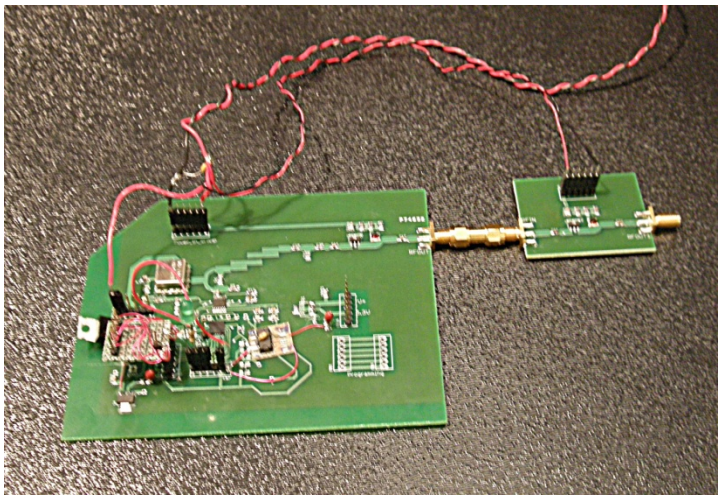
**Project 1**

# 5.8GHz Transmitter

**Team 1: Frank Parks, Saravanan T K, Yun Zhang, Grant Lohsen**



# Table of Contents:



# Table of Figures



# Table of Tables

# 1. Requirements

Project 1 consists of a 5.8 GHz RF Signal Generator (sig gen). The sig gen must comply with FCC Part 15 rules for a transmitter operating within the 5.725-5.850 GHz ISM band. This requires the following design criterion to be met:

1. Maximum output power is +7dBm (5mW).
2. Use of at least 75 frequency channels spaced 1MHz apart
3. Maximum dwell time of 0.4s on a single frequency within a 30s interval.

Further design requirements specified by the research sponsor included

4. Self Contained design on a single circuit board
5. Use of a Wilkinson power divider
6. A lowpass or bandpass filter
7. A frequency hopping algorithm.

# 2. Initial Design Decisions

Due to these requirements a board was designed that utilized a $2^{nd}$ order bandpass filter, a Wilkinson power divider, a 20MHz oscillator, an 18F2520 microcontroller, a RF amplifier, and various other sundry items. These requirements also resulted in a design that differed from the sponsor suggested design. Specifically, it was decided to include a $2^{nd}$ RF amplifier stage. This stage was designed to be separable from the project should it result in a violation of the 1st design criterion. Each of these design choices will be discussed in turn in section 3.

# 3. Detailed Design Decisions

## 3.1. Overall Board layout

The overall board layout is by necessity not the most efficient: This is due to the many debug header pins utilized on the board. The board layout is shown below in Figure 1.

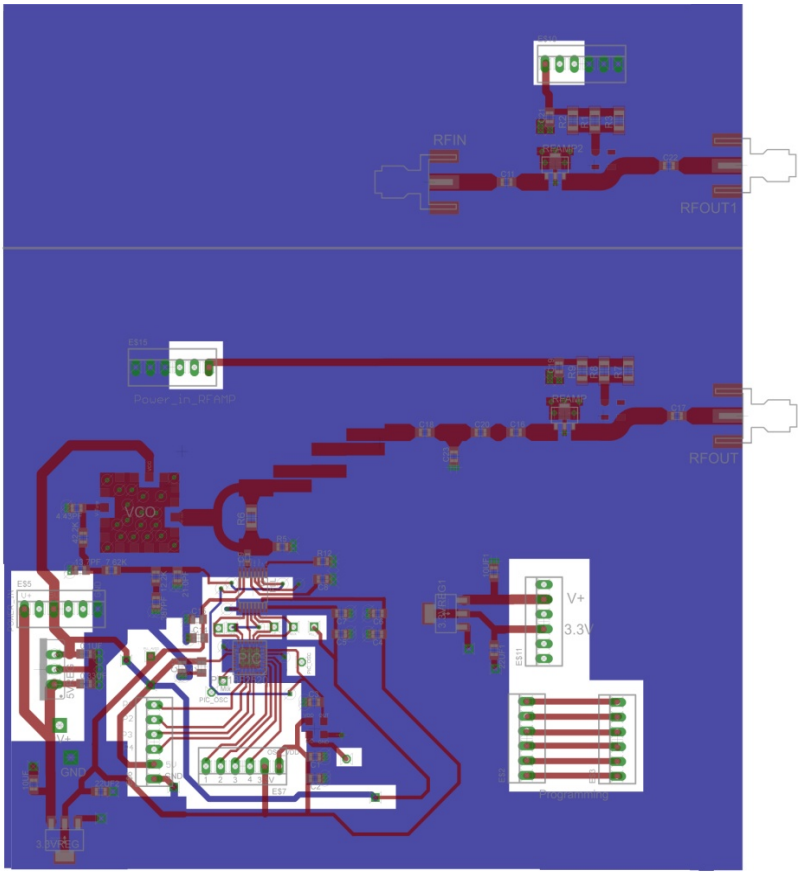



**Figure 1. Eagle PCB Layout. (upper portion is optional 2nd stage RF Amplifier lower portion is primary transmitter module)**

Utilizing this layout kept a maximum distance between the RF lines and all data, clock, and control lines. This was done to minimize noise both in the RF output and the clock lines. Note the prominent Wilkinson power divider and bandpass filter prior to the RF amplifier. An innovative debugging addition is the inclusion of the converter section in the bottom right. This allows different combinations of wires to be soldered on the left side, with a header on the right. Utilizing this one can reconfigure what inputs are provided to a standard header. This would be particularly helpful if one wanted to be able to use different programmers without having to solder near any surface mount components. Also note how the layout routes power. This allowed for traces to be cut to individual devices without affecting any downstream components. These were all useful features when debugging the layout and proved extremely valuable to the design and debugging tasks.

