Arduino Timer Interrupts

by **amandaghassaei** on June 26, 2012

Table of Contents

Arduino Timer Interrupts 1
Intro: Arduino Timer Interrupts 2
Step 1: Prescalers and the Compare Match Register 2
Step 2: Structuring Timer Interrupts 3
Step 3: Example 1: Bike Speedometer 6
Step 4: Example 2: Serial Communication 8
File Downloads 10
Related Instructables 11


Author:amandaghassaei amandaghassaei.com
Currently working for instructables!

Intro: Arduino Timer Interrupts

Timer interrupts allow you to perform a task at very specifically timed intervals regardless of what else is going on in your code. In this instructable I'll explain how to setup and execute an interrupt in Clear Timer on Compare Match or CTC Mode. Jump straight to step 2 if you are looking for sample code.

Normally when you write an Arduino sketch the Arduino performs all the commands encapsulated in the loop() {} function in the order that they are written, however, it's difficult to time events in the loop(). Some commands take longer than others to execute, some depend on conditional statements (if, while...) and some Arduino library functions (like digitalWrite or analogRead) are made up of many commands. Arduino timer interrupts allow you to momentarily pause the normal sequence of events taking place in the loop() function at precisely timed intervals, while you execute a separate set of commands. Once these commands are done the Arduino picks up again where it was in the loop().

Interrupts are useful for:

Measuring an incoming signal at equally spaced intervals (constant sampling frequency)
Calculating the time between two events
Sending out a signal of a specific frequency
Periodically checking for incoming serial data

There are a few ways to do interrupts, for now I'll focus on the type that I find the most useful/flexible, called Clear Timer on Compare Match or CTC Mode. Additionally, in this instructable I'll be writing specifically about the timers to the Arduino Uno (and any other Arduino with ATMEL 328/168... Lilypad, Duemilanove, Diecimila, Nano...). The main ideas presented here apply to the Mega and older boards as well, but the setup is a little different and the table above is specific to ATMEL 328/168.

Step 1: Prescalers and the Compare Match Register

The Uno has three timers called timer0, timer1, and timer2. Each of the timers has a counter that is incremented on each tick of the timer's clock. CTC timer interrupts are triggered when the counter reaches a specified value, stored in the compare match register. Once a timer counter reaches this value it will clear (reset to zero) on the next tick of the timer's clock, then it will continue to count up to the compare match value again. By choosing the compare match value and setting the speed at which the timer increments the counter, you can control the frequency of timer interrupts.

The first parameter I'll discuss is the speed at which the timer increments the counter. The Arduino clock runs at 16MHz, this is the fastest speed that the timers can increment their counters. At 16MHz each tick of the counter represents 1/16,000,000 of a second (~63ns), so a counter will take 10/16,000,000 seconds to reach a value of 9 (counters are 0 indexed), and 100/16,000,000 seconds to reach a value of 99.

In many situations, you will find that setting the counter speed to 16MHz is too fast. Timer0 and timer2 are 8 bit timers, meaning they can store a maximum counter value of 255. Timer1 is a 16 bit timer, meaning it can store a maximum counter value of 65535. Once a counter reaches its maximum, it will tick back to zero (this is called overflow). This means at 16MHz, even if we set the compare match register to the max counter value, interrupts will occur every 256/16,000,000 seconds (~16us) for the 8 bit counters, and every 65,536/16,000,000 (~4 ms) seconds for the 16 bit counter. Clearly, this is not very useful if you only want to interrupt once a second.

Instead you can control the speed of the timer counter incrementation by using something called a prescaler. A prescaler dictates the speed of your timer according the the following equation:

(timer speed (Hz)) = (Arduino clock speed (16MHz)) / prescaler

So a 1 prescaler will increment the counter at 16MHz, an 8 prescaler will increment it at 2MHz, a 64 prescaler = 250kHz, and so on. As indicated in the table above, the prescaler can equal 1, 8, 64, 256, and 1024. (I'll explain the meaning of CS12, CS11, and CS10 in the next step.)

