Design and Implementation of a Frequency-Hopping 5.8 GHz Signal Generator

Gary Menezes
Daniel Smith
Adam Troyer
Chen Zhou

ECE 6361 – Summer 2012

DESIGN SPECIFICATIONS

The goal of this project was to design a programmable signal generator capable of generating a CW signal in the 5.8 GHz ISM band with FCC Part 15-compliant frequency hopping. The design needed to meet the following specifications:

- Operation from 5.725-5.850 GHz with no measurable signal out of band;
- 7 dBm (5 mW) of output power;
- Must hop between at least 75 frequency channels spaced at least 1 MHz apart;
- Maximum of 0.4s spent on any one carrier frequency during any 30s interval;
- Design must be contained on a single board, but may be driven by an external power supply and microcontroller.

The designed PCB must meet the following specifications, set by the capabilities of the Georgia Tech in-house milling machine:

- Two layers maximum;
- Standard FR4 material, with relative permittivity of 4.34 and loss tangent of 0.02;
- Board thickness of either 31 mils or 59 mils;
- >12 mils isolation between traces;
- >10 mils via diameter;
- Maximum PCB size of 8” x 11”;
- Minimum PCB size of 0.5” x 0.5”.

DESIGN OVERVIEW

Figure 1 shows a high-level diagram of the signal generator design.

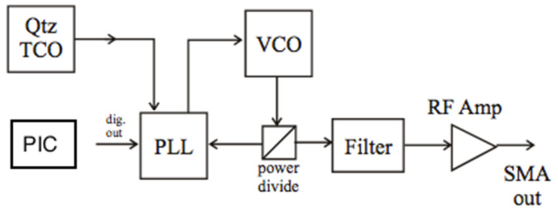


FIGURE 1. Signal generator overview.

The carrier wave will be generated by a voltage-controlled oscillator (VCO), which is controlled via a feedback loop with a phase-locked loop (PLL) chip. The frequency set-point of the PLL is programmable via SPI connection with a PIC microcontroller, which will provide the frequency-hopping functionality. A crystal oscillator provides a 10 MHz reference signal for the PLL. Finally, the VCO output will be filtered through a band-pass filter to remove any out-of-band components, and an RF amplifier will boost the signal to the desired 7dBm output power.

PLL DESIGN

The PLL was implemented using the reference circuit provided in the PLL datasheet, shown in Figure 2.

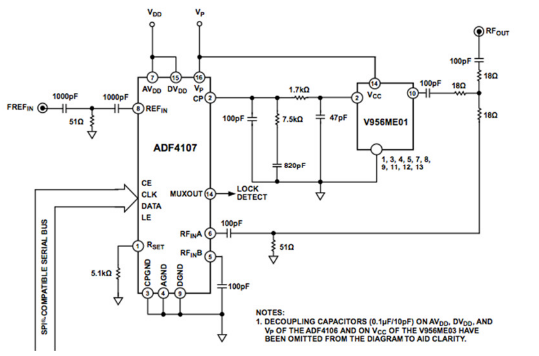


FIGURE 2. PLL reference circuit [1].

In PLL design, the critical component to design is the loop filter. There are two key parameters of the PLL that would have an impact on loop filter design: loop bandwidth and phase margin.

The loop bandwidth influences the lock time. In the project requirements, the maximum lock time is 0.4s. Based on this requirement, loop bandwidth is set to 50KHz. The simulation below indicates that the lock time is only 40us, which meets our requirement

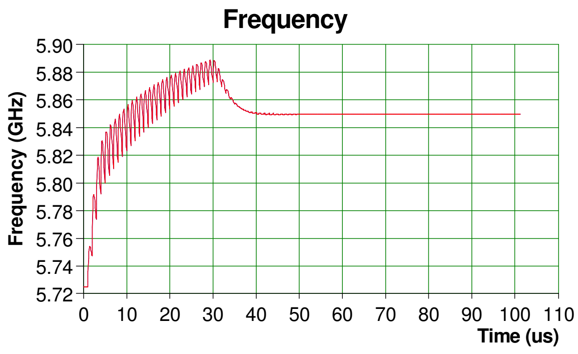


FIGURE 3. PLL lock time response.

In terms of phase margin, we set it to 60 degrees. Normally more than 45 degrees is enough is for PLL stability.

With the above design, we simulate the closed loop gain performance. As you can see, there is a little peaking in amplitude. But it's good enough to guarantee PLL stability.

