MARCH 12, 2015

AVR Timer-based One Shot Explained

Last time (https://wp.josh.com/2015/03/05/the-perfect-pulse-some-tricks-for-generating-precise-one-shots-on-avr8/), we made one-shot pulses using the AVR's built in hardware timer module. Today we are going to dive deep into the datasheets to see how this technique is able to coax the normally free-running timer into generating a single pulse. Along the way, we will learn about the low level rules that govern the operation of the timer, and use a trick or two to get around those rules. Read on!…

A Simple Timer

The AVR Timer hardware has lots of modes. For this technique we will be using Fast PWM mode 7…

Table 18-8. Waveform Generation Mode Bit Description

Mode	WGM22	WGM21	WGM20	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX= 0xFF
2. BOTTOM= 0x00

(https://wpdotjoshdotcom.files.wordpress.com/2015/03/capture.jpg)

…which follows these fundamental waveform generation rules (excerpted from the data sheet (http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf))…

1. The counter counts from BOTTOM to TOP then restarts from BOTTOM.
2. The output is set when the counter equals MATCH.
3. The output is cleared at BOTTOM.

That's it. Seems straight forward, but with such simple rules how are we going to be able to find a loophole to drive our pulse train though?

11 comments

MARCH 20, 2015 - 8:13 PM
Julian Loiacono

thank you Josh. I have used your code to increase the reliability of my electrogalvanic stimulator: https://www.youtube.com/watch?v=j3t3TejWxOI (https://www.youtube.com/watch?v=j3t3TejWxOI) . It may also find application in some transcranial magnetic stimulator (TMS) research

MARCH 25, 2015 - 4:01 AM
ubi de feo (@ubidefeo) (http://twitter.com/ubidefeo)

hi Josh
I wish this existed when I had to figure out precise timing for my metronome :D
fantastic resource, really.
I'm gonna see if I can implement some of this knowledge in the future.
thank you for sharing.
:)

JULY 20, 2015 - 6:42 AM
Adrian Velcich (https://www.facebook.com/app_scoped_user_id/10154103955844129/)

Hi Josh, did you ever take a good look at using the 16 bit timer? I have transposed your code, but I'm struggling a bit to get it to work properly. If you have any thoughts, I'd appreciate them.

Thanks!

JULY 20, 2015 - 1:47 PM
bigjosh2 (http://wp.josh.com)

I looked over the registers and did not see any obvious blocking problems, but I have not tried to actually write the code. One thing to keep in mind is that the access to the 16-bit timer registers is atomic on AVR though a temp register, so this makes things easier.

AUGUST 12, 2015 - 7:12 AM
dntruong (http://dntruong.wordpress.com)

You are able to set a LOW-HIGH pulse of any width with this trick.

How would you apply this to ws2812 timings?

… I see manually launching each bit as a pulse encoding the low of the previous pulse:
___- zero
___– one
have the ws2812 code do a sleep loop for the whole cycle-1, clear interrupts, and check if the counter is not cleared, else abort transmission, then wait for end of cycle, set next bit, set interrupts?

That way a long interrupt just glitches a refresh but does not corrupt the display?

I have not played with avr timers yet, but I wonder if we can reverse the logic to set a HIGH-LOW pulse. Either reverse the pin level, or raise the BOTTOM value to non zero.

This has the benefit of telling us on a read if we are in a range [1..2] when we can force the next bit out. You can set the LOW duration to max of ws2812 and with a cmp decide if the interrupt corrupted your display (past 2 counter == BOTTOM), yet allow you to preempt deterministically the current cycle to push the next bit…

-__1_____2
—_1_____2

AUGUST 12, 2015 - 11:27 AM
bigjosh2 (http://wp.josh.com)

Yes! This is exactly why I started on this path. It turns out that it works, but it is only of limited usefulness. If the interrupt comes in the middle of a pixel, then that pixel will get reset with incomplete color info and change to a corrupted color. To avoid this, you need to disable interrupts for the full 24 bits of each pixel.

a. Disable interrupts

b. Send a full 24 bit pixel

c. Check to see if there is a pending interrupt. Maybe test the Interrupt Flag bits, or go ahead and enable and disable ints and check to see how long it took

d. If an INT is pending, then you about sending the rest of the string, enable interrupts, let the interrupt run, then start the string over from scratch.

