```
 1: #include <hidef.h>      /* common defines and macros */
 2: #include <mc9s12dp512.h>     /* derivative information */
 3: #pragma LINK_INFO DERIVATIVE "mc9s12dp512"
 4:
 5: #include "PLL.h"
 6: #include "DAC.h"
 7: #include "music.h"
 8: #include "switch.h"
 9:
10:
11: void main(void) {
12:   // Initialize needed modules
13:   PLL_Init();
14:   DAC_Init();
15:   Switch_Init();
16:   Music_InitOC0();
17:   Music_InitOC1();
18:   Music_InitOC2();
19:   Music_InitOC3();
20:
21:   // Start off paused
22:   asm sei
23:
24:   for(;;) {
25:     // Play button
26:     if(Switch_Data()&0x08) {
27:       asm cli
28:     }
29:     // Pause button
30:     if(Switch_Data()&0x04) {
31:       asm sei
32:     }
33:     // Restart button
34:     if(Switch_Data()&0x02) {
35:       Music_Restart();
36:     }
37:   }
38: }
```

```
 1: #include <hidef.h>      /* common defines and macros */
 2: #include <mc9s12dp512.h>     /* derivative information */
 3: #pragma LINK_INFO DERIVATIVE "mc9s12dp512"
 4:
 5: //---------------------DAC_Init---------------------
 6: // initializes DAC
 7: // Input: none
 8: // Output: none
 9: void DAC_Init(void);
10:
11: //---------------------DAC_Out----------------------
12: // outputs 12 bits to DAC
13: // Input: none
14: // Output: none
15: void DAC_Out(unsigned short data);
16:
```

```
 1: #include "DAC.h"
 2:
 3: // 9S12DP512 SPI1 interface to Max539
 4: // PS6 (out) SCLK synchronous clock
 5: // PS5 (out) MOSI serial data output
 6: // PS7 (out) CS used to latch data into Max539
 7: // PS4 (in) is associated with SPI1, but not used
 8:
 9: //---------------------DAC_Init----------------------
10: // initializes DAC
11: // Input: none
12: // Output: none
13: void DAC_Init(void) {
14:   DDRS |=  0xE0;  // 1) make PS5, PS6, PS7 outputs, PS4 input
15:   DDRS &= ~0x10;  // DDRS
16:   SPI0CR1 = 0x58; // 2) enable SPI, no interrupts, master, CPOL=1, CPHA=0
17:                   // SPI0CR1 = 0101 1000
18:   SPI0CR2 = 0x00; // 3) set up PS7 as a regular output
19:                   // SSOE=0, MODFEN=0 SPI0CR1, SPI0CR2
20:   SPI0BR = 0x00;  // 4) set the baud rate, SPI0BR
21:   PTS  |= 0x80;   // 5) make PS7=CS high
22: }
23:
24: //---------------------transmitByte---------------------
25: // outputs byte to DAC
26: // Input: none
27: // Output: none
28: void transmitByte(unsigned char data) {
29:   unsigned char dummy;
30:   while(!(SPI0SR&0x20)) {}  // 1) wait for SPTEF to be 1, SPI0SR
31:   SPI0DR = data;            // 2) write 8-bit data to SPI0DR
32:   while(!(SPI0SR&0x80)) {}  // 3) wait for SPIF to be 1, SPI0SR
33:   dummy = SPI0DR;            // 4) clear the SPIF flag by reading the data
34:                             // dummy = SPI0DR;
35: }
36:
37: //---------------------DAC_Out----------------------
38: // outputs 12 bits to DAC
39: // Input: none
40: // Output: none
41: void DAC_Out(unsigned short data) {
42:   PTS &= ~0x80;                    // 1) set PS7=CS low
43:   transmitByte((data&0x3F00) >> 8); // 2) transmit most significant 8-bit data to the DAC
44:   transmitByte(data&0x00FF);        // 3) transmit least significant 8-bit data to the DAC
45:   PTS |=  0x80;                    // 4) set PS7=CS high
46: }
```

```
#include <hidef.h>      /* common defines and macros */
