Terence Henriod

HW 12

CPE 301: Microprocessors System Design

November 21, 2013

1. **Determine the appropriate bit settings for TCCR1A, TCCR1B, OCR1A, and TIMSK1 using the following specific details:**

**– use NORMAL MODE,**

**– disconnect both OC1 pins,**

**– use a pre-scale division of AN APPROPRIATE VALUE,**

**– use the TIMER OVERFLOW FLAG interrupt,**

**– use a TIMER COUNT INITIAL value such that you receive 1 interrupt per second.**

|  |  |
| --- | --- |
| TCCR1A | 0b00000000 |
| 0000 to disconnect the Output Compare pins (OC1)  00 reserved, write 0  00 as part of Normal Mode WGM11:10(see TCCR1B for rest) |
| TCCR1B | 0b00000000 |
| 00 disable InputCaptureNoiseCanceler and IC Edge Select  0 reserved, must be written 0  00 for Normal Mode WGM13:12 (seeTCCR1A for the rest)  000 to stop the clock  001 to run the timer/counter with no pre-scale (1)  010 to run the timer/counter with a pre-scale of 8  100 to run the timer/counter with a pre-scale of 256  // good for one second  101 to run the timer/counter with a pre-scale of 1024 (use \_\_\_\_\_\_\_\_\_\_\_this one for timing in full seconds (more than 1)) |
| TIMSK1 | 0b00000001 |
| 00 reserved, write 0  0 disable Input Capture Interrupt  0 reserved, write 0  000 disable Output Compare Match Interrupts for C:A  1 set the Overflow Interrupt Enable (OIE) |
| TIFR1 | 0b00101111 |
| 00 reserved  1 clear the TC Input Capture Flag  0 reserved  111 clear the TC Output Compare Match C:A flags  1 clear the timer overflow flag |
| TCNT1 | 49911 // in decimal |
| Use the formula 0xFF + 1 – ( ( 1/\*Hz\*/ / 1 ) /( 1024 / 16000000 )  With appropriate casts like long or long long |
| OCR1A | 0 |
| Simply clear the register to ensure the register is cleared of any problematic value. |

1. **Create a function that initializes timer 1 based on the values determined in problem 9.1.**

void TC1\_init()

{

// ensure clock is stopped, setup normal mode, disable output compares,

// enable the Timer Overflow Interrupt

\*myTCCR1B = 0b00000000;

// 0000 to disconnect the Output Compare pins (OC1)

// 00 reserved, write 0

// 00 as part of Normal Mode WGM11:10(see TCCR1B for others)

\*myTCCR1A = 0b00000000;

// 00 disable InputCaptureNoiseCanceler and ICEdgeSelect

// 0 reserved, must be written 0

// 00 for Normal Mode WGM13:12 (seeTCCR1A for the rest)

// 000 to stop the clock

// ======== FYI =============

// 001 to run the timer/counter with no pre-scale (1)

// 010 to run the timer/counter with a pre-scale of 8

// 100 to run the timer/counter with a pre-scale 0f 256

// 101 to run the timer/counter with a pre-scale of 1024

\*myTIFR1 = 0b00101111;

// 00 reserved

// 1 to clear TC Input Capture Flag

// 0 reserved

// 111 to clear TC Output Compare C:A Match flags

// 1 to clear TimerOVerflow flag

\*myTIMSK1 = 0b00000001;

// 00 reserved, write 0

// 0 disable Input Capture Interrupt

// 0 reserved, write 0

// 000 disable Output Compare Match Interrupts for C:A

// 1 set the Overflow Interrupt Enable (OIE)

// no return - void

}

1. **Create an ISR for timer 1. (Look ahead at 9.7)**

ISR( TIMER1\_OVF\_vect, ISR\_BLOCK )

{

// stop the timer

\*myTCCR1B &= TIMER\_STOP\_MASK;

// reset the timer counter value to count up to another second

\*myTCNT1 = ONE\_SEC\_TC\_START;

// increment the seconds counter

gSecondsCompleted++;

