SI5351 Clock calculation using the Adafruit library

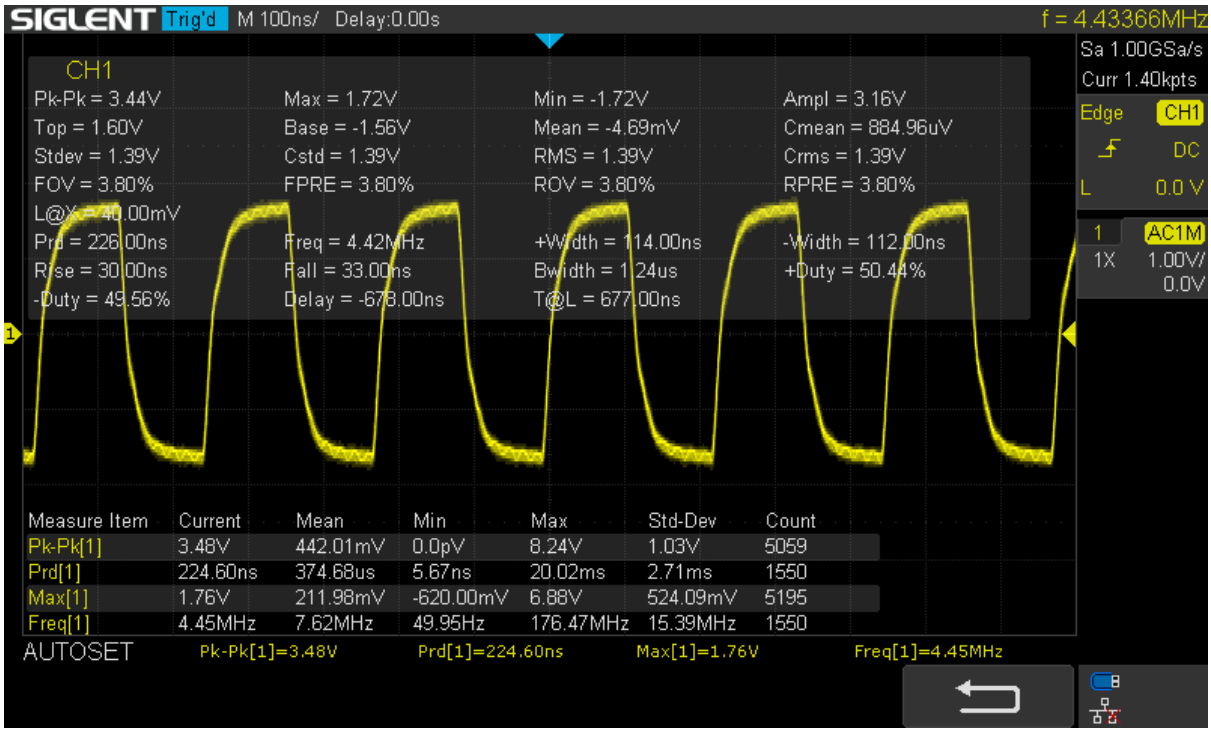
While prototyping a project, I needed to land precise PAL and NTSC SubCarrier frequencies. I had landed somewhat close a few years prior, but not for the fussy CXA1145 encoder found on the Sega Megadrive II.

I decided to document HOW to land as close as possible, this way I have a reference for future and don't need to work it out again. It can also help others using the SI5351 clockgen IC along with the Adafruit Library.

Here is the PAL SubCarrier accepted by the Sony CXA1145.

```
//PAL SubCarrier
void PAL_SUBC(){
    clockgen.setupPLLInt(SI5351_PLL_B, 36); // 900
    clockgen.setupMultisynth(1, SI5351_PLL_B, 202, 194346 , 195429);
//202.9944500753476
}
```

900/ 202.9944583454860845 = 4.4336MHz



Lets see if we can land the NTSC sub carrier while I write out this document!

Step 1: Take out target frequency which in this case is : **3.579545 MHz.**

Step 2: Add the target freqency to itself in a calculator, in this case the one found in windows OS.
Copy and paste every result that lands close to a clean integer, I pasted all of them up until the result was around 900.

25.056815
68.011355
93.06817
136.02271
161.079525
204.034065
229.09088
272.04542
314.99996
340.056775
383.011315
408.06813
451.02267
476.079485
519.034025
544.09084
587.04538
629.99992
655.056735
698.011275
723.06809
766.02263
791.079445
834.033985
859.0908
902.04534

Step 3: Which of these is closest to a clean integer when divided by 25?

ChatGPT shortcut:
“Based on these calculations, the number that yields a result closest to a clean integer when divided by 25 is 629.99992, which rounds to 625 when multiplied by 25.”

According to the adafruit documentation for the library, the multiplier can range from **15 to 90**. That would rule some of the above out, but the closest mentioned is 625 when multiplied by 25. ***25 Falls within the acceptable multiplier range.***

This gives us a clean integer PLL of **625**

clockgen.setupPLLInt(SI5351_PLL_B, 25); *// 25 * 25 = 625*

Step 4: With our PLL now set, divide the newly discovered PLL by the target frequency.

625 / TARGET (**3.579545)** = 174.6031967750091 ***(this needs to fall within 4 to 900)***

From Adafruit documentation for fractional divider

clockgen.setupMultisynth(output, SI5351_PLL_x, div, n, d);

For the output use 0, 1 or 2

For the PLL input, use either SI5351_PLL_A or SI5351_PLL_B

The final frequency is equal to the PLL / (div + n/d)

div can range from 4 to 900

n can range from 0 to 1,048,575

d can range from 1 to 1,048,575

So div now becomes **174** from above.