#include <mc9s12dp512.h>     /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dp512"

#define SIN 16
#define FREQUENCY 24000000/SIN

#define MREPEAT 36
#define HREPEAT 34
#define BREPEAT 29

#define MELODY  146
#define HARMONY  181
#define BASS  163


typedef const struct Note{
  unsigned short frequency;
  unsigned long length;

} NoteType;

//---------------------Music_InitOC0---------------------
// arm output compare 0 for melody
// also enables timer to 43 ns period
// Input: none
// Output: none
void Music_InitOC0(void);

//---------------------Music_InitOC1---------------------
// arm output compare 1 for harmony
// Input: none
// Output: none
void Music_InitOC1(void);

//---------------------Music_InitOC2---------------------
// arm output compare 2 for bass
// Input: none
// Output: none
void Music_InitOC2(void);

//---------------------Music_InitOC3---------------------
// arm output compare 3 for envelopes at 750 Hz
// Input: none
// Output: none
void Music_InitOC3(void);

//---------------------Music_Restart---------------------
// arm output compare 3 for envelopes at 750 Hz
// Input: none
// Output: none
void Music_Restart(void);
```

```
 1: #include "music.h"
 2: #include "DAC.h"
 3: #include "switch.h"
 4:
 5: // Note index for each part
 6: unsigned static short note1 = 0;
 7: unsigned static short note2 = 0;
 8: unsigned static short note3 = 0;
 9:
10: // Output for each part
11: unsigned static short output1 = 0;
12: unsigned static short output2 = 0;
13: unsigned static short output3 = 0;
14:
15: // Envelope multiplier for each part
16: unsigned static short envelope1 = 1;
17: unsigned static short envelope2 = 1;
18: unsigned static short envelope3 = 1;
19:
20: // Number of envelope interrupts for each part
21: unsigned static short interrupts1 = 1;
22: unsigned static short interrupts2 = 1;
23: unsigned static short interrupts3 = 1;
24:
25: // Stores sin wave
26: const unsigned short SinWave[SIN] = {
27:   683,
28:   944,
29:   1165,
30:   1313,
31:   1365,
32:   1313,
33:   1165,
34:   944,
35:   683,
36:   422,
37:   201,
38:   53,
39:   1,
40:   53,
41:   201,
42:   422
43: };
44:
45: const NoteType melody[MELODY] = {
46:   {FREQUENCY/466,600},
47:   {0,100},
48:   {0,100},
49:   {FREQUENCY/466,100},
50:   {FREQUENCY/466,100},
51:   {FREQUENCY/466,100},
52:   {FREQUENCY/466,100},
53:   {FREQUENCY/466,225},
54:   {FREQUENCY/415,75},
55:   {FREQUENCY/466,300},
56:   {0,100},
57:   {0,100},
58:   {FREQUENCY/466,100},
59:   {FREQUENCY/466,100},
60:   {FREQUENCY/466,100},
61:   {FREQUENCY/466,100},
62:   {FREQUENCY/466,225},
63:   {FREQUENCY/415,75},
64:   {FREQUENCY/466,300},
65:   {0,100},
66:   {0,100},
67:   {FREQUENCY/466,100},
68:   {FREQUENCY/466,100},
69:   {FREQUENCY/466,100},
70:   {FREQUENCY/466,100},
71:   {FREQUENCY/466,150},
72:   {FREQUENCY/349,75},
73:   {FREQUENCY/349,75},
74:   {FREQUENCY/349,150},
75:   {FREQUENCY/349,75},
76:   {FREQUENCY/349,75},
77:   {FREQUENCY/349,150},
78:   {FREQUENCY/349,75},
```

```
  79:   {FREQUENCY/349,75},
  80:   {FREQUENCY/349,150},
  81:   {FREQUENCY/349,150},
  82:   {FREQUENCY/466,300},
  83:   {FREQUENCY/349,525},
  84:   {FREQUENCY/466,75},
  85:   {FREQUENCY/466,75},
  86:   {FREQUENCY/523,75},
  87:   {FREQUENCY/587,75},
  88:   {FREQUENCY/622,75},
  89:   {FREQUENCY/698,600},