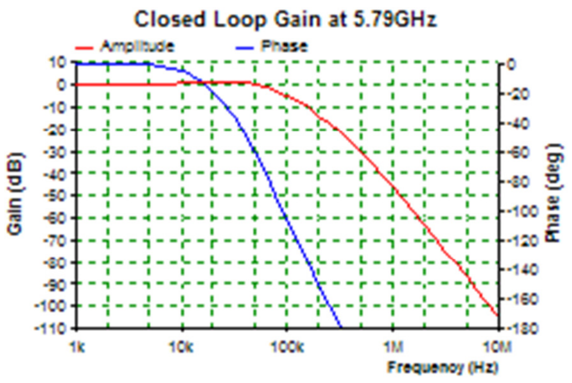


FIGURE 4. PLL closed-loop gain.

WILKINSON DIVIDER DESIGN

The PLL requires feedback from the VCO to match frequency, so the VCO output must be split. A Wilkinson divider was used for this purpose, as it provides a number of advantages over other splitting designs: it is (ideally) lossless in the forward direction, it is inherently matched at all ports (assuming an equal power split), and there is good isolation between the two output ports. Figure 5 shows the typical layout for a 3 dB Wilkinson divider.

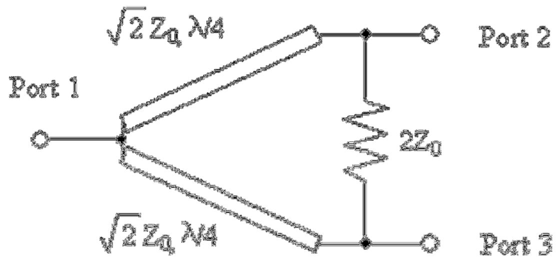


FIGURE 5. 3 dB Wilkinson divider schematic.

The divider was designed and tuned in ADS to obtain an S(2,1) and S(3,1) as close to 3 dB as possible at 5.8 GHz and matched to 50Ω. An equal power split was used for ease of design, and because we could determine no particular reason to use an uneven split. Figure 6 shows the final layout of the divider, and Table 2 gives the dimensions of the transmission lines involved.

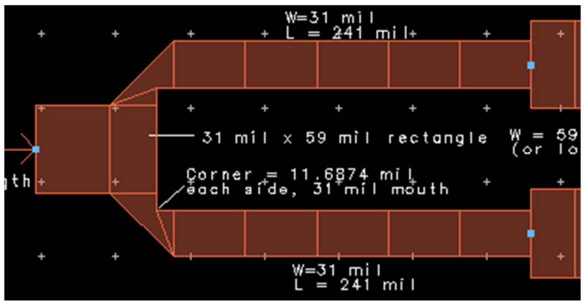


FIGURE 6. Wilkinson divider layout.

Table 2. Wilkinson Divider Dimensions

Parameter	Value
Input/Output line widths	59 mils
Leg widths	31 mils
Leg lengths	241 mils

Figure 7 shows the S(2,1) and S(3,1) response of this circuit as simulated in Momentum. The divider provides about -3.4 dB at both output ports over the entire 5.725-5.850 GHz frequency range.

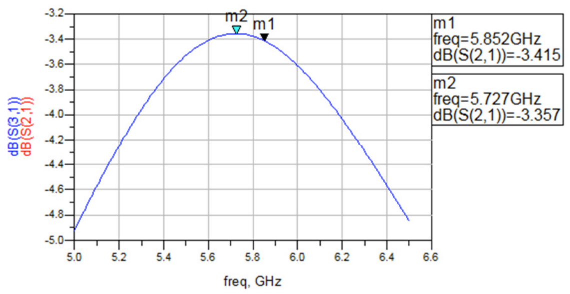


FIGURE 7. Wilkinson divider S(2,1) and S(3,1).

One output leg of the divider feeds the BPF and RF amplifier leading to the circuit output, while the other leg feeds back to the PLL reference input.

BAND-PASS FILTER DESIGN

The desired filter specifications are defined as follows:

- Center frequency (fc) = 5.7875 GHz;
- Bandwidth = 125 MHz;
- Maximally flat pass-band.

A coupled line design with a maximally-flat response is implemented so as to ensure ease of fabrication and a flat response over the desired frequency range. A low order (N=2) filter is used as higher order designs proved to be very lossy. The parameters for the design are listed in Table 3.

Table 3. Band-Pass Filter Parameters

n	g_n	Z_{even}	Z_{odd}
1	1.4142	53.51	46.49
2	1.4142	50.29	49.71
3	1	54.11	45.89

The circuit schematic and layout are shown in Figures 8 and 9 as designed and optimized in Agilent ADS.

