EEL4511 Real-time DSP Applications
Lab 9 Final Project

Title: Real-Time Pitch Shifter

First Name: Shida **Last Name:** Yang

Abstract:

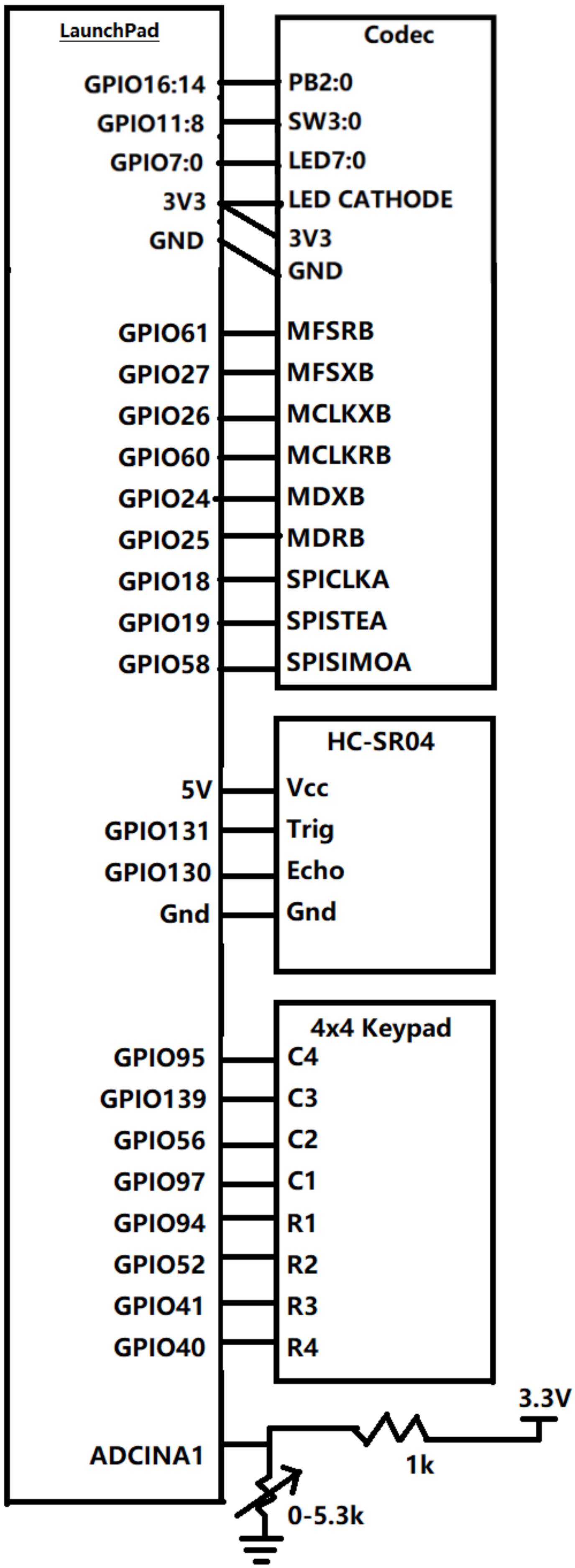
The main goal of this project is to create a real-time pitch shifter that uses a phase vocoder to do time stretching and uses interpolation to shift pitch.

Features:

1. Fs=48kHz
2. 512-point FFT, with 50% overlap
3. Real-time phase vocoder
    a. Interpolation in the frequency domain
    b. Switch between rectangular and polar coordinate
    c. Handles up to 2x time stretching in real-time
4. Pitch shifting control:
    a. HC-SR04 distance sensor mode
        i. 0m < distance < 1m : linear pitch-shifting increment from 0.5x to 2x continuously
        ii. Distance > 1m: pitch shifting = 1x
    b. 4x4 Keypad mode
        i. Continuous mode: when no key is pressed, play original sound (1x)
        ii. Non-continuous mode: when no key is pressed, mute output
        iii. In both modes: pitch shifting = 0.5+0.1*keyNum
            1. Range from 0.5x to 2x
            2. Increment is discrete
5. Volume control:
    a. Uses an ADC to sample the voltage on a potentiometer
    b. Volume increases quadratically with the voltage
6. A bug I was not able to fix:
    a. The average output rate of my algorithm is slightly different from the sampling frequency due to rounding error. Therefore, occasionally, the output DMA will collide with my writing to the output buffer, which results in a short-disordered sound.

b. **Remedy: I used a huge output buffer (16384 floating points) to make this happen less frequently, but I was not able to completely eliminate it.**

Grade:


LaunchPad
GPIO16:14
GPIO11:8
GPIO7:0
3V3
GND
GPIO61
GPIO27
GPIO26
GPIO60
GPIO24
GPIO25
GPIO18
GPIO19
GPIO58
5V
GPIO131
GPIO130
Gnd
GPIO95
GPIO139
GPIO56
GPIO97
GPIO94
GPIO52
GPIO41
GPIO40
ADCINA1
Codec
PB2:0
SW3:0
LED7:0
LED CATHODE
3V3
GND
MFSRB
MFSXB
MCLKXB
MCLKRB
MDXB
MDRB
SPICLKA
SPISTEA
SPISIMOA
HC-SR04
Vcc
Trig
Echo
Gnd
4x4 Keypad
C4
C3
C2
C1
R1
R2
R3
R4
3.3V
1k
0-5.3k

```c
#include <F28x_Project.h>

#include "SPIA.h"
#include "AIC23.h"
#include "Mcbsp.h"
#include "interrupt.h"
#include "math.h"
#include "cputimer.h"
#include "fpu_cfft.h"
#include "math.h"
#include "myAdc.h"

#define FFT_SIZE 512
#define FFT_STAGES 9
#define HOP_SIZE 256
#define TRIG_RATIO 81.48733086
#define PI 3.1415926
#define ONE_OVER_TWO_PI 0.15915494309
#define OUT_BUF_MASK 0x3FFF
#define TIMER1_DISTANCE_RES 73529 //resolution for 0.1x, 1m=2x, 0m=0.5x

__interrupt void dmaCh1ISR(void);
__interrupt void ECHO_ISR(void);
__interrupt void Timer0_ISR(void);
__interrupt void Timer1_ISR(void);
__interrupt void Timer2_ISR(void);
__interrupt void adcA1ISR();

void initDMAInput();
void initDMAOutput();
void initFFT();
void initGPIO();
void initTimer0();
void initTimer1();
void initTimer2();

void doHanning(float* inBuf);
void doFFT(float* inBuf, float* outMagnitude, float* outPhase);
void freqInterpolate(float* mag1, float* phase1, float* mag2, float* phase2, float right,
                     float* magOut, float* phaseOut);
void convertToRect(float* mag, float* phase, float* rect);
void doIFFT(float* inBuf, float* outBuf);
Uint16 timeInterpolate(float* values1, float* values2, Uint16 len, float ratio, float* output);
Uint16 checkKey();

float mySin(float rad);
float myCos(float rad);

volatile float pitchShiftRatio=1.3;
volatile float amp=1;

// CFFT_F32_STRUCT object
CFFT_F32_STRUCT cfft;
// Handle to the CFFT_F32_STRUCT object
CFFT_F32_STRUCT_Handle hnd_cfft = &cfft;

volatile Uint16 currInBuf=0;
volatile float currInterPos=0;
volatile Uint16 outBufCounter=0;
volatile Uint16 insideOutBufCounter=0;
volatile float currPos=0;
volatile bool firstTime=1;
volatile int32 startTime=0;
volatile int32 endTime=0;
volatile bool timeOut=0;
volatile Uint16 key=17;

volatile bool received256=0;
volatile bool volumeFlag=0;
volatile bool distanceFlag=0;
volatile bool isDistance=1;
volatile bool keypad_cont=0;
volatile bool keypadFlag=0;

#pragma DATA_SECTION(oneside, "FFT_INBUF");
volatile float oneside[FFT_SIZE << 1];

#pragma DATA_SECTION(inBuf0, "SOUND_INBUF");
#pragma DATA_SECTION(inBuf1, "SOUND_INBUF");
#pragma DATA_SECTION(inBuf2, "SOUND_INBUF");
volatile int32 inBuf0[FFT_SIZE];
volatile int32 inBuf1[FFT_SIZE];
volatile int32 inBuf2[FFT_SIZE];
volatile int32* currInPtr=&inBuf0[0];
volatile int32* prevInPtr=&inBuf2[0];

#pragma DATA_SECTION(magA, "MAG_BUF");
#pragma DATA_SECTION(magB, "MAG_BUF");
#pragma DATA_SECTION(magOut, "MAG_PHASE_OUT_BUF");
volatile float magA[FFT_SIZE];
volatile float magB[FFT_SIZE];
volatile float magOut[FFT_SIZE];
volatile float* leftMag=&magA[0];
volatile float* rightMag=&magB[0];

#pragma DATA_SECTION(phaseA, "PHASE_BUF");
#pragma DATA_SECTION(phaseB, "PHASE_BUF");
#pragma DATA_SECTION(phaseOut, "MAG_PHASE_OUT_BUF");
volatile float phaseA[FFT_SIZE];
volatile float phaseB[FFT_SIZE];
volatile float phaseOut[FFT_SIZE];
volatile float* leftPhase=&phaseA[0];
volatile float* rightPhase=&phaseB[0];

#pragma DATA_SECTION(phaseAccum, "RECT_BUF");
volatile float phaseAccum[FFT_SIZE];

#pragma DATA_SECTION(rect, "RECT_BUF");
volatile float rect[FFT_SIZE*2];

#pragma DATA_SECTION(ifftOutA, "IFFT_OUT_BUF");
#pragma DATA_SECTION(ifftOutB, "IFFT_OUT_BUF");
volatile float ifftOutA[FFT_SIZE << 1];
volatile float ifftOutB[FFT_SIZE << 1];
volatile float* currIfftOut=&ifftOutA[0];
volatile float* prevIfftOut=&ifftOutB[0];

#pragma DATA_SECTION(outBuf0, "OUT_BUF0");
