```
 1:
 2:
 3: NAME        TES_Zekasi
 4: ;*****************************************************************************
 5: ;  Transcranial Electrical Stimulation Device Firmware
 6: ;  Constant Current Generation with a 9V compliance voltage,
 7: ;  with offset and a maximum 30Hz sine wave using PWM timer Timer_B.
 8: ;
 9: ;  Referring to Mike Mitchell's SLAA116 App note.
10: ;
11: ;  Description: This program demonstrates the usage of a PWM timer together
12: ;  with external filters to implement a DAC.  The program shows how to
13: ;  create a 250Hz sine wave, a 125Hz ramp, and a DC level with Timer_B.
14: ;  Timer_A could also be used in the same manner.  A sine table holds the
15: ;  sample values for the sinusoid.  To create the ramp, the PWM value is
16: ;  simply incremented. The DC level is created by storing charge on an
17: ;  RC network using a PWM output to provide the charge.  The value of the DC
18: ;  voltage directly corresponds to the duty cycele of the PWM signal.  After
19: ;  initialization, the CPU is put into LPM0.  It remains there until the
20: ;  CCIFG0 interrupt from Timer_B wakes it up.  In the Timer_B ISR, the next
21: ;  value for the sinusoid is loaded into CCR1 and the ramp value is incremented
22: ;  and loded into CCR2. Upon return form the ISR, the CPU goes back into LPM0.
23: ;
24: ;  Adnan Kurt
25: ;  makeLAB
26: ;  19Jan2010
27: ;
28: ;*****************************************************************************
29: #include    "MSP430X14x.H"                  ; Include Standard Defs
30:
31: Delta       EQU     250                     ; Delta = Target DCO/8192
32:                                             ; Target DCO frequency = 2.048MHz
33:                                             ; This value is used in the
34:                                             ; software FLL routine to
35:                                             ; calibrate the DCO frequency
36:                                             ; using the 32768Hz oscillator
37:                                             ; as a reference.  For more
38:                                             ; information on stabilizing
39:                                             ; the DCO or the FLL routine
40:                                             ; see the application report
41:                                             ; titled "Controlling the DCO
42:                                             ; frequency of the MSP430x11x"
43:                                             ; Literature number SLAA074
44: ;-----------------------------------------------------------------------------
45:             RSEG    DATA16_N
46: ;        ORG    0x200
47: Ali         DS      2
48: Ada         DS      2
49: Mete        DS      2
50: Nese        DS      2
51: Aras        DS      2
52: Deniz       DS      2
53: Bat         DS      2
54: JR          DS      2
55: ;-----------------------------------------------------------------------------
56:             RSEG    CODE
57: ;-----------------------------------------------------------------------------
58: ;   MSP_Code Generated with Excel
59: Sin_K1_Tab
60:             DW  127
61:             DW  151
62:             DW  175
63:             DW  196
64:             DW  216
65:             DW  231
66:             DW  243
67:             DW  251
68:             DW  254
69:             DW  253
70:             DW  247
71:             DW  236
72:             DW  222
73:             DW  204
74:             DW  183
75:             DW  160
76:             DW  136
```

```
 77:              DW  111
 78:              DW  88
 79:              DW  65
 80:              DW  45
 81:              DW  28
 82:              DW  14
 83:              DW  5
 84:              DW  1
 85:              DW  4
 86:              DW  13
 87:              DW  27
 88:              DW  43
 89:              DW  63
 90:              DW  85
 91:              DW  109
 92: ;Sin_Tab_K2
 93:              DW  119
 94:              DW  142
 95:              DW  164
 96:              DW  184
 97:              DW  202
 98:              DW  217
 99:              DW  228
100:              DW  235
101:              DW  238
102:              DW  237
103:              DW  231
104:              DW  221
105:              DW  208
106:              DW  191
107:              DW  171
108:              DW  150
109:              DW  127
110:              DW  104
111:              DW  82
112:              DW  61
113:              DW  42
114:              DW  26
115:              DW  13
116:              DW  5
117:              DW  1
118:              DW  4
119:              DW  13
120:              DW  25
121:              DW  41
122:              DW  59
123:              DW  80
124:              DW  102
125: ;Sin_Tab_K3
126:              DW  111
127:              DW  132
128:              DW  153
129:              DW  172
130:              DW  189
131:              DW  202
132:              DW  213
133:              DW  220
134:              DW  223
135:              DW  221
136:              DW  216
137:              DW  207
138:              DW  194
139:              DW  178
140:              DW  160
141:              DW  140
142:              DW  119
143:              DW  97
144:              DW  77
145:              DW  57
146:              DW  39
147:              DW  24
148:              DW  13
149:              DW  4
150:              DW  1
151:              DW  4
152:              DW  12
```

```
153:              DW  23
154:              DW  38
155:              DW  55
156:              DW  75
157:              DW  96
158: ;Sin_Tab_K4
159:              DW  103
160:              DW  123
161:              DW  142
162:              DW  160
163:              DW  175
164:              DW  188
165:              DW  198
166:              DW  204
167:              DW  207
168:              DW  205
169:              DW  200
170:              DW  192
171:              DW  180
172:              DW  165
173:              DW  149
174:              DW  130
175:              DW  110
176:              DW  90
177:              DW  71
178:              DW  53
179:              DW  37
180:              DW  23
181:              DW  12
182:              DW  4
183:              DW  1
184:              DW  4
185:              DW  11
186:              DW  21
187:              DW  35
188:              DW  51
189:              DW  69
190:              DW  89
191: ;Sin_Tab_K5
192:              DW  95
193:              DW  113
194:              DW  131
195:              DW  147
196:              DW  162
197:              DW  173
198:              DW  182
199:              DW  188
200:              DW  191
201:              DW  190
202:              DW  185
203:              DW  177
204:              DW  166
205:              DW  153
206:              DW  137
207:              DW  120
208:              DW  102
209:              DW  83
210:              DW  66
211:              DW  49
212:              DW  34
213:              DW  21
214:              DW  11
215:              DW  4
216:              DW  1
217:              DW  3
218:              DW  10
219:              DW  20
220:              DW  32
221:              DW  47
222:              DW  64
223:              DW  82
224: ;Sin_Tab_K6
225:              DW  87
226:              DW  104
227:              DW  120
228:              DW  135
```

```
229:              DW  148
230:              DW  159
231:              DW  167
232:              DW  173
233:              DW  175
234:              DW  174
235:              DW  170
236:              DW  162
237:              DW  152
238:              DW  140
239:              DW  126
240:              DW  110
241:              DW  93
242:              DW  76
243:              DW  60
244:              DW  45
245:              DW  31
246:              DW  19
247:              DW  10
248:              DW  3
249:              DW  1
250:              DW  3
251:              DW  9
252:              DW  18
253:              DW  30
254:              DW  43
255:              DW  59
256:              DW  75
257: ;Sin_Tab_K7
258:              DW  79
259:              DW  94
260:              DW  109
261:              DW  123
262:              DW  135
263:              DW  144
264:              DW  152
265:              DW  157
266:              DW  159
267:              DW  158
268:              DW  154
269:              DW  147
270:              DW  138
271:              DW  127
272:              DW  114
273:              DW  100
274:              DW  85
275:              DW  69
276:              DW  55
277:              DW  41
278:              DW  28
279:              DW  17
280:              DW  9
281:              DW  3
282:              DW  1
283:              DW  3
284:              DW  8
285:              DW  16
286:              DW  27
287:              DW  39
288:              DW  53
289:              DW  68
290: ;Sin_Tab_K8
291:              DW  71
292:              DW  85
293:              DW  98
294:              DW  110
295:              DW  121
296:              DW  130
297:              DW  137
298:              DW  141
299:              DW  143
300:              DW  142
301:              DW  139
302:              DW  133
303:              DW  125
304:              DW  114
```

```
305:              DW  103
306:              DW  90
307:              DW  76
308:              DW  62
309:              DW  49
310:              DW  36
311:              DW  25
312:              DW  15
313:              DW  8
314:              DW  3
315:              DW  1
316:              DW  2
317:              DW  7
318:              DW  15
319:              DW  24
320:              DW  35
321:              DW  48
322:              DW  61
323: ;Sin_Tab_K9
324:              DW  63
325:              DW  75
326:              DW  87
327:              DW  98
328:              DW  108
329:              DW  115
330:              DW  121
331:              DW  125
332:              DW  127
333:              DW  126
334:              DW  123
335:              DW  118
336:              DW  111
337:              DW  102
338:              DW  91
339:              DW  80
340:              DW  68
341:              DW  55
342:              DW  44
343:              DW  32
344:              DW  22
345:              DW  14
346:              DW  7
347:              DW  2
348:              DW  1
349:              DW  2
350:              DW  6
351:              DW  13
352:              DW  21
353:              DW  31
354:              DW  42
355:              DW  54
356: ;Sin_Tab_K10
357:              DW  55
358:              DW  66
359:              DW  76
360:              DW  86
361:              DW  94
362:              DW  101
363:              DW  106
364:              DW  110
365:              DW  111
366:              DW  110
367:              DW  108
368:              DW  103
369:              DW  97
370:              DW  89
371:              DW  80
372:              DW  70
373:              DW  59
374:              DW  48
375:              DW  38
376:              DW  28
377:              DW  19
378:              DW  12
379:              DW  6
380:              DW  2
```

```
381:              DW  1
382:              DW  2
383:              DW  6
384:              DW  11
385:              DW  19
386:              DW  27
387:              DW  37
388:              DW  48
389: ;Sin_Tab_K11
390:              DW  47
391:              DW  56
392:              DW  65
393:              DW  73
394:              DW  81
395:              DW  86
396:              DW  91
397:              DW  94
398:              DW  95
399:              DW  95
400:              DW  92
401:              DW  88
402:              DW  83
403:              DW  76
404:              DW  68
405:              DW  60
406:              DW  51
407:              DW  41
408:              DW  33
409:              DW  24
410:              DW  17
411:              DW  10
412:              DW  5
413:              DW  2
414:              DW  1
415:              DW  1
416:              DW  5
417:              DW  10
418:              DW  16
419:              DW  23
420:              DW  32
421:              DW  41
422: ;Sin_Tab_K12
423:              DW  39
424:              DW  47
425:              DW  54
426:              DW  61
427:              DW  67
428:              DW  72
429:              DW  76
430:              DW  78
431:              DW  79
432:              DW  79
433:              DW  77
434:              DW  73
435:              DW  69
436:              DW  63
437:              DW  57
438:              DW  50
439:              DW  42
440:              DW  34
441:              DW  27
442:              DW  20
443:              DW  14
444:              DW  8
445:              DW  4
446:              DW  1
447:              DW  1
448:              DW  1
449:              DW  4
450:              DW  8
451:              DW  13
452:              DW  19
453:              DW  26
454:              DW  34
455: ;Sin_Tab_K13
456:              DW  31
```

```
457:              DW  37
458:              DW  43
459:              DW  49
460:              DW  54
461:              DW  57
462:              DW  60
463:              DW  62
464:              DW  63
465:              DW  63
466:              DW  61
467:              DW  59
468:              DW  55
469:              DW  51
470:              DW  45
471:              DW  40
472:              DW  34
473:              DW  27
474:              DW  22
475:              DW  16
476:              DW  11
477:              DW  7
478:              DW  3
479:              DW  1
480:              DW  1
481:              DW  1
482:              DW  3
483:              DW  6
484:              DW  10
485:              DW  15
486:              DW  21
487:              DW  27
488: ;Sin_Tab_K14
489:              DW  23
490:              DW  28
491:              DW  32
492:              DW  36
493:              DW  40
494:              DW  43
495:              DW  45
496:              DW  47
497:              DW  47
498:              DW  47
499:              DW  46
500:              DW  44
501:              DW  41
502:              DW  38
503:              DW  34
504:              DW  30
505:              DW  25
506:              DW  20
507:              DW  16
508:              DW  12
509:              DW  8
510:              DW  5
511:              DW  2
512:              DW  1
513:              DW  1
514:              DW  1
515:              DW  2
516:              DW  5
517:              DW  8
518:              DW  11
519:              DW  16
520:              DW  20
521: ;Sin_Tab_K15
522:              DW  15
523:              DW  18
524:              DW  21
525:              DW  24
526:              DW  27
527:              DW  28
528:              DW  30
529:              DW  31
530:              DW  31
531:              DW  31
532:              DW  30
```

```
533:              DW  29
534:              DW  27
535:              DW  25
536:              DW  22
537:              DW  20
538:              DW  17
539:              DW  13
540:              DW  11
541:              DW  8
542:              DW  5
543:              DW  3
544:              DW  1
545:              DW  1
546:              DW  1
547:              DW  1
548:              DW  1
549:              DW  3
550:              DW  5
551:              DW  7
552:              DW  10
553:              DW  13
554: ;Sin_Tab_K16
555:              DW  6
556:              DW  7
557:              DW  8
558:              DW  10
559:              DW  10
560:              DW  11
561:              DW  12
562:              DW  12
563:              DW  12
564:              DW  12
565:              DW  12
566:              DW  12
567:              DW  11
568:              DW  10
569:              DW  9
570:              DW  8
571:              DW  6
572:              DW  5
573:              DW  4
574:              DW  3
575:              DW  2
576:              DW  1
577:              DW  1
578:              DW  1
579:              DW  1
580:              DW  1
581:              DW  1
582:              DW  1
583:              DW  2
584:              DW  3
585:              DW  4
586:              DW  5
587:
588: ;       Zero causes glitches, Really!  ak.
589: ;   END DATA
590:
591: Sine_Tab    DW      255                     ; Sine Table.  These are the count
592:              DW      254                     ; values in decimal that will
593:              DW      246                     ; go into TBCCR1 to change the
594:              DW      234                     ; PWM duty cycle.
595:              DW      219                     ; Must use words instead of bytes
596:              DW      199                     ; because must move words into
597:              DW      177                     ; TB registers.
598:              DW      153                     ; Don't use a '0' as a sample value
599:              DW      128                     ; The timer will glitch.
600:              DW      103
601:              DW      79
602:              DW      57
603:              DW      37
604:              DW      22
605:              DW      10
606:              DW      2
607:              DW      1
608:              DW      2
```

```
609:             DW      10
610:             DW      22
611:             DW      37
612:             DW      57
613:             DW      79
614:             DW      103
615:             DW      128
616:             DW      153
617:             DW      177
618:             DW      199
619:             DW      219
620:             DW      234
621:             DW      246
622:             DW      255
623:
624: ;-----------------   Code Starts Here   -------------------------------------
625: RESET       mov     #09FEh,SP               ; Initialize stackpointer
626:
627: StopWDT     mov     #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
628: ;-------------------------------------------------------------------------
629: ;          Interface Descriptions
630: gLED   EQU   BIT7                        ; P3.7
631: rLED   EQU   BIT6                        ; P3.6
632: PBS    EQU   BIT6                        ; P6.6
633: BuzP   EQU   BIT2                        ; P1.2
634: BuzN   EQU   BIT1                        ; P1.1
635: LupSa  EQU   20000
636:                                         ; Battery Sampler    p6.7
637:         bis.b   #gLED|rLED ,&P3DIR       ; Set P1 gLED and rLED pin to output
638:         bic.b   0x00, &P3OUT
639:         bic.b   #gLED|rLED ,&P3OUT       ; LEDs off
640:         bis.b   #BuzP|BuzN ,&P1DIR       ; Set P1 Buzzer pins to output
641:         bic.b   0x00, &P1OUT
642:         bic.b   #BuzP|BuzN ,&P1OUT       ; Buzzer off
643:         bis.b   #~PBS ,&P6DIR            ; Set P6 PushButtun pin to Input
644:         bic.b   0x00, &P6OUT
645: ;-------------------------------------------------------------------------
646:
647: ;------------------------------ ADC Init ---------------------------------
648:             bis.b   #BIT3+BIT4+BIT5+BIT7,&P6SEL ; Enable A/D channel inputs
649:                                         ;
650: SetupADC12  mov     #REFON+REF2_5V+ADC12ON+MSC+SHT0_8,&ADC12CTL0
651:                                         ; Turn on ADC12, set MSC
652:             mov     #SHP+CONSEQ_1,&ADC12CTL1; Use samp. timer, single sequence
653:             bis.b   #SREF_1+INCH_3,&ADC12MCTL0     ; Vr+=Vref+, channel=O
654:             bis.b   #SREF_1+INCH_4,&ADC12MCTL1     ; Vr+=Vref+, channel=F
655:             bis.b   #SREF_1+INCH_5,&ADC12MCTL2     ; Vr+=Vref+, channel=A
656:             bis.b   #SREF_1+INCH_7+EOS,&ADC12MCTL3 ; Vr+=Vref+, channel=Batt
657:                                                    ; end seq.
658:             mov     #03600h,R7              ; Delay for needed ref start-up.

659: L$1         dec     R7                      ; See datasheet for details.
660:             jnz     L$1                     ;
661: ;           mov     #BIT3,&ADC12IE          ; Enable ADC12IFG.0 for ADC12MEM0
662:             bis     #ENC,&ADC12CTL0         ; Enable conversions
663:
664: SetupP4     bis.b   #00Eh,&P4SEL            ; Select TB1, TB2, TB3 instead of
665:             bis.b   #00Eh,&P4DIR            ; P4.x, and set as outputs
666:             bic.b   #00Eh,&P4DIR            ; P4.x, and set as outputs/ TurnOff
667:
668: SetupBC     mov.b   #0A6h,&BCSCTL1          ; ACLK is divided by 4. RSEL=6,
669:                                             ; no division for MCLK or SMCLK,
670:                                             ; DCO sources MCLK and SMCLK.
671:                                             ; XT2 is off.
672:                                             ; NOTE: To determine the value of
673:                                             ; Rsel for a desired DCO frequency,
674:                                             ; refer to the DCO table in the
675:                                             ; datasheet.
676:
677:             call    #Delay                  ; Delay for crystal stabilization.
678:                                             ; Need to put a delay here because
679:                                             ; the 32768Hz crystal is used as
680:                                             ; a reference to stabilize the DCO
681:                                             ; frequency.  Therefore, the 32768
682:                                             ; crystal needs to be stable.
683:
```

```
684:             call    #SW_FLL                 ; Call the routine to Stabilize
685:                                              ; the DCO clock.
686:
687:             call    #TB_SETUP               ; Setup Timer_B for PWM generation
688:
689:             clr     R15                     ; R15 and R14 used as pointers
690:             clr     R14                     ; to the sine table and to hold the
691:                                              ; ramp value after the DCO is
692:                                              ; stabilized
693:             clr     R11                     ; Init temporary delay
694:             mov     #0FFh,R11               ; This should set the period
695:                                              ;
696: ;           eint                            ; Enable interrupts
697: ;-------------------------------------------------------------------------
698: ; TES Loop
699: ;-------------------------------------------------------------------------
700: ; Iteration value for delay loop, and inner moredelay loop
701:
702: SubDo      EQU  1024                       ; Value for TES duration subunit
703: DelayLoops EQU  3000
704: TES_Dura   EQU  621                        ; Duration of the Stimulation Period
705:                                            ; Loop constant under these conditions
706:                                            ; is measured to be 1.30 counts/ sec
707:                                            ; 391 counts is 3minutes 9sec
708:                                            ; measured. 621 counts expected
709:                                            ; to give 5 minutes.
710: V_Crit     EQU  1555                       ; 8.77V battery reads 1965 counts
711:                                            ; Battery+ Schottky diode+ 330K + 56K
712:                                            ; Voltage read over 56K. 7 volts
713:                                            ; chosen to be a critical value.
714:
715: ; Morse Code Symbol periods in about 0.1s quantum
716: ; LETTER is the period between letters, ENDTX terminates the message
717:
718: DUB     EQU     44
719: DIP     EQU     6
720: SOSPA   EQU     99
721: PA      EQU     7
722: DOT     EQU     11
723: DASH    EQU     33
724: SPACE   EQU     22
725: LETTER  EQU     2                          ; This is Buggy. Gives an LED flash
726: ENDTX   EQU     0xFF
727: ;-------------------------------------------------------------------------
728: ; Loop Timing Setup
729: ;-------------------------------------------------------------------------
730:             mov     #4, &CCR0               ; ACLK divided by 1. 8 gives 455 Hz.
731:                                              ; 1 gives 2048 Hz.
732:             mov     #12, &Bat               ; Initial Bat value
733:             mov     #0, &Ali
734:             mov     #0, &Ada
735:             mov     #0, &Mete
736:             mov     #0, &Nese
737: Setup_LT    mov     #TASSEL_1+MC_1+ID_0+TACLR, &TACTL
738:                                              ; Start Timer_A, up to CCR0 mode,
739:                                              ; divide by 1 clock, clock from ACLK,
740:                                              ; clear timer, 32kHz xtal
741: Graceful_Start:
742:         bic.b   #00Eh,&P4DIR                ; P4.x, and set as outputs/ TurnOff
743:         mov     #2000, &Aras                ; Bip Sound duration
744:         mov     #4, &Deniz                  ; Bip frequency
745:         call    #Bip
746: Morse   mov     &Mete, R4
747:         jmp     Mes_Test                    ; Jump to test
748: Mes_Loop:
749:         bis.b   #gLED , &P3OUT              ; LEDs on
750:         mov     &Mete, R4
751:         mov.b   Start(R4), &Ali             ; Load duration of delay as parameter
752:         call    #DelayQuanta                ; Call Subroutine: Don't Forget #
753:         bic.b   #gLED , &P3OUT              ; LEDs off
754:         mov.w   #SPACE , &Ali               ; Load Duration of Space delay
755:         call    #DelayQuanta                ; Call Subroutine: Don't Forget #
756:         inc.w   &Mete                       ; Next symbol to send
757:         mov     &Mete, R4
758: Mes_Test:
759:         cmp.b   #ENDTX, Start(R4)           ; End of Message?
```

```
760:         jne     Mes_Loop                ; Repeat
761:         bic.b   #gLED , &P3OUT          ; LEDs off
762:         mov.w   #SPACE, &Ali            ; Load Duration of Space delay
763:         call    #DelayQuanta
764:         mov.w   #SPACE, &Ali            ; Load Duration of Space delay
765:         call    #DelayQuanta
766:         call    #Bip
767:         eint                            ; Enable interrupts
768: ;-------------------------------------------------------------------------
769: ; Control States
770: ;-------------------------------------------------------------------------
771:
772:         mov     #0, &Mete
773:         mov     #0, &Nese
774:         mov     #0, &Aras
775:         mov     #0, &Deniz
776:         mov     #0, &JR
777: InfLoop:
778: BattMon bis     #ADC12SC, &ADC12CTL0    ; Start conversions
779:         mov     &ADC12MEM3, R8          ; Move A7 result -B, IFG is reset
780:         cmp     &V_Crit, R8             ; Test critical voltage
781:         jn      SOSi                    ; If VBatt<VCrit then SOSi
782:         jmp     Conti                   ; Else continue
783: SOSi    dec.b   &Bat                    ; Decrement Bat value, to signal
784:         jz      Morsy                   ; SOS at increasing frequencies
785:         jmp     Conti                   ; Continue, till Bat exhausted
786: Morsy   mov.b   #4, &Bat                ; Load new Bat value
787:         mov.b   #44, &Aras              ; Load bip duration with cons value
788: ;        sub.b   #0xff, &Aras            ; Make it an increasing function
789: ;Morsy   mov.b   R8, &Bat               ; Load new Bat value
790:  ;       mov.b   R8, &Aras               ; Load bip duration with bat value
791:   ;      sub.b   #0xff, &Aras            ; Make it an increasing function
792:         mov     #1, &Deniz              ; Keep frequency a high pitch
793:         call    #Bip
794:         call    #SOS
795: Conti   call    #HeartRate              ; Normal background operation with HR
796: ;       cmp     #2, &JR                 ; Button on for 2 cycles?
797: ;       jge     ThinkOnce               ; Button pressed long enough to decide
798:         bit.b   #PBS, &P6IN             ; Test PBS
799:         jz      DeBoun                  ; Jump if zero to DeBoun
800:         jmp     InfLoop                 ; Loop to check PBS
801:                                         ; Zero on button press!
802:                                         ; Loop Forever
803: DeBoun  mov     #1000, &JR
804: DeLup   dec     &JR                     ; Increment Nese for debounce and aim
805:         jz      CheckAgain
806:         jmp     DeLup
807: CheckAgain:
808:         bit.b   #PBS, &P6IN             ; Test PBS
809:         jz      SureFall                ; Test for fall
810:         jmp     InfLoop                 ; Loop again
811: SureFall:
812:         bit.b   #PBS, &P6IN             ; Test PBS
813:         jnz     ThinkOnce
814:         jmp     SureFall
815: ThinkOnce:
816:         mov     #0, &JR                 ; Initialize debouncer
817:         mov     #0, &Mete               ; Initialize loop counter
818:         jmp     TESLoop                 ; On button press do TES
819:
820: ; The remarked code was to test more button presses and to go for different
821: ; device behavior. However, it complicates the user intervention,
822: ; so they are removed.
823: ;-------------------------------------------------------------------------
824: ; Remnant Code
825: ;-------------------------------------------------------------------------
826: ;OneMinute:
827:      ;   inc     &Mete
828:     ;    cmp     #DelayLoops, &Mete      ; Wait for 3000 counts
829:    ;     jnz     OneMinute
830:   ;      bit.b   #PBS ,&P6IN             ; Test PBS
831:  ;       jz      InfLoop                 ; Pressed PB too long enough, rerun
832: ;        mov     #0, &Mete               ; Initialize loop counter
833: ;ThinkAgain:
834:  ;       inc     &Mete
835: ;        cmp     #DelayLoops, &Mete      ; Wait for 3000 counts
```

```
836:     ;    jnz     ThinkAgain
837:    ;     bit.b   #PBS ,&P6IN              ; Test PBS
838:   ;      jnz     TESLoop                  ; Basic, 5min stimulus
839:  ;       mov     #0, &Mete                ; Initialize loop counter
840: ;        jmp     Nirvana                  ; Quick double click, awarded with
841: ;                                         ; VeryLongDuration
842: ;Nirvana:
843:       ;  inc     &Mete
844:      ;   cmp     #DelayLoops, &Mete       ; Wait for 3000 counts
845:     ;    jnz     Nirvana
846:    ;     bit.b   #PBS ,&P6IN              ; Test PBS
847:   ;      jz      InfLoop                  ; Reset the process
848:  ;       mov     #10000, &TES_Dura        ; Promised duration
849: ;        jmp     TESLoop                  ; with VeryLongDuration grace period
850: ;-------------------------------------------------------------------------
851:
852: TESLoop:                                 ; Should get Duration Value
853:                                          ; It is TES_Dura, 1000d initially
854:         mov     #2, &Deniz
855:         mov     #SOSPA, &Aras
856:         call    #Bip
857:         bis     #MC0, &TBCTL             ; Start timer_B in up mode
858:         bis.b   #00Eh, &P4DIR            ; P4.x, and set as outputs/ TurnOn
859:         mov     &SubDo, &CCR0            ; Timer A counter
860:         mov     #0, &Nese
861: Dura    cmp     &TES_Dura, &Nese
862:         jz      Donna                    ; Stimulation Done
863:         inc     &Nese
864:         bit.b   #PBS, &P6IN              ; Test rSW TES_Dura
865:         jz      EndSes                   ; On Button Press during stimulation
866:                                          ; ends the session immediately
867:         bic     #TAIFG, &TACTL           ; Clear overflow flag
868: Do      bit     #TAIFG, &TACTL           ; Wait for overflow
869:         jz      Do
870:         jmp     Dura
871: Donna   jmp     EndSes
872: ;       dint                             ; Disable Interrupts
873:
874: EndSes:
875:         bic     #MC0, &TBCTL             ; Stop timer_B in up mode
876:         bic.b   #00Eh, &P4DIR            ; P4.x, and set as outputs/ TurnOff
877:         mov     #2, &Deniz
878:         mov     #SOSPA, &Aras
879:         call    #Bip
880:         mov     #8, &Deniz
881:         mov     #SOSPA, &Aras
882:         call    #Bip
883:         mov     #0, &Mete
884:         mov     #0, &Nese
885:         mov     #0, &Aras
886:         mov     #0, &Deniz
887:         jmp     InfLoop                  ; Jump if not zero to main
888:
889:
890: // Subroutine for Delay Quantum, total R12*0.1 s
891: // Parameter is passed through R12 and destroyed. R4 used for loop counter.
892: // To take care of R12=0 condition, test is done first.
893:
894: DelayQuanta:
895:         jmp     LoopTest
896: OuterLoop:
897:         mov.w   #DelayLoops, &Ada        ; Initialize Loop Counter
898: DelayLoop:                               ; Clock Cycles in [ brackets]
899:         dec.w   &Ada                     ; Decrement Loop Counter [1]
900:         jnz     DelayLoop                ; Repeat if not zero [2]
901:         dec.w   &Ali                     ; Decrement number of quantum
902: LoopTest:
903:         cmp.w   #0, &Ali                 ; Finished all quanta?
904:         jnz     OuterLoop                ; Repeat then
905:         ret                              ; Return to mom -caller.
906:
907: ;            bis     #LPM0,SR            ; Put CPU to sleep.
908:                                          ; This is the end of the program
909:                                          ; except for handling the CCIFG0
910:                                          ; interrupt, which is where the
911:                                          ; PWM values are updated.
