Big Dan the Blogging Man

ATTiny85 Wake from Sleep on Pin State Change Code Example

Posted on August 10, 2014

I spent an inordinate amount of time trying to figure out how put my ATTiny85 to sleep and then wake it up when the state of a pin changes. So I'll document my test code in case anyone else goes searching for a simple example.

The reference I ended up using is:

> http://www.avrfreaks.net/index.php?name=PNphpBB2&file=printview&t=128982

Unfortunately, even after reading through the forum thread the final working code was evidently never posted. However, there was enough here for me to figure out the problem and get it running.

The following code is very simple. It flashes the LED quickly to let you know the program is started. It then sets up for sleep mode and goes to sleep. When the button is pushed, it awakes, flashes the LED once and goes back to sleep.

```
#include <avr/sleep.h>
#include <avr/interrupt.h>

const int switchPin                     = 3;
const int statusLED                     = 2;

void setup() {

    pinMode(switchPin, INPUT);
    digitalWrite(switchPin, HIGH);
    pinMode(statusLED, OUTPUT);

    // Flash quick sequence so we know setup has started
    for (int k = 0; k < 10; k = k + 1) {
        if (k % 2 == 0) {
            digitalWrite(statusLED, HIGH);
            }
        else {
            digitalWrite(statusLED, LOW);
            }
        delay(250);
        } // for
    } // setup

void sleep() {

    GIMSK |= _BV(PCIE);                     // Enable Pin Change Interrupts
    PCMSK |= _BV(PCINT3);                   // Use PB3 as interrupt pin
    ADCSRA &= ~_BV(ADEN);                   // ADC off
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);    // replaces above statement

    sleep_enable();                         // Sets the Sleep Enable bit in the MCUCR Register (SE BIT
    sei();                                  // Enable interrupts
    sleep_cpu();                            // sleep

    cli();                                  // Disable interrupts
    PCMSK &= ~_BV(PCINT3);                  // Turn off PB3 as interrupt pin
    sleep_disable();                        // Clear SE bit
    ADCSRA |= _BV(ADEN);                    // ADC on

    sei();                                  // Enable interrupts
    } // sleep

ISR(PCINT0_vect) {
    // This is called when the interrupt occurs, but I don't need to do anything in it
    }

void loop() {
    sleep();
    digitalWrite(statusLED, HIGH);
    delay(1000);
    digitalWrite(statusLED, LOW);
    }
```

One additional comment: in my actual program, I will need to awake if either of two buttons is pressed. To handle this, I simply add these additional statements below their near twins:

```
    PCMSK |= _BV(PCINT4);                   // Use PB4 as interrupt pin
    PCMSK &= ~_BV(PCINT4);                  // Turn off PB4 as interrupt pin
```

Share this:

Twitter Facebook Google

Like

Be the first to like this.

Related

Programming ATTiny85 Using Sparkfun Tiny AVR Programmer
In "c-tinys"

Using a KY040 Rotary Encoder with Arduino
In "c-arduino"

Low Power ATTiny85 Experiment
In "c-tinys"

This entry was posted in c-tinys and tagged ATTiny85 Sleep. Bookmark the permalink.

23 Responses to *ATTiny85 Wake from Sleep on Pin State Change Code Example*

Matt N *says:*
August 18, 2014 at 8:43 am

Thanks for posting this, it really helped me. I think I've been following the same forum posts as you but I don't have as good of a grasp of AND/OR'ing registers as you do, so seeing the code like this is perfect.

The regular Arduino PinChangeInt library is 3kb but your solution is only a few lines, much better for the ATtiny!

Reply

Sean Straw *says:*
August 21, 2014 at 4:13 pm

On uCs with limited memory, it is not uncommon to need to forego the complete library implementation of something and raid the library for the minimalistic functionality necessary to accomplish the needed task.

Other examples include LCD (quite piggy), random (brings in 700-900 bytes of overhead, which makes it almost unusable on an ATTiny2x with just 2K of flash, but for "random-ish", there are other things you can do that aren't as costly), long and floating point operations (which can sometimes be circumvented by changing how you perform math operations and what sort of precision you really need), and delay().

Delay brings in about 200 bytes of timer code that you can save by doing the following:

```
myDelay( int seconds )
{
while ( seconds– )
{
delayMicroseconds( 1000 );
}
}
```

This won't be quite as precise, but for most purposes ("wait 'x' seconds before re-arming"), is more than sufficient, and the memory saved on the uC can be used for something more critical.

Long math is slow enough that even on an 8MHz (no external crystal necessary) ATTiny8x, driving a pin high, delaying a fixed time (with delaymicroseconds()), driving the pin low, delaying, and doing long math to decrement a counter in a for loop results in a noticeable "click" in the output. See the code in http://www.arduino.cc/en/Tutorial/PlayMelody Determining the delay and number of iterations Using an int before entering the loop avoids that,

As I develop a programs, I test the builds and note the build size (for a given target, so size is apples-to-apples: if you change uC targets, binary sizes can change significantly), then note these either within comments within the module, or in the git commit comments (changes are pushed to a git repo regularly). Doing so, you easily visualize how much some changes add to your code size, and that helps a lot when you're working with limited program space.