volatile int32 outBuf0[2048];
#pragma DATA_SECTION(outBuf1, "OUT_BUF1");
volatile int32 outBuf1[2048];
#pragma DATA_SECTION(outBuf2, "OUT_BUF2");
volatile int32 outBuf2[2048];
```

```c
#pragma DATA_SECTION(outBuf3, "OUT_BUF3");
volatile int32 outBuf3[2048];
#pragma DATA_SECTION(outBuf4, "OUT_BUF4");
volatile int32 outBuf4[2048];
#pragma DATA_SECTION(outBuf5, "OUT_BUF5");
volatile int32 outBuf5[2048];
#pragma DATA_SECTION(outBuf6, "OUT_BUF6");
volatile int32 outBuf6[2048];
#pragma DATA_SECTION(outBuf7, "OUT_BUF7");
volatile int32 outBuf7[2048];

volatile int32* outBufs[8]={
    &outBuf0[0],
    &outBuf1[0],
    &outBuf2[0],
    &outBuf3[0],
    &outBuf4[0],
    &outBuf5[0],
    &outBuf6[0],
    &outBuf7[0]
};

#pragma DATA_SECTION(twiddleFactors, "TWIDDLE_TEST_BUF");
#pragma DATA_SECTION(test_output, "TWIDDLE_TEST_BUF");
volatile float test_output[FFT_SIZE << 1];

volatile float twiddleFactors[FFT_SIZE << 1];

float dph[FFT_SIZE/2+1]={
    0.000000000000000000,
    3.141592653589793116,
    6.283185307179586232,
    9.424777960769379348,
    12.566370614359172464,
    15.707963267948965580,
    18.849555921538758696,
    21.991148575128551812,
    25.132741228718344928,
    28.274333882308138044,
    31.415926535897931160,
    34.557519189487720723,
    37.699111843077517392,
    40.840704496667306955,
    43.982297150257103624,
    47.123889803846900293,
    50.265482457436689856,
    53.407075111026486525,
    56.548667764616276088,
    59.690260418206072757,
    62.831853071795862320,
    65.973445725385658989,
    69.115038378975441447,
    72.256631032565238115,
    75.398223686155034784,
    78.539816339744831453,
    81.681408993334613911,
    84.823001646924424790,
    87.964594300514207248,
    91.106186954104003917,
    94.247779607693800585,
    97.389372261283583043,
    100.530964914873379712,
    103.672557568463176381,
    106.814150222052973049,
    109.955742875642755507,
    113.097335529232552176,
    116.238928182822334634,
    119.380520836412145513,
    122.522113490001942182,
    125.663706143591724640,
    128.805298797181507098,
    131.946891450771317977,
    135.088484104361100435,
    138.230076757950882893,
    141.371669411540693773,
    144.513262065130476230,
    147.654854718720287110,
    150.796447372310069568,
    153.938040025899880447,
    157.079632679489662905,
    160.221225333079445363,
    163.362817986669227821,
    166.504410640259038701,
    169.646003293848849580,
    172.787595947438632038,
    175.929188601028414496,
    179.070781254618225375,
    182.212373908208007833,
    185.353966561797790291,
    188.495559215387601171,
    191.637151868977383629,
    194.778744522567166086,
    197.920337176156976966,
    201.061929829746759424,
    204.203522483336541882,
    207.345115136926352761,
    210.486707790516135219,
    213.628300444105946099,
    216.769893097695728557,
    219.911485751285511014,
    223.053078404875293472,
    226.194671058465104352,
    229.336263712054915231,
    232.477856365644669268,
    235.619449019234480147,
    238.761041672824291027,
    241.902634326414045063,
    245.044226980003884364,
    248.185819633593666822,
    251.327412287183449280,
    254.469004940773260159,
    257.610597594363014196,
    260.752190247952796653,
    263.893782901542635955,
    267.035375555132418413,
    270.176968208722200870,
    273.318560862311983328,
    276.460153515901765786,
    279.601746169491605087,
    282.743338823081387545,
    285.884931476671170003,
    289.026524130260952461,
    292.168116783850734919,
    295.309709437440574220,
    298.451302091030356678,
    301.592894744620139136,
    304.734487398209921594,
    307.876080051799760895,
    311.017672705389486509,
    314.159265358979325811,
    317.300858012569108269,
    320.442450666158890726,
    323.584043319748730028,
    326.725635973338455642,
    329.867228626928238100,
    333.008821280518077401,
    336.150413934107916702,
    339.292006587697699160,
    342.433599241287424775,
    345.575191894877264076,
    348.716784548467046534,
    351.858377202056828992,
    354.999969855646611450,
    358.141562509236450751,
    361.283155162826176365,
    364.424747816416015667,
    367.566340470005741281,
    370.707933123595580582,
    373.849525777185419884,
    376.991118430775202341,
    380.132711084364984799,
    383.274303737954767257,
    386.415896391544492872,
    389.557489045134332173,
    392.699081698724114631,
    395.840674352313953932,
    398.982267005903736390,
    402.123859659493518848,
    405.265452313083301306,
    408.407044966673083763,
    411.548637620262923065,
    414.690230273852705523,
    417.831822927442487980,
    420.973415581032270438,
    424.115008234622052896,
    427.256600888211892197,
    430.398193541801674655,
    433.539786195391457113,
    436.681378848981239571,
    439.822971502571022029,
    442.964564156160861330,
    446.106156809750586945,
    449.247749463340426246,
    452.389342116930208704,
    455.530934770519991162,
    458.672527424109830463,
    461.814120077699612921,
    464.955712731289338535,
    468.097305384879177836,
    471.238898038468960294,
    474.380490692058742752,
    477.522083345648582053,
    480.663675999238307668,
    483.805268652828090126,
    486.946861306417929427,
    490.088453960007768728,
    493.230046613597494343,
    496.371639267187333644,
    499.513231920777059258,
    502.654824574366898560,
    505.796417227956737861,
    508.938009881546520319,