```

```
912:
913: ;-------------------------------------------------------------------------------
914: ;           CODE ENDS HERE
915: ;-------------------------------------------------------------------------------
916:
917: ;-------------------------------------------------------------------------------
918: HeartRate;      LED Heart Rate
919: ;-------------------------------------------------------------------------------
920:         bic.b   #0,  &P3OUT              ; LEDs on
921:         bis.b   #rLED,  &P3OUT           ; LEDs on
922:         mov.b   #DUB, &Ali               ; Load duration of delay as parameter
923:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
924:         bic.b   #rLED,  &P3OUT           ; LEDs off
925:         mov.w   #SPACE , &Ali            ; Load Duration of Space delay
926:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
927:         bis.b   #rLED,  &P3OUT           ; LEDs on
928:         mov.b   #DIP, &Ali    ; Load duration of delay as parameter
929:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
930:         bic.b   #rLED,  &P3OUT           ; LEDs off
931:         mov.w   #PA , &Ali               ; Load Duration of Space delay
932:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
933:         bis.b   #rLED,  &P3OUT           ; LEDs on
934:         mov.b   #DIP, &Ali    ; Load duration of delay as parameter
935:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
936:         bic.b   #rLED,  &P3OUT           ; LEDs off
937:         mov.w   #SOSPA , &Ali            ; Load Duration of Space delay
938:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
939:         bic.b   #rLED,  &P3OUT           ; LEDs off
940:         ret
941: ;-------------------------------------------------------------------------------
942: SOS;       LED SOS -Battery Warning
943: ;-------------------------------------------------------------------------------
944:         bic.b   #0,  &P3OUT              ; LEDs on
945:         bis.b   #gLED,  &P3OUT           ; LEDs on
946:         mov.b   #DOT, &Ali               ; Load duration of delay as parameter
947:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
948:         bic.b   #gLED,  &P3OUT           ; LEDs off
949:         mov.w   #PA , &Ali               ; Load Duration of Space delay
950:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
951:         bis.b   #gLED,  &P3OUT           ; LEDs on
952:         mov.b   #DOT, &Ali    ; Load duration of delay as parameter
953:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
954:         bic.b   #gLED,  &P3OUT           ; LEDs off
955:         mov.w   #PA , &Ali               ; Load Duration of Space delay
956:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
957:         bis.b   #gLED,  &P3OUT           ; LEDs on
958:         mov.b   #DOT, &Ali    ; Load duration of delay as parameter
959:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
960:         bic.b   #gLED,  &P3OUT           ; LEDs off
961:         mov.w   #PA , &Ali               ; Load Duration of Space delay
962:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
963:         bis.b   #gLED,  &P3OUT           ; LEDs on
964:         mov.b   #DASH, &Ali              ; Load duration of delay as parameter
965:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
966:         bic.b   #gLED,  &P3OUT           ; LEDs off
967:         mov.w   #PA , &Ali               ; Load Duration of Space delay
968:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
969:         bis.b   #gLED,  &P3OUT           ; LEDs on
970:         mov.b   #DASH, &Ali    ; Load duration of delay as parameter
971:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
972:         bic.b   #gLED,  &P3OUT           ; LEDs off
973:         mov.w   #PA , &Ali               ; Load Duration of Space delay
974:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
975:         bis.b   #gLED,  &P3OUT           ; LEDs on
976:         mov.b   #DOT, &Ali               ; Load duration of delay as parameter
977:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
978:         bic.b   #gLED,  &P3OUT           ; LEDs off
979:         mov.w   #PA , &Ali               ; Load Duration of Space delay
980:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
981:         bis.b   #gLED,  &P3OUT           ; LEDs on
982:         mov.b   #DOT, &Ali    ; Load duration of delay as parameter
983:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
984:         bic.b   #gLED,  &P3OUT           ; LEDs off
985:         mov.w   #PA , &Ali               ; Load Duration of Space delay
986:         call    #DelayQuanta             ; Call Subroutine: Don't Forget #
987:         bis.b   #gLED,  &P3OUT           ; LEDs on
```

```
 988:         mov.b   #DOT, &Ali    ; Load duration of delay as parameter
 989:         call    #DelayQuanta              ; Call Subroutine: Don't Forget #
 990:         bic.b   #gLED,  &P3OUT            ; LEDs off
 991:         mov.w   #PA , &Ali                ; Load Duration of Space delay
 992:         call    #DelayQuanta              ; Call Subroutine: Don't Forget #
 993:         bic.b   #gLED,  &P3OUT            ; LEDs off
 994:         ret
 995: ;-------------------------------------------------------------------------------
 996: Bip;      Bip Sound Aras for duration, Deniz for frequency
 997: ;-------------------------------------------------------------------------------
 998:         clr.w   &Mete                     ; Init counter
 999:         mov.b   #BuzP|~BuzN, &P1OUT       ; Init Buzzer Pins parity
1000: ;       mov.b   #BuzP|BuzN, &P1OUT        ; Init No Sound
1001:         mov     &Deniz, &CCR0
1002:         clr     &Nese
1003: Zil     cmp     &Aras, &Nese
1004:         jz      Don
1005:         xor.b   #BuzP|BuzN, &P1OUT        ; Toggle Buzzer Pins
1006:         inc     &Nese
1007:         bic     #TAIFG, &TACTL            ; Clear overflow flag
1008: Done    bit     #TAIFG, &TACTL            ; Wait for overflow
1009:         jz      Done
1010:         jmp     Zil
1011: Don     ret
1012:
1013: ;-------------------------------------------------------------------------------
1014: Delay;      Software delay for crystal stabilization
1015: ;-------------------------------------------------------------------------------
1016:             mov     #0004h, R15
1017: L1          mov     #0FFFFh, R14            ; This should ideally be about a sec.
1018: L2          dec     R14                     ;
1019:             jnz     L2                      ;
1020:                                             ;
1021:             xor.b   #~rLED|gLED,  &P3OUT    ; LEDs on
1022:             dec     R15                     ;
1023:             jnz     L1                      ;
1024:             ret                             ;
1025:                                             ;
1026: ;-------------------------------------------------------------------------------
1027: SW_FLL;    Subroutine: Stabilizes DCO frequency.
1028:             ; This routine uses the 32768Hz crystal oscillator as a reference
1029:             ; frequency to stabilize and trim the DCO oscillator to the desired
1030:             ; frequency of 2.048MHz.  This is only required in applications that
1031:             ; need a specific DCO frequency and for MSP430 devices that do not
1032:             ; have an FLL module.  See the MSP430x3xx and MSP430x1xx Family
1033:             ; User's Guides (literature numbers SLAU012 and SLAU049 repsecitvely)
1034:             ; for more information on the clock systems employed on MSP430 devices
1035:             ;
1036:             ; The routine works by counting how many DCO clock cycles are inside
1037:             ; of one ACLK cycle (actually 1/4 ACLK cycle because ACLK is divided
1038:             ; by 4).  Timer_A is used to determine the number of DCO clocks and
1039:             ; this value is then compared to the target value (Delta).  If the
1040:             ; number is too high, the DCO is decremented.  If the number is too
1041:             ; low, the DCO is incremented. The comparison is then made again.
1042:             ; This process is repeated until the target value is reached.  When
1043:             ; the target value is obtained, the DCO is oscillating at the desired
1044:             ; frequency.  See the application report "Controlling the DCO
1045:             ; Frequency of the MSP430x11x devices", literature number SLAA074,
1046:             ; for more application information related to controlling the DCO.
1047:             ;
1048:             ; This routine is run only once in this example, but in an
1049:             ; application it would likely need to be run on a periodic
1050:             ; basis to make sure the DCO remained calibrated.
1051: ;-------------------------------------------------------------------------------
1052:
1053:             clr     R15                     ;
1054: Setup_TA    mov     #TASSEL1+TACLR,&TACTL   ; SMCLK clocks TA
1055: Setup_CC2   mov     #CCIS0+CM0+CAP,&CCTL2   ; Define CCR2,CAP,ACLK
1056:             bis     #MC1,&TACTL             ; Start timer_A: Continous Mode
1057: Test_DCO    bit     #CCIFG,&CCTL2           ; Test capture flag
1058:             jz      Test_DCO                ;
1059:             bic     #CCIFG,&CCTL2           ; Clear capture flag
1060:                                             ;
1061: AdjDCO      mov     &CCR2,R14               ; R14 = captured SMCLK
1062:             sub     R15,R14                 ; R14 = capture difference
1063:             mov     &CCR2,R15               ; R15 = captured SMCLK
```

```
1064:             cmp     #Delta,R14              ; Delta = SMCLK/(32768/4)
1065:             jlo     IncDCO                  ;
1066:             jeq     DoneFLL                 ;
1067: DecDCO      dec.b   &DCOCTL                 ;
1068:             jmp     Test_DCO                ;
1069: IncDCO      inc.b   &DCOCTL                 ;
1070:             jmp     Test_DCO                ;
1071: DoneFLL     clr     &CCTL2                  ; Stop CCR2
1072:             clr     &TACTL                  ; Stop timer_A
1073:             ret                             ; Return from subroutine
1074: ;-------------------------------------------------------------------------
1075: TB_SETUP;    Subroutine: Setup Timer_B for PWM generation
1076: ;-------------------------------------------------------------------------
1077:             mov     #TBSSEL1+TBCLR,&TBCTL   ; SMCLK clocks TB.
1078:             mov     #CCIE,&TBCCTL0          ; Set CCR0 in compare mode, enable
1079:                                             ; it's interrupt
1080:             mov     #0FFh,&TBCCR0           ; Put 255d in CCR0.  This will set
1081:                                             ; the period of the PWM output to
1082:                                             ; 256 counts(8-bits).  This gives
1083:                                             ; an 8-bit DAC.
1084:             mov     #02E0h,&TBCCTL1         ; Set CCRx in compare mode, disable
1085:             mov     #02E0h,&TBCCTL2         ; interrupt, set outmode to '7' which
1086:             mov     #02E0h,&TBCCTL3         ; is reset/set.  EQU0 sets the output
1087:                                             ; EQU1 will reset it. Set the load
1088:                                             ; condition for the compare latch
1089:                                             ; to be when the counter counts to
1090:                                             ; 0.
1091:             mov     #Sine_Tab,&TBCCR1       ; Load first sample value into CCR1
1092:             mov     #01h,R14                ; Load inital ramp value into R14.
1093:             mov     #001h,&TBCCR3           ; This is for the DC value.  It will
1094:                                             ; result in a voltage of approximately
1095:                                             ; 2/3 Vcc when #0AAh. It is 2/3 of
1096:                                             ; #0FFh.
1097: ;           bis     #MC0,&TBCTL             ; Start timer_B in up mode
1098:
1099:             ret
1100:
1101: ;-------------------------------------------------------------------------
1102: ; Division with Hamacher 16 bit integer division
1103: ;-------------------------------------------------------------------------
1104:
1105: div_hamacher:
1106:
1107:             ;   R12 has dividend
1108:             ;   R7 has divisor
1109:
1110:             mov.w     #16,R14
1111:             clr.w     R9
1112: start:      rla.w     R12
1113:             rlc.w     R9
1114:             bis.w     #1, R12
1115:             sub.w     R7,R9
1116:             jge       loc1
1117:             add.w     R7,R9
1118:             bic.w     #1, R12
1119: loc1:       dec.w     R14
1120:             cmp.w     #0,R14
1121:             jnz       start
1122:             ret
1123:
1124: ;-------------------------------------------------------------------------
1125: TB_ISR;    Timer_B ISR: changes the value in the CCR1 and CCR2 registers to
1126: ;          vary the PWM for the sinusoid and the ramp.
1127: ;-------------------------------------------------------------------------
1128:
1129: ;--------------------------- ADC Conversions -------------------------------
1130: ;
1131: ;*************************************************************************
1132: ;   TES ADC Sequence of Non-repeated Conversions
1133: ;
1134: ;   A single sequence of conversions is performed - one conversion each on
1135: ;   channels A3, A4, A5, and A7. Each conversion uses REF2_5V for the
1136: ;   references. The conversion results are stored in ADC12MEM0, ADC12MEM1,
1137: ;   ADC12MEM2, and ADC12MEM3 respectively and are moved to R5, R6, R7, and R8
1138: ;   respectively after the sequence is complete.
1139: ;
```

```
1140: ;
1141: ;                MSP430F149
1142: ;             -----------------
1143: ;            |                 |
1144: ;       O -->|P6.3/A3          |
1145: ;       F -->|P6.4/A4          |
1146: ;       A -->|P6.5/A5          |
1147: ;    Batt -->|P6.7/A7          |
1148: ;            |                 |
1149: ;   Ref: ;
1150: ;   M. Mitchell
1151: ;   Texas Instruments Inc.
1152: ;   Feb 2005
1153: ;
1154: ;   A. Kurt
1155: ;   makeLAB
1156: ;   Jan 2010
1157: ;   Built with IAR Assembler for MSP430 v4.21.2 (4.21.2.50066)
1158: ;*******************************************************************************
1159:
1160: ADCloop     bis     #ADC12SC,&ADC12CTL0     ; Start conversions
1161:             nop                             ; Only Required for debug
1162:             mov     &ADC12MEM0,R5           ; Move A3 result -O
1163:             mov     &ADC12MEM1,R6           ; Move A4 result -F
1164:             mov     &ADC12MEM2,R7           ; Move A5 result -A
1165:             mov     &ADC12MEM3,R8           ; Move A7 result -B, IFG is reset
1166:                                             ;
1167: ;testIFG    bit     #BIT0,&ADC12IFG         ; Conversion done?
1168: ;           jz      testIFG                 ; No, test again
1169:             rra     R5
1170:             rra     R5
1171:             rra     R5
1172:             rra     R5
1173: ;           dec     R5                      ; Divide by 16 to get max FF
1174:             mov     R5,&TBCCR3
1175:
1176:             rra     R6
1177:             rra     R6
1178:             rra     R6
1179:             rra     R6
1180:             inv.b   R6                       ; 1-F value (To easily tune F)
1181: ;           add     R6, R6
1182: ;           dec     R6                       ; Divide by 16 to get max FF
1183: ; So frequency range is 0.946 Hz to 24.92 Hz.
1184: ;
1185:             rra     R7
1186:             rra     R7
1187:             rra     R7
1188:             rra     R7
1189:             inv.b   R7
1190:             rra     R7
1191:             rra     R7
1192:             rra     R7
1193:             rra     R7
1194: ;           dec     R7                       ; Divide by 256 to get max F
1195:                                              ; and invert to start with F
1196:
1197: ;-----------------------------------------AK---------------------------------
1198:             cmp     #00h, R11
1199:             jz      D3
1200: D2          dec     R11                     ;
1201:             jz      D3                      ;
1202:             jmp     SK                      ; Wait for another ISR
1203: D3          dec     R10
1204:             jz      D0
1205:             jmp     SK
1206:
1207: D0          incd    R15                     ; Increment the pointer R15 to
1208:                                             ; to point to next word of sine
1209:                                             ; table.  Must increment by 2
1210:                                             ; because the sine table is words
1211:                                             ; not bytes.
1212:             and     #03Fh,R15               ; ANDing with 03Fh gives an
1213:                                             ; effective modulo 32 counter for
1214:                                             ; pointing to each value in the
1215:                                             ; sine table
```

```
1216:             cmp    #0h, R7
1217:             jz     ZeroSin                 ; If PotA is zero, give zero O/P
1218:             clr    R9
1219: Calc_Tab
1220:             add.w  #40h, R9                ; Calculate relevant Tab position
1221:             dec    R7
1222:             jz     Wave_Out
1223:             jmp    Calc_Tab
1224: Wave_Out
1225:             add.w  R9, R15
1226:             mov.w  Sin_K1_Tab(R15),&TBCCR1  ; Move new sine value to CCR1
1227:             jmp    Cont1
1228: ZeroSin
1229:             mov    #0h, R13
1230:             mov    R13, &TBCCR1            ; If R7 is Zero, then O/P=0
1231:
1232: ;Div         call   #div_hamacher           ; Division is OK.
1233:  ;           cmp    #00h, R9
1234:   ;          jz     A1
1235:    ;         mov    R9, R13
1236:     ;        jmp    A5
1237: ;A1          mov    #01h, R13
1238: ;A5          mov    R13, &TBCCR1            ; Amplitude scaled
1239: ;A3          nop
1240:
1241: Cont1
1242:             add    #04h,R14                ; Increment ramp value.
1243:                                            ; Changing the step size in R14
1244:                                            ; will change the frequency of
1245:                                            ; the ramp.
1246:             and    #0FFh,R14               ; And off unwanted bits
1247:             mov    R14,&TBCCR2             ; Move new ramp value to CCR2
1248:             clr    R11
1249:             clr    R10
1250: D1          mov    R6, R11                 ; This should set the period
1251: D11         mov    #0Ah, R10               ; Sets maximum frequency effectively
1252: SK          NOP
1253:             reti                           ; return with interrupts enabled
1254:
1255: ;-------------------------------------------------------------------------------
1256:         RSEG    DATA16_C                   ; Segment for const. data in Flash
1257: HR:
1258:         DB      DUB
1259:         DB      DIP, SOSPA, ENDTX
1260: Start:
1261:         DB      DOT, DOT, DOT, LETTER             ; S
1262:         DB      DOT, DASH, LETTER                 ; A
1263:         DB      DASH, DOT, DASH, DOT, LETTER      ; C
1264:         DB      DOT, DOT, LETTER                  ; I
1265:         DB      DASH, LETTER                      ; T
1266:         DB      SPACE                             ; /
1267:         DB      DASH, DOT, DASH, LETTER           ; K
1268:         DB      DOT, DASH, LETTER                 ; A
1269:         DB      DOT, DASH, DOT, LETTER            ; R
1270:         DB      DOT, DASH, LETTER                 ; A
1271:         DB      DASH, DASH, LETTER                ; M
1272:         DB      DOT, DOT, DASH, LETTER            ; U
1273:         DB      DOT, DASH, DOT, LETTER            ; R
1274:         DB      DOT, DOT, DOT, LETTER             ; S
1275:         DB      DOT, LETTER                       ; E
1276:         DB      DOT, DASH, DOT, DOT, LETTER       ; L
1277:         DB      ENDTX
1278: ; ... .- -.-. .. - / -.- .- .-. .- -- ..- .-. ... . .-..
1279: ; Sacit Karamursel
1280: Battery:
1281:         DB      DOT, DOT, DOT, LETTER
1282:         DB      DASH, DASH, LETTER
1283:         DB      DOT, DOT, DOT, ENDTX
1284: ;-------------------------------------------------------------------------------
1285: ;-------------------------------------------------------------------------------
1286:             COMMON  INTVEC                 ; MSP430x14x interrupt vectors
1287: ;-------------------------------------------------------------------------------
1288:             ORG     TIMERB0_VECTOR
1289:             DW      TB_ISR                      ; CCIFG0 interrupt
1290:             ORG     RESET_VECTOR
1291:             DW      RESET                       ; POR, ext. Reset, Watchdog
```

```
1292:             END
1293: ;-------------------------------------------------------------------------
1294:
1295:
1296:
1297:
1298: /*This file has been prepared for Doxygen automatic documentation generation.*/
1299: /*! \file ********************************************************************
1300:  *
1301: * TESsaNova: tDCS TransCranial DC Brain Stimulator
1302: * Developed for research studeies at MAKELab
1303: *
1304: * Main Control Code.
1305: *
1306: * Adnan Kurt
1307: * MakeLAB
1308: * 19Aug. 2013
1309: * Zekeriyakoy, Istanbul
1310: *
1311: * Skin Welding Laser_Controller has been rewritten, based on
1312: * the studies documented in BioLaser_Controller_v1_11Feb12.c
1313: *
1314: *
1315: * - File:               main_TESsaNova_Controller_v0_19Aug2013.c
1316: * - Compiler:           IAR EWBMSP430 5.40
1317: * - Supported devices:  MSP430F149
1318: * - Circuit:            TESsaNova_19Nov2011_F.sch
1319: *
1320: *  \author    AdKu            \n
1321: *             Adnan Kurt      \n
1322: *             MakeLAB        \n
1323: *             19Aug. 2013     \n
1324: *             Zekeriyakoy, Istanbul
1325: *
1326: *  $Name $
1327: *  $Revision: 0 $
1328: *  $RCSfile $
1329: *  $Date:    11Feb2012         $
1330: * "Store the total time used on flash memory." will not be implemented.
1331: *
1332: ******************************************************************************/
1333:
1334: # include <io430x14x.h>
1335: # include <in430.h>
1336: //# include <msp430x14x.h>
1337: //# include <intrinsics.h>
1338: # include <math.h>
1339: # include <stdint.h>
1340: # include <TESsaNova_Board_Definition_File_19Aug2013.c>
1341: # include <TESsaNova_Initializations_19Aug2013.c>
1342: # include <TESsaNova_Clock_Set_19Aug2013.c>
1343: # include <TESsaNova_PP_Loop_Timer_19Aug2013.c>
1344: # include <TESsaNova_Citi_Wall_19Aug2013.c>
1345: # include <TESsaNova_AnalogSensors_19Aug2013.c>
1346: # include <TESsaNova_Switches_19Aug2013.c>
1347: # include <TESsaNova_Citi_Wall_Content_19Aug2013.c>
1348: # include <TESsaNova_Musica_19Aug2013.c>
1349: // # include <TESsaNova_FlashWrite_19Aug2013.c>
1350: # include <TESsaNova_DAC_Drive19Aug2013.c>
1351:
1352: /*
1353: void StoreParameters(void)
1354: {
1355:     HB_on();
1356: // Parameters are less than the previous example. Take Care!
1357:     Memorize (Duration, AC_Current, DC_Current,
1358:               Frequency);
1359:     HB_off();
1360:     // Report "Parameters Memorized."
1361:     Parameters_Saved ();
1362:     wait(300);                      // Wait for 500 mseconds more
1363:
1364: }
1365: */
1366:
1367: void Check_Battery(void)
```

```
1368: {
1369:   // Battery Check Code*********************************************************
1370:           // Read Battery
1371:           // Maximum voltage expected is 18V. Normally, 9 V battery will
1372:           // be used and expected returned number is 900 cV.
1373:           // If Vs<750cV, stop and report the error -Battery Exhaust
1374:           // If Vs<800 & Vs>750cV, just warn the user and continue
1375:           // If Vs<750cV, warn the user and stop there.
1376:           Supply_Voltage();
1377:           // in centivolts cV
1378:           if ((Vs_Read_value<800)&(Vs_Read_value>750))
1379:           {
1380:           // set Error
1381:           // Report to CitiWall
1382:           Citi_Wall_Battery();
1383:           Play_it_Sam ("MahnaMahna:d=16,o=6,b=125:c#,c.,b5,8a#.5,8f.,4g#,a#,g.,4d#,8p,c#,c.,b5,8a#.5,8f.,g#.,
      8a#.,");
1384:           wait(2000);
1385:       }
1386:           // Insert Test Report here
1387:           // Battery condition and test results
1388:           // If Error_Beacon did not light up during test, it is OK.
1389:           // If Battery Error, it is reasonable, otherwise another error with the device.
1390:           Supply_Voltage();
1391:           if (Vs_Read_value<750)
1392:           {
1393:           // set Error
1394:           // Report to CitiWall
1395:           Citi_Wall_Battery();
1396: Play_it_Sam ("GoodBad:d=4,o=5,b=56:32p,32a#,32d#6,32a#,32d#6,8a#.,16f#.,16g#.,d#,32a#,32d#6,32a#,32d#6,8a#.,
      16f#.,16g#.,c#6,32a#,");
1397:           wait(2000);
1398:           while (1)
1399:           {
1400:           }
1401:           }
1402: // End of Battery Check Code**************************************************
1403: }
1404:
1405: // Initialization
1406: void Init(void) {               // Init HW&SW
1407: //      BCSCTL3 |= LFXT1S_0;        // 32 kHz Crystal
1408:         Board_Setup();
1409:         Clock_Setting ();
1410:         Clock_Set ();
1411: //      Test_Info_Memory();
1412:         Loop_Timer();
1413:         __bis_SR_register( __SR_GIE );
1414:         wait(100);                    // Wait for 500 mseconds more
1415:         tDCS_Parameters();
1416:         // Make sure that the Stimulator turns off.
1417:         // set OD high
1418:         XTR_Out_Disable = 0x01;
1419: //      Make sure that the Stimulator turns off.
1420: //      Mode5 sets output to zero.
1421:         DAC_Write(5, 1, 0, 0);
1422:     Citi_Wall_0 ();           // Welcome Screen
1423:     wait(100);            // Wait for 500 mseconds more
1424: //        Changed = 0;                // If it is 1, then store parameters to Flash.
1425: //        have to think how to store parameters
1426: //        Remember_Parameters();      // Recall whatever required
1427:
1428:         // Initially connect to the ShowroomDummy
1429:         // set BRLY = 0;
1430:         Board_Relay = 0;
1431:         // Turn_Off LEDs
1432:         Board_LED = 0;
1433:         HeartBeat_LED = 0;
1434:         Error_Beacon_off ();
1435:         // p5.7 OnBoard activity Monitor
1436:         p5_7 = 0;
1437:         Check_Battery();
1438: }
1439:
1440: void RunTestLoop(int Test_Amplitude, int Test_Duration)
1441: {
```

```
1442:           // Turn off heartbeat and turn on LED
1443:           Stimulation_On = 1;
1444:           // Route the current from Subject Body Model to the Output
1445:           // set BRLY = 1;
1446:           Board_Relay = 1;
1447:           // During relay turn on, there is a voltage peak at the output.
1448:           // I will use the following delay, to isolate it.
1449:           // Exact 10ms delay.
1450:           // OK. When I connected a second scope probe it disappeared.
1451:           // Seems to be a very low energy pulse.
1452:           wait(10);
1453:           // Turn ON the Stimulator .
1454:           // set OD low
1455:           XTR_Out_Disable = 0x00;
1456:           // Set Stimulation Marker (ZP & ZAP) output ON
1457:           // This signal is output of IC601 pin6 with 1k series resistor.
1458:           // IC601 is x4049, so inverted output. Could be used to monitor
1459:           // electrically or an LED might be connected on the panel.
1460:           Stimulation_Marker = 0x00;
1461:           // During Process Loops, it is 0.
1462:           // This flag is used to message the run status to emergency check.
1463:           Looping = 0;
1464:           // void Stimulation_Output (int AC_Amplitude, int DC_Amplitude, int Frequency, int Duration)
1465:           // Frequency should not be zero for sampling purposes! Bad style, keep it for now.
1466:           Test_Stimulation_Output (0, Test_Amplitude, 50, Test_Duration);
1467:           // Make sure that the Stimulator turns off.
1468:           // set OD high
1469:           XTR_Out_Disable = 0x01;
1470:           // Set Stimulation Marker (ZP & ZAP) output OFF
1471:           Stimulation_Marker = 0x01;
1472:           // Route the current to theSubject Body Model
1473:           // set BRLY = 1;
1474:           Board_Relay = 1;
1475:           // During Process Loops, it is 0.
1476:           // This flag is used to message the run status to emergency check.
1477:           Looping = 1;
1478:           // Turn on heartbeat and turn off LED
1479:           Stimulation_On = 0;
1480: }
1481:
1482: // Detailed testing to be done over the human model
1483: // Testing will reveal correct operation by comparing set current
1484: // and measured current. It will be repeated for a range of current
1485: // levels. That will also give information about battery status under load.
1486: void Test_It_Detail(void)
1487: {
1488:   // Test_Step determines the current steps in uA
1489:   int Test_Step = 500;
1490:   // Test_Duration is taken to be 1sec each
1491:   int Test_Duration = 1;
1492:   for (Test_Amplitude = Test_Step ; Test_Amplitude <= 2000; Test_Amplitude = Test_Amplitude + Test_Step)
1493:   {
1494:           Citi_Wall_Testing();
1495:           RunTestLoop(Test_Amplitude, Test_Duration);
1496:           Check_Battery();
1497:   }
1498:   Play_it_Sam ("MissionImp:d=16,o=6,b=95:32d,32d#,32d,32d#,32d,32d#,32d");
1499:   Play_it_Sam ("MissionImp:d=16,o=6,b=95:32d,32d#,32d,32d#,32d,32d#,32d");
1500:   wait(500);
1501: }
1502:
1503: void RunStimulationLoop(void)
1504: {
1505:           // Turn off heartbeat and turn on LED
1506:           Stimulation_On = 1;
1507:           // Route the current from Subject Body Model to the Output
1508:           // set BRLY = 1;
1509:           Board_Relay = 1;
1510:           // During relay turn on, there is a voltage peak at the output.
1511:           // I will use the following delay, to isolate it.
1512:           // Exact 10ms delay.
1513:           // OK. When I connected a second scope probe it disappeared.
1514:           // Seems to be a very low energy pulse.
1515:           wait(10);
1516:           // Turn ON the Stimulator .
1517:           // set OD low
```

```
1518:           XTR_Out_Disable = 0x00;
1519:           // Set Stimulation Marker (ZP & ZAP) output ON
1520:           // This signal is output of IC601 pin6 with 1k series resistor.
1521:           // IC601 is x4049, so inverted output. Could be used to monitor
1522:           // electrically or an LED might be connected on the panel.
1523:           Stimulation_Marker = 0x00;
1524:           // During Process Loops, it is 0.
1525:           // This flag is used to message the run status to emergency check.
1526:           Looping = 0;
1527:           // void Stimulation_Output (int AC_Amplitude, int DC_Amplitude, int Frequency, int Duration)
1528:           Stimulation_Output ((int)Amplitude_Set_value, (int)DC_Offset_Set_value, (int)Frequency_Set_value,
      Stimulation_Duration);
1529:           // Make sure that the Stimulator turns off.
1530:           // set OD high
1531:           XTR_Out_Disable = 0x01;
1532:           // Set Stimulation Marker (ZP & ZAP) output OFF
1533:           Stimulation_Marker = 0x01;
1534:           // Route the current from Output to theSubject Body Model
1535:           // set BRLY = 0;
1536:           Board_Relay = 0;
1537:           // During Process Loops, it is 0.
1538:           // This flag is used to message the run status to emergency check.
1539:           Looping = 1;
1540:           // Turn on heartbeat and turn off LED
1541:           Stimulation_On = 0;
1542: }
1543:
1544: // Process Loop
1545: int main (void)
1546:     {
1547:     // Might add default DCO here, in case an interrupt needs to be served.
1548:     // Not a necessity now.
1549:     __disable_interrupt();
1550: //  Play_it_Sam ("MissionImp:d=16,o=6,b=95:32d,32d#,32d,32d#,32d,32d#,32d,32d#,32d,32d,32d#,32e,32f,32f#,32g,g,
      8p,g,8p,a#,p,c7,p,g,8p,g,8p,f,p,f#,p,g,8p,g,8p,a#,p,c7,p,g,8p,g,8p,f,p,f#,p,a#,g,2d,32p,a#,g,2c#,32p,a#,g,2c,
      a#5,8c,2p,32p,a#5,g5,2f#,32p,a#5,g5,2f,32p,a#5,g5,2e,d#,8d");
1551:     Play_it_Sam ("MissionImp:d=16,o=6,b=95:32d,32d#,32d,32d#,32d,32d#,32d,32d#,32d,32d,32d#,32e,32f,32f#");
1552:     WDTCTL = WDTPW + WDTHOLD;         // Stop WDT
1553:     Init ();                  // Init HW&SW
1554:     // Check Process Key. If button is pressed during startup phase
1555:     // Then run the test routine.
1556:     Process_Key ();
1557:     if (Process == 1)
1558:     {
1559:       Test_It_Detail();
1560:     }
1561: while (1){                // Main Loop
1562:     // Read Panel
1563:         Panel_Read ();
1564:         // Update CitiWall and HB
1565:     Citi_Wall_Params ();
1566:     wait(10);
1567:     if (Process == 1)             // Make it 0 for Continuous Run to test!
1568:         {
1569: //  Following for storage of parameters
1570: //  if (Changed == 1){
1571: //  Changed = 0;
1572: //  StoreParameters();
1573:           Citi_Wall_Params ();
1574:           wait(100);              // Wait for 100 mseconds more
1575:           Play_it_Sam ("MissionImp:d=16,o=6,b=95:32d,32d#,32d,32d#,32d,32d#,32d");
1576:           wait(300);
1577:           Check_Battery();
1578:           Citi_Wall_Loops();
1579:           wait(10);
1580:           RunStimulationLoop();
1581: //        Graceful ending to the Session
1582: //        Report Stimulation Data
1583: //        Report Errors
1584:           Play_it_Sam ("Indiana:d=4,o=5,b=250:e,8p,8f,8g,8p,1c6,8p.,d,8p#");
1585:           wait(500);
1586:           Check_Battery();
1587:           wait(100);
1588:           Citi_Wall_Params ();
1589:   }
1590: //        Remember_Parameters();      // Recall whatever required
```

```
1591: //        Beep(100, 2);              // Beep is no Good.
