EEL4511 Real-time DSP Applications Lab 9 Final Project

Title: Real-Time Pitch Shifter

First Name: Shida Last Name: Yang

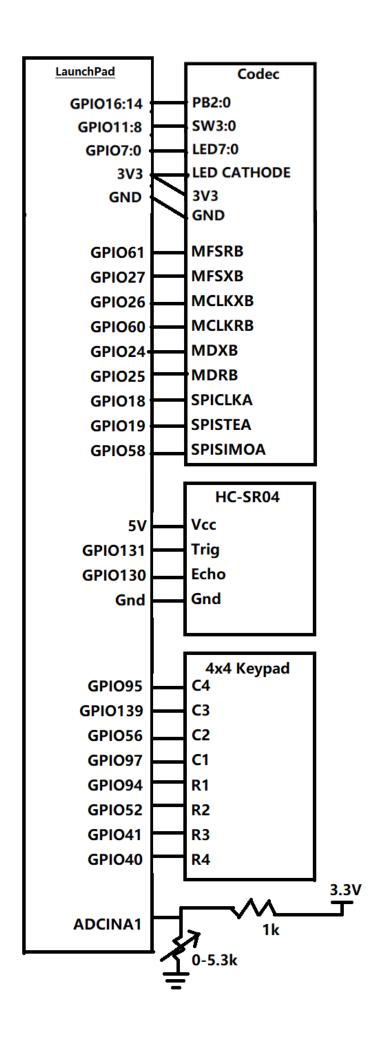
Abstract:

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:	



```
1#include <F28x_Project.h>
3 #include "SPIA.h"
4 #include "AIC23.h"
 5 #include "Mcbsp.h"
 6 #include "interrupt.h"
 7 #include "math.h"
8 #include "cputimer.h"
9 #include "fpu_cfft.h"
10 #include "math.h"
11 #include "myAdc.h"
12
13 #define FFT SIZE 512
14 #define FFT_STAGES 9
15 #define HOP_SIZE 256
16 #define TRIG_RATIO 81.48733086
17 #define PI 3.1415926
18 #define ONE_OVER_TWO_PI 0.15915494309
19 #define OUT_BUF_MASK 0x3FFF
20 #define TIMER1_DISTANCE_RES 73529 //resolution for 0.1x, 1m=2x, 0m=0.5x
22 __interrupt void dmaCh1ISR(void);
23 __interrupt void ECHO_ISR(void);
24 __interrupt void Timer0_ISR(void);
25 __interrupt void Timer1_ISR(void);
26 __interrupt void Timer2_ISR(void);
27
   28
29 void initDMAInput();
30 void initDMAOutput();
31 void initFFT();
32 void initGPIO();
33 void initTimer0();
34 void initTimer1();
35 void initTimer2();
37 void doHanning(float* inBuf);
38 void doffT(float* inBuf, float* outMagnitude, float* outPhase);
39 void freqInterpolate(float* mag1, float* phase1, float* mag2, float* phase2, float right,
                        float* magOut, float* phaseOut);
41 void convertToRect(float* mag, float* phase, float* rect);
42 void doIFFT(float* inBuf, float* outBuf);
43 Uint16 timeInterpolate(float* values1, float* values2, Uint16 len, float ratio, float*
  output);
44 Uint16 checkKey();
46 float mySin(float rad);
47 float myCos(float rad);
49 volatile float pitchShiftRatio=1.3;
50 volatile float amp=1;
51
52 // CFFT F32 STRUCT object
53 CFFT F32 STRUCT cfft;
54 // Handle to the CFFT F32 STRUCT object
55 CFFT_F32_STRUCT_Handle hnd_cfft = &cfft;
57 volatile Uint16 currInBuf=0;
58 volatile float currInterPos=0;
59 volatile Uint16 outBufCounter=0;
60 volatile Uint16 insideOutBufCounter=0;
61 volatile float currPos=0;
```

```
62 volatile bool firstTime=1;
 63 volatile int32 startTime=0;
 64 volatile int32 endTime=0;
 65 volatile bool timeOut=0;
 66 volatile Uint16 key=17;
 68 volatile bool received256=0;
 69 volatile bool volumeFlag=0;
70 volatile bool distanceFlag=0;
 71 volatile bool isDistance=1;
 72 volatile bool keypad_cont=0;
 73 volatile bool keypadFlag=0;
 75 #pragma DATA_SECTION(oneside, "FFT_INBUF");
 76 volatile float oneside[FFT_SIZE << 1];
 77
 78 #pragma DATA_SECTION(inBuf0, "SOUND_INBUF");
 79 #pragma DATA_SECTION(inBuf1, "SOUND_INBUF");
80 #pragma DATA_SECTION(inBuf2, "SOUND_INBUF");
 81 volatile int32 inBuf0[FFT SIZE];
 82 volatile int32 inBuf1[FFT_SIZE];
 83 volatile int32 inBuf2[FFT_SIZE];
 84 volatile int32* currInPtr=&inBuf0[0];
 85 volatile int32* prevInPtr=&inBuf2[0];
 87 #pragma DATA_SECTION(magA, "MAG_BUF");
88 #pragma DATA_SECTION(magB, "MAG_BUF");
 89 #pragma DATA_SECTION(magOut, "MAG_PHASE_OUT_BUF");
90 volatile float magA[FFT_SIZE];
91 volatile float magB[FFT_SIZE];
92 volatile float magOut[FFT_SIZE];
93 volatile float* leftMag=&magA[0];
94 volatile float* rightMag=&magB[0];
 96 #pragma DATA_SECTION(phaseA, "PHASE_BUF");
97 #pragma DATA_SECTION(phaseB, "PHASE_BUF");
98 #pragma DATA_SECTION(phaseOut, "MAG_PHASE_OUT_BUF");
99 volatile float phaseA[FFT SIZE];
100 volatile float phaseB[FFT_SIZE];
101 volatile float phaseOut[FFT_SIZE];
102 volatile float* leftPhase=&phaseA[0];
103 volatile float* rightPhase=&phaseB[0];
104
105 #pragma DATA_SECTION(phaseAccum, "RECT_BUF");
106 volatile float phaseAccum[FFT_SIZE];
108 #pragma DATA SECTION(rect, "RECT BUF");
109 volatile float rect[FFT_SIZE*2];
111 #pragma DATA_SECTION(ifftOutA, "IFFT_OUT_BUF");
112 #pragma DATA_SECTION(ifftOutB, "IFFT_OUT_BUF");
113 volatile float ifftOutA[FFT_SIZE << 1];</pre>
114 volatile float ifftOutB[FFT SIZE << 1];
115 volatile float* currIfftOut=&ifftOutA[0];
116 volatile float* prevIfftOut=&ifftOutB[0];
117
118 #pragma DATA_SECTION(outBuf0, "OUT_BUF0");
119 volatile int32 outBuf0[2048];
120 #pragma DATA_SECTION(outBuf1, "OUT_BUF1");
121 volatile int32 outBuf1[2048];
122 #pragma DATA_SECTION(outBuf2, "OUT_BUF2");
123 volatile int32 outBuf2[2048];
```

```
"OUT_BUF3");
124 #pragma DATA_SECTION(outBuf3,
125 volatile int32 outBuf3[2048];
                                   "OUT_BUF4");
126 #pragma DATA_SECTION(outBuf4,
127 volatile int32 outBuf4[2048];
                                  "OUT_BUF5");
128 #pragma DATA_SECTION(outBuf5,
129 volatile int32 outBuf5[2048];
                                  "OUT_BUF6");
130 #pragma DATA SECTION(outBuf6,
131 volatile int32 outBuf6[2048];
132 #pragma DATA_SECTION(outBuf7, "OUT_BUF7");
133 volatile int32 outBuf7[2048];
134
135 volatile int32* outBufs[8]={
136
       &outBuf0[0],
       &outBuf1[0],
137
       &outBuf2[0],
138
139
       &outBuf3[0],
140
       &outBuf4[0],
141
       &outBuf5[0],
142
       &outBuf6[0],
143
       &outBuf7[0]
144 };
145
146 #pragma DATA_SECTION(twiddleFactors, "TWIDDLE_TEST_BUF");
147 #pragma DATA SECTION(test output, "TWIDDLE TEST BUF");
148 volatile float test_output[FFT_SIZE << 1];
150 volatile float twiddleFactors[FFT_SIZE << 1];
151
152 float dph[FFT_SIZE/2+1]={
       153
154
       3.141592653589793116,
155
       6.283185307179586232,
156
       9.424777960769379348,
       12.566370614359172464,
157
158
       15.707963267948965580,
159
       18.849555921538758696,
       21.991148575128551812,
160
161
       25.132741228718344928,
162
       28.274333882308138044,
163
        31.415926535897931160,
164
        34.557519189487720723,
165
       37.699111843077517392,
166
       40.840704496667306955,
167
       43.982297150257103624,
168
       47.123889803846900293,
169
       50.265482457436689856,
170
       53.407075111026486525,
171
       56.548667764616276088,
172
       59.690260418206072757,
173
       62.831853071795862320,
174
       65.973445725385658989,
175
       69.115038378975441447,
176
       72.256631032565238115,
177
       75.398223686155034784,
178
       78.539816339744831453,
179
       81.681408993334613911,
180
       84.823001646924424790,
181
       87.964594300514207248,
182
       91.106186954104003917,
       94.247779607693800585,
183
184
       97.389372261283583043,
185
       100.530964914873379712,
```

```
186
        103.672557568463176381,
187
        106.814150222052973049,
188
        109.955742875642755507,
189
        113.097335529232552176,
190
        116.238928182822334634,