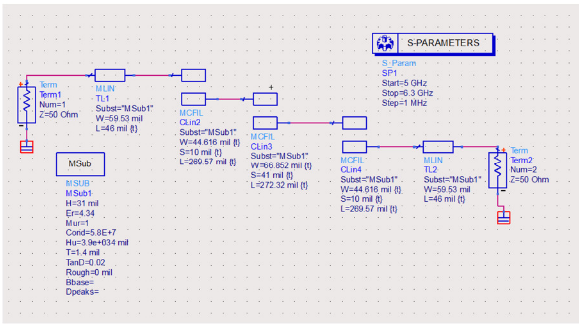


FIGURE 8. Band-pass filter schematic.

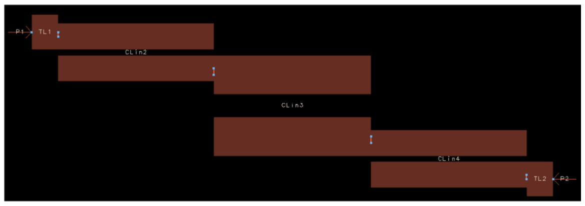


FIGURE 9. Band-pass filter layout.

The S-parameter results of the Momentum simulation of the filter are shown in Figure 10. The response shows a flat passband in the frequency range of interest, with an attenuation of about -4 dB. This loss will be offset by the following amplifier.

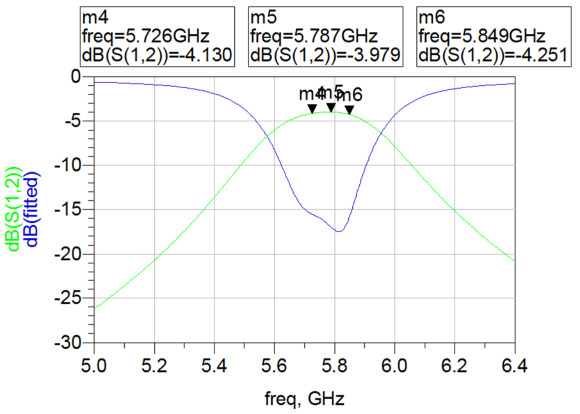


FIGURE 10. Band-pass filter performance in Momentum.

LINK BUDGET AND AMPLIFIER DESIGN

With the BPF designed, the Wilkinson divider and BPF were cascaded in ADS to simulate the total insertion loss from the divider input to the BPF output, in order to determine whether one amplifier would be sufficient to achieve the specified output power, or whether two would be necessary. Figure 11 shows the cascaded circuit layout, and Figure 12 shows the insertion loss through the circuit.

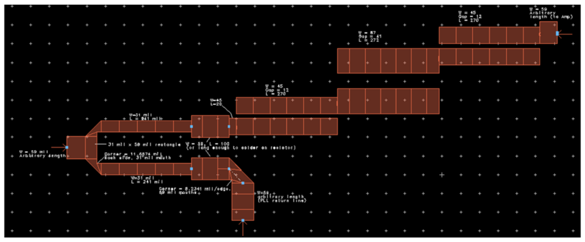


FIGURE 11. Divider + BPF layout.

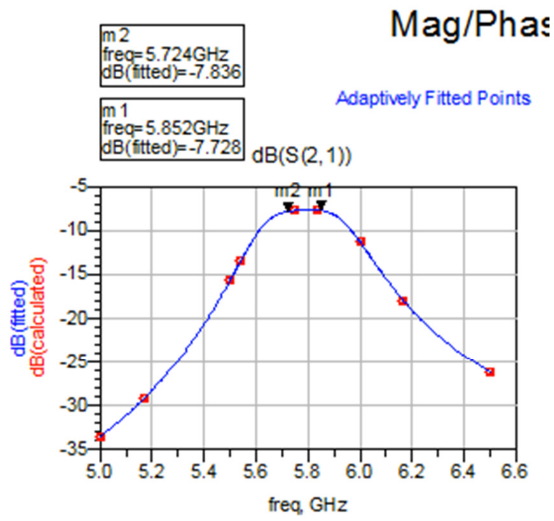


FIGURE 12. Divider + BPF simulated insertion loss.

Figure 12 shows approximately 8 dB of loss through the entire circuit. The VCO has a specified typical power output of 1.5 dBm, and the RF amplifier has a specified gain of about 17 dB. Table 4 details the link budget for this circuit, assuming a conservative 2 dB of extra loss from the VCO to the output to account for slight mismatches, dielectric loss due to FR-4's poor loss tangent, and conductor loss.

