Final Group Report

Team Leidar

Yizhi Tao
Yuanyuan Xiang
Li Zhang
Jiapeng Zhong

Design

Design Specification

- Range from 5m to 50m
- $300 Budget
- Frequency Modulated Continuous Wave Radar

Project option

We chose accuracy as our first priority in this design. We generally followed the system schematic of last quarter. Initially, we wanted to use a microcontroller to do on-board signal processing; however, the microcontroller we chose, CC3200, did not have enough memory space to do necessary signal processing, so we changed back to use computer. We also decide to experiment with two yagi antennas. However, we also had a plan of using one or two coffee can antennas as back up.

Component selection

Baseband PCB

The new radar system would be built upon quarter 1 design. Therefore, the functions of baseband circuit included conversion from 8 volt to 5 volt, generation of triangle wave for VCO and a low pass filter and a gain stage for output signal from mixer. In addition, the circuit also needed to convert 8 volt to 3.3 volt which is the voltage supply for the microcontroller, CC3200. With the knowledge on the functions of baseband circuit, we designed the circuit based on the circuit form lab1 manual from quarter 1. We decided to not change components for low pass filter and gain stage, since the given designed circuit on lab1 manual is refined and the required resistors were already provided by the lab. There is no need to order new SMD resistors and op-amps.

RF PCB

To simplify the circuit and debugging process, we chose to purchase highly integrated components instead of building on discrete components. To accomodate for the increased range and different antennas compared to quarter one, we picked LNAs to provide the gain needed for the transmitter and receiver. We then selected our components according to the gain and frequency specification.

Block Diagram

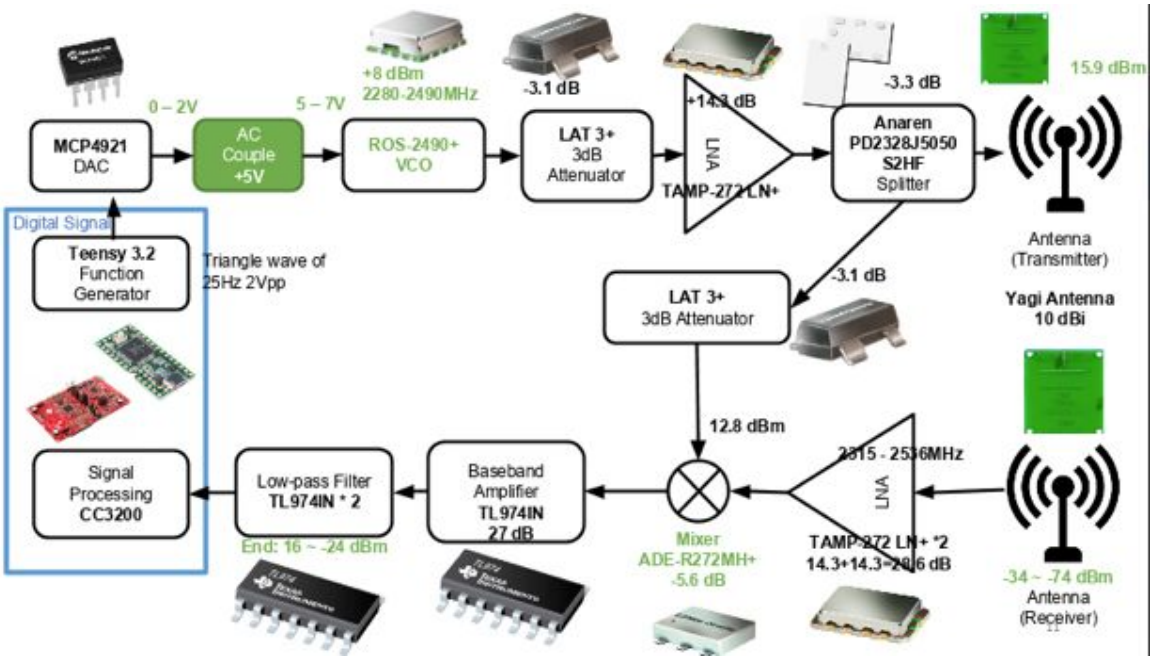


Fig 1: block diagram of radar system

ADI Simulations

Transmitting system (power approximation)

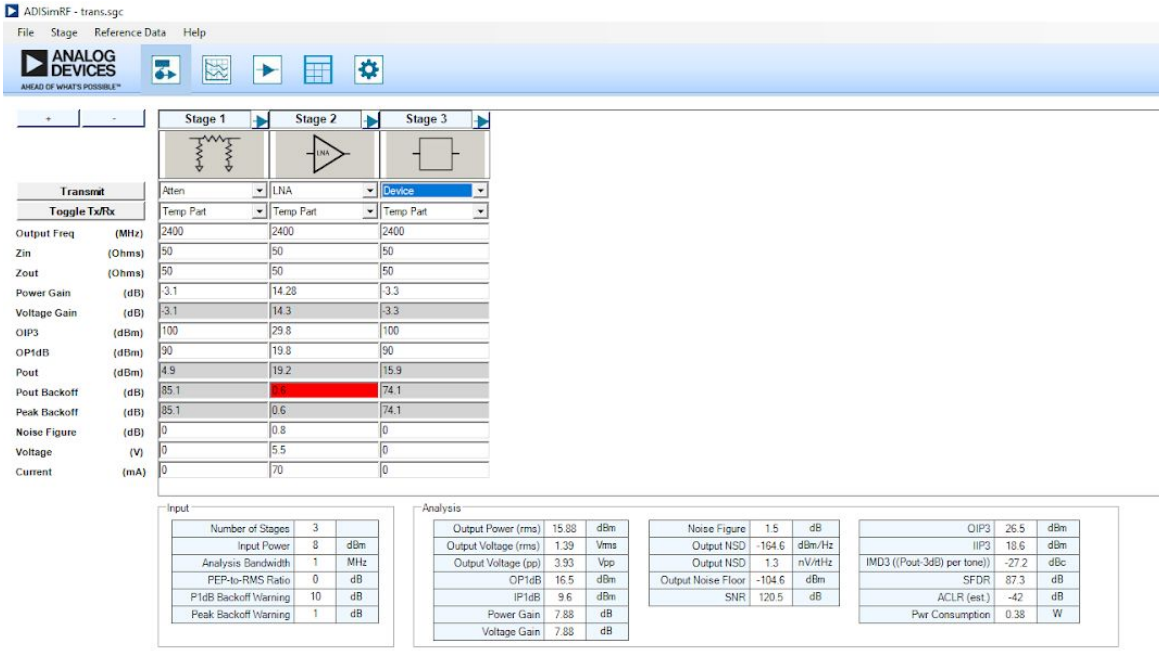


Fig 2: ADI simulation of transmitter

Receiving system (power approximation)

