQn);

//wake on isr exit

SCB->SCR &= ~SCB\_SCR\_SLEEPONEXIT\_Msk;

}

/\* request a sample \*/

void ADC\_RequestNextSample() {

//start sample

//if (adcflag != F\_ADC\_REQUEST) {

ADC14->CTL0 |= ADC14\_CTL0\_ENC | ADC14\_CTL0\_SC;

//}

adcflag = F\_ADC\_REQUEST;

}

/\* process a sample \*/

void ADC14\_IRQHandler() {

avg -= lastRead[pos];

rms -= (lastRead[pos]/10) \* (lastRead[pos]/10);

lastRead[pos] = ADC14->MEM[0]; //output

avg += lastRead[pos];

rms += (lastRead[pos]/10) \* (lastRead[pos]/10);

int prev = pos - 1;

if (max[1] < lastRead[pos]) max[1] = lastRead[pos];

if (min[1] > lastRead[pos]) min[1] = lastRead[pos];

if (pos == 0) {

prev = 3999;

max[0] = max[1];

min[0] = min[1];

max[1] = lastRead[pos];

min[1] = lastRead[pos];

}

//in up mode

if (upCount > 0) {

if (lastRead[pos] + THRESH > lastRead[prev]) {

upCount++;

} else {

freqAvg -= freq[freqPos];

freq[freqPos] = upCount;

freqAvg += freq[freqPos];

upCount = 0;

dCount = 1;

freqPos = ( freqPos + 1 ) % NUM\_FREQ;

}

} else { //down mode

if (lastRead[pos] < lastRead[prev] + THRESH) {

dCount++;

} else {

freqAvg -= freq[freqPos];

freq[freqPos] = dCount;

freqAvg += freq[freqPos];

dCount = 0;

upCount = 1;

freqPos = ( freqPos + 1 ) % NUM\_FREQ;

}

}

pos = ( pos + 1 ) % SAMPLES;

adcflag =F\_ADC\_READ\_ME;

}

/\*\* check if a sample is ready \*/

int ADC\_CheckReady() {

return adcflag == F\_ADC\_READ\_ME;

}

/\* get a raw dc value \*/

unsigned int ADC\_GetRawValue() {

adcflag = F\_ADC\_NO\_OP;

return avg / SAMPLES;

}

/\* get a raw ac value \*/

unsigned int ADC\_GetRawValueAC() {

adcflag = F\_ADC\_NO\_OP;

return sqrt(rms / SAMPLES);

}

//not used

unsigned int ADC\_GetRawValueOhm() {

adcflag = F\_ADC\_NO\_OP;

int v = lastRead[pos - 1 > 0 ? pos -1 : 3999] \* CAL;

return ((3300 + v) \* RESISTOR) / v;

}

/\* get a formatted dc value \*/

void ADC\_GetFormatedDC(char\* value) {

adcflag = F\_ADC\_NO\_OP;

unsigned long long conversion = avg / SAMPLES \* CAL /10;

int loc = 5;

value[2] = '.';

while (loc >= 0) {

value[loc] = '0' + (conversion % 10);

conversion /= 10;

if (loc == 3) loc--;

loc--;

}

}

/\* get a formatted ac value \*/

void ADC\_GetFormatedAC(char\* value) {

adcflag = F\_ADC\_NO\_OP;

unsigned long long conversion = (rms / SAMPLES \* CAL);

conversion = sqrt(conversion) \* 10;

int loc = 5;

value[2] = '.';

while (loc >= 0) {

value[loc] = '0' + (conversion % 10);

conversion /= 10;

if (loc == 3) loc--;

loc--;

}

}

/\* get a formatted ac value minus offset \*/

void ADC\_GetFormatedAC\_Calc(char\* value) {

adcflag = F\_ADC\_NO\_OP;

unsigned long long conversion = (rms / SAMPLES \* CAL);

conversion = sqrt(conversion) \* 10 - (avg / SAMPLES \* CAL);

int loc = 5;

value[2] = '.';

while (loc >= 0) {

value[loc] = '0' + (conversion % 10);

conversion /= 10;

if (loc == 3) loc--;

loc--;

