Arduino, Zigbee and Embedded Development

A helpful blog designed to get beginners up and running with Arduino, Zigbee and general embedded development.

Saturday, 26 November 2011

Sleeping Arduino - Part 4 Wake Up Via Internal Timer

Overview

Welcome to part 4 of the Sleeping Arduino Series. In this entry we will cover how to get the Arduino to wake from sleep using one of the micro-controller's internal timers. This does not require any external hardware, all is implemented in code.

Arduino Timers

So,we need to go into a little detail on the micro controller's internal timers. The Atmega168 in the Arduino Diecimila has three internal timers:

- Timer/Counter 0 - 8 bit (Max timer duration: 16.4ms)
- Timer/Counter 1 - 16 bit (Max timer duration: 4.1s)
- Timer/Counter 2 - 8 Bit (Max timer duration: 16.4ms)

Whether the timer is 8 bit or 16 bit defines how much the timer can count up to: an 8 bit timer counting to 256 and 16 bit to 65536. What drives the counter to count can be configured to be either the internal 16Mhz clock or an external clock source on T1 pin. We will be using the internal timer in this tutorial.
Once a timer's counter has reached its maximum value and increments once more, it will **overflow** and the counter will reset to zero. This overflow event can be configured to trigger an **overflow interrupt**, which we will use to wake the Arduino from sleep mode. Note you can also modify the value within a timer's counter from your code to tune the overflow period.

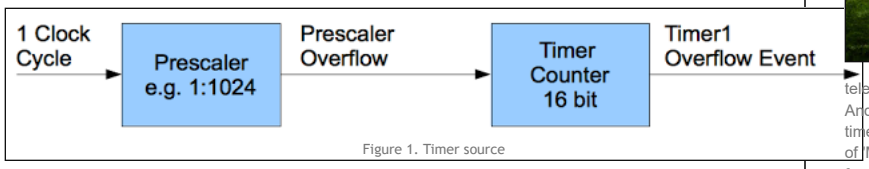

Figure 1. Timer source

One thing I haven't mentioned yet is the **prescaler**, this is another counter that is used in conjunction with the the timer counter to extend the time between timer overflows. The prescaler can be set the the following values: 1:1, 1:8, 1:64, 1:256, 1:1024, these are the ratios of prescaler overflow to clock cycles.
The following formula shows the calculation of the timeout period.

$$TimeoutPeriod = (1/16\text{Mhz}) * Prescaler * 2^{(timer\ bit\ count)}$$

Figure 2. Timeout period formula

Take the following examples:

16 bit Timer1 no prescaler (**1:1**)

Overflow period = 1/16Mhz x 1 x 2^16 = 4.09 milliseconds

16 bit Timer1 with prescaler of **1:1024**

Overflow period = 1/16Mhz x 1024 x 2^16 = 4.09 seconds

So we can see that using the maximum prescaler value of 1:1024, we can get maximum Timer1 overflow period of 4.09 seconds, this is the maximum time we can put the Arduino to sleep for using Timer1. If you want a longer sleep period than this, you can use the Watch Dog Timer, which can provide a sleep time of about 8 seconds (it can also be used in a lower power/sleep mode than Timer1, see Arduino/Atmega168 Timers For Sleeping). Using the WDT for waking from sleep is described in the 5th part of this series.

Timers and Power Modes

Pages

- Home
- Projects & Tutorials

Blog Archive

- ► 2012 (7)
- ▼ 2011 (7)
  - ► December (2)
  - ▼ November (4)
    - Sleeping Arduino - Part 5 Wake Up Via The Watchdog...
    - Sleeping Arduino - Part 4 Wake Up Via Internal Tim...
    - Arduino/Atmega168 Timers For Sleeping
    - Beagleboard Ubuntu 11.10 Single User Mode
  - ► September (1)
- ► 2010 (5)

About Me

Donal Morrissey

Senior embedded software engineer and team leader with experience in the telecommunications and automotive industries. Android application developer during my spare time. This includes the continued development of 'My Migraine Log', which is an application for tracking a user's migraine attack history. And a similar app called 'My Headache Log'. Both are currently on sale on the Android market. Specialties o Embedded software development o Android application development o SCRUM o Real-Time Operating Systems o Linux software development (embedded & desktop) o Microchip PIC16/PIC24 firmware development o Bootloaders, including fault tolerant remote update of application software o Embedded RF system software o Automotive embedded software o CAN Gateways & networks o Hardware interfacing (I2C, SPI, CAN, Etc) o Zigbee o C, C++, JAVA, Perl, Python o QT o Team Leadership o Recruitment

View my complete profile

Site Map

Site Map

As I have mentioned in part 1 of this series, not all hardware peripherals of the micro-controller are available in all power modes. Looking at our table here, we can see that timer1's lowest running power mode is IDLE, so when we put the micro-controller to sleep, we need to make sure we don't enter a sleep mode below this, or the timer will be disabled.