Now you can calculate the interrupt frequency with the following equation:

interrupt frequency (Hz) = (Arduino clock speed 16,000,000Hz) / (prescaler * (compare match register + 1))
the +1 is in there because the compare match register is zero indexed

rearranging the equation above, you can solve for the compare match register value that will give your desired interrupt frequency:

compare match register = [16,000,000Hz/ (prescaler * desired interrupt frequency)] - 1
remember that when you use timers 0 and 2 this number must be less than 256, and less than 65536 for timer1

so if you wanted an interrupt every second (frequency of 1Hz):
compare match register = [16,000,000 / (prescaler * 1)] -1
with a prescaler of 1024 you get:
compare match register = [16,000,000 / (1024 * 1)] -1
= 15,624
since 256 < 15,624 < 65,536, you must use timer1 for this interrupt.

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Image Notes
1. prescaler setup

Step 2: Structuring Timer Interrupts

Timer setup code is done inside the setup(){} function in an Arduino sketch.

The code involved for setting up timer interrupts is a little daunting to look at, but it's actually not that hard. I pretty much just copy the same main chunk of code and change the prescaler and compare match register to set the correct interrupt frequency.

The main structure of the interrupt setup looks like this:

```
void setup(){

cli();//stop interrupts

//set timer0 interrupt at 2kHz
 TCCR0A = 0;// set entire TCCR2A register to 0
 TCCR0B = 0;// same for TCCR2B
 TCNT0  = 0;//initialize counter value to 0
 // set compare match register for 2khz increments
 OCR0A = 124;// = (16*10^6) / (2000*64) - 1 (must be <256)
 // turn on CTC mode
 TCCR0A |= (1 << WGM01);
 // Set CS11 bit for 64 prescaler
 TCCR0B |= (1 << CS11) | (1 << CS10);
 // enable timer compare interrupt
 TIMSK0 |= (1 << OCIE0A);

//set timer1 interrupt at 1Hz
 TCCR1A = 0;// set entire TCCR1A register to 0
 TCCR1B = 0;// same for TCCR1B
 TCNT1  = 0;//initialize counter value to 0
 // set compare match register for 1hz increments
 OCR1A = 15624;// = (16*10^6) / (1*1024) - 1 (must be <65536)
 // turn on CTC mode
 TCCR1B |= (1 << WGM12);
 // Set CS11 bit for 1024 prescaler
 TCCR1B |= (1 << CS12) | (1 << CS10);
 // enable timer compare interrupt
 TIMSK1 |= (1 << OCIE1A);

//set timer2 interrupt at 8kHz
 TCCR2A = 0;// set entire TCCR2A register to 0
 TCCR2B = 0;// same for TCCR2B
 TCNT2  = 0;//initialize counter value to 0
 // set compare match register for 8khz increments
 OCR2A = 249;// = (16*10^6) / (8000*8) - 1 (must be <256)
 // turn on CTC mode
 TCCR2A |= (1 << WGM21);
 // Set CS11 bit for 8 prescaler
 TCCR2B |= (1 << CS11);
 // enable timer compare interrupt
 TIMSK2 |= (1 << OCIE2A);


sei();//allow interrupts

}//end setup
```

Notice how the value of OCR#A (the compare match value) changes for each of these timer setups. As explained in the last step, this was calculated according to the

following equation:

compare match register = [16,000,000Hz/ (prescaler * desired interrupt frequency)] - 1
remember that when you use timers 0 and 2 this number must be less than 256, and less than 65536 for timer1

Also notice how the setups between the three timers differ slightly in the line which turns on CTC mode:
TCCR0A |= (1 << WGM01);//for timer0
TCCR1B |= (1 << WGM12);//for timer1
TCCR2A |= (1 << WGM21);//for timer2
This follows directly from the datasheet of the ATMEL 328/168.