// re-start the timer

\*myTCCR1B |= TIMER\_PRE\_256;

// no return - ISR

}

1. **Determine the appropriate bit settings for PCICR and PCMSK2 using the following specific details:**

**– use Port D4 as an input pin,**

**– disable the internal pull-up resistor,**

**– using Port D4 corresponds to Pin Change interrupt 20.**

|  |  |
| --- | --- |
| PCICR  (0x68) | 0b00000100 |
| 00000 reserved  1 enables pin change interrupts 23:16 (particularly 20)  00 disable pin change interrupts 15:0 |
| PCMSK2  (0x6D) | 0b00010000 |
| 000 disable pin change interrupts 23:21  1 enable pin change interrupt 20 for D4  0000 disable pin change interrupts 19:16 |
| PCIFR  (0x3B) | 0b00000111 (Not Necessary) |
| 00000 reserved  1 clear the PCInterrupt flag for 23:16  11 clear the PCInterrupt flags for 15:0  The execution of the ISR will also do this |

1. **Create a function that initializes the Pin Change interrupt based on the values determined in problem 9.4.**

void portInD\_init()

{

// set D4 as input

\*DDR\_D &= 0b11101111;

// 111 leave pins 7:5 alone

// 0 set pin 4 as input

// 1111 leave pins 3:0 alone

\*portInD &= 0b11101111;

// 111 leave pins 7:5 alone

// 0 set pin 4 as input

// 1111 leave pins 3:0 alone

\*portOutD &= 0b11101111;

// 111 leave pull-up resistors 7:5 alone

// 0 disable pull-up resistor 4

// 1111 leave pull-up resistors 3:0 alone

// setup the pin change interrupt for port-in D (20)

\*myPCICR = 0b00000100;

// 00000 reserved

// 1 enables pin change interrupts 23:16 (particularly 20)

// 00 disable pin change interrupts 15:0

\*my PCMSK2 = 0b00010000

// 000 disable pin change interrupts 23:21

// 1 enable pin change interrupt 20 for D4

// 0000 disable pin change interrupts 19:16

\*myPCIFR = 0b00000111

// 00000 reserved

// 1 clear the PCInterrupt flag for 23:16

// 11 clear the PCInterrupt flags for 15:0

// likely unnecessary, execution of the ISR will also do this

// no return - void

}

1. **Create an ISR for Pin Change interrupt 20. (Look ahead at 9.7)**

ISR( PCINT2\_vect, ISR\_BLOCK )

{

// increment the change counter

gNumChangesD4++;

// no return - ISR

}

1. **Create a program that utilizes the timer interrupt and the pin change interrupt in order to measure the frequency of a periodic signal on Port Pin D4. You can do this by saving and then clearing a counter each time the timer interrupt occurs (at a rate of once per second). Additionally, you can increment the counter by the number of low to-high and high-to-low transitions on pin D4 with the pin change interrupt. Your main loop should monitor when the timer interrupt goes off, and then write the frequency to the serial port for the user to see. Note: your setup function should call your two initialization functions, and then enable global interrupts in the SREG register.**

**/\*=============================================================================**

**Creator: Terence Henriod**

**Course: CPE301**

**Section: 1101**

**Program Name: HW12 Timer and Pin Change Interrupts**

**Description: This program utilizes both timer overflow and pin change**

**Interrupts to display the frequency of the oscillation of a**

**signal at port-in D4**

**Revision #: v0.01**

**Date: 11/23/2013**

**THIS CODE MAY BE ASSUMED TO BE WRITTEN UNDER THE GNU PUBLIC LICENSE AND IS**

**THEREFORE FREE. FREE DOES NOT NECESSARILY MEAN FREE OF COST, BUT FREE FOR**

**DISTRIBUTION AND MODIFICATION. HOWEVER, THE AUTHOR OF THIS CODE MUST BE CITED**

**WHEN THIS CODE IS USED, DISTRIBUTED, OR MODIFIED.**

**==============================================================================\*/**

**/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**Constants**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/**