        119.380520836412145513,
191
192
        122.522113490001942182,
193
        125.663706143591724640,
        128.805298797181507098,
194
195
        131.946891450771317977,
196
        135.088484104361100435,
197
        138.230076757950882893,
198
        141.371669411540693773,
199
        144.513262065130476230,
200
        147.654854718720287110,
201
        150.796447372310069568,
202
        153.938040025899880447,
203
        157.079632679489662905,
204
        160.221225333079445363,
        163.362817986669227821,
205
206
        166.504410640259038701,
207
        169.646003293848849580,
208
        172.787595947438632038,
209
        175.929188601028414496,
210
        179.070781254618225375,
211
        182.212373908208007833,
212
        185.353966561797790291,
213
        188.495559215387601171,
214
        191.637151868977383629,
215
        194.778744522567166086,
        197.920337176156976966,
216
217
        201.061929829746759424,
218
        204.203522483336541882,
219
        207.345115136926352761,
220
        210.486707790516135219,
221
        213.628300444105946099,
222
        216.769893097695728557,
223
        219.911485751285511014,
224
        223.053078404875293472,
225
        226.194671058465104352,
226
        229.336263712054915231,
227
        232.477856365644669268,
228
        235.619449019234480147,
229
        238.761041672824291027,
230
        241.902634326414045063,
231
        245.044226980003884364,
232
        248.185819633593666822,
233
        251.327412287183449280,
234
        254.469004940773260159,
235
        257.610597594363014196,
236
        260.752190247952796653,
237
        263.893782901542635955,
238
        267.035375555132418413,
239
        270.176968208722200870,
240
        273.318560862311983328,
241
        276.460153515901765786,
242
        279.601746169491605087,
243
        282.743338823081387545,
244
        285.884931476671170003,
245
        289.026524130260952461,
246
        292.168116783850734919,
247
        295.309709437440574220,
```

```
248
        298.451302091030356678,
249
        301.592894744620139136,
250
        304.734487398209921594,
251
        307.876080051799760895,
252
        311.017672705389486509,
253
        314.159265358979325811,
254
        317.300858012569108269,
255
        320.442450666158890726,
256
        323.584043319748730028,
257
        326.725635973338455642,
258
        329.867228626928238100,
259
        333.008821280518077401,
260
        336.150413934107916702,
261
        339.292006587697699160,
262
        342.433599241287424775,
        345.575191894877264076,
263
264
        348.716784548467046534,
265
        351.858377202056828992,
266
        354.999969855646611450,
        358.141562509236450751,
267
268
        361.283155162826176365,
        364.424747816416015667,
269
        367.566340470005741281,
270
271
        370.707933123595580582,
272
        373.849525777185419884,
273
        376.991118430775202341,
274
        380.132711084364984799,
275
        383.274303737954767257,
276
        386.415896391544492872,
277
        389.557489045134332173,
278
        392.699081698724114631,
279
        395.840674352313953932,
280
        398.982267005903736390,
281
        402.123859659493518848,
282
        405.265452313083301306,
283
        408.407044966673083763,
284
        411.548637620262923065,
285
        414.690230273852705523,
286
        417.831822927442487980,
287
        420.973415581032270438,
288
        424.115008234622052896,
289
        427.256600888211892197,
290
        430.398193541801674655,
291
        433.539786195391457113,
292
        436.681378848981239571,
293
        439.822971502571022029,
294
        442.964564156160861330,
295
        446.106156809750586945,
        449.247749463340426246,
296
297
        452.389342116930208704,
298
        455.530934770519991162,
299
        458.672527424109830463,
300
        461.814120077699612921,
301
        464.955712731289338535,
302
        468.097305384879177836,
303
        471.238898038468960294,
        474.380490692058742752,
304
305
        477.522083345648582053,
306
        480.663675999238307668,
        483.805268652828090126,
307
308
        486.946861306417929427,
309
        490.088453960007768728,
```

```
310
       493.230046613597494343,
311
        496.371639267187333644,
        499.513231920777059258,
312
313
        502.654824574366898560,
314
        505.796417227956737861,
        508.938009881546520319,
315
316
        512.079602535136245933,
317
        515.221195188726028391,
318
        518.362787842315810849,
319
        521.504380495905593307,
320
        524.645973149495375765,
321
        527.787565803085271909,
322
        530.929158456675054367,
323
        534.070751110264836825,
324
        537.212343763854619283,
325
        540.353936417444401741,
326
        543.495529071034184199,
327
        546.637121724623966657,
328
        549.778714378213749114,
        552.920307031803531572,
329
330
        556.061899685393427717,
331
        559.203492338983210175,
332
        562.345084992572992633,
333
        565.486677646162775090,
334
        568.628270299752557548,
335
        571.769862953342340006,
336
        574.911455606932122464,
337
        578.053048260521904922,
        581.194640914111687380,
338
        584.336233567701469838,
339
        587.477826221291365982,
340
341
        590.619418874881148440,
342
        593.761011528470930898,
343
        596.902604182060713356,
344
        600.044196835650495814,
345
        603.185789489240278272,
346
        606.327382142830060729,
347
        609.468974796419843187,
348
        612.610567450009625645,
349
        615.752160103599521790,
350
        618.893752757189304248,
351
        622.035345410778973019,
352
        625.176938064368869163,
353
        628.318530717958651621,
354
        631.460123371548434079,
355
        634.601716025138216537,
        637.743308678727998995,
356
357
        640.884901332317781453,
358
        644.026493985907563911,
359
        647.168086639497460055,
        650.309679293087242513,
360
361
        653.451271946676911284,
362
        656.592864600266807429,
        659.734457253856476200,
363
364
        662.876049907446258658,
365
        666.017642561036154802,
366
        669.159235214625937260,
367
        672.300827868215833405,
368
        675.442420521805502176,
        678.584013175395398321,