Finally, notice how the setup for the prescalers follows the table above (fig 4).
TCCR0B |= (1 << CS11) | (1 << CS10); // Set CS11 bit for 64 prescaler
TCCR2B |= (1 << CS11); // Set CS11 bit for 8 prescaler
TCCR1B |= (1 << CS12) | (1 << CS10); // Set CS11 bit for 1024 prescaler

The commands you want to execute during these timer interrupts are located in the Arduino sketch encapsulated in the following:
ISR(TIMER0_COMPA_vect){ //change the 0 to 1 for timer1 and 2 for timer2
//interrupt commands here
}
This bit of code should be located outside the setup() and loop() functions. Also, try to keep the interrupt routine as short as possible, especially if you are interrupting at a high frequency.

Example- the following sketch sets up and executes 3 timer interrupts:

```
//timer interrupts
//by Amanda Ghassaei
//June 2012

/*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 3 of the License, or
* (at your option) any later version.
*
*/

//timer setup for timer0, timer1, and timer2.
//For arduino uno or any board with ATMEL 328/168.. diecimila, duemilanove, lilypad, nano, mini...

//this code will enable all three arduino timer interrupts.
//timer0 will interrupt at 2kHz
//timer1 will interrupt at 1Hz
//timer2 will interrupt at 8kHz

//storage variables
boolean toggle0 = 0;
boolean toggle1 = 0;
boolean toggle2 = 0;

void setup(){

 //set pins as outputs
 pinMode(8, OUTPUT);
 pinMode(9, OUTPUT);
 pinMode(13, OUTPUT);

cli();//stop interrupts

//set timer0 interrupt at 2kHz
 TCCR0A = 0;// set entire TCCR2A register to 0
 TCCR0B = 0;// same for TCCR2B
 TCNT0  = 0;//initialize counter value to 0
 // set compare match register for 2khz increments
 OCR0A = 124;// = (16*10^6) / (2000*64) - 1 (must be <256)
 // turn on CTC mode
 TCCR0A |= (1 << WGM01);
 // Set CS11 bit for 64 prescaler
 TCCR0B |= (1 << CS11) | (1 << CS10);
 // enable timer compare interrupt
 TIMSK0 |= (1 << OCIE0A);

//set timer1 interrupt at 1Hz
 TCCR1A = 0;// set entire TCCR1A register to 0
 TCCR1B = 0;// same for TCCR1B
 TCNT1  = 0;//initialize counter value to 0
 // set compare match register for 1hz increments
 OCR1A = 15624;// = (16*10^6) / (1*1024) - 1 (must be <65536)
 // turn on CTC mode
 TCCR1B |= (1 << WGM12);
 // Set CS11 bit for 1024 prescaler
 TCCR1B |= (1 << CS12) | (1 << CS10);
 // enable timer compare interrupt
 TIMSK1 |= (1 << OCIE1A);

//set timer2 interrupt at 8kHz
 TCCR2A = 0;// set entire TCCR2A register to 0
 TCCR2B = 0;// same for TCCR2B
 TCNT2  = 0;//initialize counter value to 0
 // set compare match register for 8khz increments
 OCR2A = 249;// = (16*10^6) / (8000*8) - 1 (must be <256)
 // turn on CTC mode
 TCCR2A |= (1 << WGM21);
 // Set CS11 bit for 8 prescaler
 TCCR2B |= (1 << CS11);
```

```
 // enable timer compare interrupt
 TIMSK2 |= (1 << OCIE2A);


sei();//allow interrupts

}//end setup

ISR(TIMER0_COMPA_vect){//timer0 interrupt 2kHz toggles pin 8
//generates pulse wave of frequency 2kHz/2 = 1kHz (takes two cycles for full wave- toggle high then toggle low)
 if (toggle0){
   digitalWrite(8,HIGH);
   toggle0 = 0;
 }
 else{
   digitalWrite(8,LOW);
   toggle0 = 1;
 }
}

ISR(TIMER1_COMPA_vect){//timer1 interrupt 1Hz toggles pin 13 (LED)
//generates pulse wave of frequency 1Hz/2 = 0.5kHz (takes two cycles for full wave- toggle high then toggle low)
 if (toggle1){
   digitalWrite(13,HIGH);
   toggle1 = 0;
 }
 else{
   digitalWrite(13,LOW);
   toggle1 = 1;
 }
}

ISR(TIMER2_COMPA_vect){//timer1 interrupt 8kHz toggles pin 9
//generates pulse wave of frequency 8kHz/2 = 4kHz (takes two cycles for full wave- toggle high then toggle low)
 if (toggle2){
   digitalWrite(9,HIGH);
   toggle2 = 0;
 }
 else{
   digitalWrite(9,LOW);
   toggle2 = 1;
 }
}


void loop(){
 //do other things here
}
```

