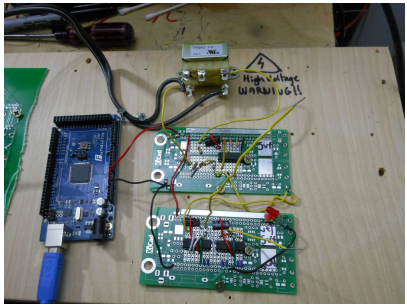
Contents

1 60Hz Divider

1.1 Overview

Let's count. There is a schematic in Practical Electronics For Beginners 4th edition. I've built that up, and will add some CPLD counter logic, along with a micro to output the SPI to a 7seg counter module.

The goal is relative accuracy. Not absolute. No GPS here. I'm going from 60 to 6,000 cycles.[1] This is just meant to be fun.



[1] Due to limitations of CPLD

Figure 1: 60 Hz Logic Divider to 1Hz

1.2 Initial Notes: Counting the Hz

pseudo code goal:

```
Using 1Hz signal
Start counting 1MHz every 1Hz
when next cycle is received,
   display count
   start counting again
```

That's all the objective is here. Easy with a micro, but goal is to complete using cmos or 74 logic.

4553 x 5 74hct132 1MHz clock (or 6MHz clock), or some variation thereof jk flip flop 74376 - quad jk flip flop 7476 - jk flip flop 1mhz clk will be main counter, 6 hz or 1 hz will be latch / reset

I ended up skipping the 74 CMOS, in favor of a CPLD. Practical Electronics also mentions this approach as favored. Even a micro alone could be used. Schematic entry in the CPLD could also be used.

1.3 MAX7219 8 digit 7 LED segment Display Driver

Basic code tested with this was the LedControl arduino library.

```
/*
 Now we need a LedControl to work with.
 ***** These pin numbers will probably not work with your hardware
 pin 12 is connected to the DataIn
 pin 11 is connected to the CLK
 pin 10 is connected to LOAD
 We have only a single MAX72XX.
 */
```

Some of the lines have to be edited to allow for all digits to be read, and also to lower intensity of display. I think also a component package (dark grey clear plastic bag) in front of the leds with intensity 1 is about right.

1.4 CPLD Programming

Using the XC9500XL series. This chip has some limitations - which are good.

As you get faster clocks, you need bigger registers to handle parsing the clocks. Bigger registers, use more power. Maybe this is one reason why high clock speeds mean more power.

1.4.1 6KHz clock

Due to limitations of the XC9500XL FPGA logic blocks, I ended up limiting the counter registers to 12+1 bits[2], so I have around 6,000 (assuming 60Hz), resolution. With this, I need a 6KHz clock. I could do this with the uno, but let's throw an attiny in there because it's a good tool for this kind of purpose and resolution. It should be able to function as a rough 6KHz timer, easily.

1.4.2 UART output

I set the CPLD to use the rising edge of the 6KHz clock and to shift the counter value out... Unsuprisingly, the baud rate is 6000. I found this by using my Open Bench Logic Sniffer[3]. It's fairly quick to configure and get working. Auto detected the UART speed easy.

However, my uart value is 12 - 14 bits, and with uart being an 8 bit protocol, it makes this unconventional. May need to bit bang something. But before that...

[2]Possibly I could use multiple smaller registers in a type of cascade, but let's not bother with that for now. I had 600KHz resolution, until I added the UART out/

[3]Phantom 3 in Repairs 2019

1.5 Divide by N Counters

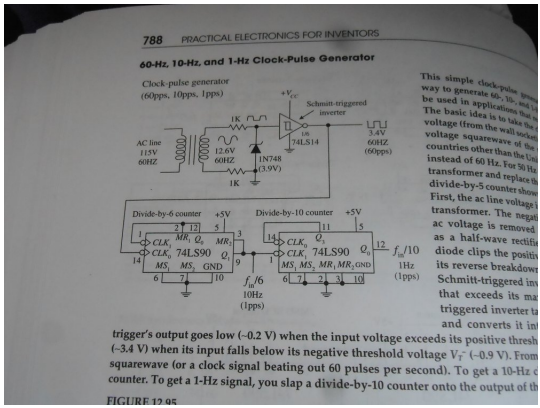


Figure 2: This divide by 6 counter, appears to not line up with what the TTL Cookbook has for a similar 7490 one.

The schematics appear to be incorrect for the divide by 6 counter in the Practical Electronics for Beginners book. Having looked at my built up circuit carefully, I see a 20Hz output from the 60Hz. I managed to get my hands on a copy of the TTL Cookbook by Don Lancaster recently, and that details correct divide by 6 and 10 counters (which are different from what's on my proto board), and while I could fix the divide by 6 counter, instead, I'm going to build another divide by 2 counter, and leave the original incorrect one there as a warning (it's also easier to just build a new one).

As it is, I'm getting 2Hz output on the pulse pin... Oops. Practical Untested Electronics for Beginners. Hax. Everything in life is hax. The earlier you realize that, the better you will feel about your own work.[4]

1.6 Attiny 6KHz Clock

A small victory here: I setup an Attiny10 with an external oscillator (programmable CMOS, not Quartz) of 1.536MHz. I then set prescaler at 256 to get 6000. Set micro fuse to enable CKOUT pin,

[4]It's possible they put the error in on purpose. It's really hard to tell...

and now I have a 6KHz clock from the 20 cent micro plus. Neat usage of the attiny10 here, thanks to my other project using it. The CPLD works with it, no problem.

1.7 Parsing of CPLD UART Stream

I have the UART stream feeding into the Atmega328/Uno. For the code, I was unsure how to handle it at first, but then I realized a simple shift in would fit.

Situation: I have a serial UART stream at 6000 baud from the CPLD. However, it's not exactly UART. In fact, it has values of 6000, which are over 8 bit. So I have a 14 bit serial stream. There is no stop bit after the 8 bits, and no two 8 bit bytes. So hardware serial will not work.

Solution: I have a serial 14 bit stream at 6000 baud. The answer is to tie the 6000 Hz CLK to a pin on the Uno, and implement a shift in, so that every clock up, the value is read on the Serial / 14 bit pin.

Problems: The timing is not 100% As a result, some values are being read incorrectly. 5996 shows up as 5048 or similar. This is likely because I quickly prototyped with digitalRead. I need to go back and access the Input direct via register reads to speed things up. A Pin register access similar to:

```
Example Code Snippet

Let's demonstrate the use of the DDRx,
PORTx and PINx registers from the
 following code snippet:

DDRC = 0x0F;
PORTC = 0x0C;

// lets assume a 4V supply comes to PORTC.6 and Vcc = 5V
if (PINC == 0b01000000)
    PORTC = 0x0B;
else
```

```
PORTC = 0x00;
```

Reference: http://maxembedded.com/2011/06/port-operations-in-avr/ may fix these issues. In the meantime, because the errors are consistent, I setup some LUTs[5].

1.8 Max7219 8 digit 7-Segment Display

I've

2 Project Rev A Complete

After another night or two of work, I have a working prototype.

[5] Lookup tables, i.e. hard coded fixes. 5048 now converts to 5996.