**/\*~~~~~ Program/Hardware Settings ~~~~~\*/**

**const unsigned int BAUD\_RATE = 9600;**

**const unsigned long int CPU\_FREQ = 16000000;**

**const unsigned char TIMER\_STOP\_MASK = 0xF8;**

**const unsigned char TIMER\_PRE\_256 = 0x04;**

**/\*~~~~~ Boolean Values ~~~~~\*/**

**const unsigned int TRUE = 1;**

**const unsigned int FALSE = 0;**

**/\*~~~~~ Character Values ~~~~~\*/**

**const unsigned char G\_INTERRUPT\_MASK = 0x80;**

**const unsigned char BS = 0x08;**

**const unsigned char ESC = 0x1B;**

**/\*~~~~~ Integer Values ~~~~~\*/**

**const unsigned int NUM\_LEN = 6; // strings of this size can contain 65535**

**const unsigned int ONE\_SEC\_TC\_START = 3036;**

**// ( 0xFFFF + 1 – (long)( ( CPU\_FREQ / 256 ) ) ) don’t like having Arduino**

**// do math unless necessary**

**/\*~~~~~ String Values ~~~~~\*/**

**const char ENDL[] = "\r\n";**

**/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**Global Variables (Yes, these are global variables!)**

**Mostly used for register pointers that can't be declared constant and ISR**

**dependencies**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/**

**/\*~~~~~ AVR Status Register ~~~~~\*/**

**volatile unsigned char\* AVR\_stat = (unsigned char\*) 0x5F;**

**/\*~~~~~ GPIO Registers ~~~~~\*/**

**volatile unsigned char\* portInD = (unsigned char\*) 0x29;**

**volatile unsigned char\* DDR\_D = (unsigned char\*) 0x2A;**

**volatile unsigned char\* portOutD = (unsigned char\*) 0x2B;**

**/\*~~~~~ Timer/Counter Registers ~~~~~\*/**

**volatile unsigned char\* myTCCR1A = (unsigned char\*) 0x80; // status/controlA**

**volatile unsigned char\* myTCCR1B = (unsigned char\*) 0x81; // status/controlB**

**volatile unsigned char\* myTIFR1 = (unsigned char\*) 0x36; // interrupts/flags**

**volatile unsigned char\* myTIMSK1 = (unsigned char\*) 0x6F; // interrupt mask**

**volatile unsigned int\* myTCNT1 = (unsigned int\*) 0x84; // for both H and L**

**/\*~~~~~ USART Registers ~~~~~\*/**

**volatile unsigned char\* myUCSR0A = (unsigned char \*) 0xC0; // Controls/StatusA**

**volatile unsigned char\* myUCSR0B = (unsigned char \*) 0xC1; // Controls/StatusB**

**volatile unsigned char\* myUCSR0C = (unsigned char \*) 0xC2; // Controls/StatusC**

**volatile unsigned char\* myUBRR0L = (unsigned char \*) 0xC4; // Baud RateL**

**volatile unsigned char\* myUBRR0H = (unsigned char \*) 0xC5; // Baud RateH**

**volatile unsigned char\* myUDR0 = (unsigned char \*) 0xC6; // Data Register**

**/\*~~~~~ Pin Change Interrupt Settings ~~~~~\*/**

**volatile unsigned char\* myPCICR = (unsigned char \*) 0x68; // PCinterruptEn**

**volatile unsigned char\* myPCMSK2 = (unsigned char \*) 0x6D; // PCinterruptMask**

**volatile unsigned char\* myPCIFR = (unsigned char \*) 0x3B; // PCinterruptFlag**

**/\*~~~~~ ISR Dependencies ~~~~~\*/**

**volatile static unsigned int gSecondsCompleted = 0;**

**volatile static unsigned int gNumChangesD4 = 0;**

**/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**Function Prototypes**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/**