## 3.2. Bandpass Filter

The design specifications for the bandpass filter were a pass band of 5.725 – 5.850 GHz. The aim was for a maximally flat pass band. While the original design was a 4 stage bandpass filter it was determined that the loss was too high to utilize it, especially after it was determined that the filter didn't need to have a sharp roll off. Rather, it was decided to not utilize the 25 MHz fringes of the band allowing for a far more lightweight design. To achieve the 5.8 GHz signal generator specifications, a coupled line

bandpass filter is designed with the center frequency of 5.8 GHz and a fractional bandwidth of 10%. The maximally flat type is used because the low insertion loss has higher priority than the sharp cutoff in our system. For the same reason, the filter order N is chosen to be 2. Higher order filters have too much insertion loss because the FR-4 substrate has a large TanD of 0.02.

The coupled line filter is the easiest way to implement a bandpass filter with a fractional bandwidth less than 20%. The detailed design procedure can be found in Pozar's text[1]. The filter model parameters are listed in Table 1. Z0e and Z0o are the even- and odd-mode line impedances for the coupled lines. Using the LineCalc tool in ADS, the coupled line's strip width (W), strip length (L), and space (S) between two strips can be calculated as a start point for the design. The optimized W, L, and S are shown in Table 1 after simulation in ADS. Considering the limited fabrication accuracy, all the values are rounded to 1 mil.

The filter design is shown below in Figure 2. Also shown is the simulated response of the filter, shown in Figure 3.

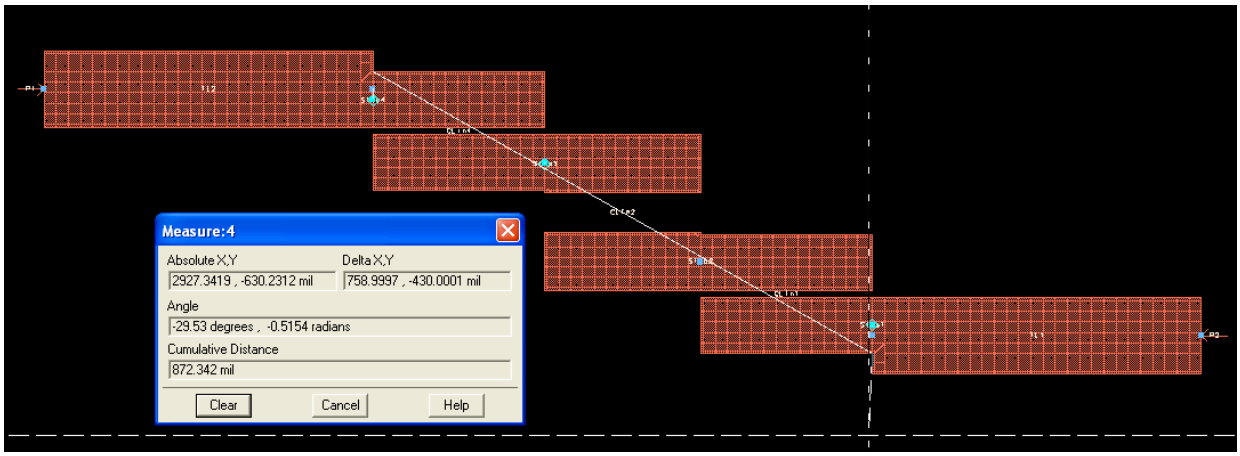



**Figure 2. Design of the Bandpass Filter.**

[1] D. M. Pozar, *Microwave engineering*, 3rd ed. Hoboken, NJ: J. Wiley, 2005.

