# EE2361 Summer 2015, HW3 Due 8/6/2015 before midnight.

*NOTE: Please submit your homework electronically in a file (either pdf, word, open office) through moodle. I expect all code to have been run and tested in the MPLABX IDE environment before submission.*

1. Configure the UART unit for a baud rate of 115200 bps, and send five characters.
   1. Show your baud rate calculations, including the baud rate error (i.e., the percentage difference of the actual baud rate compared to the target baud rate of 115200).

BRGH = 0 therefore

UxBRG = [FCY / (16\*RATE )]-1 = [16E6 / (16\*115200)]-1 = 8.681 – 1= 7.681

so UxBRG = 8 with rounding.

ERROR = (Calculated\_Rate – Desired\_Rate)/Desired\_Rate = (7.681 – 8)/8 = -0.039875

Therefore the error is -3.9875%

* 1. Write a program to configure the UART unit with the specified baud rate, and then send five characters. Use polling.

|  |
| --- |
| #include <p24Fxxxx.h>  #include <PPS.h>  #include <xc.h>  #pragma config POSCMOD = NONE // Primary Oscillator Select (Primary oscillator disabled)  #pragma config I2C1SEL = PRI // I2C1 Pin Location Select (Use default SCL1/SDA1 pins)  #pragma config IOL1WAY = OFF // IOLOCK Protection (IOLOCK may be changed via unlocking seq)  #pragma config OSCIOFNC = ON // Primary Oscillator Output Function (OSC2/CLKO/RC15 functions as port I/O (RC15))  #pragma config FCKSM = CSECME // Clock Switching and Monitor (Clock switching is enabled, Fail-Safe Clock Monitor is enabled)  #pragma config FNOSC = FRCPLL // Oscillator Select (Fast RC Oscillator with PLL module (FRCPLL))  #pragma config SOSCSEL = SOSC // Sec Oscillator Select (Default Secondary Oscillator (SOSC))  #pragma config WUTSEL = LEG // Wake-up timer Select (Legacy Wake-up Timer)  #pragma config IESO = ON // Internal External Switch Over Mode (IESO mode (Two-Speed Start-up) enabled)  // CONFIG1  #pragma config WDTPS = PS32768 // Watchdog Timer Postscaler (1:32,768)  #pragma config FWPSA = PR128 // WDT Prescaler (Prescaler ratio of 1:128)  #pragma config WINDIS = ON // Watchdog Timer Window (Standard Watchdog Timer enabled,(Windowed-mode is disabled))  #pragma config FWDTEN = OFF // Watchdog Timer Enable (Watchdog Timer is disabled)  #pragma config ICS = PGx1 // Comm Channel Select (Emulator EMUC1/EMUD1 pins are shared with PGC1/PGD1)  #pragma config GWRP = OFF // General Code Segment Write Protect (Writes to program memory are allowed)  #pragma config GCP = OFF // General Code Segment Code Protect (Code protection is disabled)  #pragma config JTAGEN = OFF // JTAG Port Enable (JTAG port is disabled)  unsigned char buffer[64];  unsigned char front = 0;  unsigned char back = 0;  void PPutch(const unsigned char c)  {  while(!IFS0bits.U1TXIF);  IFS0bits.U1TXIF = 0;  U1TXREG = c;  }  void setup(void)  {  CLKDIVbits.RCDIV = 0;  AD1PCFG = 0x9fff; // For digital I/O. If you want to use analog, you'll  // need to change this.  // set up UART  U1MODE = 0;  U1BRG = 8; // 115200 baud,  U1MODEbits.BRGH = 0;  U1MODEbits.UEN = 0;  U1MODEbits.UARTEN = 1;  U1STAbits.UTXEN = 1;  PPSUnLock;  // serial port UCA1: Uses RP4 and RP5  PPSOutput(PPS\_RP3, PPS\_U1TX);  PPSInput(PPS\_U1RX, PPS\_RP6);  PPSLock;  }  int main(void)  {  setup();    PPutch('a');  PPutch('b');  PPutch('c');  PPutch('d');  PPutch('e');  while(1); //loop back    return 0; // never reached (we hope)  } |

* 1. How many instruction cycles (Tcy) is each polling while loop in your program going to take? Explain (hint: make sure you know what the size of the output UART buffer is)

The maximum number of cycles it should take for the polling to occur corresponds for the time it takes the UART to handle a data frame (10 bits total). Since we are running at 115200 baud, it takes 10/115200 seconds, which is (10/115200)\*16000000 cycles, which is 1389 cycles.

