Lab 2 – Using the Si5351 clock generator circuit with ItsyBitsy

Yifan Zhu
Lab Partner: John Kustin

February 22, 2022

Abstract

We use an ItsyBitsy to build a Si5352 clock generator circuit to generate waveforms at various frequencies.

1 Introduction

In various circuits we need "clock signals" at some precise frequency. A modern solution is to use clock generator ICs, which have advantages of being low-cost, having reasonable performance, and being able to generate signals at different frequencies with a single IC.

In this lab we will generate clock signals using the Si5351 IC, and we will send the control I^2C signals to the Si5351 using the ItsyBitsy microcontroller and CircuitPython.

2 Experimental Setup

We did the assembly following the instructions online at adafruit, while replacing the pinout for the Arduino microcontroller with that of the ItsyBitsy microcontroller:

1. We soldered pins onto the ItsyBitsy and the Si5351 breakout board.
2. We soldered SMA connectors onto the Si5351 breakout board. SMA connectors are needed to carry the generated clock signals that are essentially RF.
3. According to the documentation of Adafruit Si5351 Breakout Board and ItsyBitsy, using jumper wires we connected
   (a) the 3V pin (ItsyBitsy) to the Vin pin (Si5351 breakout),
   (b) the GND pin (ItsyBitsy) to the GND pin (Si5351 breakout),
   (c) the SCL pin (ItsyBitsy) to the SCL pin (Si5351 breakout),
   (d) and the SDA pin (ItsyBitsy) to the SDA pin (Si5351 breakout).

4. We updated the bootloader of the ItsyBitsy following the instructions. And we installed CircuitPython following the instructions here.
5. And the CircuitPython code is taken directly from the Adafruit website.

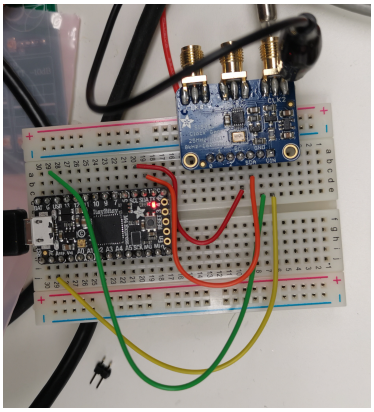
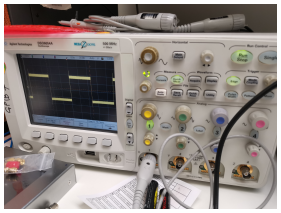


Figure 1: Si5351 Clock Generator controlled By ItsyBitsy

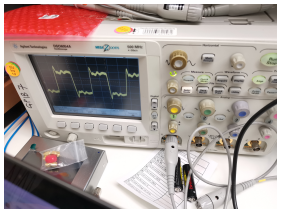
The assembled circuit is shown in Figure 1.

3 Results

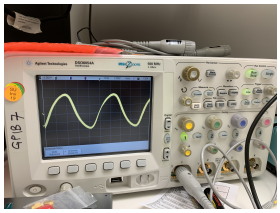
We generated three different signals at 10.706KHz (Figure 2a), 13.5531MHz (Figure 2b), and 112.5MHz (Figure 2c). The generated clock signals are shown in Figure 2.



(a) 10.706kHz generated clock signal



(b) 13.5531MHz generated clock signal



(c) 112.5MHz generated clock signal

Figure 2: Generated Clock Signals at Various Frequencies

4 Discussion

4.1 How are all these different frequencies generated?

The datasheet claims that the Si5351 can generate frequencies from 8 KHz to 160 MHz, a 20,000 times difference between the lowest frequency and the

highest frequency. In addition, the reference clock needed is only at 25MHz, more than six times lower than the highest generated frequency. How is all this accomplished?