Control Registers

The following are the timer related control registers we will use to configure Timer1

- **TCCR1B - CS10, CS11 & CS12** - Timer control register B and the pre-scaler selection bits.

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.

- **TCNT1** - 16bit counter register. This is the actual register that will count up each timer tick. When the value in this register rolls over from 65536 to 0, the overflow interrupt will fire.
- **TIMSK1 - TOIE1** - Interrupt mask register and overflow interrupt enable bit. This register contains the control bits to enable the various interrupts available to tir

For more information on these registers, see section 16.11 of the datasheet here.

Operation

Our code will operate as follows:

1. Configure the serial port and LED pin.
2. Configure Timer1's prescaler so the timer will expire every 4.09 seconds (TCCR1B).
3. Clear Timer1's counter TCNT1
4. Enable Timer1 overflow interrupt (TIMSK1).
5. The main loop will then:
    1. Enter IDLE sleep mode.
    2. When the Timer1 overflow occurs, the interrupt will wake the processor
    3. The LED state will be toggled.
    4. Re-enter IDLE sleep mode.

Code

In the following code the Arduino board will go to sleep (SLEEP_MODE_IDLE) for 4.09 seconds, wake up and toggle the LED and go back to sleep again.

```
/*
 * Sketch for testing sleep mode with wake up on timer.
 * Donal Morrissey - 2011.
 *
 */
#include <avr/sleep.h>
#include <avr/power.h>

#define LED_PIN (13)

volatile int f_timer=0;



/***************************************************
 *  Name:        ISR(TIMER1_OVF_vect)
 *
 *  Returns:     Nothing.
 *
 *  Parameters:  None.
 *
 *  Description: Timer1 Overflow interrupt.
 *
 ***************************************************/
ISR(TIMER1_OVF_vect)
{
  /* set the flag. */
   if(f_timer == 0)
   {
     f_timer = 1;
   }
}


/***************************************************
 *  Name:        enterSleep
 *
 *  Returns:     Nothing.
 *
 *  Parameters:  None.
 *
```

```
 *  Description: Enters the arduino into sleep mode.
 *
 *****************************************************/
void enterSleep(void)
{
  set_sleep_mode(SLEEP_MODE_IDLE);

  sleep_enable();


  /* Disable all of the unused peripherals. This will reduce power
   * consumption further and, more importantly, some of these
   * peripherals may generate interrupts that will wake our Arduino from
   * sleep!
   */
  power_adc_disable();
  power_spi_disable();
  power_timer0_disable();
  power_timer2_disable();
  power_twi_disable();

  /* Now enter sleep mode. */
  sleep_mode();

  /* The program will continue from here after the timer timeout*/
  sleep_disable(); /* First thing to do is disable sleep. */

  /* Re-enable the peripherals. */
  power_all_enable();
}



/*****************************************************
 *  Name:        setup
 *
 *  Returns:     Nothing.
 *
 *  Parameters:  None.
 *
 *  Description: Setup for the serial comms and the
 *                timer.
 *
 *****************************************************/
void setup()
{
  Serial.begin(9600);

  /* Don't forget to configure the pin! */
  pinMode(LED_PIN, OUTPUT);

  /*** Configure the timer.***/

  /* Normal timer operation.*/
  TCCR1A = 0x00;

  /* Clear the timer counter register.
   * You can pre-load this register with a value in order to
   * reduce the timeout period, say if you wanted to wake up
   * ever 4.0 seconds exactly.
   */
  TCNT1=0x0000;

  /* Configure the prescaler for 1:1024, giving us a
   * timeout of 4.09 seconds.
   */
  TCCR1B = 0x05;

  /* Enable the timer overlow interrupt. */
  TIMSK1=0x01;
}



/*****************************************************
 *  Name:        enterSleep
 *
 *  Returns:     Nothing.
 *
 *  Parameters:  None.
 *
 *  Description: Main application loop.
 *
 *****************************************************/
void loop()
{
  if(f_timer==1)
  {
    f_timer = 0;
    /* Toggle the LED */
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));

    /* Re-enter sleep mode. */
    enterSleep();
  }
}
```

All parts of this series:

- Part 1 Overview Of Arduino Sleep Modes
- Part 2 Wake Up Via An External Interrupt
- Part 3 Wake Up Via the UART
- Part 4 Wake Up Via Internal Timer
- Part 5 Wake Up Via The Watchdog Timer

References

- Darius' excellent blog here.
- http://www.avrfreaks.net/index.php?module=Freaks%20Devices&func=displayDev&objectid=78
- http://popdevelop.com/2010/04/mastering-timer-interrupts-on-the-arduino/
- http://www.electronicsblog.net/examples-of-using-arduinoatmega-16-bit-hardware-timer-for-digital-clock/
- Arduino Timer Interrupt

Posted by Donal Morrissey at 10:46

26 comments:

Rusty 8 August 2012 at 11:47

Thanks for the tutorial. I am new to interrupts/sleep mode and these are very helpful.

