Multimeter

Mark Nakamae

Cal Poly SLO

May 2020

Description

The goal of this project was to recreate a digital multimeter using the MSP432. Our ADC14 sampled the input at the specified rates below in the table. This project was fairly straightforward, since the ADC14 transmitted sampled data through the UART to communicate between the MSP432 board and the Analog Discovery 2. This sampled data was then interpreted to give a float value voltage reading from the input. [Here](https://drive.google.com/file/d/1qlS79zb0DiPQH7wu87ciRfQR8LAvtebh/view) is the multimeter working.

Components

MSP432P401R

MCP4912

Digilent Analog Discovery 2

Data Taken from Multimeter

|  |  |  |
| --- | --- | --- |
| ADC14 Sample Time Clocks | DC Voltage (V) | Output Values (V) |
| 16 | 0 | 0.057 |
| 16 | 1 | 1.064 |
| 16 | 2 | 2.061 |
| 16 | 3 | 3.070 |
| 96 | 0 | 0.023 |
| 96 | 1 | 1.021 |
| 96 | 2 | 2.029 |
| 96 | 3 | 3.028 |
| 192 | 0 | 0.006 |
| 192 | 1 | 1.008 |
| 192 | 2 | 2.011 |
| 192 | 3 | 3.012 |

Source Code

#include "msp.h"

#include "uart.h"

#include "adc.h"

/\*

\* main.c

\*/

**void** main(**void**)

{

**volatile** uint32\_t i, j;

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer

UART0\_init();

adc\_init();

// Enable global interrupt

\_\_enable\_irq();

**while** (1)

{

**for** (j = 20; j > 0; j--)

{

**for** (i = 20000; i > 0; i--); // Delay

adc\_record();

}

adc\_report\_avg();

}

}

**void** ADC14\_IRQHandler(**void**)

{

adc\_curr\_val(ADC14->MEM[0]);

}

/\*

\* adc.h

\*

\* Created on: May 18, 2020

\* Author: nicks

\*/

#ifndef ADC\_H\_

#define ADC\_H\_

#define AVG\_LENGTH (4)

#define MAX\_READING\_MV (3000)

**void** adc\_record();

**void** adc\_report\_avg();

**inline** **void** adc\_curr\_val(**unsigned** **int** val);

**void** adc\_log\_reading();

**unsigned** **int** adc\_get\_avg();

**void** uart\_write\_volts(**unsigned** **int** val\_mv);

**void** adc\_init();

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

/\*

\* adc.c

\*

\* Created on: May 18, 2020

\* Author: nicks

\*/

#include "adc.h"

#include "uart.h"

#include "msp.h"

**static** **unsigned** **int** adc\_value = 0;

**static** **unsigned** **int** calibration\_shift = 0;

**static** **unsigned** **int** calibration\_scale = 5;

**static** **unsigned** **int** adc\_history[AVG\_LENGTH] = {0, 0, 0, 0};

**static** **unsigned** **int** max\_value = 0;

**static** **unsigned** **int** min\_value = 16383;

**void** adc\_init()

{

// GPIO Setup

P5->SEL1 |= BIT4; // Configure P5.4 for ADC

P5->SEL0 |= BIT4;

// Sampling time, S&H=16, ADC14 on

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

ADC14->CTL1 = ADC14\_CTL1\_RES\_3; // Use sampling timer, 14-bit conversion

ADC14->MCTL[0] |= ADC14\_MCTLN\_INCH\_1; // A1 ADC input select; Vref=AVCC

ADC14->IER0 |= ADC14\_IER0\_IE0; // Enable ADC conv complete interrupt

ADC14->CTL0 |= ADC14\_CTL0\_ENC; // Enable conversions

// Enable ADC interrupt in NVIC module

NVIC->ISER[0] = 1 << ((ADC14\_IRQn) & 31);

}

**void** adc\_record(){

adc\_log\_reading();

ADC14->CTL0 |= ADC14\_CTL0\_SC; //start taking samples

}

**void** adc\_report\_avg()

{

**unsigned** **int** avg\_val = adc\_get\_avg();

uart\_write\_volts(avg\_val);