The images above show the outputs from these timer interrupts. Fig 1 shows a square wave oscillating between 0 and 5V at 1kHz (timer0 interrupt), fig 2 shows the LED attached to pin 13 turning on for one second then turning off for one second (timer1 interrupt), fig 3 shows a pulse wave oscillating between 0 and 5V at a frequency of 4khz (timer2 interrupt).

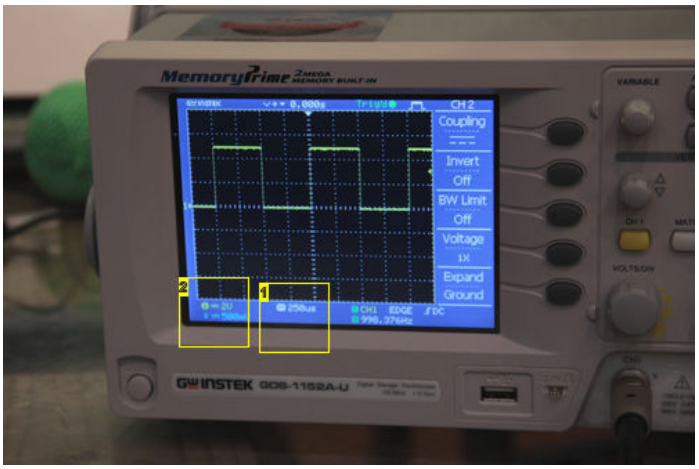


Image Notes
1. time/div = 250us
2. volts/div = 2V

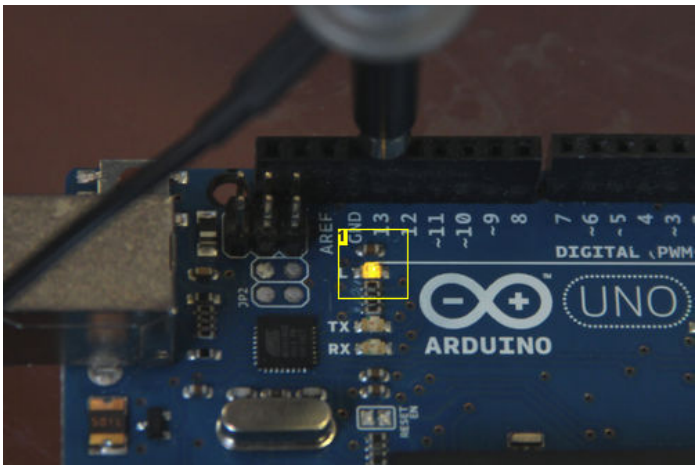


Image Notes
1. led light up for 1 sec then turns of for 1 sec

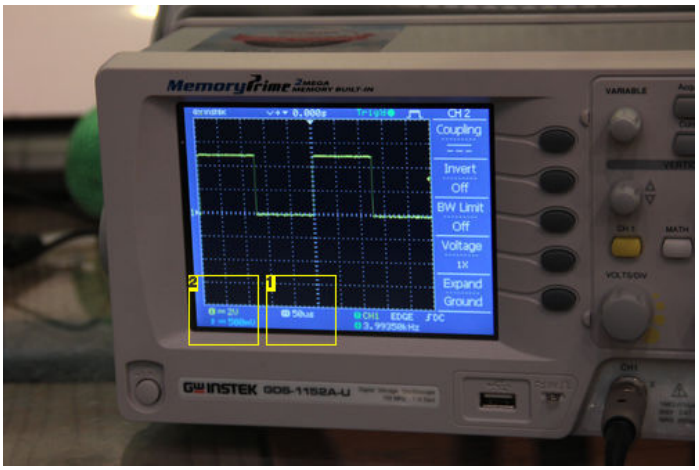


Image Notes
1. time/div = 50us
2. volts/div = 2v

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Image Notes
1. prescaler setup

Step 3: Example 1: Bike Speedometer

In this example I made an arduino powered bike speedometer (posting full project soon). It works by attaching a magnet to the wheel and measuring the amount of time it takes to pass by a magnetic switch mounted on the frame- the time for one complete rotation of the wheel.

I set timer 1 to interrupt every ms (frequency of 1kHz) to measure the magnetic switch. If the magnet is passing by the switch, the signal from the switch is high and the variable "time" gets set to zero. If the magnet is not near the switch "time" gets incremented by 1. This way "time" is actually just a measurement of the amount of time in milliseconds that has passed since the magnet last passed by the magnetic switch. This info is used later in the code to calculate rpm and mph of the bike.

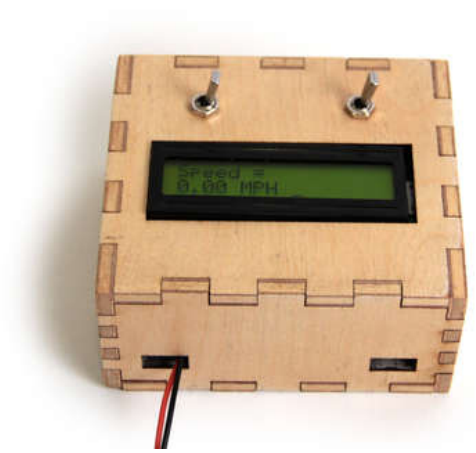
Here's the bit of code that sets up timer1 for 1kHz interrupts

```
cli();//stop interrupts
//set timer1 interrupt at 1kHz
TCCR1A = 0;// set entire TCCR1A register to 0
TCCR1B = 0;// same for TCCR1B
TCNT1 = 0;//initialize counter value to 0
// set timer count for 1khz increments
OCR1A = 1999;// = (16*10^6) / (1000*8) - 1
//had to use 16 bit timer1 for this bc 1999>255, but could switch to timers 0 or 2 with larger prescaler
// turn on CTC mode
TCCR1B |= (1 << WGM12);
// Set CS11 bit for 8 prescaler
TCCR1B |= (1 << CS11);
// enable timer compare interrupt
TIMSK1 |= (1 << OCIE1A);
sei();//allow interrupts
```