1. SPI1CON is set to 0x052D. How many Tcy cycles does it take to transmit 3 words?

Fsck = Fcy/[Pri\_Prescale \* Sec\_Prescale] = 16MHz/[6\*16] = *166.6 KHz*

Three words are transmitted, each with 16 bits, meaning 48 bits total. SPI requires no overhead bits, so the total transmission will be 48 bits (or 48 cycles). 48 / 166.6KHz = 288 uS

So, It will take 288uS to transmit 3 words.

SPI1CON = 0x052D = 0000 0101 0010 1101

PPRE : 01 Pri. Prescale 16:1

SPRE : 011 Sec. Prescale 6:1

MSTEN: 1 Master Mode

CKP: 0 Clock is idle low (active high)

SSEN: 0 Port function control SSx' pin

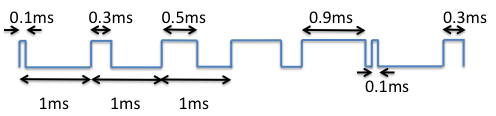
CKE: 1 Serial output changes from active clock to idle transition

SMP: 0 Data sample at middle of data output time

MODE16: 1 Communication is in word mode

DISSDO: 0 SDOx pin controlled by SPI module

DISSCK: 0 Internal SPIx clock enabled

1. Use the output compare module and polling to generate a signal that has the following properties: (1) its rising edges are 1ms apart. (2) the signal goes through a cycle in which the high pulse width increases from 0.1ms, to 0.9ms with increments of 0.2ms. The figure below shows how the signal should look like. Submit your code, along with the graph of the signal generated by the simulator (logic analyzer)  
     
   

|  |
| --- |
| #include <p24Fxxxx.h>  #include <xc.h>  #include <PPS.h>  // PIC24FJ64GA002 Configuration Bit Settings  // 'C' source line config statements  #include <xc.h>  // CONFIG2  #pragma config POSCMOD = NONE // Primary Oscillator Select (Primary oscillator disabled)  #pragma config I2C1SEL = PRI // I2C1 Pin Location Select (Use default SCL1/SDA1 pins)  #pragma config IOL1WAY = OFF // IOLOCK Protection (IOLOCK may be changed via unlocking seq)  #pragma config OSCIOFNC = OFF // Primary Oscillator Output Function (OSC2/CLKO/RC15 functions as CLKO (FOSC/2))  #pragma config FCKSM = CSECME // Clock Switching and Monitor (Clock switching is enabled, Fail-Safe Clock Monitor is enabled)  #pragma config FNOSC = FRCPLL // Oscillator Select (Fast RC Oscillator with PLL module (FRCPLL))  #pragma config SOSCSEL = SOSC // Sec Oscillator Select (Default Secondary Oscillator (SOSC))  #pragma config WUTSEL = LEG // Wake-up timer Select (Legacy Wake-up Timer)  #pragma config IESO = ON // Internal External Switch Over Mode (IESO mode (Two-Speed Start-up) enabled)  // CONFIG1  #pragma config WDTPS = PS32768 // Watchdog Timer Postscaler (1:32,768)  #pragma config FWPSA = PR128 // WDT Prescaler (Prescaler ratio of 1:128)  #pragma config WINDIS = ON // Watchdog Timer Window (Standard Watchdog Timer enabled,(Windowed-mode is disabled))  #pragma config FWDTEN = OFF // Watchdog Timer Enable (Watchdog Timer is disabled)  #pragma config ICS = PGx1 // Comm Channel Select (Emulator EMUC1/EMUD1 pins are shared with PGC1/PGD1)  #pragma config GWRP = OFF // General Code Segment Write Protect (Writes to program memory are allowed)  #pragma config GCP = OFF // General Code Segment Code Protect (Code protection is disabled)  #pragma config JTAGEN = OFF // JTAG Port Enable (JTAG port is disabled)  void setup()  {  PPSUnLock;  PPSOutput(PPS\_RP3,PPS\_OC1);  PPSLock;  T2CON = 0; //Set timer2  PR2 = 15999; // 1ms  TMR2 = 0;  IFS0bits.T2IF = 0;  OC1CON = 0;  IFS0bits.OC1IF = 0;  OC1R = 1599; //initial compare value, proportional to 0.1 ms  OC1RS = 4799; //second compare value, proportional to 0.3 ms  OC1CONbits.OCTSEL = 0; //use TMR2  OC1CONbits.OCM = 6; // PWM mode  T2CONbits.TON = 1;  }  int main(void)  {  setup();  while (1){  while (!IFS0bits.T2IF); //wait until timer interrupt  IFS0bits.T2IF = 0;  OC1RS += 3200; //increment compare value proportional to 0.2 ms  if(OC1RS > 14399) //reset compare value back to 0.1 ms after 0.9 ms  OC1RS = 1599;  Nop(); //breakpoint  }  } |
|  |

