5.8 GHz Signal Generator

Mitch Costley, Sen-wen Hsiao, Wasif Khan, and Mehdi Kiani

I. Introduction

Creating an FCC-compliant signal generator for microwave frequencies involves several steps and components. In this project, a CW signal generator for 5.8 GHz was designed, simulated piece-by-piece, fabricated, and tested. The goal was to produce a 7 dBm signal in the bandwidth 5.725-5.850 GHz wherein the signal frequency is incremented by 1 MHz every 0.4 seconds to comply with FCC part 15 regulations.

II. Circuit Design

A. Overview

The system block diagram is given in Figure 1. The PIC microcontroller programs the PLL (phase-locked loop) to lock at whatever is the current frequency in the target bandwidth. The PLL, which is referenced by the crystal oscillator (shown amplified here but ultimately directly connected), sends a tuning voltage to the VCO (voltage-controlled oscillator) to generate the RF signal. The power divider then sends half the generated power back to the PLL to provide feedback, the other half being filtered and amplified as the output signal. The mixer shown following amplification was considered for use in creating power-optimized waveforms, but this was later not implemented due to the extra complexity and the required quick turn-around.

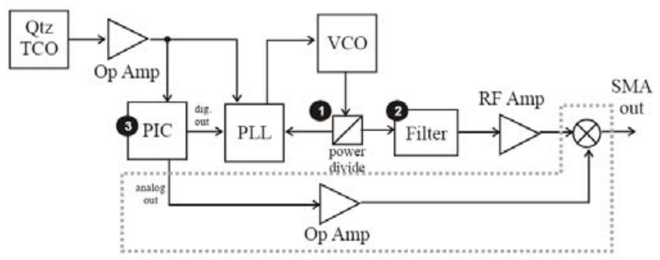


Fig. 1. System block diagram.

B. Wilkinson Power Divider

The equal power divider was designed with the topology and methodology given in [1]. A circuit schematic is shown in Figure 2. Substrate and operating parameters are given in Table 1.

The impedances of the two quarter-wavelength sections were calculated by

$$z_{0A} = z_{0B} = z_0\sqrt{2} = 70.7\Omega .$$

For the equal-split divider, the value of resistor R_W can be calculated as

$$R_W = 2z_0 = 100\Omega .$$

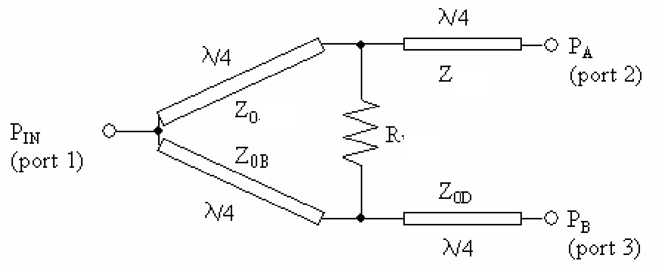


Fig. 2. Wilkinson power divider topology (Image courtesy of www.microwaves101.com).

The two quarter-wavelength transformers preceding ports 2 and 3 in Figure 2 were unnecessary in this application because the equal-split divider has 50 Ω output at each of those ports. The widths of the 50 Ω and 70.7 Ω microstrip line segments were determined by ADS LineCalc to be about 121 and 63 mil, respectively.

Table 1. Substrate and Operating Parameters

Parameter	Value
Relative permittivity	4.2
Substrate thickness	62 mil
Metallization thickness	1mil
Loss tangent	0.0009
Wavelength	1007 mil
z_0	50 Ω
σ	$5.813\text{x}10^7$ S/m

Before constructing a simulation schematic, the layout of the power divider was considered. Rather than arranging the output ports to be adjacent to each other, they were deemed better placed on opposite sides of the power divider since one would lead to other RF components while one would return to the digital PLL as part of the feedback loop. Coupling between digital and RF signals and unnecessary lengths of transmission line were reduced in this configuration.

Separating the quarter-wavelength sections in this way made the placement of the 100 Ω resistor more difficult. To keep package size small (limiting unwanted inductance), two half-wavelength transmission line sections were added connecting the resistor leads to ports 2 and 3. In terms of the standing wave pattern, this is equivalent to directly connecting the resistor leads, but it allowed for the necessary geometry to be realized on the physical board. The layout is given in Figure 3.

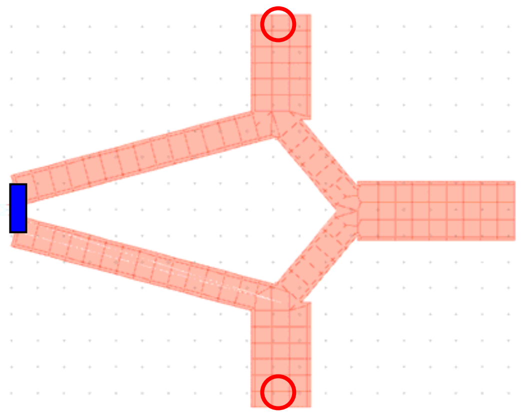


Fig. 3. Wilkinson power divider layout. The blue circle indicates the placement of the 100 Ω resistor and the red circles the locations of ports 2 and 3.