5m

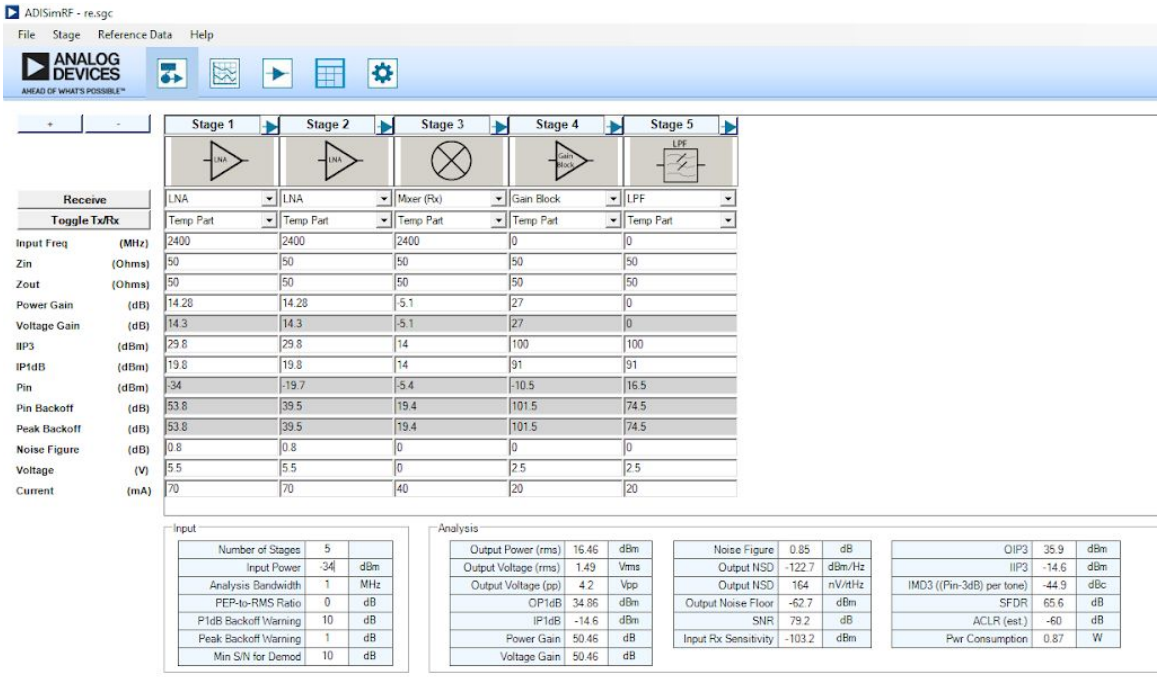


Fig 3: ADI simulation of receiver at 5m

50m

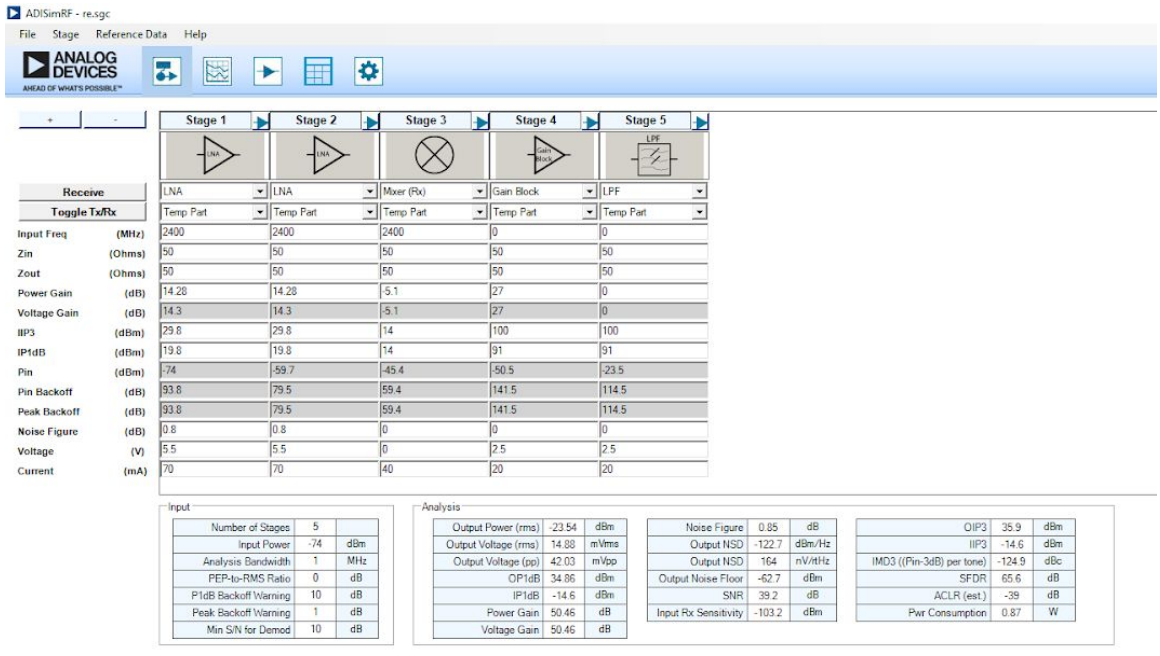


Fig 4: ADI simulation of receiver at 50m

Implementation

Baseband

Schematic

The first part of baseband circuit is the conversion of voltages. The detailed schematic is shown in Fig 5.

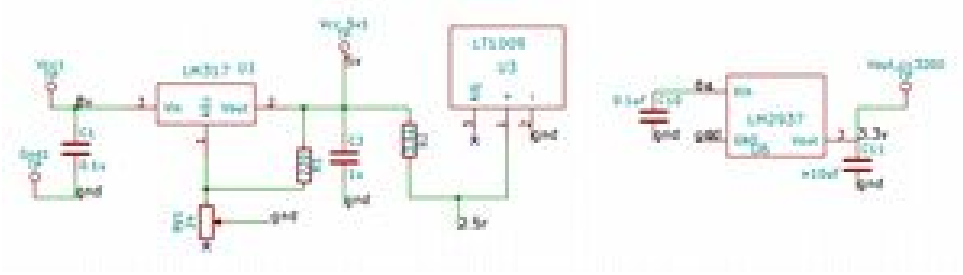


Fig 5: voltage conversion circuit

In this part, LM317 adjustable voltage regulator is used to convert 8 volt to 5 volt. And LT1009 2.5v voltage reference is used to generate 2.5 volt as the biased voltage for low pass filter and gain stage and the reference voltage for DAC which converts digital signal from teensy to analog signal. In addition, LM2937 3.3v is used to convert 8 volt to 3.3 volt as voltage supply for CC3200. The circuit is based on the evaluated circuit shown on the data sheet.

Fig 6 shows the low pass filter and gain stage design.

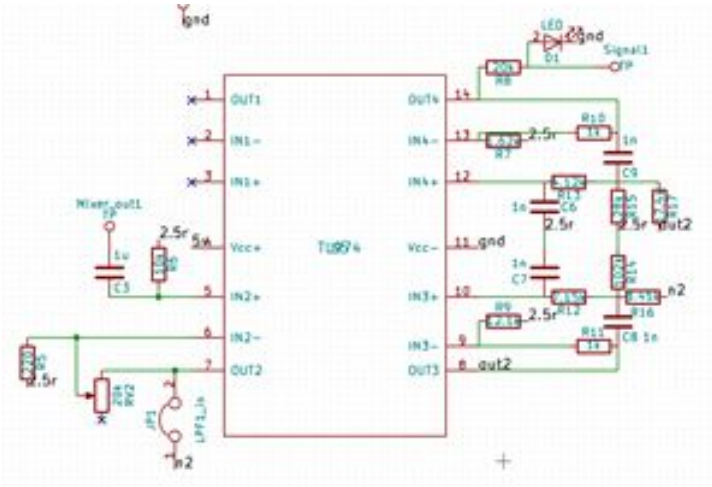


Fig 6: low pass filter and gain stage

There is no significant change on this part compared to quarter 1. The noticeable change is that the output signal is connected to an LED which prevents output voltage from exceeding 1.4 volt, the maximum input voltage for CC3200 ADC.

For Teensy and MCP4921 (DAC), the schematics are the same as quater1. The following graph is the overall schematic.