1592:         }
1593:     }
1594:
1595: /*
1596: // Timer B0 interrupt service routine
1597: #pragma vector=TIMERB0_VECTOR
1598: __interrupt void Timer_B0 (void)
1599: {
1600:       HB ^= 0x01;              // To monitor interrupt cycle
1601:       bit.TickISR = 1;
1602: }
1603: */
1604:
1605:
1606: // ******************************************************************************** */
1607:
1608:
1609: /*This file has been prepared for Doxygen automatic documentation generation.*/
1610: /*! \file ****************************************************************
1611:  *
1612: * TESsaNova: tDCS TransCranial DC Brain Stimulator
1613: * Developed for research studies at MAKELab
1614: *
1615: * Analog Sensor Conversions File
1616: *
1617: * Adnan Kurt
1618: * MakeLAB
1619: * 19Aug. 2013
1620: * Zekeriyakoy, Istanbul
1621: * - File:               TESsaNova_Board_Definition_File_19Aug2013.c
1622: * - Compiler:           IAR EWBMSP430 5.40
1623: * - Supported devices:  MSP430F149
1624: * - Circuit:            TESsaNova_19Nov2011_F.sch
1625: *
1626: *  \author    AdKu            \n
1627: *             Adnan Kurt      \n
1628: *             MakeLAB        \n
1629: *             19Aug. 2013     \n
1630: *             Zekeriyakoy, Istanbul
1631: *
1632: # define  Current_Sample_Read   P6IN_bit.P0
1633: // 0x01 // 0000 0001b p6  // Output Current Sample
1634: # define  Amplitude_Set_Pot     P6IN_bit.P1
1635: // 0x02 // 0000 0010b p6  // Amplitude Set Pot
1636: # define  DC_Offset_Set_Pot     P6IN_bit.P2
1637: // 0x04 // 0000 0100b p6  // DC Offset Set Pot
1638: # define  Freq_Set_Pot          P6IN_bit.P3
1639: // 0x08 // 0000 1000b p6  // Frequency Set Pot
1640: # define  Battery_Monitor       P6IN_bit.P5
1641: // 0x20 // 0010 0000b p6  // Battery Monitor
1642: *
1643: * Pot values need to be scaled for comparision and initial value settings!
1644: *
1645: ********************************************************************************/
1646:
1647: unsigned int  Amplitude_Set = 0;
1648: double  Amplitude_Set_value = 0;
1649: unsigned int  DC_Offset_Set = 0;
1650: double  DC_Offset_Set_value = 0;
1651: unsigned int  Frequency_Set = 0;
1652: double  Frequency_Set_value = 0;
1653:
1654: // Amplitude_Set_Pot Read Routine
1655: void Amplitude_Set_Read (void) {
1656:   ADC12CTL1 = 0;
1657:   ADC12CTL0 = 0;
1658:   Amplitude_Set = 0;
1659:   Amplitude_Set_value = 0;
1660:   int (loopy)= 101;                    // Number of samples to average.
1661:   ADC12MCTL0 = INCH_1 | SREF_0 ;
1662:   ADC12CTL1 = CSTARTADD_0 | SHS_0 | ADC12DIV_2 | ADC12SSEL_2 | CONSEQ_0 | SHP;
1663:   ADC12CTL0 = ADC12ON | REFON | SHT0_8 | REF2_5V;
1664:   ADC12CTL0_bit.ENC = 1;
1665:               Delay(_10us);
1666:               Delayx100us(2);           //2ms
```

```
1667:               while (loopy > 1)
1668:               {
1669:                 loopy --;
1670:                 ADC12CTL0_bit.ADC12SC = 1;
1671:                 while (ADC12CTL1_bit.ADC12BUSY == 1){
1672:                                                     }
1673:                 // Amplitude_Set_Read
1674:                 // No need to scale with voltage reference.
1675:                 // Amplitude_Set_Read = (ADCread/4096)*2500mV max.
1676:                 // Scale Amplitude_Set_value = (Amplitude_Set_value/4096)*2000 uA
1677:                 Amplitude_Set = ADC12MEM0;
1678:                 if (NoiseTest1 > 0)
1679:                 {
1680:                 if (fabs(NoiseTest1-Amplitude_Set)> 2000)
1681:                 {
1682:                     Amplitude_Set = (int)NoiseTest1;
1683:                     Amplitude_Set_value = Amplitude_Set_value + Amplitude_Set;
1684:                 }
1685:                   else {Amplitude_Set_value = Amplitude_Set_value + Amplitude_Set;
1686:                 }
1687:                 }
1688:                   else {Amplitude_Set_value = Amplitude_Set_value + Amplitude_Set;
1689:                 }
1690:                 }
1691:   ADC12CTL0_bit.ENC = 0;
1692:   Amplitude_Set_value = Amplitude_Set_value / 100;
1693:   NoiseTest1 = (int)Amplitude_Set_value;
1694:   Amplitude_Set_value = (Amplitude_Set_value/4096)*4000;
1695:   if (Amplitude_Set_value >= MaxAmplitude){
1696:     Amplitude_Set_value = MaxAmplitude;
1697:   }
1698:   if (Amplitude_Set_value < 1.0){
1699:     Amplitude_Set_value = 0.0;
1700:   }
1701:   ADC12CTL0 = 0;
1702:   ADC12CTL1 = 0;
1703: }
1704:
1705: // DC_Offset_Set_Pot Read Routine
1706: void DC_Offset_Set_Read (void){
1707:   ADC12CTL1 = 0;
1708:   ADC12CTL0 = 0;
1709:   DC_Offset_Set = 0;
1710:   DC_Offset_Set_value = 0;
1711:   int (loopy)= 101;                     // Number of samples to average.
1712:   ADC12MCTL0 = INCH_2 | SREF_0 ;
1713:   ADC12CTL1 = CSTARTADD_0 | SHS_0 | ADC12DIV_2 | ADC12SSEL_2 | CONSEQ_0 | SHP;
1714:   ADC12CTL0 = ADC12ON | REFON | SHT0_8 | REF2_5V;
1715:   ADC12CTL0_bit.ENC = 1;
1716:               Delay(_10us);
1717:               Delayx100us(2);           //2ms
1718:               while (loopy > 1)
1719:               {
1720:                 loopy --;
1721:                 ADC12CTL0_bit.ADC12SC = 1;
1722:                 while (ADC12CTL1_bit.ADC12BUSY == 1){
1723:                                                     }
1724:                 // DC_Offset_Set_Read
1725:                 // No need to scale with voltage reference.
1726:                 // DC_Offset_Set_Read = (ADCread/4096)*2500mV max.
1727:                 // Scale DC_Offset_Set_value = (DC_Offset_Set_value/4096)*2000 uA
1728:                 DC_Offset_Set = ADC12MEM0;
1729:                 if (NoiseTest2 > 0)
1730:                 {
1731:                 if (fabs(NoiseTest2-DC_Offset_Set)> 2000)
1732:                 {
1733:                     DC_Offset_Set = (int)NoiseTest2;
1734:                     DC_Offset_Set_value = DC_Offset_Set_value + DC_Offset_Set;
1735:                 }
1736:                   else {DC_Offset_Set_value = DC_Offset_Set_value + DC_Offset_Set;
1737:                 }
1738:                 }
1739:                   else {DC_Offset_Set_value = DC_Offset_Set_value + DC_Offset_Set;
1740:                 }
1741:                 }
1742:   ADC12CTL0_bit.ENC = 0;
```

```
1743:   DC_Offset_Set_value = DC_Offset_Set_value / 100;
1744:   NoiseTest2 = (int)DC_Offset_Set_value;
1745:   DC_Offset_Set_value = (DC_Offset_Set_value/4096)*4000;
1746:   if (DC_Offset_Set_value >= MaxDCOffset){
1747:     DC_Offset_Set_value = MaxDCOffset;
1748:   }
1749:   if (DC_Offset_Set_value < 1.0){
1750:     DC_Offset_Set_value = 0.0;
1751:   }
1752:   ADC12CTL0 = 0;
1753:   ADC12CTL1 = 0;
1754: }
1755:
1756: // Frequency_Set_Pot Read Routine
1757: void Frequency_Set_Read (void){
1758:   ADC12CTL1 = 0;
1759:   ADC12CTL0 = 0;
1760:   Frequency_Set = 0;
1761:   Frequency_Set_value = 0;
1762:   int (loopy)= 101;                     // Number of samples to average.
1763:   ADC12MCTL0 = INCH_3 | SREF_0 ;
1764:   ADC12CTL1 = CSTARTADD_0 | SHS_0 | ADC12DIV_2 | ADC12SSEL_2 | CONSEQ_0 | SHP;
1765:   ADC12CTL0 = ADC12ON | REFON | SHT0_8 | REF2_5V;
1766:   ADC12CTL0_bit.ENC = 1;
1767:               Delay(_10us);
1768:               Delayx100us(2);           //2ms
1769:               while (loopy > 1)
1770:               {
1771:                 loopy --;
1772:                 ADC12CTL0_bit.ADC12SC = 1;
1773:                 while (ADC12CTL1_bit.ADC12BUSY == 1){
1774:                                                     }
1775:                 // Frequency_Set_read
1776:                 // No need to scale with voltage reference.
1777:                 // Frequency_Set_read = (ADCread/4096)*2500mV max.
1778:                 // Scale Frequency_Set_value = (Frequency_Set_value/4096)*200 Hz
1779:                 Frequency_Set = ADC12MEM0;
1780:                 if (NoiseTest3 > 0)
1781:                 {
1782:                 // Check value (2000) caused freezing.
1783:                 // Increasing solved the problem.
1784:                 if (fabs(NoiseTest3-DC_Offset_Set)> 4000)
1785:                 {
1786:                     Frequency_Set = (int)NoiseTest3;
1787:                     Frequency_Set_value = Frequency_Set_value + Frequency_Set;
1788:                 }
1789:                   else {Frequency_Set_value = Frequency_Set_value + Frequency_Set;
1790:                 }
1791:                 }
1792:                   else {Frequency_Set_value = Frequency_Set_value + Frequency_Set;
1793:                 }
1794:                 }
1795:   ADC12CTL0_bit.ENC = 0;
1796:   Frequency_Set_value = Frequency_Set_value / 100;
1797:   NoiseTest3 = (int)Frequency_Set_value;
1798:   // Maximum frequency that we can get is 80Hz with 4*subsampling at 4MHz SMCLK
1799:   // So, dHz unit will be used, and down to 0.1Hz can be achieved at low end.
1800:   Frequency_Set_value = (Frequency_Set_value/4096)*800;
1801:   if (Frequency_Set_value >= MaxFrequency){
1802:     Frequency_Set_value = MaxFrequency;
1803:   }
1804:   if (Frequency_Set_value < 1.0){
1805:     Frequency_Set_value = 1.0;
1806:   }
1807:   ADC12CTL0 = 0;
1808:   ADC12CTL1 = 0;
1809: }
1810:
1811: // Sense_i Read Routine
1812: // Output current sampling. However, this routine might better be embedded into
1813: // Interrupt service request
1814: void Sense_Current (void){
1815:   ADC12CTL1 = 0;
1816:   ADC12CTL0 = 0;
1817:   Sense_i_read = 0;
1818:   int (loopy)= 100;                    // Number of samples to average.
```

```
1819:   ADC12MCTL0 = INCH_0 | SREF_1 ;       // Use 2.5V reference
1820:   ADC12CTL1 = CSTARTADD_0 | SHS_0 | ADC12DIV_2 | ADC12SSEL_2 | CONSEQ_0 | SHP;
1821:   ADC12CTL0 = ADC12ON | REFON | SHT0_8 | REF2_5V;
1822:   ADC12CTL0_bit.ENC = 1;
1823:               Delay(_10us);
1824:               Delayx100us(2);           //2ms
1825:               while (loopy > 1)
1826:               {
1827:                 loopy --;
1828:                 ADC12CTL0_bit.ADC12SC = 1;
1829:                 while (ADC12CTL1_bit.ADC12BUSY == 1){
1830:                                                     }
1831:                 // Sense_i_read
1832:                 // No need to scale with voltage reference.
1833:                 // Sense_i_read = (ADCread/4096)*2500mV max.
1834:                 // CUSA floats at 375mV baseline. That is as expected.
1835:                 // There is 190mV reference offset at INA118 output.
1836:                 // So, that is reflected to the OPA336 output as
1837:                 // (1+24/33)*190=378mV.
1838:                 Sense_i_read = ADC12MEM0;
1839:                 if (NoiseTest4 > 0)
1840:                 {
1841:                 if (fabs(NoiseTest4-Sense_i_read)> 2000)
1842:                 {
1843:                     Sense_i_read = (int)NoiseTest4;
1844:                     Sense_i_value = Sense_i_value + Sense_i_read;
1845:                 }
1846:                   else {Sense_i_value = Sense_i_value + Sense_i_read;
1847:                 }
1848:                 }
1849:                   else {Sense_i_value = Sense_i_value + Sense_i_read;
1850:                 }
1851:                 }
1852:   ADC12CTL0_bit.ENC = 0;
1853:   Sense_i_value = Sense_i_value / 100;
1854:   NoiseTest4 = (int)Sense_i_value;
1855:   // So, that is reflected to the OPA336 output as
1856:   // (1+24/33)*190=378mV.
1857:   // Sense_i_value = ((Sense_i_value*2500)/4096-378)/285; mA
1858:   // in uA
1859:   Sense_i_value = ((Sense_i_value*(305.0/500.0)-378.0)*(1000.0/285.0));
1860:   if (Sense_i_value >= MaxCurrent){
1861:     Sense_i_value = MaxCurrent;
1862:   }
1863:   if (Sense_i_value < 1.0){
1864:     Sense_i_value = 0.0;
1865:   }
1866:   ADC12CTL0 = 0;
1867:   ADC12CTL1 = 0;
1868: }
1869:
1870: // Supply Voltage Read Routine
1871: void Supply_Voltage (void){
1872:   ADC12CTL1 = 0;
1873:   ADC12CTL0 = 0;
1874:   Vs_Read_Read = 0;
1875:   int (loopy)= 101;                     // Number of samples to average.
1876:   ADC12MCTL0 = INCH_5 | SREF_1 ;        // 2.5V reference used
1877:   ADC12CTL1 = CSTARTADD_0 | SHS_0 | ADC12DIV_2 | ADC12SSEL_2 | CONSEQ_0 | SHP;
1878:   ADC12CTL0 = ADC12ON | REFON | SHT0_8 | REF2_5V;
1879:   ADC12CTL0_bit.ENC = 1;
1880:               Delay(_10us);
1881:               Delayx100us(2);           //2ms
1882:               while (loopy > 1)
1883:               {
1884:                 loopy --;
1885:                 ADC12CTL0_bit.ADC12SC = 1;
1886:                 while (ADC12CTL1_bit.ADC12BUSY == 1){
1887:                                                     }
1888:                 // Vs_Read
1889:                 // No need to scale with voltage reference.
1890:                 // Vs_Read = (ADCread/4096)*2500mV max.
1891:                 Vs_Read_Read = ADC12MEM0;
1892:                 if (NoiseTest5 > 0)
1893:                 {
1894:                 if (fabs(NoiseTest5-Vs_Read_Read)> 2000)
```

```
1895:                 {
1896:                     Vs_Read_Read = NoiseTest5;
1897:                     Vs_Read_value = Vs_Read_value + Vs_Read_Read;
1898:                 }
1899:                   else {Vs_Read_value = Vs_Read_value + Vs_Read_Read;
1900:                 }
1901:                 }
1902:                   else {Vs_Read_value = Vs_Read_value + Vs_Read_Read;
1903:                 }
1904:                 }
1905:   ADC12CTL0_bit.ENC = 0;
1906:   Vs_Read_value = Vs_Read_value / 100;
1907:   NoiseTest5 = (int)Vs_Read_value;
1908:   // Maximum voltage that we can read is 18V
1909:   // So, cV unit will be used
1910:   // Vs_Read_value = (Vs_Read_value/4096)*2500*(18/2500);
1911:   // In place of (18/2500), use the exact transfer function
1912:   // Vs_Read_value = (Vs_Read_value/4096)*2500*(18/1500);
1913:   // Vs_Read_value = (Vs_Read_value/4096.0)*9*(25.0/15.0);
1914:   // in centivolts cV
1915:   Vs_Read_value = (Vs_Read_value*15.0/4096.0)*100;
1916:   if (Vs_Read_value >= MaxVoltage){
1917:     Vs_Read_value = MaxVoltage;
1918:   }
1919:   if (Vs_Read_value < 1.0){
1920:     Vs_Read_value = 0.0;
1921:   }
1922:   ADC12CTL0 = 0;
1923:   ADC12CTL1 = 0;
1924: }
1925:
1926: // Board_Tempera Read Routine
1927: // Board temperature monitor -just in case.
1928: void Board_Tempera_Sensor (void) {
1929: int long IntDegC;
1930:   ADC12CTL0 = SHT0_8 + REFON + ADC12ON;
1931:   ADC12CTL1 = SHP;                          // enable sample timer
1932:   ADC12MCTL0 = SREF_1 + INCH_10;
1933:   ADC12CTL0 |= ENC;
1934:   ADC12CTL0 |= ADC12SC;                   // Sampling and conversion start
1935: //  oC = ((x/4096)*1500mV)-986mV)*1/3.55mV = x*423/4096 - 278
1936: //  IntDegC = (ADC12MEM0 - 2692)* 423/4096
1937:   while (ADC12CTL1_bit.ADC12BUSY == 1){
1938:   }
1939:   Board_Temperature = ADC12MEM0;          // Move results, IFG is cleared
1940:   IntDegC = (int)((Board_Temperature - 2692) * 423);
1941:   IntDegC = IntDegC / 4096;
1942:   Board_Temperature = IntDegC;
1943:   ADC12CTL0 = 0;
1944:   ADC12CTL1 = 0;
1945:   }
1946:
1947: /*
1948: Circular Buffer Example
1949: //-----------------------------------------------------------------------------
1950: // The Timer_A CCR0 ISR is called on each TACCR0 capture event to obtain
1951: // the time stamp of the input signal transition. An 8-tap moving average
1952: // filter is used to minimize measurement error.
1953: //-----------------------------------------------------------------------------
1954: #pragma vector = TIMERA0_VECTOR
1955: __interrupt void TimerA0_ISR(void)
1956: {
1957:   SpeedMemSum -= SpeedMem[pSpeedMem];           // Remove oldest value
1958:   SpeedMem[pSpeedMem] = (unsigned int)(TACCR0 - LastTACCR);  // Replace with current
1959:   SpeedMemSum += SpeedMem[pSpeedMem++];         // Update running sum
1960:   CurrentSpeed = SpeedMemSum >> 3;              // Calc speed by div 8
1961:   pSpeedMem &= 0x07;                            // Adjust circular pointer
1962:
1963:   LastTACCR = TACCR0;
1964:   TACCR1 = LastTACCR + MIN_SPEED;               // Set timeout for minimum speed
1965: }                                               // to be read out
1966: */
1967:
1968: // ***************************************************************************** */
1969:
1970: /*This file has been prepared for Doxygen automatic documentation generation.*/
```

```
1971: /*! \file ********************************************************************
1972:  *
1973: * TESsaNova: tDCS TransCranial DC Brain Stimulator
1974: * Developed for research studeies at MAKELab
1975: *
1976: * Board Definition and Pin Connections File
1977: *
1978: * Adnan Kurt
1979: * MakeLAB
1980: * 19Aug. 2013
1981: * Zekeriyakoy, Istanbul
1982: * - File:               TESsaNova_Board_Definition_File_19Aug2013.c
1983: * - Compiler:           IAR EWBMSP430 5.40
1984: * - Supported devices:  MSP430F149
1985: * - Circuit:            TESsaNova_19Nov2011_F.sch
1986: *
1987: *  \author    AdKu            \n
1988: *             Adnan Kurt      \n
1989: *             MakeLAB        \n
1990: *             19Aug. 2013     \n
1991: *             Zekeriyakoy, Istanbul
1992: *
1993: *  Integer type Example
1994: *  Binary  1010b, b'1010'
1995: *  Octal  1234q, q'1234'
1996: *  Decimal  1234, -1, d'1234'
1997: *  Hexadecimal  0FFFFh, 0xFFFF, h'FFFF'
1998: *
1999: *  More pins available. Complete those
2000: *
2001: ***************************************************************************/
2002:
2003: // Pin Definitions
2004: // Find PCB connector layout on 19Aug2013 notes
2005: // This naming convention is compatible with schematic
2006: # define  ZAP1            0x02 // 0000 0010b p1  // Triggers Stimulation
2007: # define  BZZR            0x04 // 0000 0100b p1  // Amplified Buzzer
2008: # define  INCREMENT       0x08 // 0000 1000b p1  // Increment Switch
2009: # define  DECREMENT       0x10 // 0001 0000b p1  // Decrement Switch
2010: # define  ZAP             0x20 // 0010 0000b p1  // Current Out Monitor ?
2011: # define  BRLY            0x40 // 0100 0000b p1  // Relay on Board -Human Model
2012: # define  XTR_ERROR       0x80 // 1000 0000b p1  // XTR Control Error Flag
2013: # define  IA_             0x01 // 0000 0001b p2  // BCD data A 7Segment LED
2014: # define  IB_             0x02 // 0000 0010b p2  // BCD data B 7Segment LED
2015: # define  IC_             0x04 // 0000 0100b p2  // BCD data C 7Segment LED
2016: # define  ID_             0x08 // 0000 1000b p2  // BCD data D 7Segment LED
2017: # define  LEH_            0x10 // 0001 0000b p2  // High Digit Enable
2018: # define  LEL_            0x20 // 0010 0000b p2  // Low Digit Enable
2019: # define  BLED            0x40 // 0100 0000b p2  // LED on Board
2020: # define  NOTSYNC_        0x01 // 0000 0001b p3  // SPI sync
2021: # define  DIN_            0x02 // 0000 0010b p3  // DAC Data -SPI
2022: # define  SCLK_           0x08 // 0000 1000b p3  // SPI clk
2023: # define  LELL_           0x10 // 0001 0000b p3  // LowerDigit  enable
2024: # define  LELLL_          0x20 // 0010 0000b p3  // LowestDigit  enable
2025: # define  Beat            0x02 // 0000 0010b p4  // Heart Beat LED-pwm
2026: # define  D4_             0x04 // 0000 0100b p4  // LCD Data D4
2027: # define  D5_             0x08 // 0000 1000b p4  // LCD Data D5
2028: # define  D6_             0x10 // 0001 0000b p4  // LCD Data D6
2029: # define  D7_             0x20 // 0010 0000b p4  // LCD Data D7
2030: # define  ELCD_           0x40 // 0100 0000b p4  // Enable LCD
2031: # define  RS_             0x80 // 1000 0000b p4  // Reset Signal LCD
2032: // R/W of LCD pin to be connected to GND
2033: # define  OD              0x08 // 0000 1000b p5  // Output Disable -XTR off
2034: # define  CUSA            0x01 // 0000 0001b p6  // Output Current Sample
2035: # define  AMPL            0x02 // 0000 0010b p6  // Amplitude Set Pot
2036: # define  DCOS            0x04 // 0000 0100b p6  // DC Offset Set Pot
2037: # define  FREQ            0x08 // 0000 1000b p6  // Frequency Set Pot
2038: # define  BATT_SAMPLE     0x20 // 0010 0000b p6  // Battery Monitor
2039: # define  p5_0_           0x01 // 0000 0001b p5  // Buffered p5.0
2040: # define  p5_1_           0x02 // 0000 0010b p5  // Buffered p5.1
2041: # define  p5_2_           0x04 // 0000 0100b p5  // Buffered p5.2
2042: # define  p5_6_           0x40 // 0100 0000b p5  // p5.6
2043: # define  p5_7_           0x80 // 1000 0000b p5  // p5.7
2044: # define  p4_0_           0x01 // 0000 0001b p4  // p4.0
2045: # define  p1_0_           0x01 // 0000 0001b p1  // p1.0
2046:
```

```
2047: // This naming scheme is to enhance readability in the program
2048: # define  Stimulation_trigger   P1IN_bit.P1
2049: // 0x02 // 0000 0010b p1  // Triggers Stimulation
2050: # define  Buzzer               P1OUT_bit.P2
2051: // 0x04 // 0000 0100b p1  // Amplified Buzzer
2052: # define  Increment            P1IN_bit.P3
2053: // 0x08 // 0000 1000b p1  // Increment Switch
2054: # define  Decrement            P1IN_bit.P4
2055: // 0x10 // 0001 0000b p1  // Decrement Switch
2056: # define  Error_Beacon         P1OUT_bit.P5
2057: // 0x20 // 0010 0000b p1  // Blinks at inconsistent current outputs
2058: # define  Board_Relay          P1OUT_bit.P6
2059: // 0x40 // 0100 0000b p1  // Relay on Board -Human Model-Subject Switcher
2060: # define  XTR_Error            P1IN_bit.P7
2061: // 0x80 // 1000 0000b p1  // XTR Control Error Flag
2062: # define  IA                   P2OUT_bit.P0
2063: // 0x01 // 0000 0001b p2  // BCD data A 7Segment LED
2064: # define  IB                   P2OUT_bit.P1
2065: // 0x02 // 0000 0010b p2  // BCD data B 7Segment LED
2066: # define  IC                   P2OUT_bit.P2
2067: // 0x04 // 0000 0100b p2  // BCD data C 7Segment LED
2068: # define  ID                   P2OUT_bit.P3
2069: // 0x08 // 0000 1000b p2  // BCD data D 7Segment LED
2070: # define  LEH                  P2OUT_bit.P4
2071: // 0x10 // 0001 0000b p2  // High Digit Enable
2072: # define  LEL                  P2OUT_bit.P5
2073: // 0x20 // 0010 0000b p2  // Low Digit Enable
2074: # define  LELL                 P3OUT_bit.P4
2075: // 0x10 // 0001 0000b p3  // LowerDigit  enable
2076: # define  LELLL                P3OUT_bit.P5
2077: // 0x20 // 0010 0000b p3  // LowestDigit  enable
2078: # define  Board_LED            P2OUT_bit.P6
2079: // 0x40 // 0100 0000b p2  // LED on Board
2080: # define  NOTSYNC              P3OUT_bit.P0
2081: // 0x01 // 0000 0001b p3  // SPI sync
2082: # define  DIN                  P3OUT_bit.P1
2083: // 0x02 // 0000 0010b p3  // DAC Data -SPI
2084: # define  SCLK                 P3OUT_bit.P3
2085: // 0x08 // 0000 1000b p3  // SPI clk
2086: # define  HeartBeat_LED        P4OUT_bit.P1
2087: // 0x02 // 0000 0010b p4  // Heart Beat LED-pwm
2088: # define  D4                   P4OUT_bit.P2   // LCD
2089: # define  D5                   P4OUT_bit.P3   // LCD
2090: # define  D6                   P4OUT_bit.P4   // LCD
2091: # define  D7                   P4OUT_bit.P5   // LCD
2092: # define  EN                   P4OUT_bit.P6   // LCD
2093: # define  RS                   P4OUT_bit.P7   // LCD
2094: // R/W of LCD pin to be connected to GND
2095: # define  XTR_Out_Disable      P5OUT_bit.P3
2096: // 0x08 // 0000 1000b p5  // Output Disable -XTR off
2097: # define  Current_Sample_Read  P6IN_bit.P0
2098: // 0x01 // 0000 0001b p6  // Output Current Sample
2099: # define  Amplitude_Set_Pot    P6IN_bit.P1
2100: // 0x02 // 0000 0010b p6  // Amplitude Set Pot
2101: # define  DC_Offset_Set_Pot    P6IN_bit.P2
2102: // 0x04 // 0000 0100b p6  // DC Offset Set Pot
2103: # define  Freq_Set_Pot         P6IN_bit.P3
2104: // 0x08 // 0000 1000b p6  // Frequency Set Pot
2105: # define  Battery_Monitor      P6IN_bit.P5
2106: // 0x20 // 0010 0000b p6  // Battery Monitor
2107: # define  p5_0                 P5OUT_bit.P0
2108: // 0x01 // 0000 0001b p5  // Buffered p5.0
2109: # define  Stimulation_Marker   P5OUT_bit.P1
2110: // 0x20 // 0010 0000b p1  // Stimulation Synch Output
2111: # define  p5_1                 P5OUT_bit.P1
2112: // 0x02 // 0000 0010b p5  // Buffered p5.1 also used for clock monitoring
2113: # define  p5_2                 P5OUT_bit.P2
2114: // 0x04 // 0000 0100b p5  // Buffered p5.2
2115: # define  p5_6                 P5IN_bit.P5
2116: // 0x20 // 0010 0000b p5  // p5.6
2117: # define  p5_7                 P5OUT_bit.P7
2118: // 0x20 // 0010 0000b p5  // p5.7 OnBoard activity Monitor
2119: # define  p4_0                 P4OUT_bit.P0
2120: // 0x80 // 0000 0001b p4  // p4.0
2121: # define  p1_0                 P1OUT_bit.P0
2122: // 0x80 // 0000 0001b p1  // p1.0
```

```
2123:
2124: void  Board_Setup (void)
2125: {
2126:   P1DIR =  p1_0_ | BZZR | BRLY | ZAP ;
2127:   P1SEL =  0x00;
2128:   P1OUT =  0x00;
2129:   P2DIR =  IA_ | IB_ | IC_ | ID_ | LEH_ | LEL_ | BLED ;
2130:   P2SEL =  0x00;
2131:   P2OUT =  0x00;
2132:   P3DIR =  NOTSYNC_ | DIN_ | SCLK_ | LELL_ | LELLL_ ;
2133:   P3SEL =  DIN_ | SCLK_ ;     // Check out!
2134:   P3OUT =  0x00;
2135:   P4DIR =  Beat | D4_ | D5_ | D6_ | D7_ | ELCD_ | RS_ ;
2136:   P4SEL =  0x00;
2137:   P4OUT =  0x00;
2138:   P5DIR =  p5_0_ | p5_1_ | p5_2_ | p5_7_ | OD ;
2139:   P5SEL =  0x00;
2140:   P5OUT =  0x00;
2141:   P6DIR =  0x00;
2142:   P6SEL =  CUSA | AMPL | DCOS | FREQ | BATT_SAMPLE ;
2143:   P6OUT =  0x00;
2144: }
2145:
2146: // ********************************************************************************** */
2147:
2148: /*This file has been prepared for Doxygen automatic documentation generation.*/
2149: /*! \file ****************************************************************
2150:  *
2151: * TESsaNova: tDCS TransCranial DC Brain Stimulator
2152: * Developed for research studeies at MAKELab
2153: *
2154: * LCD User Interface File
2155: * modified version of Andreas Dannenberg
2156: *
2157: * Adnan Kurt
2158: * MakeLAB
2159: * 19Aug. 2013
2160: * Zekeriyakoy, Istanbul
2161: * - File:               TESsaNova_Board_Definition_File_19Aug2013.c
2162: * - Compiler:           IAR EWBMSP430 5.40
2163: * - Supported devices:  MSP430F149
2164: * - Circuit:            TESsaNova_19Nov2011_F.sch
2165: *
2166: *  \author    AdKu            \n
2167: *             Adnan Kurt      \n
2168: *             MakeLAB        \n
2169: *             19Aug. 2013     \n
2170: *             Zekeriyakoy, Istanbul
2171: *
2172: * NB. This program must be updated with a better formed and documented one.