Generating a schematic from the layout, a genetic optimization was performed to find the critical microstrip lengths and widths yielding s_{21} and s_{31} nearest as possible to 3 dB and minimum s_{11}. The *s*-parameter results from the Momentum (method of moments) simulation are shown in Figure 4 with the corresponding dimensions given in Table 2. Simulations predicted output power to be about -3.5 dB with s_{11} about -25 dB.

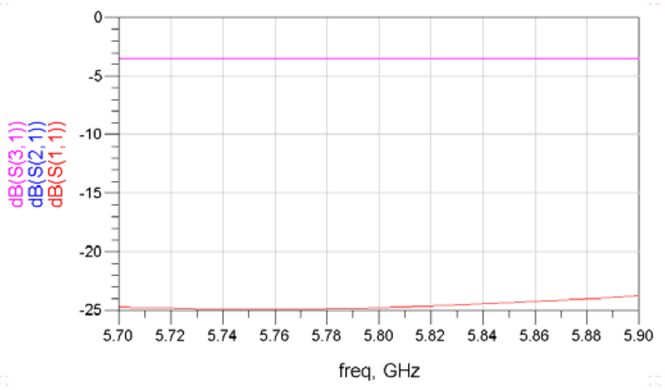


Fig. 4. Momentum simulation result for power divider using Table 2 dimensions.

Table 2. Power Divider Optimal Microstrip Dimensions

Dimension	Value
λ/4 Length	228 mil
λ/4 Width	60 mil
λ/2 Length	526 mil
λ/2 Width	60 mil

C. *Bandpass Filter*

A coupled line design was chosen for the bandpass filter due to its ease of implementation at 5.8 GHz and wide bandwidth. The design methodology and topology given in [1] was used. A maximally flat filter was chosen to provide consistent passband gain across the bandwidth. Strong attenuation near the passband was not needed because no signals were expected to be generated in that range if the PLL was locked. The number of resonators was fixed at 4 to provide wide bandwidth without the additional attenuation of more resonators. The equations below were used to determine the even and odd characteristic impedances of the coupled sections (where gk are the filter prototype values for a 3rd order maximally flat lowpass filter, available in the literature).

$$Z_o J_1 = \sqrt{\frac{\pi D}{2g_1}}$$

$$Z_o J_n = \frac{\pi D}{2\sqrt{g_{n-1}g_n}}, n = 2,3$$

$$Z_o J_{N+1} = \sqrt{\frac{\pi D}{2g_N g_{N+1}}}$$

$$Z_{oe} = Z_o\left[1 + JZ_o + \left(JZ_o\right)^2\right]$$

$$Z_{oo} = Z_o\left[1 - JZ_o + \left(JZ_o\right)^2\right]$$

Again, a genetic optimization was performed to optimize the performance of the filter based on the critical dimensions of each resonator pair (length, line width, and spacing). S-parameter results from the Momentum simulation of the optimized layout are shown in Figure 5.

The simulation results indicated quite high performance, although less was expected after the circuit was fabricated. Based on this simulation data, the layout shown in Figure 6 was chosen. Relevant dimensions are given in Table 3, where *w* indicates resonator line width, *l* resonator length, *s* the spacing between lines in the resonator, and the subscripts the resonators each dimension refers to (i.e. “14” indicates the 1st and 4th resonators).

D. *PCB Design*

The PIC44 connector board designed by the Georgia Tech IEEE Hardware team was used as the microcontroller interface. All relevant schematics and part lists can be found at [2].

The schematic for the additional PCB designed for this project is available in Appendix A. The power divider and bandpass filter layouts were exported from ADS to Allegro to integrate with the rest of the PCB layout. 3.3 volt and 5 volt regulators are used as the supply voltage for different parts. These were placed as far from the RF components as possible to avoid RF noise in the power supplies and digital or analog

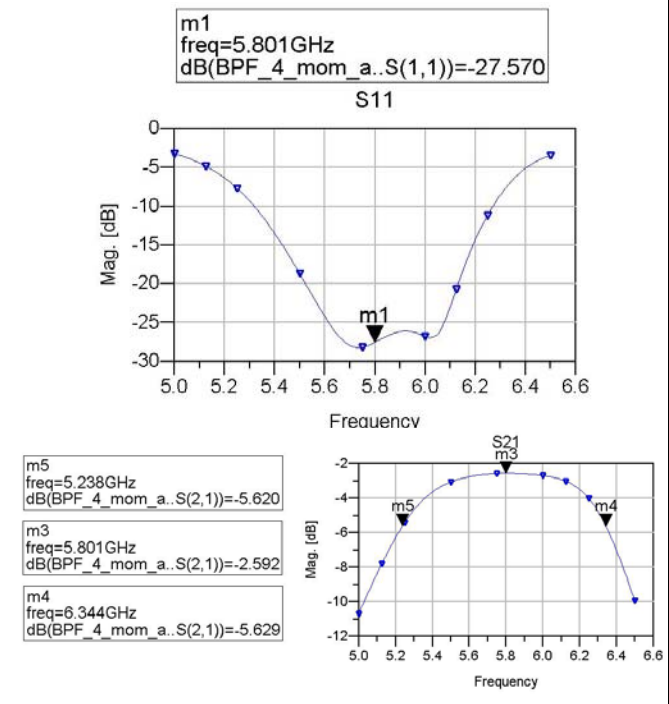


Fig. 5. Momentum results for BPF s_{11} and s_{21}.