Reply

xunker *says:*

August 22, 2014 at 8:32 am

Excellent insight, Sean! I love it.

Even though I consider myself an experienced programmer (15 years professionally), all my skill is in very-high-level languages on machines megabytes/gigabytes/terabytes of memory and storage, so shifting designs to focus on individual bytes is a huge — but rewarding — change in focus. Saving 200 bytes by rolling your own delay() function is something that would have never occurred to me.

Reply

Dan TheMan *says:*

August 25, 2014 at 10:51 am

I started programming in the late 70's in what would have been considered high-level then (Fortran and COBOL) but we still had to watch every byte we used. Learning to program microcontrollers has been like going back to my programming roots. Every byte, and every instruction counts and there aren't any fancy debugging tools. 🙂

Sean Straw *says:*

September 2, 2014 at 12:51 am

I started programming in assembler on Z80 (CP/M) followed by 6502 CPUs (Apple and Atari), and there you had at best 64KB of memory total for the entire system, including the OS (and the ROM). Things improved with the x86, though having gotten used to writing compact code, all that extra memory often went unused (often, one used extra memory to buffer things for increased performance, but because every PC had a different memory config, apps needed to run on the lower configs if you wanted them to be usable by a large audience).

Despite developing using higher level languages (C, C++, Perl, Ruby (ugh, did I just vomit a little?), etc) for most things these days, memory consumption still remains important. Poorly written OOP code can chew through memory. Some languages are versatile – I've found PHP to be nice for web dev, but dang, it's memory representation of some structures is gross. The venerable 8051 uC is still in fairly common use, and while it only has 16 bit addressing (64KB addressable space), it has different banks of memory (data, and program being the chief ones in use), so program is in 64K, data can be in a separate 64K. Using external address line switching and linker trickery, you can bank switch ROM to increase program size on the 8051. It'd been years since I did any 8051 dev, but as an ASIC at my employer utilizes the 8051 core, I recently got back to it for some projects, though otherwise most of my work is on an ARM based embedded system.

I've used the ATTiny24V for some projects – it has just 2K of flash. I have some ATTiny13A uCs (which sport just 1K of flash), but have yet to use them in anything (and, arguably, they're not so much cheaper than the '85 that it really makes sense to use them, outside of some lower voltage capabilities).

I'm eager to check out the ATTiny43U. 4K flash, and it has an integrated boost converter capability: it can drive itself off of a sub 1.5V battery *AND* reportedly drive some external loads once bootstrapped. Obviously though, it has a different pinout than its x4 and x5 ATTiny siblings, so won't be a drop-in design. For some of my own projects, I have to use a boost module I designed, which charges an UltraCap, but I don't have a control in place to shut off the boost when the Cap is above a target voltage (that's what I planned to use the ATTiny13A's for) – integrating this directly into the uC is really attractive.

Personally, my next step with the AVR is to switch or tweak development toolchains – the Arduino IDE is entirely too simplistic and limited — as it is, I set the option to use an external editor, and just use the IDE to launch the compiler and drive the binary through my programmer devices (AVRisp or Arduino-as-ISP with a ICSP ZIF shield I made). The gnu toolchain (c/c++ compiler, assembler, library tools, and linker – though the versions distributed in the Arduino IDE are fairly dated), is still underneath the IDE, so it's a matter of generating standard makefiles for projects and building libs with assembler, then the IDE can be ditched. AVR Studio 6.x is an alternative, but sadly, it's purely a windows offering.

Reply

Pingback: *Low Power ATTiny85 Experiment | Big Dan the Blogging Man*

José *says:*

October 24, 2014 at 2:12 pm

Hello, thank you for your code. Can i use the same code for an Attiny84 ? Thanks.

Reply

Dan TheMan *says:*

October 24, 2014 at 2:39 pm

Looking at the datasheet, (http://www.atmel.com/Images/doc8006.pdf), I think it would work.

I do not think you would hurt the 84 if it doesn't work.

Reply

Matt N *says:*

October 24, 2014 at 3:35 pm

I'm using the code on an '84 currently and I can confirm it works.

Sean Straw *says:*

October 24, 2014 at 3:02 pm

84 and 85 should differ only in the number of I/O (and their various capabilities), so the sleep code should be fine. Other members of the AVR family (such as the Tiny13A or 24) may not support all of the same Watchdog intervals, but those aren't employed here in Dan's example (a watchdog timout would be how you could sleep and wake periodically to check something, not just in response to an external event).

Reply

Dan TheMan *says:*

October 24, 2014 at 3:56 pm

It's always cool to see others jump in and help. Thanks guys!