**/\*~~~~~ Initialization Functions ~~~~~\*/**

**void TC1\_init();**

**void USART0\_init( unsigned int uBaud );**

**void portInD\_init();**

**/\*~~~~~ ISRs ~~~~~\*/**

**ISR( TIMER1\_OVF\_vect, ISR\_BLOCK );**

**ISR( PCINT2\_, ISR\_BLOCK );**

**/\*~~~~~ Essential Functions ~~~~~\*/**

**unsigned char kb\_hit();**

**unsigned char getch();**

**void putch( unsigned char data );**

**void myDelay( unsigned int units, unsigned int useMicro );**

**/\*~~~~~ String Functions ~~~~~\*/**

**void printString( const char\* str0 );**

**unsigned int strLen( const char\* str0 );**

**char\* myItoStr( unsigned int integer, char\* intString );**

**/\*~~~~~ Program Functionality ~~~~~\*/**

**/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**Setup**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/**

**void setup()**

**{**

**// initialize TimerCounter1**

**TC1\_init();**

**// initialize USART0**

**USART0\_init( BAUD\_RATE );**

**// initialize portInD**

**portInD\_init();**

**// enable interrupts globally**

**\*AVR\_stat |= G\_INTERRUPT\_MASK;**

**// no return - setup**

**}**

**/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**Loop**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/**

**void loop()**

**{**

**// variables**

**volatile static unsigned int lastSecond = gSecondsCompleted;**

**volatile char input = '$';**

**char theIntString[ NUM\_LEN ] = "";**

**char\* hertzVal = theIntString;**

**// notify user how to start**

**printString( "Press 'S' to begin the measurement\r\n" );**

**// wait for the user to start the measurement**

**while( ( input != 's' ) && ( input != 'S' ) )**

**{**

**// case: the user pressed a key**

**if( kb\_hit() )**

**{**

**// read the input**

**input = getch();**

**}**

**}**

**// notify user that reading has started**

**printString( "Now reading the oscillation frequency of PinD4\r\n" );**

**// start the timer in a mode that will count 1 second**

**\*myTCNT1 = ONE\_SEC\_TC\_START;**

**\*myTCCR1B |= TIMER\_PRE\_256;**

**// while the user wishes to measure the portInD oscillation (hasn’t quit)**

**// wait for the user to start the measurement**

**while( input != ESC )**

**{**

**// case: the timer counted a second**

**if( gSecondsCompleted != lastSecond )**

**{**

**// disable global interrupts for the duration of this operation**

**\*AVR\_stat &= ~G\_INTERRUPT\_MASK;**

**// determine the number of completed oscillations**

**gNumChangesD4 /= 2;**

**// convert the oscillation count to a string**

**hertzVal = myItoStr( gNumChangesD4, hertzVal );**

**// reset the pin change counter**

**gNumChangesD4 = 0;**

**// update the last second tracker**

**lastSecond = gSecondsCompleted;**

**// display the frequency to the screen**

**printString( hertzVal );**

**printString( " Hz" );**

**printString( ENDL );**

**// re-enable global interrupts to resume operation**

**\*AVR\_stat |= G\_INTERRUPT\_MASK;**

**}**

**// case: the user pressed a key**

**if( kb\_hit() )**

**{**

**// read the input**

**input = getch();**

**}**

**}**

**// no return – loop – restart from main**

**}**

**/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**Function Implementations**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/**