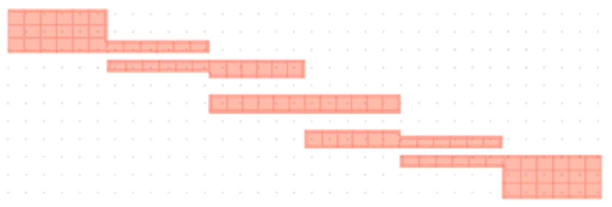


Fig. 6. Bandpass filter PCB layout.

Table 3. Bandpass Filter Critical Dimensions

Dimension	Value (mil)
$w_{50\Omega}$	121
w_{14}	34
s_{14}	20
l_{14}	285
w_{23}	50
s_{23}	48
l_{23}	268

noise in the RF signals. At the output of the crystal oscillator, an op-amp was used as a buffer. In measurement it was seen that the output of the crystal oscillator is enough to drive the PLL chip, and the op-amp was therefore not used. A 3rd order loop filter was designed at the output of the PLL chip. Two RF amplifiers were used, one of them connected directly to the output of the filter. A stand alone amplifier circuit was isolated to use as another gain stage, which can be seen in the schematic. The final PCB layout is available in Appendix B.

E. Loop Filter Design

The goals in the design of this PLL frequency synthesizer were to achieve low phase noise, low spurious output and to step, or hop, between 75 channels 1 MHz wide, switching every 0.4 seconds. The loop filter is important for phase locking. Figure 7 shows the circuit diagram of the loop filter and Figure 8 the schematic diagram in ADS.

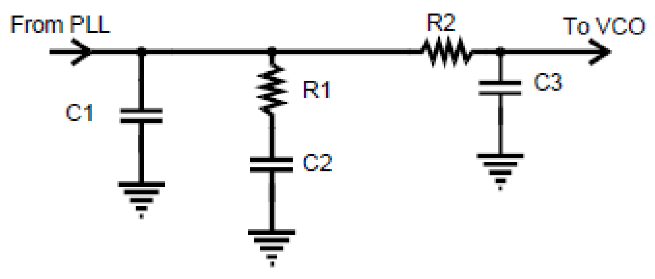


Fig. 7. Schematic of loop filter.

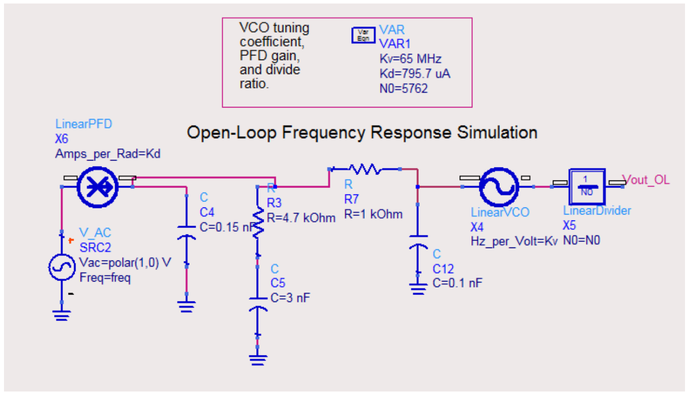


Fig. 8. ADS schematic of loop filter.

The loop filter is needed to remove any unwanted high frequency components which might pass out of the phase detector and appear in the VCO tune line. They would then appear on the output of the VCO as spurious signals. The filter affects the ability of the loop to change frequencies quickly. With very low cutoff frequency, changes in tune voltage will only take place slowly, while higher cutoff frequency results in quicker frequency changes.

The phase margin of the loop filter is very important as it determines stability. The phase margin should be greater than 45° for stable operation. If the phase margin is too large, it slows down the frequency locking process. Therefore, a balanced value is required.

The following steps were taken in the loop filter design process:

1. Determine the maximum dividing ratio N.
$$N = (\text{Max VCO Freq}) / (\text{Channel Spacing})$$
2. Calculate natural frequency.
$$f_n = \left(-\frac{1}{2\pi t_s \zeta} \right) \ln \left(\frac{f_a}{f_{step}} \right)$$
3. Calculate capacitor C2.
$$C_2 = \frac{I_{cp} K_{VCO}}{(2\pi f_n N)^2}$$

4. Calculate resistor R1.

$$R_1 = 2\zeta\sqrt{\frac{N}{I_{cp}K_{VCO}C_2}}$$

5. Calculate C1.

$$C_1 = 0.1C_2$$

6. Calculate R2 and C3.

R2 and C3 are used to reduce any spurs cause by the reference frequency. The product of R2 and C3 should be at least 1/10 the product of C2 and R1. Table 4 shows the values for components of the loop filter.

Table 4. Loop Filter Component Values

Loop Filter Component	Optimized Value
N	5762
C2	3nF
R1	4.7k
C1	.15nF
R2	1k
C3	0.1nF

Figures 9 and 10 show the output voltage and phase responses of the loop filter respectively. Bandwidth is 38.2 kHz and the phase margin 57.6°, which meet the criteria for the stability of the PLL.