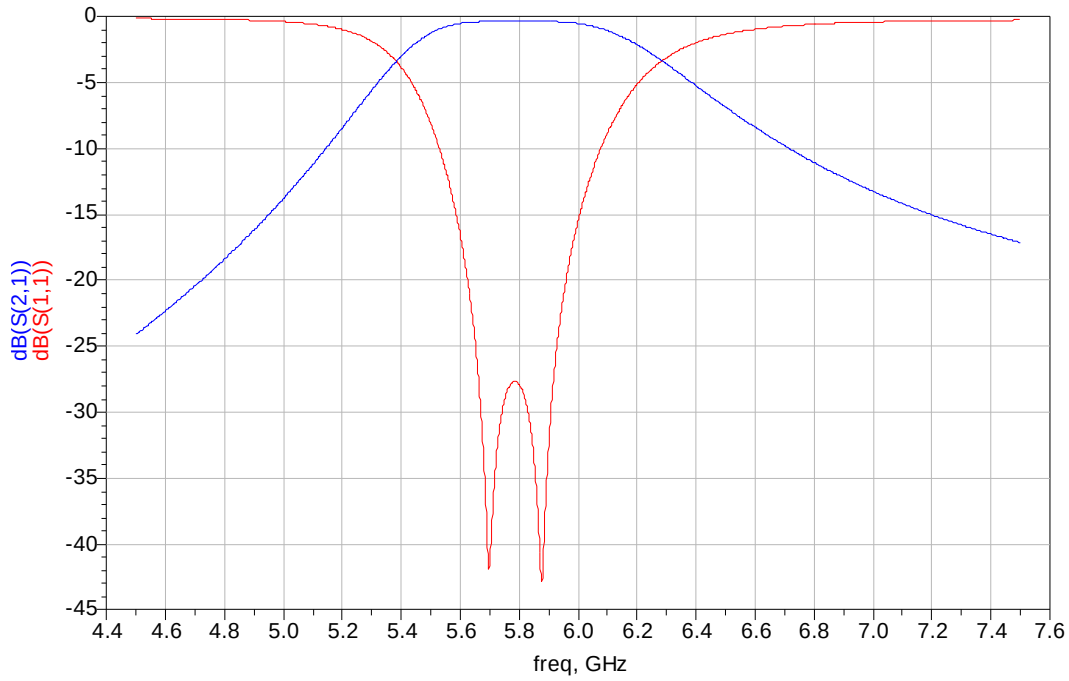



**Figure 3. Simulated Response of the 2 stage Bandpass Filter.**

## 3.3. Wilkinson Power Divider

A standard Wilkinson Power Divider design was utilized due to specified requirements (#5). Compared to other three-port networks such as T-junctions and resistive dividers, although the Wilkinson power divider is a lossy network, it enjoys simultaneous matching at all three ports with perfect isolation between output ports. In this 5.8 GHz signal generator, a 1:1 Wilkinson power divider is designed to split the output power from the VCO. The output power 1 provides the feedback to the PLL, and the output power 2 works as the output from the signal generator.

The design takes into account the matching at all three ports, isolation between output ports, and the transmission coefficients from input to output to meet the requirement. After manual calculation for initial design parameters by Pozar's equation[2], ADS is used for parameter optimization and simulation. In order for the circuits to be applicable, the optimized microstrip line dimensions are rounded to 1 mil. The design parameters, power divider's schematic, and simulation results are shown in

[2] D. M. Pozar, *Microwave engineering*, 3rd ed. Hoboken, NJ: J. Wiley, 2005.

Table 2, Figure 4, and Figure 5, respectively. In the working frequency band, the transmission coefficients are -3.32 dB, and the isolation between the two output ports is below -30 dB. Shown below in Figure 4 is the design of the implemented Wilkinson Power Divider. The results of the ADS simulation of the Wilkinson Power Divider are shown in Figure 5.