369
370
        681.725605828985067092,
371
        684.867198482574849550,
```

```
372
       688.008791136164745694,
373
       691.150383789754528152,
374
       694.291976443344196923,
375
       697.433569096934093068,
376
       700.575161750523875526,
377
       703.716754404113657984,
378
       706.858347057703440441,
379
       709.999939711293222899,
380
       713.141532364883005357,
381
       716.283125018472901502,
382
       719.424717672062683960,
383
       722.566310325652352731,
384
       725.707902979242248875,
385
       728.849495632832031333,
386
       731.991088286421813791,
387
       735.132680940011482562,
388
       738.274273593601378707,
389
       741.415866247191161165,
390
       744.557458900781057309,
391
       747.699051554370839767,
392
       750.840644207960622225,
393
       753.982236861550404683,
394
       757.123829515140187141,
395
       760.265422168729969599,
396
       763.407014822319638370,
397
       766.548607475909534514,
398
       769.690200129499203285,
399
       772.831792783088985743,
400
       775.973385436678881888,
401
       779.114978090268664346,
402
       782.256570743858560490,
403
       785.398163397448229261,
404
       788.539756051038125406,
       791.681348704627907864,
405
406
       794.822941358217690322,
407
       797.964534011807472780,
408
       801.106126665397255238,
409
       804.247719318987037695
410 };
411
412 const float hann[FFT SIZE] = {
413
       414
       0.000037796577274096,
415
       0.000151180594771427,
416
       0.000340134910380874,
417
       0.000604630956796859,
       0.000944628745838338,
418
419
       0.001360076874494465,
420
       0.001850912532696092,
421
       0.002417061512811680,
422
       0.003058438220866544,
423
       0.003774945689483389,
424
       0.004566475592542640,
425
       0.005432908261559732,
426
       0.006374112703777302,
427
       0.007389946621969679,
428
       0.008480256435956068,
429
       0.009644877305819977,
430
       0.010883633156830497,
431
       0.012196336706062738,
432
       0.013582789490712122,
433
       0.015042781898099433,
```

```
434
       0.016576093197361197,
435
       0.018182491572821535,
436
        0.019861734159038968,
437
        0.021613567077524876,
438
        0.023437725475126125,
439
        0.025333933564067435,
440
        0.027301904663646570,
441
        0.029341341243576458,
442
        0.031451934968968087,
443
        0.033633366746946003,
444
        0.035885306774891212,
445
        0.038207414590302524,
446
        0.040599339122270095,
447
        0.043060718744552196,
448
        0.045591181330248531,
449
        0.048190344308060407,
450
        0.050857814720130512,
451
        0.053593189281452569,
452
        0.056396054440842724,
453
        0.059265986443462593,
454
        0.062202551394885286,
455
        0.065205305326694496,
456
        0.068273794263606080,
457
        0.071407554292103159,
458
        0.074606111630573402,
459
        0.077868982700938449,
460
        0.081195674201764101,
461
        0.084585683182840765
462
        0.088038497121222858,
463
        0.091553593998715044,
464
        0.095130442380794267,
465
        0.098768501496955430,
466
        0.102467221322468549,
467
        0.106226042661534792,
468
        0.110044397231829294,
469
        0.113921707750417878,
470
        0.117857388021033904,
471
        0.121850843022703492,
472
        0.125901468999704391,
473
        0.130008653552845688,
474
        0.134171775732053911,
475
        0.138390206130252547,
476
        0.142663306978519533,
477
        0.146990432242509073,
478
        0.151370927720123671,
479
        0.155804131140419966,
480
        0.160289372263735408,
481
        0.164825972983019098,
482
        0.169413247426352498,
        0.174050502060643708,
483
484
        0.178737035796480204,
485
        0.183472140094123992,
486
        0.188255099070633203,
487
        0.193085189608093566,
488
        0.197961681462944072,
489
        0.202883837376379883,
490
        0.207850913185815778,
491
        0.212862157937392882,
492
        0.217916813999513403,
493
        0.223014117177383564,
494
        0.228153296828549457,
495
        0.233333575979407848,
```

496 0.238554171442673879, 497 0.243814293935788129, 498 0.249113148200245660, 499 0.254449933121827454, 500 0.259823841851719028, 0.265234061928493969, 501 0.270679775400947453, 502 503 0.276160158951759327, 504 0.281674384021967983, 505 0.287221616936237656, 506 0.292801019028898213, 507 0.298411746770740227, 508 0.304052951896545298, 509 0.309723781533331410, 510 0.315423378329296344, 511 0.321150880583437159, 512 0.326905422375827870, 513 0.332686133698534003, 514 0.338492140587146939, 0.344322565252914603, 515 516 0.350176526215451256, 517 0.356053138436005390, 518 0.361951513451265472, 519 0.367870759507683276, 520 0.373809981696294757, 521 0.379768282088018327, 522 0.385744759869409370, 523 0.391738511478850693, 524 0.397748630743158804, 0.403774209014584995, 525 0.409814335308190114, 526 527 0.415868096439573343, 528 0.421934577162933144, 529 0.428012860309439747, 530 0.434102026925898610, 0.440201156413683459, 531 532 0.446309326667918449, 533 0.452425614216887317, 534 0.458549094361650311, 535 0.464678841315845415, 536 0.470813928345654997, 537 0.476953427909915018, 538 0.483096411800346681, 539 0.489241951281888854, 540 0.495389117233109688, 541 0.501536980286677703, 542 0.507684610969869055, 543 0.513831079845091021, 544 0.519975457650400319, 545 0.526116815439995000, 546 0.532254224724658109, 547 0.538386757612132172, 548 0.544513486947404646, 549 0.550633486452880572, 550 0.556745830868422997, 551 0.562849596091239857, 552 0.568943859315595546, 553 0.575027699172326323, 554 0.581100195868139213, 555 0.587160431324671883, 0.593207489317293168, 556 557 0.599240455613624601,