Table 4. Link Budget

Component	Gain
VCO output	+1.5 dBm
Divider+BPF	-8 dB
Extra losses	-2 dB
Amplifier	+17 dB
Output power	+8.5 dBm

Based on this calculation, one amplifier should be sufficient to provide the 7 dBm output power specified.

The amplifier was laid out and biased following the reference circuit provided in the amplifier datasheet [2] and shown in Figure 13.

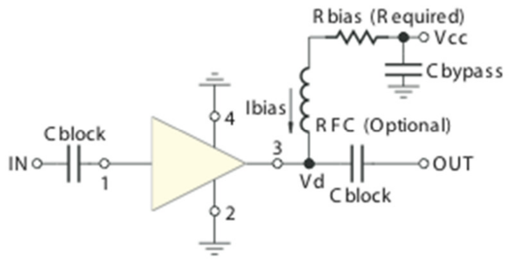


FIGURE 13. RF amplifier biasing circuit [2].

The bias resistor was chosen to be 162Ω based on the amplifier datasheet and our 9V supply voltage, and the blocking capacitors were chosen to be 10 nF to provide a low impedance (< 2Ω) at 5.8 GHz.

MICROCONTROLLER PROGRAMMING

A Microchip PIC184F2321 28-pin microcontroller was selected for the design to control the PLL and implement frequency hopping. This device was selected due to its available and ease of programming. The PIC was out on the same board and communicates with the PLL via an SPI interface and configures the frequency of operation by loading a series of 24 bit latches. With the correct latch configuration, the PLL will select the voltage for the VCO and lock to the chosen frequency.

The PIC was programmed with a list of 75 different frequencies in a random order. Every 0.25 seconds, the PIC would configure the PLL to hop to the next frequency in the list. After reaching the end of the list, it begins again at the start. This fulfills the frequency hopping requirement of the project.

PCB LAYOUT

The PCB layout for the signal generator was designed to be as simple as possible. The only required connectors are a 9V power connector and an SMA output for the signal. Optional programming and SPI headers are also available.

The layout was designed for a two layer board with minimal via holes. Most routing was done on the top layer with surface mount components, except for a few power traces on the bottom layer. The rest of the bottom layer was a single ground plane for both digital and RF.

The board was fabricated on 31 mil FR4 with 0.5 oz copper on each side. A milling machine etched out the circuitry and drilled holes, and a chemical process was used to plate via holes for improved grounding. The schematic and layout are both included in Appendix 1 and 2.

IMPLEMENTATION AND PERFORMANCE

The final signal generator was a complete success and fully compliant. It hops to all 75 channels randomly and outputs about

8.6 dBm, though the output power does vary depending on frequency.

Figure 14 shows an image from a spectrum analyzer connected to the output of the signal generator when the generator was locked to a single frequency. There is only a single continuous wave signal with no spurious emissions.

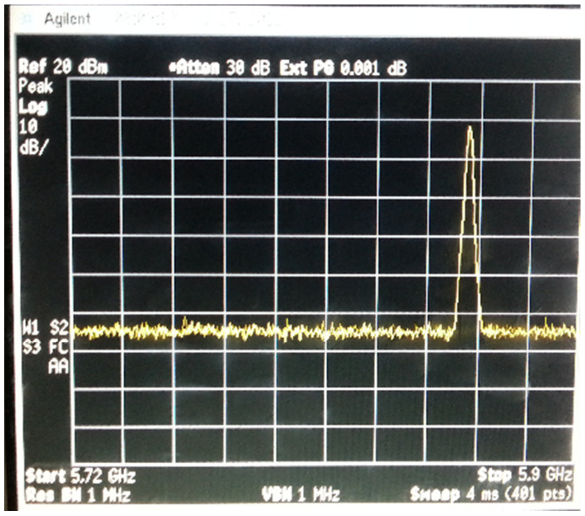


FIGURE 14. Signal generator output on spectrum analyzer.

Figure 15 shows the same signal on the spectrum analyzer with a 5 MHz span and 10 kHz resolution bandwidth. It can be seen that the first harmonic is 1 MHz from the center carrier and is 40 dB below the carrier in power, which is more than acceptable for a communication system.

Figure 16 shows a picture of the final system connected to a spectrum analyzer.