}

}

/\* get a formatted V peak to peak value \*/

void ADC\_GetFormatedVpp(char\* value) {

adcflag = F\_ADC\_NO\_OP;

unsigned long long conversion = (max[0] - min[0]) \* CAL /10;

int loc = 5;

value[2] = '.';

while (loc >= 0) {

value[loc] = '0' + (conversion % 10);

conversion /= 10;

if (loc == 3) loc--;

loc--;

}

}

/\* get a formatted freq value \*/

void ADC\_GetFormatedFreq(char\* value) {

value[0] = ' ';

value[1] = ' ';

value[2] = ' ';

adcflag = F\_ADC\_NO\_OP;

unsigned long long conversion = 40000/ freqAvg;

if (conversion < 250) conversion++;

int loc = 3;

while (loc >= 0 && conversion > 0) {

value[loc] = '0' + (conversion % 10);

conversion /= 10;

loc--;

}

}int maxCount;

// not used

void ADC\_GetFormatedOhm(char\* value) {

value[0] = ' ';

value[1] = ' ';

value[2] = ' ';

adcflag = F\_ADC\_NO\_OP;

long long v = lastRead[pos == 0 ? 3999 : pos - 1] \* CAL;

unsigned long long conversion = ((3300 + v) \* RESISTOR) / (v + 1) \* 25 / 208000;

int loc = 5;

while (loc >= 0 && conversion > 0) {

value[loc] = '0' + (conversion % 10);

conversion /= 10;

loc--;

}

}

### timer.h

/\*

\* timer.h

\*

\* Created on: May 15, 2017

\* Author: kmrosent

\*/

#ifndef TIMER\_H\_

#define TIMER\_H\_

void Timer\_Setup();

void TA0\_0\_IRQHandler(void);

#endif /\* TIMER\_H\_ \*/

### timer.c

/\*

\* timer.c

\*

\* Created on: May 15, 2017

\* Author: kmrosent

\*/

#include "timer.h"

#include "adc.h"

#include "msp.h"

/\*\* setup and enables timer \*/

void Timer\_Setup() {

TIMER\_A0->CCTL[0] = TIMER\_A\_CCTLN\_CCIE; // TACCR0 interrupt enabled

TIMER\_A0->CCR[0] = 750;

TIMER\_A0->CTL = TIMER\_A\_CTL\_SSEL\_\_SMCLK | // SMCLK, continuous mode

TIMER\_A\_CTL\_MC\_\_UP |

TIMER\_A\_CTL\_ID\_\_1;

NVIC\_EnableIRQ(TA0\_0\_IRQn);

}

/\*\* requests another sample at 4000 samples/sec rate \*/

void TA0\_0\_IRQHandler(void) {

TIMER\_A0->CCTL[0] &= ~TIMER\_A\_CCTLN\_CCIFG;

ADC\_RequestNextSample();

}

### multi.h

/\*

\* multi.h

\*

\* Created on: May 18, 2017

\* Author: Kyle

\*/

#ifndef MULTI\_H\_

#define MULTI\_H\_

#define BAR\_RAW\_VAL 305

char dc[8];

char rmst[8];

char calc[8];

char vpp[8];

char freqt[8];

char dc\_s[8];

char rms\_s[8];

char calc\_s[8];

char vpp\_s[8];

char freq\_s[8];

void Refresh\_Display();

void Draw\_Background();

void Store();

void Setup\_Multi();

#endif /\* MULTI\_H\_ \*/

### multi.c

/\*

\* multi.c

\*

\* Created on: May 18, 2017

\* Author: Kyle

\*/

#include "multi.h"

#include "uart.h"

#include "adc.h"

/\* zeros the multimeter store \*/

void Setup\_Multi() {

int i;

for (i = 0; i<8; i++) {

dc\_s[i] = ' ';

rms\_s[i] = ' ';

calc\_s[i] = ' ';

vpp\_s[i] = ' ';

freq\_s[i] = ' ';

}

}

/\* draws the background \*/

void Draw\_Background() {

UART\_String("\033[2J", 4, 0);

UART\_String("\033[H", 3, 0);

UART\_String("------------------------------------------------------------", 60, 1);

UART\_String("| Digital Multimeter |", 60, 1);

UART\_String("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|", 60, 1);

UART\_String("| DC-V || TrueRms || CalcRms || V-pp || Freq |", 60, 1);