558 0.605258418111758623, 559 0.611260466978157169, 0.617245694785204857, 560 561 0.623213196648400469, 562 0.629162070363162851, 0.635091416541231801, 563 564 0.641000338746643195, 565 0.646887943631258122, 566 0.652753341069824855, 567 0.658595644294552973, 568 0.664413970029181344, 569 0.670207438622517193, 570 0.675975174181426941, 571 0.681716304703259501, 572 0.687429962207681378, 573 0.693115282867903137, 574 0.698771407141277834, 575 0.704397479899252943, 576 0.709992650556653615, 0.715556073200279030, 577 578 0.721086906716794096, 579 0.726584314919892948, 0.732047466676720049, 580 0.737475536033525003, 581 582 0.742867702340536451, 583 0.748223150376032375, 584 0.753541070469590402, 585 0.758820658624499988, 586 0.764061116639314086, 0.769261652228527604, 587 588 0.774421479142359481, 589 0.779539817285623160, 590 0.784615892835665907, 0.789648938359360342, 591 592 0.794638192929130849, 593 0.799582902237993443, 594 0.804482318713598321, 595 0.809335701631251569, 596 0.814142317225903689, 597 0.818901438803083304, 598 0.823612346848764609, 599 0.828274329138148024, 600 0.832886680843337612, 601 0.837448704639903396, 602 0.841959710812305140, 603 0.846419017358169601, 0.850825950091398830, 604 605 0.855179842744098417, 606 0.859480037067308134, 607 0.863725882930519662, 608 0.867916738419968303, 609 0.872051969935680127, 610 0.876130952287265563, 0.880153068788437798, 611 612 0.884117711350247859, 613 0.888024280573020630, 614 0.891872185836973586, 615 0.895660845391512361, 616 0.899389686443182290, 0.903058145242267907, 617 618 0.906665667168022993, 619 0.910211706812522614,

620 0.913695728063121049, 621 0.917117204183504731, 0.920475617893327325, 622 623 0.923770461446415503, 624 0.927001236707533538, 0.930167455227694173, 625 626 0.933268638318005328, 627 0.936304317122042096, 628 0.939274032686730265, 629 0.942177336031734702, 630 0.945013788217338391, 631 0.947782960410804343, 632 0.950484433951209517, 633 0.953117800412740190, 634 0.955682661666440669, 635 0.958178629940404480, 636 0.960605327878400805, 637 0.962962388596924956, 638 0.965249455740666562, 0.967466183536385804, 639 640 0.969612236845188491, 0.971687291213196302, 641 642 0.973691032920597666, 643 0.975623159029079812, 644 0.977483377427627587, 645 0.979271406876687012, 646 0.980986977050685494, 647 0.982629828578900022, 648 0.984199713084672023, 0.985696393222956990, 649 0.987119642716209222, 650 651 0.988469246388590905, 652 0.989745000198503755, 0.990946711269438341, 653 654 0.992074197919132872, 655 0.993127289687042447, 656 0.994105827360109551, 0.995009662996835020, 657 658 0.995838659949644933, 659 0.996592692885549747, 660 0.997271647805092920, 661 0.997875422059586015, 662 0.998403924366628281, 663 0.998857074823906177, 664 0.999234804921274922, 665 0.999537057551114994, 0.999763787016967109, 666 667 0.999914959040439921, 668 0.999990550766393427, 0.999990550766393427, 669 670 0.999914959040439921, 671 0.999763787016967109, 672 0.999537057551114994, 0.999234804921274922, 673 0.998857074823906177, 674 675 0.998403924366628281, 0.997875422059586015, 676 0.997271647805092920, 677 678 0.996592692885549747, 679 0.995838659949644933, 680 0.995009662996835020, 681 0.994105827360109551,

0.993127289687042447, 682 683 0.992074197919132872, 684 0.990946711269438341, 0.989745000198503755, 685 686 0.988469246388590905, 0.987119642716209222, 687 688 0.985696393222956990, 689 0.984199713084672023, 0.982629828578900022, 690 691 0.980986977050685494, 692 0.979271406876687012, 693 0.977483377427627587, 694 0.975623159029079812, 695 0.973691032920597666, 696 0.971687291213196302, 697 0.969612236845188491, 698 0.967466183536385804, 699 0.965249455740666562, 700 0.962962388596924956, 701 0.960605327878400805, 702 0.958178629940404480, 703 0.955682661666440669, 704 0.953117800412740190, 705 0.950484433951209517, 706 0.947782960410804343, 707 0.945013788217338391, 708 0.942177336031734702, 709 0.939274032686730265, 710 0.936304317122042096, 0.933268638318005328, 711 712 0.930167455227694173, 713 0.927001236707533538, 714 0.923770461446415503, 0.920475617893327325, 715 716 0.917117204183504731, 717 0.913695728063121049, 718 0.910211706812522614, 719 0.906665667168022993, 720 0.903058145242267907, 721 0.899389686443182290, 722 0.895660845391512361, 723 0.891872185836973586, 724 0.888024280573020630, 725 0.884117711350247859, 726 0.880153068788437798, 727 0.876130952287265563, 728 0.872051969935680127, 729 0.867916738419968303, 730 0.863725882930519662, 731 0.859480037067308134, 732 0.855179842744098417, 733 0.850825950091398830, 734 0.846419017358169601, 735 0.841959710812305140, 0.837448704639903396, 736 737 0.832886680843337612, 738 0.828274329138148024, 739 0.823612346848764609, 740 0.818901438803083304, 741 0.814142317225903689, 742 0.809335701631251569, 743 0.804482318713598321,

744 0.799582902237993443, 745 0.794638192929130849, 746 0.789648938359360342, 747 0.784615892835665907, 748 0.779539817285623160, 749 0.774421479142359481, 750 0.769261652228527604, 751 0.764061116639314086. 752 0.758820658624499988, 753 0.753541070469590402, 754 0.748223150376032375, 755 0.742867702340536451, 756 0.737475536033525003, 757 0.732047466676720049, 758 0.726584314919892948, 759 0.721086906716794096, 760 0.715556073200279030, 761 0.709992650556653615, 762 0.704397479899252943, 763 0.698771407141277834, 0.693115282867903137, 764 765 0.687429962207681378, 766 0.681716304703259501, 767 0.675975174181426941, 768 0.670207438622517193, 769 0.664413970029181344, 770 0.658595644294552973, 771 0.652753341069824855, 0.646887943631258122, 772 0.641000338746643195, 773 774 0.635091416541231801, 775 0.629162070363162851, 776 0.623213196648400469, 0.617245694785204857, 777 778 0.611260466978157169, 779 0.605258418111758623, 780 0.599240455613624601, 781 0.593207489317293168, 782 0.587160431324671883, 783 0.581100195868139213, 784 0.575027699172326323, 785 0.568943859315595546, 786 0.562849596091239857, 787 0.556745830868422997, 788 0.550633486452880572, 789 0.544513486947404646, 790 0.538386757612132172, 791 0.532254224724658109, 792 0.526116815439995000, 793 0.519975457650400319, 794 0.513831079845091021, 795 0.507684610969869055, 796 0.501536980286677703, 797 0.495389117233109688, 798 0.489241951281888854, 799 0.483096411800346681, 800 0.476953427909915018, 801 0.470813928345654997, 802 0.464678841315845415, 803 0.458549094361650311, 804 0.452425614216887317, 805 0.446309326667918449,

```
806
       0.440201156413683459,
807
       0.434102026925898610,
808
        0.428012860309439747,
809
        0.421934577162933144,
810
        0.415868096439573343,
        0.409814335308190114,
811
812
        0.403774209014584995,
813
        0.397748630743158804,
        0.391738511478850693,
814
815
        0.385744759869409370,
816
        0.379768282088018327,
817
        0.373809981696294757,
818
        0.367870759507683276,
819
        0.361951513451265472,
820
        0.356053138436005390,
821
        0.350176526215451256,
822
        0.344322565252914603,
823
        0.338492140587146939,
