Andrea Smith
EE 2361
Lab 6 Report
19 April 2020

Overview:

Analog-to-Digital Converters (ADCs) translate electrical signals into digital values for processing in data acquisition systems. In this lab, an ADC is used to display the variable Vdd through a potentiometer in the LCD circuit from Lab 5.

Buffer Functions:

1. putVal(int newValue);

```
void putVal(int newValue) {
    buffer[buffIdx++] = newValue;
    if (buffIdx >= BUFFER_SIZE) { // Reset if buffer size exceeded
        buffIdx = 0;
    }
}
```

A circular buffer is an array of fixed size (16 in this case) with an index that points to the most recently modified value. The function putVal() takes a single integer argument **newValue** and adds it to a circular buffer.

2. getAvg();

```
int getAvg() {
    double sum = 0;
    int i;
    for (i = 0; i < BUFFER_SIZE; i++) {
        sum = sum + buffer[i]; // Add up all the values in the buffer
    }

    return sum/BUFFER_SIZE; // Return avg
}
```

This simple function sums all the values inside the buffer and returns their average.

3. void initBuffer() ;

```
void initBuffer() {
   long int i = 0;
   for (i = 0; i <= BUFFER_SIZE; i++) {
      buffer[i] = 0;
   }
}
```

As its name implies, the function initBuffer() initializes the buffer by setting every value in the circular buffer to zero.

ADC Functions:

1. *setup()*

```
void setup(void) {
   CLKDIVbits.RCDIV = 0;
   AD1PCFG = 0x9FFE; // AN1 set as analog
   TRISA |= 1;

   I2C2CONbits.I2CEN = 0; // Good practice to disable before changing BRG
   I2C2BRG = 0x9D; // Value for 16 MHz
   I2C2CONbits.I2CEN = 1;
   IFS3bits.MI2C2IF = 0; // Clear interrupt, just in case

   initBuffer();
   lcd_init();
   lcd_setCursor(0,0);

   AD1CON1 = 0; // A/D control reg 1; Off for now
   AD1CON1bits.FORM = 0b00;
   AD1CON1bits.SSRC = 0b010; // Timer 3 compare match (clk divider for fixed sample rate)
   AD1CON1bits.ASAM = 1; // Sample begins immediately after last conversion completes; SAMP
auto reset

   AD1CON2 = 0; // A/D control reg 2
   AD1CON2bits.CSCNA = 0; // scan off
   AD1CON2bits.SMPI = 0b0; // Interrupts at every conversion
   AD1CON2bits.BUFM = 0; // Config as 2 eight-word buffers

   AD1CON3 = 0; // A/D control reg 3
   AD1CON3bits.ADRC = 0;
   AD1CON3bits.SAMC = 0b1;
   AD1CON3bits.ADCS = 0b1; // 2 * Tcy
```

```
AD1CON1bits.ADON = 1; // Turn on module

    AD1CHS = 0; // A/D input channel select register
    AD1CHSbits.CH0NB = 0; // Neg input is VR-
    AD1CHSbits.CH0SB = 0; // Pos input is AN0
    AD1CHSbits.CH0NA = 0;
    AD1CHSbits.CH0SA = 0; // Same as CH0SB

    // Timer 3 setup
    T3CON = 0;
    TMR3 = 0;
    T3CONbits.TON = 0;
    T3CONbits.TCKPS = 0b10; // 64 PRE
    PR3 = 1562; // Value for 0.0625s or every 1/16 of a second
    T3CONbits.TON = 1; // Turn on

    // Timer 2 setup
    T2CON = 0;
    TMR2 = 0;
    T2CONbits.TON = 0;
    T2CONbits.TCKPS = 0b10; // 64 PRE
    PR2 = 24999; // Value for ~0.1s or every 100ms
    T2CONbits.TON = 1;

    // Enable AD1 interrupt
    IEC0bits.AD1IE = 1; // Interrupt after ADC complete
    IFS0bits.AD1IF = 0;

    // Enable T2 interrupt
    IEC0bits.T2IE = 1; // Timer2 interrupt on ADC complete
    IFS0bits.T2IF = 0;
}
```

This function sets up the A/D Converter module, Timer 2, and Timer 3. Timer 3 is set up so the analog input is sampled 16 times per second. The PR3 value is calculated such that the sample occurs every 0.0625 seconds:

$$PR3 = \frac{delay}{Tcy*PRE} - 1 = \frac{0.0625}{(62.5*10^{-9})*64} - 1 = 1562\ .$$

Timer 2 is set up so the most recent digital sample's average value is converted to a floating point voltage every 100ms. The PR2 value was calculated as follows:

$$PR2 = \frac{delay}{Tcy*PRE} - 1 = \frac{0.1}{(62.5*10^{-9})*64} - 1 = 25000\ .$$

2. ADC1 Interrupt

```
void __attribute__((__interrupt__, __auto_psv__)) _ADC1Interrupt(void) {
   IFS0bits.AD1IF = 0;
   unsigned long int value = ADC1BUF0;
   putVal(value);
}
```

This ISR's purpose is to add values to the circular buffer. It is triggered by Timer 3 and therefore samples 16 analog values per second and adds them to the buffer.

3. Timer 2 Interrupt

```
void __attribute__((__interrupt__, __auto_psv__)) _T2Interrupt(void) {
   unsigned long avg;
   _T2IF = 0;
   TMR2 = 0;
   char adStr[20];
   lcd_setCursor(0,0);
   avg = getAvg();
   sprintf(adStr, "%6.4f V", (3.3/1024)*avg);
   lcd_printStr(adStr);
}
```

The Timer 2 interrupt performs the actual conversion of the averaged floating-point value to a string to be displayed on the LCD.

Questions:

1. State the maximum sampling rate that a PIC24 chip with Fcy=16MHz can handle, assuming we do not need to output anything on the LCD? In other words, for the maximum value that K can take, what is 1/K?

T_{SMP} = SAMC<4:0> * T_{AD}

Find SAMC:
fcy = 16 MHz
Tcy = 62.5 nsec

T_{AD} >= 2 * T_{CY} = 125 nsec
ADCS = 1.

SAMC * T_{AD} >= 2.5 us
SAMC = $\frac{2.5*10^{-6}}{125*10^{-9}} = 20$
T_{SMP} = 20 * T_{AD} = **2500 samp/sec**

2. State the maximum display refresh rate assuming we only want to convert 10-bit digital values to the string format "x.xxxx V" and show the 8-character strings on the LCD? Ignore A/D times here, i.e., assume the data is already in the buffer. In other words, how far can we push the 100ms refresh rate?

$T_{CNV} = 12 * T_{AD} =$ **1500**

3. Assuming the A/D sampling rate described in Item 3, and the fastest LCD update calculated in Item 4, how many samples do we have to drop and not show on the LCD?

1000

I am sorry if this is very wrong I really didn't know how to do it