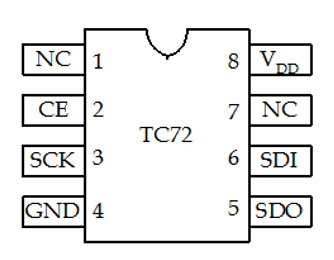
Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Email:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. (40 points) Your manager has asked you to use the TC72 temperature sensor to measure the temperature.

* The on-chip A/D converter is scaled from -128?C to +127C, but can only read temperature from -55C to +125C.
* TC72 has an SPI interface. The pin diagram is given below. Note that when CE = 1, the chip will be enabled.



* The chip can be used in continuous conversion mode and one-shot conversion mode.
* The continuous conversion mode measures temperature approximately every 150ms.
* The one-shot conversion mode measures the temperature once and enters into power-saving mode.
* TC72 uses 10 bits to represent temperature. The 10 bits data are stored in two 8-bit registers MSBT and LSBT. The least significant 2 bits of the 10-bit data are stored in the most significant 2 bits of LSBT. For example, if the 10-bit value is 0b1111111111, then MSBT = 0b**11111111**, LSBT = 0b**11**000000.
* Temperature is represented using 2’s complement format. For example, in the above example, the temperature is -0.25C.
* The registers of TC72 temperature sensor is given below. If the OS bit in the Control register is set to 1, it will use the continuous conversion mode. If the SHDN bit is cleared to 0, it will enter into the power-saving mode. T9~T0 are the 10 bits of the temperature data.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Register** | **Read Address** | **Write Address** | **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| **Control** | 0x00 | 0x80 | 0 | 0 | 0 | OS | 0 | 0 | 0 | SHDN |
| **LSBT** | 0x01 | N/A | T1 | T0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **MSBT** | 0x02 | N/A | T9 | T8 | T7 | T6 | T5 | T4 | T3 | T2 |

Write a program to implement the following functions:

1. (5 points) Connect RP4 of PIC24 to the SDK of TC72, RP5 of PIC24 to the SDI of TC72, RP6 of PIC24 to the SDO of TC72, and RP7 of PIC24 to the CE of TC72;
2. (10 points) Initialize TC72 so that it works in the continuous conversion mode.
3. (10 points) Read the TC72’s results every 150ms.
4. (5 points) Configure the PIC24 SPI to work in the master mode, byte mode, and SDK = 8MHz (Tcy = 16MHz).
5. (10 points) Convert the 10-bit value to a Celsius degree. Using variable **isNegative** to store the sign of the Celsius degree, using 8-bit variable **tempInteger** to store the integer part, and 8-bit variable **tempFraction** to store the fraction part. Note that both tempInteger and tempFraction has a data type char. For example, if the 10-bit value you read from the sensor is 0x03FF, the corresponding Celsius degree is -0.25C. We have isNegative = 1, tempInteger = 0b00000000, tempFraction = 0b01000000. (hint: we only needs two bits to represent the fraction, 0b00 for 0.00C, 0b01 for 0.25C, 0b10 for 0.50C, and , 0b11 for 0.75C)

Paste your solution below.