    512.079602535136245933,
    515.221195188726028391,
    518.362787842315810849,
    521.504380495905593307,
    524.645973149495375765,
    527.787565803085271909,
    530.929158456675054367,
    534.070751110264836825,
    537.212343763854619283,
    540.353936417444401741,
    543.495529071034184199,
    546.637121724623966657,
    549.778714378213749114,
    552.920307031803531572,
    556.061899685393427717,
    559.203492338983210175,
    562.345084992572992633,
    565.486677646162775090,
    568.628270299752557548,
    571.769862953342340006,
    574.911455606932122464,
    578.053048260521904922,
    581.194640914111687380,
    584.336233567701469838,
    587.477826221291365982,
    590.619418874881148440,
    593.761011528470930898,
    596.902604182060713356,
    600.044196835650495814,
    603.185789489240278272,
    606.327382142830060729,
    609.468974796419843187,
    612.610567450009625645,
    615.752160103599521790,
    618.893752757189304248,
    622.035345410778973019,
    625.176938064368869163,
    628.318530717958651621,
    631.460123371548434079,
    634.601716025138216537,
    637.743308678727998995,
    640.884901332317781453,
    644.026493985907563911,
    647.168086639497460055,
    650.309679293087242513,
    653.451271946676911284,
    656.592864600266807429,
    659.734457253856476200,
    662.876049907446258658,
    666.017642561036154802,
    669.159235214625937260,
    672.300827868215833405,
    675.442420521805502176,
    678.584013175395398321,
    681.725605828985067092,
    684.867198482574849550,
    688.008791136164745694,
    691.150383789754528152,
    694.291976443344196923,
    697.433569096934093068,
    700.575161750523875526,
    703.716754404113657984,
    706.858347057703440441,
    709.999939711293222899,
    713.141532364883005357,
    716.283125018472901502,
    719.424717672062683960,
    722.566310325652352731,
    725.707902979242248875,
    728.849495632832031333,
    731.991088286421813791,
    735.132680940011482562,
    738.274273593601378707,
    741.415866247191161165,
    744.557458900781057309,
    747.699051554370839767,
    750.840644207960622225,
    753.982236861550404683,
    757.123829515140187141,
    760.265422168729969599,
    763.407014822319638370,
    766.548607475909534514,
    769.690200129499203285,
    772.831792783088985743,
    775.973385436678881888,
    779.114978090268664346,
    782.256570743858560490,
    785.398163397448229261,
    788.539756051038125406,
    791.681348704627907864,
    794.822941358217690322,
    797.964534011807472780,
    801.106126665397255238,
    804.247719318987037695
};

const float hann[FFT_SIZE] = {
    0.000000000000000000,
    0.000037796577274096,
    0.000151180594771427,
    0.000340134910380874,
    0.000604630956796859,
    0.000944628745838338,
    0.001360076874494465,
    0.001850912532696092,
    0.002417061512811680,
    0.003058438220866544,
    0.003774945689483389,
    0.004566475592542640,
    0.005432908261559732,
    0.006374112703777302,
    0.007389946621969679,
    0.008480256435956068,
    0.009644877305819977,
    0.010883633156830497,
    0.012196336706062738,
    0.013582789490712122,
    0.015042781898099433,
    0.016576093197361197,
    0.018182491572821535,
    0.019861734159038968,
    0.021613567077524876,
    0.023437725475126125,
    0.025333933564067435,
    0.027301904663646570,
    0.029341341243576458,
    0.031451934968968087,
    0.033633366746946003,
    0.035885306774891212,
    0.038207414590302524,
    0.040599339122270095,
    0.043060718744552196,
    0.045591181330248531,
    0.048190344308060407,
    0.050857814720130512,
    0.053593189281452569,
    0.056396054440842724,
    0.059265986443462593,
    0.062202551394885286,
    0.065205305326694496,
    0.068273794263606080,
    0.071407554292103159,
    0.074606111630573402,
    0.077868982700938449,
    0.081195674201764101,
    0.084585683182840765,
    0.088038497121222858,
    0.091553593998715044,
    0.095130442380794267,
    0.098768501496955430,
    0.102467221322468549,
    0.106226042661534792,
    0.110044397231829294,
    0.113921707750417878,
    0.117857388021033904,
    0.121850843022703492,
    0.125901468999704391,
    0.130008653552845688,
    0.134171775732053911,
    0.138390206130252547,
    0.142663306978519533,
    0.146990432242509073,
    0.151370927720123671,
    0.155804131140419966,
    0.160289372263735408,
    0.164825972983019098,
    0.169413247426352498,
    0.174050502060643708,
    0.178737035796480204,
    0.183472140094123992,
    0.188255099070633203,
    0.193085189608093566,
    0.197961681462944072,
    0.202883837376379883,
    0.207850913185815778,
    0.212862157937392882,
    0.217916813999513403,
    0.223014117177383564,
    0.228153296828549457,
    0.233333575979407848,
0.238554171442673879,
0.243814293935788129,
0.249113148200245660,
0.254449933121827454,
0.259823841851719028,
0.265234061928493969,
0.270679775400947453,
0.276160158951759327,
0.281674384021967983,
0.287221616936237656,
0.292801019028898213,
0.298411746770740227,
0.304052951896545298,
0.309723781533331410,
0.315423378329296344,
0.321150880583437159,
0.326905422375827870,
0.332686133698534003,
0.338492140587146939,
0.344322565252914603,
0.350176526215451256,
0.356053138436005390,
0.361951513451265472,
0.367870759507683276,
0.373809981696294757,
0.379768282088018327,
0.385744759869409370,
0.391738511478850693,
0.397748630743158804,
0.403774209014584995,