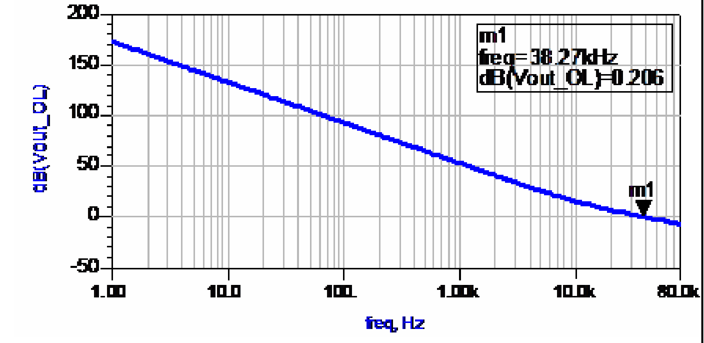


Fig. 9. Loop filter output voltage.

F. PLL Programming

The microcontroller used here is the 18F4520, one of the PIC microcontroller families. The PIC was to provide necessary data for the PLL in order to perform frequency hopping. Based on the specifications, the operation frequency includes the 5.725 – 5.850 GHz ISM (unlicensed) band. There are 75 channels spaced 1 MHz apart. Also, a maximum 0.4 s dwell time on 1 carrier frequency is tolerated during a 30s interval. Based on the PLL chip, there are three signals that are needed for PLL programming named latch enable (LE), data, and clock. The timing diagram is shown in Figure 11.

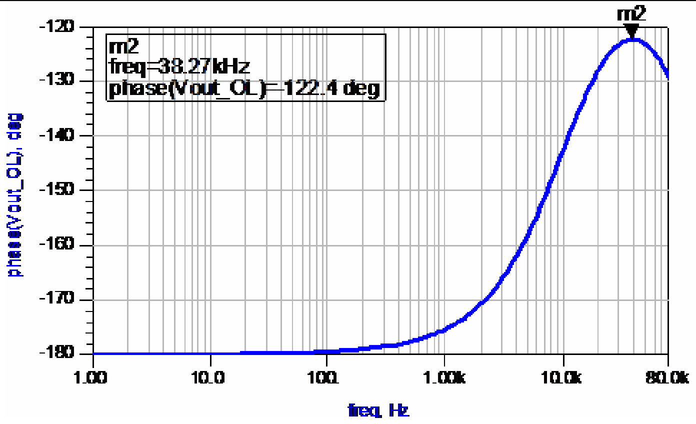


Fig. 10. Loop filter phase response.

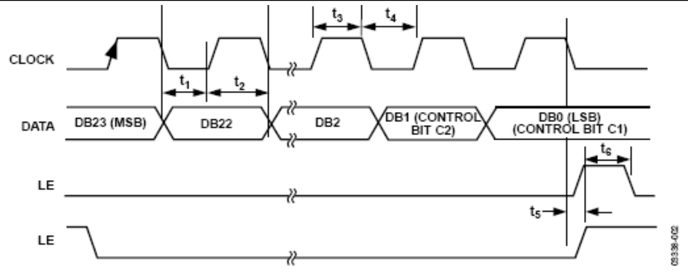


Fig. 11. Timing diagram for PLL programming.

The required data is sent in serial (SPI) to the PLL with the LSB bit sent first. Programming the PLL consists of 4 steps:

1. Do a function latch load (10 in two LSBs). As part of this, load 1 to the F1 bit. This enables the counter reset.
2. Do an R counter load (00 in two LSBs).
3. Do an AB counter load (01 in two LSBs).
4. Do a function latch load (10 in two LSBs). As part of this, load 0 to the F1 bit. This disables the counter reset.

Therefore the code should provide the values of the function latch, R counter and AB counter. The crystal oscillator frequency is 20 MHz, but the reference frequency for the PLL was chosen to be 1 MHz because in the integer-N synthesizer the output frequency step is equal to the reference signal. As the frequency step at the output should be 1 MHz, this was chosen as the reference signal. A divider was needed to divide the crystal oscillator frequency by 20.

The value of the function latch and reference latch are always constant because for frequency hopping only the AB counter value was changed. Figure 12 shows the divider block diagram. The prescaler value was selected to be 64 i.e. p=64. By having B=89 and A=29 the divider value will be N=BP+A=5725. By considering the 1 MHz reference frequency the output frequency will be 5.725 GHz. To output the next channel the divider number should be incremented by one. Therefore the A counter will change from 29 to its maximum of 63 and provide 35 channels. After that the B counter becomes 90 and the A starts from 0 to 39 to provide the other 40 channels. By knowing these values the R and AB latch values have been determined.

