Terence Henriod

HW 7

CPE 301: Microprocessors System Design

October 17, 2013

1. Give a detailed question about how ATmega timers work or how they are programmed.

Why are the timers designed such that the TOV flag needs to be manually cleared (in a strange manner at that – writing a logic 1 to set the bit to 0) or cleared using an input or output compare function or by serial interrupt? Why is the flag not reset every time the counter is stopped (i.e. when the CS0, CS1, and CS2 bits in the timer/counter control B register are set to 000 to stop the timer/counter)?

Also, if pre-scaling is a feature for the timer, why are two 2-byte registers necessary for the timer/counter? Is this just a lingering design feature from when Arduino clocks were slower? 16 MHz is awfully fast for doing things like timing in milliseconds. So is the first part of this question valid, or will we be programming things that require much more precise timing, that is, timing in small fractions of milliseconds?

1. Modify the Blink program you did for Lab 5 (Arduino UNO version on page 85 of the textbook) so that the function MyDelay(mSecondsApx) uses the Arduino ATmega2560 timer1 in Normal mode to generate a delay of (mSecondsApx \* one millisecond) before returning.

void setup()

{

// variables

volatile unsigned char\* portDDRB = (unsigned char\*) 0x24; // portB DDR (LED)

\*portDDRB |= 0x80; // set pin 7 to output

volatile unsigned char\* tcc1A = (unsigned char\*) 0x80; // timer/counter1 control A

\*tcc1A = 0; // set all bits to 0 for normal mode

// specifically WGM11 and WGM10

volatile unsigned char\* tcc1B = (unsigned char\*) 0x81; // timer/counter1 control B

\*tcc1B = 0; // set all bits to 0 for normal mode

// specifically WGM13 and WGM11

// and CS2:0 for clock stop

volatile unsigned char\* tcc1C = (unsigned char\*) 0x82; // timer/counter1 control C

\*tcc1C = 0; // ensure register flags are cleared

volatile unsigned char\* timsk1 = (unsigned char\*) 0x6F; // interrupt mask register

\*timsk1 = 0; // ensure these ar cleared too

volatile unsigned char\* tc1Flag = (unsigned char\*) 0x36; // timer/counter interrupt flag register

// NOT 0x16!!!

\*tc1Flag |= 0x01; // write 1 to clear flag (seriously)

// no return - void setup

}

void MyDelay( unsigned long int mSeconds )

{

// variables

// TimerCouNTer1 registers

volatile unsigned char\* tcnt1H = (unsigned char\*) 0x85; // more significant, 15:8

volatile unsigned char\* tcnt1L = (unsigned char\*) 0x84; // 7:0

// tcRegisters replaces the two above at the same time

volatile unsigned int\* tcRegisters; // unsigned int is assumed to be 16 bit

tcRegisters = (unsigned int\*) 0x84;

volatile unsigned char\* tcc1A = (unsigned char\*) 0x80; // timer/counter1 control A

volatile unsigned char\* tcc1B = (unsigned char\*) 0x81; // timer/counter1 control B

volatile unsigned char\* tc1Flag = (unsigned char\*)0x36; // contains TOV1 flag

volatile unsigned int timerStartVal = 0;

volatile unsigned int countupVal = 0;

// ensure the clock is stopped

// set the clock select bits for clock stopping

// i.e. CS02, CS01, CS00 to (000)

\*tcc1B &= 0xF8; // (1111 1000)

// calculate the number of ticks we need to wait

/\* This means we need to figure out how many ticks are necessary.

Assuming the frequency of the clock in the Arduino is 16.000 MHz,

this means that we get 16,000,000 ticks per second, 16 ticks

per millisecond.

If we use a pre-scale factor of 1024, then we perceive the timer

to oscillate at 15625 Hz. Invert this to find that every tick is then

perceived to occur every 6.4 E-5 seconds.

Now, we need to find a constant that will be correspond to the number

of ticks per millisecond. This is found by dividing 1 miliisecond

by the time it takes for a tick. This is 15.625. I will "ceiling" this

number (if necessary) and say that there are 16 ticks per millisecond. (10h)

\*/

// get countup value

countupVal = (unsigned int) ( mSeconds \* 15.625);

// calculate timer start value (MAX + 1 - countupVal)

timerStartVal = (unsigned int) (65536 - (long) countupVal);

// setup for timing

// store the calculated timer start value

//TODO: Use the 16 bit type defined in the header file

\*tcRegisters = timerStartVal;

// clear the TOV flag

\*tc1Flag |= 0x01;

// start timing

// start the timer to count in the 1024 pre-scale

// (set CS2, CS1, CS0 to 101 (XXXX X101) )

\*tcc1B = 0x05;

// let TCNT1 count up to max (that is, TOV1 flag gets set)

while( (\*tc1Flag & 0x01) == 0x00 )

{}

// clean up - stop counter

// set the clock select bits for clock stopping

// i.e. CS02, CS01, CS00 to (000)

\*tcc1B = 0; // (1111 1000)

// clear the TOV1 flag

\*tc1Flag |= 0x01;

// no return - void

}

1. Write an Arduino Mega C language program using the Arduino ATmega2560 timer1 in Normal mode to generate a 16 kHz square wave on PortB.6 using Timer 1.