2173: *
2174: ********************************************************************************/
2175:
2176: // #include <msp430x22x4.h>
2177: // #include "Citi_Wall.h"
2178: // #include <string.h>
2179:
2180: void initDisplay();
2181: void putc(char c);
2182: void clearDisplay();
2183: void printDecimal(int Number);
2184: void printHex(unsigned int Number);
2185: void printString(char *String);
2186: void gotoSecondLine();
2187: void printByte(unsigned int theByte);
2188:
2189: #define bitset(var,bitno) ((var) |= 1 << (bitno))
2190: #define bitclr(var,bitno) ((var) &= ~(1 << (bitno)))
2191:
2192: // #define          LCD_Data           P2OUT
2193: #define          _100us             14      // 4 about 1oous  1 at 2Mhz
2194: #define          _10us              5       // 1 about 25us   1 at 2Mhz
2195: #define          EE                 3       //3        //P2.3
2196: #define          RSS                2       //2        //P2.2
2197: #define          CR                 0x0d
2198: #define          LF                 0x0a
```

```
2199: #define     DISP_ON         0x0c            //LCD control constants
2200: #define     DISP_OFF        0x08            //
2201: #define     CLR_DISP        0x01        //
2202: #define     CUR_HOME        0x02            //
2203: #define     CUR_LEFT        0x10            //
2204: #define     ENTRY_INC       0x06            //
2205: #define     DD_RAM_ADDR     0x80            //
2206: #define     DD_RAM_ADDR2        0xc0            //
2207: #define     DD_RAM_ADDR3        0x28            //
2208: #define     CG_RAM_ADDR     0x40            //
2209:
2210: int LCD_REG;
2211:
2212: void Delay (unsigned int a);
2213: void Delayx100us(unsigned char b);
2214: void SEND_CHAR (unsigned char c);
2215: void SEND_CMD (unsigned char e);
2216: void _E(void);
2217: void InitLCD(void);
2218:
2219: void LCD_Con (void) {
2220:   D7 = (LCD_REG & BIT7) / 0x4F;
2221:   D6 = (LCD_REG & BIT6) / 0x2F;
2222:   D5 = (LCD_REG & BIT5) / 0x1F;
2223:   D4 = (LCD_REG & BIT4) / 0xF;
2224:   EN = (LCD_REG & BIT3) / 0x8;
2225:   RS = (LCD_REG & BIT2) / 0x4;
2226: }
2227:
2228: void initDisplay() {
2229:     InitLCD();
2230:     clearDisplay();
2231: }
2232: void putc(char c) {
2233:     SEND_CHAR(c);
2234: }
2235: void oha(char a) {
2236: }
2237: void clearDisplay() {
2238:     SEND_CMD(CLR_DISP);
2239:     Delayx100us(10);
2240: }
2241: void Back() {
2242:     SEND_CMD(CUR_LEFT);
2243:     Delayx100us(1);
2244: }
2245: void gotoSecondLine() {
2246: //    SEND_CMD(CLR_DISP);
2247:     SEND_CMD(DD_RAM_ADDR2);
2248: }
2249: void gotoFirstLine() {          // Check the operation. I've added it. AK
2250:     SEND_CMD(DD_RAM_ADDR);
2251: }
2252: void printString(char *String) {
2253:   while(*String)
2254:     putc(*String++);
2255: }
2256: void printgString(char *Gtring) {
2257:     oha(*Gtring++);
2258:     oha(*Gtring++);
2259:     oha(*Gtring++);
2260:     oha(*Gtring++);
2261:     oha(*Gtring++);
2262:     while(*Gtring)
2263:     putc(*Gtring++);
2264: }
2265: char HexDigit(int digitvalue) {
2266:   if (digitvalue < 10)
2267:     return(digitvalue + '0');
2268:   else
2269:     return(digitvalue + 'A' - 10);
2270: }
2271: void printByte(unsigned int theByte) {
2272:   char HexBuffer[3];
2273:   HexBuffer[2] = 0;
2274:   HexBuffer[1] = HexDigit(theByte & 0x000f);
```

```
2275:   theByte = theByte >> 4;
2276:   HexBuffer[0] = HexDigit(theByte & 0x000f);
2277:   printString(HexBuffer);
2278: }
2279: void printHex(unsigned int Number) {
2280:   char HexBuffer[5];
2281:   HexBuffer[4] = 0;
2282:   HexBuffer[3] = HexDigit(Number & 0x000f);
2283:   Number = Number >> 4;
2284:   HexBuffer[2] = HexDigit(Number & 0x000f);
2285:   Number = Number >> 4;
2286:   HexBuffer[1] = HexDigit(Number & 0x000f);
2287:   Number = Number >> 4;
2288:   HexBuffer[0] = HexDigit(Number & 0x000f);
2289:   printString(HexBuffer);
2290: }
2291:
2292: void print4Decimal(int Number) {
2293:   // need to move to long int to account for
2294:   // negative 32768
2295:   char DecimalBuffer[10];
2296:   int lNumber = Number;
2297:
2298:   DecimalBuffer[9] = '\0';      // correct termination AK
2299:   DecimalBuffer[8] = (lNumber % 10)+'0';
2300:   lNumber = lNumber / 10;
2301:   DecimalBuffer[7] = (lNumber % 10)+'0';
2302:   lNumber = lNumber / 10;
2303:   DecimalBuffer[6] = (lNumber % 10)+'0';
2304:   lNumber = lNumber / 10;
2305:   DecimalBuffer[5] = (lNumber % 10)+'0';
2306:   DecimalBuffer[4] = '0';
2307:   DecimalBuffer[3] = '0';
2308:   DecimalBuffer[2] = '0';
2309:   DecimalBuffer[1] = '0';
2310:   DecimalBuffer[0] = '0';
2311:   printgString(DecimalBuffer);
2312: }
2313:
2314: void printDecimal(int Number) {
2315:   // need to move to long int to account for
2316:   // negative 32768
2317:   char DecimalBuffer[9];
2318:   long lNumber = Number;
2319:   DecimalBuffer[8] = '\0';      // correct termination AK
2320:   DecimalBuffer[7] = (lNumber % 10)+'0';
2321:   lNumber = lNumber / 10;
2322:   DecimalBuffer[6] = (lNumber % 10)+'0';
2323:   lNumber = lNumber / 10;
2324:   DecimalBuffer[5] = (lNumber % 10)+'0';
2325:   DecimalBuffer[4] = '0';
2326:   DecimalBuffer[3] = '0';
2327:   DecimalBuffer[2] = '0';
2328:   DecimalBuffer[1] = '0';
2329:   DecimalBuffer[0] = '0';
2330:   printgString(DecimalBuffer);
2331: }
2332: /*
2333: void printDecimal(int Number) {
2334:   // need to move to long int to account for
2335:   // negative 32768
2336:   char DecimalBuffer[7];
2337:   long lNumber = Number;
2338:   DecimalBuffer[6] = 0;
2339:   if (lNumber < 0) {
2340:     DecimalBuffer[0] = '-';
2341:     lNumber = -lNumber;
2342:   } else
2343:     DecimalBuffer[0] = '+';
2344:   DecimalBuffer[5] = (lNumber % 10)+'0';
2345:   lNumber = lNumber / 10;
2346:   DecimalBuffer[4] = (lNumber % 10)+'0';
2347:   lNumber = lNumber / 10;
2348:   DecimalBuffer[3] = (lNumber % 10)+'0';
2349:   lNumber = lNumber / 10;
2350:   DecimalBuffer[2] = (lNumber % 10)+'0';
```

```
2351:   lNumber = lNumber / 10;
2352:   DecimalBuffer[1] = (lNumber % 10)+'0';
2353:   printString(DecimalBuffer);
2354: }
2355:
2356: void printDecimal(int Number) {
2357:   // need to move to long int to account for
2358:   // negative 32768
2359:   // Reformulated to get only 3 digits. No sign. AK
2360:   char DecimalBuffer[3];
2361:   int lNumber = Number;
2362:   DecimalBuffer[2] = (lNumber % 10)+'0';
2363:   lNumber = lNumber / 10;
2364:   DecimalBuffer[1] = (lNumber % 10)+'0';
2365:   lNumber = lNumber / 10;
2366:   DecimalBuffer[0] = (lNumber % 10)+'0';
2367:   printString(DecimalBuffer);
2368: }
2369: */
2370:
2371: void Delay (unsigned int a)
2372: {
2373:   int k;
2374:   for (k=0 ; k != a; ++k) {
2375: /*
2376:     _NOP();
2377:     _NOP();
2378:     _NOP();
2379:     _NOP();
2380: */
2381:   }
2382: }
2383:
2384: void Delayx100us(unsigned char b)
2385: {
2386:   int j;
2387:   for (j=0; j!=b; ++j) {
2388: //  ledB ^= 1;                              // delay measurement
2389:                                             // -temporary on p1.7
2390:                                             // should read 100us pulses
2391:     Delay (_100us);
2392: }
2393: //  ledB_off();                             // delay measurement
2394: }
2395:
2396: void _E(void)
2397: {
2398:   bitset(LCD_REG,EE);       //toggle E for LCD
2399:         LCD_Con ();
2400:     Delay(_10us);
2401:     bitclr(LCD_REG,EE);
2402:         LCD_Con ();
2403: }
2404:
2405: void SEND_CHAR (unsigned char d)
2406: {
2407:         int temp;
2408:     Delayx100us(2);                 //.5ms
2409:     temp = d & 0xf0;        //get upper nibble
2410:     LCD_REG &= 0x0f;
2411:     LCD_Con ();
2412:     LCD_REG |= temp;
2413:     LCD_Con ();
2414:     bitset(LCD_REG,RSS);            //set LCD to data mode
2415:         LCD_Con ();
2416:     _E();                           //toggle E for LCD
2417:     temp = d & 0x0f;
2418:     temp = temp << 4;               //get down nibble
2419:     LCD_REG &= 0x0f;
2420:     LCD_Con ();
2421:     LCD_REG |= temp;
2422:     LCD_Con ();
2423:     bitset(LCD_REG,RSS);            //set LCD to data mode
2424:         LCD_Con ();
2425:     _E();                           //toggle E for LCD
2426: }
```

```
2427:
2428: void SEND_CMD (unsigned char e)
2429: {
2430:         int temp;
2431:         Delay(_10us);
2432:         Delay(_10us);
2433: //  Delayx100us(2);                 //10ms
2434:     temp = e & 0xf0;        //get upper nibble
2435:     LCD_REG &= 0x0f;
2436:     LCD_Con ();
2437:     LCD_REG |= temp;
2438:         LCD_Con ();                     //send CMD to LCD
2439:     bitclr(LCD_REG,RSS);        //set LCD to CMD mode
2440:         LCD_Con ();
2441:     _E();                           //toggle E for LCD
2442:     temp = e & 0x0f;
2443:     temp = temp << 4;               //get down nibble
2444:     LCD_REG &= 0x0f;
2445:     LCD_Con ();
2446:     LCD_REG |= temp;
2447:     LCD_Con ();
2448:     bitclr(LCD_REG,RSS);            //set LCD to CMD mode
2449:         LCD_Con ();
2450:     _E();                           //toggle E for LCD
2451: }
2452:
2453: void InitLCD(void)
2454: {
2455:     bitclr(LCD_REG,RSS);
2456:     LCD_Con ();
2457:     Delayx100us(25);                   //Delay 100ms
2458:     Delayx100us(25);
2459:     Delayx100us(25);
2460:     Delayx100us(25);
2461:     LCD_REG |= BIT4 | BIT5;
2462:     LCD_Con ();         //D7-D4 = 0011
2463:     LCD_REG &= ~BIT6 & ~BIT7;
2464:     LCD_Con ();
2465:     _E();                               //toggle E for LCD
2466:     Delayx100us(10);                    //10ms
2467:     _E();                               //toggle E for LCD
2468:     Delayx100us(10);                    //10ms
2469:     _E();                               //toggle E for LCD
2470:     Delayx100us(10);                    //10ms
2471:     LCD_REG &= ~BIT4;
2472:        LCD_Con ();
2473:     _E();                               //toggle E for LCD
2474:
2475:     SEND_CMD(DISP_ON);
2476:     SEND_CMD(CLR_DISP);
2477:     Delayx100us(25);
2478:     Delayx100us(25);
2479:     Delayx100us(25);
2480:     Delayx100us(25);
2481: }
2482:
2483:
2484: // ******************************************************************************** */
2485:
2486:
2487:
2488: /*This file has been prepared for Doxygen automatic documentation generation.*/
2489: /*! \file ********************************************************************
2490:  *
2491: * TESsaNova: tDCS TransCranial DC Brain Stimulator
2492: * Developed for research studeies at MAKELab
2493: *
2494: * User Interface, LCD Content File
2495: *
2496: * Adnan Kurt
2497: * MakeLAB
2498: * 19Aug. 2013
2499: * Zekeriyakoy, Istanbul
2500: * - File:               TESsaNova_Board_Definition_File_19Aug2013.c
2501: * - Compiler:           IAR EWBMSP430 5.40
2502: * - Supported devices:  MSP430F149
```

```
2503: * - Circuit:            TESsaNova_19Nov2011_F.sch
2504: *
2505: *  \author    AdKu            \n
2506: *             Adnan Kurt      \n
2507: *             MakeLAB        \n
2508: *             19Aug. 2013     \n
2509: *             Zekeriyakoy, Istanbul
2510: *
2511: *
2512: ********************************************************************************/
2513:
2514: // LCD BoardSet
2515: // Welcome Screen
2516: void Citi_Wall_0  (void)
2517: {
2518:   initDisplay();
2519:   printString("     tDCS       ");    // modulated transcranial dc stimulator
2520:   gotoSecondLine();
2521:   printString("    makeLAB     ");
2522:   wait (1000);
2523:   initDisplay();
2524:   printString(" MakeLAB  2013 ");
2525:   gotoSecondLine();
2526:   printString("   TESsaNova    ");
2527:   wait (1000);
2528: }
2529:
2530: void Citi_Wall_Params (void)        // Stimulation parameters display
2531:   // initDisplay();
2532: {                                   // 10*uA, 10*uA, 10*Hz, and s.
2533:   gotoFirstLine();                  // set values for AC stimulation current
2534:   printString("acI dcI frQ dur ");  // DC stimulation current, AC frequency,
2535:   gotoSecondLine();                 // stimulation duration;
2536:   // Displays AC current Amplitude in 10*uA
2537:   printDecimal((int)Amplitude_Set_value/10);
2538:   printString(".");
2539:   // Displays DC current Amplitude in 10*uA
2540:   printDecimal((int)DC_Offset_Set_value/10);
2541:   printString(".");
2542:   // Displays Frequency in 10*Hz
2543:   printDecimal((int)Frequency_Set_value);
2544:   printString(".");
2545:   // 10s to 3600s in Seconds, 10sec steps
2546:   printDecimal((int)Stimulation_Duration/10);
2547:   printString("0");
2548: }
2549:
2550: void Citi_Wall_Params_DC (void)     // DC Stimulation parameters display
2551:   // initDisplay();
2552: {                                   // 10*uA, 10*uA, 10*Hz, and s.
2553:   gotoFirstLine();                  // set values for AC stimulation current
2554:   printString("Current Duration");  // DC stimulation current, Duration,
2555:   gotoSecondLine();                 // stimulation duration;
2556:   // Displays DC current Amplitude in 1*uA
2557:   print4Decimal((int)DC_Offset_Set_value);
2558:   printString("uA   ");
2559:   // 10s to 3600s in Seconds, 10sec steps
2560:   printDecimal((int)Stimulation_Duration/10);
2561:   printString("0sec");
2562: }
2563:
2564: // Following piece of code kept as a programming example.
2565: /*
2566:   if (Physical_FlowRate >= 10000)
2567:   {
2568:   printString("Flow in ul/sec  ");  // Speed (ul/ min),
2569:   gotoSecondLine();
2570:   printDecimal((unsigned int)(Physical_FlowRate/1000));
2571:   printString(".");
2572:   printString("0");
2573:   printString(" ");
2574:   printThreeDecimal((unsigned int)Volume_per_Step);
2575:   printString("nlps");              // Gears (1 to 16), nanoliters per step
2576: //  printString(" Gear=");
2577: //  printShortDecimal(Gear);
2578:   }
```

```
2579:   gotoFirstLine();
2580:   if ((Physical_FlowRate < 10000) && (Physical_FlowRate >= 10))
2581:   {
2582:   printString("Flow in nl/sec  ");  // Speed (nl/ sec),
2583:   gotoSecondLine();
2584:   printDecimal((unsigned int)Physical_FlowRate);
2585:   printString(".");
2586:   printString("0");
2587:   printString(" ");
2588:   printThreeDecimal((unsigned int)Volume_per_Step);
2589:   printString("nlps");              // Gears (1 to 16), nanoliters per step
2590: //  printString(" Gear=");
2591: //  printShortDecimal(Gear);
2592:   }
2593: }
2594: */
2595:
2596: // Display during process
2597: void Citi_Wall_Loops(void)
2598: {
2599:   clearDisplay();
2600:   gotoFirstLine();
2601: //printString("acI dcI frQ dur ");        //use as a template
2602:   printString("Stimulating ");
2603:   gotoSecondLine();
2604:   printString(">  > >>> ");
2605:   printDecimal((int)Stimulation_Duration/10 );   // 10s to 3600s S c a l e to display
2606:   printString("0 s");
2607: }
2608:
2609: // Display during self testing
2610: void Citi_Wall_Testing(void)
2611: {
2612:   clearDisplay();
2613:   gotoFirstLine();
2614: //printString("acI dcI frQ dur ");        //use as a template
2615:   printString("TESsaNova  tDCS ");
2616:   gotoSecondLine();
2617:   printString("self testing... ");
2618: }
2619:
2620: // Display during clock-set
2621: void Clock_Setting (void)
2622: {
2623:   initDisplay();
2624:   gotoFirstLine();
2625:   printString("Clock Tuning");
2626:   gotoSecondLine();
2627:   printString("ooooooooooo");
2628: }
2629:
2630: // Display after saving
2631: void Parameters_Saved (void)
2632: {
2633:   initDisplay();
2634:   gotoFirstLine();
2635:   printString("Parameters  ");
2636:   gotoSecondLine();
2637:   printString("Memorized.  ");
2638: }
2639:
2640: // Display after Stimulation End
2641: void CitiWall_FinalReport (void)
2642: {
2643:   clearDisplay();
2644:   gotoFirstLine();
2645:   printString("Stimulation Done");
2646:   gotoSecondLine();
2647:   printString("<io> = ");
2648:   printDecimal((int)Last_RMS_Current);
2649:   printString(" mA");
2650:   printString("duration= ");
2651:   printDecimal((int)Duration / 10);    // 100ms to 100s S c a l e to display
2652:   printString("0");
2653:   printString("mS");
2654: }
```

```
2655:
2656: void Citi_Wall_Battery (void)
2657: {
2658:   clearDisplay();
2659:   gotoFirstLine();
2660: //printString("acI dcI frQ dur ");        //use as a template
2661:   printString("Vbattery=");
2662:   printDecimal((int)Vs_Read_value);       // 1.0 to 18.0 V to display
2663:   gotoSecondLine();
2664:   printString("Change Battery! ");
2665: }
2666:
2667: void Citi_Wall_Overcurrent_Warning (void)
2668: {
2669:   clearDisplay();
2670:   gotoFirstLine();
2671: //printString("acI dcI frQ dur ");        //use as a template
2672:   printString("acI+dcI > 4000uA");
2673:   gotoSecondLine();
2674:   printString("Revise Values!  ");
2675: }
2676: /*
2677: // Error display: get data from error_list()
2678: void Citi_Wall_Err (void)
2679: {
2680:   clearDisplay();
2681:   gotoFirstLine();
2682:   printString("Error Report: ");
2683:   gotoSecondLine();
2684:   error_list(set_fault);
2685:   printString(error_is);
2686: }
2687: */
2688:
2689: // ********************************************************************************** */
2690:
2691: //*****************************************************************************
2692: //  MSP-FET430P140 Demo - Basic Clock, Implement Auto RSEL SW FLL
2693: //
2694: //  Description: Set DCO clock to (Delta)*(4096) using software FLL. DCO clock
2695: //  is output on P5.5 as SMCLK.  DCO clock, which is the selected SMCLK source
2696: //  for timer_A is integrated over LFXT1/8 (4096) until SMCLK is is equal
2697: //  to Delta.  CCR2 captures ACLK.  To use Set_DCO Timer_A must be
2698: //  operating in continous mode.  Watch crystal for ACLK is required for
2699: //  this example.  Delta must be kept in a range that allows possible
2700: //  DCO speeds.  Minimum Delta must ensure that Set_DCO loop
2701: //  can complete within capture interval. Maximum delta can be calculated be
2702: //  f(DCOx7) / 4096.  f(DCOx7) can be found in device specific datasheet.
2703: //  ACLK = LFXT1/8 = 32768/8, MCLK = SMCLK = target DCO
2704: //  //* External watch crystal installed on XIN XOUT is required for ACLK *//
2705: //
2706: //           MSP430F149
2707: //         ---------------
2708: //     /|\|            XIN|-
2709: //      | |               | 32kHz
2710: //      --|RST        XOUT|-
2711: //        |               |
2712: //        |           P5.5|--> SMLCK = target DCO
2713: //        |           P5.6|--> ALCK = 4096
2714: //
2715: //
2716: //  M. Buccini
2717: //  Texas Instruments Inc.
2718: //  Feb 2005
2719: //  Built with CCE Version: 3.2.0 and IAR Embedded Workbench Version: 3.21A
2720: //
2721: //  Modifications done to work with PP_Controller
2722: //  A. Kurt
2723: //  30Jan2011
2724: //
2725: //*****************************************************************************
2726:
2727: // #include  <msp430x14x.h>
2728: void Set_DCO (void);
2729: void Clock_Set(void);
2730:
```

```
2731: /*
2732: void Clock_Set(void)
2733: {
2734:   DCOCTL = DCO0 + DCO1;              // Max DCO- 4.77MHz. OK. AdKu
2735:   BCSCTL1 = RSEL1 + RSEL2;           // Set to 2.02MHz
2736: }
2737: */
2738:
2739: void Clock_Set(void)
2740: {
2741:   WDTCTL = WDTPW + WDTHOLD;  // Stop WDT
2742:   P5DIR |= 0x60;             // P5.5,6 output Monitor the SMCLK during setting
2743:   P5SEL |= 0x60;             // P5.5,6 SMCLK, ACLK output
2744:   Set_DCO();
2745:   P5SEL = 0x00;         // Restore P5 port !IMPORTANT
2746:   // Restored using initial configuration data
2747:   P5DIR =  p5_0_ | p5_1_ | p5_2_ | p5_7_ | OD ;
2748:   P5SEL =  0x00;
2749:   P5OUT =  0x00;
2750: }
2751:
2752: //-------------------------------------------------------------------------
2753: void Set_DCO(void)                        // Set DCO to selected frequency
2754: //-------------------------------------------------------------------------
2755: {
2756: int Repeat = 2000;
2757: #define DELTA 977            // target DCO = DELTA*(4096)   = 4Mhz
2758: //#define DELTA 900              // target DCO = DELTA*(4096)   = 3686400
2759: //#define DELTA 256              // target DCO = DELTA*(4096)   = 1048576
2760: //#define DELTA 70               // target DCO = DELTA*(4096)   = 286720
2761: //#define DELTA 392              // target DCO = DELTA*(4096)   = 4*401408
2762: //#define DELTA 489                // target DCO = DELTA*(4096) = 2MHz
2763:
2764:   unsigned int Compare, Oldcapture = 0;
2765:
2766:   BCSCTL1 |= DIVA_3;                        // ACLK= LFXT1CLK/8
2767:   CCTL2 = CM_1 + CCIS_1 + CAP;              // CAP, ACLK
2768:   TACTL = TASSEL_2 + MC_2 + TACLR;          // SMCLK, cont-mode, clear
2769:
2770:   while (Repeat)
2771:   {
2772:   Repeat--;                                 // Must be (1) ref fet140_fll_01.c
2773:                                             // Basic Clock, Auto RSEL SW FLL
2774:     while (!(CCIFG & CCTL2));               // Wait until capture occured
2775:     CCTL2 &= ~CCIFG;                        // Capture occured, clear flag
2776:     Compare = CCR2;                         // Get current captured SMCLK
2777:     Compare = Compare - Oldcapture;         // SMCLK difference
2778:     Oldcapture = CCR2;                      // Save current captured SMCLK
2779:
2780:     if (DELTA == Compare) break;            // If equal, leave "while(1)"
2781:     else if (DELTA < Compare)               // DCO is too fast, slow it down
2782:     {
2783:       DCOCTL--;
2784:       if (DCOCTL == 0xFF)
2785:       {
2786:         if (!(BCSCTL1 == (XT2OFF + DIVA_3)))
2787:         BCSCTL1--;                          // Did DCO roll under?, Sel lower RSEL
2788:       }
2789:     }
2790:     else
2791:     {
2792:       DCOCTL++;
2793:       if (DCOCTL == 0x00)
2794:         {
2795:           if (!(BCSCTL1 == (XT2OFF + DIVA_3 + 0x07)))
2796:           BCSCTL1++;                        // Did DCO roll over? Sel higher RSEL
2797:         }
2798:     }
2799:   }
2800:   CCTL2 = 0;                                // Stop CCR2
2801:   TACTL = 0;                                // Stop Timer_A
2802: }
2803:
2804:
2805: // ******************************************************************************** */
2806:
```

```
2807:
2808: /*This file has been prepared for Doxygen automatic documentation generation.*/
2809: /*! \file ********************************************************************
2810:  *
2811: * TESsaNova: tDCS TransCranial DC Brain Stimulator
2812: * Developed for research studies at MAKELab
2813: *
2814: * DAC Drive & Wave Generation File
2815: *
2816: * Adnan Kurt
2817: * MakeLAB
2818: * 12Sep. 2013
2819: * Zekeriyakoy, Istanbul
2820: * - File:               TESsaNova_DAC_Drive19Aug2013.c
2821: * - Compiler:           IAR EWBMSP430 5.40
2822: * - Supported devices:  MSP430F149
2823: * - Circuit:            TESsaNova_19Nov2011_F.sch
2824: *
2825: *  \author    AdKu            \n
2826: *             Adnan Kurt      \n
2827: *             MakeLAB        \n
2828: *             12Sep. 2013     \n
2829: *             Zekeriyakoy, Istanbul
2830: *
2831: //
2832: //  Description: Drive the DAC8551 through SPI to Generate CuCu -
2833: //  Currunt Control Signal
2834: //  DAC output to be 0-5Vdc, and AD8603 opamp will scale it down
2835: //  to 0-2Vdc. IC801 output to be fed intoIC501B OPA2336B, which
2836: //  forms the current sink loop. It generates 0-200mA with 0-2V input.
2837: //
2838: //  SPI connection to DAC8551:
2839: //
2840: //            MSP430F149
2841: //            -----------------
2842: //        /|\|              XIN|-
2843: //         | |                 |           DAC8551
2844: //         --|RST          XOUT|-       -------------
2845: //           |             P3.0|------>|FS        OUT|--> ~ 390Hz
2846: //           |       SIMO0/P3.1|------>|DIN           |
2847: //           |       UCLK0/P3.3|------>|SCLK       CS|-|
2848: //           |                 |       |             | v
2849: //
2850: //
2851: //
2852: //
2853: //
2854: //  A.Kurt
2855: //  MakeLAB
2856: //  29Oct.2012
2857: //
2858: //  Description: USART0 in SPI mode interface to DAC8551 DAC.
2859: //  USART0 is used to transmit data to DAC, software generated frame sync
2860: //  pulse, DAC is updated inside CCR0 ISR operating in continuos mode.
2861: //  ACLK= n/a, MCLK= SMCLK= UCLK= default DCOCLK
2862: //
2863: //
2864: //  Power on Reset to Zero O/P!
2865: //  O/P Voltage: 0 to Vref
2866: //  10uS settling time
2867: //  Vout = (Din/65536)*Vref :Din -decimal equivalent of binary code
2868: //  ADR444 used as a reference. 4096mV
2869: //  DCO set to 4Mhz in Clock_Set()
2870: //
2871: //  Adnan Kurt
2872: //  MakeLAB
2873: //  Zekeriyakoy
2874: //  30Oct2012
2875: // To be done:
2876: // Generate Exponential, Linear and constant Waves. Select whichever
2877: // required with the calling function.
2878: // Wave to be generated with the given parameters. May be it would be
2879: // better to create the wave first, and then call the DAC_Write routine
2880: // So, first call CurrentWave_SetUp (Output_Current_Set), then DAC_Write()
2881: //
2882: // Some reference codes, left orphan during SW development. I keep those
```

```
2883: // here for future reference.
2884: //
2885: // Adnan Kurt
2886: // 19Dec2012
2887: //
2888: // LED measurements spend very small current, so I2m keeping them active now.
2889: // Later, could be used for other purposes anyway.
2890: // Adnan Kurt
2891: // 17Sep2013
2892: //
2893: ********************************************************************************/
2894:
2895: /*
2896: //Good for standalone DAC output
2897: void DAC_Write(int Wave_tab[])
2898: {
2899:   P3OUT |= 0x1;                             // FS set
2900:   P3OUT &= ~0x1;                            // FS reset
2901:   TXBUF0 = 0x00;
2902:     while(!(U0TCTL & 0x01))
2903:     {
2904:     }
2905:   TXBUF0 = Wave_tab[pointer] >> 8;          // Hi Byte
2906:     while(!(U0TCTL & 0x01))
2907:     {
2908:     }
2909:   TXBUF0 = Wave_tab[pointer];               // Lo Byte
2910:     while(!(U0TCTL & 0x01))
2911:     {
2912:     }
2913:   pointer++;
2914:     if (pointer >= Wave_Size)
2915:     {
2916:     pointer = 0;                            // Will not repeat. So change it!
2917:     }
2918: }
2919: */
2920:
2921: /*
2922: Another reference, with subsampling
2923: // Timer A0 interrupt service routine
2924: #pragma vector=TIMERA0_VECTOR
2925: __interrupt void Timer_A0(void)
2926: {
2927:   P3OUT |= 0x1;                             // FS set
2928:   P3OUT &= ~0x1;                            // FS reset
2929:
2930:   TXBUF0 = 0x00;
2931:   while(!(U0TCTL & 0x01)){
2932:   }
2933: //  output = Sin_tab[pointer]*amplitude/5.25;
2934:   output = Sin_tab[pointer];
2935:   TXBUF0 = output >> 8;
2936:   while(!(U0TCTL & 0x01)){
2937:   }
2938:   TXBUF0 = output;
2939:   while(!(U0TCTL & 0x01)){
2940:   }
2941:
2942:      P3OUT |= 0x1;                            // FS set
2943:      pointer++;
2944:      pointer++;
2945: //   pointer++;     // Sub Sample
2946: //   pointer++;     // With 4 ++, maximum frequency of sin wave is 436Hz.