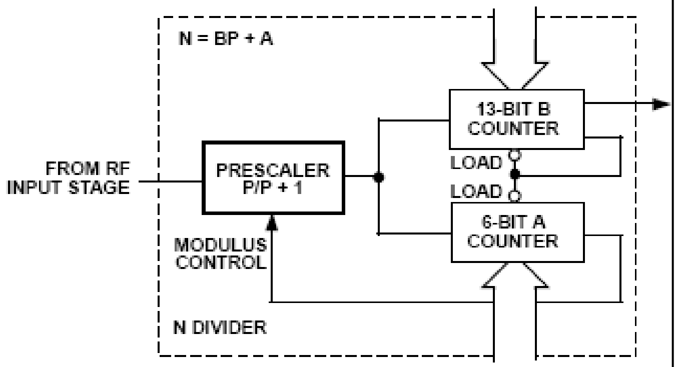


Fig. 12. Frequency divider block diagram. (PLL datasheet)

In the AB counter latch the first two bits (LSB bits) are devoted to the control bits, and the next 6 bits are A values and the other 13 bits B values as shown in the PLL datasheet. Therefore when the value of A increases for one step then the value of the AB register should change by 4 steps. By this method the different values for the AB counter were determined and are generated in the code.

G. *PIC Code*

The algorithm of frequency hopping on which the code is based is shown in Figure 13. A full code listing is provided in Appendix C. Port B of the PIC is used for communication with the PLL. The PortB.0 is devoted to the clock, PortB.1 is for data, and PortB.1 is devoted to the LE for the PLL. In the main code there is a loop which always calls the function program_pll() (In the header file the functions are defined along with the different values for the function latch, R counter latch and AB counter latch). In this function the PLL is programmed 75 times, after which it exits, returning to the main program. Thus, frequency hopping is executed indefinitely. In the program_pll() function, an array named pll_bytes is defined containing the different values of the R, AB counters, and function latch. There is a loop in which each bit is determined, and the data is clocked into the PLL. After this loop sending the necessary data of the function latch, R counter latch, AB counter latch and again the function latch, another loop is used to provide a delay of 0.4 seconds.

In the function latch of the PLL there are 3 bits named the "Mux out control." These bits were set to serial data output so that when data is received by the PLL, it can be visually monitored on an oscilloscope. In measurement the data was visible every 0.4 seconds, meaning the PLL was successfully programmed.

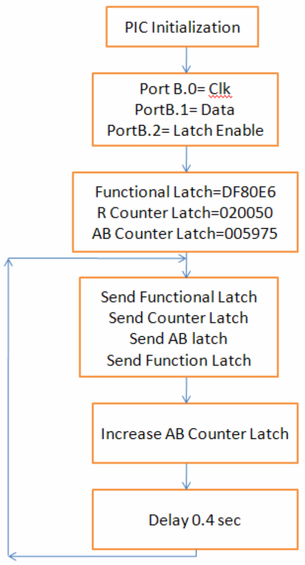


Fig. 13. Block diagram of frequency hopping PIC code.

III. System Testing

The microcontroller was tested to make sure the code was working properly. Figure 14 shows the latch enable output of the PIC microcontroller. The LE, which is normally high, should be zero every 0.4 seconds to send data. In this figure it can be seen that every 0.4 second latch enable becomes zero for a short time.

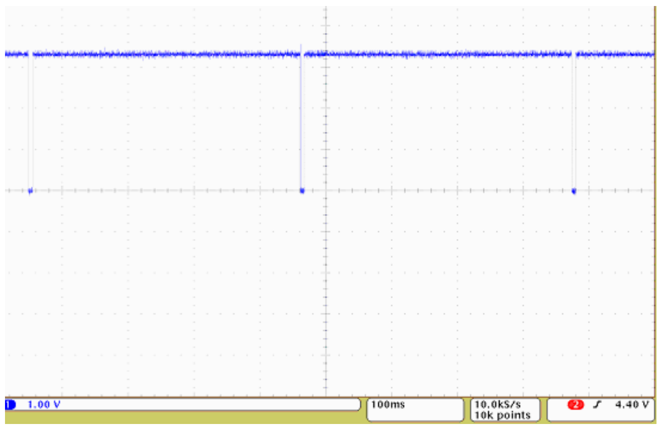


Fig. 14. Signal observed on LE pin.

The clock output of the microcontroller is shown in Figure 15. Based on this figure every 0.4 second there are some clock signals for sending data, then nothing. These signals show that the code is correct and the microcontroller is working well.

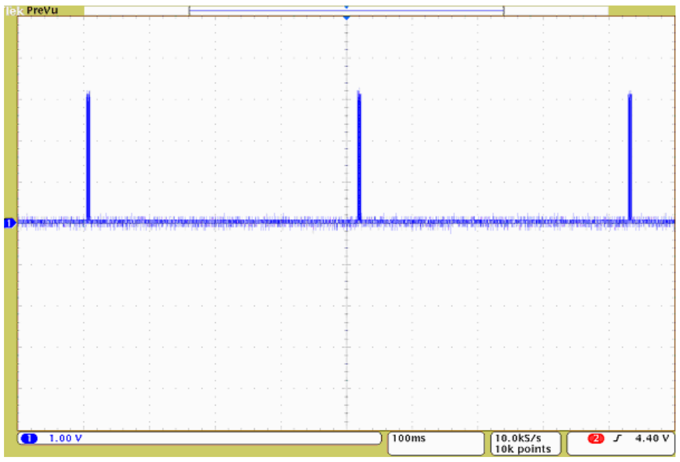


Fig. 15. Microcontroller clock output.

