Jean Rabault website - UiO

Sections:

Using the Arduino Uno watchdog

Note: the content of this post corresponds to the Arduino UNO. Other Arduino boards, based on other micro controllers, may have different registers. If you use a different micro controller, you will have to look for the corresponding registers in the datasheet.

Note: all the code presented here is available on Github.

Principle of a watchdog

Watchdog timers are used to avoid malfunction in electronic devices, for example hangs. For this, a physical timer keeps increasing in parallel of the program and drives an interrupt that can reset the microcontroller (in the case of the Arduino) if it hits a given value. In normal operation the timer is regularly set to zero in the program flow, but if the code hangs, the timer reset cannot occur and the watchdog will fire and trigger actions (such as rebooting) that can solve the problem.

Using the watchdog for preventing failures

As any 'low level' microcontroller feature, using the watchdog of an Arduino requests setting bit registers for selecting the behaviour of the chip. The corresponding registers and their meanings are described in the datasheet of the microcontroller embedded in each Arduino board, which can be quite painful to read. Fortunately, the Arduino IDE comes with a few functions and macros for easing the process, that can be imported through the *#include <avr/wdt.h>* command. This way, the watchdog can be enabled by calling the *wdt_enable()* function. The argument of the *wdt_enable()* function indicates the time before reboot of the board in the case when the watchdog does not get reset, and some convenient aliases are defined:

```
time before watchdog firing    argument of wdt_enable()
-----------------------------------------------------
15mS                           WDTO_15MS
30mS                           WDTO_30MS
60mS                           WDTO_60MS
120mS                          WDTO_120MS
```

```
250mS                         WDTO_250MS
500mS                         WDTO_500MS
1S                            WDTO_1S
2S                            WDTO_2S
4S                            WDTO_4S
8S                            WDTO_8S
```

For example, the following code will reboot the Arduino board after the setup loop and a 15 milliseconds frozen time (you will see the led 13 blinking as the board goes through each setup):

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

#define led 13 // pin 13 is connected to a led on Arduino Uno

void setup()
{
  pinMode(led, OUTPUT);      // set pin mode
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(100);                // wait for a tenth of a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(100);                // wait for a tenth of a second
}

void loop(){
  wdt_enable(WDTO_15MS);     // enable the watchdog
                             // will fire after 15 ms without reset

  while(1)
  {
//     wdt_reset();          // uncomment to avoid reboot
  }
}
```

For avoiding the board to reboot in normal operation, the *wdt_reset();* command should be called regularly in the program for reseting the watchdog.

Using the watchdog for energy savings

There are other ways in which the Arduino watchdog can be useful. In particular, the microcontrollers used in the Arduino boards can be set to low power sleep modes. As for the watchdog, the gestion of the low power modes is done via bit registers, but macros / functions / libraries are available to ease the programmer work (I let you browse the web if you need more details on this).
The point of importance is that in the lowest power mode, that can be configured by the *set_sleep_mode(SLEEP_MODE_PWR_DOWN);* command and activated by the *sleep_mode();* function of the #include <avr/sleep.h> library, the microcontroller is sleeping so deep that only an interrupt will be able to wake up the processor. Such an interrupt can come for example from the watchdog. Using an interrupt based on rising, falling, high or low level on one of the interrupt enabled pins would be another option, see here.
Therefore, making sure that an Arduino board would sleep consuming the least amount of current while still waking up periodically imposes to use the watchdog to wake up periodically the board. This can be done for example through the following program, where a few additional measures have been taken to save power, such as shutting off the ADC. Shutting off the brown-out voltage detection is also a possibility to further reduce power consumption during sleep, but is dangerous for the board functionning.

```
#include <avr/wdt.h>            // library for default watchdog functions
#include <avr/interrupt.h>      // library for interrupts handling
#include <avr/sleep.h>          // library for sleep
#include <avr/power.h>          // library for power control
```

```
// how many times remain to sleep before wake up
int nbr_remaining;

// pin on which a led is attached on the board
#define led 13

// interrupt raised by the watchdog firing
// when the watchdog fires during sleep, this function will be executed
// remember that interrupts are disabled in ISR functions
ISR(WDT_vect)
{
        // not hanging, just waiting
        // reset the watchdog
        wdt_reset();
}

// function to configure the watchdog: let it sleep 8 seconds before firing
// when firing, configure it for resuming program execution
void configure_wdt(void)
{

  cli();                           // disable interrupts for changing the registers