824
        0.332686133698534003,
        0.326905422375827870,
825
826
        0.321150880583437159,
827
        0.315423378329296344,
828
        0.309723781533331410,
829
        0.304052951896545298,
830
        0.298411746770740227,
831
        0.292801019028898213,
832
        0.287221616936237656,
833
       0.281674384021967983,
        0.276160158951759327,
834
        0.270679775400947453,
835
        0.265234061928493969,
836
837
        0.259823841851719028,
838
        0.254449933121827454,
839
        0.249113148200245660,
840
        0.243814293935788129,
841
        0.238554171442673879,
842
        0.233333575979407848,
843
        0.228153296828549457,
844
        0.223014117177383564,
845
        0.217916813999513403,
846
        0.212862157937392882,
847
        0.207850913185815778,
848
        0.202883837376379883,
849
        0.197961681462944072,
850
        0.193085189608093566,
851
        0.188255099070633203,
852
        0.183472140094123992,
853
        0.178737035796480204,
854
        0.174050502060643708,
855
        0.169413247426352498,
856
        0.164825972983019098,
857
        0.160289372263735408,
858
        0.155804131140419966,
859
        0.151370927720123671,
860
        0.146990432242509073,
861
        0.142663306978519533,
862
        0.138390206130252547,
863
        0.134171775732053911,
864
        0.130008653552845688,
        0.125901468999704391,
865
866
        0.121850843022703492,
867
        0.117857388021033904,
```

```
868
       0.113921707750417878,
       0.110044397231829294,
869
       0.106226042661534792,
870
       0.102467221322468549,
871
872
       0.098768501496955430,
873
       0.095130442380794267,
874
       0.091553593998715044,
875
       0.088038497121222858,
876
       0.084585683182840765,
877
       0.081195674201764101,
878
       0.077868982700938449,
879
       0.074606111630573402,
880
       0.071407554292103159,
881
       0.068273794263606080,
882
       0.065205305326694496,
883
       0.062202551394885286,
884
       0.059265986443462593,
885
       0.056396054440842724,
886
       0.053593189281452569,
887
       0.050857814720130512,
888
       0.048190344308060407,
889
       0.045591181330248531,
890
       0.043060718744552196,
891
       0.040599339122270095,
892
       0.038207414590302524,
893
       0.035885306774891212,
894
       0.033633366746946003,
895
       0.031451934968968087,
896
       0.029341341243576458,
897
       0.027301904663646570,
898
       0.025333933564067435,
899
       0.023437725475126125,
900
       0.021613567077524876,
901
       0.019861734159038968,
902
       0.018182491572821535,
903
       0.016576093197361197,
904
       0.015042781898099433,
905
       0.013582789490712122,
906
       0.012196336706062738,
907
       0.010883633156830497,
908
       0.009644877305819977,
909
       0.008480256435956068,
910
       0.007389946621969679,
911
       0.006374112703777302,
912
       0.005432908261559732,
913
       0.004566475592542640,
914
       0.003774945689483389,
915
       0.003058438220866544,
916
       0.002417061512811680,
917
       0.001850912532696092,
918
       0.001360076874494465,
919
       0.000944628745838338,
920
       0.000604630956796859,
921
       0.000340134910380874,
922
       0.000151180594771427,
923
       0.000037796577274096,
924
       0.0000000000000000000
925 };
926
927
928 int main(){
       //init system clocks and get board speed running at 200 MHz
929
```

```
InitSysCtrl();
930
931
       EALLOW;
932
       //disable all interrupt
933
       Interrupt_initModule();
934
       //init interrupt table
935
       Interrupt initVectorTable();
       // initialize the timer structs and setup the timers in their default states
936
937
       InitCpuTimers();
938
939
       EALLOW;
940
       initDMAInput();
941
        EALLOW;
942
       initDMAOutput();
943
       EALLOW;
       initFFT();
944
945
       /*
946
       GpioCtrlRegs.GPAMUX1.bit.GPIO0=0;
947
       GpioCtrlRegs.GPAMUX1.bit.GPIO1=0;
948
       GpioCtrlRegs.GPAGMUX1.bit.GPIO0=0;
949
       GpioCtrlRegs.GPAGMUX1.bit.GPIO1=0;
950
951
       GpioCtrlRegs.GPADIR.bit.GPIO0=1;
952
       GpioCtrlRegs.GPADIR.bit.GPI01=1;
953
954
       //switch
955
       //GpioCtrlRegs.GPAMUX1.bit.GPIO8=0;
956
       //GpioCtrlRegs.GPAGMUX1.bit.GPIO8=0;
957
       GpioCtrlRegs.GPAPUD.bit.GPIO8=0;
958
       //GpioCtrlRegs.GPAMUX1.bit.GPIO9=0;
959
       //GpioCtrlRegs.GPAGMUX1.bit.GPIO9=0;
960
       GpioCtrlRegs.GPAPUD.bit.GPIO9=0;
961
962
        * P95 C2 col4
963
                            input
        * P139 E1 col3
964
965
        * P56 B2 col2
        * P97 D1 col1
966
        * P94 C2 row1
967
                            output
        * P52 B2 row2
968
        * P41 B1 row3
969
970
        * P40 B1 row4
971
972
       /*
973
       GpioCtrlRegs.GPCMUX2.bit.GPIO95=0;
974
       GpioCtrlRegs.GPEMUX1.bit.GPI0139=0;
975
       GpioCtrlRegs.GPBMUX2.bit.GPI056=0;
976
       GpioCtrlRegs.GPDMUX1.bit.GPIO97=0;
977
       GpioCtrlRegs.GPCMUX2.bit.GPIO94=0;
978
       GpioCtrlRegs.GPBMUX2.bit.GPIO52=0;
979
       GpioCtrlRegs.GPBMUX1.bit.GPIO41=0;
980
       GpioCtrlRegs.GPBMUX1.bit.GPIO40=0;
981
       GpioCtrlRegs.GPCGMUX2.bit.GPI095=0;
982
       GpioCtrlRegs.GPEGMUX1.bit.GPI0139=0;
983
       GpioCtrlRegs.GPBGMUX2.bit.GPI056=0;
984
       GpioCtrlRegs.GPDGMUX1.bit.GPIO97=0;
985
       GpioCtrlRegs.GPCGMUX2.bit.GPIO94=0;
986
       GpioCtrlRegs.GPBGMUX2.bit.GPI052=0;
987
       GpioCtrlRegs.GPBGMUX1.bit.GPIO41=0;
988
       GpioCtrlRegs.GPBGMUX1.bit.GPIO40=0;
989
990
991
       GpioCtrlRegs.GPCDIR.bit.GPI094=1;
```

```
992
        GpioCtrlRegs.GPBDIR.bit.GPI052=1;
 993
        GpioCtrlRegs.GPBDIR.bit.GPIO41=1;
 994
        GpioCtrlRegs.GPBDIR.bit.GPI040=1;
 995
 996
        GpioCtrlRegs.GPCPUD.bit.GPIO95=0;
 997
        GpioCtrlRegs.GPEPUD.bit.GPI0139=0;
 998
        GpioCtrlRegs.GPBPUD.bit.GPI056=0;
 999
        GpioCtrlRegs.GPDPUD.bit.GPIO97=0;
1000
1001
1002
         //init ADCA, prescaler=4.0, 12 bit, single-ended
         initAdc(ADCA_BASE, ADC_CLK_DIV_4_0, ADC_RESOLUTION_12BIT,
1003
1004
                 ADC MODE SINGLE ENDED);
1005
         * ADCA
1006
         * SOC0
1007
         * Trigger source=Timer0 <u>Int</u>
1008
         * channel: ACDIN1
1009
1010
         * sampleWindow=50
         * trigger ADCA INT1 when complete
1011
1012
1013
        initAdcSoc(ADCA BASE, ADC SOC NUMBERO, ADC TRIGGER CPU1 TINTO,