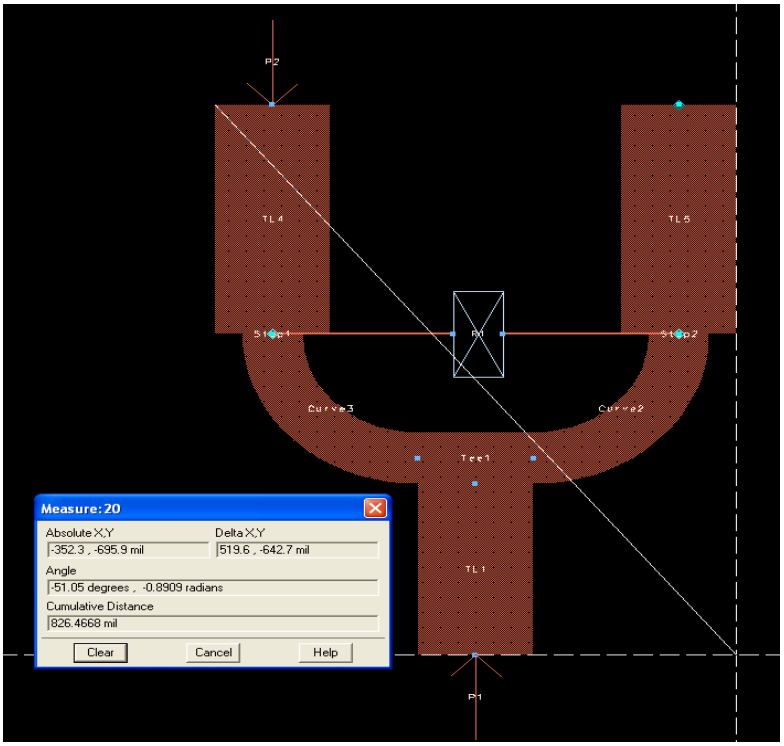



**Figure 4. Design of the Wilkinson Power Divider.**

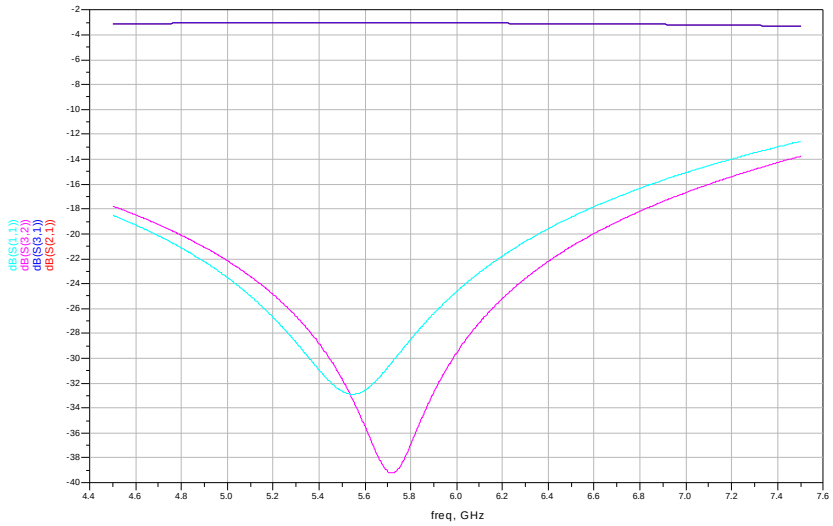



**Figure 5. ADS Simulation of the Wilkinson Power Divider**

## 3.4. 2nd stage RF Amplifier

The RF Amplifier was biased according to the recommended application circuit in the data sheet. This was accomplished by applying a DC voltage to the output of the amplifier, while isolating the DC source from RF with an RF choke. Here, it was necessary to control the bias current with a series resistor. This value was originally calculated to be 243 Ohms for a 12V bias voltage according to the datasheet [3], however it was realized that typical surface mount resistors would not be able to dissipate the required power (~0.59W). Thus, a parallel combination of three large resistors (1206 package) were used to achieve an identical resistance, while allowing for a total power dissipation of 0.75W.

*The Second RF Amp*

According to the datasheet[4], the RF amplifier could be expected to provide a gain of 16 dB at 5.8GHz. Assuming a 1.5dBm output from the VCO, and the various losses detailed below, the best case output of the RF amplifier was approximated to 5.5dBm. Since this did not meet the design requirements (7dBm+), a second RF amplifier was cascaded in order to meet this specification.

*Operating in Saturation*

One issue that was anticipated when designing this system was the possibility that the second RF amplifier would likely be operating in saturation. In case waveform distortion became a problem, a "T"

[3] http://minicircuits.com/pdfs/GALI-39+.pdf
[4] Ibid.

attenuator network was added to prevent this saturation. In the end, however, this attenuator network was populated with 0 Ohm resistors, despite a slight oversaturation of the second RF amp.

## 3.5. Microcontroller

The microcontroller was used to control the PLL which in turn controlled the RF output of the device. The microcontroller was connected to the PLL by four lines. The lines used are 2 SPI lines, 1 LE (Load Enable), and 1 CE (Chip Enable). Chip enable turns the chips charge pump output on and off. Load Enable latches data in the buffer into the proper register in the PLL. Data is sent using the 3 SPI lines[5].

Communication via the PLL and PIC is accomplished with SPI lines. SPI requires Clock, Data, and LE lines to operate. Communications uses the following sequence: Load Enable low, 24 bits of data, Load Enable high. This allows for communication between the microcontroller and the PLL and thus allows for the frequency hopping algorithm to be put into use. This communication protocol is depicted in Figure 6.

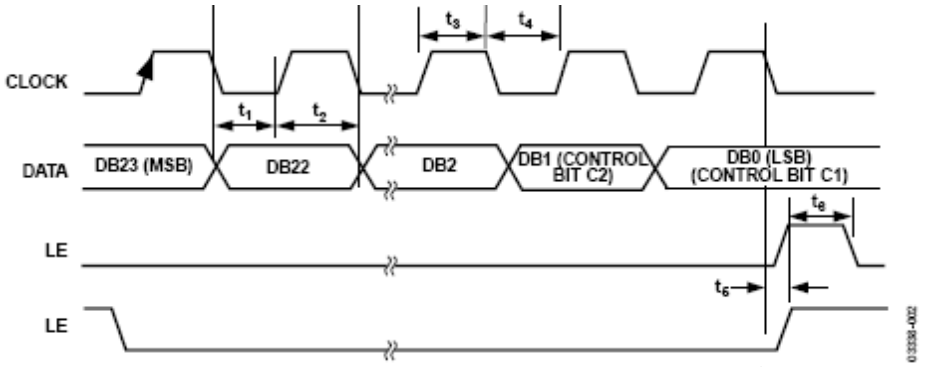



**Figure 6. SPI Communication protocol for the ADF4107[6].**

The frequency hopping algorithm chosen for use is a simple step algorithm where each frequency is one step higher than the last. In this project a step frequency of 1 MHz was chosen to satisfy requirement #2 (*Use of at least 75 frequency channels spaced 1MHz apart*). Given both that and the requirement that the signal be between 5.725 and 5.850 GHz it was decided to have the algorithm run from 5.750-5.825 GHz in order to minimize the chance of having a spur outside of the allowed frequency range while still meeting the requirements set forth above.

Also included were debug pins. These were utilized in conjunction with a series Resistor/LED in order to provide a blink alive and an indication of when a frequency hop occurs (on rising and falling LED edges).

# 4. Performance

The unit performed well within design specifications. The measured output ranged from 6.8-7.58 dBm (Figure 7) from 5.75-5.825GHz, with a step size of 1 MHz (Figure 8), and with frequency hopping

[5] http://www.analog.com/static/imported-files/data_sheets/ADF4107.pdf
[6] Ibid.

occurring at 0.393 seconds (Figure 9), which was inside of the acceptable range. The board performs optimally at 12 Volts and ~200mA of current draw. While the board will function above this voltage and current level, it would violate FCC Part 15 rules and therefore should not be run at anything more than 12 Volts. The board will run at lower voltage levels, however it will have a substantial power output decrease.