**/\*########################################### Initialization Functions ###\*/**

**void TC1\_init()**

**{**

**// ensure clock is stopped, setup normal mode, disable output compares,**

**// enable the Timer Overflow Interrupt**

**\*myTCCR1B = 0b00000000;**

**// 0000 to disconnect the Output Compare pins (OC1)**

**// 00 reserved, write 0**

**// 00 as part of Normal Mode WGM11:10(see TCCR1B for others)**

**\*myTCCR1A = 0b00000000;**

**// 00 disable InputCaptureNoiseCanceler and ICEdgeSelect**

**// 0 reserved, must be written 0**

**// 00 for Normal Mode WGM13:12 (seeTCCR1A for the rest)**

**// 000 to stop the clock**

**// ======== FYI =============**

**// 001 to run the timer/counter with no pre-scale (1)**

**// 010 to run the timer/counter with a pre-scale of 8**

**// 100 to run the timer/counter with a pre-scale 0f 256**

**// 101 to run the timer/counter with a pre-scale of 1024**

**\*myTIFR1 = 0b00101111;**

**// 00 reserved**

**// 1 to clear TC Input Capture Flag**

**// 0 reserved**

**// 111 to clear TC Output Compare C:A Match flags**

**// 1 to clear TimerOVerflow flag**

**\*myTIMSK1 = 0b00000001;**

**// 00 reserved, write 0**

**// 0 disable Input Capture Interrupt**

**// 0 reserved, write 0**

**// 000 disable Output Compare Match Interrupts for C:A**

**// 1 set the Overflow Interrupt Enable (OIE)**

**// no return - void**

**}**

**void USART0\_init( unsigned int uBaud )**

**{**

**// variables**

**unsigned int baudRegVal;**

**// clear the USART Controls and Status Register A**

**\*myUCSR0A = 0x20; // Set the Data Register Empty and Frame Error bits.**

**// clear all other flags: Receive Complete,**

**// Transmit Complete, Data OverRun, Parity Error,**

**// Double TX speed, and Multi-Processor Communication Mode**

**// setup USART Controls and Status Register B**

**\*myUCSR0B = 0x18; // only set the Receiver Enable and Transmitter Enable**

**// clear the Interrupt Enables (significant bits) and**

**// set character size to 5 bits (less significant) TODO: setto8bits?**

**// setup USART Controls and Status Register C**

**\*myUCSR0C = 0x06; // set an 8 bit character size (2:1)**

**// while setting the UMSEL (7:6) to asynchronus mode, disabling**

**// the Parity Mode (5:4), set the Stop Bit Select to 0 (1-bit),**

**// and set the UCPOL (Clock POLarity) to zero to receive sampled**

**// data on the falling edge of the clock and change the**

**// data transmitted on the rising edge**

**// compute the baud rate register value**

**/\* Note: using the given formula:**

**(((unsigned long)(CPU\_FREQ / ((unsigned long)( 16 \* uBaud )))) - 1)**

**the value to load the baud rate register with, can be done, however,**

**the computation is prone to error, so it may be easiest to simply**

**select the appropriate register value for a given baud rate.**

**\*/**

**// case: 2400 baud**

**if( uBaud == 2400 )**

**{**

**// load register appropriately**

**baudRegVal = 416;**

**}**

**// case: 4800 baud**

**else if( uBaud == 4800 )**

**{**

**// load register appropriately**

**baudRegVal = 207;**

**}**

**// case: 9600 baud**

**else if( uBaud == 9600 )**

**{**

**// load register appropriately**

**baudRegVal = 103;**

**}**

**// case: 115200 baud**

**else if( uBaud == 115200 )**

**{**

**// load register appropriately**

**baudRegVal = 8;**

**}**

**// case: a valid value was not given**

**else**

**{**

**// use 9600 baud by default**

**baudRegVal = 103;**

**}**

**// load the baud rate value into the baud rate registers**

**\*myUBRR0H = ( baudRegVal >> 8 ); // put only the high byte in high register**

**// ((1111 0000) >> 8) == (0000 1111)**

**\*myUBRR0L = ( baudRegVal & 0xFF ); // store only the low byte in low register**

**// no return - void**

**}**

**void portInD\_init()**

**{**

**// set D4 as input**

**\*DDR\_D &= 0b11101111;**

**// 111 leave pins 7:5 alone**

**// 0 set pin 4 as input**

**// 1111 leave pins 3:0 alone**

**\*portInD &= 0b11101111;**