0.409814335308190114,
0.415868096439573343,
0.421934577162933144,
0.428012860309439747,
0.434102026925898610,
0.440201156413683459,
0.446309326667918449,
0.452425614216887317,
0.458549094361650311,
0.464678841315845415,
0.470813928345654997,
0.476953427909915018,
0.483096411800346681,
0.489241951281888854,
0.495389117233109688,
0.501536980286677703,
0.507684610969869055,
0.513831079845091021,
0.519975457650400319,
0.526116815439995000,
0.532254224724658109,
0.538386757612132172,
0.544513486947404646,
0.550633486452880572,
0.556745830868422997,
0.562849596091239857,
0.568943859315595546,
0.575027699172326323,
0.581100195868139213,
0.587160431324671883,
0.593207489317293168,
0.599240455613624601,
    0.605258418111758623,
    0.611260466978157169,
    0.617245694785204857,
    0.623213196648400469,
    0.629162070363162851,
    0.635091416541231801,
    0.641000338746643195,
    0.646887943631258122,
    0.652753341069824855,
    0.658595644294552973,
    0.664413970029181344,
    0.670207438622517193,
    0.675975174181426941,
    0.681716304703259501,
    0.687429962207681378,
    0.693115282867903137,
    0.698771407141277834,
    0.704397479899252943,
    0.709992650556653615,
    0.715556073200279030,
    0.721086906716794096,
    0.726584314919892948,
    0.732047466676720049,
    0.737475536033525003,
    0.742867702340536451,
    0.748223150376032375,
    0.753541070469590402,
    0.758820658624499988,
    0.764061116639314086,
    0.769261652228527604,
    0.774421479142359481,
    0.779539817285623160,
    0.784615892835665907,
    0.789648938359360342,
    0.794638192929130849,
    0.799582902237993443,
    0.804482318713598321,
    0.809335701631251569,
    0.814142317225903689,
    0.818901438803083304,
    0.823612346848764609,
    0.828274329138148024,
    0.832886680843337612,
    0.837448704639903396,
    0.841959710812305140,
    0.846419017358169601,
    0.850825950091398830,
    0.855179842744098417,
    0.859480037067308134,
    0.863725882930519662,
    0.867916738419968303,
    0.872051969935680127,
    0.876130952287265563,
    0.880153068788437798,
    0.884117711350247859,
    0.888024280573020630,
    0.891872185836973586,
    0.895660845391512361,
    0.899389686443182290,
    0.903058145242267907,
    0.906665667168022993,
    0.910211706812522614,
    0.913695728063121049,
    0.917117204183504731,
    0.920475617893327325,
    0.923770461446415503,
    0.927001236707533538,
    0.930167455227694173,
    0.933268638318005328,
    0.936304317122042096,
    0.939274032686730265,
    0.942177336031734702,
    0.945013788217338391,
    0.947782960410804343,
    0.950484433951209517,
    0.953117800412740190,
    0.955682661666440669,
    0.958178629940404480,
    0.960605327878400805,
    0.962962388596924956,
    0.965249455740666562,
    0.967466183536385804,
    0.969612236845188491,
    0.971687291213196302,
    0.973691032920597666,
    0.975623159029079812,
    0.977483377427627587,
    0.979271406876687012,
    0.980986977050685494,
    0.982629828578900022,
    0.984199713084672023,
    0.985696393222956990,
    0.987119642716209222,
    0.988469246388590905,
    0.989745000198503755,
    0.990946711269438341,
    0.992074197919132872,
    0.993127289687042447,
    0.994105827360109551,
    0.995009662996835020,
    0.995838659949644933,
    0.996592692885549747,
    0.997271647805092920,
    0.997875422059586015,
    0.998403924366628281,
    0.998857074823906177,
    0.999234804921274922,
    0.999537057551114994,
    0.999763787016967109,
    0.999914959040439921,
    0.999990550766393427,
    0.999990550766393427,
    0.999914959040439921,
    0.999763787016967109,
    0.999537057551114994,
    0.999234804921274922,
    0.998857074823906177,
    0.998403924366628281,
    0.997875422059586015,
    0.997271647805092920,
    0.996592692885549747,
    0.995838659949644933,
    0.995009662996835020,
    0.994105827360109551,
    0.993127289687042447,
    0.992074197919132872,
    0.990946711269438341,
    0.989745000198503755,
    0.988469246388590905,
    0.987119642716209222,
    0.985696393222956990,
    0.984199713084672023,
    0.982629828578900022,
    0.980986977050685494,
    0.979271406876687012,
    0.977483377427627587,
    0.975623159029079812,
    0.973691032920597666,
    0.971687291213196302,
    0.969612236845188491,
    0.967466183536385804,
    0.965249455740666562,
    0.962962388596924956,
    0.960605327878400805,
    0.958178629940404480,
    0.955682661666440669,
    0.953117800412740190,
    0.950484433951209517,
    0.947782960410804343,
    0.945013788217338391,
    0.942177336031734702,
    0.939274032686730265,
    0.936304317122042096,
    0.933268638318005328,
    0.930167455227694173,
    0.927001236707533538,
    0.923770461446415503,
    0.920475617893327325,
    0.917117204183504731,
    0.913695728063121049,
    0.910211706812522614,
    0.906665667168022993,
    0.903058145242267907,
    0.899389686443182290,
    0.895660845391512361,
    0.891872185836973586,
    0.888024280573020630,
    0.884117711350247859,
    0.880153068788437798,
    0.876130952287265563,
    0.872051969935680127,
    0.867916738419968303,
    0.863725882930519662,
    0.859480037067308134,
    0.855179842744098417,
    0.850825950091398830,
    0.846419017358169601,
    0.841959710812305140,
    0.837448704639903396,
    0.832886680843337612,
    0.828274329138148024,
    0.823612346848764609,
    0.818901438803083304,
    0.814142317225903689,