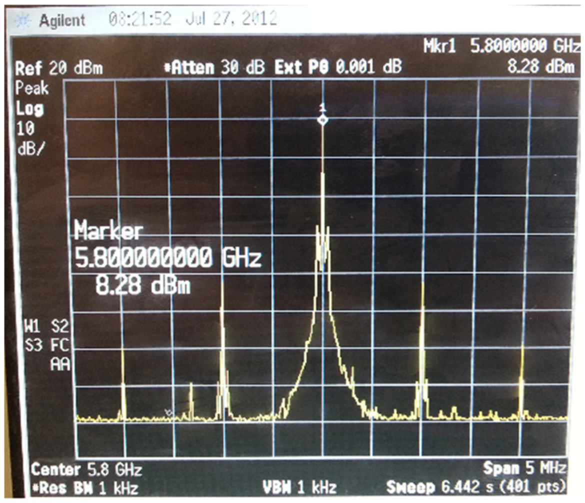


FIGURE 15. Signal generator output on spectrum analyzer with 5 MHz span.

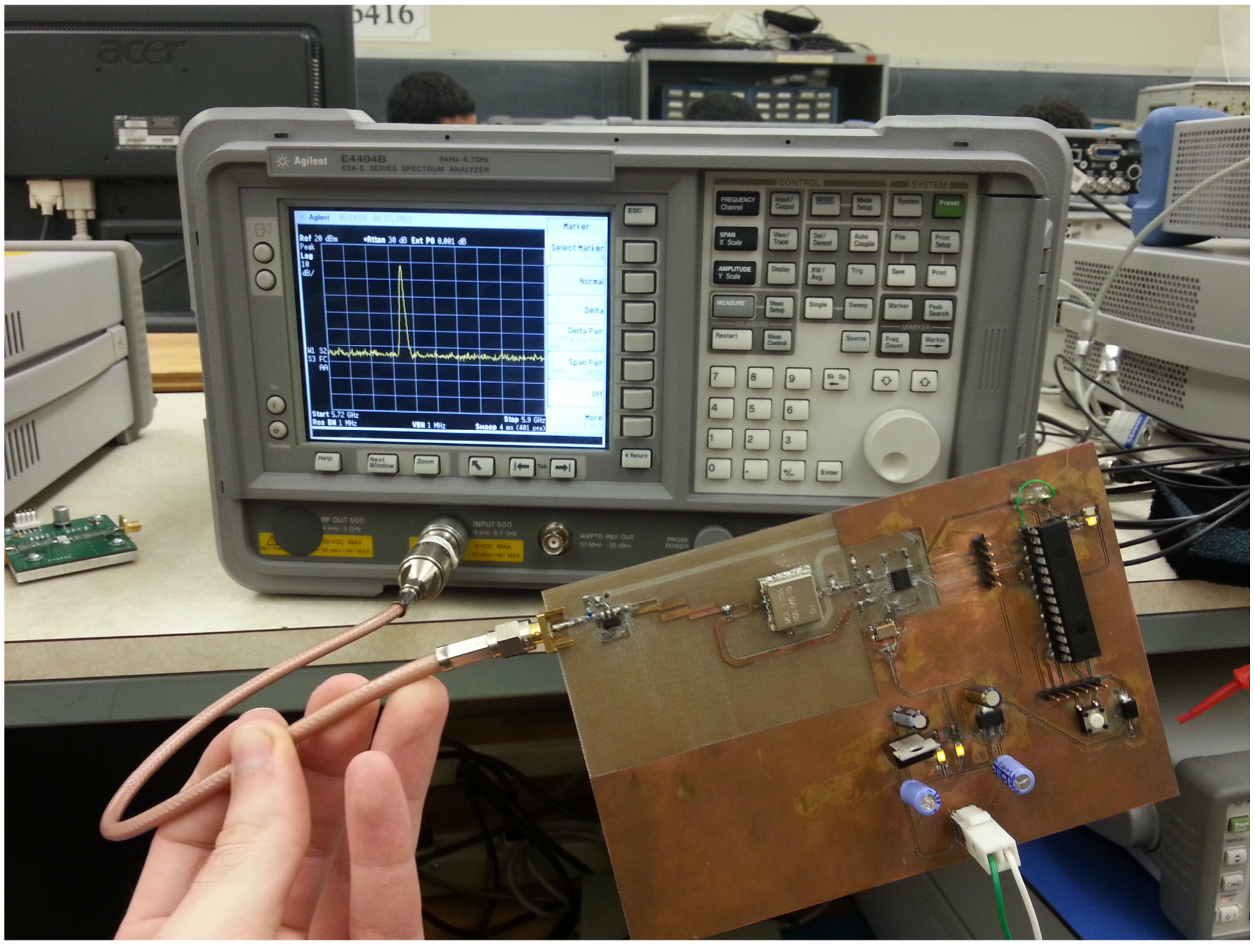
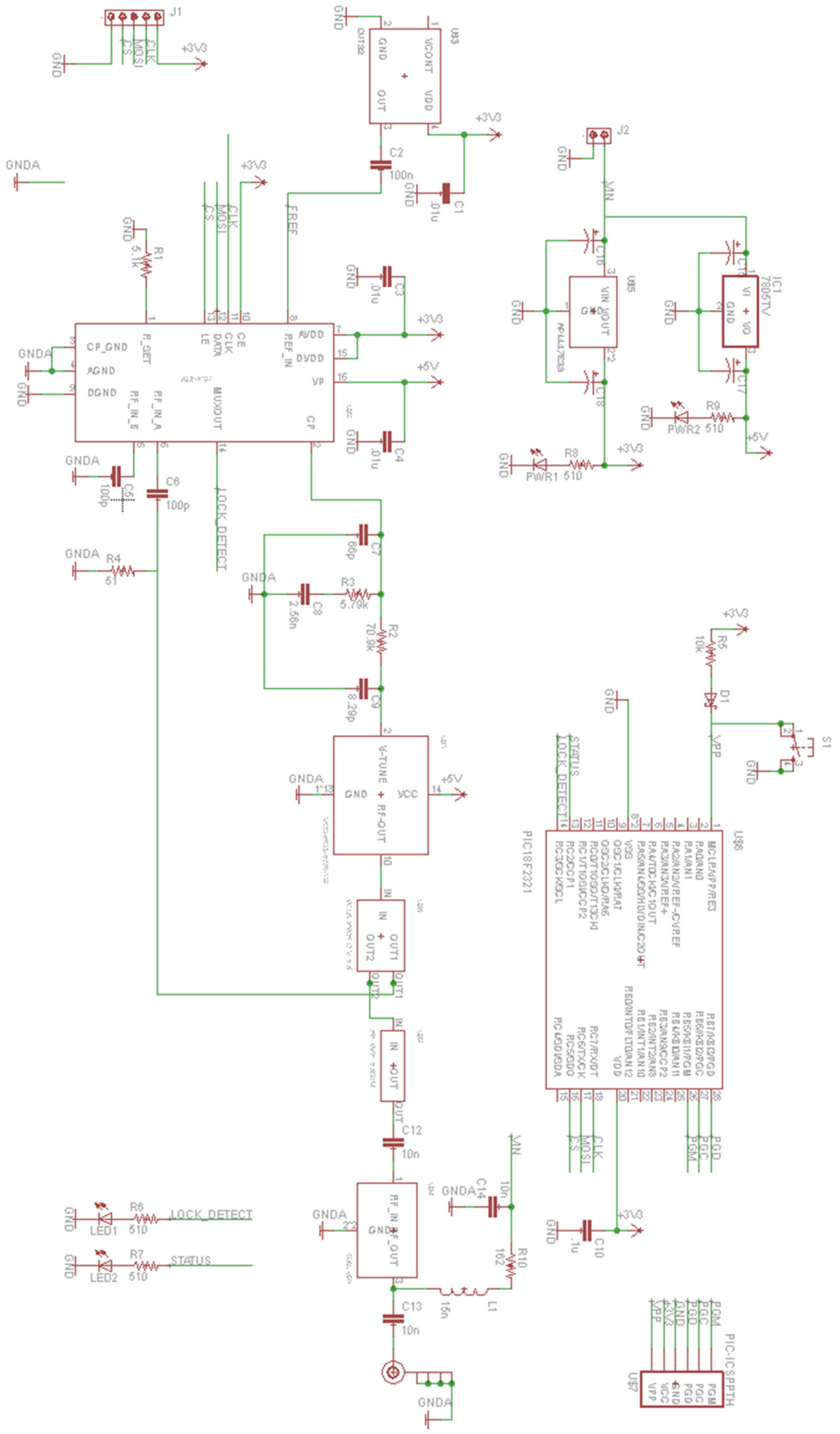


FIGURE 16. Final signal generator design connect to spectrum analyzer.