  MCUSR = 0;                       // reset status register flags

                                   // Put timer in interrupt-only mode:
  WDTCSR |= 0b00011000;            // Set WDCE (5th from left) and WDE (4th from left) to enter config mode,
                                   // using bitwise OR assignment (leaves other bits unchanged).
  WDTCSR =  0b01000000 | 0b100001; // set WDIE: interrupt enabled
                                   // clr WDE: reset disabled
                                   // and set delay interval (right side of bar) to 8 seconds

  sei();                           // re-enable interrupts

  // reminder of the definitions for the time before firing
  // delay interval patterns:
  //  16 ms:     0b000000
  //  500 ms:    0b000101
  //  1 second:  0b000110
  //  2 seconds: 0b000111
  //  4 seconds: 0b100000
  //  8 seconds: 0b100001

}

// Put the Arduino to deep sleep. Only an interrupt can wake it up.
void sleep(int ncycles)
{
  nbr_remaining = ncycles; // defines how many cycles should sleep

  // Set sleep to full power down.  Only external interrupts or
  // the watchdog timer can wake the CPU!
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);

  // Turn off the ADC while asleep.
  power_adc_disable();

  while (nbr_remaining > 0){ // while some cycles left, sleep!

  // Enable sleep and enter sleep mode.
  sleep_mode();

  // CPU is now asleep and program execution completely halts!
  // Once awake, execution will resume at this point if the
  // watchdog is configured for resume rather than restart

  // When awake, disable sleep mode
  sleep_disable();

  // we have slept one time more
```

```
    nbr_remaining = nbr_remaining - 1;

    }

    // put everything on again
    power_all_enable();

  }

  void setup(){

    // use led 13 and put it in low mode
    pinMode(led, OUTPUT);
    digitalWrite(led, LOW);

    delay(1000);

    // configure the watchdog
    configure_wdt();

    // blink twice
    digitalWrite(led, HIGH);
    delay(500);
    digitalWrite(led, LOW);
    delay(500);
    digitalWrite(led, HIGH);
    delay(500);
    digitalWrite(led, LOW);
    delay(500);

  }

  void loop(){

    // sleep for a given number of cycles (here, 5 * 8 seconds) in lowest power mode
    sleep(5);

    // usefull stuff should be done here before next long sleep
    // blink three times
    digitalWrite(led, HIGH);
    delay(500);
    digitalWrite(led, LOW);
    delay(500);
    digitalWrite(led, HIGH);
    delay(500);
    digitalWrite(led, LOW);
    delay(500);
    digitalWrite(led, HIGH);
    delay(500);
    digitalWrite(led, LOW);
    delay(500);

  }
```

You can notice the call to the *ISR(WDT_vec)* function. This is an interrupt function, that will be called when the watchdog fires. WDT_vec indicates that this is the interrupt that will be called by the watchdog. Other sources can call interrupts and will have different _vec parameters. There are a few things to know about ISR functions. First, interrupts are disabled in ISR functions, so interrupt based functions will not work properly (for example, *Serial.write*). Moreover, usual variables cannot be modified in ISR functions. If you need to modify a variable in an interrupt, you should declare it as volatile the first time you use it (see next section). In general, ISR functions should be as short as possible, to allow the normal program flow to resume as fast as possible (do not forget that interrupts are disabled during ISR, so receiving on serial or pin driven interrupts will not work when executing ISR core).
One starts to see here the limits of general purpose libraries for detailed control of the microcontroller behavior. As more specific behavior is needed, it becomes necessary to start digging in the datasheet and bit registers to get full control over the microcontroller.

Using the watchdog for both preventing failures and energy savings

We have learnt to use the watchdog to avoid code hangs and to put the Arduino in its lowest power mode. We can also use both functionnalities in the same program, by using a little bit more code in the *ISR()* function. This is exactly what the next example does.