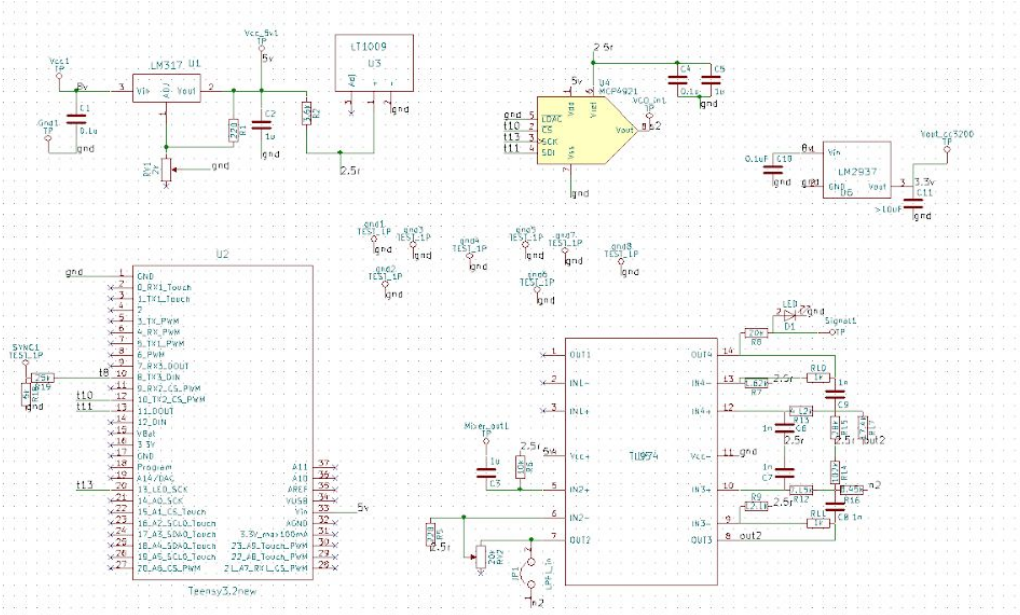


Fig 7: overall baseband schematic

Footprint

After schematic design, we needed to create footprint for each component. The rule of thumb is that carefully review the PCB design the PCB footprint provided on the datasheet and make sure it is bottom view or top view. It is important to know the unit on the datasheet is mm or inch. In addition, assigning each pad the correct pin number is required, because later when laying out the PCB, the track net is based on the pin number on the footprint and the pin number on schematic. Make sure the two pin numbers are matched. Furthermore, using non-copper line to outline the approximate size of the components can help to define the distance between each component when doing PCB layout.

Layout

At first, we outlined a fixed size (1.5inch by 2.5inch) for my baseband PCB to make sure RF and baseband PCBs have the same size. The, load netlist to the layout window. Based on netlist to place the components. Using track via can help avoid cross of tracks. And use fill zone to fill ground is a good way to save tracks from grounding. Before, upload the PCB design, it is necessary to refill zone.

Stacking

For good connection between two PCBs, we choose to use stack two PCBs together. The stacking PCB is shown in Fig 8.

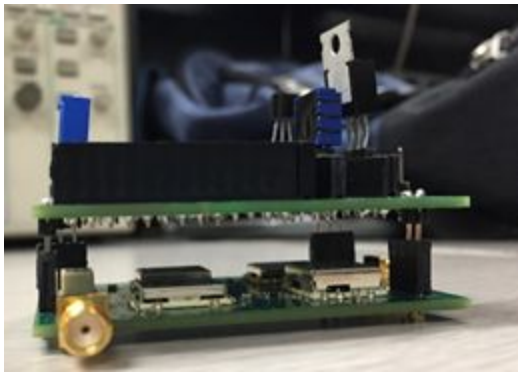


Fig 8: stacked PCB

First, we created three pins: VCO input, ground and 5 volt on the top left corner. And make sure the positions of pins are the same as the positions of pins on the RF band. Using larger grid makes the position matching easier. Then, to make sure the RF PCB can support baseband PCB, we created several ground pins at other three corners. The final PCB layout is shown below in Fig 9.

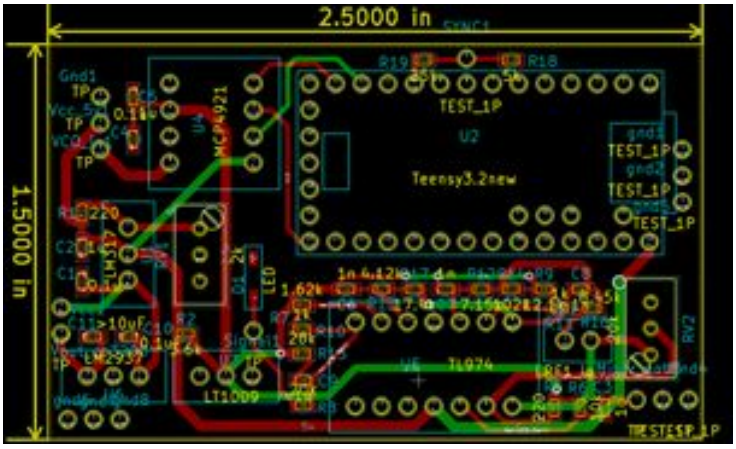


Fig 9: final baseband PCB layout

Assembling

The first step to assembly the PCB is to collect all needed components and mark each component correctly. Then, based on the PCB design file, we soldered each component on its place. Personally, placing solder paste to all the SMD

copper sections and then place the components is the most efficient way to assembly a PCB.

The first soldered PCB is shown below in Fig 10.

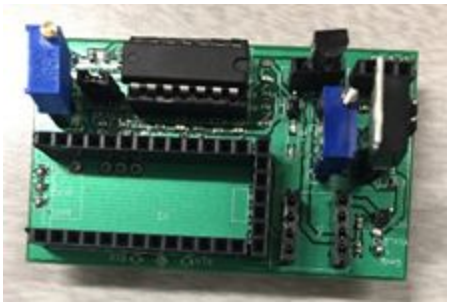


Fig 10: soldered PCB

We used normal female pin headers for Teensy, op-amp, DAC and potentiometers. And then we test this baseband PCB. It works fine. Later, when we stacked the baseband PCB on the RF PCB and tested the whole system, we had troubled on our baseband PCB. Since the potentiometers have very thin pin, the contacts between potentiometers and pin headers are loose. Also, there are loose contacts between op-amp pin headers. Therefore, we decided to re-soldered baseband PCB.

We used “14 Position Pin Standard Circular Connector” for op-amp and “8 Position Pin Standard Circular Connector” for DAC to avoid loose contacts. Also, we directly soldered potentiometers on PCB. Fig 11. below shows the re-soldered PCB.

Fig 11: re-soldered PCB

This re-soldered PCB worked much better than the old one.

RF PCB

Schematic

The schematic of the RF PCB looks similar to quarter one RF system, Fig 12. I placed the transmitter on top and receiver on bottom to easily differentiate from them. I also placed capacitors between vcc and gnd and between some components. Capacitors between vcc and gnd are called decoupling or bypass capacitors and they are used to decouple the AC signal from power supply and protect the circuit. On the other hand, capacitors in between RF components can block the DC offset.