Here's the complete code if you want to take a look:

```
//bike speedometer
//by Amanda Ghassaei 2012

/*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 3 of the License, or
* (at your option) any later version.
*
*/

//sample calculations
//tire radius ~ 13.5 inches
//circumference = pi*2*r =~85 inches
//max speed of 35mph =~ 616inches/second
//max rps =~7.25

#define reed A0//pin connected to read switch

//storage variables
float radius = 13.5;// tire radius (in inches)- CHANGE THIS FOR YOUR OWN BIKE

int reedVal;
long time = 0;// time between one full rotation (in ms)
float mph = 0.00;
float circumference;
boolean backlight;

int threshold = 700;//threshold to determine if switch is on/off
int maxReedCounter = 100;//min time (in ms) of one rotation (for debouncing)
int reedCounter;


void setup(){

 reedCounter = maxReedCounter;
```

```
 circumference = 2*3.14*radius;
 pinMode(1,OUTPUT);//tx
 pinMode(2,OUTPUT);//backlight switch

 checkBacklight();

 Serial.write(12);//clear

 // TIMER SETUP- the timer interrupt allows preceise timed measurements of the reed switch
 //for mor info about configuration of arduino timers see http://arduino.cc/playground/Code/Timer1
 cli();//stop interrupts

 //set timer1 interrupt at 1kHz
 TCCR1A = 0;// set entire TCCR1A register to 0
 TCCR1B = 0;// same for TCCR1B
 TCNT1  = 0;//initialize counter value to 0;
 // set timer count for 1khz increments
 OCR1A = 1999;// = (16*10^6) / (1000*8) - 1
 // turn on CTC mode
 TCCR1B |= (1 << WGM12);
 // Set CS11 bit for 8 prescaler
 TCCR1B |= (1 << CS11);
 // enable timer compare interrupt
 TIMSK1 |= (1 << OCIE1A);

 sei();//allow interrupts
 //END TIMER SETUP

 Serial.begin(9600);
}

void checkBacklight(){
 backlight = digitalRead(2);
 if (backlight){
   Serial.write(17);//turn backlight on
 }
 else{
   Serial.write(18);//turn backlight off
 }
}

ISR(TIMER1_COMPA_vect) {//Interrupt at freq of 1kHz to measure reed switch
 reedVal = analogRead(reed);//get val of A0
 if (reedVal>threshold){//if reed switch is closed
   if (reedCounter == 0){//min time between pulses has passed
     mph = (56.8*float(circumference))/float(time);//calculate miles per hour
     time = 0;//reset timer
     reedCounter = maxReedCounter;//reset reedCounter
   }
   else{
     if (reedCounter > 0){//don't let reedCounter go negative
       reedCounter -= 1;//decrement reedCounter
     }
   }
 }
 else{//if reed switch is open
   if (reedCounter > 0){//don't let reedCounter go negative
     reedCounter -= 1;//decrement reedCounter
   }
 }
 if (time > 2000){
   mph = 0;//if no new pulses from reed switch- tire is still, set mph to 0
 }
 else{
   time += 1;//increment timer
 }
}

void displayMPH(){
 Serial.write(12);//clear
 Serial.write("Speed =");
 Serial.write(13);//start a new line
 Serial.print(mph);
 Serial.write(" MPH ");
 //Serial.write("0.00 MPH ");
}

void loop(){
 //print mph once a second
 displayMPH();
 delay(1000);
 checkBacklight();
}
```