```
#include <avr/wdt.h>            // library for default watchdog functions
#include <avr/interrupt.h>      // library for interrupts handling
#include <avr/sleep.h>          // library for sleep
#include <avr/power.h>          // library for power control

// how many times remain to sleep before wake up
// volatile to be modified in interrupt function
volatile int nbr_remaining;

// pin on which a led is attached on the board
#define led 13

// interrupt raised by the watchdog firing
// when the watchdog fires, this function will be executed
// remember that interrupts are disabled in ISR functions
// now we must check if the board is sleeping or if this is a genuine
// interrupt (hanging)
ISR(WDT_vect)
{
    // Check if we are in sleep mode or it is a genuine WDR.
    if(nbr_remaining > 0)
    {
        // not hang out, just waiting
        // decrease the number of remaining avail loops
        nbr_remaining = nbr_remaining - 1;
        wdt_reset();
    }
    else
    {
        // must be rebooted
        // configure
        MCUSR = 0;                          // reset flags

                                            // Put timer in reset-only mode:
        WDTCSR |= 0b00011000;               // Enter config mode.
        WDTCSR =  0b00001000 | 0b000000;    // clr WDIE (interrupt enable...7th from left)
                                            // set WDE (reset enable...4th from left), and set delay interval
                                            // reset system in 16 ms...
                                            // unless wdt_disable() in loop() is reached first

        // reboot
        while(1);
    }
}

// function to configure the watchdog: let it sleep 8 seconds before firing
// when firing, configure it for resuming program execution by default
// hangs will correspond to watchdog fired when nbr_remaining <= 0 and will
// be determined in the ISR function
void configure_wdt(void)
{

  cli();                            // disable interrupts for changing the registers

  MCUSR = 0;                        // reset status register flags

                                    // Put timer in interrupt-only mode:
  WDTCSR |= 0b00011000;             // Set WDCE (5th from left) and WDE (4th from left) to enter config mode,
                                    // using bitwise OR assignment (leaves other bits unchanged).
```

```
    WDTCSR =  0b01000000 | 0b100001; // set WDIE: interrupt enabled
                                     // clr WDE: reset disabled
                                     // and set delay interval (right side of bar) to 8 seconds

    sei();                           // re-enable interrupts
  }

  // Put the Arduino to deep sleep. Only an interrupt can wake it up.
  void sleep(int ncycles)
  {
    nbr_remaining = ncycles; // defines how many cycles should sleep

    // Set sleep to full power down.  Only external interrupts or
    // the watchdog timer can wake the CPU!
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);

    // Turn off the ADC while asleep.
    power_adc_disable();

    while (nbr_remaining > 0){ // while some cycles left, sleep!

    // Enable sleep and enter sleep mode.
    sleep_mode();

    // CPU is now asleep and program execution completely halts!
    // Once awake, execution will resume at this point if the
    // watchdog is configured for resume rather than restart

    // When awake, disable sleep mode
    sleep_disable();

    }

    // put everything on again
    power_all_enable();

  }

  void setup(){

    // use led 13 and put it in low mode
    pinMode(led, OUTPUT);
    digitalWrite(led, LOW);

    delay(1000);

    // configure the watchdog
    configure_wdt();

    // blink twice
    digitalWrite(led, HIGH);
    delay(500);
    digitalWrite(led, LOW);
    delay(500);
    digitalWrite(led, HIGH);
    delay(500);
    digitalWrite(led, LOW);
    delay(500);

  }

  void loop(){

    // sleep for a given number of cycles (here, 5 * 8 seconds) in lowest power mode
    sleep(5);

    // usefull stuff
    // blink three times
    digitalWrite(led, HIGH);
    delay(500);
```

```
  digitalWrite(led, LOW);
  delay(500);
  digitalWrite(led, HIGH);
  delay(500);
  digitalWrite(led, LOW);
  delay(500);
  digitalWrite(led, HIGH);
  delay(500);
  digitalWrite(led, LOW);
  delay(500);

  // now a real hang is happening: this will rebood the board
  // after 8 seconds
  // (at the end of the sleep, nrb_remaining = 0)
  while (1);

}
```

Conclusion

You can now use the Arduino watchdog for both failure prevention and energy savings. Since the detailed gestion of the watchdog is done explicitly in the program, it is easy for you to adapt the code to your needs. For example, if you need to push some content to the internet and that some operations request some time, you can call the following code before each time consuming operation

```
wdt_reset();
nbr_remaining = 5;
```

to let the following operation several watchdog times (here, 5 * 8 = 40s) to perform before rebooting (I found it useful when working with the CC3000 WiFi shield).

Disclaimer

Disclaimer: the content of this page is the result of a mix and classification work of the content I read on a number of other webpages, including this one, this one, and this one. I hope I do not forget to cite additional sources of inspiration. If I clearly used some verbatim pieces of your code to build the functions on this page and forget to cite you, contact me and I may add a link to your work.