// This function does utilize “MyMicroDelay(unsigned long int uSeconds)” which is

// a version of MyDelay that uses a different pre-scale to be able to time

// microsecond delays

void genRectangleWave( unsigned long int frequencyHz, unsigned int dutyCycle );

void loop()

{

// variables

// none needed, all vars in functions

// generate a 16kHz wave with a 50% duty cycle

genRectangleWave( 16000, 50 );

// no return – void – call loop again from main

}

void genRectangleWave( unsigned long int frequencyHz, unsigned int dutyCycle )

{

// variables

volatile unsigned char\* portB = (unsigned char\*) 0x25; // for B7

\*portB &= 0x7F; // turn it off

volatile unsigned long int period = 0;

volatile unsigned long int hiDelay = 0;

volatile unsigned long int loDelay = 0;

volatile unsigned long int counter = 0;

// find the period of the wave

// if the wave vibrates at frequency x, the period is x^-1

// then x1000000 to convert to useconds conversion

period = (unsigned long int)( 1000000.0 / frequencyHz );

// scale the high and low periods according to the duty cycle

hiDelay = (unsigned long int) ((dutyCycle / 100) \* period);

loDelay = (unsigned long int) (((100 - dutyCycle) / 100) \* period);

// I want the wave to run for 3 seconds, find out how many wave periods per second

counter = (unsigned long int) (3000000 / period); // x1000000 to adjust for useconds

// generate wave indefinitely (until we learn escape)

for( ; counter > 0; counter-- )

{

// generate wave indefinitely (until we learn escape)

// turn output high

\*portB |= 0x40;

// maintain for the period of the wave

MyMicroDelay( hiDelay );

// turn output low

\*portB &= 0xBF;

// maintain for off period of wave

MyMicroDelay( loDelay );

}

// no return - void

}

1. The international tuning standard for musical instruments is “A” above middle C” at a frequency of 440Hz. Write an Arduino Mega C language program to generate this tuning frequency and sound a 440 Hz tone on a loudspeaker connected to PortB.6 using Timer 1.

// uses same function defined in #3

void loop()

{

// variables

// none needed, all vars in functions

// generate a 440Hz wave with a 50% duty cycle

genRectangleWave( 16000, 50 );

// no return – void – call loop again from main

}

1. Write an Arduino Mega C language program to generate a 500Hz signal on PortB.6 using Timer 1 in Normal mode. The wave should have a 30% duty cycle (duty cycle = high time / period).

// uses same function defined in #3

void loop()

{

// variables

// none needed, all vars in functions

// generate a 500Hz wave with a 30% duty cycle

genRectangleWave( 500, 30 );

// no return – void – call loop again from main

}

1. [Extra Credit] Write an Arduino Mega C language function using Timer 1 in Normal mode to open and close a digital camera shutter. Assume when the shutter release is pressed, the shutter speed is passed to your function and that PortB.7 controls the camera shutter. When PortB.7 is 0 the shutter is closed and when PortB.7 is 1 the shutter is open. Assume an initialization program has initialized and cleared PortB.7.

Typical shutter speeds are expressed in fractions of a second. Create a table for your function to use to create shutter speeds of: 1, 1/2, 1/4, 1/8, 1/15. 1/30, 1/60, 1/125, 1/250, 1/500, 1/1000. The values passed corresponding to these shutter speeds are 0 through 10. That is, if a 0 is passed the shutter speed is to be 1 second and if a 1 is passed the shutter speed is to be one half second, etc.

void openShutter( unsigned int speedCode );

void openShutter( unsigned int speedCode )

{

// variables

volatile unsigned char\* portB = (unsigned char\*) 0x25; // for B7

\*portB &= 0x7F; // ensure shutter is closed

volatile unsigned long int openTime = 0; // in milliseconds

// determine how long to open the shutter based on the chosen code

switch( speedCode )

{

// shutter speed 0

case 0:

// we have decided in this case that the shutter will be open

// for 1 second. Find the period in milliseconds

openTime = 1000;

break;

// shutter speed 1

case 1:

// we have decided in this case that the shutter will be open

// for 0.5 second. Find the period in milliseconds

openTime = 500;

break;

// shutter speed 2

case 2:

// we have decided in this case that the shutter will be open

// for 0.25 second. Find the period in milliseconds

openTime = 250;

break;

// shutter speed 3

case 3:

// we have decided in this case that the shutter will be open

// for 0.125 second. Find the period in milliseconds

openTime = 125;

break;

// shutter speed 4

case 4:

// we have decided in this case that the shutter will be open

// for 1/15 (o.067) second. Find the period in milliseconds

openTime = 67;

break;

// shutter speed 5

case 5:

// we have decided in this case that the shutter will be open

// for 1/30 (0.0333) second. Find the period in milliseconds

openTime = 33;

break;

// shutter speed 6

case 6:

// we have decided in this case that the shutter will be open

// for 1/60 (0.0167) second. Find the period in milliseconds

openTime = 17;

break;

// shutter speed 7

case 7:

// we have decided in this case that the shutter will be open

// for 1/125 (0.008) second. Find the period in milliseconds

openTime = 8;

break;

// shutter speed 8

case 8:

// we have decided in this case that the shutter will be open

// for 1/250 second. Find the period in milliseconds

openTime = 4;

break;

// shutter speed 9

case 9:

// we have decided in this case that the shutter will be open

// for 1/500 second. Find the period in milliseconds

openTime = 2;

break;

// shutter speed 10

case 10:

// we have decided in this case that the shutter will be open

// for 1/1000 second. Find the period in milliseconds

openTime = 1;

break;

}

// open the shutter

\*portB |= 0x80;

// leave shutter open for specified time

MyDelay( openTime );

// close the shutter

\*portB &= 0x7F;

// no return - void

}