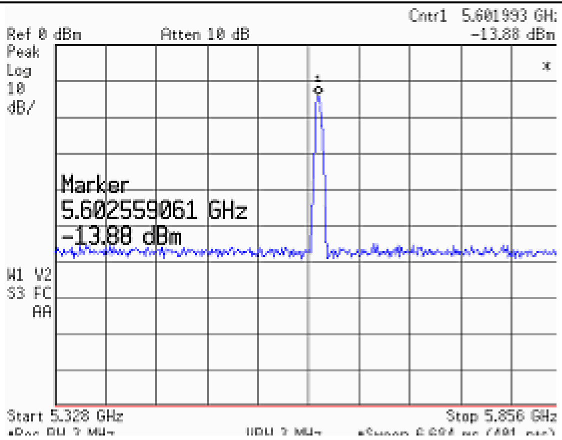


Fig. 16. VCO free running frequency as seen on spectrum analyzer.

The following steps were taken in testing the signal generator:

- The output of PIC microcontroller was tested on oscilloscope. The correct signals for clock, latch enable and data were observed. Clock and latch enable signals were writing the data in PLL after 0.4 seconds.
- VCO was working correctly; the free running frequency was observed on the spectrum analyzer shown in Figure 16.
- Locking and frequency hopping were seen for some time, but then some problem occurred.
- The PLL was programmed using three different methods suggested in the PLL data sheet.
- The PLL chip was replaced due to possible damage from the signal provided by the powerful crystal oscillator.
- An external ref-in signal generator with low voltage values was used to ensure that PLL ref-in was getting the proper voltage.
- For testing, data at MUX out was sent. The correct data was received, meaning the PLL was correctly programmed.
- Another loop filter with better phase margin was implemented to ensure if there was any instability because of inadequate phase margin. A phase margin of about 57° was achieved.

IV. Conclusions

The PLL can accept a signal at the RF-IN port with minimum strength -5 dBm. Our output after the power divider was -15 dBm. Thus, the signal with -15 dBm strength would be too weak for the PLL to read. There may be some problem in the board because of which the signal with strength better than -5 dBm was not available for the proper operation of the PLL. The problem seems to be in the layout from VCO output to power divider where there is a weak signal. This problem may be in the capacitor, which may not be operating correctly at 5.8 GHz.