We just need to calculate 'n' and 'd', we want N/D to yield a result as close as possible to our result past the decimal point above, being 0.6031967750091.

Shortcut again, ChatGPT.

Tell it the rules from above and have it have a crack a few times until it gets the best result. This won't be it's first attempt. Ask it a few times and take the closest result.

I ask it this:

n can range from 0 to 1,048,575

d can range from 1 to 1,048,575

I want N divided by D to be as close as possible to 0.6031967750091.

SHIT RESPONSE received!!!

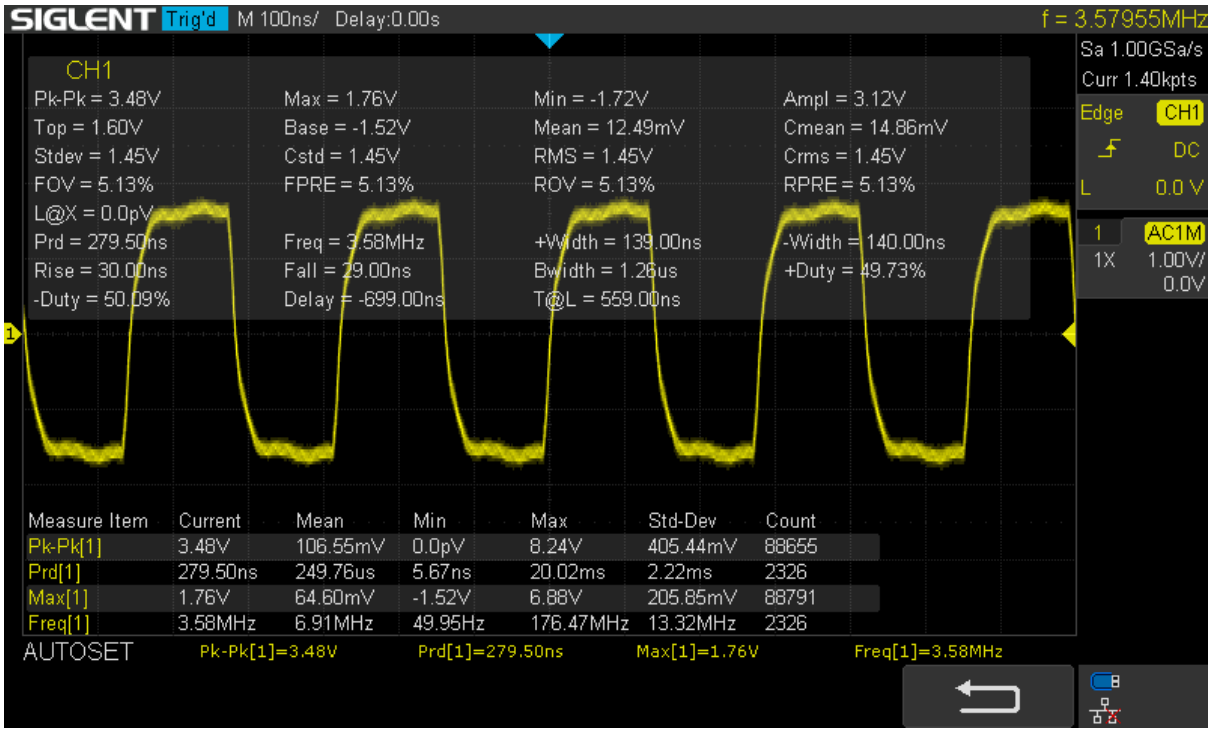
So we say (lol)

Work it out yourself, don't give me python scripts

Again, shit responses received. You may have to try quite a few times. It really struggled, but I finally have 200/331 = 0.6042296072507553.

625/ 174.6042296072507 = 3.5795238260027 MHz

Lets plug this into the code and see what results we get:



clockgen.setupPLLInt(SI5351_PLL_B, 25); *// 25 * 25 = 625*
clockgen.setupMultisynth(1, SI5351_PLL_B, 174,200, 331); //174.6042296072507553.

And we land again, just as the PAL SubCarrier, accuracy to 4 decimal places.

Is the system happy with this?

NTSC SubCarrier derived from SI5351

PAL SubCarrier derived from SI5351

Though there are “on paper” / technical advantages to having step-locked clocks, as in the sub carrier being derived from division of the systems master clock (stock system), these systems are just noisy as all hell. There is VERY apparent rainbow banding from the moment you switch the system on in this JP model 2 VA1’s case. You instantly see small rainbow colours in the white boot up text and will see rainbow banding often in game play. Esecially across any white pixels.

Even on this protoype with lengthy wires picking up noise and running all over the shop, the composite output is ***significantly better.*** I can not see rainbows anywhere and even @60Hz you may prefer the more vivid colours found with the PAL SubCarrier which is an optional video mode. The project can continue, and I may rework the master clocks with a similar method.

Keep in mind there are other ways to do this, this is just a method I figured out. You can use a fractional PLL instead of an integer as well, but this may produce a shoddier clock signal.

I am in no way good at maths, just a decent problem solver sharing my experience with the SI5351.

It’s also to be noted, the PAL and JP Megadrive differ in some values for the sine wave conversion of the clocks square signal. In series through 10k , then 1k pulldown, past a puff rated cap with one end to GND and finally through a coupling cap into the encoder on a JP system. On my PAL system its the same but its through a 4k7 in series.

Looking at the CXA1145 datasheet though, we see it recommends a 1k series, 1k pulldown then coupling cap for either SubCarrier signal. You can bypass all this and input the square wave straight into the encoder and it still works for some reason. Either way, using my eyes on several sets, this is a win/ win situation.

Enjoy!

Lionel
VajskiDs Consoles 2024