One question tho: how can I verify my device actually goes to sleep? When I constantly print to the Serial port, there is a steady flow of data, even when I think the device should be sleeping. However, the LED does blink every 4 seconds so it seems like the timer is working correctly.

Any help would be greatly appreciated.

Reply

Donal Morrissey 9 August 2012 at 05:55

Hi Rusty,
The best way to verify your Arduino is in sleep mode is to monitor the current it draws, as it should draw less current while in these modes.
You probably have two main options to do this: 1. use a bench power supply that shows the current consumption or 2. use a multimeter to measure current (see here: http://www.electronics-radio.com/articles/test-methods/meters/how-to-measure-current.php).

While in a sleep mode your code should not be executing, so if your Arduino is sending out data from the serial port, then it looks like it is not asleep. The LED is blinking, so perhaps you are not calling the enterSleep() function in the appropriate place?

Only other thing I can think of is if you share your code so I can have a look?

Cheers,
Donal

Reply

Rusty 9 August 2012 at 12:19

Thanks for the response.

I just uploaded your sketch to a ATmega328 board, and hooked it up to the multimeter. It's definitely not going to sleep.

Any ideas as to why this could be? It seems like the sleep_mode() command does not work, as it only takes a millisecond or so to execute, rather than the 4 seconds that I was expecting.

Reply

Donal Morrissey 10 August 2012 at 04:39

Hi Rusty,
Looks like I was not explicitly disabling Timer0 and Timer2, which were still powered on in SLEEP_MODE_IDLE and generating interrupts that kept waking the Arduino! I was a little surprised to find this, given that I hadn't configured the interrupts for those timers in my sketch.
I've updated the sketch to disable all unnecessary peripherals including the timers (which should reduce power consumption even further) before entering sleep and re-enabling all peripherals after waking up.
Thank you for the feedback, very much appreciated!

Cheers,
Donal

Reply

Rusty 10 August 2012 at 11:32

Works better, but I believe that there is still a bug here somewhere. I placed the line Serial.println(millis()) at the very end of the loop() function to check on the device's status.

It prints every few milliseconds for the first 4 seconds or so, which is expected, then seems to sleep as expected. But then after this only prints once for each 4 second block, and I expect it to print constantly when the device is awake. Also, the prints are a few milliseconds apart so I assume the millis() timer is paused during sleep.

Could it be that the Timer1 is not reset after coming out of sleep?

Thanks.

Reply

Donal Morrissey 11 August 2012 at 07:47

Hi Rusty,
I've copied the code as is currently on the page above and added in the call to Serial.println(millis()); at the very end of the loop() function.
The sketch goes straight to sleep after setup is complete, then wakes up every 4 seconds. When it wakes up it will print out the millisecond count only once, toggle the state of the LED and go straight back to sleep (as f_timer is set to 1 by the interrupt).
I'm using the "Arduino 0012 Alpha" IDE and an Arduino Duemilanove (Atmega168).
What are you using?
Cheers,
Donal

Reply

Rusty 14 August 2012 at 06:26

I think I am misunderstanding how this sketch should work. After the first time the device sleeps, loop() only iterates once per sleep cycle, meaning that f_timer is always 1 (or perhaps very briefly 0).

How would I reset/disable the timer so that loop runs for a given amount of time, and wouldn't immediately go back to sleep mode the next iteration? (awake n seconds > sleep n seconds > repeat)

Reply

Donal Morrissey 14 August 2012 at 13:02

Hi Rusty,
You could modify the loop() function to something like the following to keep the Arduino awake and doing something for a period of time before going back to sleep:

```
/*************************************/
#define WAKE_DURATION_MS (1000ul)
unsigned long current_wake_duration;
unsigned long wakeup_timestamp;

void loop()
{

/* Do something while awake */
//...

/* Record how long we have been awake for. */
current_wake_duration = millis() - wakeup_timestamp;

/* Check if the wake duration has elapsed */
if( current_wake_duration >= WAKE_DURATION_MS )
{
/* Clear the timer-counter register.
* This will make sure we sleep for the corrrect duration.
* as we have not stopped the timer.
*/
TCNT1=0x0000;

/* Re-enter sleep mode. */
enterSleep();

/* Take a timestamp from when we have woken up. */
wakeup_timestamp = millis();
}

}
/*************************************/
```

If you expected your sketch to be running for longer than 50 days you would need to handle the roll-over of the value returned from the millis() function. See here:

http://arduino.cc/en/Reference/Millis
Also note that I have removed the f_timer flag as it isn't really needed.