    0.809335701631251569,
    0.804482318713598321,
    0.799582902237993443,
    0.794638192929130849,
    0.789648938359360342,
    0.784615892835665907,
    0.779539817285623160,
    0.774421479142359481,
    0.769261652228527604,
    0.764061116639314086,
    0.758820658624499988,
    0.753541070469590402,
    0.748223150376032375,
    0.742867702340536451,
    0.737475536033525003,
    0.732047466676720049,
    0.726584314919892948,
    0.721086906716794096,
    0.715556073200279030,
    0.709992650556653615,
    0.704397479899252943,
    0.698771407141277834,
    0.693115282867903137,
    0.687429962207681378,
    0.681716304703259501,
    0.675975174181426941,
    0.670207438622517193,
    0.664413970029181344,
    0.658595644294552973,
    0.652753341069824855,
    0.646887943631258122,
    0.641000338746643195,
    0.635091416541231801,
    0.629162070363162851,
    0.623213196648400469,
    0.617245694785204857,
    0.611260466978157169,
    0.605258418111758623,
    0.599240455613624601,
    0.593207489317293168,
    0.587160431324671883,
    0.581100195868139213,
    0.575027699172326323,
    0.568943859315595546,
    0.562849596091239857,
    0.556745830868422997,
    0.550633486452880572,
    0.544513486947404646,
    0.538386757612132172,
    0.532254224724658109,
    0.526116815439995000,
    0.519975457650400319,
    0.513831079845091021,
    0.507684610969869055,
    0.501536980286677703,
    0.495389117233109688,
    0.489241951281888854,
    0.483096411800346681,
    0.476953427909915018,
    0.470813928345654997,
    0.464678841315845415,
    0.458549094361650311,
    0.452425614216887317,
    0.446309326667918449,
    0.440201156413683459,
    0.434102026925898610,
    0.428012860309439747,
    0.421934577162933144,
    0.415868096439573343,
    0.409814335308190114,
    0.403774209014584995,
    0.397748630743158804,
    0.391738511478850693,
    0.385744759869409370,
    0.379768282088018327,
    0.373809981696294757,
    0.367870759507683276,
    0.361951513451265472,
    0.356053138436005390,
    0.350176526215451256,
    0.344322565252914603,
    0.338492140587146939,
    0.332686133698534003,
    0.326905422375827870,
    0.321150880583437159,
    0.315423378329296344,
    0.309723781533331410,
    0.304052951896545298,
    0.298411746770740227,
    0.292801019028898213,
    0.287221616936237656,
    0.281674384021967983,
    0.276160158951759327,
    0.270679775400947453,
    0.265234061928493969,
    0.259823841851719028,
    0.254449933121827454,
    0.249113148200245660,
    0.243814293935788129,
    0.238554171442673879,
    0.233333575979407848,
    0.228153296828549457,
    0.223014117177383564,
    0.217916813999513403,
    0.212862157937392882,
    0.207850913185815778,
    0.202883837376379883,
    0.197961681462944072,
    0.193085189608093566,
    0.188255099070633203,
    0.183472140094123992,
    0.178737035796480204,
    0.174050502060643708,
    0.169413247426352498,
    0.164825972983019098,
    0.160289372263735408,
    0.155804131140419966,
    0.151370927720123671,
    0.146990432242509073,
    0.142663306978519533,
    0.138390206130252547,
    0.134171775732053911,
    0.130008653552845688,
    0.125901468999704391,
    0.121850843022703492,
    0.117857388021033904,
    0.113921707750417878,
    0.110044397231829294,
    0.106226042661534792,
    0.102467221322468549,
    0.098768501496955430,
    0.095130442380794267,
    0.091553593998715044,
    0.088038497121222858,
    0.084585683182840765,
    0.081195674201764101,
    0.077868982700938449,
    0.074606111630573402,
    0.071407554292103159,
    0.068273794263606080,
    0.065205305326694496,
    0.062202551394885286,
    0.059265986443462593,
    0.056396054440842724,
    0.053593189281452569,
    0.050857814720130512,
    0.048190344308060407,
    0.045591181330248531,
    0.043060718744552196,
    0.040599339122270095,
    0.038207414590302524,
    0.035885306774891212,
    0.033633366746946003,
    0.031451934968968087,
    0.029341341243576458,
    0.027301904663646570,
    0.025333933564067435,
    0.023437725475126125,
    0.021613567077524876,
    0.019861734159038968,
    0.018182491572821535,
    0.016576093197361197,
    0.015042781898099433,
    0.013582789490712122,
    0.012196336706062738,
    0.010883633156830497,
    0.009644877305819977,
    0.008480256435956068,
    0.007389946621969679,
    0.006374112703777302,
    0.005432908261559732,
    0.004566475592542640,
    0.003774945689483389,
    0.003058438220866544,
    0.002417061512811680,
    0.001850912532696092,
    0.001360076874494465,
    0.000944628745838338,
    0.000604630956796859,
    0.000340134910380874,
    0.000151180594771427,
    0.000037796577274096,
    0.000000000000000000
};


int main(){
    //init system clocks and get board speed running at 200 MHz
    InitSysCtrl();
    EALLOW;
    //disable all interrupt
    Interrupt_initModule();
    //init interrupt table
    Interrupt_initVectorTable();
    // initialize the timer structs and setup the timers in their default states
    InitCpuTimers();

    EALLOW;
    initDMAInput();
    EALLOW;
    initDMAOutput();
    EALLOW;
    initFFT();
    /*
    GpioCtrlRegs.GPAMUX1.bit.GPIO0=0;
    GpioCtrlRegs.GPAMUX1.bit.GPIO1=0;
    GpioCtrlRegs.GPAGMUX1.bit.GPIO0=0;
    GpioCtrlRegs.GPAGMUX1.bit.GPIO1=0;
    */
    GpioCtrlRegs.GPADIR.bit.GPIO0=1;
    GpioCtrlRegs.GPADIR.bit.GPIO1=1;

    //switch
    //GpioCtrlRegs.GPAMUX1.bit.GPIO8=0;