|  |
| --- |
| #include <p24Fxxxx.h>  #include <PPS.h>  #include <xc.h>  #include <stdlib.h>  #ifdef USECRYSTAL  #pragma config POSCMOD = HS // Primary Oscillator Select (Primary oscillator disabled)  #else  #pragma config POSCMOD = NONE  #endif  #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 port I/O (RC15))  #pragma config FCKSM = CSECME // Clock Switching and Monitor (Clock switching is enabled, Fail-Safe Clock Monitor is enabled)  #ifdef USECRYSTAL  #pragma config FNOSC = PRIPLL // Oscillator Select (Fast RC Oscillator with PLL module (FRCPLL))  #else  #pragma config FNOSC = FRCPLL  #endif  #pragma config SOSCSEL = SOSC // Sec Oscillator Select (Default Secondary Oscillator (SOSC))  #pragma config WUTSEL = LEG // Wake-up timer Select (Legacy Wake-upTimer)  #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 unsigned char buffer[32];  volatile unsigned char front = 0;  volatile unsigned char back = 0;  // This code is for interfacing with the MCP4821 DAC unit, which has an SPI interface.  // We need to send 16-bit SPI output, where the four most significant bits are the commands  // determining the gain and whether to shut down the DAC unit, followed by 12 bits of data.  // The SPI clock speed of 16MHz is not too fast (the DAC manual says 20MHz is the max), and if we  // want slower transmission, we can choose a PRE of 1:2 in the SPI clock.  // We setup T2 to give us 22.7us for each analog voltage. The 22.7us is safely longer than the  // 8 \* SPI\_CLOCK\_PERIOD, which is 0.5us with a primary and secondary SPI clock pre set to 1.  #define DACMASK 0x1FFF  #define DACBITS 0x1000  volatile unsigned short int sample = DACBITS;  void setup(void)  {  CLKDIVbits.RCDIV = 0;  AD1PCFG = 0x9fff; // For digital I/O. If you want to use analog, you'll  // need to change this.  TRISB = 0xffff; // make all PORTBbits inputs initially  // add your configuration commands below  PORTB = 0xffff;  PORTBbits.RB7 = 0; //CE will be low (off)    TRISBbits.TRISB7 = 0; //RB7 will be CE, enables it as output  T2CON = 0;  PR2 = 9374; //approx 150ms period  T2CONbits.TCKPS = 3; //prescale 1:256  IFS0bits.T2IF = 0;    PPSUnLock;  // SPI uses RP5 and RP4 (ignores SDI1)  PPSOutput(PPS\_RP4, PPS\_SCK1OUT);  PPSOutput(PPS\_RP5, PPS\_SDO1); // RP5 outputs to SDI of slave  PPSInput(PPS\_SDI1, PPS\_RP6); //Slave output goes to RP6  PPSLock;  SPI1CON1 = 0;  SPI1CON1bits.MSTEN = 1; // master mode  SPI1CON1bits.MODE16 = 0; // Single byte  SPI1CON1bits.CKE = 0; //the next two lines make active edge the rising one  SPI1CON1bits.CKP = 0;  SPI1CON1bits.SPRE = 0b110; // secondary prescaler = 2 //these lines correspond to 8MHz  SPI1CON1bits.PPRE = 0b11; // primary prescaler = 1;  // SPI1CON1bits.PPRE = 0b01; // primary prescaler = 16;  SPI1CON2 = 0;  SPI1CON2bits.SPIBEN = 1; //enhanced buffer enabled  SPI1STAT = 0;  SPI1STATbits.SISEL = 0b001; //interrupt when data available in receive buf  SPI1STATbits.SPIEN = 1;    IFS0bits.SPI1IF = 0;  IFS0bits.T2IF = 0;  TMR2 = 0;    IPC1bits.T2IP = 5;  IEC0bits.T2IE = 1;  T2CONbits.TON = 1;    TRISBbits.TRISB7 = 1; //enable TC72 (pull CE high)    //TC72 configuration, enhanced mode so back to back can be done with no wait  SPI1BUF = 0x80; //address byte to FIFO  SPI1BUF = 0b00010001; //Sets TC72 to continuous mode to FIFO    }  int main(int argc, char \*argv[])  {  int i, msb, lsb, cont,temp,isnegative;  setup();          /\*  // if you want to send data back-to-back, use either polling  // like below, or use an ISR for the SPI  for (i=0 ; i<10 ; i++) {  SPI1BUF = i;  while (\_SPI1IF==0);  \_SPI1IF = 0;  }  \*/  while (1){  while(IFS0bits.T2IF == 0); //wait 150ms  IFS0bits.T2IF =0;    SPI1BUF = 0x02; //addresses read opeation  while(IFS0bits.SPF1IF == 0); //wait for data to be available in receive  IFS0bits.SPF1IF = 0;  msb = SPI1BUF; //msb of temp  lsb = SPI1BUF; //lsb of temp  cont = SPI1BUF; //control byte    if(cont != 0b00010001) //if control not right  SPI1BUF = 0b00010001; //rewrite    temp = msb;  temp <<= 8; //shift over 8 bits  temp += lsb;  temp >>= 8; //shift back 8 bits        }  return 0; // never reached (we hope)  } |

2. (40 points) Use the output compare module and UART module to generate a signal that has the following properties:

1. (5 points) The output compare module works in the PWM mode.
2. (5 points) The rising edges of the output signal are 1ms apart.
3. (5 points) The initial high pulse width is 0.1ms.
4. (5 points) The baud rate of the UART is 9600 bps, using two stop bits and even parity.
5. (10 points) If the UART is received a character ‘+’, the high pulse width will be increased by 0.2ms. The high pulse width should not exceed 0.9ms.
6. (10 points) If the UART is received a character ‘-’, the high pulse width will be decreased by 0.2ms. The high pulse width should not below 0.1ms.