Note that the longer your string, and the more frequent your interrupts then the more chance that a string update will get aborted and so the more stale the far end of the string will be.

AUGUST 18, 2015 - 3:38 PM
dntruong (http://dntruong.wordpress.com)

I think it's worth trying and putting into a lib as an alternative. It also allows 4mhz drivers to work reliably. can the bottom value be controlled to simplify pulse control?
This would give a good reason to fix the time interrupt code.

JANUARY 20, 2016 - 3:42 PM
Chris Hahn

Hi Josh:
This is an excellent article and I am interested in using it for a project I am working on, however, when I loaded this into my ATMEGA2560 board I was unable to get it to run. Is there an issue running this on the larger processors? Thanks for your help.

JANUARY 20, 2016 - 4:14 PM
bigjosh2 (http://wp.josh.com)

Should be no problem – there are plenty of timers (5!) available on that chip and they all seem to have the required functionality. Are you sure you are looking for the output on the pin corresponding to the timer you are using?

JANUARY 20, 2016 - 5:29 PM
Chris Hahn

Thanks for the quick reply Josh. I loaded the program exactly as written into my ATMEGA2560 board. It compiled correctly and gave no errors, however, I am not seeing the expected pulse signal on the specified pin, or any pin for that matter. Did I miss something?

FEBRUARY 2, 2016 - 1:30 PM
Chris Hahn

Ok, I think I've got it figured out. I've transposed your original code to use a 16-bit timer (timer-3), and also to run on an ATMEGA 2560 platform. For those interested, here is the code:

#include
#include

#define OSP_SET_WIDTH(cycles) (OCR3B = 0xffff-(cycles-1))

```
// Setup the one-shot pulse generator and initialize with a pulse width that is (cycles) clock counts long

void osp_setup(uint16_t cycles) {

TCCR3B = 0; // Halt counter by setting clock select bits to 0 (No clock source).
// This keeps anyhting from happeneing while we get set up

TCNT3 = 0x0000; // Start counting at bottom.
OCR3A = 0; // Set TOP to 0. This effectively keeps us from counting becuase the counter just keeps reseting back to 0.
// We break out of this by manually setting the TCNT higher than 0, in which case it will count all the way up to MAX
// and then overflow back to 0 and get locked up again.
OSP_SET_WIDTH(cycles); // This also makes new OCR values get loaded from the buffer on every clock cycle.

TCCR3A = (1<<COM3B0) | (1<<COM3B1) | (1<<WGM30) | (1<<WGM31); // OC3B=Set on Match, clear on BOTTOM. Mode 15 Fast PWM.
TCCR3B = (1<<WGM32) | (1<<WGM33) | (1<<CS30); // Start counting now. Mode 15 Fast PWM.

DDRE = (1 <0)

// Fire a one-shot pusle with the specififed width.
// Order of operations in calculating m must avoid overflow of the unint8_t.
// TCNT2 starts one count lower than the match value becuase the chip will block any compare on the cycle after setting a TCNT.

#define OSP_SET_AND_FIRE(cycles) {uint16_t m=0xffff-(cycles-1); OCR3B=m; TCNT3 = m-1;}

void setup()
{
osp_setup();

}

void loop()
{
// Step though 0-19 cycle long pulses for demo purposes

for (uint16_t o = 0; o < 20; o++) {

OSP_SET_AND_FIRE(o);

while (OSP_INPROGRESS()); // This just shows how you would wait if necessary - not nessisary in this application.
```

```
  delay(1000); // Wait a sec to let the audience clap

  }

  }
```

The Perfect Pulse- generating precise one-shots on AVR8 (https://wp.josh.com/2015/03/05/the-perfect-pulse-some-tricks-for-generating-precise-one-shots-on-avr8/)

Plates vs Coils – An alternative approach to wireless power transmission (https://wp.josh.com/2015/07/11/the-other-way-to-do-wireless-power-capacitive-power-transmision-proof-of-concept/)