    //GpioCtrlRegs.GPAGMUX1.bit.GPIO8=0;
    GpioCtrlRegs.GPAPUD.bit.GPIO8=0;
    //GpioCtrlRegs.GPAMUX1.bit.GPIO9=0;
    //GpioCtrlRegs.GPAGMUX1.bit.GPIO9=0;
    GpioCtrlRegs.GPAPUD.bit.GPIO9=0;

    /*
     * P95  C2  col4    input
     * P139 E1  col3
     * P56  B2  col2
     * P97  D1  col1
     * P94  C2  row1    output
     * P52  B2  row2
     * P41  B1  row3
     * P40  B1  row4
     */
    /*
    GpioCtrlRegs.GPCMUX2.bit.GPIO95=0;
    GpioCtrlRegs.GPEMUX1.bit.GPIO139=0;
    GpioCtrlRegs.GPBMUX2.bit.GPIO56=0;
    GpioCtrlRegs.GPDMUX1.bit.GPIO97=0;
    GpioCtrlRegs.GPCMUX2.bit.GPIO94=0;
    GpioCtrlRegs.GPBMUX2.bit.GPIO52=0;
    GpioCtrlRegs.GPBMUX1.bit.GPIO41=0;
    GpioCtrlRegs.GPBMUX1.bit.GPIO40=0;
    GpioCtrlRegs.GPCGMUX2.bit.GPIO95=0;
    GpioCtrlRegs.GPEGMUX1.bit.GPIO139=0;
    GpioCtrlRegs.GPBGMUX2.bit.GPIO56=0;
    GpioCtrlRegs.GPDGMUX1.bit.GPIO97=0;
    GpioCtrlRegs.GPCGMUX2.bit.GPIO94=0;
    GpioCtrlRegs.GPBGMUX2.bit.GPIO52=0;
    GpioCtrlRegs.GPBGMUX1.bit.GPIO41=0;
    GpioCtrlRegs.GPBGMUX1.bit.GPIO40=0;
    */

    GpioCtrlRegs.GPCDIR.bit.GPIO94=1;
    GpioCtrlRegs.GPBDIR.bit.GPIO52=1;
    GpioCtrlRegs.GPBDIR.bit.GPIO41=1;
    GpioCtrlRegs.GPBDIR.bit.GPIO40=1;

    GpioCtrlRegs.GPCPUD.bit.GPIO95=0;
    GpioCtrlRegs.GPEPUD.bit.GPIO139=0;
    GpioCtrlRegs.GPBPUD.bit.GPIO56=0;
    GpioCtrlRegs.GPDPUD.bit.GPIO97=0;


    //init ADCA, prescaler=4.0, 12 bit, single-ended
    initAdc(ADCA_BASE, ADC_CLK_DIV_4_0,  ADC_RESOLUTION_12BIT,
            ADC_MODE_SINGLE_ENDED);
    /*
     * ADCA
     * SOC0
     * Trigger source=Timer0 Int
     * channel: ACDIN1
     * sampleWindow=50
     * trigger ADCA INT1 when complete
     */
    initAdcSoc(ADCA_BASE, ADC_SOC_NUMBER0, ADC_TRIGGER_CPU1_TINT0,
               ADC_CH_ADCIN1, 50, ADC_INT_NUMBER1);

    //init AIC23
    initMcbspbI2S();
    //init SPIA
    InitSPIA();
    //InitAIC23_SR(SR32);
    InitAIC23();

    initGPIO();
    initTimer0();
    initTimer1();
    initTimer2();

    //point INT to my ISR
    Interrupt_register(INT_DMA_CH1, &dmaCh1ISR);
    Interrupt_register(INT_ADCA1, &adcA1ISR);
    //enable INT
    Interrupt_enable(INT_DMA_CH1);
    Interrupt_enable(INT_ADCA1);
    //enable global interrupt
    Interrupt_enableMaster();

    while(1){
        isDistance=GpioDataRegs.GPADAT.bit.GPIO8;
        if(distanceFlag){
            GpioDataRegs.GPESET.bit.GPIO131=1;
            CPUTimer_startTimer(CPUTIMER2_BASE);
            distanceFlag=0;
        }
        if(keypadFlag){
            key=checkKey();
            if(key==17){
                pitchShiftRatio=1;
                keypad_cont=GpioDataRegs.GPADAT.bit.GPIO9;
                if(!keypad_cont){
                    amp=0;
                }
            }
            else{
                pitchShiftRatio=0.5+0.1*key;
            }
            keypadFlag=0;
        }
        if(received256){
            GpioDataRegs.GPASET.bit.GPIO0=1;
            //format FFT input
            for(int i=0; i<FFT_SIZE; i+=2){
                oneside[i]=hann[(i>>1)]*(float)prevInPtr[i];
                //oneside[i]=(float)prevInPtr[i];
                oneside[i+1]=0;
            }
            for(int i=FFT_SIZE; i<2*FFT_SIZE; i+=2){
                oneside[i]=hann[(i>>1)]*(float)currInPtr[i-512];
                //oneside[i]=(float)currInPtr[i-512];;
                oneside[i+1]=0;
            }

            float step=1/pitchShiftRatio;
            //do FFT
            doFFT(oneside, rightMag, rightPhase);
            float* temp=leftMag;
            leftMag=rightMag;
            rightMag=temp;
            temp=leftPhase;
            leftPhase=rightPhase;
            rightPhase=temp;

            if(firstTime){
                firstTime=0;
                for(int i=0; i<=FFT_SIZE/2; i++){
                    phaseAccum[i]=leftPhase[i];
                }
            }

            //phase vocoder
            while(currInterPos>=0 && currInterPos<1){
                freqInterpolate(rightMag, rightPhase, leftMag, leftPhase, currInterPos,
                                magOut, phaseOut);
                convertToRect(magOut, phaseOut, currIfftOut);
                doIFFT(currIfftOut, rect);

                //hanning
                for(int i=0; i<FFT_SIZE*2; i+=2){
                    currIfftOut[(i>>1)]=currIfftOut[i]*hann[(i>>1)];
                }

                //Uint16 count=0;

                while(currPos<FFT_SIZE/2-1){
                    Uint16 leftPos=(int16)currPos;
                    Uint16 rightPos=leftPos+1;
                    float right=currPos-leftPos;
                    float left=1-right;

                    float currPoint=left*(currIfftOut[leftPos]+prevIfftOut[leftPos
+256])+right*(currIfftOut[rightPos]+prevIfftOut[rightPos+256]);
                    currPoint*=amp;

                    outBufs[outBufCounter>>11][outBufCounter&0x7FF]=(int32)currPoint;
                    outBufCounter=(outBufCounter+1)&OUT_BUF_MASK;
                    outBufs[outBufCounter>>11][outBufCounter&0x7FF]=(int32)currPoint;
                    outBufCounter=(outBufCounter+1)&OUT_BUF_MASK;
                    currPos+=pitchShiftRatio;
                    //count++;
                }
                currPos-=(FFT_SIZE/2-1);
                temp=currIfftOut;
                currIfftOut=prevIfftOut;
                prevIfftOut=temp;

                //outBufCounter=(outBufCounter+count)&OUT_BUF_MASK;

                currInterPos+=step;
            }
            currInterPos--;

