# EE2361 Summer 2015, HW2 Due 7/17/2015 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.* ***INCLUDE SCREEN SHOTS of mplabx showing your code works (multiple screen shots for multiple test cases)***

1. Write a program (in C) to flip RB2 every 1/9 ms using Timer 1. Use polling. How accurate is your solution?

|  |
| --- |
| Show your timer calculations:  TIME = ((PR1 +1 ) \*PRESCALE)/INSTRUCT\_FREQ  111.11E-6 = ((PR1 +1) \*1)/16000000  111.11E-6 \* 16E6 = PR1 +1  1777.78 = PR1 +1  PR1 = 1776.78 which is approx 1777 |
| #include <p24Fxxxx.h>  #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)  /\*  \*  \*/  int main(void) {  CLKDIVbits.RCDIV = 0; //set clock  AD1PCFG = 0x9fff; //sets io to be digital  TRISBbits.TRISB2 = 0; //set RB2 to output    T1CON = 0; //timer no prescaling, disabled  T1CONbits.TCKPS = 0; // set pre-scalar 1:1  PR1 = 1777; //  IFS0bits.T1IF = 0;  TMR1 = 0; //reset timer  T1CONbits.TON = 1;    while(1){  while(IFS0bits.T1IF == 0); //wait 1/9 ms  LATBbits.LATB2 ^= 1; //toggle RB2  IFS0bits.T1IF = 0;  }  } |
| I used a break point at the point in the loop where the output changes and then used the stopwatch to find the amount of time it takes between output changes. This gave me either 111 us or 111.3125 us for every trial, over ten tries yielding a variance of 0.16 us and an average of 111.125 us (almost exactly 1/9 ms) |

1. The same as above, but use interrupts. How accurate is your solution?

|  |
| --- |
| The same calculation was used as the first problem, leading to PR1 = 1777 |
| #include <p24Fxxxx.h>  #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\_\_)) \_T1Interrupt(void)  {  LATBbits.LATB2 ^= 1; //toggle output  IFS0bits.T1IF = 0;    }  int main(void) {  CLKDIVbits.RCDIV = 0; //set clock  AD1PCFG = 0x9fff; //sets io to be digital  TRISBbits.TRISB2 = 0; //set RB2 to output    T1CON = 0; //timer no prescaling, disabled  T1CONbits.TCKPS = 0; // set pre-scalar 1:1  PR1 = 1777; //  TMR1 = 0; //reset timer    IFS0bits.T1IF = 0; //clear interrupt flag  IPC0bits.T1IP = 3; //set interrupt priority  IEC0bits.T1IE = 1; //enable interrupts    T1CONbits.TON = 1;    while(1){  asm("nop"); //give the loop something (not really) to do  }  } |
| I put a break point in the ISR where the output changes and then used the stop watch to measure the time it took for it to hit that point from the last time it was at it. The time ended up being 111.125 over every try, meaning no deviation and the timing almost being exactly 1/9 ms (111.111 us) |

1. With a prescalar value of 1:64, what are the minimum and maximum delays that you can make with Timer 2 as a 16-bit timer? How about Timer 2 paired with Timer 3 as a 32-bit timer?