1. Modify the above program to use interrupts instead.

|  |
| --- |
| #include <p24Fxxxx.h>  #include <xc.h>  #include <PPS.h>  // PIC24FJ64GA002 Configuration Bit Settings  // 'C' source line config statements  #include <xc.h>  // CONFIG2  #pragma config POSCMOD = NONE // Primary Oscillator Select (Primary oscillator disabled)  #pragma config I2C1SEL = PRI // I2C1 Pin Location Select (Use default SCL1/SDA1 pins)  #pragma config IOL1WAY = OFF // IOLOCK Protection (IOLOCK may be changed via unlocking seq)  #pragma config OSCIOFNC = OFF // Primary Oscillator Output Function (OSC2/CLKO/RC15 functions as CLKO (FOSC/2))  #pragma config FCKSM = CSECME // Clock Switching and Monitor (Clock switching is enabled, Fail-Safe Clock Monitor is enabled)  #pragma config FNOSC = FRCPLL // Oscillator Select (Fast RC Oscillator with PLL module (FRCPLL))  #pragma config SOSCSEL = SOSC // Sec Oscillator Select (Default Secondary Oscillator (SOSC))  #pragma config WUTSEL = LEG // Wake-up timer Select (Legacy Wake-up Timer)  #pragma config IESO = ON // Internal External Switch Over Mode (IESO mode (Two-Speed Start-up) enabled)  // CONFIG1  #pragma config WDTPS = PS32768 // Watchdog Timer Postscaler (1:32,768)  #pragma config FWPSA = PR128 // WDT Prescaler (Prescaler ratio of 1:128)  #pragma config WINDIS = ON // Watchdog Timer Window (Standard Watchdog Timer enabled,(Windowed-mode is disabled))  #pragma config FWDTEN = OFF // Watchdog Timer Enable (Watchdog Timer is disabled)  #pragma config ICS = PGx1 // Comm Channel Select (Emulator EMUC1/EMUD1 pins are shared with PGC1/PGD1)  #pragma config GWRP = OFF // General Code Segment Write Protect (Writes to program memory are allowed)  #pragma config GCP = OFF // General Code Segment Code Protect (Code protection is disabled)  #pragma config JTAGEN = OFF // JTAG Port Enable (JTAG port is disabled)  void \_\_attribute\_\_((\_\_interrupt\_\_,\_\_auto\_psv\_\_)) \_T2Interrupt(void)  {  IFS0bits.T2IF = 0;  OC1RS += 3200; //increment compare value proportional to 0.2 ms  if(OC1RS > 14399) //reset compare value back to 0.1 ms after 0.9 ms  OC1RS = 1599;  Nop(); //breakpoint  }  void setup()  {  PPSUnLock;  PPSOutput(PPS\_RP3,PPS\_OC1);  PPSLock;  T2CON = 0; //Set timer2  PR2 = 15999; // 1ms  TMR2 = 0;  IFS0bits.T2IF = 0;  IEC0bits.T2IE = 1;  OC1CON = 0;  IFS0bits.OC1IF = 0;    OC1R = 1599; //initial compare value, proportional to 0.1 ms  OC1RS = 4799; //second compare value, proportional to 0.3 ms  OC1CONbits.OCTSEL = 0; //use TMR2  OC1CONbits.OCM = 6; // PWM mode  T2CONbits.TON = 1;  }  int main(void)  {  setup();  while (1);    return 0; //HAHHHAHAHA NEVER  } |
|  |

1. What is the effective resolution, in volts, of an A/D conversion using PIC24 in which the signal is not conditioned (i.e., its range is not mapped to 0-3.3V), and the input signal range is from 0.2v – 0.45v>

The resolution of the PIC24 ADC converter is 10 bit on 0 - 3.3V which to corresponds 3.3/(2^10-1) = 3.226 mV granularity in its measurement. So the effective resolution is 3.226mV.

The range of measurement from 0.2 to 0.45 volts is 0.45-0.2 = 0.25 volts. This means that the values in this range can be resolved down into 0.25 / 0.003226 (range/granularity) = 77.50 steps, 77 rounding down.

1. Write a program to configure the A/D unit to perform these tasks:
   * Sample and convert the analog input on the AN0 pin at the fastest rate possible subject to the following constraints.
   * Sampling time should be 3us or higher.
   * AD1IF should be raised after four samples are converted.
   * Use interrupts to fill a circular buffer of size 64 with the data.
   * Main keeps a running average of all 64 data samples.

Submit your code, and answer the following question too:

* + - * 1. What is the sampling rate? i.e., how many sample and conversions are done per second?

ADCS = 1, so TAD = Tcy\*(ADCS+1) = [1/16000000] \* 2 = 125ns > 75ns minimum

Tsample = SAMC \* TAD, for a 3us min we set Tsample = 3us

so 3us = 0.125 \* SAMC ==> SAMC = 24 (integer). Sample acquisition time should therefore only take 3us since the SAMC worked out to be a integer (no rounding).