In the future a feedback system could be used to set specific power outputs. The saturation of the 2nd RF Amp could be mitigated in the future and any board redesign from version 0 would take that into account. Further, the board functioned perfectly as a debug platform and bears the scars of the development process. These changes had a positive effect on the outcome of the project and should the board be produced in the future it would take advantage of these changes.

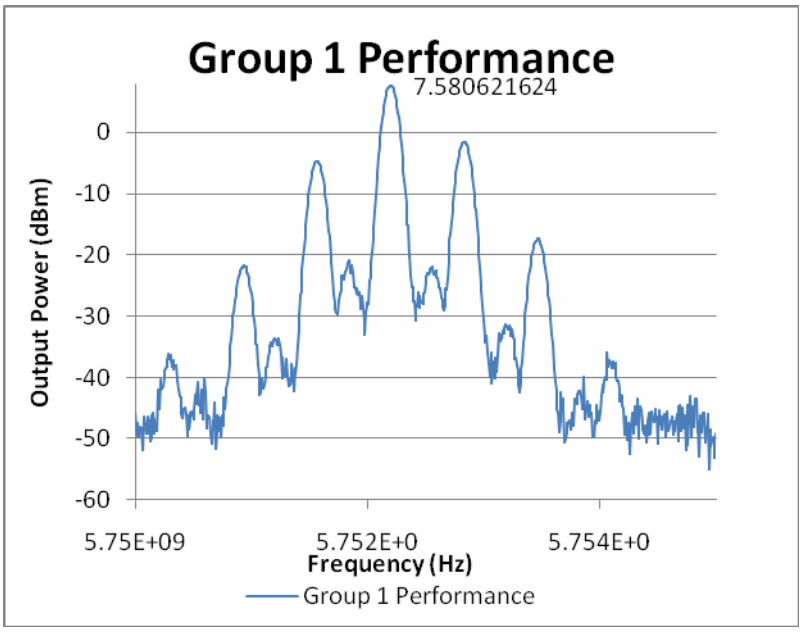



**Figure 7. Output Spectrum of the 5.8GHz transmitter**

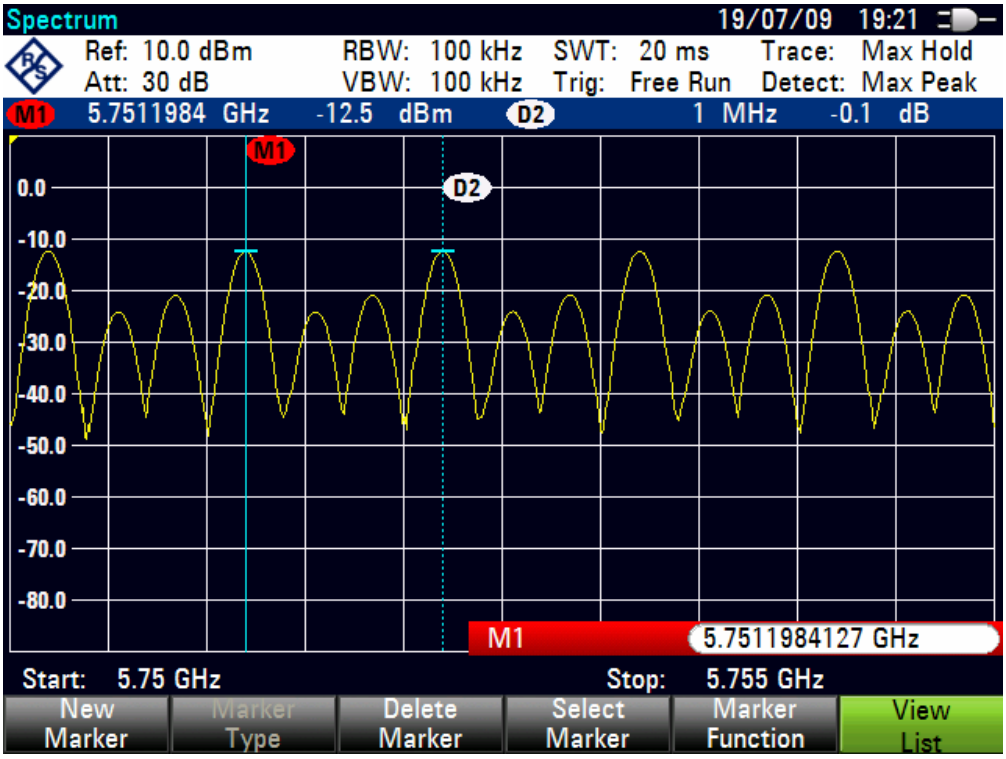



**Figure 8. Spectrum Analyzer capture showing 1MHz frequency spacing.**

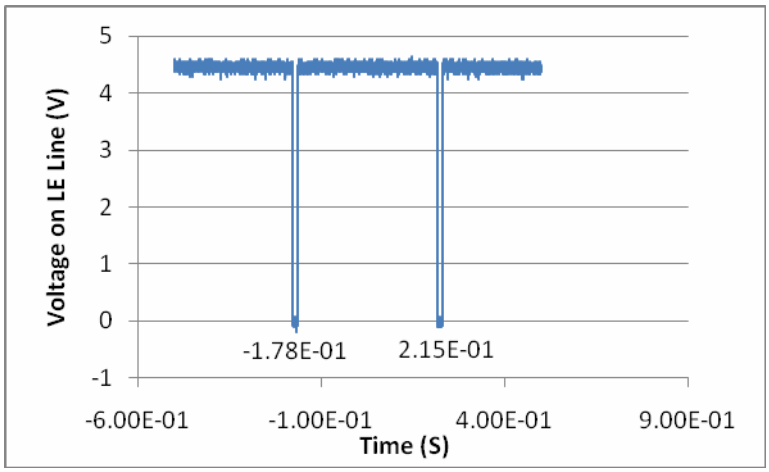