Time = ((PRx + 1 ) \* PRESCALE)/INSTRUCT\_FREQ

16bit MINIMUM: PRx = 0x0000

((0x0000 + 1) \*64 )/ 16E6 = 64 / 16E6 = 4 us

16bit MAXIMUM: PRx = 0xFFFF

((0xFFFF + 1) \*64 )/ 16E6 = 65536 \*64 / 16E6

= 4194304/16E6 = 0.2621 second

32bit MINUMUM: PRx = 0x00000000

((0x00000000 + 1)/ 16E6 = 64 / 16E6 = 4 us

32bit MAXIMUM: PRx = 0xFFFFFFFF

((0xFFFFFFFF + 1)\*64))/16E= 274877906944/16E6 = 17179.8 s = 4.77 hours

1. Write a program to flip RB2 50ms after the user presses a key connected to RA0 (internal pull-up resistor enabled).

|  |
| --- |
| Show your timer calculations:  Prescale = 64  0.050 = ((PR1 + 1) \*64)/16E6  800000 = (PR1 + 1) \*64  12500 = PR1 + 1 → PR1 = 12499 |
| #include <p24Fxxxx.h>  #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)  /\*  \*  \*/  int main(void) {  CLKDIVbits.RCDIV = 0; //set clock  AD1PCFG = 0x9fff; //sets io to be digital  TRISBbits.TRISB2 = 0; //set RB2 to output  TRISAbits.TRISA0 = 1; //set RA0 as input  CNPU1bits.CN2PUE = 1; //pull up RA0    T1CON = 0; //timer no prescaling, disabled  T1CONbits.TCKPS = 2; // set pre-scalar 1:64  PR1 = 12499; //50 ms equiv    IFS0bits.T1IF = 0;  TMR1 = 0; //reset timer      while(1){  asm("nop"); //breakpoint  while(PORTAbits.RA0 == 1); //wait while not pressed  T1CONbits.TON = 1; //turn on timer  while(IFS0bits.T1IF == 0); //wait 50 ms  LATBbits.LATB2 ^= 1; //toggle RB2    IFS0bits.T1IF = 0; //reset int. flag  T1CONbits.TON = 0; //kill timer  TMR1 = 0; //reset timer  }  } |
| I applied a stimulus to RA0 that started off high, went low at 25ms, went high at 50ms, went low at 100ms and went high again at 125ms. I had a breakpoint right before the debounce for the switch so when I used a stop watch I could tell how long the wait for the switch to close plus the delay was. I set it up so that there would be 25ms of wait, meaning the total time the stopwatch should read to return to the breakpoint should be 75ms (25ms wait + 50 ms delay). My test of it found the stopwatch had the expected delay of 75ms, meaning the code worked. |

1. We have two high-pulse sources. The goal is to count pulses from either source and turn on RB2 when the total count is 10 or greater. Use the INT0 pin as one source, and RP8 connected to INT1 as the other source. Use interrupts for both the INT0 and INT1 events.

|  |
| --- |
| #include <p24Fxxxx.h>  #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 unsigned int pulses = 0;  void \_\_attribute\_\_((\_\_interrupt\_\_,\_\_auto\_psv\_\_)) \_INT0Interrupt(void)  {  \_INT0IF = 0;  pulses++;  }  void \_\_attribute\_\_((\_\_interrupt\_\_,\_\_auto\_psv\_\_)) \_INT1Interrupt(void)  {  \_INT1IF = 0;  pulses++;  }  int main(void) {  CLKDIVbits.RCDIV = 0; //set clock  AD1PCFG = 0x9fff; //sets io to be digital  TRISBbits.TRISB2 = 0; //set RB2 to output  TRISBbits.TRISB7 = 1; //set RB7 as input (INT0 pin)  TRISBbits.TRISB8 = 1; //set RB8 as input (INT1 pin)  PORTB = 0;    \_INT0IF = 0;  IPC0bits.IC1IP = 4; //INT0 priority = 4  INTCON2bits.INT0EP = 0; //interrupt on pos edge    \_INT1IF = 0;  RPINR0bits.INT1R = 8; //assign int1 to rp7  IPC5bits.INT1IP = 3; //priority = 3  INTCON2bits.INT1EP = 0; //interrupt on pos edge  IEC0bits.INT0IE = 1; //turn on interrupts  IEC1bits.INT1IE = 1;    while(1){  if(pulses >= 10)  LATBbits.LATB2 = 1; //at 10 pulse set RB2 high  }  } |
| I set up a stimulus that put a 10us positive pulse alternating on RP7 and RP8 every 100us. First, I made it so it only had 9 pulses and observed that RB2 remained low. I added another pulse and it went high. |
| A problem I found is if you have a two fast pulses (say high width of 1 cycle separated by one cycle) on one input, the stimulus will register as one pulse because the micro controller simply can't respond to interrupt flags fast enough (it takes 6 cycles after for an interrupt to even start performing an ISR). Essentially two fast pulses will cause the flag to be set and when the second pulse comes the flag will still be set/untended by the processor. |

1. Write a program to flip RB2 every 500 ms. Make RB3 to flip at the same frequency, but with a 100ms phase shift compared to RB2. Use Timer 1 for RB2 and Timer 2 for RB3 and use interrupts on both. How accurate is your solution?