Conversion time is between 11 and 12 TAD, so assume worst case of 12:

Tconversion = 12 \* 125ns = 1.5us

Total sample/conversion sequence time : Ttotal = Tsample + Tconversion

Ttotal = 1.5us + 3us = 4.5us

Frequency = 1/Ttotal = 1 /4.5us = 222.2 Khz

* + - * 1. Explain how you accessed the AD buffers ADC1BUFx.

In the ISR for the ADC1 interrupt (which is executed after four samples), the values of ADC1BUF0:3 are written successively to a 64 element buffer array starting at the index stored at a global variable (index) up to that index +3 . The index is then incremented by 4 so that the next samples are stored in the next 4 elements after the samples that were just buffered. If the index is out of range it is reset to 0. These values then can be later accessed by the main function by reading the values stored in the 64 element array.

|  |
| --- |
| #include <p24Fxxxx.h>  #include <xc.h>  // PIC24FJ64GA002 Configuration Bit Settings  // 'C' source line config statements  #include <xc.h>  // CONFIG2  #pragma config POSCMOD = NONE // Primary Oscillator Select (Primary oscillator disabled)  #pragma config I2C1SEL = PRI // I2C1 Pin Location Select (Use default SCL1/SDA1 pins)  #pragma config IOL1WAY = OFF // IOLOCK Protection (IOLOCK may be changed via unlocking seq)  #pragma config OSCIOFNC = OFF // Primary Oscillator Output Function (OSC2/CLKO/RC15 functions as CLKO (FOSC/2))  #pragma config FCKSM = CSECME // Clock Switching and Monitor (Clock switching is enabled, Fail-Safe Clock Monitor is enabled)  #pragma config FNOSC = FRCPLL // Oscillator Select (Fast RC Oscillator with PLL module (FRCPLL))  #pragma config SOSCSEL = SOSC // Sec Oscillator Select (Default Secondary Oscillator (SOSC))  #pragma config WUTSEL = LEG // Wake-up timer Select (Legacy Wake-up Timer)  #pragma config IESO = ON // Internal External Switch Over Mode (IESO mode (Two-Speed Start-up) enabled)  // CONFIG1  #pragma config WDTPS = PS32768 // Watchdog Timer Postscaler (1:32,768)  #pragma config FWPSA = PR128 // WDT Prescaler (Prescaler ratio of 1:128)  #pragma config WINDIS = ON // Watchdog Timer Window (Standard Watchdog Timer enabled,(Windowed-mode is disabled))  #pragma config FWDTEN = OFF // Watchdog Timer Enable (Watchdog Timer is disabled)  #pragma config ICS = PGx1 // Comm Channel Select (Emulator EMUC1/EMUD1 pins are shared with PGC1/PGD1)  #pragma config GWRP = OFF // General Code Segment Write Protect (Writes to program memory are allowed)  #pragma config GCP = OFF // General Code Segment Code Protect (Code protection is disabled)  #pragma config JTAGEN = OFF // JTAG Port Enable (JTAG port is disabled)  volatile int buff[64] = {0};  volatile int index = 0;  //ISR  void \_\_attribute\_\_((\_\_interrupt\_\_)) \_ADC1Interrupt(void) {  \_AD1IF = 0;  buff[index] = ADC1BUF0;  buff[index+1] = ADC1BUF1;  buff[index+2] = ADC1BUF2;  buff[index+2] = ADC1BUF3;    index +=4;  if(index == 64)  index = 0;  }  void setup(void) {  CLKDIVbits.RCDIV = 0;  AD1PCFG = 0x9ffe; // Set A0 analog input, burn!  TRISAbits.TRISA0 = 1;  // Setup on A/D  AD1CON1 = 0;  AD1CON2 = 0;  AD1CON3 = 0;  AD1CHS = 0;  AD1CSSL = 0;  AD1CON1bits.ASAM = 1; // auto-sampling?  AD1CON1bits.SSRC = 0b111; // auto-conversion  AD1CON2bits.SMPI = 3; //interrupt after 4 samples converted    AD1CON3bits.SAMC = 24; // corresponds to 3us sample period  AD1CON3bits.ADCS = 1; // Fastest time for Tad (125ns)    // Configure A/D interrupt  \_AD1IF = 0; // clear flag  \_AD1IE = 1; // enable interrupt  AD1CON1bits.ADON = 1;  }  int main(void) {  setup();  int i, avg;  // AD1CON1bits.SAMP = 1; // start sampling  while (1) {  for(i = 0; i < 64 ; i++) //sums the samples  avg += buff[i];  avg /= 64; // avg ends up being the running average  }  } |