  90:   {0,150},
  91:   {FREQUENCY/698,150},
  92:   {FREQUENCY/698,100},
  93:   {FREQUENCY/740,100},
  94:   {FREQUENCY/831,100},
  95:   {FREQUENCY/932,600},
  96:   {0,100},
  97:   {FREQUENCY/932,100},
  98:   {FREQUENCY/932,100},
  99:   {FREQUENCY/932,100},
 100:   {FREQUENCY/831,100},
 101:   {FREQUENCY/740,100},
 102:   {FREQUENCY/831,225},
 103:   {FREQUENCY/740,75},
 104:   {FREQUENCY/698,600},
 105:   {FREQUENCY/698,300},
 106:   {FREQUENCY/622,150},
 107:   {FREQUENCY/622,75},
 108:   {FREQUENCY/698,75},
 109:   {FREQUENCY/740,600},
 110:   {FREQUENCY/698,150},
 111:   {FREQUENCY/622,150},
 112:   {FREQUENCY/554,150},
 113:   {FREQUENCY/554,75},
 114:   {FREQUENCY/622,75},
 115:   {FREQUENCY/698,600},
 116:   {FREQUENCY/622,150},
 117:   {FREQUENCY/554,150},
 118:   {FREQUENCY/523,150},
 119:   {FREQUENCY/523,75},
 120:   {FREQUENCY/587,75},
 121:   {FREQUENCY/659,600},
 122:   {FREQUENCY/784,300},
 123:   {FREQUENCY/698,150},
 124:   {FREQUENCY/349,75},
 125:   {FREQUENCY/349,75},
 126:   {FREQUENCY/349,150},
 127:   {FREQUENCY/349,75},
 128:   {FREQUENCY/349,75},
 129:   {FREQUENCY/349,150},
 130:   {FREQUENCY/349,75},
 131:   {FREQUENCY/349,75},
 132:   {FREQUENCY/349,150},
 133:   {FREQUENCY/349,150},
 134:   {FREQUENCY/466,300},
 135:   {FREQUENCY/349,525},
 136:   {FREQUENCY/466,75},
 137:   {FREQUENCY/466,75},
 138:   {FREQUENCY/523,75},
 139:   {FREQUENCY/587,75},
 140:   {FREQUENCY/622,75},
 141:   {FREQUENCY/698,600},
 142:   {0,150},
 143:   {FREQUENCY/698,150},
 144:   {FREQUENCY/698,100},
 145:   {FREQUENCY/740,100},
 146:   {FREQUENCY/831,100},
 147:   {FREQUENCY/932,600},
 148:   {0,300},
 149:   {FREQUENCY/1109,300},
 150:   {FREQUENCY/1047,300},
 151:   {FREQUENCY/880,300},
 152:   {0,300},
 153:   {FREQUENCY/698,300},
 154:   {FREQUENCY/740,600},
 155:   {0,300},
 156:   {FREQUENCY/932,300},
```

```
157:   {FREQUENCY/880,300},
158:   {FREQUENCY/698,300},
159:   {0,300},
160:   {FREQUENCY/698,300},
161:   {FREQUENCY/740,600},
162:   {0,300},
163:   {FREQUENCY/932,300},
164:   {FREQUENCY/880,300},
165:   {FREQUENCY/698,300},
166:   {0,300},
167:   {FREQUENCY/587,300},
168:   {FREQUENCY/622,600},
169:   {0,300},
170:   {FREQUENCY/740,300},
171:   {FREQUENCY/698,300},
172:   {FREQUENCY/554,300},
173:   {0,300},
174:   {FREQUENCY/466,300},
175:   {FREQUENCY/523,150},
176:   {FREQUENCY/523,75},
177:   {FREQUENCY/587,75},
178:   {FREQUENCY/659,300},
179:   {0,300},
180:   {FREQUENCY/784,300},
181:   {FREQUENCY/698,150},
182:   {FREQUENCY/349,75},
183:   {FREQUENCY/349,75},
184:   {FREQUENCY/349,150},
185:   {FREQUENCY/349,75},
186:   {FREQUENCY/349,75},
187:   {FREQUENCY/349,150},
188:   {FREQUENCY/349,75},
189:   {FREQUENCY/349,75},
190:   {FREQUENCY/349,150},
191:   {FREQUENCY/349,150}
192: };
193:
194: const NoteType harmony[HARMONY] = {
195:   {FREQUENCY/294,600},
196:   {0,100},
197:   {0,100},
198:   {FREQUENCY/294,100},
199:   {FREQUENCY/294,100},
200:   {FREQUENCY/294,100},
201:   {FREQUENCY/294,100},
202:   {FREQUENCY/262,225},
203:   {FREQUENCY/262,75},
204:   {FREQUENCY/262,300},
205:   {0,300},
206:   {FREQUENCY/262,100},
207:   {FREQUENCY/262,100},
208:   {FREQUENCY/262,100},
209:   {FREQUENCY/277,225},
210:   {FREQUENCY/277,75},
211:   {FREQUENCY/277,300},
212:   {0,100},
213:   {0,100},
