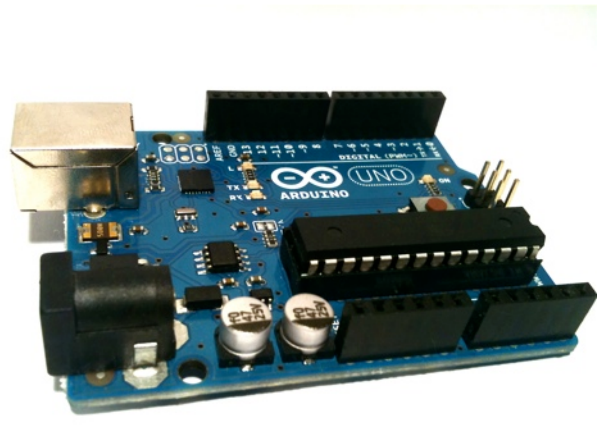

BASIC WATCHDOG TIMER
(Arduino UNO/ATmega328)
by Nicolas Larsen

There seems to be a lot confusion regarding the setup of the watchdog timer which is what I hope to clarify in a very basic manner. I will try my best to explain things in clear english. This is only applicable to boards running the Opti-bootloader (standard Uno bootloader) and those running Lady Ada's bootloader (although I have yet to test it on the latter). Using Arduino IDE 0022

The watchdog timer is a very useful feature for projects that are intended to run for extended periods of time or contain unstable loops of code. The watchdog is essentially a small timer that will force a full system restart should it not receive a "system ok" signal from the microcontroller within a preset time. Should in any application the micro-controller freeze, hang, stall or crash the watchdog will time out and force a reset to the same effect as pressing the reset button on your board.

Begin by importing the watchdog timer into your code.

```
#include <avr/wdt.h>
```

You'll now need to configure your watchdog timer through one of the **registers** known as **WDTCSR**. A register is a byte or several in the system's memory that is reserved for saving system settings (in lay-mans terms). Every feature will have a register allocated to it. The microcontroller knows where to go in its memory to get these registers. The watchdog is no different. The watchdog register, composed of one byte, thus has 8 bits in on/off settings. A '1' indicates 'ON' and a '0' indicates 'OFF'. By default everything is set to 'OFF' so we only need to set the relevant bits to 'ON'. To the right is a table of the different bits and their function.

Bit	Name
7	WDIF
6	WDIE
5	WDP3
4	WDCE
3	WDE
2	WDP2
1	WDP1
0	WDP0

WDIF - Sets an interrupt flag but you wont need to worry about this. It is automatically flagged high and low by the system.

WDIE - Enables Interrupts. This will give you the chance to include one last dying wish (or a few lines of code...) before the board is reset. This is a great way of performing interrupts on a regular interval should the watchdog be configured to not reset on time-out.

WDCE - This is a safety to enable a configuration mode that will last 4 clock cycles. Set this bit and WDE high before attempting any changes to the watchdog register. *There isn't really any logic behind this, you just have to set WDCE and WDE to '1' to enter a sort of 'setup mode' in the watchdog timer.*

WDE - Enables system reset on time-out. Whenever the Watchdog timer times out the micocontroller will be reset. This is probably what you were all looking for. Set this to '1' to activate.

WDP0/WDP1/WDP2/WDP3 - These four bits determine how long the timer will count for before resetting. The exact time is set by setting combinations of the 4 bits in such a pattern.

WDP 3	WDP 2	WDP 1	WDP 0	Time-out (ms)
0	0	0	0	16
0	0	0	1	32
0	0	1	0	64
0	0	1	1	125
0	1	0	0	250
0	1	0	1	500
0	1	1	0	1000
0	1	1	1	2000
1	0	0	0	4000
1	0	0	1	8000

So you can see the watchdog can be set anywhere between 16ms and 8 seconds. To let the watchdog timer know that everything is running ok and that it needn't panic or take any action your going to have to keep reseting the timer within your main loop. This is done by periodically entering in:

```
wdt_reset();
```

Remember: the watchdog is a timer, if you don't reset it regularly it will time-out, prompting the reset and or interrupt.

Now you know the basic setting we are going to create a function that sets up the watchdog timer to include interrupts, a reset and time-out after 1000ms. It is good practice to comment in what you are setting each of the bits to for later reference.

```
void watchdogSetup(void)
{
  cli();

  wdt_reset();

/*
   WDTCSR configuration:
   WDIE  = 1: Interrupt Enable
   WDE   = 1 :Reset Enable
   See table for time-out variations:
   WDP3 = 0 :For 1000ms Time-out
   WDP2 = 1 :For 1000ms Time-out
   WDP1 = 1 :For 1000ms Time-out
   WDP0 = 0 :For 1000ms Time-out
*/

// Enter Watchdog Configuration mode:
WDTCSR |= (1<<WDCE) | (1<<WDE);

// Set Watchdog settings:
 WDTCSR = (1<<WDIE) | (1<<WDE) |
(0<<WDP3)  | (1<<WDP2) | (1<<WDP1)  |
(0<<WDP0);

sei();
}
```

Lets cover this line by line. **"cli();"** disables all interrupts on the microcontroller so that configuration is never disrupted and left unfinished.