**Figure 9. Output of LE pin of the 5.8GHz transmitter showing the timing of frequency hopping (~0.39 seconds)**

# Appendix A. ADS Design Parameters

# Bandpass Filter Design

| n | $g_n$ | $J_nZ_0$ | $Z_{0e}$ (Ω) | $Z_{0o}$ (Ω) | W (mil) | S (mil) | L (mil) |
|---|---|---|---|---|---|---|---|
| 1 | 1.4142 | 0.3333 | 72.22 | 38.89 | 73 | 12 | 225 |
| 2 | 1.4142 | 0.1111 | 56.17 | 45.06 | 80 | 60 | 273 |
| 3 | 1.0000 | 0.3333 | 72.22 | 38.89 | 73 | 12 | 225 |

**Table 1. Design parameters for a 5.8 GHz 2-order coupled line bandpass filter.**

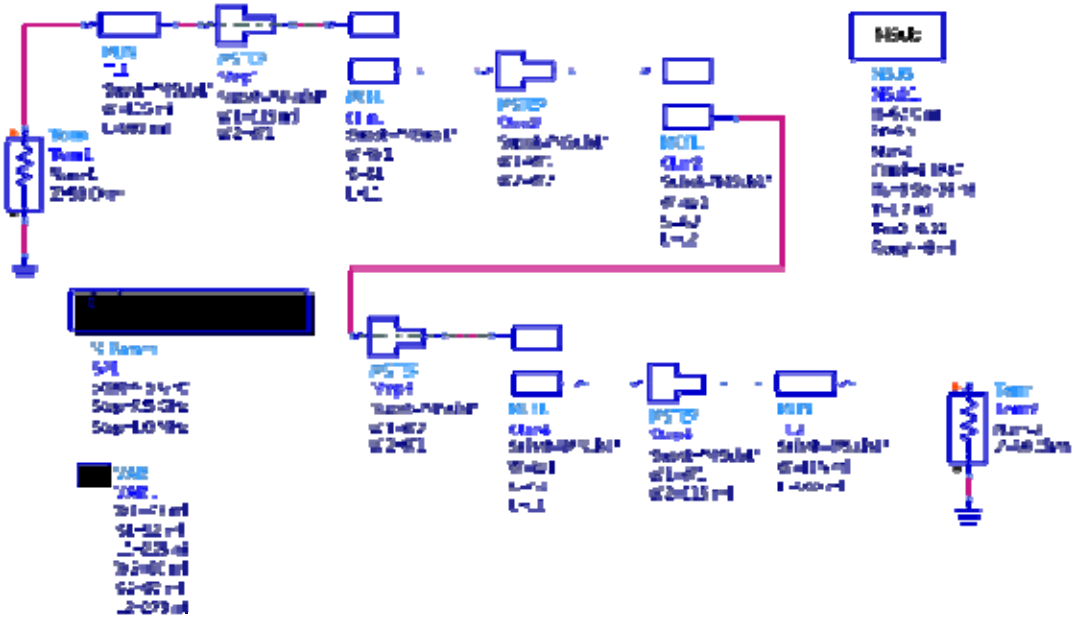


**Figure 10. ADS Layout for the bandpass filter.**

# Wilkinson Power Divider

| Section | Impedance (Ω) | Width (mil) | Length (mil) |
|---|---|---|---|
| $Z_0$ | 50 | 115 | 200 |
| $Z_2$ | 70.71 | 60 | 267 |
| $Z_3$ | 70.71 | 60 | 267 |
| $T_2$ | 50 | 115 | 200 |
| $T_3$ | 50 | 115 | 200 |
| R | 100 | 50 | 80 |