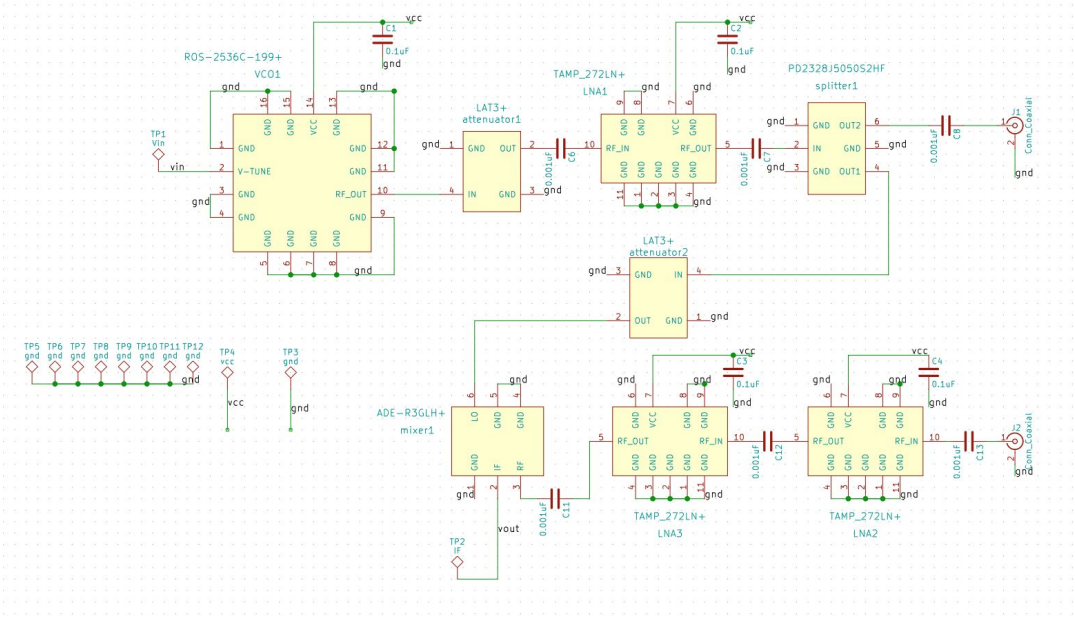


Fig 12.. RF PCB schematic

Footprint

The next task is creating custom-designed footprints. Because KiCAD likely does not have the correct footprints for the component and you should not trust footprints on the internet, so it is safest to create your own footprints. Study the outline drawing and outline dimensions on the datasheet. Having correct footprint is essential to the success of PCB, double check and triple check if necessary. Fig 13 below shows an example of a VCO footprint.

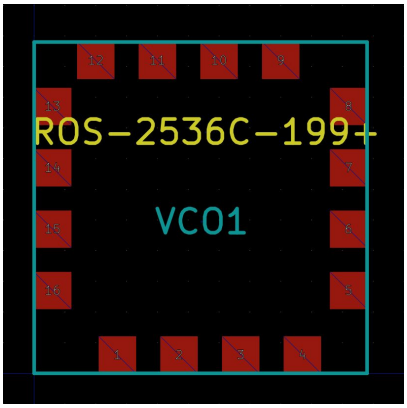


Fig 13: an example of VCO footprint

PCB Layout

In Schematic editor, click generate netlist. Then open Pcbnew and click read netlist. After that, you should see all the components stacked together in one place. Use right click -> Move or keyboard shortcut M to separate them and place them in the desired location. Note that you can change the layout any time in the future, so it is not necessary to decide the location of each component in the beginning.

On RF PCB, there are two types of traces you should distinguish between, low frequency (or DC) signal paths and high frequency signal paths. In my design, the only low frequency and DC signals are the V-tune signal from baseband PCB and VCC power lines. Other traces are all high frequency RF (2.4 GHz) signals in transmitter and receiver. For low frequency signals, you do not need to worry too much about the track width as long as they are not too thin. However, for high frequency RF signal traces, you do have to pay attention to the track width and length.

Track width for RF signal is important because we need to have a 50 Ω transmission line to minimize signal reflection. To determine the RF track width, use the built-in PCB calculator in KiCAD, shown in Fig 14. I chose to use Coplanar wave guide with ground plane because it gives relatively thin traces compared to having Microstrip lines. Note that you need to fill the top plate with ground for this method to work. First enter the substrate parameters given by the TA. On the right-hand side in Physical parameter, you can select W and S values

and calculate Z0. Conversely, you can specify Z0 and one of W and S parameters to calculate the other.

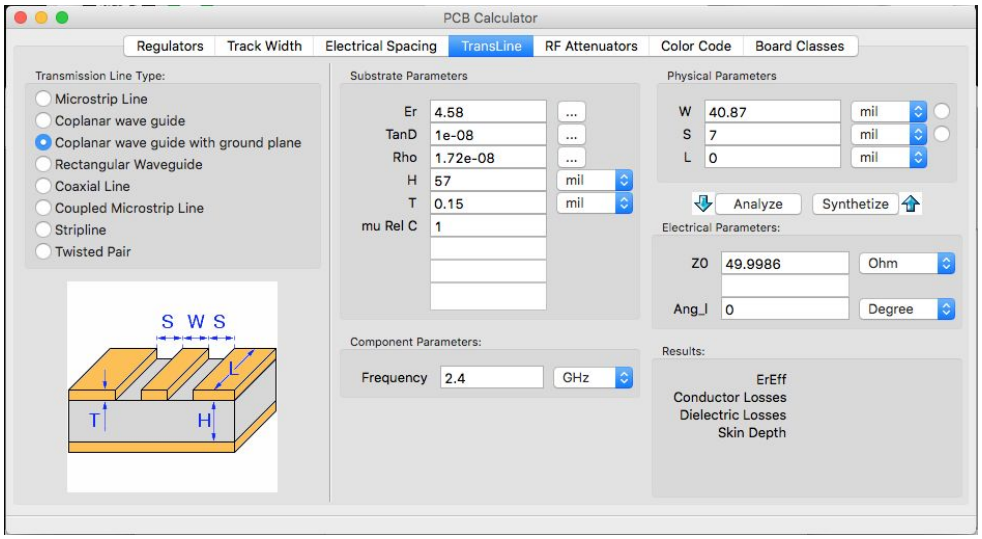


Fig 14: PCB calculator

At first, I tried to use 6 mil (minimum copper to copper width of Bay Area Circuit) as the separation. However, the DFM reported less than 6 mil (5.45mil) copper to copper width and never passed the DFM check, Fig 15. This is due to the fact that some copper fills do not have smooth round edges and the actually clearance is less than 6 mil. In the end, I switched to 7 mil clearances and passed the DFM check.

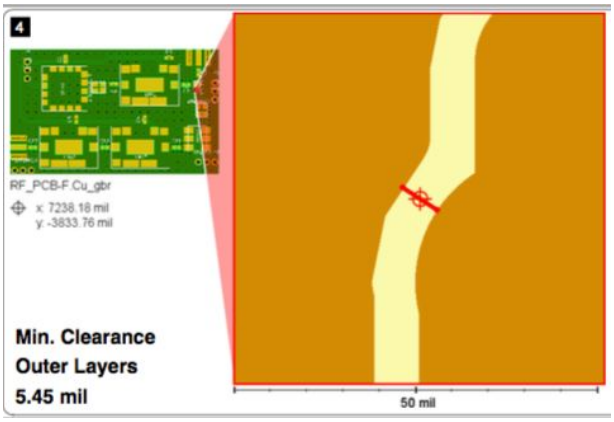


Fig 15: DFM min. Spacing error

In addition to selecting the correct track width, designing short and straight RF traces is another important thing to consider. Long and bending RF traces can create loss in the signal. Therefore, I centered the design of my RF PCB and

placement of components around achieving the shortest and never bending RF traces. Below is the final RF PCB design, Fig 16.