"wdt_reset();" resets the watchdog timer. This isn't always necessary but you certainly don't want your watchdog timing out and resetting while you are trying to set it.

```
// Enter Watchdog Configuration mode:
WDTCSR |= (1<<WDCE) | (1<<WDE);
```

The above will prompt configuration mode enabling you to make changes to the register. These changes need to be set within 4-cycles so the settings should follow immediately after this line. It is unusual for registers to require a configuration like this which may account for all the watchdog timer confusion. Regardless, its in the ATmega328 data sheet so thats what we must all abide by.

Now we enter in the various bits as we laid out in the comments. It is not necessary to set anything to zero as I have done for WDP3/0, as by default everything in the register is already at zero. I included that so you can quickly modify the time-settings should you wish to make changes.

```
// Set Watchdog settings:
 WDTCSR = (1<<WDIE) | (1<<WDE) |
(0<<WDP3)  | (1<<WDP2) | (1<<WDP1)  |
(0<<WDP0);
```

If you are unfamiliar with bitwise and compound operations the syntax may appear a bit odd to you. I suggest you have a look on the **Arduino Reference Page** for better understanding. The same settings could just have easily been applied in binary by:

```
// Enter Watchdog Configuration mode:
WDTCSR |= B00011000;
// Set Watchdog settings:
 WDTCSR = B01001110;
```

Now that your done with the setup you can re-enable interrupts by typing **"sei()"**. The watchdog includes interrupts so be sure to re-enable this.

The only thing left to insert in your code is the interrupt, should you have chosen to include it. Note that the interrupt, like all interrupts cannot make use of the **"delay()"** function. It is also poor practice to call functions such as **"Serial.println()"** in the interrupt as any error in the function may cause the microcontroller to hang in the interrupt preventing the watchdog from restarting.

```
ISR(WDT_vect)
{
// Include your interrupt code here.
}
```

The example code included on the next page sets the watchdog timer to 2 seconds with an interrupt. An expanding loop is included so that the watchdog will eventually time-out and cause a reset. What you will hopefully notice from the serial feed is that the watchdog is not reset exactly after 2 seconds. The watchdog is a very inaccurate timer and as such should not be used for time critical applications. With that said - nor is my 'timing' code strictly accurate...

I hope this clarifies the basic setup of the watchdog timer and some of its features. It is certainly not an exhaustive list of features! If you find any errors please let me know.

```
/*
Watchdog Timer Basic Example
10 June 2011
Nicolas Larsen
*/

#include <avr/wdt.h>

int loop_count = 0;

void setup()
{
  Serial.begin(9600);
  Serial.println("Starting up...");
  pinMode(13,OUTPUT);
  digitalWrite(13,HIGH);
  delay (500);
  watchdogSetup();
}

void watchdogSetup(void)
{
cli();                    // disable all interrupts
wdt_reset();          // reset the WDT timer
/*
   WDTCSR configuration:
   WDIE  = 1: Interrupt Enable
   WDE   = 1 :Reset Enable
   WDP3 = 0 :For 2000ms Time-out
   WDP2 = 1 :For 2000ms Time-out
   WDP1 = 1 :For 2000ms Time-out
   WDP0 = 1 :For 2000ms Time-out
*/
// Enter Watchdog Configuration mode:
WDTCSR |= (1<<WDCE) | (1<<WDE);
// Set Watchdog settings:
 WDTCSR = (1<<WDIE) | (1<<WDE) | (0<<WDP3)  | (1<<WDP2) | (1<<WDP1)  | (1<<WDP0);
sei();
}

void loop()
{
for (int i = 0; i <= loop_count;i++){
  digitalWrite(13,HIGH);
  delay(100);
  digitalWrite(13,LOW);
  delay(100);
  }
  loop_count++;
  wdt_reset();
  Serial.print(loop_count);
  Serial.print(". Watchdog fed in approx. ");
  Serial.print(loop_count*200);
  Serial.println(" milliseconds.");
}

ISR(WDT_vect) // Watchdog timer interrupt.
{
// Include your code here - be careful not to use functions they may cause the interrupt to hang and
// prevent a reset.
}
```