                    ADC_CH_ADCIN1, 50, ADC_INT_NUMBER1);
1014
1015
1016
         //init AIC23
1017
        initMcbspbI2S();
1018
        //init SPIA
1019
        InitSPIA();
1020
        //InitAIC23_SR(SR32);
1021
        InitAIC23();
1022
1023
        initGPIO();
1024
        initTimer0();
1025
        initTimer1();
1026
        initTimer2();
1027
        //point INT to my ISR
1028
1029
        Interrupt register(INT DMA CH1, &dmaCh1ISR);
        Interrupt_register(INT_ADCA1, &adcA1ISR);
1030
1031
        //enable INT
        Interrupt_enable(INT_DMA_CH1);
1032
1033
        Interrupt_enable(INT_ADCA1);
1034
        //enable global interrupt
1035
        Interrupt_enableMaster();
1036
1037
        while(1){
1038
             isDistance=GpioDataRegs.GPADAT.bit.GPIO8;
1039
             if(distanceFlag){
1040
                 GpioDataRegs.GPESET.bit.GPI0131=1;
                 CPUTimer_startTimer(CPUTIMER2_BASE);
1041
1042
                 distanceFlag=0;
1043
             if(keypadFlag){
1044
                 key=checkKey();
1045
1046
                 if(key==17){
1047
                     pitchShiftRatio=1;
1048
                     keypad_cont=GpioDataRegs.GPADAT.bit.GPIO9;
1049
                     if(!keypad_cont){
1050
                         amp=0;
1051
                     }
1052
                 }
                 else{
1053
```

```
1054
                     pitchShiftRatio=0.5+0.1*key;
1055
1056
                 keypadFlag=0;
1057
             if(received256){
1058
1059
                 GpioDataRegs.GPASET.bit.GPIO0=1;
                 //format FFT input
1060
                 for(int i=0; i<FFT_SIZE; i+=2){</pre>
1061
1062
                     oneside[i]=hann[(i>>1)]*(float)prevInPtr[i];
1063
                     //oneside[i]=(float)prevInPtr[i];
1064
                     oneside[i+1]=0;
1065
                 for(int i=FFT SIZE; i<2*FFT SIZE; i+=2){</pre>
1066
1067
                     oneside[i]=hann[(i>>1)]*(float)currInPtr[i-512];
1068
                     //oneside[i]=(float)currInPtr[i-512];;
1069
                     oneside[i+1]=0;
                 }
1070
1071
1072
                 float step=1/pitchShiftRatio;
1073
                 //do FFT
                 doFFT(oneside, rightMag, rightPhase);
1074
1075
                 float* temp=leftMag;
1076
                 leftMag=rightMag;
1077
                 rightMag=temp;
1078
                 temp=leftPhase;
1079
                 leftPhase=rightPhase;
1080
                 rightPhase=temp;
1081
1082
                 if(firstTime){
1083
                     firstTime=0;
                     for(int i=0; i<=FFT_SIZE/2; i++){</pre>
1084
1085
                         phaseAccum[i]=leftPhase[i];
1086
                     }
1087
                 }
1088
1089
                 //phase vocoder
1090
                 while(currInterPos>=0 && currInterPos<1){</pre>
1091
                     freqInterpolate(rightMag, rightPhase, leftMag, leftPhase, currInterPos,
1092
                                      magOut, phaseOut);
1093
                     convertToRect(magOut, phaseOut, currIfftOut);
1094
                     doIFFT(currIfftOut, rect);
1095
1096
                     //hanning
1097
                     for(int i=0; i<FFT_SIZE*2; i+=2){</pre>
                         currIfftOut[(i>>1)]=currIfftOut[i]*hann[(i>>1)];
1098
1099
1100
1101
                     //Uint16 count=0;
1102
                     while(currPos<FFT_SIZE/2-1){</pre>
1103
                         Uint16 leftPos=(int16)currPos;
1104
                         Uint16 rightPos=leftPos+1;
1105
1106
                         float right=currPos-leftPos;
1107
                         float left=1-right;
1108
                         float currPoint=left*(currIfftOut[leftPos]+prevIfftOut[leftPos
    +256])+right*(currIfftOut[rightPos]+prevIfftOut[rightPos+256]);
                         currPoint*=amp;
1110
1111
1112
                         outBufs[outBufCounter>>11][outBufCounter&0x7FF]=(int32)currPoint;
                         outBufCounter=(outBufCounter+1)&OUT_BUF_MASK;
1113
1114
                         outBufs[outBufCounter>>11][outBufCounter&0x7FF]=(int32)currPoint;
```

```
1115
                         outBufCounter=(outBufCounter+1)&OUT_BUF_MASK;
1116
                         currPos+=pitchShiftRatio;
1117
                         //count++;
                     }
1118
1119
                     currPos-=(FFT_SIZE/2-1);
1120
                     temp=currIfftOut;
1121
                     currIfftOut=prevIfftOut;
1122
                     prevIfftOut=temp;
1123
1124
                     //outBufCounter=(outBufCounter+count)&OUT_BUF_MASK;
1125
1126
                     currInterPos+=step;
1127
                 currInterPos--;
1128
1129
1130
                 received256=0;
1131
                 GpioDataRegs.GPACLEAR.bit.GPI00=1;
1132
            }
1133
        }
1134 }
1135
     interrupt void dmaCh1ISR(void){
1136
1137
        GpioDataRegs.GPATOGGLE.bit.GPIO1=1;
1138
1139
        volatile int16* src addr1=&McbspbRegs.DRR2.all;
1140
        volatile int16* dest_addr1=0;
1141
1142
        if(currInBuf==0){
1143
            currInPtr=&inBuf0[0];
1144
            prevInPtr=&inBuf2[0];
1145
            currInBuf=1;
            dest_addr1=(int16*)(&inBuf1[0])+1;
1146
1147
1148
        else if(currInBuf==1){
1149
            currInPtr=&inBuf1[0];
1150
            prevInPtr=&inBuf0[0];
1151
            currInBuf=2;
1152
            dest_addr1=(int16*)(&inBuf2[0])+1;
1153
        }
1154
        else{
            currInPtr=&inBuf2[0];
1155
1156
            prevInPtr=&inBuf1[0];
1157
            currInBuf=0;
1158
            dest_addr1=(int16*)(&inBuf0[0])+1;
1159
        }
1160
1161
        DMACH1AddrConfig(dest_addr1, src_addr1);
1162
        received256=1;
1163
        Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP7);
1164
        StartDMACH1();
1165
1166 }
1167
1168
     _interrupt void ECHO_ISR(void){
1169
        //rising edge
1170
        if(GpioDataRegs.GPEDAT.bit.GPIO130==1){
1171
            CPUTimer_startTimer(CPUTIMER1_BASE);
1172
            timeOut=0;
1173
1174
        //falling edge
1175
        else{
1176
            if(timeOut){
```

```
1177
                 pitchShiftRatio=1;
1178
                 timeOut=0;
             }
1179
            else{
1180
1181
                 //stop timer
1182
                 CPUTimer stopTimer(CPUTIMER1 BASE);
                 int32 timer1Count=CpuTimer1.RegsAddr->PRD.all - CpuTimer1.RegsAddr->TIM.all;