2947:      pointer &= 0xFF;
2948: }
2949: */
2950: // ********************************************************************
2951:  /*
2952:   This switch selection is good for arbitary wave generation
2953:   however, memory management problems might arise. Instead, preset
2954:   Wave tables will be used.
2955:     switch (fMode)
2956:     {
2957:     case "DC" :
2958:       for (Time_Step = 0; Time_Step <= on_Time; Time_Step++)
```

```
2959:       {
2960:       Wave_tab [Time_Step] = (int) Output_Current_Set * 0xFFFF / 4.096;
2961:       //  Wave_tab [Time_Step] = 1;
2962:       };
2963:     break;
2964:     case "Exponential" :
2965:       for (Time_Step = 0; Time_Step <= on_Time; Time_Step++)
2966:       {
2967:       Wave_tab [Time_Step] = (int) Output_Current_Set * 0xFFFF / 4.096;
2968:       //  Wave_tab [Time_Step] = 1;
2969:       };
2970:     break;
2971:     case "Linear" :
2972:       for (Time_Step = 0; Time_Step <= on_Time; Time_Step++)
2973:       {
2974:       Wave_tab [Time_Step] = (int) Output_Current_Set * 0xFFFF / 4.096;
2975:       //  Wave_tab [Time_Step] = 1;
2976:       };
2977:     break;
2978:     case "Sine" :
2979:       for (Time_Step = 0; Time_Step <= on_Time; Time_Step++)
2980:       {
2981:       Wave_tab [Time_Step] = (int) Output_Current_Set * 0xFFFF / 4.096;
2982:       //  Wave_tab [Time_Step] = 1;
2983:       };
2984:     break;
2985:     default:
2986:       for (Time_Step = 0; Time_Step <= on_Time; Time_Step++)
2987:       {
2988:       Wave_tab [Time_Step] = (int) Output_Current_Set * 0xFFFF / 4.096;
2989:       //  Wave_tab [Time_Step] = 1;
2990:       };
2991:     break;
2992:     }
2993:   */
2994: //
2995: // ***********************************************************************
2996: //
2997: /* Peripherals Used
2998: # define  &SYNC                   P3OUT_bit.P3OUT_0   // SPI Sync
2999: # define  DIN                     P3OUT_bit.P3OUT_1   // DAC Data SPI
3000: # define  SCLK                    P3OUT_bit.P3OUT_3   // SPI Clock
3001: */
3002:
3003: /* Function Prototypes
3004: void CurrentWave_SetUp ()             // Vout generated 0-5Vdc
3005: // Wave to be generated with the given parameters. May be it would be
3006: // better to create the wave first, and then call the DAC_Write routine
3007: void DAC_Write(io_set)
3008: // So, first call Current_Set(), then DAC_Write()
3009: */
3010: //
3011: //
3012: //*****************************************************************************
3013:
3014: // variable declarations
3015: // int Time_Step = 0;
3016: // int on_Time = 100;
3017: int Wave_Size = 500;
3018: int Wave_tab [100] = {0};
3019: int sample_Wave_tab = 0;
3020: unsigned int Time_Step;                    // 16-bit value to write
3021: // experimental data
3022: // Io= DAC#/12240 mA
3023: // DAC# = Sin_Wave_Tab[step] * (12240/0xFFFF) * Io_set_Value mA)
3024: // DAC# = (Sin_Wave_Tab[step] / 5354)* Io_set_Value (uA)
3025: // DAC# = (Sin_Wave_Tab[step] / 54)* Io_set_Value (mA)*10
3026: // DAC# = (Sin_Wave_Tab[step] / 27)* Io_set_Value (mA)*5
3027: // This should be better:
3028: // DAC# = (Sin_Wave_Tab[step] / 54)* Io_set_Value (mA)*10
3029: // Scaled Amplitude is served in uA, up to 2000uA
3030: unsigned int DAC_Scale = 54;
3031: unsigned int io_set;
3032: unsigned int scaled_AC_Amplitude;
3033: unsigned int scaled_DC_Amplitude;
3034: // Smoother is the scaling variable for output ramping
```

```
3035: unsigned int Smoother = 0;
3036: int Let_Wave_Out;
3037: int SubSamples = 4;
3038:
3039: // Sinusoidal wave generated for 500 samples, max value is 0xFFFF
3040: // corresponding to 0-2PI period, with a single sine wave
3041: // Subsampling might require more data, so added 40 more
3042: const unsigned int Sin_Wave_tab[541] = {
3043: 33179,33590,34002,34413,34824,35235,35646,36056,36465,36874,
3044: 37282,37690,38096,38502,38907,39311,39714,40116,40516,40916,
3045: 41314,41711,42106,42500,42893,43284,43673,44060,44446,44830,
3046: 45211,45591,45969,46345,46719,47090,47459,47826,48191,48553,
3047: 48912,49269,49624,49976,50325,50671,51014,51355,51693,52027,
3048: 52359,52687,53013,53335,53654,53969,54282,54591,54896,55198,
3049: 55496,55791,56082,56370,56653,56933,57210,57482,57750,58015,
3050: 58275,58532,58784,59032,59276,59516,59752,59984,60211,60434,
3051: 60652,60866,61076,61281,61481,61677,61869,62056,62238,62416,
3052: 62589,62757,62921,63079,63233,63383,63527,63666,63801,63931,
3053: 64056,64175,64290,64400,64505,64605,64700,64790,64874,64954,
3054: 65029,65098,65163,65222,65276,65325,65369,65408,65441,65470,
3055: 65493,65511,65524,65532,65535,65532,65524,65511,65493,65470,
3056: 65441,65408,65369,65325,65276,65222,65163,65098,65029,64954,
3057: 64874,64790,64700,64605,64505,64400,64290,64175,64056,63931,
3058: 63801,63666,63527,63383,63233,63079,62921,62757,62589,62416,
3059: 62238,62056,61869,61677,61481,61281,61076,60866,60652,60434,
3060: 60211,59984,59752,59516,59276,59032,58784,58532,58275,58015,
3061: 57750,57482,57210,56933,56653,56370,56082,55791,55496,55198,
3062: 54896,54591,54282,53969,53654,53335,53013,52687,52359,52027,
3063: 51693,51355,51014,50671,50325,49976,49624,49269,48912,48553,
3064: 48191,47826,47459,47090,46719,46345,45969,45591,45211,44830,
3065: 44446,44060,43673,43284,42893,42500,42106,41711,41314,40916,
3066: 40516,40116,39714,39311,38907,38502,38096,37690,37282,36874,
3067: 36465,36056,35646,35235,34824,34413,34002,33590,33179,32767,
3068: 32355,31944,31532,31121,30710,30299,29888,29478,29069,28660,
3069: 28252,27844,27438,27032,26627,26223,25820,25418,25018,24618,
3070: 24220,23823,23428,23034,22641,22250,21861,21474,21088,20704,
3071: 20323,19943,19565,19189,18815,18444,18075,17708,17343,16981,
3072: 16622,16265,15910,15558,15209,14863,14520,14179,13841,13507,
3073: 13175,12847,12521,12199,11880,11565,11252,10943,10638,10336,
3074: 10038,9743,9452,9164,8881,8601,8324,8052,7784,7519,7259,7002,
3075: 6750,6502,6258,6018,5782,5550,5323,5100,4882,4668,4458,4253,
3076: 4053,3857,3665,3478,3296,3118,2945,2777,2613,2455,2301,2151,
3077: 2007,1868,1733,1603,1478,1359,1244,1134,1029,929,834,744,660,
3078: 580,505,436,371,312,258,209,165,126,93,64,41,23,10,2,0,2,10,
3079: 23,41,64,93,126,165,209,258,312,371,436,505,580,660,744,834,
3080: 929,1029,1134,1244,1359,1478,1603,1733,1868,2007,2151,2301,
3081: 2455,2613,2777,2945,3118,3296,3478,3665,3857,4053,4253,4458,
3082: 4668,4882,5100,5323,5550,5782,6018,6258,6502,6750,7002,7259,
3083: 7519,7784,8052,8324,8601,8881,9164,9452,9743,10038,10336,10638,
3084: 10943,11252,11565,11880,12199,12521,12847,13175,13507,13841,
3085: 14179,14520,14863,15209,15558,15910,16265,16622,16981,17343,
3086: 17708,18075,18444,18815,19189,19565,19943,20323,20704,21088,
3087: 21474,21861,22250,22641,23034,23428,23823,24220,24618,25018,
3088: 25418,25820,26223,26627,27032,27438,27844,28252,28660,29069,
3089: 29478,29888,30299,30710,31121,31532,31944,32355,32767,33179,
3090: 33179,33590,34002,34413,34824,35235,35646,36056,36465,36874,
3091: 37282,37690,38096,38502,38907,39311,39714,40116,40516,40916,
3092: 41314,41711,42106,42500,42893,43284,43673,44060,44446,44830,
3093: 45211,45591,45969,46345,46719,47090,47459,47826,48191,48553,
3094: };
3095:
3096: // Sigmoid wave generated for 101 samples, max value is 1000dec
3097: const unsigned int Sigmoid_Wave_tab[103] = {
3098: 1, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 27, 29, 31, 33, 35, 38,
3099: 41, 44, 48, 52, 57, 62, 69, 76, 83, 93, 103, 115, 130, 146, 165,
3100: 187, 213, 242, 276, 314, 356, 401, 450, 500, 549, 598, 643, 685,
3101: 723, 757, 786, 812, 834, 853, 869, 884, 896, 906, 916, 923, 930,
3102: 937, 942, 947, 951, 955, 958, 961, 964, 966, 968, 970, 972, 974,
3103: 975, 977, 978, 979, 980, 981, 982, 983, 984, 985, 985, 986, 987,
3104: 987, 988, 988, 989, 989, 989, 990, 990, 991, 991, 991, 991, 992,
3105: 992, 992, 992, 993, 993, 1000,
3106: };
3107:
3108: // Ramp wave generated for 500 samples, max value is 0xFFFF
3109: const unsigned int Ramp_Wave_tab[501] = {
3110: 0,131,262,393,524,655,786,917,1048,1179,1310,1441,1572,1703,
```

```
3111: 1834,1966,2097,2228,2359,2490,2621,2752,2883,3014,3145,3276,
3112: 3407,3538,3669,3801,3932,4063,4194,4325,4456,4587,4718,4849,
3113: 4980,5111,5242,5373,5504,5636,5767,5898,6029,6160,6291,6422,
3114: 6553,6684,6815,6946,7077,7208,7339,7470,7602,7733,7864,7995,
3115: 8126,8257,8388,8519,8650,8781,8912,9043,9174,9305,9437,9568,
3116: 9699,9830,9961,10092,10223,10354,10485,10616,10747,10878,11009,
3117: 11140,11272,11403,11534,11665,11796,11927,12058,12189,12320,
3118: 12451,12582,12713,12844,12975,13107,13238,13369,13500,13631,
3119: 13762,13893,14024,14155,14286,14417,14548,14679,14810,14941,
3120: 15073,15204,15335,15466,15597,15728,15859,15990,16121,16252,
3121: 16383,16514,16645,16776,16908,17039,17170,17301,17432,17563,
3122: 17694,17825,17956,18087,18218,18349,18480,18611,18743,18874,
3123: 19005,19136,19267,19398,19529,19660,19791,19922,20053,20184,
3124: 20315,20446,20577,20709,20840,20971,21102,21233,21364,21495,
3125: 21626,21757,21888,22019,22150,22281,22412,22544,22675,22806,
3126: 22937,23068,23199,23330,23461,23592,23723,23854,23985,24116,
3127: 24247,24379,24510,24641,24772,24903,25034,25165,25296,25427,
3128: 25558,25689,25820,25951,26082,26214,26345,26476,26607,26738,
3129: 26869,27000,27131,27262,27393,27524,27655,27786,27917,28048,
3130: 28180,28311,28442,28573,28704,28835,28966,29097,29228,29359,
3131: 29490,29621,29752,29883,30015,30146,30277,30408,30539,30670,
3132: 30801,30932,31063,31194,31325,31456,31587,31718,31850,31981,
3133: 32112,32243,32374,32505,32636,32767,32898,33029,33160,33291,
3134: 33422,33553,33684,33816,33947,34078,34209,34340,34471,34602,
3135: 34733,34864,34995,35126,35257,35388,35519,35651,35782,35913,
3136: 36044,36175,36306,36437,36568,36699,36830,36961,37092,37223,
3137: 37354,37486,37617,37748,37879,38010,38141,38272,38403,38534,
3138: 38665,38796,38927,39058,39189,39321,39452,39583,39714,39845,
3139: 39976,40107,40238,40369,40500,40631,40762,40893,41024,41155,
3140: 41287,41418,41549,41680,41811,41942,42073,42204,42335,42466,
3141: 42597,42728,42859,42990,43122,43253,43384,43515,43646,43777,
3142: 43908,44039,44170,44301,44432,44563,44694,44825,44957,45088,
3143: 5219,45350,45481,45612,45743,45874,46005,46136,46267,46398,
3144: 46529,46660,46791,46923,47054,47185,47316,47447,47578,47709,
3145: 47840,47971,48102,48233,48364,48495,48626,48758,48889,49020,
3146: 49151,49282,49413,49544,49675,49806,49937,50068,50199,50330,
3147: 50461,50593,50724,50855,50986,51117,51248,51379,51510,51641,
3148: 51772,51903,52034,52165,52296,52428,52559,52690,52821,52952,
3149: 53083,53214,53345,53476,53607,53738,53869,54000,54131,54262,
3150: 54394,54525,54656,54787,54918,55049,55180,55311,55442,55573,
3151: 55704,55835,55966,56097,56229,56360,56491,56622,56753,56884,
3152: 57015,57146,57277,57408,57539,57670,57801,57932,58064,58195,
3153: 58326,58457,58588,58719,58850,58981,59112,59243,59374,59505,
3154: 59636,59767,59898,60030,60161,60292,60423,60554,60685,60816,
3155: 60947,61078,61209,61340,61471,61602,61733,61865,61996,62127,
3156: 62258,62389,62520,62651,62782,62913,63044,63175,63306,63437,
3157: 63568,63700,63831,63962,64093,64224,64355,64486,64617,64748,
3158: 64879,65010,65141,65272,65403,65535,
3159: };
3160:
3161: // Generate a wave, to output at DAC
3162: // Time base should be TimerA generated PP_Loop_Timer signal
3163: // to synchronize with rest of the program.
3164: // Vout generated 0-5Vdc
3165: void CurrentWave_SetUp (void)
3166: {
3167:   P3SEL |= 0xA;                             // P3.1,3 SPI option select
3168:   P3DIR |= 0xB;                             // P3.0,1,3 output direction
3169:   P3OUT &= ~0x01;                            // FS reset
3170:   ME1 |= USPIE0;                            // Enable USART0 SPI
3171:   UCTL0 |= CHAR + SYNC + MM;                // 8-bit SPI Master **SWRST**
3172:   UTCTL0 = CKPH + CKPL + SSEL1+SSEL0+STC;   // Inv. delayed, SMCLK, 3-pin
3173:   UBR00 = 0x02;                             // SMCLK/2 for baud rate AK.
3174:   UBR10 = 0x0;                              // SMCLK/2 for baud rate AK.
3175:   UMCTL0 = 0x0;                             // Clear modulation
3176:   UCTL0 &= ~SWRST;                          // Initialize USART state machine
3177:   Time_Step = 0;                            // Clear pointer
3178: // SMCLK = 4Mhz and step size to be determined with PP_Loop_Timer at DAC output.
3179: }
3180:
3181: // DAC_Write(io_set) function receives a 16bit current value, from the main loop
3182: // and send the data to DAC. It does not take care of timing. However, pointer
3183: // addressing the value in the Wave_tab[] is incremented and reset if required.
3184: /*
3185:     case "DC" : 1
3186:     case "Exponential" : 2
```

```
3187:     case "Linear" : 3
3188:     case "Sine" : 4
3189: */
3190: void DAC_Write(int fMode, int Time_Step, int scaledAmplitude, int scaled_Offset)
3191: {
3192:   // Timing could be observed over LED p5_7 as:
3193:   // LED901 or R901
3194:   //  ___|WaveCalculations|_|SPI communications|__
3195:   // In order to measure loop period, Turn on p5_7 Board LED
3196:   p5_7 = 1;
3197:   // This section took 100uS to process case2
3198:   // This section took  56uS to process case1
3199:   switch (fMode)
3200:     {
3201:     case 1 :
3202:       // DAC# = (Sin_Wave_Tab[step] / 54)* Io_set_Value (mA)*10
3203:       // SuperPosed = DAC# + DC_Offset_Value
3204:       io_set = scaledAmplitude * (Sin_Wave_tab [Time_Step] / DAC_Scale);
3205:       io_set = io_set + scaled_Offset;
3206:     break;
3207:     case 2 :
3208:       // DAC# = (Sin_Wave_Tab[step] / 54)* Io_set_Value (mA)*10
3209:       // SuperPosed = DAC# + DC_Offset_Value
3210:       io_set = scaledAmplitude * (Sin_Wave_tab [Time_Step] / DAC_Scale);
3211:       io_set = io_set + scaled_Offset;
3212:       io_set = (io_set/100)*Smoother;
3213:     break;
3214:     case 3 :
3215:       // This is to generate constant DC current output
3216:       io_set = scaled_Offset;
3217:     break;
3218:     case 4 :
3219:       // DAC# = (Sin_Wave_Tab[step] / 54)* Io_set_Value (mA)*10
3220:       io_set = scaledAmplitude * (Sin_Wave_tab [Time_Step] / DAC_Scale);
3221:     break;
3222:     case 5 :
3223:       // To set output to Zero!
3224:       io_set = 0x00;
3225:     break;
3226:     default:
3227:       // To set output to Zero!
3228:       io_set = 0x00;
3229:     break;
3230:     }
3231:   // Setting the clock to 4MHz, achieved 18Hz in Mode4. TAU=400, minimum value.
3232:   // Took 11-12 uS to calculate in Mode2, TAU set to 400, f to 10 Hz.
3233:   // Took 11.2 uS to calculate in Mode2. TAU set t0 4000 & 20000.
3234:   // Took 29 uS to calculate in Mode2.  TAU set t0 40. Limiting with ISR
3235:   // Took 175 uS to calculate in Mode2.  TAU set t0 20. Limiting with ISR
3236:   // Took 100 uS to calculate in Mode4, TAU set to 400, f to 10 Hz.
3237:   // Took 90 uS to calculate in Mode4. TAU set t0 4000 & 20000.
3238:   // Took 241 uS to calculate in Mode4.  TAU set t0 40. Limiting with ISR
3239:   // Took 1900 uS to calculate in Mode4.  TAU set t0 20. Limiting with ISR
3240:   // In order to measure Calculation loop period, Turn off p5_7 Board LED
3241:   p5_7 = 0;
3242:   // Took 1.6uS
3243:   __no_operation();
3244:   p5_7 = 1;
3245:   P3OUT |= 0x01;                            // FS set
3246:   P3OUT &= ~0x01;                           // FS reset
3247:   TXBUF0 = 0x00;
3248:     while(!(U0TCTL & 0x01))
3249:     {
3250:     }
3251:   TXBUF0 = io_set >> 8;              // Hi Byte
3252:     while(!(U0TCTL & 0x01))
3253:     {
3254:     }
3255:   TXBUF0 = io_set;                   // Lo Byte
3256:     while(!(U0TCTL & 0x01))
3257:     {
3258:     }
3259:     // Well, whatever the frequency set value is, when CCIE is set,
3260:     // ie., ISR served, it takes longer time. When CCIE is 0,
3261:     // It takes about 12uS to do the calculation! It took 120uS with
3262:     // frequency set to 190 Hz. Yes, ISR takes rime and makes timing chaotic.
```

```
3263:     // Took 110 uS to calculate in Mode2, TAU set to 400, f to 10 Hz.
3264:     // Took 95uS to calculate in Mode2. TAU set t0 4000 & 20000.
3265:     // Took 200 uS to calculate in Mode2.  TAU set t0 40. Limiting with ISR
3266:     // Took 920 uS to calculate in Mode2.  TAU set t0 20. Limiting with ISR
3267:     // Took 189 uS to calculate in Mode4, TAU set to 400, f to 10 Hz.
3268:     // Took 220 uS to calculate in Mode4. TAU set t0 4000 & 20000.
3269:     // Took 365 uS to calculate in Mode4.  TAU set t0 40. Limiting with ISR
3270:     // Took 2800 uS to calculate in Mode4.  TAU set t0 20. Limiting with ISR
3271:     // Took 100 uS for transmission. -somewhere in between.
3272:     // In order to measure loop period, Turn off p5_7 Board LED
3273:     // Took 30 uS to process from   __no_operation(); mode2
3274:     // Took 30 uS to process from   __no_operation(); mode1
3275:     p5_7 = 0;
3276: }
3277:
3278: void Init_Output_Timing(int TAU)
3279: {
3280:   TACCTL0 = CCIE;                           // CCR0 interrupt enabled
3281: //  TACCR0 = 64-1;                          // ~ 390Hz Clock period
3282: //  Default DCO 800kHz, and with 32 samples, fDAC is 390Hz
3283:   TACCR0 = TAU-1;
3284:   TACTL = TASSEL_2 + MC_1;                  // SMCLK, Up-mode
3285: }
3286:
3287: void Halt_Output_Timing(void)
3288: {
3289:   TACCTL0 &= ~CCIE;                         // CCR0 interrupt disabled
3290:   TACCR0 = 0;                               // Reset Clock period
3291:   TACTL = TASSEL_2 + MC_0;                  // SMCLK, Stop
3292: }
3293:
3294: // This function is used for testing on human model measurements
3295: // and device testing
3296: // CuSa -output current will be measured and averaged
3297: void Test_Stimulation_Output(int AC_Amplitude, int DC_Amplitude, int Frequency, int Duration)
3298: {
3299:   // fMode is 3, DC constant current output
3300:    CurrentWave_SetUp();
3301:   // DAC# = (Sin_Wave_Tab[step] / 54)* Io_set_Value (mA/10)
3302:   // io_set = scaledAmplitude * (Sin_Wave_tab [Time_Step] / DAC_Scale);
3303:   // void DAC_Write(int fMode, int Time_Step, int Amplitude)
3304:   // scaled_AC_Amplitude = (AC_Amplitude * 10)/1000;
3305:    scaled_AC_Amplitude = AC_Amplitude/100;
3306:   // DC offset value will be calculated and added to the tabulated
3307:   // waveform data. A self correction algorithm might be implemented.
3308:   // Minimum and maximum values cause confusion. So that might be
3309:   // handled automatically.
3310:    scaled_DC_Amplitude = (DC_Amplitude/100)*(0xFFFF/DAC_Scale);
3311:   SubSamples = 4;
3312:   int TAU = (80000/ Frequency)*SubSamples ;
3313:   Let_Wave_Out = 0;
3314:   bit.TickISR = 0;
3315:   unsigned int Session_Counter = 0;
3316:   unsigned int Session_Length = 0;
3317:   unsigned int Step = 0;
3318:   Init_Output_Timing(TAU);
3319:   while (Session_Length < Duration)
3320:   {
3321:     if (bit.TickISR)
3322:     {
3323:       Session_Counter ++;
3324:       // Counts Session length in seconds. Seconds counted with
3325:       // ms pulses from TimerB ISR
3326:       if (Session_Counter >= 100)
3327:       {
3328:         Session_Counter = 0;
3329:         Session_Length ++;
3330:       }
3331:       bit.TickISR = 0;
3332:     }
3333:     if (Let_Wave_Out)
3334:     {
3335:       Step = Step + SubSamples;
3336:       Let_Wave_Out = 0;
3337:       // Error Beacon will light when there is a current drive error.
3338:       Error_Beacon = XTR_Error;
```

```
3339:       // Mode1 is required for non-corrected waveform generation.
3340:       DAC_Write(1, Step, scaled_AC_Amplitude, scaled_DC_Amplitude);
3341:       // Read Current Samples
3342:       Sense_Current();
3343:       if ((fabs (DC_Amplitude - Sense_i_value)) > (DC_Amplitude/10))
3344:       {
3345:         // Error Beacon will light when there is a current drive error.
3346:         // Error expected to be less than 10%
3347:         Error_Beacon_on();
3348:       }
3349:       else
3350:       {
3351:         Error_Beacon_off();
3352:       }
3353:       if (Step >= 500)
3354:       {
3355:         Step = 0;
3356:       }
3357:     }
3358:   }
3359:     // Graceful Ending
3360:     // Set Output to Zero!
3361:     // Resulted in 24mV output. Acceptable.
3362:       Error_Beacon_off ();
3363:       DAC_Write(5, Step, scaled_AC_Amplitude, scaled_DC_Amplitude);
3364: Halt_Output_Timing();
3365: }
3366:
3367: // This function is required for short stimulations
3368: // or whenever ramp needs to be eliminated.
3369: void Rampless_Stimulate(int AC_Amplitude, int DC_Amplitude, int Frequency, int Duration)
3370: {
3371:   // fMode is 4, Sin output
3372:   // DCO is 4Mhz, so 500 points Wave period is (4000k/TAU)/500 Hz
3373:   // TAU = 4000000/(ACfreq*500)
3374:   // int TAU = (80000/ Frequency)*SubSamples;
3375:    CurrentWave_SetUp();
3376:   // DAC# = (Sin_Wave_Tab[step] / 54)* Io_set_Value (mA/10)
3377:   // io_set = scaledAmplitude * (Sin_Wave_tab [Time_Step] / DAC_Scale);
3378:   // void DAC_Write(int fMode, int Time_Step, int Amplitude)
3379:   // scaled_AC_Amplitude = (AC_Amplitude * 10)/1000;
3380:    scaled_AC_Amplitude = AC_Amplitude/100;
3381:   // DC offset value will be calculated and added to the tabulated
3382:   // waveform data. A self correction algorithm might be implemented.
3383:   // Minimum and maximum values cause confusion. So that might be
3384:   // handled automatically.
3385:    scaled_DC_Amplitude = (DC_Amplitude/100)*(0xFFFF/DAC_Scale);
3386:   // Sampling to be redefined, in order to get better resolution
3387:   // at lower frequencies
3388:   // frequency unit will be dHz!
3389:   // So, different subsampling definitions need to be done:
3390:   if (Frequency >0 && Frequency <= 10)
3391:   {
3392:     SubSamples = 1;
3393:   }
3394:     if (Frequency >0 && Frequency <= 100)
3395:   {
3396:     SubSamples = 1;
3397:   }
3398:     if (Frequency >100 && Frequency <= 200)
3399:   {
3400:     SubSamples = 2;
3401:   }
3402:     if (Frequency >200 && Frequency <= 300)
3403:   {
3404:     SubSamples = 3;
3405:   }
3406:     if (Frequency >300 && Frequency <= 500)
3407:   {
3408:     SubSamples = 4;
3409:   }
3410:     if (Frequency >500 && Frequency <= 600)
3411:   {
3412:     SubSamples = 5;
3413:   }
3414:     if (Frequency >600 && Frequency <= 800)
```

```
3415:   {
3416:     SubSamples = 6;
3417:   }
3418:     if (Frequency >800 && Frequency <= 1000)
3419:   {
3420:     SubSamples = 7;
3421:   }
3422:     if (Frequency > 1000)
3423:   {
3424:     SubSamples = 8;
3425:   }
3426:   // Frequency accuracy is about 1%. Think about it.
3427:   // minimum frequency that will be delivered from Pot is 0.1Hz, 1dHz.
3428:   // maximum frequency to get is 80Hz, or 800dHz.
3429:   // frequency unit will be dHz!
3430:   // Minimum TAU was found to be 400, including all the overhead.
3431:   // So, an offset will help to get required frequency.
3432:   // In order to get frequencies lower than 10 dHz, oversampling required.
3433:   int TAU = (80000/ Frequency)*SubSamples ;
3434:   Let_Wave_Out = 0;
3435:   bit.TickISR = 0;
3436:   unsigned int Session_Counter = 0;
3437:   // Calculate Smoother number for Sigmoid_Length of time
3438:   unsigned int Session_Length = 0;
3439:   unsigned int Step = 0;
3440:   Init_Output_Timing(TAU);
3441:   // There happen to be a current peaking, maybe just from relay switching.
3442:   // Trying to eliminate it:
3443:   DAC_Write(1, 1, 10, 10);
3444:   while (Session_Length < Duration)
3445:   {
3446:     if (bit.TickISR)
3447:     {
3448:       Session_Counter ++;
3449:       // Counts Session length in seconds. Seconds counted with
3450:       // ms pulses from TimerB ISR
3451:       if (Session_Counter >= 1000)
3452:       {
3453:         Session_Counter = 0;
3454:         Session_Length ++;
3455:       }
3456:       bit.TickISR = 0;
3457:     }
3458:     if (Let_Wave_Out)
3459:     {
3460:       Step = Step + SubSamples;
3461:       Let_Wave_Out = 0;
3462:       // Error Beacon will light when there is a current drive error.
3463:       Error_Beacon = XTR_Error;
3464:       // Mode1 is required for non-corrected waveform generation.
3465:       DAC_Write(1, Step, scaled_AC_Amplitude, scaled_DC_Amplitude);
3466:       if (Step >= 500)
3467:       {
3468:         Step = 0;
3469:       }
3470:     }
3471:   }
3472:     // Graceful Ending
3473:     // Set Output to Zero!
3474:     // Resulted in 24mV output. Acceptable.
3475:       Error_Beacon_off ();
3476:       DAC_Write(5, Step, scaled_AC_Amplitude, scaled_DC_Amplitude);
3477: Halt_Output_Timing();
3478: }
3479:
3480: void Stimulation_Output (int AC_Amplitude, int DC_Amplitude, int Frequency, int Duration)
3481: {
3482: // fMode is 4, Sin output
3483: // DCO is 4Mhz, so 500 points Wave period is (4000k/TAU)/500 Hz
3484: // TAU = 4000000/(ACfreq*500)
3485: // int TAU = (80000/ Frequency)*SubSamples;
3486:    CurrentWave_SetUp();
3487:   // DAC# = (Sin_Wave_Tab[step] / 54)* Io_set_Value (mA/10)
3488:   // io_set = scaledAmplitude * (Sin_Wave_tab [Time_Step] / DAC_Scale);
3489:   // void DAC_Write(int fMode, int Time_Step, int Amplitude)
3490:   // scaled_AC_Amplitude = (AC_Amplitude * 10)/1000;
```

```
3491:    scaled_AC_Amplitude = AC_Amplitude/100;
3492:    // DC offset value will be calculated and added to the tabulated
3493:    // waveform data. A self correction algorithm might be implemented.
3494:    // Minimum and maximum values cause confusion. So that might be
3495:    // handled automatically.