Write a program to implement the above requirements.

|  |
| --- |
| #include <p24Fxxxx.h>  #include <PPS.h>  #include <xc.h>  #include <stdlib.h>  #if defined(USE\_CRYSTAL)  #pragma config POSCMOD = HS // Primary Oscillator Select (Primary oscillator disabled)  #else  #pragma config POSCMOD = NONE // don't use primar oscillator at all  #endif  #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)  #if defined(USE\_CRYSTAL)  #pragma config OSCIOFNC = OFF // Primary Oscillator Output Function (OSC2/CLKO/RC15 functions as part of clock)  #else  #pragma config OSCIOFNC = ON // Primary Oscillator Output Function (OSC2/CLKO/RC15 functions as port I/O (RC15))  #endif  #pragma config FCKSM = CSECME // Clock Switching and Monitor (Clock switching is enabled, Fail-Safe Clock Monitor is enabled)  #if defined(USE\_CRYSTAL)  #pragma config FNOSC = PRIPLL // Oscillator Select (Fast RC Oscillator with PLL module (FRCPLL))  #else  #pragma config FNOSC = FRCPLL  #endif  #pragma config SOSCSEL = SOSC // Sec Oscillator Select (Default Secondary Oscillator (SOSC))  #pragma config WUTSEL = LEG // Wake-up timer Select (Legacy Wake-upTimer)  #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 unsigned int width = 1599; //initial high width  void \_\_attribute\_\_((\_\_interrupt\_\_,\_\_auto\_psv\_\_)) \_T2Interrupt(void)  {  IFS0bits.T2IF = 0;  OC1RS = width; //set pulse width  Nop(); //breakpoint  }  void PPutch(const unsigned int c)  {  while (!IFS0bits.U1TXIF) ;  IFS0bits.U1TXIF = 0;  U1TXREG = c;  }  unsigned int PGetch(void)  {  unsigned int x;  while (!IFS0bits.U1RXIF) ;  x = U1RXREG;  IFS0bits.U1RXIF = 0;  return x;  }  void setup(void)  {  CLKDIVbits.RCDIV = 0;  AD1PCFG = 0x9fff; // For digital I/O. If you want to use analog, you'll  // need to change this.  // add your configuration commands below  PORTB = 0xffff;  // TRISB = 0x7fff; // bit 15 is an output    U1MODE = 0;  U1MODEbits.STSEL = 1; //two stop bits  U1MODEbits.PDSEL = 1; //even parity  U1MODEbits.BRGH = 0; //low speed mode  U1BRG = 103; // 9600 baud  U1MODEbits.UEN = 0;  U1MODEbits.UARTEN = 1;  U1STAbits.UTXEN = 1;    PPSUnLock;  PPSOutput(PPS\_RP3,PPS\_OC1); //set output compare pin  PPSInput(PPS\_U1RX,PPS\_RP5); // set UART pins  PPSOutput(PPS\_RP4,PPS\_U1TX);  PPSLock;  T2CON = 0; //Set timer2  PR2 = 15999; // 1ms  TMR2 = 0;  IFS0bits.T2IF = 0;  IEC0bits.T2IE = 1;  OC1CON = 0;    OC1R = 1599; //initial compare value, proportional to 0.1 ms  OC1RS = 1599; //set OC1R to this value if not change by uart thing  OC1CONbits.OCTSEL = 0; //use TMR2  OC1CONbits.OCM = 6; //PWM mode  T2CONbits.TON = 1;  }  int main(int argc, char \*argv[])  {  unsigned int returned;    setup();  while(1){  returned = PGetch(); //get character  if(returned == '+'){  if(width < 14399) //increment 0.2 ms if not already 0.9ms  width += 3200;  else;    }  if(returned = '-') {  if(width > 1599) //decrement by 0.2 ms if not already at 0.1ms  width -=3200;  else;  }      }  return 0; // never reached (we hope)  } |

3. (20 points) Write a program to configure the PIC24 ADC module to perform the following tasks:

1. (5 points) Sample and convert the analog input on the AN2 pin and AN8 pin at the fastest TAD. (Note that TAD > 75ns, and TCY = 62.5ns)
2. (2 points) Set the sampling time to 625ns.
3. (8 points) Sample and convert each pin every 200ms by using Timer 1.
4. (2 points) AD1IF should be raised after 8 samples are converted.
5. (3 points) Store the average sampling results of AN2 in the variable AN2Buffer. Store the average sampling results of AN8 in the variable AN8Buffer.