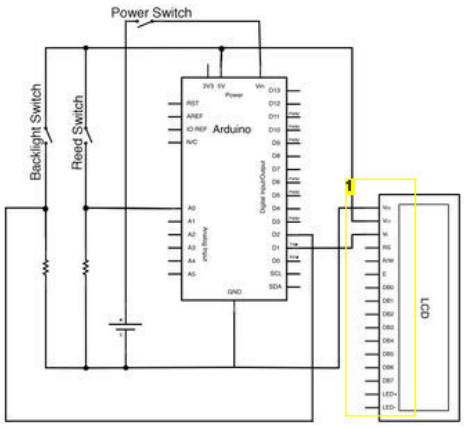


Image Notes
1. ignore the pin labels on the LCD

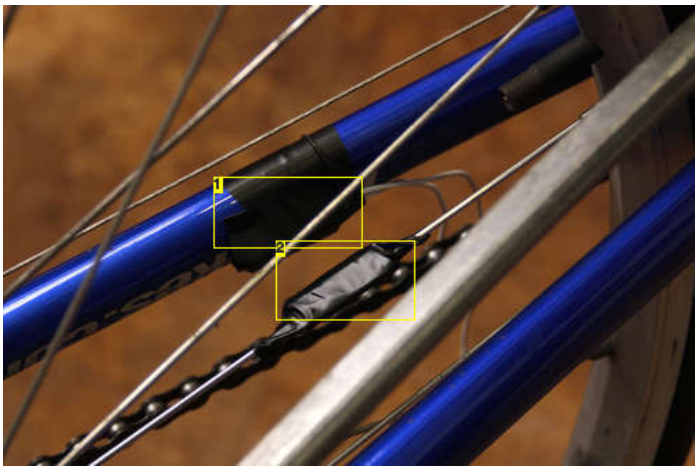


Image Notes
1. switch attached to bike frame
2. magnet attached to spoke

Step 4: Example 2: Serial Communication

This project is a 4x4 backlit button pad. The project connects to my computer via usb, it sends information about the buttons to the computer and receives information about how to light up the LEDs. Here is a video:

For this project, I used timer2 interrupts to periodically check if there was any incoming serial data, read it, and store it in the matrix "ledData[]". If you take a look at the code you will see that the main loop of the sketch is what is actually responsible for using the info in ledData to light up the correct LEDs and checking on the status of the buttons (a function called "shift()"). The interrupt routine is as short as possible- just checking for incoming bytes and storing them appropriately.

Here is the setup for timer2:

```
cli();//stop interrupts
//set timer2 interrupt every 128us
TCCR2A = 0;// set entire TCCR2A register to 0
TCCR2B = 0;// same for TCCR2B
TCNT2 = 0;//initialize counter value to 0
// set compare match register for 7.8khz increments
OCR2A = 255;// = (16*10^6) / (7812.5*8) - 1 (must be <256)
// turn on CTC mode
TCCR2A |= (1 << WGM21);
// Set CS11 bit for 8 prescaler
TCCR2B |= (1 << CS11);
// enable timer compare interrupt
TIMSK2 |= (1 << OCIE2A);
sei();//allow interrupts
```