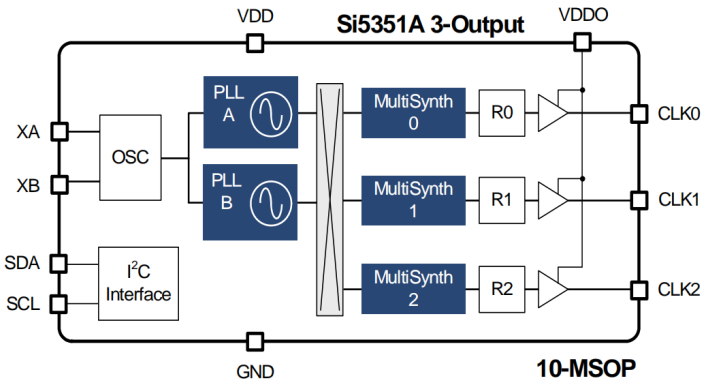


Figure 3: Detailed Block Diagram of Si5351A 3-Output

The detailed block diagram of Si5351 is shown in Figure 3. As we can see, internally, the input clock is first multiplied to high intermediate frequencies PLL A and PLL B using phase lock loops. This not only enables the IC to synthesize output frequencies higher than the input frequency, but also allows for better resolution in the output frequencies. The phase lock loops support both integer and fractional multiples of the input clock frequency. For our purposes, we configured PLL A to be 36 times the input clock (900MHz), and PLL B to be $24\frac{2}{3}$ times the input clock (616.6667MHz).

Then, the higher intermediate frequencies are passed through high resolution MultiSynth fractional dividers, which can divide the intermediate frequency by either integers or fractions. For our purposes, we configured clock 0 to be PLL A divided by 8 (112.5MHz = 900MHz / 8), clock 1 to be PLL B divided by $45\frac{1}{2}$ (13.5531MHz=616.6667MHz / 45.5).

Finally, an ouput R divider can divide the frequency further by $2^k, k \in \{0, ..., 7\}$, to get frequencies as low as 8 kHz. For our purposes, to obtain clock 2, we first used the MultiSynth to divide PLL B by 900, and then applied an additional R divider of 64 (10.706KHz = 616.6667MHz / 900 / 64).

In conclusion, by having one frequency-multiple stage and two frequency-divide stages, the Si5351 is able to generate three non-integer related output frequencies from 8KHz to 160MHz.

4.2 Why do the waveforms in for different frequencies look so different?

As we can see in Figure 2, the output for 10.7KHz (Figure 2a) looks very much like a square wave, the output for 13.5MHz (Figure 2b) has some ripples, while the output for 112.5MHz (Figure 2c) doesn't resemble a square wave at all.

This shows that as frequency goes up, the output waveforms are more "smoothed" from the ideal square wave. Mathematically, smoothing in the

time domain just corresponds to low-passing in the frequency domain. This is reasonable to expect, as all circuits have some capacitive parasitics, which can effectively act as low-pass filters at high-enough frequencies.

4.3 Quadrature Signals

Conveniently, the Si5351 offers configurable initial phase offset for all clock outputs. This means that for certain frequencies, we can connect two output clocks to the same PLL, and set their time delays to be different by exactly 1/4 the output period to generate a pair of I and Q clock signals.

However, this doesn't work for all output frequencies. The initial phase offset is specified as a 7-bit unsigned integer with one LSB equivalent to a time delay of Tvco/4, where Tvco is the period of the VCO/PLL associated with the output. This means that to get an output phase difference of exactly 90 degrees, the output frequency cannot be less than $1/2^7 = 1/128$ of the PLL. In addition, if we use fractional frequency division in the MultiSynth dividers, we may not be able to get exactly 90 degrees difference at the output.

5 Conclusions

In this lab, we demonstrated that a clock generator IC is easy to use and can generate a wide range of different output frequencies. We explored the implementation of Si5351 (Section 4.1), why the output waveforms are different at different frequencies (Section 4.2), and potential use of the IC to generate IQ signals (Section 4.3). This work can be extended by better understanding the limitations of the Si5351 IC, as well as exploring other potential applications.