3496:    scaled_DC_Amplitude = (DC_Amplitude/100)*(0xFFFF/DAC_Scale);
3497:   // Sampling to be redefined, in order to get better resolution
3498:   // at lower frequencies
3499:   // frequency unit will be dHz!
3500:   // So, different subsampling definitions need to be done:
3501:   if (Frequency >0 && Frequency <= 10)
3502:   {
3503:     SubSamples = 1;
3504:   }
3505:     if (Frequency >0 && Frequency <= 100)
3506:   {
3507:     SubSamples = 1;
3508:   }
3509:     if (Frequency >100 && Frequency <= 200)
3510:   {
3511:     SubSamples = 2;
3512:   }
3513:     if (Frequency >200 && Frequency <= 300)
3514:   {
3515:     SubSamples = 3;
3516:   }
3517:     if (Frequency >300 && Frequency <= 500)
3518:   {
3519:     SubSamples = 4;
3520:   }
3521:     if (Frequency >500 && Frequency <= 600)
3522:   {
3523:     SubSamples = 5;
3524:   }
3525:     if (Frequency >600 && Frequency <= 800)
3526:   {
3527:     SubSamples = 6;
3528:   }
3529:     if (Frequency >800 && Frequency <= 1000)
3530:   {
3531:     SubSamples = 7;
3532:   }
3533:     if (Frequency > 1000)
3534:   {
3535:     SubSamples = 8;
3536:   }
3537:   // Frequency accuracy is about 1%. Think about it.
3538:   // minimum frequency that will be delivered from Pot is 0.1Hz, 1dHz.
3539:   // maximum frequency to get is 80Hz, or 800dHz.
3540:   // frequency unit will be dHz!
3541:   // Minimum TAU was found to be 400, including all the overhead.
3542:   // So, an offset will help to get required frequency.
3543:   // In order to get frequencies lower than 10 dHz, oversampling required.
3544:   int TAU = (80000/ Frequency)*SubSamples ;
3545:   Let_Wave_Out = 0;
3546:   bit.TickISR = 0;
3547:   unsigned int Session_Counter = 0;
3548:   // Calculate Smoother number for Sigmoid_Length of time
3549:   unsigned int Session_Length = 0;
3550:   // Sigmoid_Multiple is the variable that holds scaling period.
3551:   // It is taken to be 150mS
3552:   unsigned int Sigmoid_Multiple = 150;
3553:   // Ramp_Count is used to track ramping period counts.
3554:   // It will count to 100 (103 indeed) to get 15sec ramp
3555:   unsigned int Ramp_Count = 0;
3556:   // Sigmoid_Counter is used to store total ramping duration
3557:   unsigned int Sigmoid_Counter = 0;
3558:   unsigned int Sigmoid_Length = 0;
3559:   // Tail_Duration is the ending down ramping of the session
3560:   unsigned int Tail_Duration = 15;
3561:   // Calculate Smoother number for Sigmoid_Length of time
3562:   unsigned int Tail_Length = 0;
3563:   unsigned int Step = 0;
3564:   Init_Output_Timing(TAU);
3565:   if (Duration < 60)
3566:   {
```

```
3567:   Rampless_Stimulate(AC_Amplitude, DC_Amplitude, Frequency, Duration);
3568:   }
3569:   else
3570:   {
3571:   // Instead of calculating 101*Sigmoid_Multiple/1000
3572:   // I will directly use 15 seconds.
3573:   Duration = Duration-Tail_Duration;
3574:   while (Session_Length < Duration)
3575:   {
3576:      // In order to measure loop period, Turn on Board_LED
3577:      // Measure over LED401 R413
3578:      Board_LED = 1;
3579:      // In order to measure loop period, Turn on p5_7 Board LED
3580:      //    p5_7 = 1;
3581:     if (bit.TickISR)
3582:     {
3583:       Session_Counter ++;
3584:       Sigmoid_Length ++;
3585:      // Counts Session length in seconds. Seconds counted with
3586:      // ms pulses from TimerB ISR
3587:       if (Session_Counter >= 1000)
3588:       {
3589:         Session_Counter = 0;
3590:         Session_Length ++;
3591:       }
3592:       bit.TickISR = 0;
3593:     }
3594:     // Calculate Smoother number for Sigmoid_Length of time
3595:     // which is 101*150ms
3596:     if (Session_Length < Tail_Duration)
3597:     {
3598:     // Count for delivering Sigmoid Data
3599:     // We have 103 data points, and willing to have 15seconds ramp.
3600:     // During start, at each 150mS period, Sigmoid Data Scaling should be done.
3601:     // Sigmoid_Multiple is the variable that holds scaling period.
3602:     // It is taken to be 150mS
3603:      if (Sigmoid_Length >= Sigmoid_Counter)
3604:       {
3605:         Ramp_Count ++;
3606:         Sigmoid_Counter = Sigmoid_Counter + Sigmoid_Multiple;
3607:         // At Sigmoid_Multiple times, get a Sigmoid_Wave data for scaling
3608:         Smoother = Sigmoid_Wave_tab [Ramp_Count];
3609:         Smoother = Smoother/10;
3610:       }
3611:     }
3612:     else
3613:     {
3614:       Smoother = 100;
3615:     }
3616:     if (Let_Wave_Out)
3617:     {
3618:       // Took 16uS from p5_7 = 1; line.
3619:       // In order to measure loop period, Turn on Board_LED
3620:       // Board_LED = 1;
3621:       // When subsampled 4 times, Frequency of the generated SineWave
3622:       // Increases to 32 Hz, Instead of 8Hz 500 data points wave.
3623:       // Not so good.
3624:       // Achieved 75 Hz by subsampling quad. DCO=4Mhz.
3625:       // Use subsampling to generate waves of a wider frequency range.
3626:       // Could achieve 930mhz with TAU=32000.
3627:       // So, Step++ could have been used to get slower and higher
3628:       // resolution waves.
3629:       Step = Step + SubSamples;
3630:       // Could achieved 85 Hz. Very Stepwise.
3631:       // Step = Step + 20;
3632:       // Got 17 Hz sine wave Mode2, no calculations. TAU=400 DCO= 2Mhz
3633:       // Got 8.6 Hz sine wave Mode4, integer division. TAU=400 DCO= 2Mhz
3634:       // Step++;
3635:       Let_Wave_Out = 0;
3636:       DAC_Write(2, Step, scaled_AC_Amplitude, scaled_DC_Amplitude);
3637:       // Error Beacon will light when there is a current drive error.
3638:       Error_Beacon = XTR_Error;
3639:       if (Step >= 500)
3640:       {
3641:         Step = 0;
3642:       }
```

```
3643:       // Took 1.316 mS when frequency is set to 197 Hz.
3644:       // Took 106 uS when frequency is set to 1 Hz.
3645:       // Took 1000uS to process
3646:       // Took about 170uS when using Mode2, no calculations.
3647:       // Took 160 uS, when Double operations were converted to
3648:       // integers and no calculation!
3649:       // Took 440 uS with Integer division for scaling.
3650:       // In order to measure loop period, Turn off Board_LED
3651:       // Board_LED = 0;
3652:     }
3653:     // Took 2.5uS from Board_LED = 0; line.
3654:     // In order to measure loop period, Turn off p5_7 Board LED
3655:     //    p5_7 = 0;
3656:     // In order to measure loop period, Turn on Board_LED
3657:     Board_LED = 0;
3658: }
3659:
3660: // Reset the variables.
3661: Session_Counter = 0;
3662: Sigmoid_Length = 0;
3663: Sigmoid_Counter = 0;
3664: Ramp_Count = 0;
3665: Smoother = 0;
3666: Error_Beacon_off ();
3667:
3668: // In order to end the session with a ramp down function, this
3669: // Tail algorithm will be used. Tail_Duration=15, normally.
3670:   while (Tail_Length < Tail_Duration)
3671:   {
3672:     if (bit.TickISR)
3673:     {
3674:       Session_Counter ++;
3675:       Sigmoid_Length ++;
3676:       // Counts Session length in seconds. Seconds counted with
3677:       // ms pulses from TimerB ISR
3678:       if (Session_Counter >= 1000)
3679:       {
3680:         Session_Counter = 0;
3681:         Tail_Length ++;
3682:       }
3683:       bit.TickISR = 0;
3684:     }
3685:     // Calculate Smoother number for Sigmoid_Length of time
3686:     // which is 101*150ms
3687:     if (Tail_Length <= 101*Sigmoid_Multiple)
3688:     {
3689:     // Count for delivering Sigmoid Data
3690:     // We have 103 data points, and willing to have 15seconds ramp.
3691:     // During start, at each 150mS period, Sigmoid Data Scaling should be done.
3692:     // Sigmoid_Multiple is the variable that holds scaling period.
3693:     // It is taken to be 150mS
3694:      if (Sigmoid_Length >= Sigmoid_Counter)
3695:       {
3696:         Ramp_Count ++;
3697:         Sigmoid_Counter = Sigmoid_Counter + Sigmoid_Multiple;
3698:         // At Sigmoid_Multiple times, get a Sigmoid_Wave data for scaling
3699:         // Reverse the Smoother selection process
3700:         // Reverse the Sigmoid function
3701:         Smoother = Sigmoid_Wave_tab [101-Ramp_Count];
3702:         Smoother = Smoother/10;
3703:       }
3704:     }
3705:     else
3706:     {
3707:       Smoother = 100;
3708:     }
3709:     if (Let_Wave_Out)
3710:     {
3711:       Step = Step + SubSamples;
3712:       Let_Wave_Out = 0;
3713:       DAC_Write(2, Step, scaled_AC_Amplitude, scaled_DC_Amplitude);
3714:       if (Step >= 500)
3715:       {
3716:         Step = 0;
3717:       }
3718:     }
```

```
3719:   }
3720:     // Graceful Ending
3721:     // Set Output to Zero!
3722:     // Resulted in 24mV output. Acceptable.
3723:       DAC_Write(5, Step, scaled_AC_Amplitude, scaled_DC_Amplitude);
3724:       Error_Beacon_off ();
3725: Halt_Output_Timing();
3726: }
3727: }
3728: // Timer A0 interrupt service routine
3729: #pragma vector=TIMERA0_VECTOR
3730: __interrupt void Timer_A(void)
3731: {
3732:   // Flag to set wave-number out
3733:   Let_Wave_Out = 1;
3734:   // Measure wave timing over p5_0 ic401 pin12
3735:   p5_1 ^= 1;
3736: }
3737:
3738: // ******************************************************************************** */
3739:
3740: //*******************************************************************
3741: //
3742: //  newEsti_FlashWrite_04Nov2012.c
3743: //
3744: //  newEsti -stimulation parameters to be written and fetched
3745: //  From the A segment of Info Memory
3746: //
3747: //  MSP430F14x Demo - Flash In-System Programming, BlockWrite
3748: //
3749: //  Description: This program first copies the FlashWrite routine to RAM, then
3750: //  erases flash seg A, then it increments all values in seg A using the 64
3751: //  byte block write mode.
3752: //
3753: //  Assumed default MCLK = DCO ~2000 kHz.
3754: //  Minimum RAM requirement = 512 bytes
3755: //
3756: //              MSP430F149
3757: //           -----------------
3758: //       /|\|              XIN|-
3759: //        | |                 |
3760: //        --|RST          XOUT|-
3761: //          |                 |
3762: //
3763: //  H. Grewal / L. Westlund
3764: //  Texas Instruments Inc.
3765: //  Jun 2006
3766: //
3767: // Adopted from Grewal&Westlund, to work with newEsti. -stimulation parameters
3768: // to be written to Info Memory. When all the A segment is used, the
3769: // Flash segment block will be erased. So, the number erase cycles attempted
3770: // to be decreased by 1/64. ReadFlash routine reads the last value written
3771: // to the flash by comparing to 0xFFFF value. An index is generated, and
3772: // a value for loops that was most recently written.
3773: //
3774: // Adnan Kurt
3775: // MakeLAB
3776: // 04Apr2011
3777: // Advise by A.Tugrul Anildi
3778: // Revised for newEsti
3779: // 05Nov2012
3780: // Adnan Kurt
3781: //
3782: //
3783: // Parameter storage reformed with a series of storing items. 7 parameters
3784: // were stored sequentially, and recovered in a reverse fashion.
3785: // Free space were checked and, serial storage and serial fetching
3786: // is done. Needs to be correctly debugged.
3787: //
3788: // 23Dec2012
3789: //
3790: // Debugged and corrected. Working fine.
3791: // AdKU
3792: // 28Dec2012
3793: //
3794: // Dangerously mixes up after filling the flash space. Have to solve the
```

```
3795: // operational fallacies by careful debugging.
3796: // Found two places where the initialization of pointers were not done
3797: // appropriately.
3798: // AdKu
3799: // 26Jun2013
3800: //
3801: //*****************************************************************************
3802:
3803: // Global variables
3804: unsigned int pi  = 0x3145;
3805: unsigned int Flash_Start= 0x1234;
3806: unsigned int Flash_End = 0xCDEF;
3807: unsigned int Test_Flash_Start;
3808: /*
3809: unsigned int Duration = 2;             // default 16-bit value to write to segment A
3810: unsigned int Laser_Current = 10;       // default 16-bit value to write to segment A
3811: unsigned int Period = 20;              // default 16-bit value to write to segment A
3812: unsigned int OnTime = 25;              // default 16-bit value to write to segment A
3813: unsigned int OffTime = 75;             // default 16-bit value to write to segment A
3814: unsigned int Compliance_Voltage = 10;  // kept for consistency.
3815: int Stimulation_Mode = 1;              // kept for consistency.
3816: */
3817: /*
3818: // To place factory data to info memory
3819: // const unsigned char port_bit @ 0x1800 = BIT0;
3820: const unsigned char _pi @ 0x1000 = 0x31;
3821: const unsigned char _pj @ 0x1001 = 0x45;
3822: const unsigned int _Flash_Start @ 0x1002 = 1234;
3823: const unsigned int _Duration @ 0x1004 = 2;
3824: const unsigned int _Laser_Current @ 0x1006 = 0;
3825: const unsigned int _Period @ 0x1008 = 10;
3826: const unsigned int _OnTime @ 0x100A = 20;
3827: const unsigned int _OffTime @ 0x100C = 80;
3828: const unsigned int _Compliance_Voltage @ 0x100E = 2;
3829: const unsigned int _Stimulation_Mode @ 0x1010 = 1;
3830: const unsigned int _Flash_End @ 0x1012 = 0xCDEF;
3831:
3832: #pragma location = 0x1000
3833: const unsigned int _pi  = 0x3145;
3834: const unsigned int _Flash_Start= 1234;
3835: const unsigned int _Duration = 2;
3836: const unsigned int _Laser_Current = 0;
3837: const unsigned int _Period = 10;
3838: const unsigned int _OnTime = 20;
3839: const unsigned int _OffTime = 80;
3840: const unsigned int _Compliance_Voltage = 2;
3841: const unsigned int _Stimulation_Mode = 1;
3842: const unsigned int _Flash_End = 0xCDEF;
3843: */
3844:
3845: short * Flash_ptr;            // Flash pointer
3846: int Pointer_First;
3847: int Pointer_Second;
3848: int m = 0;
3849: int index_LV = 0;
3850: int set_fault;
3851:
3852: // Function prototypes
3853: void FlashWrite();
3854: void FlashRead();
3855: void End_of_FlashWrite();
3856:
3857: void Memorize (short Duration, short Laser_Current, short Period,
3858:                short OnTime, short OffTime, short Compliance_Voltage,
3859:                short Stimulation_Mode)
3860: {
3861:   _DINT();                                  // Disable Interrupts
3862:     Pointer_First = 0;
3863:     Pointer_Second = 0;
3864:     m = 0;
3865:     Flash_ptr = (short *) 0x1000;
3866:     Pointer_First = *Flash_ptr;
3867:     if (Pointer_First != 0xFFFF)
3868:     {
3869:     Pointer_Second = *Flash_ptr;  // This statement was missing, I think will help.
3870:       while (Pointer_Second != 0xFFFF)      // Find the free space
```

```
3871:       {
3872:         Flash_ptr++;                    // Initialize Flash pointer
3873:     m++;
3874:         Pointer_Second = *Flash_ptr;        // Set the pointer
3875:       }
3876:       index_LV = m;
3877:     }
3878:     else
3879:     {
3880:       index_LV = 0;
3881:     }
3882:     if (index_LV >= 0x33)                   // If enough space cease to exist 96bytes
3883:     {
3884:       Flash_ptr = (short *) 0x1000;         // Initialize Flash pointer
3885:       FCTL2 = FWKEY + FSSEL_2 + FN3;        // SMCLK/16 for Flash Timing Generator
3886:       FCTL1 = FWKEY + ERASE;                // Set Erase bit
3887:       FCTL3 = FWKEY;                        // Clear Lock bit
3888:       *Flash_ptr = 0;                       // Dummy write to erase Flash segment
3889:       while(!(FCTL3 & WAIT));               // WAIT until Flash is ready
3890:       while(FCTL3 & BUSY);                  // WAIT until Flash is ready
3891:       FCTL1 = FWKEY;
3892:       FCTL3 = FWKEY + LOCK;
3893:       /*
3894:       Flash_ptr = (short *) 0x1080;         // Initialize Flash pointer
3895:       FCTL2 = FWKEY + FSSEL_2 + FN3;        // SMCLK/16 for Flash Timing Generator
3896:       FCTL1 = FWKEY + ERASE;                // Set Erase bit
3897:       FCTL3 = FWKEY;                        // Clear Lock bit
3898:       *Flash_ptr = 0;                       // Dummy write to erase Flash segment
3899:       while(!(FCTL3 & WAIT));               // WAIT until Flash is ready
3900:       while(FCTL3 & BUSY);                  // WAIT until Flash is ready
3901:       FCTL1 = FWKEY;
3902:       FCTL3 = FWKEY + LOCK;
3903:       */
3904:       _EINT();
3905:       index_LV = 0;
3906:     }
3907:   FlashWrite(0x3145, index_LV + 0);
3908:     FlashWrite(Flash_Start, index_LV + 1);
3909:       FlashWrite(Duration, index_LV + 2);
3910:         FlashWrite(Laser_Current, index_LV + 3);
3911:           FlashWrite(Period, index_LV + 4);
3912:             FlashWrite(OnTime, index_LV + 5);
3913:               FlashWrite(OffTime, index_LV + 6);
3914:                 FlashWrite(Compliance_Voltage, index_LV + 7);
3915:                   FlashWrite(Stimulation_Mode, index_LV + 8);
3916:                     FlashWrite(Flash_End, index_LV + 9);
3917:
3918:                                             // SET BREAKPOINT HERE
3919:   _EINT();                                  // Enable Interrupts
3920: }
3921:
3922: // If Info memory is free, then initialize with original data.
3923: void Test_Info_Memory (void)
3924: {
3925:      Flash_ptr = (short *) 0x1000;
3926:      Test_Flash_Start = *Flash_ptr;
3927:      if (Test_Flash_Start == 0xFFFF)
3928:      {
3929:        Memorize (Duration, Laser_Current, Period,
3930:                OnTime, OffTime, Compliance_Voltage,
3931:                Stimulation_Mode); // This must be done before anything else.
3932:      }
3933: }
3934:
3935: void Remember_Parameters(void)
3936: {
3937:     Pointer_First = 0;
3938:     Pointer_Second = 0;
3939:     m = 0;
3940:     Flash_ptr = (short *) 0x1000;
3941:     Pointer_Second = *Flash_ptr;  // This statement was missing, I think will help.
3942:     while (Pointer_Second != 0xFFFF)
3943:     {
3944:     Pointer_First = *Flash_ptr;
3945:         Flash_ptr++;                     // Initialize Flash pointer
3946:         m++;
```

```
3947:         Pointer_Second = *Flash_ptr;
3948:     }
3949:     int Test_Flash_End = Pointer_First;
3950:     Flash_ptr--;
3951:     Flash_ptr--;
3952:     Stimulation_Mode = *Flash_ptr;
3953:     Flash_ptr--;
3954:     Compliance_Voltage = *Flash_ptr;
3955:     Flash_ptr--;
3956:     OffTime = *Flash_ptr;
3957:     Flash_ptr--;
3958:     OnTime = *Flash_ptr;
3959:     Flash_ptr--;
3960:     Period = *Flash_ptr;
3961:     Flash_ptr--;
3962:     Laser_Current = *Flash_ptr;
3963:     Flash_ptr--;
3964:     Duration = *Flash_ptr;
3965:     Flash_ptr--;
3966:     int Test_Flash_Start = *Flash_ptr;
3967:
3968:     _NOP();                                 // SET BREAKPOINT HERE
3969:     if (m >= 64)
3970:     {
3971:   Beep();
3972:   wait(40);
3973:   Beep();
3974:     set_fault = 8;                       // Storage Error
3975:     }
3976:     if ((Stimulation_Mode == -1)
3977:         | (Test_Flash_End != 0xCDEF)
3978:           | (Test_Flash_Start != 0x1234))
3979:     {
3980:   Beep();
3981:   wait(40);
3982:   Beep();
3983:   wait(100);
3984:   Beep();
3985:     set_fault = 8;                       // Storage Error
3986:     }
3987: }
3988:
3989: void FlashWrite(int Memo, int index_LV)
3990: {
3991:   FCTL2 = FWKEY + FSSEL_2 + FN3;             // MCLK/2 for Flash Timing Generator
3992:   FCTL1 = FWKEY + ERASE;                      // Set Erase bit
3993:   FCTL3 = FWKEY;                              // Clear Lock bit
3994:   Flash_ptr = (short*)0x1000 + index_LV ;     // Initialize Flash pointer
3995:     if (index_LV == 0)
3996:     {
3997:         Flash_ptr = (short*)0x1000;
3998:     }                        // Initialize Flash pointer
3999:   while(FCTL3 & BUSY);                        // Check Flash BUSY bit
4000:   FCTL1 = FWKEY + WRT;                        // Enable write operation
4001:   *Flash_ptr = Memo;                          // Write value to flash
4002:   while(!(FCTL3 & WAIT));                     // WAIT until Flash is ready
4003:   FCTL1 = FWKEY;                              // Clear BLKWRT & WRT bits
4004:   while(FCTL3 & BUSY);                        // Check Flash BUSY bit
4005:   FCTL3 = FWKEY + LOCK;                       // Reset LOCK bit
4006:   return;                                     // Exits routine
4007: }
4008:
4009: void End_of_FlashWrite(){}                    // Marks end of FlashWrite
4010:
4011:
4012: // ********************************************************************************** */
4013:
4014: /*This file has been prepared for Doxygen automatic documentation generation.*/
4015: /*! \file ********************************************************************
4016:  *
4017: * TESsaNova: tDCS TransCranial DC Brain Stimulator
4018: * Developed for research studeies at MAKELab
4019: *
4020: * Parameters Initializations File
4021: *
4022: * Adnan Kurt
```

```
4023: * MakeLAB
4024: * 22Aug. 2013
4025: * Zekeriyakoy, Istanbul
4026: * - File:               TESsaNova_Initializations_19Aug2013.c
4027: * - Supported devices:  MSP430F149
4028: * - Circuit:            TESsaNova_19Nov2011_F.sch
4029: *
4030: *  \author    AdKu            \n
4031: *             Adnan Kurt      \n
4032: *             MakeLAB        \n
4033: *             19Aug. 2013     \n
4034: *             Zekeriyakoy, Istanbul
4035: *
4036: *
4037: ******************************************************************************/
4038:
4039: # define  SIZE 100  //waveform period
4040:
4041: // variable declerations
4042: # define _100us  14         // 4 about 1oous
4043: # define _10us   5          // 1 about 25us
4044: unsigned int  wait_delay;   // in 1ms units
4045: unsigned int  tick_count_1;
4046: unsigned int  tick_count_2;
4047:
4048: double Sense_i_read;
4049: double Sense_i_value;
4050: double Vs_Read_value;
4051: double Vs_Read_Read;
4052: double Board_Temperature;
4053: double NoiseTest1;
4054: double NoiseTest2;
4055: double NoiseTest3;
4056: double NoiseTest4;
4057: double NoiseTest5;
4058: void   Clock_Set (void);
4059: void   Loop_Timer(void);
4060: int    Process;
4061: int    Emergency;
4062: int    errorValue;
4063: int Stimulation_On;
4064: unsigned int tick_count;
4065: int Looping;
4066: double MaxAmplitude;
4067: double MaxDCOffset;
4068: double MaxFrequency;
4069: double MaxVoltage;
4070: double MaxCurrent;
4071: unsigned int Period;
4072: unsigned int Duration;
4073: unsigned int Compliance_Voltage;
4074: int Stimulation_Mode;
4075: int Last_RMS_Current;
4076: int Test_Amplitude;
4077: int Test_Duration;
4078:
4079: // default parameters
4080: void tDCS_Parameters (void) {
4081:   tick_count = 0;                   // TimerA loop clock count
4082:   wait_delay = 10;                  // number of TimerA IT loops to wait
4083:                     // 5.0 ms/Tick
4084:   Stimulation_On = 0;
4085:   Looping = 1;              // During Process Loops, it is 0.
4086:   MaxAmplitude = 4000.0;            // Maximum AC Stimulation Current in uA
4087:   MaxDCOffset = 4000.0;             // Maximum DC Stimulation Current in uA
4088:   MaxFrequency = 800.0;             // Maximum Stimulation Frequency in cHz
4089:   MaxVoltage = 1800;                // Maximum Supply Voltage in cV
4090:   MaxCurrent = 5000.0;              // Maximum Output Current Read in uA
4091:
4092:   Compliance_Voltage = 20;    // Kept for Consistency in Flash
4093:   Stimulation_Mode = 1;       // Kept for Consistency in Flash
4094: }
4095:
4096: volatile struct
4097: {
4098:   unsigned char TickISR:1;          // ISR Handling
```

```
4099: } bit;
4100:
4101: void wait (int wait_delay){        // 1 ms/tick set in PP_Loop_Timer.c
4102: //    ledB ^= 1;                        // delay measurement
4103:                                       // -temporary on p1.6 check from the list
4104:     tick_count_1 = tick_count;
4105:     tick_count_2 = tick_count-tick_count_1;
4106:     while (tick_count_2 < wait_delay){
4107:     tick_count_2 = tick_count-tick_count_1;
4108:   }
4109: //    ledB = 1;                         // delay measurement
4110: }
4111:
4112: void Beep (int length, int freq)
4113: {
4114:   int l = length;
4115:   while ( l >1 )
4116:   {
4117:     Buzzer ^= 1;
4118:     wait ( freq );
4119:     l --;
4120:   }
4121: }
4122:
4123: void Error_Beacon_on (void)
4124: {
4125:   // LED is connected to inverting output of CD4049 IC601
4126:   Error_Beacon = 0;
4127: }
4128:
4129: void Error_Beacon_off (void)
4130: {
4131:   // LED is connected to inverting output of CD4049 IC601
4132:   Error_Beacon = 1;
4133: }
4134:
4135: // ***************************************************************************** */
4136:
4137: //*********************************************************************//
4138: //
4139: // newEsti_Musica_29Dec2012.c
4140: // Buzzer tone generator
4141: //
4142: // Adnan Kurt
4143: // 11Aug. 2012
4144: // Based on work shared at:
4145: // https://forum.43oh.com/topic/361-playing-music/
4146: //
4147: // Modified, simplified to work with newEsti.
4148: // AdKu
4149: // 28Dec2012
4150: // MakeLAB
4151: // Etiler
4152: //
4153: // Works best with Murata, 1" piezo transducer.