Reply

KW *says:*
March 27, 2015 at 10:00 am

This was great! Definately saved me a couple of hours, thanks!
2 small points to mention:
1. I use Proteus for simulation and it doesn't like delaymicroseconds() for whatever reason. Yes, It is buggy but it was worth the money…I've used it to develop dozens of projects.
2. I adding a boolean (bRelease) initialized HIGH so you only trigger on button closure, not both close and open:

```
sleep(); // sleep until change of state
if (bReleased) { // is button in released state (open)
bReleased = LOW; // button is now in pressed state (closed)
digitalWrite(LED, HIGH); // pulse LED
delay(100);
digitalWrite(LED, LOW);
}
else { // button was in presse state so has been released
bReleased = HIGH; // so set flag and go take a nap
}
```

Reply

Pingback: *AtTiny85 put in sleep mode and awaken through a 4N35 optocoupler | curiosityBikes*

Pingback: *Arduino ile Attiny85 Programlama Guncel | Genel | Burak Öztürk*

Pingback: *[ASK] arduino - ATtiny85 (Sparfun Clone/Trinket), detect/read operation voltage | Some Piece of Information*

zach *says:*
June 23, 2016 at 7:14 pm

Hey thanks for the simple and concise code, just what a beginner like me was looking for for my project!!

Zach

Reply

imginit *says:*
January 13, 2017 at 10:17 pm

Thank you for the great tutorial. I got this to work with the ATTiny84 by making small edits. Here's my code:

```
void sleep() {

GIMSK |= _BV(PCIE0); // Enable Pin Change Interrupts
PCMSK0 |= _BV(PCINT3); // Use PB3 as interrupt pin
PCMSK0 |= _BV(PCINT4); // Use PB4 as interrupt pin
ADCSRA &= ~_BV(ADEN); // ADC off
set_sleep_mode(SLEEP_MODE_PWR_DOWN); // replaces above statement

sleep_enable(); // Sets the Sleep Enable bit in the MCUCR Register (SE BIT)
sei(); // Enable interrupts
sleep_cpu(); // sleep
cli(); // Disable interrupts
PCMSK0 &= ~_BV(PCINT3); // Turn off PB3 as interrupt pin
PCMSK0 &= ~_BV(PCINT4); // Use PB4 as interrupt pin
sleep_disable(); // Clear SE bit
ADCSRA |= _BV(ADEN); // ADC on

sei(); // Enable interrupts
} // sleep
```

Switched the PCIE to PCIE0 and PCMSK to PCMSK0.

Reply

Pingback: *ARDUINO: ATTINY85 AS AN ARDUINO AND SLEEP MODE – Simple-EE*

Victor Eng *says:*
June 21, 2017 at 2:02 pm

I did combined the codes it was provided by Imginit . It Works for my ATTiny44-20PU (Mouser P/N 556-ATTINY44-20PU) using Arduino UNO as ISP. Wake up 17.3mA(330ohm) or 5.5mA(10K); Sleep current 630uA.
I am analog person in Silicon-Valley USA for many years and never touch any code until three months ago. It is just for fun beside my work.
As we all know, this code sets sleep mode by the the internal delay(xxxxxx). Could any one help modify it makes both sleep and wake by external interrupt (momentary switch)?

Thank you for all.

Reply

Victor Eng *says:*
June 22, 2017 at 9:18 pm

Thank you for the posting! It Works for my ATTiny44-20PU (Mouser P/N 556-ATTINY44-20PU) using Arduino UNO as ISP just a small edit.

Wake up 17.3mA(330ohm) or 5.8mA(10K); Sleep current 20uA.

I am analog person in Silicon-Valley USA for many years and never touch any code until three months ago. It is just for fun beside my work.

As we all knew, this code sets sleep mode by the the internal processor. Could any one help modify it makes both sleep and wake by external interrupt (momentary switch)?

Thank you for all.

```
void sleep() {
GIMSK |= _BV(PCIE0); // Enable Pin Change Interrupts
PCMSK0 |= _BV(PCINT3); // Use PB3 as interrupt pin
ADCSRA &= ~_BV(ADEN); // ADC off
set_sleep_mode(SLEEP_MODE_PWR_DOWN); // replaces above statement

sleep_enable(); // Sets the Sleep Enable bit in the MCUCR Register (SE BIT)
sei(); // Enable interrupts
sleep_cpu(); // sleep

cli(); // Disable interrupts
PCMSK0 &= ~_BV(PCINT3); // Turn off PB3 as interrupt pin
sleep_disable(); // Clear SE bit
ADCSRA |= _BV(ADEN); // ADC on

sei(); // Enable interrupts
} // sleep

ISR(PCINT0_vect) {
// This is called when the interrupt occurs, but I don't need to do anything in it
}
```

Reply

Pingback: *Atduino: go to sleep tiny ATtiny | Tahium*

Pingback: *Arduino: go to sleep tiny ATtiny – All Great Offers Today*

Viktor Sobolyev *says:*

August 19, 2017 at 9:31 am

I've been trying out your code and unfortunately, when I wash flashing the uC for the second time it bricked, it repeated twice under the same conditions and I've managed to recover only one of them. I think it could be related to "digitalWrite(switchPin, HIGH);" in the setup, it doesn't see the external oscilator and the output is always high.
Did anyone experience this ? Does anyone know the proper configuration to unbrick the attiny ?

Reply

Big Dan the Blogging Man

Blog at WordPress.com.