214:   {FREQUENCY/277,100},
215:   {FREQUENCY/277,100},
216:   {FREQUENCY/277,100},
217:   {FREQUENCY/277,100},
218:   {FREQUENCY/277,150},
219:   {FREQUENCY/220,75},
220:   {FREQUENCY/220,75},
221:   {FREQUENCY/220,150},
222:   {FREQUENCY/220,75},
223:   {FREQUENCY/220,75},
224:   {FREQUENCY/220,150},
225:   {FREQUENCY/220,75},
226:   {FREQUENCY/220,75},
227:   {FREQUENCY/220,150},
228:   {FREQUENCY/220,150},
229:   {FREQUENCY/294,300},
230:   {FREQUENCY/294,100},
231:   {FREQUENCY/294,100},
232:   {FREQUENCY/262,100},
233:   {FREQUENCY/294,225},
234:   {FREQUENCY/294,75},
```

```
235:   {FREQUENCY/294,75},
236:   {FREQUENCY/311,75},
237:   {FREQUENCY/349,75},
238:   {FREQUENCY/392,75},
239:   {FREQUENCY/415,225},
240:   {FREQUENCY/466,75},
241:   {FREQUENCY/466,75},
242:   {FREQUENCY/523,75},
243:   {FREQUENCY/587,75},
244:   {FREQUENCY/622,75},
245:   {FREQUENCY/698,300},
246:   {FREQUENCY/415,100},
247:   {FREQUENCY/466,100},
248:   {FREQUENCY/523,100},
249:   {FREQUENCY/554,225},
250:   {FREQUENCY/370,75},
251:   {FREQUENCY/370,75},
252:   {FREQUENCY/415,75},
253:   {FREQUENCY/466,75},
254:   {FREQUENCY/523,75},
255:   {FREQUENCY/554,200},
256:   {FREQUENCY/554,100},
257:   {FREQUENCY/554,100},
258:   {FREQUENCY/523,100},
259:   {FREQUENCY/466,100},
260:   {FREQUENCY/554,225},
261:   {FREQUENCY/415,75},
262:   {FREQUENCY/415,100},
263:   {FREQUENCY/415,100},
264:   {FREQUENCY/370,100},
265:   {FREQUENCY/415,225},
266:   {FREQUENCY/415,75},
267:   {FREQUENCY/415,100},
268:   {FREQUENCY/370,100},
269:   {FREQUENCY/415,100},
270:   {FREQUENCY/370,150},
271:   {FREQUENCY/370,75},
272:   {FREQUENCY/349,75},
273:   {FREQUENCY/370,150},
274:   {FREQUENCY/370,75},
275:   {FREQUENCY/415,75},
276:   {FREQUENCY/466,300},
277:   {FREQUENCY/415,150},
278:   {FREQUENCY/370,150},
279:   {FREQUENCY/349,150},
280:   {FREQUENCY/349,75},
281:   {FREQUENCY/311,75},
282:   {FREQUENCY/349,150},
283:   {FREQUENCY/349,75},
284:   {FREQUENCY/370,75},
285:   {FREQUENCY/415,300},
286:   {FREQUENCY/370,150},
287:   {FREQUENCY/349,150},
288:   {FREQUENCY/330,300},
289:   {FREQUENCY/330,150},
290:   {FREQUENCY/330,75},
291:   {FREQUENCY/349,75},
292:   {FREQUENCY/392,150},
293:   {FREQUENCY/392,75},
294:   {FREQUENCY/440,75},
295:   {FREQUENCY/466,150},
296:   {FREQUENCY/523,150},
297:   {FREQUENCY/440,150},
298:   {FREQUENCY/220,75},
299:   {FREQUENCY/220,75},
300:   {FREQUENCY/220,150},
301:   {FREQUENCY/220,75},
302:   {FREQUENCY/220,75},
303:   {FREQUENCY/220,150},
304:   {FREQUENCY/220,75},
305:   {FREQUENCY/220,75},
306:   {FREQUENCY/220,150},
307:   {FREQUENCY/220,150},
308:   {FREQUENCY/294,300},
309:   {FREQUENCY/294,100},
310:   {FREQUENCY/294,100},
311:   {FREQUENCY/262,100},
312:   {FREQUENCY/294,225},
```

```
313:   {FREQUENCY/294,75},
314:   {FREQUENCY/294,75},
315:   {FREQUENCY/311,75},
316:   {FREQUENCY/349,75},
317:   {FREQUENCY/392,75},
318:   {FREQUENCY/415,225},
319:   {FREQUENCY/466,75},
320:   {FREQUENCY/466,75},
321:   {FREQUENCY/523,75},
322:   {FREQUENCY/587,75},
323:   {FREQUENCY/622,75},
324:   {FREQUENCY/698,300},
325:   {FREQUENCY/415,100},
326:   {FREQUENCY/466,100},
327:   {FREQUENCY/523,100},
328:   {FREQUENCY/554,600},
329:   {0,300},
330:   {FREQUENCY/659,300},
331:   {FREQUENCY/622,300},
332:   {FREQUENCY/523,300},
333:   {0,300},