UART\_String("|----------------------------------------------------------|", 60, 1);

UART\_String("| || || || || |", 60, 1);

UART\_String("|----------------------------------------------------------|", 60, 1);

UART\_String("| DC Meter |", 60, 1);

UART\_String("| 0.0V | 0.5V | 1.0V | 1.5V | 2.0V | 2.5V | 3V |", 60, 1);

UART\_String("| |", 60, 1);

UART\_String("|----------------------------------------------------------|", 60, 1);

UART\_String("| Calc-RMS Meter |", 60, 1);

UART\_String("| 0.0V | 0.5V | 1.0V | 1.5V | 2.0V | 2.5V | 3V |", 60, 1);

UART\_String("| |", 60, 1);

UART\_String("|----------------------------------------------------------|", 60, 1);

UART\_String("| Stored Vals: |", 60, 1);

UART\_String("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|", 60, 1);

UART\_String("| DC-V || TrueRms || CalcRms || V-pp || Freq |", 60, 1);

UART\_String("|----------------------------------------------------------|", 60, 1);

UART\_String("| || || || || |", 60, 1);

UART\_String("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|", 60, 1);

}

/\* draws the values \*/

void Refresh\_Display() {

char arr[8];

int i,j,k;

//DC

UART\_String("\033[6;4H", 6, 0);

ADC\_GetFormatedDC(dc);

for (i = 1; i < 6; i++) {

UART0Tx(dc[i]);

}

UART0Tx('V');

//AC - true

UART\_String("\033[6;16H", 7, 0);

ADC\_GetFormatedAC(rmst);

for (i = 1; i < 6; i++) {

UART0Tx(rmst[i]);

}

UART0Tx('V');

//AC - calc

UART\_String("\033[6;28H", 7, 0);

ADC\_GetFormatedAC\_Calc(calc);

for (i = 1; i < 6; i++) {

UART0Tx(calc[i]);

}

UART0Tx('V');

//V peak to peak

UART\_String("\033[6;40H", 7, 0);

ADC\_GetFormatedVpp(vpp);

for (i = 1; i < 6; i++) {

UART0Tx(vpp[i]);

}

UART0Tx('V');

//freq

UART\_String("\033[6;52H", 7, 0);

ADC\_GetFormatedFreq(freqt);

for (i = 0; i < 4; i++) {

UART0Tx(freqt[i]);

}

UART0Tx('H');

UART0Tx('z');

//DC BAR

UART\_String("\033[10;2H", 7, 0);

i = ADC\_GetRawValue();

j = i / BAR\_RAW\_VAL;

for (k = 0; k<58;k++) {

UART0Tx(k<j ? ']' : ' ');

}

//AC BAR

UART\_String("\033[14;2H", 7, 0);

i = ADC\_GetRawValueAC();

j = i / BAR\_RAW\_VAL;

for (k = 0; k<58;k++) {

UART0Tx(k<j ? ']' : ' ');

}

//DC -store

UART\_String("\033[20;4H", 7, 0);

for (i = 1; i < 6; i++) {

UART0Tx(dc\_s[i]);

}

UART0Tx('V');

//AC - true -store

UART\_String("\033[20;16H", 8, 0);

for (i = 1; i < 6; i++) {

UART0Tx(rms\_s[i]);

}

UART0Tx('V');

//AC - calc -store

UART\_String("\033[20;28H", 8, 0);

for (i = 1; i < 6; i++) {

UART0Tx(calc\_s[i]);

}

UART0Tx('V');

//V peak to peak -store

UART\_String("\033[20;40H", 8, 0);

for (i = 1; i < 6; i++) {

UART0Tx(vpp\_s[i]);

}

UART0Tx('V');

//freq -store

UART\_String("\033[20;52H", 8, 0);

ADC\_GetFormatedFreq(arr);

for (i = 0; i < 4; i++) {

UART0Tx(freq\_s[i]);

}

UART0Tx('H');

UART0Tx('z');

}

/\* stores the values \*/

void Store() {

int i;

for (i = 0; i<8; i++) {

dc\_s[i] = dc[i];

rms\_s[i] = rmst[i];

calc\_s[i] = calc[i];

vpp\_s[i] = vpp[i];

freq\_s[i] = freqt[i];

}

}