**Table 2. Design parameters for a 5.8 GHz Wilkinson power divider.**

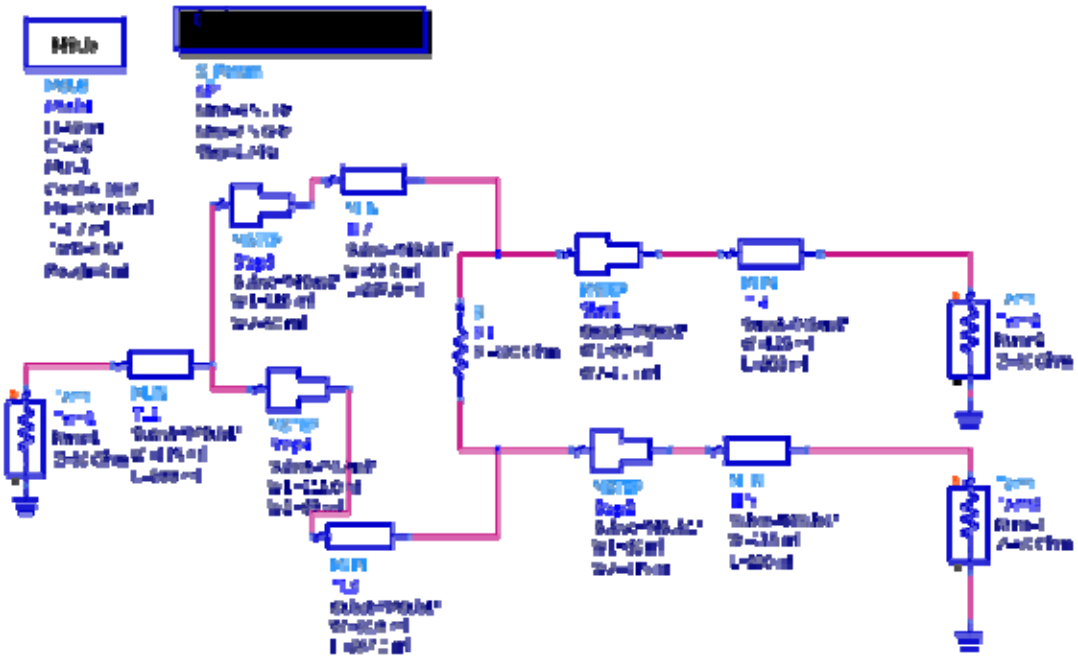


**Figure 11. ADS Layout for the Wilkinson power divider.**

# Appendix B. Microcontroller Code

```
#ifndef __18F2520
#define __18F2520
#endif

#include <p18cxxx.h>

#include <spi.h>

//configure
#pragma config OSC = HS ,  IESO = OFF                    //FCMEM = ON
#pragma config PWRT = ON, BOREN = OFF, BORV = 1 //, VREGEN = OFF
#pragma config WDT = OFF, WDTPS = 4  , MCLRE = ON
#pragma config LPT1OSC = OFF, PBADEN = OFF, CCP2MX = PORTBE
#pragma config LVP = OFF


/*******************************
 * Aliases
 * Pin Name -> Port name
 *******************************
 */
//TEMP  #define BIT7 0b10000000
#define BlinkAlive PORTAbits.RA3
#define BlinkAliveTRIS TRISAbits.TRISA3

#define CEpin PORTCbits.RC6
#define CEtris TRISCbits.TRISC6
#define LEpin PORTCbits.RC2
#define LEtris TRISCbits.TRISC2


/*******************************
 * Constants
 *******************************
 */

const int B_Initial_Ref_Latch = 179;
const int A_Initial_Ref_Latch = 22;                   //max 63


/*******************************
 * Global variables
 *******************************
 */
char TMR0X;                    // Eight-bit extension to TMR1
int A_Ref_Latch;               //data to latch into A register of N latch
int B_Ref_Latch;               //data to latch into B register of N latch
unsigned char Count = 0;
unsigned char Init_Latch[3] = {0x9F, 0x80, 0xE3};
unsigned char Func_Latch[3] ={0x9F, 0x80, 0xE6};
unsigned char Func_Latch2[3] ={0x9F, 0x80, 0xE2};
unsigned char Ref_Latch[3] = {0x00,0x00,0x50};
unsigned char AB_Initial_Latch[3] = {0x20, 0xB3, 0x59};


/*******************************
 * Function prototypes
 *******************************
 */
void main(void);
```

```
void InitalizePIC(void);
void InitalizeADF4107(void);
void SendFrequency(int Frequency);
void HiPriISR(void);
void LoPriISR(void);
void WriteToADF4107(unsigned char *stringToSend);
void TMR0handler(void);  //just increment an 8 bit value that keeps track of
//the high end
void FrequencyHop(void);
void Delay(void);


//the hi and lo pri isr need specific mem locations.
//that is the purpose of these functions


//////////start load into the high priority interupt vector memory location
#pragma code highVector=0x08
void atHighVector(void)
{
 _asm GOTO HiPriISR _endasm                                          //point to
//location of hi priority interrupt routine c code
}
#pragma code
//////////end load into the high priority interupt vector memory location


//////////start load into the low priority interupt vector memory location
#pragma code lowVector=0x18
void atLowVector(void)
{
 _asm GOTO LoPriISR _endasm                                          //point to
//location of low priority interrupt routine c code
}
#pragma code
//////////end load into the low priority interupt vector memory location



void main (void)
{

      InitalizePIC();                                                      //PIC
//initalize

      CEpin =1;
      //adf4107 chip enable
      LEpin = 1;
      //adf4107 chip do not latch data

      Delay();
      //give voltage time to stabalize
      Delay();
      //give voltage time to stabalize
      Delay();
      //give voltage time to stabalize
      Delay();
      //give voltage time to stabalize
      Delay();
      //give voltage time to stabalize
```

```
      InitalizeADF4107();
      //ADF4017 PLL Initalize

      T0C0Nbits.TMR00N = 1;                                          //enable
//the timer
    RC0Nbits.SB0REN = 0;                                             //disable
//software brownout reset

    while (1)
    {    }
      //wait here forever, the interrupts will handle everything

}



//PIC2520 Initalization routine
void InitalizePIC (void)
{
      T0C0N =    0b00001000;                                         //timer
//stoped, 16bit, internal clock CLK0 no prescalar

      //0001 1110  1000 0100  1000 0000 = 0.4 second with F0sc/4 = 5MHz
      RC0Nbits.IPEN = 1;                                             // Enable
//priority levels
      INTC0N =   0b11100000;                                         //high, low
//priority interrupts enable, Timer0InterruptEnable
      TMR0X = 0;
      //reset 24-16bit portion of counter/tiomer
      INTC0Nbits.GIEL = 1;                                           // Enable
//low-priority interrupts to CPU
      INTC0Nbits.GIEH = 1;                                           // Enable
//all interrupts

      A_Ref_Latch = A_Initial_Ref_Latch;
      B_Ref_Latch = B_Initial_Ref_Latch;

      /////////setup ground for led
      TRISAbits.TRISA2 = 0;
      P0RTAbits.RA2 = 0;