334:   {FREQUENCY/440,300},
335:   {FREQUENCY/494,600},
336:   {0,300},
337:   {FREQUENCY/554,300},
338:   {FREQUENCY/523,300},
339:   {FREQUENCY/440,300},
340:   {0,300},
341:   {FREQUENCY/440,300},
342:   {FREQUENCY/494,600},
343:   {0,300},
344:   {FREQUENCY/554,300},
345:   {FREQUENCY/523,300},
346:   {FREQUENCY/440,300},
347:   {0,300},
348:   {FREQUENCY/440,300},
349:   {FREQUENCY/370,600},
350:   {0,300},
351:   {FREQUENCY/494,300},
352:   {FREQUENCY/466,300},
353:   {FREQUENCY/349,300},
354:   {0,300},
355:   {FREQUENCY/277,300},
356:   {FREQUENCY/330,300},
357:   {FREQUENCY/330,150},
358:   {FREQUENCY/330,75},
359:   {FREQUENCY/349,75},
360:   {FREQUENCY/392,150},
361:   {FREQUENCY/392,75},
362:   {FREQUENCY/440,75},
363:   {FREQUENCY/466,150},
364:   {FREQUENCY/523,150},
365:   {FREQUENCY/440,150},
366:   {FREQUENCY/220,75},
367:   {FREQUENCY/220,75},
368:   {FREQUENCY/220,150},
369:   {FREQUENCY/220,75},
370:   {FREQUENCY/220,75},
371:   {FREQUENCY/220,150},
372:   {FREQUENCY/220,75},
373:   {FREQUENCY/220,75},
374:   {FREQUENCY/220,150},
375:   {FREQUENCY/220,150}
376: };
377:
378: const NoteType bass[BASS] = {
379:   {FREQUENCY/117,300},
380:   {FREQUENCY/117,100},
381:   {FREQUENCY/117,100},
382:   {FREQUENCY/117,100},
383:   {FREQUENCY/117,300},
384:   {FREQUENCY/117,100},
385:   {FREQUENCY/117,100},
386:   {FREQUENCY/117,100},
387:   {FREQUENCY/104,300},
388:   {FREQUENCY/104,100},
389:   {FREQUENCY/104,100},
390:   {FREQUENCY/104,100},
```

```
391:   {FREQUENCY/104,300},
392:   {FREQUENCY/104,100},
393:   {FREQUENCY/104,100},
394:   {FREQUENCY/104,100},
395:   {FREQUENCY/92,300},
396:   {FREQUENCY/92,100},
397:   {FREQUENCY/92,100},
398:   {FREQUENCY/92,100},
399:   {FREQUENCY/92,300},
400:   {FREQUENCY/92,100},
401:   {FREQUENCY/92,100},
402:   {FREQUENCY/92,100},
403:   {FREQUENCY/87,300},
404:   {FREQUENCY/87,300},
405:   {FREQUENCY/87,300},
406:   {FREQUENCY/98,150},
407:   {FREQUENCY/110,150},
408:   {FREQUENCY/117,300},
409:   {FREQUENCY/117,100},
410:   {FREQUENCY/117,100},
411:   {FREQUENCY/104,100},
412:   {FREQUENCY/117,300},
413:   {FREQUENCY/117,300},
414:   {FREQUENCY/104,300},
415:   {FREQUENCY/104,100},
416:   {FREQUENCY/104,100},
417:   {FREQUENCY/92,100},
418:   {FREQUENCY/104,300},
419:   {FREQUENCY/104,300},
420:   {FREQUENCY/92,300},
421:   {FREQUENCY/92,100},
422:   {FREQUENCY/92,100},
423:   {FREQUENCY/82,100},
424:   {FREQUENCY/92,300},
425:   {FREQUENCY/92,300},
426:   {FREQUENCY/139,300},
427:   {FREQUENCY/139,100},
428:   {FREQUENCY/139,100},
429:   {FREQUENCY/131,100},
430:   {FREQUENCY/139,300},
431:   {FREQUENCY/139,300},
432:   {FREQUENCY/123,300},
433:   {FREQUENCY/123,100},
434:   {FREQUENCY/123,100},
435:   {FREQUENCY/117,100},
436:   {FREQUENCY/123,300},
437:   {FREQUENCY/123,100},
438:   {FREQUENCY/123,100},
439:   {FREQUENCY/123,100},
440:   {FREQUENCY/117,300},
441:   {FREQUENCY/117,100},
442:   {FREQUENCY/117,100},
443:   {FREQUENCY/104,100},
444:   {FREQUENCY/117,300},
445:   {FREQUENCY/117,100},
446:   {FREQUENCY/117,100},
447:   {FREQUENCY/117,100},
448:   {FREQUENCY/131,300},
449:   {FREQUENCY/131,100},
450:   {FREQUENCY/131,100},
451:   {FREQUENCY/123,100},
452:   {FREQUENCY/131,300},
453:   {FREQUENCY/131,100},
454:   {FREQUENCY/131,100},
455:   {FREQUENCY/131,100},