            received256=0;
            GpioDataRegs.GPACLEAR.bit.GPIO0=1;
        }
    }
}

__interrupt void dmaCh1ISR(void){
    GpioDataRegs.GPATOGGLE.bit.GPIO1=1;

    volatile int16* src_addr1=&McbspbRegs.DRR2.all;
    volatile int16* dest_addr1=0;

    if(currInBuf==0){
        currInPtr=&inBuf0[0];
        prevInPtr=&inBuf2[0];
        currInBuf=1;
        dest_addr1=(int16*)(&inBuf1[0])+1;
    }
    else if(currInBuf==1){
        currInPtr=&inBuf1[0];
        prevInPtr=&inBuf0[0];
        currInBuf=2;
        dest_addr1=(int16*)(&inBuf2[0])+1;
    }
    else{
        currInPtr=&inBuf2[0];
        prevInPtr=&inBuf1[0];
        currInBuf=0;
        dest_addr1=(int16*)(&inBuf0[0])+1;
    }

    DMACH1AddrConfig(dest_addr1, src_addr1);
    received256=1;
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP7);
    StartDMACH1();

}

__interrupt void ECHO_ISR(void){
    //rising edge
    if(GpioDataRegs.GPEDAT.bit.GPIO130==1){
        CPUTimer_startTimer(CPUTIMER1_BASE);
        timeOut=0;
    }
    //falling edge
    else{
        if(timeOut){
            pitchShiftRatio=1;
            timeOut=0;
        }
        else{
            //stop timer
            CPUTimer_stopTimer(CPUTIMER1_BASE);
            int32 timer1Count=CpuTimer1.RegsAddr->PRD.all - CpuTimer1.RegsAddr->TIM.all;
            if(timer1Count>=1176400){
                pitchShiftRatio=2;
            }
            else{
                pitchShiftRatio=0.5+(timer1Count*1.5/1177000.0);
            }
            timeOut=0;
        }
    }
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP12);
}

__interrupt void Timer0_ISR(void){

    volumeFlag=1;

    if(isDistance){
        distanceFlag=1;
    }
    else{
        keypadFlag=1;
    }

    //ack group 1 interrupt for TIMER0
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP1);
}

__interrupt void Timer1_ISR(void){
    timeOut=1;
    //stop timer
    CPUTimer_stopTimer(CPUTIMER1_BASE);
    //ack group 13 interrupt for TIMER1
    Interrupt_clearACKGroup(INTERRUPT_CPU_INT13);
}

__interrupt void Timer2_ISR(void){
    GpioDataRegs.GPECLEAR.bit.GPIO131=1;
    //stop timer
    CPUTimer_stopTimer(CPUTIMER2_BASE);
    //ack group 14 interrupt for TIMER2
    Interrupt_clearACKGroup(INTERRUPT_CPU_INT14);
}

__interrupt void adcA1ISR(){
    //read ADC result
    Uint16 result=ADC_readResult(ADCARESULT_BASE, ADC_SOC_NUMBER0);
    amp=result*0.011936;
    amp=0.1*amp*amp;
    if(isDistance==0 && key==17){
        keypad_cont=GpioDataRegs.GPADAT.bit.GPIO9;
        if(!keypad_cont){
            amp=0;
        }
    }
    // Clear the interrupt flag and issue ACK
    ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP1);
}

//DMA1=ping-pong input
void initDMAInput(){
    //get src/dest addr
    volatile int16* src_addr=&McbspbRegs.DRR2.all;
    volatile int16* dest_addr=(int16*)(&inBuf0[0])+1;

    //each burst (one 32-bit McbspB sample) has 2 words
    Uint16 burst_size=2;
    int16 src_burst_step=1;     //DRR2->DRR1
    int16 dest_burst_step=-1;   //little endian

    Uint16 trans_size=FFT_SIZE;
    int16 src_trans_step=-1;    //DRR1->DRR2
    int16 dest_trans_step=3;    //move to high word of next data

    //disable addr wrapping
    Uint16 src_wrap_size=0xFFFF;
    int16 src_wrap_step=0;
    Uint16 dest_wrap_size=0xFFFF;
    int16 dest_wrap_step=0;

    //trigger source=McbspB receive
    Uint16 per_sel=74;

    //reset DMA
    DMAInitialize();
    //set the src and dest addr
    DMACH1AddrConfig(dest_addr, src_addr);
    //configure burst
    DMACH1BurstConfig(burst_size-1, src_burst_step, dest_burst_step);
    //configure transfer
    DMACH1TransferConfig(trans_size-1, src_trans_step, dest_trans_step);
    //configure wrap
    DMACH1WrapConfig(src_wrap_size, src_wrap_step, dest_wrap_size, dest_wrap_step);
    //configure mode
    DMACH1ModeConfig(
            per_sel,
            PERINT_ENABLE,
            ONESHOT_DISABLE,
            CONT_DISABLE,
            SYNC_DISABLE,
            SYNC_SRC,
            OVRFLOW_DISABLE,
            SIXTEEN_BIT,
            CHINT_END,
            CHINT_ENABLE
    );
    EALLOW;
    CpuSysRegs.SECMSEL.bit.PF2SEL = 1;
}

//DMA2=output
void initDMAOutput(){
    //get src/dest addr
    volatile int16* src_addr=(int16*)(&(outBufs[0][0]))+1;
    volatile int16* dest_addr=&McbspbRegs.DXR2.all;

    //each burst (one 32-bit McbspB sample) has 2 words
    Uint16 burst_size=2;
    int16 src_burst_step=-1;     //DRR2->DRR1
    int16 dest_burst_step=1;   //little endian

    Uint16 trans_size=2048*8;
    int16 src_trans_step=3;    //DRR1->DRR2
    int16 dest_trans_step=-1;    //move to high word of next data

    //disable addr wrapping
    Uint16 src_wrap_size=0xFFFF;
    int16 src_wrap_step=0;
    Uint16 dest_wrap_size=0xFFFF;
    int16 dest_wrap_step=0;

    //trigger source=McbspB receive
    Uint16 per_sel=74;

    //reset DMA
    //set the src and dest addr
    DMACH2AddrConfig(dest_addr, src_addr);
    //configure burst
    DMACH2BurstConfig(burst_size-1, src_burst_step, dest_burst_step);
    //configure transfer
    DMACH2TransferConfig(trans_size-1, src_trans_step, dest_trans_step);
    //configure wrap
    DMACH2WrapConfig(src_wrap_size, src_wrap_step, dest_wrap_size, dest_wrap_step);
    //configure mode
    DMACH2ModeConfig(
            per_sel,
            PERINT_ENABLE,
            ONESHOT_DISABLE,
            CONT_ENABLE,
            SYNC_DISABLE,
            SYNC_SRC,
            OVRFLOW_DISABLE,
            SIXTEEN_BIT,