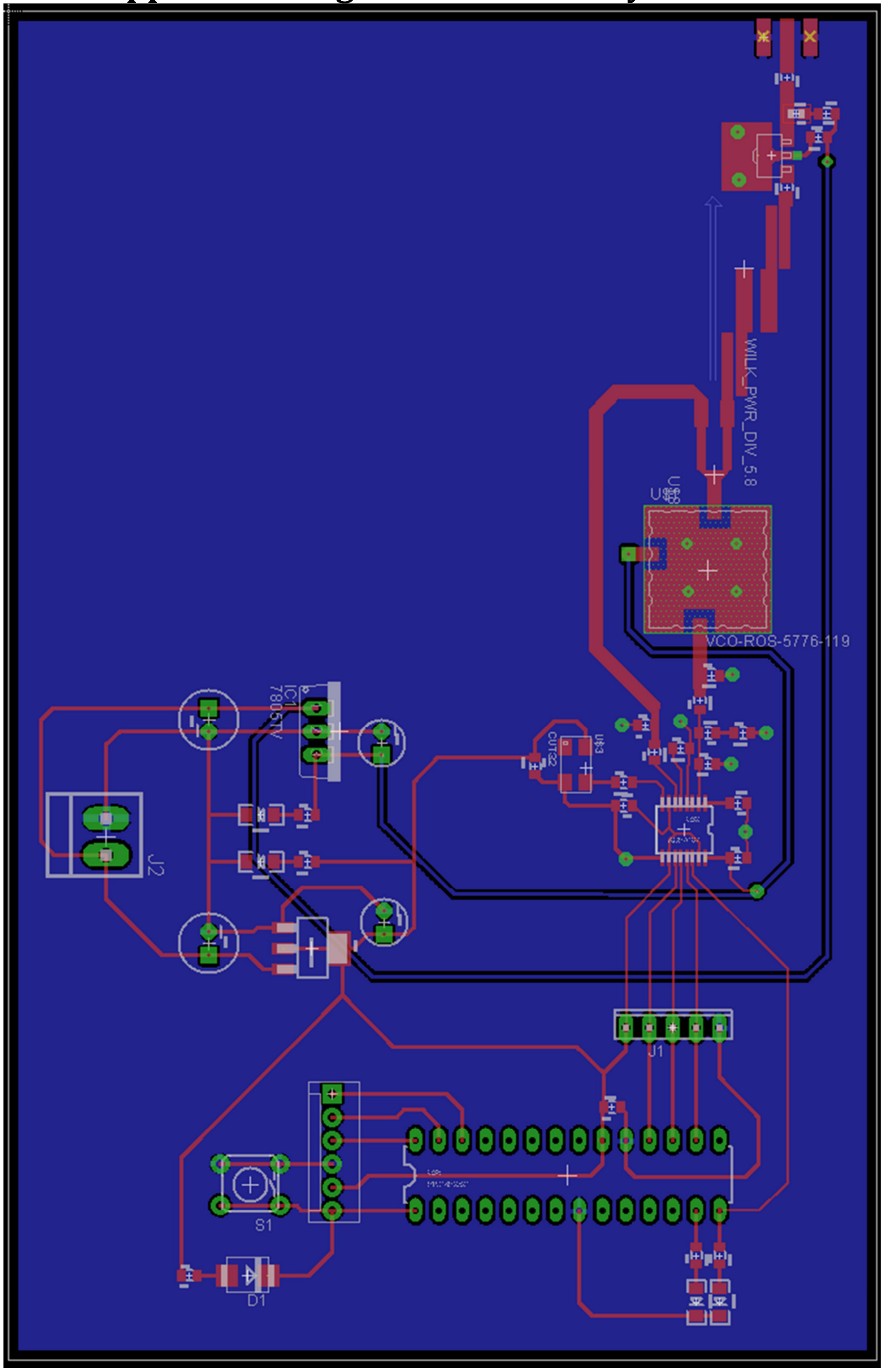
References

1. Analog Devices, *ADF4107 PLL Frequency Synthesizer Datasheet*, http://www.analog.com/static/imported-files/data_sheets/ADF4107.pdf.
2. Mini-Circuits, *Gali-39+ Surface Mount Monolithic Amplifier Datasheet*, http://www.minicircuits.com/pdfs/GALI-39+.pdf.