|  |
| --- |
| 500 ms delay: Used 256:1 prescale  0.5 = ((PRx+1)\*256)/16E6  8E6/256 = PRx + 1  31250 = PRx +1  PRx = 312500  100 ms delay: Used 256:1 prescale  100ms is 1/5 of 500ms,  therefore the PRx value should be 1/5 that of the PRx for 500ms  31249 / 5 = 6299 = PRx |
| #include <p24Fxxxx.h>  #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\_\_)) \_T1Interrupt(void)  {  IFS0bits.T1IF = 0;  LATBbits.LATB2 ^= 1;  }  void \_\_attribute\_\_((\_\_interrupt\_\_,\_\_auto\_psv\_\_)) \_T2Interrupt(void)  {  IFS0bits.T2IF = 0;  LATBbits.LATB3 ^= 1;  }  int main(void) {  CLKDIVbits.RCDIV = 0; //set clock  AD1PCFG = 0x9fff; //sets io to be digital  TRISBbits.TRISB2 = 0; //set RB2 to output  TRISBbits.TRISB3 = 0; //set RB3 as output  PORTB = 0;    T1CON = 0; //timer no prescaling, disabled  T1CONbits.TCKPS = 3; // set pre-scalar 1:254    PR1 = 31249; //500 ms equiv    T2CON = 0; //timer no prescaling, disabled  T2CONbits.TCKPS = 3; // set pre-scalar 1:254  PR2 = 6299; //100 ms equiv    IFS0bits.T1IF = 0; //clear flag, enable tmr 1 interrupt  IEC0bits.T1IE = 1;  TMR1 = 0; //reset timer    IFS0bits.T2IF = 0;  TMR2 = 0;    T1CONbits.TON = 1; //start timers  T2CONbits.TON = 1;    while(IFS0bits.T2IF == 0); //delay RB3 100 ms  T2CONbits.TON = 0; //stop 100ms tmr2  IFS0bits.T2IF = 0;  TMR2 = 0;  PR2 = 31249; //configure RB3 500ms toggle period  IEC0bits.T2IE = 1; //enable tmr 2 interrupt  T2CONbits.TON = 1; //start 500 ms timer for tmr2    while(1){  asm("nop"); //some menial task  }  } |
| I tested it by putting break points in the ISR's for each timer, and then using the stopwatch to measure the delay between the changes in output in the ISR's. I measured that RB3 changed 100.785 ms after RB2, which is what was expected, and that RB2 changed 399.199 ms after RB3. This corresponds to a 500 ms time between output changes on each pin as desired. I verified the period to be 500 ms by putting a break in a single ISR and measuring the time between the ISR being executed. It was 500 ms most of the time, and occasionally off by a fraction of a microsecond. Overall the timing is quite accurate, with just the 100 ms delay being slightly off due to the resolution of the timer with a 1:256 prescale |

1. The same as above, but only use only Timer 1.

|  |
| --- |
| 100 ms delay with 64:1 prescale:  0.1 = ((PR1 + 1)\*64)/16E6  0.1\*16E6/64 = PR1 +1  PR1 = 24999 |
| #include <p24Fxxxx.h>  #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 delay(int a){ //delays the number used as argument \* 100ms  while(a > 0){  TMR1 = 0;  T1CONbits.TON = 1;    while(IFS0bits.T1IF==0);  IFS0bits.T1IF = 0;  T2CONbits.TON = 0;  a--;  }  }  int main(void) {  CLKDIVbits.RCDIV = 0; //set clock  AD1PCFG = 0x9fff; //sets io to be digital  TRISBbits.TRISB2 = 0; //set RB2 to output  TRISBbits.TRISB3 = 0; //set RB3 as output  PORTB = 0;    T1CON = 0; //timer no prescaling, disabled  T1CONbits.TCKPS = 2; // set pre-scalar 1:64    PR1 = 24999; //100 ms equiv  IFS0bits.T1IF = 0;    while(1){  LATBbits.LATB2 ^= 1; //toggle RB2  delay(1); //delay RB3 100 ms  LATBbits.LATB3 ^= 1; //toggle RB3  delay(4);  }  } |
| I tested the program by putting break points at the lines where I toggle RB2 and RB3, and then ran the code and used the stopwatch to determine the time delays between the two pins changing (looking at PORTB in watch to make sure they change). I found that RB3 changed almost exactly 100 ms as expected after RB2 (99.998625 ms) and RB2 change almost exactly 400 ms after RB3 (399.989375 ms). This corresponds to 500 ms between changes. |