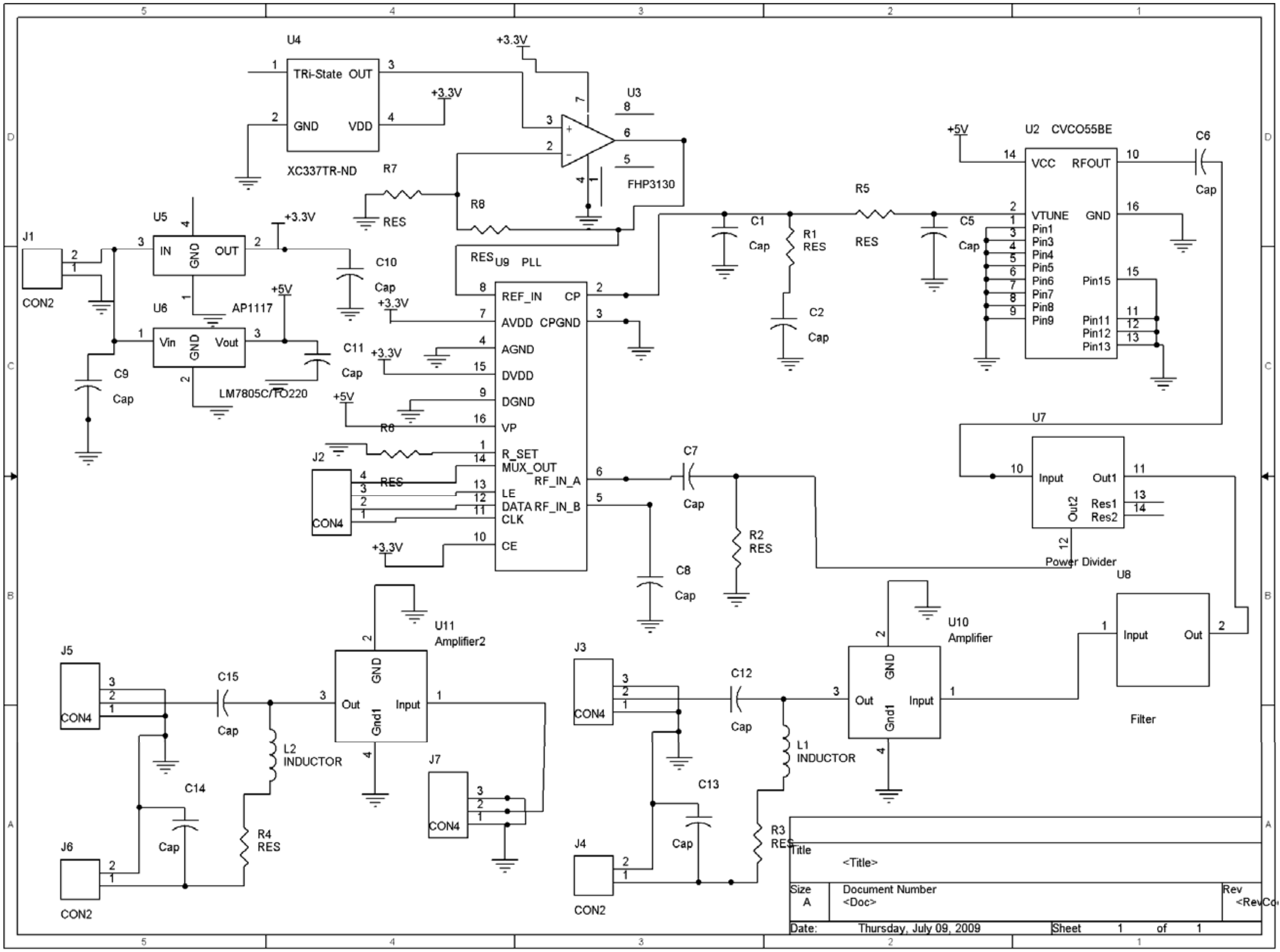
References

[1] D. M. Pozar, *Microwave Engineering*, 3rd ed. Hoboken: John Wiley, 2005.
[2] Georgia Tech IEEE Student Branch Hardware Team: http://hardware.gt-ieee.org/resources/pic44board-1

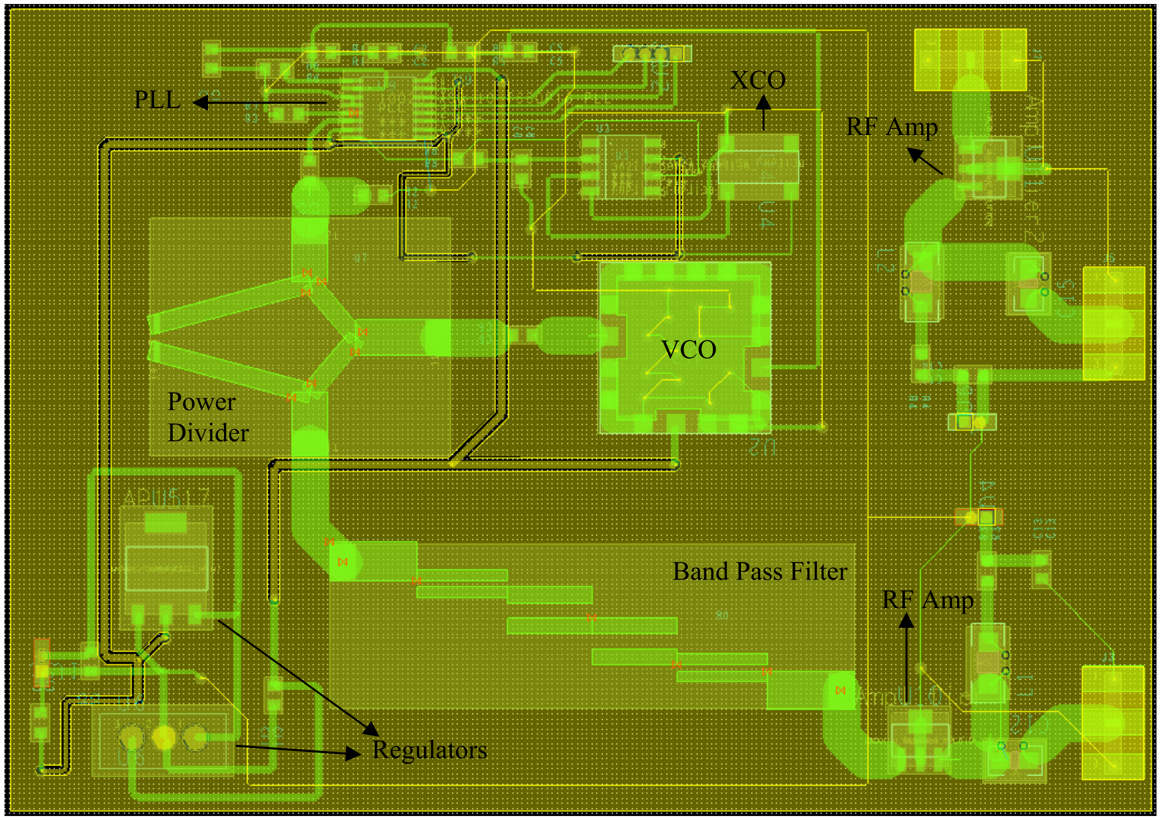
Bill of Materials

Component xQuantity	Part Number/Description
3.3 V Reg. x1	AP1117E33GDIDKR
5.0 V Reg. x1	LM7805CT-ND
PLL x1	ADF4107BRUZ
VCO x1	ROS-5776-119+
RF Amplifier x2	GALI-39+
Crystal Oscillator x1	ECS-3953M-BN
PIC Connector Board x1	PIC44 ([2])
60 mil FR4 board x1	www.barebonespcb.com
Low frequency caps	SMT Zone cap kits
High frequency caps	Dielectric Labs C06 series
Inductors	SMT Zone inductor kits
Resistors	SMT Zone resistor kits