Appendix 1: Signal Generator Schematic



Appendix 2: Signal Generator Layout



Appendix 3: PIC Source Code

```
#include <p18f2321.h>
#include <timers.h>
#include "config_bits.h"
#include "init.h"

#define STATUS_LED PORTCbits.RC2
#define CS PORTCbits.RC6

#define PIN PORTAbits.RA0
#define PIN_PULSE PIN = 1; PIN = 0;

#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
    _asm GOTO HiPriISR _endasm
}
#pragma code
#pragma interrupt HiPriISR


unsigned long buffer;       // buffer for sending data over SPI
unsigned int timer_count;   // used for measuring time


#define NUM_CHANNELS 75
unsigned char channel_sequence[] = {47, 103, 118, 86, 62, 43, 28, 136, 68, 56, 131, 149, 139, 20,
142, 107, 51, 93, 130, 7, 116, 58, 133, 48, 121, 21, 141, 63, 82, 64, 5, 140, 65, 33, 24, 78,
106, 135, 98, 52, 71, 125, 15, 73, 69, 99, 13, 44, 22, 35, 87, 143, 128, 122, 66, 1, 41, 61, 31,
10, 120, 102, 59, 119, 137, 76, 67, 127, 38, 34, 147, 70, 91, 81, 11};
int current_channel=0;
int cFreq=5725;



void main() {

    Initial();

    STATUS_LED = 1;

    // Timer 1: trigger every 0.4 ms
    // Register R/W in one 16-bit operation, 1:8 Prescale
    T1CON = 0b10110000;

    PIE1bits.TMR1IE = 1;        // Timer1 overflow interrupt enabled
    IPR1bits.TMR1IP = 1;        // Timer1 gets high priority interrupts
    RCONbits.IPEN = 1;          // Enable high/low priority interrupt feature
    INTCONbits.GIEH = 1;         // Enable both high priority interrupts
    PIR1bits.TMR1IF = 0;         // Clear interrupt flag

    T1CONbits.TMR1ON = 1;       // enable Timer 1;
    timer_count=0;

    CS = 1; // set chip select high to start

    // INITIALIZE SPI as MASTER
    SSPBUF = 0;
    SSPSTAT = 0b01000000;
    SSPCON1 = 0b00100000;

    while (1)
    {
        while (timer_count <1) { } // wait for timer (every .5 seconds)
        timer_count = 0;

        PIN_PULSE;

        // send the next frequency in the sequence to the PLL

        sendPLL(channel_sequence[current_channel]+5725);
        STATUS_LED = !STATUS_LED;
        current_channel++;
        if (current_channel >= NUM_CHANNELS)
            current_channel = 0;

    }
}
```

```
void SPI_send()
{
    // send 24 bits (3 bytes) to PLL

    unsigned char toSend;

    CS = 0;            // Chip select low to start SPI (LE on PLL)

        toSend = (buffer >> 16) & 0xFF;
        PIR1bits.SSPIF = 0;         // Clear SPI flag
        SSPBUF = toSend;            // Send byte
        while (!PIR1bits.SSPIF);   // Wait for transmission to complete

        toSend = (buffer >> 8) & 0xFF;
        PIR1bits.SSPIF = 0;         // Clear SPI flag
        SSPBUF = toSend;            // Send byte
        while (!PIR1bits.SSPIF);   // Wait for transmission to complete

        toSend = buffer  & 0xFF;
        PIR1bits.SSPIF = 0;         // Clear SPI flag
        SSPBUF = toSend;            // Send byte
        while (!PIR1bits.SSPIF);   // Wait for transmission to complete


    CS = 1;            // Chip select goes high (LE on PLL)
}

void sendPLL(int freq)
{
    // change the PLL frequency
    // range is 5725 to 5875 (in MHz)

    unsigned int B=89;  // A counter value for 5725 MHz
    unsigned char A=29; // B counter value for 5725 MHz (6 bits)
    int k;  // used for calculations

    if (freq < 5725) freq = 5725;
    if (freq > 5875) freq = 5875;

    // maths to calculate A and B values
    // R is 20.  using a base frequency of 5725 MHz
    k = freq - 5725 + A;   // number of "B counts" we need.  64 "B counts" = 1 "A count"
    B += k >> 6;            // divide by 64 to get number of "B counts"
    A = k & 0x3F;           // the remainder (lower 6 bits) is the number of "A Counts"

    buffer = (unsigned long)0xDF8096;   // function latch
    SPI_send();

    buffer = (unsigned long)0x000050;   // R counter
    SPI_send();

    buffer = (unsigned long)((B << 8) + (A << 2) + 1);  // N counter latch
    SPI_send();

    buffer = (unsigned long)0xDF8092;       // function latch
    SPI_send();
}


 void HiPriISR()
{
   // triggers on timer overflow (should be every .25 seconds)

   //WriteTimer1(65055);       // Preload timer value to shorten period
   PIR1bits.TMR1IF = 0;         // Clear interrupt flag
   timer_count++;
}
```