4154: // Adnan Kurt
4155: // MakeLAB
4156: // 29Dec2012
4157: // Zekeriyakoy Istanbul
4158: //
4159: /*********************************************************************/
4160:
4161: /*
4162: //#include <msp430g2211.h>
4163: //#include <signal.h>
4164: # include <io430x14x.h>
4165: # include <in430.h>
4166: # include <math.h>
4167: # include <stdint.h>
4168: # include <Music_Clock_Set.c>
4169: */
4170:
4171: typedef unsigned char byte;
4172:
4173: #define NOTE_B0  31
4174: #define NOTE_C1  33
```

```
4175: #define NOTE_CS1 35
4176: #define NOTE_D1  37
4177: #define NOTE_DS1 39
4178: #define NOTE_E1  41
4179: #define NOTE_F1  44
4180: #define NOTE_FS1 46
4181: #define NOTE_G1  49
4182: #define NOTE_GS1 52
4183: #define NOTE_A1  55
4184: #define NOTE_AS1 58
4185: #define NOTE_B1  62
4186: #define NOTE_C2  65
4187: #define NOTE_CS2 69
4188: #define NOTE_D2  73
4189: #define NOTE_DS2 78
4190: #define NOTE_E2  82
4191: #define NOTE_F2  87
4192: #define NOTE_FS2 93
4193: #define NOTE_G2  98
4194: #define NOTE_GS2 104
4195: #define NOTE_A2  110
4196: #define NOTE_AS2 117
4197: #define NOTE_B2  123
4198: #define NOTE_C3  131
4199: #define NOTE_CS3 139
4200: #define NOTE_D3  147
4201: #define NOTE_DS3 156
4202: #define NOTE_E3  165
4203: #define NOTE_F3  175
4204: #define NOTE_FS3 185
4205: #define NOTE_G3  196
4206: #define NOTE_GS3 208
4207: #define NOTE_A3  220
4208: #define NOTE_AS3 233
4209: #define NOTE_B3  247
4210: #define NOTE_C4  262
4211: #define NOTE_CS4 277
4212: #define NOTE_D4  294
4213: #define NOTE_DS4 311
4214: #define NOTE_E4  330
4215: #define NOTE_F4  349
4216: #define NOTE_FS4 370
4217: #define NOTE_G4  392
4218: #define NOTE_GS4 415
4219: #define NOTE_A4  440
4220: #define NOTE_AS4 466
4221: #define NOTE_B4  494
4222: #define NOTE_C5  523
4223: #define NOTE_CS5 554
4224: #define NOTE_D5  587
4225: #define NOTE_DS5 622
4226: #define NOTE_E5  659
4227: #define NOTE_F5  698
4228: #define NOTE_FS5 740
4229: #define NOTE_G5  784
4230: #define NOTE_GS5 831
4231: #define NOTE_A5  880
4232: #define NOTE_AS5 932
4233: #define NOTE_B5  988
4234: #define NOTE_C6  1047
4235: #define NOTE_CS6 1109
4236: #define NOTE_D6  1175
4237: #define NOTE_DS6 1245
4238: #define NOTE_E6  1319
4239: #define NOTE_F6  1397
4240: #define NOTE_FS6 1480
4241: #define NOTE_G6  1568
4242: #define NOTE_GS6 1661
4243: #define NOTE_A6  1760
4244: #define NOTE_AS6 1865
4245: #define NOTE_B6  1976
4246: #define NOTE_C7  2093
4247: #define NOTE_CS7 2217
4248: #define NOTE_D7  2349
4249: #define NOTE_DS7 2489
4250: #define NOTE_E7  2637
```

```
4251: #define NOTE_F7  2794
4252: #define NOTE_FS7 2960
4253: #define NOTE_G7  3136
4254: #define NOTE_GS7 3322
4255: #define NOTE_A7  3520
4256: #define NOTE_AS7 3729
4257: #define NOTE_B7  3951
4258: #define NOTE_C8  4186
4259: #define NOTE_CS8 4435
4260: #define NOTE_D8  4699
4261: #define NOTE_DS8 4978
4262:
4263: #define OCTAVE_OFFSET 0
4264:
4265: int notes[] = { 0,
4266: NOTE_C4, NOTE_CS4, NOTE_D4, NOTE_DS4, NOTE_E4, NOTE_F4, NOTE_FS4, NOTE_G4, NOTE_GS4, NOTE_A4, NOTE_AS4, NOTE_B4,
4267: NOTE_C5, NOTE_CS5, NOTE_D5, NOTE_DS5, NOTE_E5, NOTE_F5, NOTE_FS5, NOTE_G5, NOTE_GS5, NOTE_A5, NOTE_AS5, NOTE_B5,
4268: NOTE_C6, NOTE_CS6, NOTE_D6, NOTE_DS6, NOTE_E6, NOTE_F6, NOTE_FS6, NOTE_G6, NOTE_GS6, NOTE_A6, NOTE_AS6, NOTE_B6,
4269: NOTE_C7, NOTE_CS7, NOTE_D7, NOTE_DS7, NOTE_E7, NOTE_F7, NOTE_FS7, NOTE_G7, NOTE_GS7, NOTE_A7, NOTE_AS7, NOTE_B7
4270: };
4271:
4272: //char *song = "The Simpsons:d=4,o=5,b=160:c.6,e6,f#6,8a6,g.6,e6,c6,8a,8f#,8f#,8f#,2g,8p,8p,8f#,8f#,8f#,8g,a#.,
      8c6,8c6,8c6,c6";
4273: //char *song = "Indiana:d=4,o=5,b=250:e,8p,8f,8g,8p,1c6,8p.,d,8p,8e,1f,p.,g,8p,8a,8b,8p,1f6,p,a,8p,8b,2c6,2d6,
      2e6,e,8p,8f,8g,8p,1c6,p,d6,8p,8e6,1f.6,g,8p,8g,e.6,8p,d6,8p,8g,e.6,8p,d6,8p,8g,f.6,8p,e6,8p,8d6,2c6";
4274: //char *song = "TakeOnMe:d=4,o=4,b=160:8f#5,8f#5,8f#5,8d5,8p,8b,8p,8e5,8p,8e5,8p,8e5,8g#5,8g#5,8a5,8b5,8a5,8a5,
      8a5,8e5,8p,8d5,8p,8f#5,8p,8f#5,8p,8f#5,8e5,8e5,8f#5,8e5,8f#5,8f#5,8f#5,8d5,8p,8b,8p,8e5,8p,8e5,8p,8e5,8g#5,8g#5,
      8a5,8b5,8a5,8a5,8a5,8e5,8p,8d5,8p,8f#5,8p,8f#5,8p,8f#5,8e5,8e5";
4275: //char *song = "Entertainer:d=4,o=5,b=140:8d,8d#,8e,c6,8e,c6,8e,2c.6,8c6,8d6,8d#6,8e6,8c6,8d6,e6,8b,d6,2c6,p,8d,
      8d#,8e,c6,8e,c6,8e,2c.6,8p,8a,8g,8f#,8a,8c6,e6,8d6,8c6,8a,2d6";
4276: //char *song = "Muppets:d=4,o=5,b=250:c6,c6,a,b,8a,b,g,p,c6,c6,a,8b,8a,8p,g.,p,e,e,g,f,8e,f,8c6,8c,8d,e,8e,8e,
      8p,8e,g,2p,c6,c6,a,b,8a,b,g,p,c6,c6,a,8b,a,g.,p,e,e,g,f,8e,f,8c6,8c,8d,e,8e,d,8d,c";
4277: //char *song = "Xfiles:d=4,o=5,b=125:e,b,a,b,d6,2b.,1p,e,b,a,b,e6,2b.,1p,g6,f#6,e6,d6,e6,2b.,1p,g6,f#6,e6,d6,
      f#6,2b.,1p,e,b,a,b,d6,2b.,1p,e,b,a,b,e6,2b.,1p,e6,2b.";
4278: //char *song = "Looney:d=4,o=5,b=140:32p,c6,8f6,8e6,8d6,8c6,a.,8c6,8f6,8e6,8d6,8d#6,e.6,8e6,8e6,8c6,8d6,8c6,8e6,
      8c6,8d6,8a,8c6,8g,8a#,8a,8f";
4279: //char *song = "20thCenFox:d=16,o=5,b=140:b,8p,b,b,2b,p,c6,32p,b,32p,c6,32p,b,32p,c6,32p,b,8p,b,b,b,32p,b,32p,b,
      32p,b,32p,b,32p,b,32p,b,32p,g#,32p,a,32p,b,8p,b,b,2b,4p,8e,8g#,8b,1c#6,8f#,8a,8c#6,1e6,8a,8c#6,8e6,1e6,8b,8g#,
      8a,2b";
4280: //char *song = "Bond:d=4,o=5,b=80:32p,16c#6,32d#6,32d#6,16d#6,8d#6,16c#6,16c#6,16c#6,16c#6,32e6,32e6,16e6,8e6,
      16d#6,16d#6,16d#6,16c#6,32d#6,32d#6,16d#6,8d#6,16c#6,16c#6,16c#6,16c#6,32e6,32e6,16e6,8e6,16d#6,16d6,16c#6,
      16c#7,c.7,16g#6,16f#6,g#.6";
4281: //char *song = "MASH:d=8,o=5,b=140:4a,4g,f#,g,p,f#,p,g,p,f#,p,2e.,p,f#,e,4f#,e,f#,p,e,p,4d.,p,f#,4e,d,e,p,d,p,e,
      p,d,p,2c#.,p,d,c#,4d,c#,d,p,e,p,4f#,p,a,p,4b,a,b,p,a,p,b,p,2a.,4p,a,b,a,4b,a,b,p,2a.,a,4f#,a,b,p,d6,p,4e.6,d6,b,
      p,a,p,2b";
4282: //char *song = "StarWars:d=4,o=5,b=45:32p,32f#,32f#,32f#,8b.,8f#.6,32e6,32d#6,32c#6,8b.6,16f#.6,32e6,32d#6,
      32c#6,8b.6,16f#.6,32e6,32d#6,32e6,8c#.6,32f#,32f#,32f#,8b.,8f#.6,32e6,32d#6,32c#6,8b.6,16f#.6,32e6,32d#6,32c#6,
      8b.6,16f#.6,32e6,32d#6,32e6,8c#6";
4283: //char *song = "GoodBad:d=4,o=5,b=56:32p,32a#,32d#6,32a#,32d#6,8a#.,16f#.,16g#.,d#,32a#,32d#6,32a#,32d#6,8a#.,
      16f#.,16g#.,c#6,32a#,32d#6,32a#,32d#6,8a#.,16f#.,32f.,32d#.,c#,32a#,32d#6,32a#,32d#6,8a#.,16g#.,d#";
4284: //char *song = "TopGun:d=4,o=4,b=31:32p,16c#,16g#,16g#,32f#,32f,32f#,32f,16d#,16d#,32c#,32d#,16f,32d#,32f,16f#,
      32f,32c#,16f,d#,16c#,16g#,16g#,32f#,32f,32f#,32f,16d#,16d#,32c#,32d#,16f,32d#,32f,16f#,32f,32c#,g#";
4285: //char *song = "A-Team:d=8,o=5,b=125:4d#6,a#,2d#6,16p,g#,4a#,4d#.,p,16g,16a#,d#6,a#,f6,2d#6,16p,c#.6,16c6,16a#,
      g#.,2a#";
4286: //char *song = "Flinstones:d=4,o=5,b=40:32p,16f6,16a#,16a#6,32g6,16f6,16a#.,16f6,32d#6,32d6,32d6,32d#6,32f6,
      16a#,16c6,d6,16f6,16a#.,16a#6,32g6,16f6,16a#.,32f6,32f6,32d#6,32d6,32d6,32d#6,32f6,16a#,16c6,a#,16a6,16d.6,
      16a#6,32a6,32a6,32g6,32f#6,32a6,8g6,16g6,16c.6,32a6,32a6,32g6,32g6,32f6,32e6,32g6,8f6,16f6,16a#.,16a#6,32g6,
      16f6,16a#.,16f6,32d#6,32d6,32d6,32d#6,32f6,16a#,16c.6,32d6,32d#6,32f6,16a#,16c.6,32d6,32d#6,32f6,16a#6,16c7,
      8a#.6";
4287: //char *song = "Jeopardy:d=4,o=6,b=125:c,f,c,f5,c,f,2c,c,f,c,f,a.,8g,8f,8e,8d,8c#,c,f,c,f5,c,f,2c,f.,8d,c,a#5,
      a5,g5,f5,p,d#,g#,d#,g#5,d#,g#,2d#,d#,g#,d#,g#,c.7,8a#,8g#,8g,8f,8e,d#,g#,d#,g#5,d#,g#,2d#,g#.,8f,d#,c#,c,p,a#5,
      p,g#.5,d#,g#";
4288: //char *song = "Gadget:d=16,o=5,b=50:32d#,32f,32f#,32g#,a#,f#,a,f,g#,f#,32d#,32f,32f#,32g#,a#,d#6,4d6,32d#,32f,
      32f#,32g#,a#,f#,a,f,g#,f#,8d#";
4289: //char *song = "Smurfs:d=32,o=5,b=200:4c#6,16p,4f#6,p,16c#6,p,8d#6,p,8b,p,4g#,16p,4c#6,p,16a#,p,8f#,p,8a#,p,4g#,
      4p,g#,p,a#,p,b,p,c6,p,4c#6,16p,4f#6,p,16c#6,p,8d#6,p,8b,p,4g#,16p,4c#6,p,16a#,p,8b,p,8f,p,4f#";
4290: //char *song = "MahnaMahna:d=16,o=6,b=125:c#,c.,b5,8a#.5,8f.,4g#,a#,g.,4d#,8p,c#,c.,b5,8a#.5,8f.,g#.,8a#.,4g,8p,
      c#,c.,b5,8a#.5,8f.,4g#,f,g.,8d#.,f,g.,8d#.,f,8g,8d#.,f,8g,d#,8c,a#5,8d#.,8d#.,4d#,8d#.";
4291: //char *song = "LeisureSuit:d=16,o=6,b=56:f.5,f#.5,g.5,g#5,32a#5,f5,g#.5,a#.5,32f5,g#5,32a#5,g#5,8c#.,a#5,32c#,
      a5,a#.5,c#.,32a5,a#5,32c#,d#,8e,c#.,f.,f.,f.,f,32e,d#,8d,a#.5,e,32f,e,32f,c#,d#.,c#";
4292: //char *song = "MissionImp:d=16,o=6,b=95:32d,32d#,32d,32d#,32d,32d#,32d,32d#,32d,32d,32d#,32e,32f,32f#,32g,g,8p,
      g,8p,a#,p,c7,p,g,8p,g,8p,f,p,f#,p,g,8p,g,8p,a#,p,c7,p,g,8p,g,8p,f,p,f#,p,a#,g,2d,32p,a#,g,2c#,32p,a#,g,2c,a#5,
      8c,2p,32p,a#5,g5,2f#,32p,a#5,g5,2f,32p,a#5,g5,2e,d#,8d";
4293:
4294: volatile unsigned int time = 0;
4295: #define isdigit(n) (n >= '0' && n <= '9')
```

```
4296:
4297: /** Delay function. **/
4298: void delay(unsigned int ms)
4299: {
4300:   unsigned int i, ms2;
4301:   unsigned int tt=0;
4302:   i = time;
4303:   ms2 = ms*2;
4304:   while (tt < ms2)
4305:   {
4306:     tt = time-i;
4307:     int nop = 567 * 54;
4308:     nop = nop / 2;
4309:   }
4310: }
4311:
4312: void play(unsigned int hz){
4313:   CCR0 = (1000000/hz) -1;
4314:   CCR1 = (1000000/hz)/2;
4315:   TACTL = TASSEL_2 + ID_1 + MC_1;
4316:   // Added ID_1, to divide by two, SMCLK is 4Mhz.
4317: }
4318:
4319: void stop(){
4320:   TACTL = TASSEL_2 + ID_1 + MC_3;  //stop
4321:   CCR0 = 0;
4322:   // Added ID_1, to divide by two, SMCLK is 4Mhz.
4323: }
4324:
4325: int Play_it_Sam (char *song)
4326: {
4327:   WDTCTL = WDTPW+WDTTMSEL+WDTCNTCL+WDTIS1; // Set interval mode, set to zero and interval to 0.5 ms
4328:   IE1 |= WDTIE; // Enable WDT interrupt
4329:   __bis_SR_register( __SR_GIE );
4330:
4331: //  TA0 pins: 14 p1.2 18 p1.6 23 p2.3
4332: //  P2DIR |= BIT3; // P2.3 to output
4333:   P1DIR |= BIT2;   // P1.2 to output  amplified buzzer
4334:   P1OUT &= ~BIT2;  // pull BIT2 down
4335:   P1SEL |= BIT2;   // P1.2 to TA0.1
4336:   P3SEL =  0x00 ;  // Disconnect DAC SPI!
4337:
4338:   CCTL1 = OUTMOD_7; // CCR1 reset/set
4339:
4340:   byte default_dur = 4;
4341:   byte default_oct = 6;
4342:   int bpm = 63;
4343:   int num;
4344:   long wholenote;
4345:   long duration;
4346:   byte note;
4347:   byte scale;
4348:   char *p=song;
4349:
4350:   while(*p != ':') p++;    // ignore name
4351:   p++;                     // skip ':'
4352:
4353:   // get default duration
4354:   if(*p == 'd')
4355:   {
4356:     p++; p++;              // skip "d="
4357:     num = 0;
4358:     while(isdigit(*p))
4359:     {
4360:       num = (num * 10) + (*p++ - '0');
4361:     }
4362:     if(num > 0) default_dur = num;
4363:     p++;                   // skip comma
4364:   }
4365:
4366:   // get default octave
4367:   if(*p == 'o')
4368:   {
4369:     p++; p++;              // skip "o="
4370:     num = *p++ - '0';
4371:     if(num >= 3 && num <=7) default_oct = num;
```

```
4372:     p++;                    // skip comma
4373:   }
4374:
4375:   // get BPM
4376:   if(*p == 'b')
4377:   {
4378:     p++; p++;               // skip "b="
4379:     num = 0;
4380:     while(isdigit(*p))
4381:     {
4382:       num = (num * 10) + (*p++ - '0');
4383:     }
4384:     bpm = num;
4385:     p++;                    // skip colon
4386:   }
4387:
4388:   // BPM usually expresses the number of quarter notes per minute
4389:   wholenote = (60 * 1000L / bpm) * 4;  // this is the time for whole note (in milliseconds)
4390:
4391:   // now begin note loop
4392:   while(*p)
4393:   {
4394:     // first, get note duration, if available
4395:     num = 0;
4396:     while(isdigit(*p))
4397:     {
4398:       num = (num * 10) + (*p++ - '0');
4399:     }
4400:
4401:     if(num) duration = wholenote / num;
4402:     else duration = wholenote / default_dur;  // we will need to check if we are a dotted note after
4403:
4404:     // now get the note
4405:     note = 0;
4406:
4407:     switch(*p)
4408:     {
4409:       case 'c':
4410:         note = 1;
4411:         break;
4412:       case 'd':
4413:         note = 3;
4414:         break;
4415:       case 'e':
4416:         note = 5;
4417:         break;
4418:       case 'f':
4419:         note = 6;
4420:         break;
4421:       case 'g':
4422:         note = 8;
4423:         break;
4424:       case 'a':
4425:         note = 10;
4426:         break;
4427:       case 'b':
4428:         note = 12;
4429:         break;
4430:       case 'p':
4431:       default:
4432:         note = 0;
4433:     }
4434:     p++;
4435:
4436:     // now, get optional '#' sharp
4437:     if(*p == '#')
4438:     {
4439:       note++;
4440:       p++;
4441:     }
4442:
4443:     // now, get optional '.' dotted note
4444:     if(*p == '.')
4445:     {
4446:       duration += duration/2;
4447:       p++;
```

```
4448:     }
4449:
4450:     // now, get scale
4451:     if(isdigit(*p))
4452:     {
4453:       scale = *p - '0';
4454:       p++;
4455:     }
4456:     else
4457:     {
4458:       scale = default_oct;
4459:     }
4460:
4461:     scale += OCTAVE_OFFSET;
4462:
4463:     if(*p == ',')
4464:       p++;       // skip comma for next note (or we may be at the end)
4465:
4466:     // now play the note
4467:
4468:     if(note)
4469:     {
4470:       play(notes[(scale - 4) * 12 + note]);
4471:       delay(duration);
4472:       stop();
4473:     }
4474:     else
4475:     {
4476:       delay(duration);
4477:     }
4478:   }
4479:   WDTCTL = WDTPW + WDTHOLD;             // Stop WDT
4480: // Reconnect DAC SPI:
4481:   P3SEL =  DIN_ | SCLK_ ;    // Check out!
4482:   P1DIR |= BIT2;   // P1.2 to output  amplified buzzer
4483:   P1OUT &= ~BIT2;  // pull BIT2 down
4484:   P1SEL &= ~BIT2;  // P1.2 disconnect from TA0.1
4485: //  __bic_SR_register( __SR_GIE );
4486: //  _BIS_SR(LPM4_bits);                 // Do not Sleep!
4487:   return 0;
4488: }
4489:
4490: #pragma vector = WDT_VECTOR
4491: __interrupt void watchdog_timer (void)  //__interrupt void watchdog_timer
4492: {
4493:   time++;
4494: }
4495:
4496: // ******************************************************************************** */
4497:
4498: /*This file has been prepared for Doxygen automatic documentation generation.*/
4499: /*! \file ********************************************************************
4500: *
4501: * TESsaNova: tDCS TransCranial DC Brain Stimulator
4502: * Developed for research studeies at MAKELab
4503: *
4504: * Adnan Kurt
4505: * MakeLAB
4506: * 19Aug. 2013
4507: * Zekeriyakoy, Istanbul
4508: *
4509: //  PP_Loop_Timer
4510: //  Timer_B, Toggle P4.0-3, Cont. Mode ISR, DCO SMCLK
4511: //
4512: //  Use Timer_B CCRx units and overflow to generate four
4513: //  independent timing intervals. not used: {For demonstration, CCR0, CCR1 and CCR2
4514: //  output units are optionally selected with port pins P4.1, P4.2 and P4.3
4515: //  in toggle mode}. As such, these pins will toggle when respective CCRx
4516: //  registers match the TAR counter. Interrupts are also enabled with all
4517: //  CCRx units, software loads offset to next interval only - as long as the
4518: //  interval offset is added to CCRx, toggle rate is generated in hardware.
4519: //  Timer_B overflow ISR is used to toggle P4.0 with software. Proper use of
4520: //  the TBIV interrupt vector generator is demonstrated.
4521: //  ACLK = n/a, MCLK = SMCLK = TACLK = default DCO ~800kHz
4522: //  AdKu Made DCO = 2000 kHz. AdKu ACLK = 4kHz  (32kHz/8)
4523: //
```

```
4524: //  As coded with TBCLK ~2000kHz DCO, toggle rates are:
4525: //  P4.0= CCR0 = 2000kHz/(2*200)  ~5kHz     -Measured 4,88 kHz
4526: //  P4.1= CCR1 = 2000kHz/(2*5000) ~200Hz    -Measured 195,3 Hz
4527: //  P4.2= CCR2 = 2000kHz/(2*1000) ~1000Hz   -Measured 980,3 Hz   OK! AdKu
4528: //  1 ms CCR2 pulsewidths
4529: //  P4.6= overflow = 2000kHz/(2*65536) ~15Hz
4530: //  -Measured 342 Hz (interfering with CitiWall Code) Measured 15Hz clean!
4531: //
4532: //              MSP430F149
4533: //           -----------------
4534: //       /|\|              XIN|-
4535: //        | |                 |
4536: //        --|RST          XOUT|-
4537: //          |                 |
4538: //          |         P4.0/Tb0|--> CCR0              pin 36
4539: //          |         P4.1/Tb1|--> CCR1              pin 37
4540: //          |         P4.2/Tb2|--> CCR2              pin 38
4541: //          |             P4.6|--> Overflow/software pin 42
4542: //  Adopted from:
4543: //  M. Buccini
4544: //  Texas Instruments Inc.
4545: //  Feb 2005
4546: //
4547: // This version of Loop Timer runs on TimerB. DAC_Write function relies on
4548: // TimerA interrupts, so this might be a better way to isolate the problems.
4549: // Adnan Kurt
4550: // MakeLAB
4551: // 12Feb.2012
4552: // Etiler
4553: //
4554: // I've retouched the code and the comments.
4555: // 10Sep2013
4556: // AdKu
4557: //
4558: // Scaled with 2 for 4Mhz clock
4559: //
4560: ******************************************************************************/
4561:
4562: //#include  <msp430x14x.h>
4563: int main (void);
4564: void heart_beat (void);
4565: void HB_on (void);
4566: void HB_off (void);
4567: void EmergencyKey (void);
4568: int EmergencyStopOld;
4569: int EmergencyStopNew;
4570: int Emergency;
4571: unsigned int hr_count;
4572: // unsigned int tick_count;
4573:
4574: void HB_on      (void){
4575:   HeartBeat_LED  = 0;
4576: }
4577: void HB_off         (void){
4578:   HeartBeat_LED  = 1;
4579: }
4580:
4581: void Loop_Timer(void)
4582: {
4583:                         // P4 o/p are good for debugging!
4584: //  WDTCTL = WDTPW + WDTHOLD;              // Stop WDT
4585: //  P4SEL |= 0x0F;                        // P4.0 - P4.3 option select
4586: //  P4DIR |= 0xFF;                        // P4.0 - P4.7 outputs
4587:
4588:   TBCCTL0 = OUTMOD_4 + CCIE;              // CCR0 toggle, interrupt enabled
4589:   TBCCTL1 = OUTMOD_4 + CCIE;              // CCR1 toggle, interrupt enabled
4590:   TBCCTL2 = OUTMOD_4 + CCIE;              // CCR2 toggle, interrupt enabled
4591:   TBCTL = TBSSEL_2 +  MC_2 + TBIE;        // SMCLK, Contmode, int enabled
4592:
4593:   EmergencyStopOld = 0;
4594:   EmergencyStopNew = 0;
4595:   heart_beat ();
4596: }
4597:
4598: // HeartBeat beats when Stimulation is OFF, during StandBy.
4599: // When Stimulation_On is On, HB LED Turns On.
```

```
4600: void heart_beat (void){
4601:   if (Stimulation_On){
4602:     HB_on ();
4603:   } else{
4604:     hr_count = (int)tick_count;
4605:         if (((hr_count >> 10) & 0x01) == 1){
4606:         HB_on ();
4607:     } else { HB_off ();}
4608:         if (((hr_count >> 8) & 0x01) == 1){
4609:         HB_off ();
4610:     }
4611:   }
4612: }
4613:
4614: #pragma vector=TIMERB0_VECTOR
4615: __interrupt void Timer_B0 (void)
4616: {
4617:   TBCCR0 += 400;                              // Add Offset to CCR0
4618: }
4619:
4620: #pragma vector=TIMERB1_VECTOR
4621: __interrupt void Timer_B1 (void)
4622: {
4623:   switch( TBIV )
4624:   {
4625: //  P4.1= CCR1 = 4000kHz/(2*10000) ~200Hz   //
4626: //  P4.2= CCR2 = 4000kHz/(2*4000) ~500Hz    // 1ms pulse widths
4627: //  1 ms CCR2 pulsewidths
4628:
4629:   case  2: TBCCR1 += 10000;                 // Add Offset to CCR1
4630:         heart_beat ();
4631:         break;
4632:   case  4: TBCCR2 += 4000;                  // Add Offset to CCR2 1ms loop
4633:     {
4634:         bit.TickISR  = 1;                   // ISR handler used for timerB.
4635:         tick_count ++;                      // 1ms tick
4636: //      P4OUT ^= 0x80;                      // counted signal -temporary on p4.7
4637: // HeartBeat function adds big time jitter. Try using the function elsewhere.
4638: //  heart_beat ();
4639:     }
4640:         break;
4641:   case 14:
4642: //      P4OUT ^= 0x40;                      // Timer_B7 overflow on p4.6
4643:         EmergencyKey();
4644:         // Normally, Stimulation_trigger/ EmergencyKey input is at 0V.
4645:     EmergencyStopNew = Emergency;       // Test RunProcess button during loops
4646:     if ((EmergencyStopOld & EmergencyStopNew) & !(Looping == 1))
4647:         {
4648:             // TimerA would better be stopped. Otherwise conflicts might arise
4649:             TACCTL0 &= ~CCIE;            // CCR0 interrupt disabled
4650:             TACTL = TASSEL_2 + MC_0;     // SMCLK, Stop
4651:             TACCR0 = 0;                  // Stop the TimerA
4652:             // Additionally, OutPut Disable pin, OD might be pulled up.
4653:             // Make sure that the Stimulator turns off.
4654:             // set OD high
4655:             XTR_Out_Disable = 0x01;
4656:             // Sometimes unexpected freezing happens. Clear the looping as well
4657:             Looping = 1;
4658:         main();          // If pressed long enough and loop is
4659:     }                    // active then run Main
4660:     EmergencyStopOld = EmergencyStopNew;
4661:         break;
4662:  }
4663: }
4664:
4665:
4666: // ********************************************************************************** */
4667:
4668:
4669: /*This file has been prepared for Doxygen automatic documentation generation.*/
4670: /*! \file ****************************************************************
4671:  *
4672: * TESsaNova: tDCS TransCranial DC Brain Stimulator
4673: * Developed for research studies at MAKELab
4674: *
4675: * Switch Readings File
```

```
4676: *
4677: * Adnan Kurt
4678: * MakeLAB
4679: * 12Sep. 2013
4680: * Zekeriyakoy, Istanbul
4681: * - File:               TESsaNova_Switches_12Sep2013.c
4682: * - Compiler:           IAR EWBMSP430 5.40
4683: * - Supported devices:  MSP430F149
4684: * - Circuit:            TESsaNova_19Nov2011_F.sch
4685: *
4686: *  \author    AdKu            \n
4687: *             Adnan Kurt      \n
4688: *             MakeLAB        \n
4689: *             12Sep. 2013     \n
4690: *             Zekeriyakoy, Istanbul
4691: *
4692: *
4693: ******************************************************************************/
4694:
4695: int More;
4696: int Less;
4697: int length;
4698: int Changed;
4699: int Stimulation_Duration_select = 0;
4700: int Stimulation_Duration_Set = 0;
4701: unsigned int Stimulation_Duration = 10;
4702: void Citi_Wall_Params (void);
4703:
4704: /*
4705: unsigned int Duration = 2;             // default 16-bit value to write to segment A
4706: unsigned int Stimulation_Duration = 10;// default 16-bit value to write to segment A
4707: unsigned int Period = 20;              // default 16-bit value to write to segment A
4708: unsigned int OnTime = 25;              // default 16-bit value to write to segment A
4709: unsigned int OffTime = 75;             // default 16-bit value to write to segment A
4710: unsigned int Compliance_Voltage = 10;  // kept for consistency.
4711: int Stimulation_Mode = 1;              // kept for consistency.
4712: */
4713:
4714: /*
4715: // Accelerated Switch Read
4716: // Add error statement (More & Less) == 1 then error. Later
4717: void Read_Stimulation_Duration (void){
4718: More = Increment;
4719: Less = Decrement;
4720:   if ((More | Less) == 1){
4721:   int LoopConstant = 200;               // number of delay constants
4722:   int LoopAcceleration = 1;             // minimum loop constant = 1
4723:   wait (1);
4724:   while (More | Less == 1){
4725:     if ((More == 1) & (Less == 0)){
4726:     Stimulation_Duration += 10 ;    // tested +=LoopAcceleration, too fast.
4727:     }
4728:     if ((More == 0) & (Less == 1)){
4729:     Stimulation_Duration -= 10 ;    // 10 sec steps
4730:     }
4731:     Citi_Wall_Params();                // This should be parameter update
4732:     wait (LoopConstant);               // Acceleration
4733:     LoopConstant = 200 / LoopAcceleration;
4734:         LoopAcceleration = LoopAcceleration + 1;
4735: More = Increment;    // Resample switches
4736: Less = Decrement;    // Resample switches
4737:         if (Stimulation_Duration > 3600)
4738:         {
4739:           Stimulation_Duration = 10;
4740:         }
4741:         if (Stimulation_Duration < 11)
4742:         {
4743:           Stimulation_Duration = 10;
4744:         }
4745:     if (LoopAcceleration > 190)
4746:         {
4747:       LoopAcceleration = 200;
4748:         }
4749:   }
4750:     Changed = 1;
4751:   }
```

```
4752:   }
4753: */
4754:
4755: void EmergencyKey (void)
4756: {
4757:
4758: // Normally, Stimulation_trigger/ EmergencyKey input is at 0V.
4759: // To adapt to the Emergency Key check in PP_Loop Timer, I inverted the signal.
4760: // Emergency = !Stimulation_trigger;
4761: // When Button is pressed, Emergency is 1
4762:    Emergency = Stimulation_trigger;
4763: }
4764:
4765: // Read ProcessKey
4766: // If PushButton is pressed, check noise, and then, Wait until release
4767: // to start processing.
4768: // Normally, Stimulation_trigger input is at 0V.
4769: void Process_Key (void) {       // Check Process button
4770:   if (Stimulation_trigger == 1){
4771:     length = 100;
4772:     Beep(100, 1);
4773:     wait (20);
4774:     if (Stimulation_trigger == 1){
4775:       while (Stimulation_trigger == 1) {
4776:       HB_off ();
4777:       }
4778:       Process = 1 ;
4779:     }
4780:   } else{
4781:       Process = 0 ;
4782:     }
4783: }
4784:
4785: // stimulation_duration (Stimulation Duration in seconds) selection using
4786: // E24 Renard Numbers in S:
4787: // Instead, I will use Octal series:
4788: /*
4789: unsigned int  a = 100;
4790: unsigned int  b = 120;
4791: unsigned int  c = 150;
4792: unsigned int  d = 180;
4793: unsigned int  e = 220;
4794: unsigned int  f = 270;
4795: unsigned int  g = 330;
4796: unsigned int  h = 390;
4797: */
4798:
4799: // Stimulation Session Duration spans 10sec-3600sec. Fair enough.