Here's the complete Arduino sketch:

```
//BUTTON TEST w/ 74HC595 and 74HC165 and serial communication
//by Amanda Ghassaei
//June 2012
```

```
/*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
*/

//this firmware will send data back and forth with the maxmsp patch "beat slicer"

//pin connections
#define ledLatchPin A1
#define ledClockPin A0
#define ledDataPin A2
#define buttonLatchPin 9
#define buttonClockPin 10
#define buttonDataPin A3

//looping variables
byte i;
byte j;
byte k;
byte ledByte;

//storage for led states, 4 bytes
byte ledData[] = {0, 0, 0, 0};
//storage for buttons, 4 bytes
byte buttonCurrent[] = {0,0,0,0};
byte buttonLast[] = {0,0,0,0};
byte buttonEvent[] = {0,0,0,0};
byte buttonState[] = {0,0,0,0};
//button debounce counter- 16 bytes
byte buttonDebounceCounter[4][4];


void setup() {
 DDRC = 0xF7;//set A0-2 and A4-5 output, A3 input
 DDRB = 0xFF;//digital pins 8-13 output

 Serial.begin(57600);

 cli();//stop interrupts

 //set timer2 interrupt every 128us
 TCCR2A = 0;// set entire TCCR2A register to 0
 TCCR2B = 0;// same for TCCR2B
 TCNT2  = 0;//initialize counter value to 0
 // set compare match register for 7.8khz increments
 OCR2A = 255;// = (16*10^6) / (7812.5*8) - 1 (must be <256)
 // turn on CTC mode
 TCCR2A |= (1 << WGM21);
 // Set CS11 bit for 8 prescaler
 TCCR2B |= (1 << CS11);
 // enable timer compare interrupt
 TIMSK2 |= (1 << OCIE2A);

 sei();//allow interrupts

}

// buttonCheck - checks the state of a given button.
//this buttoncheck function is largely copied from the monome 40h firmware by brian crabtree and joe lake
void buttonCheck(byte row, byte index)
{
 if (((buttonCurrent[row] ^ buttonLast[row]) &(1 << index)) &&   // if the current physical button state is different from the
 ((buttonCurrent[row] ^ buttonState[row]) &(1 << index))) {  // last physical button state AND the current debounced state

   if (buttonCurrent[row] &(1 << index)) {                     // if the current physical button state is depressed
     buttonEvent[row] = 1 << index;              // queue up a new button event immediately
     buttonState[row] |= (1 << index);                         // and set the debounced state to down.
 }
   else{
     buttonDebounceCounter[row][index] = 12;
   }  // otherwise the button was previously depressed and now
   // has been released so we set our debounce counter.
 }
 else if (((buttonCurrent[row] ^ buttonLast[row]) &(1 << index)) == 0 &&  // if the current physical button state is the same as
 (buttonCurrent[row] ^ buttonState[row]) &(1 << index)) {        // the last physical button state but the current physical
   // button state is different from the current debounce
   // state...