Another solution to this would be to use another timer, and upon exiting sleep, enable that timer. When the timer expires and the interrupt is executed, a flag could be updated that indicates the desired wake period has expired. Your loop() function could continuously check this flag to see if it needed to go back to sleep.

Hope this helps.

Cheers,
Donal

Reply

Rusty 15 August 2012 at 06:34

Great!! That's working exactly the way I wanted. After reviewing your code its really quite simple, I'm just new to these uncommon Arduino functions.

Thanks a lot for all the help.

Reply

wisesm0 26 June 2013 at 10:58

Hey Donal,

Is there a way I can email you? I am working with an Arduino Nano AtMega328p and an SP02 sensor. I am having the same problems Rusty had above. The LED will flash on and off; however, my data continues to output to the serial monitor. It will never stop.

Reply

Replies

wisesm0 27 June 2013 at 14:49

This comment has been removed by the author.

Donal Morrissey 27 June 2013 at 15:18

Hi wisesm0, I will have a look this weekend and post a response.
Donal

Donal Morrissey 27 June 2013 at 15:19

Hi wisesm0, I will have a look this weekend and post a response.
Donal

Donal Morrissey 30 June 2013 at 04:56

Hi wisesm0,
Probably best you email me with your code. My email address is: firstname.lastname@gmail.com (obviously replacing the two words with my name...)
Cheers,
Donal

wisesm0 2 July 2013 at 10:50

Thank you. I've sent you an email. I hope it works out!

Reply

wisesm0 1 July 2013 at 11:38

Awesome. Thanks!

Reply

Jack Pierce 25 July 2013 at 12:13

Hi Donal,
I copied and pasted your code above into my IDE, and it compiled, and loaded into my arduino leonardo fine. It runs, led off and on about 4 seconds, but when I put my amp meter on the power, it is not dropping as I had expected. I expected it to be below 2ma while sleeping, but it is is drawing 31ma to 33ma, not going lower. Am I wrong thinking sleep will dray low amps?
The green power led stays on. Is it not able to turn off?
Thanks, Jack

Reply

kk 5 August 2013 at 09:04

Hi, very good explanantion!!
I wonder if it should be possible in practice to use this sleep code for this project:
- common supermarket batteries
- wake up arduino
- power up a little water pump (let's say) for 5 minutes to bring water from bottle to plant;
- send start,end time and battery charge via zigbee (or other wireless low power) to a raspberry pi (that can for example send an email or SMS to me)
- back to sleep
Could You help me, have any ideas if it can work?
Thank you!

Reply

Replies

Donal Morrissey 13 August 2013 at 23:41

Hi KK,
Sorry for the delay in replying to you.
That sounds like a cool project. It would be reasonably straight forward to do I'd say. You should be able to use an xbee shield along with a simple circuit with a relay switch to power switch the motor on and off.
Using the Arduino and Xbee shield wouldn't be the most power efficient, but it would be a good start. Once you have the system working, you could look into switching from the Arduino board.
Donal

Reply

Jim Piccirillo 11 August 2013 at 11:11

Donal, I am not sure u are still responding to this site.
I am trying to use your code. I am a novice using a new Arduino
UNO. Using a battery, I want to take 100 ADC sampels (output via USB and then go to sleep. Wake up one minute later and do it over and over.
Problem is when I copy an paste your code, the LED comes "ON" and never blinks.
Does it mater that I am using an UNO? Is there some way I can look at wha tis goin on? Error file?
I am not sure what diagnostics can be used. Thank You with your help. Jim P

Reply

Replies

Donal Morrissey 13 August 2013 at 23:42

Hi Jim,
Sorry for the delay in getting back to you.
Hmm, I'll test this code with my UNO tonight and get back to you.
Cheers,
Donal

Donal Morrissey 14 August 2013 at 11:57

Hi Jim,
I was missing the pin configuration!
pinMode(LED_PIN,OUTPUT);
The code is updated now.
Cheers,
Donal

Reply

Rossati 19 March 2015 at 03:06

Hello

Tahks for the code. A question Timer 0 awoke Arduino? I need a correct time.

Jhon Rossati

Reply

Replies

Donal Morrissey 20 March 2015 at 12:02

Hi Jhon,
I don't fully understand your question. Are you looking to track real time, perhaps with an external realtime clock?
Donal

Reply

bh 23 August 2015 at 11:04

```
void pin2_isr()
{
while(!digitalRead(BUTTON)) {}
detachInterrupt(0);

}
```

solves the problem

Reply

Sören Wellhöfer 23 April 2017 at 17:52

Hi,
ISR(TIMER1_OVF_vect) seems never to be fired! (I'm using an Arduino UNO). Any clues to why this might be?
Thank you very much

Reply

Newer Post Home Older Post