456:   {FREQUENCY/87,300},
457:   {FREQUENCY/87,300},
458:   {FREQUENCY/87,300},
459:   {FREQUENCY/98,150},
460:   {FREQUENCY/110,150},
461:   {FREQUENCY/117,300},
462:   {FREQUENCY/117,100},
463:   {FREQUENCY/117,100},
464:   {FREQUENCY/104,100},
465:   {FREQUENCY/117,300},
466:   {FREQUENCY/117,300},
467:   {FREQUENCY/104,300},
468:   {FREQUENCY/104,100},
```

```
469:   {FREQUENCY/104,100},
470:   {FREQUENCY/92,100},
471:   {FREQUENCY/104,300},
472:   {FREQUENCY/104,300},
473:   {FREQUENCY/92,300},
474:   {FREQUENCY/92,100},
475:   {FREQUENCY/92,100},
476:   {FREQUENCY/82,100},
477:   {FREQUENCY/92,300},
478:   {FREQUENCY/92,300},
479:   {FREQUENCY/87,300},
480:   {FREQUENCY/87,100},
481:   {FREQUENCY/87,100},
482:   {FREQUENCY/82,100},
483:   {FREQUENCY/87,300},
484:   {FREQUENCY/87,300},
485:   {FREQUENCY/82,100},
486:   {FREQUENCY/117,100},
487:   {FREQUENCY/139,100},
488:   {FREQUENCY/165,100},
489:   {FREQUENCY/233,100},
490:   {FREQUENCY/277,100},
491:   {FREQUENCY/330,300},
492:   {0,300},
493:   {FREQUENCY/349,300},
494:   {FREQUENCY/87,100},
495:   {FREQUENCY/87,100},
496:   {FREQUENCY/87,100},
497:   {FREQUENCY/87,300},
498:   {0,300},
499:   {FREQUENCY/82,100},
500:   {FREQUENCY/117,100},
501:   {FREQUENCY/139,100},
502:   {FREQUENCY/165,100},
503:   {FREQUENCY/233,100},
504:   {FREQUENCY/277,100},
505:   {FREQUENCY/330,300},
506:   {0,300},
507:   {FREQUENCY/349,300},
508:   {FREQUENCY/87,100},
509:   {FREQUENCY/87,100},
510:   {FREQUENCY/87,100},
511:   {FREQUENCY/87,300},
512:   {0,300},
513:   {FREQUENCY/123,300},
514:   {FREQUENCY/123,100},
515:   {FREQUENCY/123,100},
516:   {FREQUENCY/117,100},
517:   {FREQUENCY/123,300},
518:   {FREQUENCY/123,100},
519:   {FREQUENCY/123,100},
520:   {FREQUENCY/123,100},
521:   {FREQUENCY/117,300},
522:   {FREQUENCY/117,100},
523:   {FREQUENCY/117,100},
524:   {FREQUENCY/104,100},
525:   {FREQUENCY/117,300},
526:   {FREQUENCY/117,100},
527:   {FREQUENCY/117,100},
528:   {FREQUENCY/117,100},
529:   {FREQUENCY/131,300},
530:   {FREQUENCY/131,100},
531:   {FREQUENCY/131,100},
532:   {FREQUENCY/123,100},
533:   {FREQUENCY/65,300},
534:   {FREQUENCY/65,100},
535:   {FREQUENCY/65,100},
536:   {FREQUENCY/65,100},
537:   {FREQUENCY/87,300},
538:   {FREQUENCY/87,300},
539:   {FREQUENCY/87,300},
540:   {FREQUENCY/98,150},
541:   {FREQUENCY/110,150}
542: };
543:
544: //---------------------Music_InitOC0---------------------
545: // arm output compare 0 for melody
546: // also enables timer to 43 ns period
```

```
547: // Input: none
548: // Output: none
549: void Music_InitOC0(void){
550:   TSCR1 = 0x80;   // Enable TCNT, 24MHz boot mode, 8MHz in run mode
551:   TSCR2 = 0x00;   // divide by 1 TCNT prescale, TOI disarm, sets period to 42.67 ns
552:   PACTL = 0;      // timer prescale used for TCNT
553:
554:   TIOS |= 0x01;   // activate TC0 as output compare
555:   TIE  |= 0x01;   // arm OC0
556:   TC0   = TCNT+50;// first interrupt right away
557: }
558:
559: //---------------------Music_InitOC1---------------------
560: // arm output compare 1 for harmony
561: // Input: none
562: // Output: none
563: void Music_InitOC1(void){
564:   TIOS |= 0x02;   // activate TC1 as output compare
565:   TIE  |= 0x02;   // arm OC1
566:   TC1   = TCNT+50;// first interrupt right away
567: }
568:
569: //---------------------Music_InitOC2---------------------
570: // arm output compare 2 for bass
571: // Input: none
572: // Output: none
573: void Music_InitOC2(void){
574:   TIOS |= 0x04;   // activate TC1 as output compare
575:   TIE  |= 0x04;   // arm OC1
576:   TC2   = TCNT+50;// first interrupt right away
577: }
578:
579: //---------------------Music_InitOC3---------------------
580: // arm output compare 3 for envelopes at 750 Hz
581: // Input: none
582: // Output: none
583: void Music_InitOC3(void){
584:   TIOS |= 0x08;   // activate TC1 as output compare
585:   TIE  |= 0x08;   // arm OC1
586:   TC3   = TCNT+50;// first interrupt right away
587: }
588:
589: // OC handler for melody
590: interrupt 8 void TC0Handler() {
591:   unsigned static char i = 0;
592:
593:   TFLG1 = 0x01;
594:
595:   // Checks if the note is a rest
596:   if(melody[note1].frequency) {
597:     // Outputs the proper sin value plus the other two lines' outputs to the DAC
598:     DAC_Out((SinWave[i%SIN] * envelope1) + output2 + output3);
599:     // Sets output1 to the proper sin value
600:     output1 = SinWave[i%SIN] * envelope1;
601:     i++;
602:     // Sets the next interrupt according to the frequency
603:     TC0 = TC0 + melody[note1].frequency;
604:   }
605:   else {
606:     // Sets output to zero
607:     output1 = 0;
608:     // Arbitrary next interrupt to check if note changed
609:     TC0 = TC0 + 480;
610:   }
611: }
612:
613: // OC handler for harmony
614: interrupt 9 void TC1Handler() {
615:   unsigned static char i = 0;
616:
617:   TFLG1 = 0x02;
618:
619:   if(harmony[note2].frequency) {
620:     // Outputs the proper sin value plus the other two lines' outputs to the DAC
621:     DAC_Out((SinWave[i%SIN] * envelope2) + output1 + output3);
622:     // Sets output1 to the proper sin value
623:     output2 = SinWave[i%SIN] * envelope2;
624:     i++;
```

```
625:     // Sets the next interrupt according to the frequency
626:     TC1 = TC1 + harmony[note2].frequency;
627:   }
628:   else {
629:     // Sets output to zero
630:     output2 = 0;
631:     // Arbitrary next interrupt to check if note changed
632:     TC1 = TC1 + 480;
633:   }
634:
635: }
636:
637: interrupt 10 void TC2Handler() {
638:   unsigned static char i = 0;
639:
640:   TFLG1 = 0x04;
641:
642:   if(note3 < BASS && bass[note3].frequency) {
643:     // Outputs the proper sin value plus the other two lines' outputs to the DAC
644:     DAC_Out((SinWave[i%SIN] * envelope3) + output1 + output2);
645:     // Sets output1 to the proper sin value
646:     output3 = SinWave[i%SIN] * envelope3;
647:     i++;
648:     // Sets the next interrupt according to the frequency
649:     TC2 = TC2 + bass[note3].frequency;
650:   }
651:   else {
652:     // Sets output to zero
653:     output3 = 0;
654:     // Arbitrary next interrupt to check if note changed
655:     TC2 = TC2 + 480;
656:   }
657: }
658:
659: interrupt 11 void TC3Handler() {
660:   TFLG1 = 0x08;
661:
662:   // Counts number of interrupts triggered for each note
663:   interrupts1++;
664:   interrupts2++;
665:   interrupts3++;
666:
667:   // Checks if note is finished
668:   if(interrupts1 >= melody[note1].length) {
669:       // If reverse button is pushed, decrement note
670:       if(Switch_Data()&0x01) {
671:         note1--;
672:       }