      /////////end setup ground for led

      TRISCbits.TRISC1 = 1;                                          //MUXIN?

      ///////set lines to digital outputs
      TRISAbits.TRISA4 = 0;                                          //RA4 or P3
//on debug is for me to provide blink alive a ground path
      BlinkAliveTRIS = 0;
      CEtris  = 0;
      LEtris = 0;
      ///////end set lines to digital outputs

      P0RTAbits.RA4 =0;                                              //set RA4
//or P3 on debug bus to 0 in order to provide the ground path for the LED
      0penSPI(SPI_F0SC_4, M0DE_00, SMPEND);                 //open SPI comm
      return;
}


//ADF4107 initalization routines
void InitalizeADF4107(void)
```

```
{
      Delay();
      WriteToADF4107(Func_Latch);                                  //send
//function latch specifiying how to setup the pll
      Delay();
      WriteToADF4107(Ref_Latch);                                   //send
//reference latch
      Delay();
      WriteToADF4107(AB_Initial_Latch);                      //send ab latch
      Delay();
      WriteToADF4107(Func_Latch2);                           //send function
//latch 2 (enables operation)
      Delay();
      return;
}



#pragma interrupt HiPriISR                                         // High-
priority interrupt service routine
void HiPriISR(void)                                          // Included to
//show form
                                                                   // Supports
retfie FAST automatically
{
      INTC0Nbits.TMR0IE = 0;                                       //disable
//tmr0 interrupts
    while(1)
    {

      if (INTC0Nbits.TMR0IF)
        {
                  TMR0handler();
      //Call handling routine for timer 0
              continue;
        }
        break;
    }
      INTC0Nbits.TMR0IE = 1;                                       //enable
//tmr0 interrupts
      return;
}

#pragma interruptlow LoPriISR                                // Low-priority
//interrupt service routine
void LoPriISR(void)
{

}

void TMR0handler()
{
     TMR0X++;                                                // Increment
//Timer1 extension
     INTC0Nbits.TMR0IF= 0;                                   // Clear flag and
//return to polling routine

     if (TMR0X == 0b00011110)                                //~0.4seconds has
//passed
     {
```

```
          TMR0X = 0;
      //reset TMR0X
            FrequencyHop();
      //determine and set new frequency
     }
      return;
}


void Delay()
{
      int i;
      for (i=0; i<1000; i++)
      {}
      //sit around long enough for a few microseconds
      return;

}

void FrequencyHop(void)
{
      unsigned char AB_Ref_Latch[3];


      //generate next frequency
      if ((B_Ref_Latch >= 182)&&(A_Ref_Latch>=1))
      {
            A_Ref_Latch = A_Initial_Ref_Latch;              //reset to first
//frequency in sequence
            B_Ref_Latch = B_Initial_Ref_Latch;              //reset to first
//requency in sequence
      }else
      {
      BlinkAlive = (++Count) % 2;                           //blink alive
//transitions on-off or off-on at each freq hop
            A_Ref_Latch++;
            //otherwise just increment cur freq
            if (A_Ref_Latch >= 32)                                //if A ref
//latch value is >=32 it is time to inc b and reset A
            {
                  A_Ref_Latch = 0;                                //reset a
                  B_Ref_Latch++;                                  //inc b
            }

      }
      //send SPI of value to set next frequency
      AB_Ref_Latch[0] = 0x20;                                     //this is a
//given
      AB_Ref_Latch[1] = B_Ref_Latch;                              //store b
//ref latch
      AB_Ref_Latch[2] = (A_Ref_Latch*4)+1;                  //store a ref
//latch and the control bits (0,1)
      WriteToADF4107(AB_Ref_Latch);                         //write new AB
//latch value to pll
      return;
}



void WriteToADF4107(unsigned char *stringToSend)
{
```

```
      int i;


      LEpin = 0;
      //transition LE line to low to allow for data to be latched in
      Delay();
      Delay();
      for (i = 0; i <=2; i++)
       {
          PIR1bits.SSPIF = 0;                                    // Clear
//SPI flag
          SSPBUF = stringToSend[i];                          // Send byte
          while (!PIR1bits.SSPIF);                               // Wait for
//transmission to complete
       }
      Delay();
      Delay();
      LEpin = 1;                                                        //le
//line high  (set the data in the buffer to a latch)


      return;
}
```