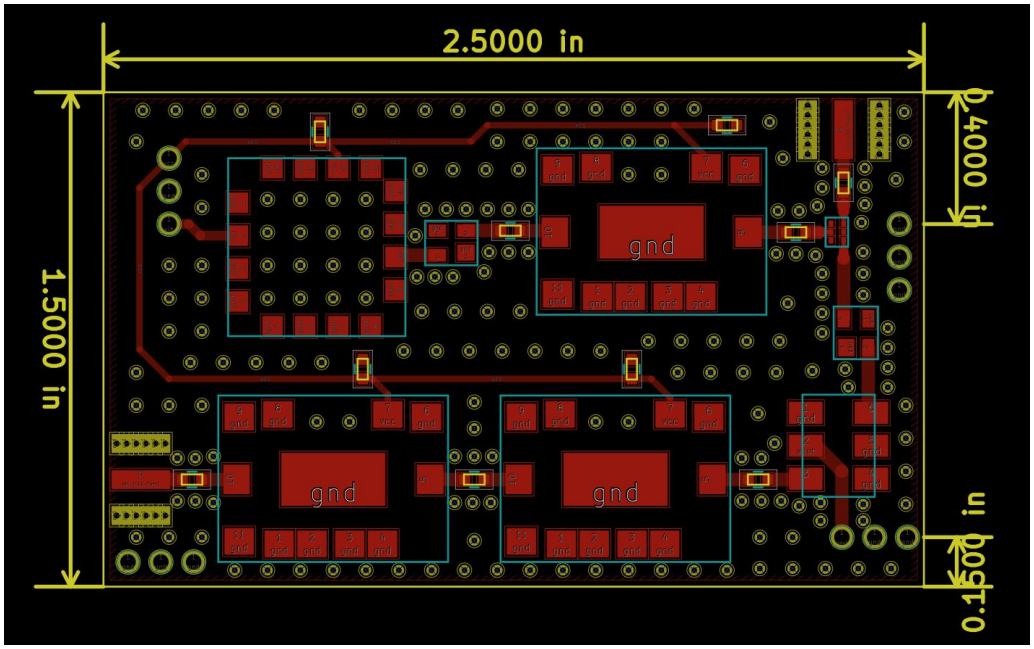


Fig 16. Final RF PCB layout

When placing the components, it is also helpful to use different grid sizes to line up pads of two adjacent components. Otherwise, the traces might have a very small bend at one end. When placing the via fences, use array tool to save time. Although there is no exact rule to how far via fences should be placed from the signal trace, I recommend leaving some space between via fences and track. I also created 4 sets of through-holes on each corner to connect to baseband PCB via pin headers. Vcc and V-tune signals are placed on the top left and IF signal is placed on lower right, with other connections connected to gnd. Having extra ground connections between RF and baseband PCB provides better electrical and mechanical stability so it is highly recommended.

Soldering

When soldering, one critical thing to pay attention to is having the correct orientation of components. For example, the VCO we used has a square footprint so it is easy to mistaken the orientation. So always consult the datasheet before placing down the component. In addition, some components can be very small (for

example our signal splitter), so be extra careful when applying solder paste to make sure that pads are not shorted together. Fig 17 shown below is the soldered RF PCB.

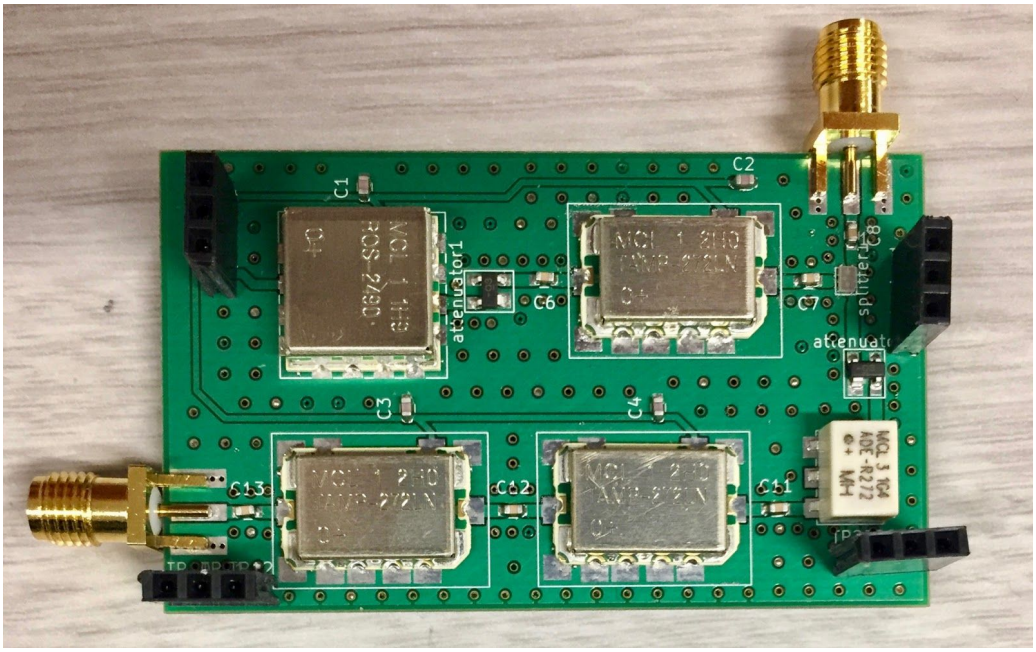


Fig 17: soldered RF PCB

Testing

RF band circuit: First step is to verify the transmitter part, which includes voltage-controlled oscillator (VCO), attenuator, low noise amplifier (LNA) and splitter. Based on our power budget calculation, we are expecting to get a power of 15.9dBm ideally at the antenna. Power on the RF circuit, set the Vtune to 5V, we are able to get 13.7dBm power (excluding 0.3dBm loss of SMA cables) measured by spectrum analyzer shown in Fig 18. Thus, it's reasonable to say that the transmitter is working properly.

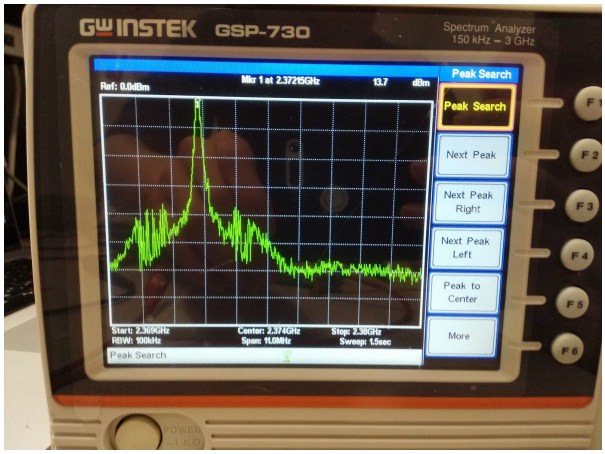


Fig 18: power measured at transmitter

Also, the VCO sensitivity is measured to get accurate result for the use of signal processing. As we change the Vtune, the output frequency changes accordingly as shown in Fig 19, where the relationship is pretty linear with a sensitivity of 33.1MHz/V.

Control Voltage/V	Output Frequency/GHz
0	2.2052
0.512	2.224
0.982	2.2352
1.4816	2.2494
2.075	2.2682
2.498	2.2818
3.061	2.3004
3.52	2.3162
4.052	2.3352
4.577	2.3544
5.082	2.3724
5.516	2.3876
6.015	2.4054
6.576	2.4246
7.019	2.4394
7.577	2.4578
8.015	2.4716
8.548	2.4882
8.984	2.501
9.521	2.5166

Sensitivity
33.1 MHz/V

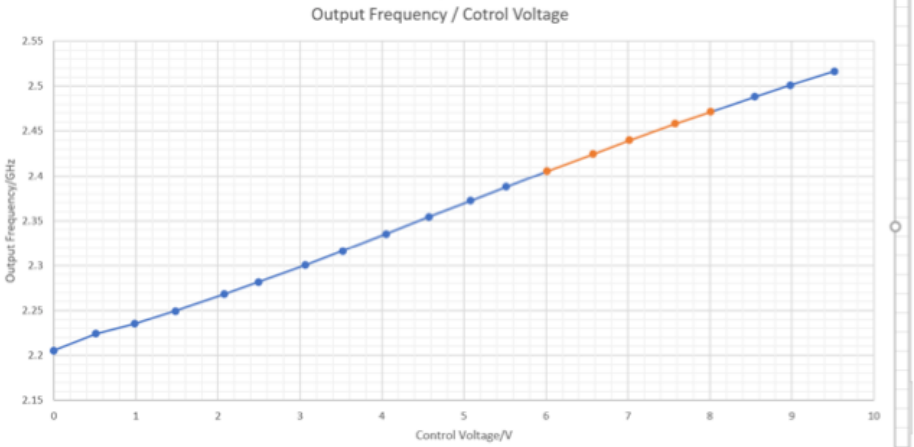


Fig 19: relation between Vtune and output frequency of VCO

To test the receiver part, which includes LNA and mixer, we set the Vtune to a specific DC (5V here), and use TPI synthesizer to generate a power of -35dBm at 2.47GHz. Taking LNA gain and mixer conversion loss into consideration, we are able to get -12.4dBm power at 100MHz as expected. So far, our RF works fine.