            CHINT_END,
            CHINT_DISABLE
    );
    //start DMA
    StartDMACH1();
    StartDMACH2();
}

void initFFT(){
    CFFT_f32_setOutputPtr(hnd_cfft, test_output);
    CFFT_f32_setStages(hnd_cfft, FFT_STAGES);
    CFFT_f32_setFFTSize(hnd_cfft, FFT_SIZE);
    CFFT_f32_setTwiddlesPtr(hnd_cfft, twiddleFactors);
    CFFT_f32_sincostable(hnd_cfft);
}

void initGPIO(){
    EALLOW;

    //P130->echo (input)
    //P131->trig (output)
    /*
    GpioCtrlRegs.GPEMUX1.bit.GPIO130=0;
    GpioCtrlRegs.GPEMUX1.bit.GPIO131=0;
    GpioCtrlRegs.GPEGMUX1.bit.GPIO130=0;
    GpioCtrlRegs.GPEGMUX1.bit.GPIO131=0;
```

```
     */
    GpioCtrlRegs.GPEPUD.bit.GPIO130=0;

    GpioCtrlRegs.GPEDIR.bit.GPIO130=0;
    GpioCtrlRegs.GPEDIR.bit.GPIO131=1;

    //P130 need both edge interrupt trigger
    //P130=INPUT6=XINT3
    InputXbarRegs.INPUT6SELECT=130;
    XintRegs.XINT3CR.bit.POLARITY=3;
    XintRegs.XINT3CR.bit.ENABLE=1;
    //point int to ISR
    Interrupt_register(INT_XINT3, &ECHO_ISR);
    //enable int in PIE and IER
    Interrupt_enable(INT_XINT3);

}

void initTimer0(){
    //timer0, cpu freq=200MHz, timer period=100ms
    ConfigCpuTimer(&CpuTimer0, 200, 100000);
    Interrupt_register(INT_TIMER0, &Timer0_ISR);
    Interrupt_enable(INT_TIMER0);
    CPUTimer_startTimer(CPUTIMER0_BASE);
}

void initTimer1(){
    //timer1, cpu freq=200MHz, timer period=10ms
    ConfigCpuTimer(&CpuTimer1, 200, 10000);
    Interrupt_register(INT_TIMER1, &Timer1_ISR);
    Interrupt_enable(INT_TIMER1);
}

void initTimer2(){
    //timer2, cpu freq=200MHz, timer period=10us
    ConfigCpuTimer(&CpuTimer2, 200, 50);
    Interrupt_register(INT_TIMER2, &Timer2_ISR);
    Interrupt_enable(INT_TIMER2);
}

void doFFT(float* inBuf, float* outMagnitude, float* outPhase){
    CFFT_f32_setInputPtr(hnd_cfft, inBuf);
    CFFT_f32_setOutputPtr(hnd_cfft, test_output);
    CFFT_f32(hnd_cfft);
    float* p_temp=CFFT_f32_getCurrInputPtr(hnd_cfft);
    //number of stage is odd, output in currInputPtr
    //doIFFT(p_temp, test_output);
    CFFT_f32_setCurrInputPtr(hnd_cfft, p_temp);
    CFFT_f32_setCurrOutputPtr(hnd_cfft, outMagnitude);
    CFFT_f32_mag(hnd_cfft);
    CFFT_f32_setCurrInputPtr(hnd_cfft, p_temp);
    CFFT_f32_setCurrOutputPtr(hnd_cfft, outPhase);
    CFFT_f32_phase(hnd_cfft);
}

void freqInterpolate(float* mag1, float* phase1, float* mag2, float* phase2, float right,
                     float* magOut, float* phaseOut){
    float left=1-right;
    for(Uint16 i=0; i<=FFT_SIZE/2; i++){
        magOut[i]=left*mag1[i]+right*mag2[i];
        float dp=phase2[i]-phase1[i]-dph[i];
        dp=dp-2*PI*(roundf(dp*ONE_OVER_TWO_PI));
        phaseOut[i]=phaseAccum[i]-2*PI*(roundf(phaseAccum[i]*ONE_OVER_TWO_PI));
        phaseAccum[i]=phaseAccum[i]+dph[i]+dp;
        if(phaseAccum[i]>1000000){
            firstTime=1;
        }
    }
}

void convertToRect(float* mag, float* phase, float* rect){
    for(Uint16 i=0; i<=FFT_SIZE; i+=2){
        rect[i]=mag[i/2]*myCos(phase[i/2]);
        rect[i+1]=mag[i/2]*mySin(phase[i/2]);
    }

    Uint16 totalLen=FFT_SIZE*2;
    for(Uint16 i=FFT_SIZE+2; i<totalLen; i+=2){
        rect[i]=rect[totalLen-i];
        rect[i+1]=-rect[totalLen+1-i];
    }

}


void doIFFT(float* inBuf, float* outBuf){
    CFFT_f32_setInputPtr(hnd_cfft, inBuf);
    CFFT_f32_setOutputPtr(hnd_cfft, outBuf);
    ICFFT_f32(hnd_cfft);
}

/*
 * P95  C2  col4    input
 * P139 E1  col3
 * P56  B2  col2
 * P97  D1  col1
 *
 * P94  C2  row1    output
 * P52  B2  row2
 * P41  B1  row3
 * P40  B1  row4
 */

volatile Uint16 cols[4];

void getCols(){
    cols[3]=GpioDataRegs.GPCDAT.bit.GPIO95;
    cols[2]=GpioDataRegs.GPEDAT.bit.GPIO139;
    cols[1]=GpioDataRegs.GPBDAT.bit.GPIO56;
    cols[0]=GpioDataRegs.GPDDAT.bit.GPIO97;
}

void clearRows(){
    GpioDataRegs.GPCCLEAR.bit.GPIO94=1;
    GpioDataRegs.GPBCLEAR.bit.GPIO52=1;
    GpioDataRegs.GPBCLEAR.bit.GPIO41=1;
    GpioDataRegs.GPBCLEAR.bit.GPIO40=1;
}

Uint16 prev=0;
Uint16 checkKey(){
    clearRows();
    getCols();
    for(int i=0; i<4; i++){
        if(cols[i]==0){
            GpioDataRegs.GPCSET.bit.GPIO94=1;
            DELAY_US(1);
            getCols();
            if(cols[i]==1){
                prev=i;
                return i;
            }
            clearRows();
            GpioDataRegs.GPBSET.bit.GPIO52=1;
            DELAY_US(1);
            getCols();
            if(cols[i]==1){
                prev=4+i;
                return 4+i;
            }
            clearRows();
            GpioDataRegs.GPBSET.bit.GPIO41=1;
            DELAY_US(1);
            getCols();
            if(cols[i]==1){
                prev=8+i;
                return 8+i;
            }
            clearRows();
            GpioDataRegs.GPBSET.bit.GPIO40=1;
            DELAY_US(1);
            getCols();
            if(cols[i]==1){
                prev=12+i;
                return 12+i;
            }
            return prev;
        }

    }
    return 17;
}

float mySin(float rad){
   if(rad<0){
       rad=-rad;
       Uint16 index=(Uint16)roundf(rad*TRIG_RATIO);
       if(index>=256){
           index-=256;
       }
       return -twiddleFactors[index];
   }
   Uint16 index=(Uint16)roundf(rad*TRIG_RATIO);
   if(index>=256){
      index-=256;
   }
   return twiddleFactors[index];
}

float myCos(float rad){
    if(rad<0){
        rad=-rad;
    }
    Uint16 index=(Uint16)roundf(rad*TRIG_RATIO)+128;
    if(index>=384){
       index-=256;
    }
    return twiddleFactors[index];
}

```