Appendix A: PCB Schematic



Appendix B: PCB Layout



Appendix C: PIC Code

```
/***********************************************
/* Freq_Hop serial programming routine
/* for PIC18F4520
/***********************************************/
/***********************************************
/* Freq_Hop serial programming routine header
/* for PIC18F4520
/***********************************************/
#include "config_18f4520.h"
#include <p18f4520.h>
#include "Freq_Hop.h"

void main() {
/***
 * Disable Interrupts during configuration
 */
   PIE2 = 0;         // B4=EE B3=BCL B2=LVD B1=TMR3 B0=CCP2
   PIE1 = 0;         // B7=PSP B6=AD B5=RC1 B4=TX1 B3=SSP B2=CCP1 B1=TMR2 B0=TMR1
   INTCON = 0;
 /* IO Ports configuration
*/
   TRISA = 0b00111111;  //set TRIS to all inputs before setting initial value, port A is 6 bits wide
   PORTA = 0b00000000;  //Set initial value for PORTA
   TRISA = 0b11111111;  //Set I/O for PORTA as input (1) or output (0)

   TRISB = 0b11111111;  //set TRIS to all inputs before setting initial value, port B is 8 bits wide
   PORTB = 0b00000000;  //Set initial value for PORTB
   TRISB = 0b11110000;  //Set I/O for PORTB as input (1) or output (0)

   TRISC = 0b11111111;  //set TRIS to all inputs before setting initial value, port C is 8 bits wide
   PORTC = 0b00000000;  //Set initial value for PORTC
   TRISC = 0b11111001;  //Set I/O for PORTC as input (1) or output (0)

   TRISD = 0b11111111;  //set TRIS to all inputs before setting initial value, port D is 8 bits wide
   PORTD = 0b00000000;  //Set initial value for PORTD
   TRISD = 0b11111111;  //Set I/O for PORTD as input (1) or output (0)

   TRISE = 0b00000111;  //set TRIS to all inputs before setting initial value, port E is 3 bits wide!
   PORTE = 0b00000111;  //Set initial value for PORTE
   TRISE = 0b00000111;  //Set I/O for PORTE as input (1) or output (0)

   ADCON1 = 0b00001111; // Make all ADC/digital pins digital

while (1)
   {
     program_pll();
   }
   }

/***********************************************
/* Freq_Hop serial programming routine header
/* for PIC18F4520
/***********************************************/
#ifndef __Freq_Hop_H
#define __Freq_Hop_H
```

```
#include <p18f4520.h>

#define SCLK LATBbits.LATB0
#define SDATA LATBbits.LATB1
#define LE LATBbits.LATB2
#define TEST LATBbits.LATB3
//#define SYNC PORTAbits.RA2
#define BIT7 0b10000000
#define BIT6 0b01000000
#define BIT5 0b00100000
#define BIT4 0b00010000
#define BIT3 0b00001000
#define BIT2 0b00000100
#define BIT1 0b00000010
#define BIT0 0b00000001
#define PLLROWS 4
#define PLLCOLS 3

#define clockset() (SCLK = 1)
#define clockclr() (SCLK = 0)
#define dataset() (SDATA = 1)
#define dataclr() (SDATA = 0)
#define pllen() (LE = 0)
#define plldis() (LE = 1)
//#define Delay(x) DELAY = x; while(--DELAY){ Nop(); Nop(); }    // define a delay
extern unsigned int DELAY;
typedef unsigned char byte;

 const static byte latchR[1][3]= {
    0x50,
    0x00,
    0x02
  };
static byte latchN[1][3]= {
    0x75,
    0x59,
    0x00
  };
 const static byte latchF[1][3]= {
    0xE6,
    0x80,
    0xDF
  };
const static byte latchI[1][3]= {
    0xE2,
    0x80,
    0xDF
  };
// set pll for 5.75 GHz
static byte pll_bytes[PLLROWS][PLLCOLS]= {
  {
    0xE6,
    0x80,
    0xDF
  },
  {
    0x50,
    0x00,
    0x02
  },
  {
    0x75,
    0x59,
    0x00
  },
  {
    0xE2,
    0x80,
    0xDF
  }
};

void program_pll(void) {

  byte j,k,m,bangthis;
int i=0;
int p=0;
int n=0;     //use to track 75 different frequencies
    n=0;
    latchN[0][0]=0x75;
    latchN[0][1]=0x59;
    latchN[0][2]=0x00;
    for( n=0; n < 75; n++) {
      clockclr();

  for( j=0; j < PLLROWS; j++) {
    pllen();
    for( m=0; m < PLLCOLS; m++) {
      bangthis = pll_bytes[j][m];
      for( k=0; k<8; k++ ) {
        if ((bangthis & BIT0) != 0) {
            dataset();
        } else {
            dataclr();
        }
        clockset();
      for(i=0;i<=5;i++)
        {
          ;
        }
        clockclr();
      for(i=0;i<=5;i++)
        {
          ;
        }
        bangthis = bangthis >> 1;
      } // for k
    } // for m

   if ( j == 2) {

      if ( pll_bytes[2][0] != 0xFD ) {
         pll_bytes[2][0] +=4;          //we are changing the A value of the counter
      }
      else {
      pll_bytes[2][1]=0x5A;            // The value of the counter B should be changed because the Counter A will overflow
      pll_bytes[2][0]=0x01;
      }
    }
    plldis();

  } // for j
TEST = 0;
//int i=0;
for(i=0;i<=500;i++)
{
for (p=0;p<=200;p++)
;
}
//Delay(102000);
TEST = 1;
}  // for n
//}  // for while
} // program_pll

#endif
```