Paste your solution below.

|  |
| --- |
| #include <p24Fxxxx.h>  #include <PPS.h>  #include <xc.h>  #include <stdlib.h>  #if defined(USE\_CRYSTAL)  #pragma config POSCMOD = HS // Primary Oscillator Select (Primary oscillator disabled)  #else  #pragma config POSCMOD = NONE // don't use primar oscillator at all  #endif  #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)  #if defined(USE\_CRYSTAL)  #pragma config OSCIOFNC = OFF // Primary Oscillator Output Function (OSC2/CLKO/RC15 functions as part of clock)  #else  #pragma config OSCIOFNC = ON // Primary Oscillator Output Function (OSC2/CLKO/RC15 functions as port I/O (RC15))  #endif  #pragma config FCKSM = CSECME // Clock Switching and Monitor (Clock switching is enabled, Fail-Safe Clock Monitor is enabled)  #if defined(USE\_CRYSTAL)  #pragma config FNOSC = PRIPLL // Oscillator Select (Fast RC Oscillator with PLL module (FRCPLL))  #else  #pragma config FNOSC = FRCPLL  #endif  #pragma config SOSCSEL = SOSC // Sec Oscillator Select (Default Secondary Oscillator (SOSC))  #pragma config WUTSEL = LEG // Wake-up timer Select (Legacy Wake-upTimer)  #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 unsigned int AN2Buff, AN8Buff;  void \_\_attribute\_\_((\_\_interrupt\_\_,\_\_auto\_psv\_\_)) \_T1Interrupt(void)  {  unsigned int temp2, temp8;    IFS0bits.T2IF = 0;  AD1CHSbits.CH0SA =2;    IFS0bits.AD1IF = 0;  AD1CON1bits.SAMP = 1; // start sampling, start converting  while (!IFS0bits.AD1IF);  IFS0bits.AD1IF = 0;  temp2 = ADC1BUF0;    AN2Buff = temp2 + 7 \* AN2Buff;  AN2Buff /= 7;    AD1CHSbits.CH0SA = 8;    IFS0bits.AD1IF = 0;  AD1CON1bits.SAMP = 1; // start sampling, start converting  while (!IFS0bits.AD1IF);  IFS0bits.AD1IF = 0;  temp8 = ADC1BUF0;    AN8Buff = temp8 + 7 \* AN8Buff;  AN8Buff /= 7;    }  void setup(void)  {  CLKDIVbits.RCDIV = 0;  AD1PCFG = 0x9fff; // For digital I/O. If you want to use analog, you'll  // need to change this.  // add your configuration commands below  PORTB = 0xffff;  // TRISB = 0x7fff; // bit 15 is an output  AD1CON1 = 0;  AD1CHS = 0;  AD1PCFGbits.PCFG2 = 0; //set AN0 and AN8 to be analog NOTE not avail on 28 pin pic  AD1PCFGbits.PCFG8 = 0;  AD1CON2 = 0;  AD1CON3bits.ADCS = 1; // Tad = 2\*Tcy = 125ns (minimum that is > 75ns)  AD1CON3bits.ADRC = 0;  AD1CON3bits.SAMC = 5; // sample time = 5\*125 ns = 625ns  AD1CHSbits.CH0SA = 0; // channel selected is channel 0  AD1CON1bits.SSRC = 0b111; // auto-conversion  AD1CON1bits.ASAM = 0; // manual-sample  AD1CON1bits.ADON = 1;  AD1CON2bits.SMPI = 7; //interrupt after 8 samples    T1CON = 0; //Set timer2  T1CONbits.TCKPS = //3; 1:256 prescale  PR1 = 12499; // 200ms period  TMR1 = 0;  IFS0bits.T1IF = 0;  IEC0bits.T1IE = 1;    T1CONbits.TON = 1;  }  int main(int argc, char \*argv[])  {    setup();  while (1)  {    }  return 0; // never reached (we hope)  } |

Bonus Question: (10 points) Use the CORDIC method to perform 1.3 / 1.5. Show me your solution for 5 iterations.

1. (3 points) What is the initial value for X, Y, and Z?
2. (5 points) Complete the table below.
3. (2 points) Where is the final approximation results of 1.3/1.5 after 5 iterations?

|  |  |  |  |
| --- | --- | --- | --- |
| Number of Iteration | X = | Y = | Z = |
| 1 | X1 = | Y1 = | Z1 = |
| 2 | X2 = | Y2 = | Z2 = |
| 3 | X3 = | Y3 = | Z3 = |
| 4 | X4 = | Y4 = | Z4 = |
| 5 | X5 = | Y5 = | Z5 = |