1183
1184
                 if(timer1Count>=1176400){
1185
                     pitchShiftRatio=2;
1186
1187
                 else{
1188
                     pitchShiftRatio=0.5+(timer1Count*1.5/1177000.0);
1189
1190
                 timeOut=0;
1191
             }
1192
        }
1193
        Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP12);
1194 }
1195
1196 __interrupt void Timer0_ISR(void){
1197
1198
        volumeFlag=1;
1199
1200
        if(isDistance){
1201
             distanceFlag=1;
1202
1203
        else{
1204
             keypadFlag=1;
1205
        }
1206
         //ack group 1 interrupt for TIMER0
1207
1208
        Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP1);
1209 }
1210
1211 __interrupt void Timer1_ISR(void){
1212
        timeOut=1;
1213
        //stop timer
1214
        CPUTimer stopTimer(CPUTIMER1 BASE);
1215
        //ack group 13 interrupt for TIMER1
1216
        Interrupt_clearACKGroup(INTERRUPT_CPU_INT13);
1217 }
1218
1219 __interrupt void Timer2_ISR(void){
1220
        GpioDataRegs.GPECLEAR.bit.GPI0131=1;
1221
        //stop timer
1222
        CPUTimer_stopTimer(CPUTIMER2_BASE);
1223
        //ack group 14 interrupt for TIMER2
1224
        Interrupt_clearACKGroup(INTERRUPT_CPU_INT14);
1225 }
1226
1227
      _interrupt void adcA1ISR(){
1228
        //read ADC result
1229
        Uint16 result=ADC readResult(ADCARESULT BASE, ADC SOC NUMBER0);
1230
        amp=result*0.011936;
1231
        amp=0.1*amp*amp;
1232
        if(isDistance==0 && key==17){
1233
             keypad_cont=GpioDataRegs.GPADAT.bit.GPIO9;
1234
             if(!keypad_cont){
1235
                 amp=0;
1236
             }
1237
1238
        // Clear the interrupt flag and issue ACK
```

```
1239
        ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);
1240
        Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP1);
1241 }
1242
1243 //DMA1=ping-pong input
1244 void initDMAInput(){
        //get src/dest addr
        volatile int16* src_addr=&McbspbRegs.DRR2.all;
1246
1247
        volatile int16* dest_addr=(int16*)(&inBuf0[0])+1;
1248
1249
        //each burst (one 32-bit McbspB sample) has 2 words
1250
        Uint16 burst_size=2;
1251
        int16 src burst step=1;
                                     //DRR2->DRR1
1252
        int16 dest_burst_step=-1;
                                     //little endian
1253
1254
        Uint16 trans_size=FFT_SIZE;
1255
        int16 src_trans_step=-1;
                                      //DRR1->DRR2
1256
        int16 dest_trans_step=3;
                                      //move to high word of next data
1257
1258
        //disable addr wrapping
1259
        Uint16 src_wrap_size=0xFFFF;
1260
        int16 src_wrap_step=0;
1261
        Uint16 dest_wrap_size=0xFFFF;
1262
        int16 dest_wrap_step=0;
1263
1264
        //trigger source=McbspB receive
1265
        Uint16 per_sel=74;
1266
1267
        //reset DMA
1268
        DMAInitialize();
1269
        //set the <u>src</u> and <u>dest</u> <u>addr</u>
1270
        DMACH1AddrConfig(dest_addr, src_addr);
1271
        //configure burst
1272
        DMACH1BurstConfig(burst_size-1, src_burst_step, dest_burst_step);
1273
        //configure transfer
1274
        DMACH1TransferConfig(trans_size-1, src_trans_step, dest_trans_step);
1275
        //configure wrap
1276
        DMACH1WrapConfig(src_wrap_size, src_wrap_step, dest_wrap_size, dest_wrap_step);
1277
        //configure mode
1278
        DMACH1ModeConfig(
1279
                 per_sel,
1280
                 PERINT_ENABLE,
1281
                 ONESHOT_DISABLE,
1282
                 CONT_DISABLE,
1283
                 SYNC_DISABLE,
1284
                 SYNC_SRC,
1285
                 OVRFLOW DISABLE,
1286
                 SIXTEEN_BIT,
                 CHINT_END,
1287
1288
                 CHINT_ENABLE
1289
1290
        EALLOW;
1291
        CpuSysRegs.SECMSEL.bit.PF2SEL = 1;
1292 }
1293
1294 //DMA2=output
1295 void initDMAOutput(){
1296
        //get src/dest addr
        volatile int16* src_addr=(int16*)(&(outBufs[0][0]))+1;
1297
1298
        volatile int16* dest addr=&McbspbRegs.DXR2.all;
1299
1300
        //each burst (one 32-bit McbspB sample) has 2 words
```

```
Uint16 burst_size=2;
1301
1302
        int16 src_burst_step=-1;
                                      //DRR2->DRR1
1303
        int16 dest_burst_step=1;
                                    //little endian
1304
1305
        Uint16 trans_size=2048*8;
1306
        int16 src trans step=3;
                                    //DRR1->DRR2
1307
        int16 dest_trans_step=-1;
                                      //move to high word of next data
1308
1309
        //disable addr wrapping
1310
        Uint16 src_wrap_size=0xFFFF;
1311
        int16 src_wrap_step=0;
1312
        Uint16 dest_wrap_size=0xFFFF;
1313
        int16 dest_wrap_step=0;
1314
        //trigger source=McbspB receive
1315
1316
        Uint16 per_sel=74;
1317
1318
        //reset DMA
1319
        //set the src and dest addr
1320
        DMACH2AddrConfig(dest_addr, src_addr);
1321
        //configure burst
1322
        DMACH2BurstConfig(burst_size-1, src_burst_step, dest_burst_step);
1323
        //configure transfer
1324
        DMACH2TransferConfig(trans_size-1, src_trans_step, dest_trans_step);
1325
        //configure wrap
1326
        DMACH2WrapConfig(src_wrap_size, src_wrap_step, dest_wrap_size, dest_wrap_step);
1327
        //configure mode
1328
        DMACH2ModeConfig(
1329
                 per_sel,
1330
                 PERINT_ENABLE,
1331
                 ONESHOT_DISABLE,
1332
                 CONT_ENABLE,
1333
                 SYNC_DISABLE,
1334
                 SYNC SRC,
1335
                 OVRFLOW_DISABLE,
                 SIXTEEN_BIT,
1336
                 CHINT_END,
1337
                CHINT DISABLE
1338
1339
        );
1340
        //start DMA
        StartDMACH1();
1341
1342
        StartDMACH2();
1343 }
1344
1345 void initFFT(){
1346
        CFFT f32 setOutputPtr(hnd cfft, test output);
1347
        CFFT f32 setStages(hnd cfft, FFT STAGES);
        CFFT_f32_setFFTSize(hnd_cfft, FFT_SIZE);
1348
1349
        CFFT_f32_setTwiddlesPtr(hnd_cfft, twiddleFactors);
1350
        CFFT_f32_sincostable(hnd_cfft);
1351 }
1352
1353 void initGPIO(){
1354
        EALLOW;
1355
1356
        //P130->echo (input)
1357
        //P131->trig (output)
1358
1359
        GpioCtrlRegs.GPEMUX1.bit.GPI0130=0;
        GpioCtrlRegs.GPEMUX1.bit.GPI0131=0;