Baseband circuit: The baseband circuit should be able to give desired gain and sufficient bandwidth. Before we have mentioned the sensitivity of VCO to be 33.1MHz/V, Vtune is a 6 to 8V triangle wave at 25Hz. So the maximum frequency

that is fed in the baseband circuit would be (this happens when target is 50m away):

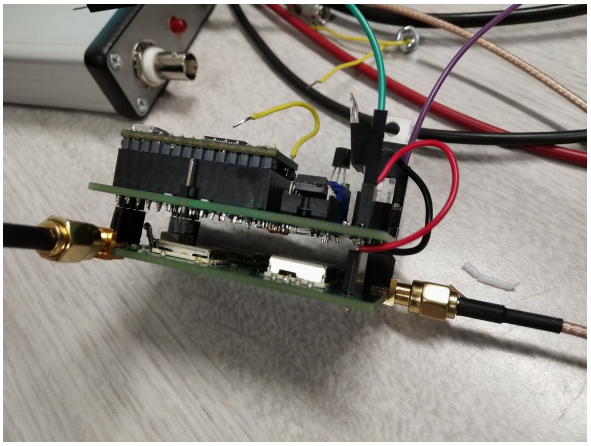
$$F\max = \frac{\frac{2*Range}{C}}{\frac{T}{2}} * \Delta V * Sensitivity = 1.1KHz$$

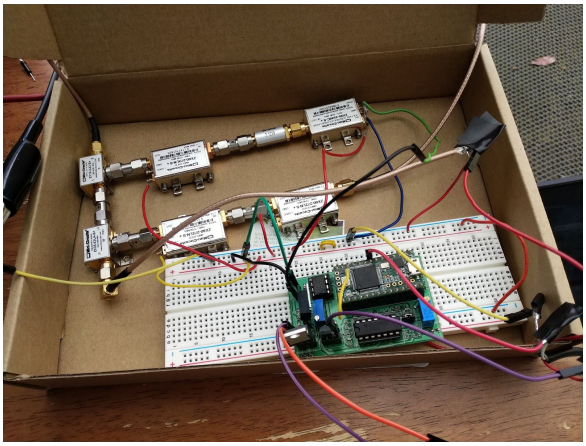
When we feed in a 100mVpp sinusoidal signal to baseband signal, the gain is adjusted to 40 at 1KHz (remain almost same for lower frequency), which results in a 4Vpp output. Vpp reduces to 2.8V when input frequency is around 6KHz. Thus, the bandwidth of the baseband circuit is 6 times of the Fmax, which is sufficient for this system.

Whole system testing:

We then compared the performance of baseband + RF PCB system (Fig 20a) against baseband PCB + quarter 1 RF system (Fig 20b). We found that baseband + quarter 1 RF system generally produced a cleaner graph than baseband + RF PCB. However, because the quarter 1 RF system is noticeably heavier than the RF PCB, we opted for the stacked baseband + PCB combination as planned in the beginning.

We also tested three combinations of antenna combinations: two yagi antennas, one coffee can on transmitter and one yagi on receiver, and two coffee cans. As a result, two-yagi-antenna combination produced very poor result. This is due to the fact that yagi antenna has worse directivity than coffee can. One coffee can and one yagi antenna combination produced similar results as two coffee cans. Therefore, we chose one coffee can and one yagi antenna as our final design because this system is significantly lighter than two coffee can setup.





(a) (b)

Fig 20: two system combinations

Result

Our whole radar system is shown below in Fig 21. We used a piece of styrofoam as the base because it is light and can provide the necessary mechanical support for the whole system. We used electric tape to secure all components and wires in place.

Fig 21: Whole radar system

Final testing result

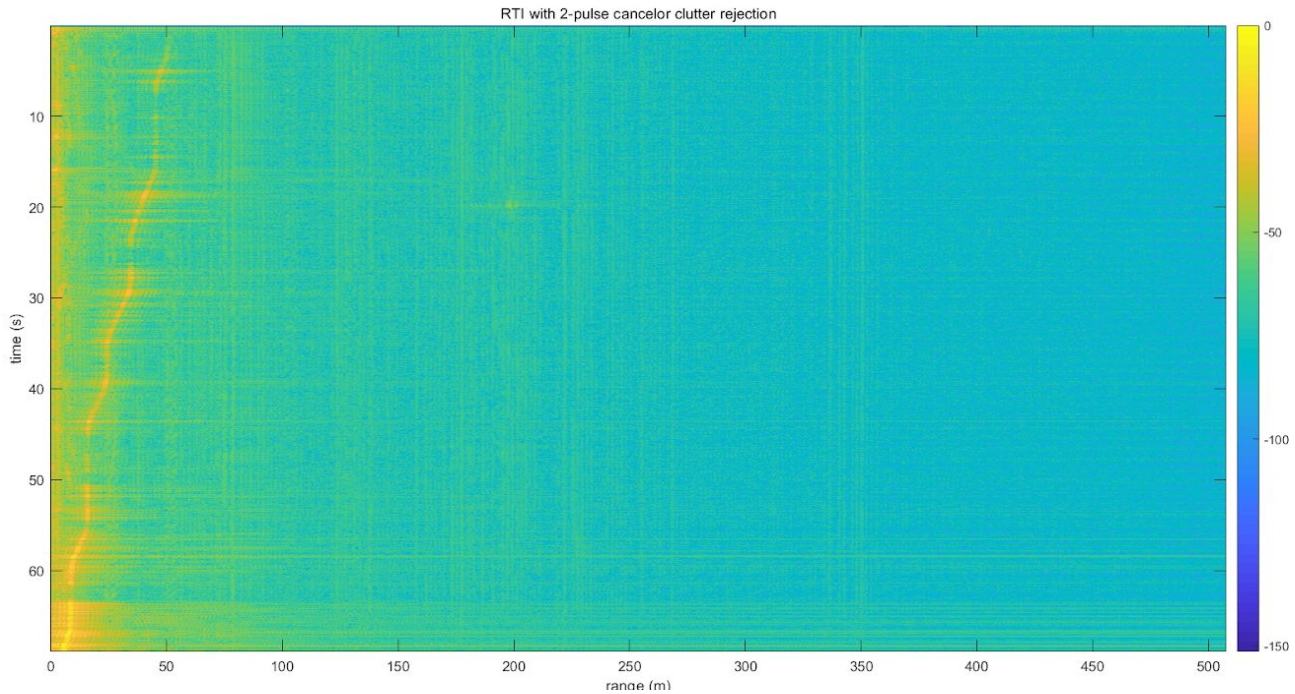


Fig 22: Testing result generated from wav file

Test result

Actual Distance	Our Measurement
43.7388	44.88
33.2232	34.27
22.86	24.07
14.478	15.1
7.3152	7.75

Discussion

From one quarter of designing, assembling and testing, we learned that building a fully working radar system requires a lot of hard work and thinking. We also learned to test each sub systems and come up with ways to test them to make sure they work before testing the whole system. Sometimes, even when all sub systems seem to be working, the whole system might not work in the way we like it to be. The key was to keep testing while making modifications to the system,

such as adjusting the gain in baseband circuit or modifying a piece of code to make sure all subsystems not only work on their own but also work together as a whole.