1. Use Input Capture to determine if the input signal on RP7 has a frequency between 10KHz and 500KHz. Use interrupts. RB2 should turn on when the frequency is between the two values and turn off when it isn’t.

|  |
| --- |
| Show your timer calculations:  10 KHz period = 16E6/10E3 -1 = 1599 cycles  500 Khz period = 16E6/5E5 -1 = 31 cycls |
| #include <p24Fxxxx.h>  #include <xc.h>  #include <PPS.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)  /\*  This program monitors an external signal connected to the IC1/RP3 pin, and  \* sets RB0 if the duty cycle is more than 50% in each period. Uses polling on  \* IC1IF. Uses T2 overflow count in T2Int to ensure timer overflows do not  \* result in the wrong subtraction of consecutive ICBUF values.  \*  \* To test the program: Use Window > Simulator > Stimulus. Click on the  \* Pin/Register Actions, choose cyc as Time Units. Add the IC1 signal. Add the  \* following rows:  \* Time IC1  \* 1000 1  \* 1200 0  \* 1500 1  \* 2000 0  \* 2100 1  \*  \* The first cycle has a (1500-1200)/(1200-1000) duty cycle ==> RB0 = 0  \* The second cycle has a (2100-2000)/(2000-1500) duty cycle ==> RB0 = 1  \*/  volatile unsigned long int tOverflowCount = 0; // count timer overflows  void \_\_attribute\_\_((\_\_interrupt\_\_,\_\_auto\_psv\_\_)) \_T2Interrupt(void)  {  IFS0bits.T2IF = 0;  ++tOverflowCount;  }  void setup(void)  {  CLKDIVbits.RCDIV = 0; // make 16MHz  // setup IC pin as input, RB0 output  AD1PCFG = 0x9fff;  TRISBbits.TRISB7 = 1; // RP7 input  TRISBbits.TRISB2 = 0; // RB2 output  PORTB=0;  // setup T2 to get a long time before roll over  T2CON = 0;  PR2 = 64999;  TMR2 = 0;  IFS0bits.T2IF = 0;  T2CONbits.TON = 1;  // Setup the IC pin  PPSUnLock;  PPSInput(PPS\_IC1,PPS\_RP7);  PPSLock;  // Setup the IC unit to wait for a rising edge, use T2 as timer base.  IC1CON=0;  IC1CONbits.ICTMR = 1; // use T2 as timer base.  \_IC1IF = 0;  IC1CONbits.ICM = 3; // every rising edge  }  // MAKE SURE you set ICM to the right edge polarity before calling this.  // returns IC1BUF + overflow time for T2.  // temporarily disables interrupts to ensure the 32-bit value  // of tOverFlowCount is fetched correctly (e.g., if the val is 0x12FFFF,  // the interrupt might change it to 0x130000 in the middle of fetching the  // 32-bit value, ending up in getting either 0x120000 or 0x13FFFF).  unsigned long int getEdgeTime()  {  char prevIntLevel;  unsigned long int ovCount;    while (\_IC1IF==0);  \_IC1IF = 0;  prevIntLevel = \_IPL;  \_IPL = 7; // disable interrupts temporarily  ovCount = tOverflowCount;  \_IPL = prevIntLevel; // re-enable interrupts  return IC1BUF + 65000L\*ovCount;  }  int main(void)  {  unsigned long int alpha = 0, beta;  unsigned long int period; // see the comment below.  setup();  /\*  \* Waveform:  \* +-----------+ +-------+ +  \* | | | | |  \* ------------+ +-------+ +-----------+  \* alpha beta gamma  \*  \* alpha: beginning of cycle  \* beta: falling edge  \* gamma: end of cycle = beginning of next cycle  \*  \* positive = beta - alpha  \* negativ = gamma - beta  \*/  // get the first alpha  IC1CONbits.ICM = 3; // every rising edge  alpha = getEdgeTime();  while (1) {    IC1CONbits.ICM = 3; // get rising edge  beta = getEdgeTime();  period = beta - alpha;  if ((period > 32)&&(period << 1599)) //checks and sets RB2  PORTBbits.RB2 = 1; //according to freq  else PORTBbits.RB2 = 0;  // we are all set for the next cycle: alpha is already set.  }  } |
| How did you test your program? Include a screen shot of mplabx showing that your program works. Show four test cases: one just below 10KHz, one just above 10KHz, and two slightly above and below 500KHz. |