**// 111 leave pins 7:5 alone**

**// 0 set pin 4 as input**

**// 1111 leave pins 3:0 alone**

**\*portOutD &= 0b11101111;**

**// 111 leave pull-up resistors 7:5 alone**

**// 0 disable pull-up resistor 4**

**// 1111 leave pull-up resistors 3:0 alone**

**// setup the pin change interrupt for port-in D (20)**

**\*myPCICR = 0b00000100;**

**// 00000 reserved**

**// 1 enables pin change interrupts 23:16 (particularly 20)**

**// 00 disable pin change interrupts 15:0**

**\*myPCMSK2 = 0b00010000;**

**// 000 disable pin change interrupts 23:21**

**// 1 enable pin change interrupt 20 for D4**

**// 0000 disable pin change interrupts 19:16**

**\*myPCIFR = 0b00000111;**

**// 00000 reserved**

**// 1 clear the PCInterrupt flag for 23:16**

**// 11 clear the PCInterrupt flags for 15:0**

**// likely unnecessary, execution of the ISR will also do this**

**// no return - void**

**}**

**/\*############################################################### ISRs ###\*/**

**ISR( TIMER1\_OVF\_vect, ISR\_BLOCK )**

**{**

**// stop the timer**

**\*myTCCR1B &= TIMER\_STOP\_MASK;**

**// reset the timer counter value to count up to another second**

**\*myTCNT1 = ONE\_SEC\_TC\_START;**

**// increment the seconds counter**

**gSecondsCompleted++;**

**// re-start the timer**

**\*myTCCR1B |= TIMER\_PRE\_256;**

**// no return - ISR**

**}**

**ISR( PCINT2\_vect, ISR\_BLOCK )**

**{**

**// increment the change counter**

**gNumChangesD4++;**

**// no return - ISR**

**}**

**/\*################################################ Essential Functions ###\*/**

**unsigned char kb\_hit()**

**{**

**// return the truth of Read Data Available flag in bit 7**

**return ( \*myUCSR0A & 0x80 );**

**}**

**unsigned char getch()**

**{**

**// wait for a kb\_hit**

**while( !kb\_hit() ) {};**

**// return the data read from Rx register**

**return \*myUDR0;**

**}**

**void putch( unsigned char data )**

**{**

**// wait for the transmit buffer to empty (TBE flag in bit 5)**

**while( ( \*myUCSR0A & 0x20 ) == FALSE ) {};**

**// write the data to the buffer to output it**

**\*myUDR0 = data;**

**// no return - void**

**}**

**/\*################################################### String Functions ###\*/**

**void printString( const char\* str0 )**

**{**

**// variables**

**volatile unsigned int ndx = 0;**

**// output all characters up to NULL**

**while( str0[ ndx ] != '\0' )**

**{**

**// output the character**

**putch( str0[ ndx ] );**

**// move to the next one**

**ndx++;**

**}**

**// no return - void**

**}**

**unsigned int strLen( const char\* str0 )**

**{**

**// variables**

**volatile unsigned int ndx = 0;**

**// output all characters up to NULL**

**while( str0[ ndx ] != '\n' )**

**{**

**// move to the next one**

**ndx++;**

**}**

**// return the number of characters**

**return ndx;**

**}**

**char\* myItoStr( unsigned int integer, char\* intString )**

**{**

**// variables**

**int interimInt = integer;**

**int ndx = 0;**

**// place the null terminator**

**intString[ NUM\_LEN - 1 ] = '\0';**

**// fill out the string with the digits of the integer**

**for( ndx = ( NUM\_LEN - 2 ); ndx >= 0; ndx-- )**

**{**

**// strip the least significant digit and place it in the string**

**intString[ ndx ] = ( ( interimInt % 10 ) + 0x30 );**

**// throw out the least significant digit**

**interimInt /= 10;**

**}**

**// return the resulting integer string**

**return intString;**

**}**

**Extra Credit**

1. **Connect a controllable signal generator source to port pin D4.Warning: you should calibrate the signal before connecting it to the Arduino. Do so by connecting the signal generator to an oscilloscope and verify you have a 0- to-5V source. Be sure to note the frequency. Use the program created in problem 9.7. Compare your output with that determined by the oscilloscope. Note: don’t forget the pin change interrupt occurs on both rising and falling edges, so you will need to account for that in your output to the user.**