1360
1361
        GpioCtrlRegs.GPEGMUX1.bit.GPI0130=0;
1362
        GpioCtrlRegs.GPEGMUX1.bit.GPI0131=0;
```

```
*/
1363
1364
        GpioCtrlRegs.GPEPUD.bit.GPI0130=0;
1365
        GpioCtrlRegs.GPEDIR.bit.GPI0130=0;
1366
1367
        GpioCtrlRegs.GPEDIR.bit.GPI0131=1;
1368
1369
        //P130 need both edge interrupt trigger
1370
        //P130=INPUT6=XINT3
1371
        InputXbarRegs.INPUT6SELECT=130;
1372
        XintRegs.XINT3CR.bit.POLARITY=3;
1373
        XintRegs.XINT3CR.bit.ENABLE=1;
1374
        //point int to ISR
1375
        Interrupt_register(INT_XINT3, &ECHO_ISR);
1376
        //enable int in PIE and IER
1377
        Interrupt_enable(INT_XINT3);
1378
1379 }
1380
1381 void initTimer0(){
        //timer0, cpu freq=200MHz, timer period=100ms
1383
        ConfigCpuTimer(&CpuTimer0, 200, 100000);
1384
        Interrupt_register(INT_TIMER0, &Timer0_ISR);
        Interrupt_enable(INT_TIMER0);
1385
        CPUTimer_startTimer(CPUTIMER0_BASE);
1386
1387 }
1388
1389 void initTimer1(){
1390
        //timer1, <a href="mailto:cpu">cpu</a> freq=200MHz, timer period=10ms
1391
        ConfigCpuTimer(&CpuTimer1, 200, 10000);
1392
        Interrupt_register(INT_TIMER1, &Timer1_ISR);
1393
        Interrupt_enable(INT_TIMER1);
1394 }
1395
1396 void initTimer2(){
1397
        //timer2, cpu freq=200MHz, timer period=10us
        ConfigCpuTimer(&CpuTimer2, 200, 50);
1398
        Interrupt_register(INT_TIMER2, &Timer2_ISR);
1399
1400
        Interrupt_enable(INT_TIMER2);
1401 }
1402
1403 void doFFT(float* inBuf, float* outMagnitude, float* outPhase){
1404
        CFFT_f32_setInputPtr(hnd_cfft, inBuf);
1405
        CFFT_f32_setOutputPtr(hnd_cfft, test_output);
1406
        CFFT_f32(hnd_cfft);
1407
        float* p_temp=CFFT_f32_getCurrInputPtr(hnd_cfft);
1408
        //number of stage is odd, output in currInputPtr
1409
        //doIFFT(p_temp, test_output);
1410
        CFFT_f32_setCurrInputPtr(hnd_cfft, p_temp);
1411
        CFFT_f32_setCurrOutputPtr(hnd_cfft, outMagnitude);
1412
        CFFT_f32_mag(hnd_cfft);
        CFFT_f32_setCurrInputPtr(hnd_cfft, p_temp);
1413
        CFFT_f32_setCurrOutputPtr(hnd_cfft, outPhase);
1414
1415
        CFFT f32 phase(hnd cfft);
1416 }
1417
1418 void freqInterpolate(float* mag1, float* phase1, float* mag2, float* phase2, float right,
1419
                          float* magOut, float* phaseOut){
1420
        float left=1-right;
1421
        for(Uint16 i=0; i<=FFT_SIZE/2; i++){</pre>
            magOut[i]=left*mag1[i]+right*mag2[i];
1422
1423
             float dp=phase2[i]-phase1[i]-dph[i];
1424
            dp=dp-2*PI*(roundf(dp*ONE_OVER_TWO_PI));
```

```
phaseOut[i]=phaseAccum[i]-2*PI*(roundf(phaseAccum[i]*ONE_OVER_TWO_PI));
1425
1426
            phaseAccum[i]=phaseAccum[i]+dph[i]+dp;
1427
            if(phaseAccum[i]>1000000){
1428
                 firstTime=1;
1429
            }
1430
        }
1431 }
1432
1433 void convertToRect(float* mag, float* phase, float* rect){
1434
        for(Uint16 i=0; i<=FFT_SIZE; i+=2){</pre>
1435
             rect[i]=mag[i/2]*myCos(phase[i/2]);
1436
            rect[i+1]=mag[i/2]*mySin(phase[i/2]);
1437
1438
        Uint16 totalLen=FFT_SIZE*2;
1439
1440
        for(Uint16 i=FFT_SIZE+2; i<totalLen; i+=2){</pre>
1441
            rect[i]=rect[totalLen-i];
1442
            rect[i+1]=-rect[totalLen+1-i];
1443
        }
1444
1445 }
1446
1447
1448 void doIFFT(float* inBuf, float* outBuf){
        CFFT_f32_setInputPtr(hnd_cfft, inBuf);
1449
1450
        CFFT f32 setOutputPtr(hnd cfft, outBuf);
1451
        ICFFT_f32(hnd_cfft);
1452 }
1453
1454 /*
1455 * P95 C2 col4
                         input
1456 * P139 E1 col3
1457 * P56 B2 col2
1458 * P97 D1 col1
1459 *
1460 * P94 C2 row1
                         output
1461 * P52 B2 row2
1462 * P41 B1 row3
1463 * P40 B1 row4
1464 */
1465
1466 volatile Uint16 cols[4];
1467
1468 void getCols(){
1469
        cols[3]=GpioDataRegs.GPCDAT.bit.GPIO95;
1470
        cols[2]=GpioDataRegs.GPEDAT.bit.GPI0139;
1471
        cols[1]=GpioDataRegs.GPBDAT.bit.GPIO56;
1472
        cols[0]=GpioDataRegs.GPDDAT.bit.GPIO97;
1473 }
1474
1475 void clearRows(){
        GpioDataRegs.GPCCLEAR.bit.GPIO94=1;
1476
1477
        GpioDataRegs.GPBCLEAR.bit.GPI052=1;
1478
        GpioDataRegs.GPBCLEAR.bit.GPIO41=1;
1479
        GpioDataRegs.GPBCLEAR.bit.GPIO40=1;
1480 }
1481
1482 Uint16 prev=0;
1483 Uint16 checkKey(){
1484
        clearRows();
1485
        getCols();
1486
        for(int i=0; i<4; i++){</pre>
```

```
1487
             if(cols[i]==0){
1488
                 GpioDataRegs.GPCSET.bit.GPIO94=1;
1489
                 DELAY_US(1);
1490
                 getCols();
1491
                 if(cols[i]==1){
1492
                      prev=i;
1493
                      return i;
1494
1495
                 clearRows();
1496
                 GpioDataRegs.GPBSET.bit.GPI052=1;
                 DELAY_US(1);
1497
1498
                 getCols();
1499
                 if(cols[i]==1){
1500
                      prev=4+i;
1501
                      return 4+i;
1502
                 }
1503
                 clearRows();
1504
                 GpioDataRegs.GPBSET.bit.GPIO41=1;
1505
                 DELAY_US(1);
1506
                 getCols();
1507
                 if(cols[i]==1){
1508
                      prev=8+i;
1509
                      return 8+i;
1510
1511
                 clearRows();
1512
                 GpioDataRegs.GPBSET.bit.GPIO40=1;
1513
                 DELAY_US(1);
                 getCols();
1514
1515
                 if(cols[i]==1){
1516
                      prev=12+i;
1517
                      return 12+i;
1518
1519
                 return prev;
1520
             }
1521
1522
1523
         return 17;
1524 }
1525
1526 float mySin(float rad){
        if(rad<0){</pre>
1527
1528
            rad=-rad;
1529
            Uint16 index=(Uint16)roundf(rad*TRIG_RATIO);
1530
            if(index>=256){
1531
                index-=256;
1532
            }
1533
            return -twiddleFactors[index];
1534
1535
        Uint16 index=(Uint16)roundf(rad*TRIG_RATIO);
1536
        if(index>=256){
1537
           index-=256;
1538
1539
        return twiddleFactors[index];
1540 }
1541
1542 float myCos(float rad){
1543
         if(rad<0){</pre>
1544
             rad=-rad;
1545
1546
         Uint16 index=(Uint16)roundf(rad*TRIG RATIO)+128;
1547
         if(index>=384){
            index-=256;
1548
```

```
Final.c
```

```
1549     }
1550     return twiddleFactors[index];
1551 }
1552
```