673:       // Otherwise, increment note
674:       else {
675:         note1++;
676:         // If note is past the end of the song, repeat
677:         if(note1 >= MELODY) {
678:           note1 = MREPEAT;
679:         }
680:       }
681:       // Reset counter and envelope for next note
682:       interrupts1 = 0;
683:       envelope1 = 1;
684:   }
685:   // Set envelope for only 2/3 of note length
686:   else if(interrupts1 >= (melody[note1].length*2)/3) {
687:     envelope1 = 0;
688:   }
689:
690:   // Checks if note is finished
691:   if(interrupts2 >= harmony[note2].length) {
692:       // If reverse button is pushed, decrement note
693:       if(Switch_Data()&0x01) {
694:         note2--;
695:       }
696:       // Otherwise, increment note
697:       else {
698:         note2++;
699:         // If note is past the end of the song, repeat
700:         if(note2 >= HARMONY) {
701:           note2 = HREPEAT;
702:         }
```

```
703:        }
704:        // Reset counter and envelope for next note
705:        interrupts2 = 0;
706:        envelope2 = 1;
707:    }
708:    // Set envelope for only 2/3 of note length
709:    else if(interrupts2 >= (harmony[note2].length*2)/3) {
710:      envelope2 = 0;
711:    }
712:
713:    // Checks if note is finished
714:    if(interrupts3 >= bass[note3].length) {
715:        if(Switch_Data()&0x01) {
716:        // If reverse button is pushed, decrement note
717:          note3--;
718:        }
719:        // Otherwise, increment note
720:        else {
721:          note3++;
722:          // If note is past the end of the song, repeat
723:          if(note3 >= BASS) {
724:            note3 = BREPEAT;
725:          }
726:        }
727:        // Reset counter and envelope for next note
728:        interrupts3 = 0;
729:        envelope3 = 1;
730:    }
731:    // Set envelope for only 2/3 of note length
732:    else if(interrupts3 >= (bass[note3].length*2)/3) {
733:      envelope3 = 0;
734:    }
735:
736:    TC3 = TC3 + 32000;
737: }
738:
739: //---------------------Music_Restart----------------------
740: // Restarts music
741: // Input: none
742: // Output: none
743: void Music_Restart(void) {
744:    note1 = 0;
745:    note2 = 0;
746:    note3 = 0;
747:
748:    interrupts1 = 0;
749:    interrupts2 = 0;
750:    interrupts3 = 0;
751:
752:    envelope1 = 1;
753:    envelope2 = 1;
754:    envelope3 = 1;
755: }
756:
757:
```

```
 1: #include <hidef.h>      /* common defines and macros */
 2: #include <mc9s12dp512.h>     /* derivative information */
 3: #pragma LINK_INFO DERIVATIVE "mc9s12dp512"
 4:
 5: //---------------------Switch_Init---------------------
 6: // sets DDRP for switches
 7: // Input: none
 8: // Output: none
 9: void Switch_Init(void);
10:
11: //---------------------Switch_Data---------------------
12: // Input: none
13: // Output: Switch data
14: unsigned char Switch_Data(void);
```

```
 1: #include "switch.h"
 2:
 3: //---------------------Switch_Init---------------------
 4: // sets DDRP for switches
 5: // Input: none
 6: // Output: none
 7: void Switch_Init(void) {
 8:   DDRP &= ~0x07;
 9: }
10:
11: //---------------------Switch_Data---------------------
12: // Input: none
13: // Output: Switch data
14: unsigned char Switch_Data(void) {
15:   return (PTP&0x07);
16: }
```