   if (buttonDebounceCounter[row][index] > 0 && --buttonDebounceCounter[row][index] == 0) {  // if the the debounce counter has
     // been decremented to 0 (meaning the
     // the button has been up for
     // kButtonUpDefaultDebounceCount
     // iterations///

     buttonEvent[row] = 1 << index;    // queue up a button state change event

     if (buttonCurrent[row] &(1 << index)){          // and toggle the buttons debounce state.
       buttonState[row] |= (1 << index);
     }
     else{
       buttonState[row] & ~(1 << index);
     }
   }
 }
}


void shift(){
```

```
 for (i=0;i<4;i++){

   buttonLast[i] = buttonCurrent[i];

   byte dataToSend = (1 << (i+4)) | (15 &~ledData[i]);

   // set latch pin low so the LEDs don't change while sending in bits
   digitalWrite(ledLatchPin, LOW);
   // shift out the bits of dataToSend
   shiftOut(ledDataPin, ledClockPin, LSBFIRST, dataToSend);
   //set latch pin high so the LEDs will receive new data
   digitalWrite(ledLatchPin, HIGH);

   //once one row has been set high, receive data from buttons
   //set latch pin high
   digitalWrite(buttonLatchPin, HIGH);
   //shift in data
   buttonCurrent[i] = shiftIn(buttonDataPin, buttonClockPin, LSBFIRST) >> 3;
   //latchpin low
   digitalWrite(buttonLatchPin, LOW);

   for (k=0;k<4;k++){
     buttonCheck(i,k);
     if (buttonEvent[i]<<k){
       if (buttonState[i]&<<k){
         Serial.write(((3-k)<<3)+(i<<1)+1);
       }
       else{
         Serial.write(((3-k)<<3)+(i<<1)+0);
       }
       buttonEvent[i] & ~(1<<k);
     }
   }
 }
}


ISR(TIMER2_COMPA_vect) {
 do{
   if (Serial.available()){
     ledByte = Serial.read();//000xxyys
     boolean ledstate = ledByte &1;
     byte ledy = (ledByte >> 1) &3;
     byte ledx = (ledByte >> 3) &3;
     if (ledstate){
       ledData[ledy] |= 8 >> ledx;
     }
     else{
       ledData[ledy] & ~ (8 >> ledx);
     }
   }//end if serial available
 }//end do
while (Serial.available() > 8);
}


void loop() {
 shift();//updates leds and receives data from buttons
}
```

download the MaxMSP patch below (it will run in Max Runtime as well).

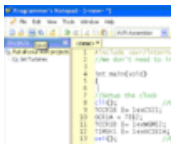
File Downloads

beat slicer.zip (15 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'beat slicer.zip']

Related Instructables

Girino - Fast Arduino Oscilloscope by Caffeinomane

How to build an Air Guitar with Arduino, aka the AIRduino Guitar by Jell



Start-up Instructions for Programming Microcontrollers on a PC with WinAVR using an Atmega 328p Chip and USBTiny ISP Programmer for Examples by deyb1

Easy PIC micro state transition Interrupt code by leevonk

Debugging AVR code in Linux with simavr by hardwarehank

USB controlled home automation hack by chr