}

**inline** **void** adc\_curr\_val(**unsigned** **int** val)

{

adc\_value = val;

}

**void** adc\_log\_reading()

{

**int** i;

**for**(i = 0; i < AVG\_LENGTH - 1; i++)

{

adc\_history[i + 1] = adc\_history[i];// shift old values up one

}

adc\_history[0] = adc\_value; // store the most recent value

}

**unsigned** **int** adc\_get\_avg()

{

**int** i;

**unsigned** **int** sum = 0;

**for**(i = 0; i < AVG\_LENGTH; i++)

{

sum += adc\_history[i];

}

**return** sum / AVG\_LENGTH;

}

**void** uart\_write\_volts(**unsigned** **int** val\_mv)

{

//got the scaling factor from max val 16383/3.3 which is the max voltage

uint32\_t meh = val\_mv; //in order to get the eight resolution some more bits were needed

uart\_write\_int(val\_mv / 4964);//output the volts

uart\_transmitChar('.');

uart\_write\_int((val\_mv / 496) % 10 );//output decivolts

uart\_write\_int((meh \* 100 /4964) % 10 );//output centivolts

uart\_write\_int((meh \* 1000 /4964) % 10); //output milivolts

uart\_transmitChar('V');

uart\_transmit\_nl();

}

#ifndef UART\_H\_

#define UART\_H\_

#include "stdint.h"

#define ASCII\_OFFSET 48

#define MAX\_DAC 1023

#define MAX\_BUF 5

uint8\_t uart\_getChar(**void**);

**void** uart\_transmitChar(uint8\_t);

**void** UART0\_init(**void**);

**void** scale(uint16\_t \*, uint8\_t);

**void** uart\_transmit\_nl();

**void** uart\_write\_int(**unsigned** **int** num);

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

/\*

\* uart.c

\*

\* Created on: May 18, 2020

\* Author: nicks

\*/

// outputs a char to UART

#include "msp.h"

#include "UART.h"

**void** uart\_write\_int(**unsigned** **int** num)

{

**if**(num >= 10)

{

uart\_write\_int(num / 10);

}

uart\_transmitChar((uint8\_t)((num % 10) + ASCII\_OFFSET));

}

uint8\_t uart\_getChar(**void**)

{

**while**(!(EUSCI\_A0->IFG & EUSCI\_A\_IFG\_RXIFG)) { }

**return** EUSCI\_A0->RXBUF;

}

// outputs a char to UART buffer

**void** uart\_transmitChar(uint8\_t c)

{

// check to see if transmit buffer is ready

**while**(!(EUSCI\_A0->IFG & EUSCI\_A\_IFG\_TXIFG)) { }

EUSCI\_A0->TXBUF = c;

}

**void** uart\_transmit\_nl()

{

uart\_transmitChar('\n');

uart\_transmitChar(0xD);

}

**void** UART0\_init(**void**)

{

EUSCI\_A0->CTLW0 |= EUSCI\_B\_CTLW0\_SWRST; /\* put in reset mode for config \*/

EUSCI\_A0->MCTLW &= ~EUSCI\_A\_MCTLW\_OS16; /\* disable oversampling \*/

EUSCI\_A0->CTLW0 = EUSCI\_B\_CTLW0\_UCSSEL\_2 | EUSCI\_B\_CTLW0\_SWRST; /\* 1 stop bit, no parity, SMCLK, 8-bit data \*/

EUSCI\_A0->BRW = 26; /\* 3,000,000 / 115200 = 26 \*/

P1->SEL0 |= (BIT2|BIT3); /\* P1.3, P1.2 for UART \*/

P1->SEL1 &= ~(BIT2|BIT3);

EUSCI\_A0->CTLW0 &= ~EUSCI\_B\_CTLW0\_SWRST; /\* take UART out of reset mode \*/

}

**void** scale(uint16\_t \*data, uint8\_t tmp)

{

(\*data) \*= 10;

(\*data) += tmp;

}