Mistakes are bound to happen when designing a new system, but it is important to come up with solutions or remedies to the situation. Whether it is to add a wire, re-solder a PCB, redesign a new PCB or use the back-up system, we need to consider the severity of the situation and time allowed. I think we learned a lot about fixing our mistakes throughout the design of the radar system.

Even though we decided to do on-board processing at the beginning of quarter and able to find the frequency using FFT, we found out that the our current algorithm cannot detect the distance due to the system we are using has to involve some doppler feature. After reading the python code provided in quarter 1 and the matlab code from Team One in year 2016-2017, we figured out that we need to sample multiple set of samples and do dynamic frequency calculation to find out the distance. The processing time would be too long to capture all distance changes of the TA with on-board process. Thus, we decided to switched to matlab code from Team One and instead of using "real-time" which periodically record audio and perform processing, we decided to using audacity to record the entire motion of the TA and perform process for the entire recording. We actually did field test using rulers to measure distance and calibrated our coefficient to match the actual distance measured. However, due to the line generated from the wav file have a length corresponding to field distance of 3 meters (from the left of line to the right of line on the same horizontal level), our final result still have error due to the actual point selected on the graph.

BOM

Component	Model	Unit Price	Quantity used	Price	Quantity to order	Price	Link
VCO	ROS-2536C-199+	$ 23.95	1	$23.95	2	$47.90	https://www.minicircuits.com/WebStore/dashboard.html?model=ROS-2536C-119%2B

Attenuator	LAT 3+	$ 2.15	2	$4.30	2	$4.30	https://www.minicircuits.com/WebStore/dashboard.html?model=LAT-3%2B
LNA	TAMP-272 LN+	$ 14.95	3	$44.85	5	$74.75	https://www.minicircuits.com/WebStore/dashboard.html?model=TAMP-272LN%2B
Splitter	Anaren PD2328J5050S2HF	$ 0.74	1	$0.74	2	$1.48	https://www.digikey.com/product-detail/en/anaren/PD2328J5050S2HF/1173-1098-1-ND/3069297
Mixer	ADE-R3GLH	$ 4.85	1	$4.85	2	$9.70	https://www.minicircuits.com/WebStore/dashboard.html?model=ADE-R3GLH%2B
Amplifier	TL974IN	$ 0.98	3	$2.94	6	$5.88	https://www.digikey.com/products/en?mpart=TL974IN&v=296
Yagi Antenna		$ 6.00	2	$12.00	2	$12.00	http://wa5vjb.com/products2.html
Voltage regulator	LM2937	$ 1.69	1	$1.69	2	$3.38	https://www.digikey.com/product-detail/en/texas-instruments/LM2937ET-5.0-NOPB/LM2937ET-5.0-NOPB-ND/212651
Diode	VSMF2893RGX01CT-ND	$ 1.09	1	$1.09	2	$2.18	https://www.digikey.com/product-detail/en/vishay-semiconductor-opto-division/VSMF2893RGX01/VSMF2893RGX01CT-ND/5323940
Total				$96.41		$161.57	

CC3200 Code

```
// Standard includes
#include <string.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>

// Driverlib includes
#include "utils.h"
#include "hw_memmap.h"
#include "hw_common_reg.h"
#include "hw_types.h"
#include "hw_adc.h"
#include "hw_ints.h"
#include "hw_gprcm.h"
#include "rom.h"
#include "rom_map.h"
#include "interrupt.h"
#include "prcm.h"
#include "uart.h"
#include "pin_mux_config.h"
#include "pin.h"
#include "adc.h"
#include "adc_userinput.h"
#include "uart_if.h"
#include "gpio.h"

#include "complex.h"
#include "spi.h"
#include "Adafruit_GFX.h"
#include "Adafruit_SSD1351.h"
#include "test.h"
#include "coeff.h"

#define USER_INPUT
#define UART_PRINT        Report
#define FOREVER           1
#define APP_NAME          "ADC Reference"
#define NO_OF_SAMPLES     1024

#define PI 3.14159265358979323846264338
#define GP_base GPIOA2_BASE
#define GP_pin 0x40
```

```
unsigned char GP_flag;
unsigned long pulAdcSamples[1024];
double result[1024];
short sample[1024];

#define MASTER_MODE      1

#define SPI_IF_BIT_RATE  100000


//*****************************************************************************
//                    GLOBAL VARIABLES
//*****************************************************************************
#if defined(ccs)
extern void (* const g_pfnVectors[])(void);
#endif
#if defined(ewarm)
extern uVectorEntry __vector_table;
#endif

/*****************************************************************************/
/*                    LOCAL FUNCTION PROTOTYPES                              */
/*****************************************************************************/
static void BoardInit(void);
//static void DisplayBanner(char * AppName);
static void FFT_CooleyTukey(int N, int N1, int N2);
static void GPIntHandler(void);
//*****************************************************************************
//
//! Application startup display on UART
//!
//! \param  none
//!
//! \return none
//!
//*****************************************************************************
/*
static void
DisplayBanner(char * AppName)
{
    Report("\n\n\n\r");
    Report("\t\t *************************************************\n\r");
    Report("\t\t       CC3200 %s Application       \n\r", AppName);
```

```
    Report("\t\t **********************************************\n\r");
    Report("\n\n\n\r");
}
*/
//*****************************************************************************
//
//! Board Initialization & Configuration
//!
//! \param  None
//!
//! \return None
//
//*****************************************************************************
static void
BoardInit(void)
{
/* In case of TI-RTOS vector table is initialize by OS itself */
#ifndef USE_TIRTOS
    //
    // Set vector table base
    //
#if defined(ccs)
    MAP_IntVTableBaseSet((unsigned long)&g_pfnVectors[0]);
#endif
#if defined(ewarm)
    MAP_IntVTableBaseSet((unsigned long)&__vector_table);
#endif
#endif
    //
    // Enable Processor
    //
    MAP_IntMasterEnable();
    MAP_IntEnable(FAULT_SYSTICK);

    PRCMCC3200MCUInit();
}
//*****************************************************************************
static void
Total_Initial(void) {
    // Enable the SPI module clock
    MAP_PRCMPeripheralClkEnable(PRCM_GSPI,PRCM_RUN_MODE_CLK);
    // Reset the peripheral
    MAP_PRCMPeripheralReset(PRCM_GSPI);
    // Reset SPI
```

```
    MAP_SPIReset(GSPI_BASE);
    // Configure SPI interface
    MAP_SPIConfigSetExpClk(GSPI_BASE,MAP_PRCMPeripheralClockGet(PRCM_GSPI),
                           SPI_IF_BIT_RATE,SPI_MODE_MASTER,SPI_SUB_MODE_0,
                           (SPI_SW_CTRL_CS |
                           SPI_4PIN_MODE |
                           SPI_TURBO_OFF |
                           SPI_CS_ACTIVEHIGH |
                           SPI_WL_8));
    MAP_SPIEnable(GSPI_BASE);
    Adafruit_Init();
    fillScreen(BLACK);
    setTextColor(BLUE,1);

    // Configure ADC timer which is used to timestamp the ADC data samples
    MAP_ADCTimerConfig(ADC_BASE,2^17);

    // Register the interrupt handlers
    MAP_GPIOIntRegister(GP_base, GPIntHandler);

    // Configure rising edge interrupts on SW2 and SW3
    MAP_GPIOIntTypeSet(GP_base, GP_pin, GPIO_RISING_EDGE);

}

//*****************************************************************************
//** GP_IntHandler
//*****************************************************************************
static void GPIntHandler(void) {
    unsigned long ulStatus;
    ulStatus = MAP_GPIOIntStatus (GP_base, true);
    MAP_GPIOIntClear(GP_base, ulStatus); // clear interrupts
    GP_flag = 1;
}


//*****************************************************************************
//** Implements the Cooley-Tukey FFT algorithm.
//*****************************************************************************
static
void FFT_CooleyTukey(int N, int N1, int N2) {
    int k1, k2;
    int k, n;
    /* Allocate columnwise matrix */
```

```
   signed long long** columns_real = (signed long long**) malloc(sizeof(signed long long*) *
N1);
   for(k1 = 0; k1 < N1; k1++) {
      columns_real[k1] = (signed long long*) malloc(sizeof(signed long long) * N2);
   }

   /* Reshape input into N1 columns */
   for (k1 = 0; k1 < N1; k1++) {
      for(k2 = 0; k2 < N2; k2++) {
         columns_real[k1][k2] = (signed long long) sample[N1*k2 + k1];
      }
   }

   complex** columns = (complex**) malloc(sizeof(struct complex_t*) * N1);
   for(k1 = 0; k1 < N1; k1++) {
      columns[k1] = (complex*) malloc(sizeof(struct complex_t) * N2);
   }

   /* Compute N1 DFTs of length N2 using naive method */
   for (k1 = 0; k1 < N1; k1++) {
      //columns[k1] = DFT_naive(columns[k1], N2);
      for(k = 0; k < N2; k++) {
         columns[k1][k].re = 0;
         columns[k1][k].im = 0;
         for(n = 0; n < N2; n++) {
            columns[k1][k].re = columns[k1][k].re + columns_real[k1][n] * cos_Naive[k][n];
            columns[k1][k].im = columns[k1][k].im + columns_real[k1][n] * sin_Naive[k][n];
         }
      }
         free(columns_real[k1]);
   }
   free(columns_real);

   /* Allocate rowwise matrix */
   complex ** rows = (complex**) malloc(sizeof(struct complex_t*) * N2);
   for(k2 = 0; k2 < N2; k2++) {
      rows[k2] = (complex*) malloc(sizeof(struct complex_t) * N1);
   }

   /* Multiply by the twiddle factors  ( e^(-2*pi*j/N * k1*k2)) and transpose */
   for(k1 = 0; k1 < N1; k1++) {
      for (k2 = 0; k2 < N2; k2++) {
         rows[k2][k1].re = (columns[k1][k2].re*cos_twiddle[k1][k2] -
columns[k1][k2].im*sin_twiddle[k1][k2]) >> 14;
```

```
            rows[k2][k1].im = (columns[k1][k2].re*sin_twiddle[k1][k2] +
columns[k1][k2].im*cos_twiddle[k1][k2]) >> 14;
        }
        free(columns[k1]);
    }
    free(columns);

    complex* X_row = (complex*) malloc(sizeof(struct complex_t) * N1);
    /* Compute N2 DFTs of length N1 using naive method */
    for (k2 = 0; k2 < N2; k2++) {
        //rows[k2] = DFT_naive(rows[k2], N1);
        for(k = 0; k < N1; k++) {
            X_row[k].re = 0.0;
            X_row[k].im = 0.0;
            for(n = 0; n < N1; n++) {
                X_row[k].re = X_row[k].re + ((rows[k2][n].re*cos_Naive[k][n] -
rows[k2][n].im*sin_Naive[k][n]) >> 14);
                X_row[k].im = X_row[k].im + ((rows[k2][n].im*cos_Naive[k][n] +
rows[k2][n].re*sin_Naive[k][n]) >> 14);
            }
        }
        for(n = 0; n < N1; n++) rows[k2][n] = X_row[n];
    }
    free(X_row);

    /* Flatten into single output */
    for(k1 = 0; k1 < N1; k1++) {
        for (k2 = 0; k2 < N2; k2++) {
            result[N2*k1 + k2] = mag(rows[k2][k1]);
        }
    }

    for(k2 = 0; k2 < N2; k2++) free(rows[k2]);
    free(rows);

    return;
}




//*****************************************************************************
//
//! main - calls Crypt function after populating either from pre- defined vector
```

```
//! or from User
//!
//! \param  none
//!
//! \return none
//!
//*****************************************************************************
void
main()
{

   // Initialize Board configurations
   BoardInit();
   PinMuxConfig();
   Total_Initial();
   int i;
   unsigned int  uiChannel = ADC_CH_3;
   unsigned long ulSample;
   unsigned long ulStatus;
   unsigned char sample_flag = 0;
   unsigned short max_index;
   double max_val;
   int freq;
   GP_flag = 0;
   char frequency[4];

   ulStatus = MAP_GPIOIntStatus(GP_base, false);
   MAP_GPIOIntClear(GP_base, ulStatus);
   MAP_GPIOIntEnable(GP_base, GP_pin);

   while(FOREVER)
   {
      if (GP_flag > 0){
         MAP_GPIOIntDisable(GP_base, GP_pin);
         fillRect(0,0,40,40, BLACK);
         setCursor(0, 0);
         Outstr("start");

#ifdef CC3200_ES_1_2_1
      // Enable ADC clocks.###IMPORTANT###Need to be removed for PG 1.32
      HWREG(GPRCM_BASE + GPRCM_O_ADC_CLK_CONFIG) = 0x00000043;
      HWREG(ADC_BASE + ADC_O_ADC_CTRL) = 0x00000004;
      HWREG(ADC_BASE + ADC_O_ADC_SPARE0) = 0x00000100;
      HWREG(ADC_BASE + ADC_O_ADC_SPARE1) = 0x0355AA00;
```

```
#endif

        // Enable ADC timer which is used to timestamp the ADC data samples
        MAP_ADCTimerEnable(ADC_BASE);

        // Enable ADC module
        MAP_ADCEnable(ADC_BASE);

        // Enable ADC channel
        MAP_ADCChannelEnable(ADC_BASE, uiChannel);

        i = 0;
        sample_flag = 0;
        while (i < 1024){
           if(MAP_ADCFIFOLvlGet(ADC_BASE, uiChannel)) {
              ulSample = MAP_ADCFIFORead(ADC_BASE, uiChannel);
              sample_flag++;
              if (sample_flag == 20){
                 pulAdcSamples[i] = ulSample;
                 i++;
                 sample_flag = 0;
              }
           }
        }

        MAP_ADCChannelDisable(ADC_BASE, uiChannel);

        for (i=0;i<1024;i++) sample[i] = ((pulAdcSamples[i] >> 2 ) & 0x0FFF);

        //for (i=0;i<1024;i++) sample[i] = (signed short) (cos(0.02*i)*5+5);
        FFT_CooleyTukey(1024, 32, 32);
        max_val = result[1];
        max_index = 1;
        for (i=2;i<512;i++){
           if (result[i] > max_val){
              max_index = i;
              max_val = result[i];
           }
        }
        freq = max_index * 3125 / 1024;
        frequency[0] = 48+freq/1000;
        frequency[1] = 48 + (freq%1000)/100;
        frequency[2] = 48 + (freq%100)/10;
        frequency[3] = 48 + (freq%10);
```

```
        setCursor(0, 10);
        Outstr(frequency);
        setCursor(0, 20);
        Outstr("end");

        GP_flag--;
        if (GP_flag==0) {
           ulStatus = MAP_GPIOIntStatus(GP_base, false);
           MAP_GPIOIntClear(GP_base, ulStatus);
           MAP_GPIOIntEnable(GP_base, GP_pin);
        }
     }
  }
}
```