4800: unsigned int  a = 10;
4801: unsigned int  b = 60;
4802: unsigned int  c = 120;
4803: unsigned int  d = 300;
4804: unsigned int  e = 600;
4805: unsigned int  f = 900;
4806: unsigned int  g = 1200;
4807: unsigned int  h = 1800;
4808: unsigned int  m = 3600;
4809:
4810: void Stimulation_Duration_Calculate(int Stimulation_Duration_select)
4811: {
4812:     switch (Stimulation_Duration_select){
4813:     case 0 : Stimulation_Duration_Set = a;
4814:     break;
4815:     case 1 : Stimulation_Duration_Set = b;
4816:     break;
4817:     case 2 : Stimulation_Duration_Set = c;
4818:     break;
4819:     case 3 : Stimulation_Duration_Set = d;
4820:     break;
4821:     case 4 : Stimulation_Duration_Set = e;
4822:     break;
4823:     case 5 : Stimulation_Duration_Set = f;
4824:     break;
4825:     case 6 : Stimulation_Duration_Set = g;
4826:     break;
4827:     case 7 : Stimulation_Duration_Set = h;
```

```
4828:     break;
4829:     case 8 : Stimulation_Duration_Set = m;
4830:     break;
4831:     default: Stimulation_Duration_Set = a;
4832:     break;
4833:     }
4834:     Stimulation_Duration = (unsigned int)Stimulation_Duration_Set;
4835: }
4836:
4837: // Read Pulsewidth using inc/ dec switches
4838: void Read_Stimulation_Duration(void){
4839:     if (Stimulation_Duration_select == 9){
4840:       Stimulation_Duration_select = 0;
4841:     }
4842:     if (Increment == 1)
4843:     {
4844:       wait (1);
4845:       if (Increment == 1)
4846:       {
4847:           // void Beep (int length, int freq)
4848:           // Acknowledge the key progress
4849:           Beep (40, 2);
4850:         while (Increment == 1)
4851:         {
4852:         }
4853:         Stimulation_Duration_select ++;
4854:       }
4855:     }
4856:     if (Decrement == 1)
4857:     {
4858:       wait (1);
4859:       if (Decrement == 1)
4860:       {
4861:           // void Beep (int length, int freq)
4862:           // Acknowledge the key progress
4863:           Beep (40, 3);
4864:         while (Decrement == 1)
4865:         {
4866:         }
4867:         Stimulation_Duration_select --;
4868:       }
4869:     }
4870:     Stimulation_Duration_Calculate(Stimulation_Duration_select);
4871:   }
4872:
4873: void Panel_Read (void){
4874:   Amplitude_Set_Read();
4875:   DC_Offset_Set_Read();
4876:   Frequency_Set_Read();
4877: //  Sense_Current();
4878: //  Supply_Voltage();
4879: //  Board_Tempera_Sensor();
4880:   Read_Stimulation_Duration();
4881:   Process_Key ();
4882: }
4883:
4884: // ****************************************************************************** */
4885:
4886:
4887:
4888:
4889:
4890:
4891:
4892:
4893:
4894:
4895:
4896:
4897:
4898:
4899:
4900:
4901: CHERE
4902:
4903: \ ================================================================
```

```
4904: \ //
4905: \ //Reads ShamCodes and Runs Related Procedure- Sham or True tDCS
4906: (*
4907: When ShamRoutine runs, Parallel Leads are shorted to a 50Ohms Load. So, subject receives a tenth or less of a
      stimulation current. Sham Codes are supplied by a separate SW running on a distant PC and
4908: code is delivered to the experiment site over phone. Internal Decoder, selects Sham or True stimulation,
      referring to this code.
4909:
4910: When Sham stimulation is active, a Relay, connected to p2.5, is activated. In order to mask the relay chatter,
      a synhronised buzz is generated using the piezo buzzer or vibration motor. In order to eliminate any
      estimation/ recognition, buzzing is supplied every 5 minutes after the session is activated.
4911:
4912: Stimulation duration is set and stored in Flash, when the system is started with a hidden key. (Or, this could
      be activated after system started, and a DurationSet code is given to the unit.)
4913:
4914: Stimulation start detection is done with sampling the stimulation lead voltage.
4915: After about 5 minutes, stimulation current will be shorted if ShamCode is active. Then, the stimulation will be
      reverted during the last 5 minutes, calculated using the stimulation duration data.
4916: *)
4917: \ //
4918: \ // MAIN CODE.
4919: \ //
4920: \ // ADNAN KURT
4921: \ // MAKELAB
4922: \ // 16 Dec. 2018
4923: \ // Etiler, ISTANBUL
4924: \ //     LaunchPad set to 8MHz clock!
4925: \ //     mov.b   &CALBC1_8MHZ, &BCSCTL1   ; Set DCO
4926: \ //     mov.b   &CALDCO_8MHZ, &DCOCTL    ; to 8 MHz.
4927: \ //
4928: \ // BlueLED and RedLED Board @ p1.0
4929: \ // GreenLED on Board  @ p1.6
4930: \ // VibrationMotor     @ p1.6
4931: \ // Buzzer             @ p1.6/ over ULN2003
4932: \ // BLDCMOTOR      @ p2.5 on UsluKukla Board.
4933: \ // Vibration Motor, Buzzer and GreenLED all connected to p1.6!
4934: \ //
4935: \ //
4936: \ // Previous Stable version: ShamAn_v01_20181217.f
4937: \ // Some words commented, ROM is limited 25Dec2018
4938: \ //
4939: \ //
4940: \ ******************************************************************
4941: \ //
4942: \ //
4943: \ // U_Sw       @ p2.2  digital debounced switch input
4944: \ // V_Sw       @ p2.7  digital debounced switch input
4945: \ // W_Sw       @ p1.7  digital debounced switch input
4946: \ // PB_Sw  @ p2.1  also SW101 on UsluKukla
4947: \ // Stim_Sense@ p1.4   over R109 -check schematics
4948: \ // BlueLED    @ p1.0  RedLED as well
4949: \ // Out_Rly    @ p2.5  on UsluKukla Board
4950: \ // Buzzer     @ p1.6  over ULN2003
4951: \ // GreenLED  @ p1.6
4952: \ // VibMotor   @ p1.6
4953: \ //
4954: \ ******************************************************************
4955:
4956:
4957: \ *=================================================================
4958: \ STANDARD BITS
4959: HEX
4960: (*
4961: 0001 CONSTANT BIT0
4962: 0002 CONSTANT BIT1
4963: 0004 CONSTANT BIT2
4964: 0008 CONSTANT BIT3
4965: 0010 CONSTANT BIT4
4966: 0020 CONSTANT BIT5
4967: 0040 CONSTANT BIT6
4968: 0080 CONSTANT BIT7
4969: 0100 CONSTANT BIT8
4970: 0200 CONSTANT BIT9
4971: 0400 CONSTANT BITA
4972: 0800 CONSTANT BITB
4973: 1000 CONSTANT BITC
```

```
4974: 2000 CONSTANT BITD
4975: 4000 CONSTANT BITE
4976: 8000 CONSTANT BITF
4977: *)
4978: 0020  CONSTANT P1IN        \  Port 1 Input
4979: 0021  CONSTANT P1OUT       \  Port 1 Output
4980: 0022  CONSTANT P1DIR       \  Port 1 Direction
4981: 0023  CONSTANT P1IFG       \  Port 1 Interrupt Flag
4982: 0024  CONSTANT P1IES       \  Port 1 Interrupt Edge Select
4983: 0025  CONSTANT P1IE        \  Port 1 Interrupt Enable
4984: 0026  CONSTANT P1SEL       \  Port 1 Selection
4985: 0041  CONSTANT P1SEL2      \  Port 1 Selection 2
4986: 0027  CONSTANT P1REN       \  Port 1 Resistor Enable
4987:
4988: 0028  CONSTANT P2IN        \  Port 2 Input
4989: 0029  CONSTANT P2OUT       \  Port 2 Output
4990: 002A  CONSTANT P2DIR       \  Port 2 Direction
4991: 002B  CONSTANT P2IFG       \  Port 2 Interrupt Flag
4992: 002C  CONSTANT P2IES       \  Port 2 Interrupt Edge Select
4993: 002D  CONSTANT P2IE        \  Port 2 Interrupt Enable
4994: 002E  CONSTANT P2SEL       \  Port 2 Selection
4995: 0042  CONSTANT P2SEL2      \  Port 2 Selection 2
4996: 002F  CONSTANT P2REN       \  Port 2 Resistor Enable
4997:
4998: 0004  CONSTANT TACLR       \  Timer A counter clear
4999: 0004  CONSTANT OUT         \  PWM Output signal if output mode 0
5000:
5001: (*
5002: 0 20  * CONSTANT OUTMOD_0   \  PWM output mode: 0 - output only
5003: 1 20  * CONSTANT OUTMOD_1   \  PWM output mode: 1 - set
5004: 2 20  * CONSTANT OUTMOD_2   \  PWM output mode: 2 - PWM toggle/reset
5005: 3 20  * CONSTANT OUTMOD_3   \  PWM output mode: 3 - PWM set/reset
5006: 4 20  * CONSTANT OUTMOD_4   \  PWM output mode: 4 - toggle
5007: 5 20  * CONSTANT OUTMOD_5   \  PWM output mode: 5 - Reset
5008: 6 20  * CONSTANT OUTMOD_6   \  PWM output mode: 6 - PWM toggle/set
5009: 7 20  * CONSTANT OUTMOD_7   \  PWM output mode: 7 - PWM reset/set
5010: 0 10  * CONSTANT MC_0       \  Timer A mode control: 0 - Stop
5011: 1 10  * CONSTANT MC_1       \  Timer A mode control: 1 - Up to CCR0
5012: 2 10  * CONSTANT MC_2       \  Timer A mode control: 2 - Continous up
5013: 3 10  * CONSTANT MC_3       \  Timer A mode control: 3 - Up/Down
5014: 0 40  * CONSTANT ID_0       \  Timer A input divider: 0 - /1
5015: 1 40  * CONSTANT ID_1       \  Timer A input divider: 1 - /2
5016: 2 40  * CONSTANT ID_2       \  Timer A input divider: 2 - /4
5017: 3 40  * CONSTANT ID_3       \  Timer A input divider: 3 - /8
5018: 0 100 * CONSTANT TASSEL_0   \  Timer A clock source select: 0 - TACLK
5019: 1 100 * CONSTANT TASSEL_1   \  Timer A clock source select: 1 - ACLK
5020: 2 100 * CONSTANT TASSEL_2   \  Timer A clock source select: 2 - SMCLK
5021: 3 100 * CONSTANT TASSEL_3   \  Timer A clock source select: 3 - INCLK
5022: 011E  CONSTANT TA1IV       \  Timer1_A3 Interrupt Vector Word
5023: 0180  CONSTANT TA1CTL      \  Timer1_A3 Control
5024: 0182  CONSTANT TA1CCTL0    \  Timer1_A3 Capture/Compare Control 0
5025: 0184  CONSTANT TA1CCTL1    \  Timer1_A3 Capture/Compare Control 1
5026: 0186  CONSTANT TA1CCTL2    \  Timer1_A3 Capture/Compare Control 2
5027: 0190  CONSTANT TA1R        \  Timer1_A3
5028: 0192  CONSTANT TA1CCR0     \  Timer1_A3 Capture/Compare 0
5029: 0194  CONSTANT TA1CCR1     \  Timer1_A3 Capture/Compare 1
5030: 0196  CONSTANT TA1CCR2     \  Timer1_A3 Capture/Compare 2
5031: *)
5032:
5033: HEX
5034:  004A     CONSTANT ADC10AE0    \  ADC10 Analog Enable 0
5035: 0 400   * CONSTANT SHS_0       \  ADC10SC
5036: 01B0      CONSTANT ADC10CTL0   \  ADC10 Control 0
5037: 01B2      CONSTANT ADC10CTL1   \  ADC10 Control 1
5038: 0 20    * CONSTANT ADC10DIV_0  \  ADC10 Clock Divider Select 0
5039: 010       CONSTANT ADC10ON     \  ADC10 On/Enable
5040: 01B4      CONSTANT ADC10MEM    \  ADC10 Memory
5041:   001     CONSTANT ADC10SC     \  ADC10 Start Conversion
5042: 2 800   * CONSTANT ADC10SHT_2  \  16 x ADC10CLKs
5043: 002       CONSTANT ENC         \  ADC10 Enable Conversion
5044: 4 1000  * CONSTANT INCH_4      \  Selects Channel 4
5045: 040       CONSTANT REF2_5V     \  ADC10 Ref 0:1.5V / 1:2.5V
5046: 020       CONSTANT REFON       \  ADC10 Reference on
5047: 1 2000  * CONSTANT SREF_1      \  VR +  = VREF +  and VR- = AVSS
5048: 021       CONSTANT p1          \  p1 P1out port address
5049: 029       CONSTANT p2          \  p2 P2out port address
```

```
5050:
5051: \ *================================================================
5052: \ Initialization Vocabulary
5053: HEX
5054:     BN 00000100 constant U_Sw
5055:     BN 10000000 constant V_Sw
5056:     BN 10000000 constant W_Sw
5057:     BN 00000010 constant P_Sw
5058:     BN 00010000 constant Stim_S
5059:     BN 00000001 constant BluLED
5060:     BN 00100000 constant Out_Rly
5061:     BN 01000000 constant VibMotr
5062:     BN 01000000 constant Buzzer
5063:     BN 01000000 constant GrnLED
5064:
5065: : Init_Ports
5066:   \ set direction register of P2.2, P1.7, P2.7 = IN.
5067:     0 p1DIR !
5068:     0 p2DIR !
5069:     0 p1OUT !
5070:     0 p2OUT !
5071:     0 p2sel !
5072:     0 p2sel2 !
5073: \   0 p1sel ! Can't do this! UART
5074: \   0 p1sel2 !
5075: \   0 p1ren !
5076: \   0 p2ren !
5077:     U_Sw    p2DIR *BIC
5078:     V_Sw    p2DIR *BIC
5079:     W_Sw    p1DIR *BIC
5080:     P_Sw    p2DIR *BIC
5081:     BluLED  p1DIR *BIS
5082:     GrnLED  p1DIR *BIS
5083:     Out_Rly p2DIR *BIS
5084:     VibMotr p1DIR *BIS
5085:     Buzzer  p1DIR *BIS
5086:     Stim_S  p1DIR *BIC
5087:   ;
5088:
5089: \ *================================================================
5090: \ LED Vocabulary
5091: HEX
5092: \ LED Control Words
5093: : >LEDS         ( b -- )    021 *BIS ( P1OUT ) ;
5094: : <LEDS         ( b -- )    021 *BIC ( P1OUT ) ;
5095:
5096: \ *================================================================
5097: \ ADC Vocabulary
5098: \ ADC on and sample time at 64 clocks
5099: : InitADC ( -- )
5100: \   Previous files had *bic, clearing ADC10CTL0 bits.
5101: \   I've changed on 24Jan2019
5102: \   The 16 bits versions are: **BIC **BIS **BIX BIT**
5103:     02   1B0 **bic  ( ADC10CTL0  Clear ENC )
5104:     10   04A c!     ( ADC10AE0   PP1.4 = ADC in )
5105: \   3870 1B0 ! ;        ( ADC10CTL0  Sampletime 64 clocks, ADC on )
5106: \                      ( SREF_1 ADC10SHT_2+ REF2_5V+ REFON+ ADC10ON+ )
5107:     3830 1B0 ! ;        ( ADC10CTL0  Sampletime 64 clocks, ADC on )
5108: \                      ( SREF_1 ADC10SHT_2+ REF1_5V+ REFON+ ADC10ON+ )
5109: \ We need to clear the ENC bit before setting a new input channel
5110: : ADC@    ( +n -- u )
5111: \   GrnLED >LEDS            ( Green LED  Turns On )
5112:     02 1B0 **bic            ( ADC10CTL0  Clear ENC )
5113:     F000 and 80 or 1B2 !    ( ADC10CTL1  Select input, MCLK/5 )
5114:     03 1B0 **bis            ( ADC10CTL0  Set ENC & ADC10SC )
5115:     begin
5116:     1 1B2 bit** 0=
5117:     until           ( ADC10CTL1  ADC10 busy? )
5118:     1B4 @                   ( ADC10MEM   Read result )
5119: \   GrnLED <LEDS        ( Green LED  Turns Off -period measurement )
5120:     ;
5121: (*
5122: : Read_Stim_Status
5123:     INCH_4 adc@         ( 4000 Read level at P1.4 )
5124:     ;
5125:
```

```
5126: \ ADCTest word reads p1.4 and displays on terminal.
5127: decimal
5128: : ADCTest InitADC begin hx 4000 ADC@ . 100 MS key? until ;
5129: *)
5130: \ *==================================================================
5131: \ Buzzer Vocabulary
5132: \ Toene
5133:
5134: variable dauer    300 dauer !  ( da )
5135: variable /freq    500 /freq !  ( fr )
5136: variable lull     500 lull  !
5137: BN 01000000 constant BUZOUT
5138: : init_Buzzer
5139:     DECIMAL
5140:     300 dauer !  ( da )
5141:     500 /freq !  ( fr )
5142:     500 lull  !
5143:     \ BN 01000000 constant BUZOUT
5144:     ;
5145:
5146: : pause ( -- ) lull @ MS       ; ( viiiiel warten)
5147: : warte ( -- ) /freq @ 0 do loop ; ( wenig warten)
5148:
5149: : TON ( d f -- )
5150:   /freq !  buzout p1 1 + *BIS
5151:   0 DO Buzzer p1 *BIS warte Buzzer p1 *BIC warte LOOP ;
5152:
5153: : SWEEP ( -- )
5154:     init_Buzzer
5155:     1000 100 DO I . dauer @ I TON pause
5156:     key? IF LEAVE THEN 50 +LOOP ;
5157:
5158: \ : Test  begin  300 200 ton  key? until ;
5159:
5160: : HiSo
5161:   buzout p1 1 + *BIS
5162:   500 45 ton
5163:   buzout p1 1 + *BIC
5164:   ;
5165: : LoSo
5166:   buzout p1 1 + *BIS
5167:   500 40 ton
5168:   buzout p1 1 + *BIC
5169:   ;
5170:
5171: \ *==================================================================
5172: \ Relay and VibrationMotor Vocabulary
5173:
5174: \ run Relay
5175: : Relay_on  Out_Rly p2OUT *BIS ;
5176: : Relay_off Out_Rly p2OUT *BIC ;
5177: Relay_off
5178:
5179: \ action menue:
5180: \ press 2 to turn-on  relay
5181: \ press 9 to turn-off relay
5182: \ press any other key to exit routine
5183: : action ( tf key -- tf )
5184:     swap drop
5185:     dup [char] 2 = IF drop Relay_on  false exit THEN
5186:     dup [char] 9 = IF drop Relay_off false exit THEN
5187:     drop Relay_off
5188:     true
5189:   ;
5190:
5191: (*
5192: : Relaytest
5193:     Relay_off
5194:     BEGIN
5195:     key? dup
5196:     IF
5197:     key
5198:     action
5199:     THEN
5200:     UNTIL
5201:     ;
```

```
5202: *)
5203:
5204: \ run tiny vibration rotor
5205:   : VibMotor_on
5206:         VibMotr p1DIR *BIS
5207:         VibMotr p1OUT *BIS
5208:         ;
5209:   : VibMotor_off
5210:         VibMotr p1OUT *BIC
5211:         VibMotr p1DIR *BIC
5212:         ;
5213:
5214: VibMotor_off
5215: (*
5216: \ action menue:
5217: \ press 1 to activate motor
5218: \ press 0 to stop motor
5219: \ press any other key to exit routine
5220: : actionvib ( f key -- f )
5221:     swap drop
5222:     dup [char] 1 = IF drop VibMotor_on  false exit THEN
5223:     dup [char] 0 = IF drop VibMotor_off false exit THEN
5224:     drop VibMotor_off
5225:     true
5226:     ;
5227:
5228: : vibtest
5229:     VibMotor_off
5230:     BEGIN
5231:     key? dup
5232:     IF
5233:     key
5234:     actionvib
5235:     THEN ( dup . )
5236:     UNTIL
5237:     ;
5238: *)
5239: (*
5240: \ *==================================================================
5241: \ Switches Vocabulary
5242: : Read_U_Sw
5243:     Begin
5244:     p2IN @ U_Sw AND U_Sw =
5245:     while
5246:     BluLED >LEDS
5247:     Repeat
5248:     p2IN @ U_Sw AND U_Sw <>
5249:     BluLED <LEDS     ( BlueLED Turns Off )
5250:     Drop
5251:     ;
5252:
5253: \ V_Sw does not work. Middle Toggle Switch on the panel.
5254: \ p2.7 is connected to CS of IC105.
5255: \ Problem was with port setting.
5256: \ Need to init ports SEL parameters as well.
5257: : Read_V_Sw
5258:     Begin
5259:     p2IN @ V_Sw AND V_Sw =
5260:     while
5261:     BluLED >LEDS
5262:     Repeat
5263:     p2IN @ V_Sw AND V_Sw <>
5264:     BluLED <LEDS     ( BlueLED Turns Off )
5265:     Drop
5266:     ;
5267:
5268: : Read_W_Sw
5269:     Begin
5270:     p1IN @ W_Sw AND W_Sw =
5271:     while
5272:     BluLED >LEDS
5273:     Repeat
5274:     p1IN @ W_Sw AND W_Sw <>
5275:     BluLED <LEDS     ( BlueLED Turns Off )
5276:     Drop
5277:     ;
```

```
5278:
5279: : Read_P_Sw
5280:     Begin
5281:     p2IN @ P_Sw AND P_Sw =
5282:     while
5283:     BluLED >LEDS
5284:     Repeat
5285:     p2IN @ P_Sw AND P_Sw <>
5286:     BluLED <LEDS     ( BlueLED Turns Off )
5287:     Drop
5288:     ;
5289:
5290: *)
5291: \ *==================================================================
5292: \ 7SegmentDisplay Vocabulary
5293:
5294: \    76543210
5295:   BN 00000001 constant SER \ P2.0
5296:   BN 00001000 constant SCK \ P2.3
5297:   BN 00010000 constant RCK \ P2.4
5298:
5299: \ set direction register of P2.0,3,4 = OUT.
5300: \ BN 00011001 p2 1+ 2constant 7SEGOUT
5301:   BN 00011001 constant 7SEGOUT
5302:
5303: : ini7seg    7segout P2 1+ *BIS ;
5304:
5305: \ Adnan 20170713
5306: hex
5307: \ shift in a 16 bit pattern and output it
5308: \ RLA    n1 -- n2 f   rotate left through
5309: : RLAA ( n -- n f )
5310:   DUP  7FFF OR
5311:   8000 AND ;
5312:
5313: decimal
5314: \ create a 7 segment display buffer
5315: create 7sdb  4 cells allot align
5316:
5317: \ do a single puls to shift in 1 bit into IC74595 (7seg,LCD)
5318: : 1SCK    ( -- ) sck P2 *BIC sck P2 *BIS ;
5319:
5320: \ do a single puls to output pattern
5321: : 1RCK    ( -- ) rck P2 *BIC rck P2 *BIS ;
5322:
5323: : >SER ( p -- )
5324:   16 0 DO
5325:   RLAA IF ser p2 *BIS ELSE ser p2 *BIC THEN 1sck
5326:   1 lshift
5327:   LOOP
5328:   1rck drop ;
5329: \ Adnan 20170713
5330:
5331: \ shift in 4 patterns
5332: : 7SEG    8 0 DO 7sdb i + @ >ser 2 +LOOP ;
5333:
5334: \ convert number to a 7seg pattern
5335: \    7654321076543210
5336:   BN 0000000100000000 constant dig1
5337:   BN 0000001000000000 constant dig2
5338:   BN 0000010000000000 constant dig3
5339:   BN 0000100000000000 constant dig4
5340:
5341:   chere
5342: \    hgfedcba     segment
5343:   BN 00111111 c, \ 0
5344:   BN 00000110 c, \ 1
5345:   BN 01011011 c, \ 2
5346:   BN 01001111 c, \ 3
5347:   BN 01100110 c, \ 4
5348:   BN 01101101 c, \ 5
5349:   BN 01111101 c, \ 6
5350:   BN 00000111 c, \ 7
5351:   BN 01111111 c, \ 8
5352:   BN 01100111 c, \ 9
5353:   BN 01110111 c, \ A
```

```
5354:   BN 01111100 c, \ B
5355:   BN 01011000 c, \ C
5356:   BN 01011110 c, \ D
5357:   BN 01111001 c, \ E
5358:   BN 01110001 c, \ F
5359:   BN 10000000 c, \ dot
5360:   align
5361:   constant pat0
5362:
5363: decimal
5364: : PAT   ( n -- p ) pat0 + c@ ;
5365: : 4DIG  ( n -- n4 n3 n2 n1 )  4 0 DO base @ /mod LOOP drop ;
5366: : .7SEG ( n -- )
5367:   4dig
5368:   pat dig1 or  7sdb     !
5369:   pat dig2 or  7sdb 2 + !
5370:   pat dig3 or  7sdb 4 + !
5371:   pat dig4 or  7sdb 6 + ! ;
5372:
5373: HEX
5374: \ Decipoint adds a decimal point to the given digit
5375: \ MSDigit decimal point is not connected!
5376: \ 2nd digit has (0)DP2, (2)DP3, (3)DP4 available. (1)DP5DP6 connected
5377: \ parallel and is semi column.
5378: : Decipoint_On ( n -- )
5379:     2 * dup 7sdb + @ 80 or swap 7sdb + !
5380:     ;
5381:
5382: : Decipoint_Off ( n -- )
5383:     2 * dup 7sdb + @ F7F and swap 7sdb + !
5384:     ;
5385:
5386: : 7SegBlank
5387:     0 dig1 or  7sdb     !
5388:     0 dig2 or  7sdb 2 + !
5389:     0 dig3 or  7sdb 4 + !
5390:     0 dig4 or  7sdb 6 + !
5391:     ;
5392:
5393:
5394: \ testing 7seg display: show pattern stored in 7seg display buffer
5395: : LED_TEST  ini7seg BEGIN 7seg key? UNTIL 0 >ser ;
5396: : Write_Led 7seg 0 >ser ;
5397:
5398:
5399:
5400: shield ShamAn\
5401: FREEZE
5402:
5403: ( finis )
5404: CHERE SWAP - .
5405:
5406: \ ******************************************************************************
5407:
5408: CHERE
5409: \ //
5410: \ // Previous Stable version: UserSwitches_v08_20190114.f
5411: \ // This version differs with commented out monitors
5412: \ // Connection to the serial umbilical cut.
5413: \ // 15Jan2019
5414: \ //
5415: \ ****************************************************************
5416:
5417: variable U_Set      0 U_Set !
5418: variable V_Set          0 V_Set !
5419: variable W_Set          0 W_Set !
5420: variable P_Set          0 P_Set !
5421: variable U_Value        0 U_Value !
5422: variable V_Value        0 V_Value !
5423: variable W_Value        0 W_Value !
5424: variable P_Value        0 P_Value !
5425: variable ShamValue      false ShamValue !
5426: variable ModeSet        false ModeSet !
5427: variable RunMode        false RunMode !
5428: variable P_Press        false P_Press !
5429: variable P_Long_Press   false   P_Long_Press !
```

```
5430: variable Loops      DM 100 Loops !
5431:
5432: : Init_Switches
5433: \ HEX
5434: \ Interferes with Base consistency operations.
5435:         0 U_Set !
5436:         0 V_Set !
5437:         0 W_Set !
5438:         0 P_Set !
5439:         0 U_Value !
5440:         0 V_Value !
5441:         0 W_Value !
5442:         0 P_Value !
5443:         0 ShamValue !
5444:         false P_Press !
5445:         false ModeSet !
5446:         false RunMode !
5447:         HX 1040 @
5448:         4DIG
5449:         DROP
5450:         0
5451:         (*
5452:         DUP . P_Value !
5453:         DUP . W_Value !
5454:         DUP . V_Value !
5455:         DUP . U_Value !
5456:         *)
5457: \       Monitoring
5458:         P_Value !
5459:         W_Value !
5460:         V_Value !
5461:         U_Value !
5462:         ;
5463:
5464: : Read_Ports
5465:     p2IN @ U_Sw AND
5466:     IF  U_Sw =
5467:         false U_Set !
5468:         ELSE
5469:         true U_Set !
5470:         THEN
5471:     p2IN @ V_Sw AND
5472:     IF  V_Sw =
5473:         false V_Set !
5474:         ELSE
5475:         true V_Set !
5476:         THEN
5477:     p1IN @ W_Sw AND
5478:     IF  W_Sw =
5479:         false W_Set !
5480:         ELSE
5481:         true W_Set !
5482:         THEN
5483:     p2IN @ P_Sw AND
5484:     IF  P_Sw =
5485:         false P_Set !
5486:         ELSE
5487:         true P_Set !
5488:         THEN
5489:     ;
5490:
5491: : Read_U_Sw_Release
5492:     Read_Ports
5493:     U_Set @ V_Set @ W_Set @ P_Set @ OR OR INVERT AND
5494:     IF
5495:     DM 10 MS
5496:     BEGIN
5497:     HX 05 MS
5498:     BluLED <LEDS
5499:         DM 03
5500:         Decipoint_Off
5501:         HX 20 0 ?DO
5502:             Write_Led
5503:         LOOP
5504:     Read_Ports
5505:     U_Set @
```

```
5506:     WHILE
5507:     HX 20 MS
5508:     BluLED >LEDS
5509:     1 U_Value +!
5510:         U_Value @
5511: \       0F >
5512: \   In order to make the algorith Base independent:
5513: \   Get base value, and limit the maximum with respect to that.
5514:         BASE @ 1 -
5515:         >
5516:         IF
5517:         0 U_Value !
5518:         THEN
5519:     U_Value @ pat dig4 or 7sdb 6 + !
5520:         DM 03
5521:         Decipoint_On
5522:         HX 60 0 ?DO
5523:             Write_Led
5524:         LOOP
5525:     REPEAT
5526:     THEN
5527:     ;
5528:
5529: : Read_V_Sw_Release
5530:     Read_Ports
5531:     V_Set @ U_Set @ W_Set @ P_Set @ OR OR INVERT AND
5532:     IF
5533:     DM 10 MS
5534:     BEGIN
5535:     HX 05 MS
5536:     BluLED <LEDS
5537:         DM 02
5538:         Decipoint_Off
5539:         HX 20 0 ?DO
5540:             Write_Led
5541:         LOOP
5542:     Read_Ports
5543:     V_Set @
5544:     WHILE
5545:     HX 20 MS
5546:     BluLED >LEDS
5547:     1 V_Value +!
5548:         V_Value @
5549: \       0F >
5550: \   In order to make the algorith Base independent:
5551: \   Get base value, and limit the maximum with respect to that.
5552:         BASE @ 1 -
5553:         >
5554:         IF
5555:         0 V_Value !
5556:         THEN
5557:     V_Value @ pat dig3 or 7sdb 4 + !
5558:         DM 02
5559:         Decipoint_On
5560:         HX 60 0 ?DO
5561:             Write_Led
5562:         LOOP
5563:     REPEAT
5564:     THEN
5565:     ;
5566:
5567: : Read_W_Sw_Release
5568:     Read_Ports
5569:     W_Set @ V_Set @ U_Set @ P_Set @ OR OR INVERT AND
5570:     IF
5571:     DM 10 MS
5572:     BEGIN
5573:     HX 05 MS
5574:     BluLED <LEDS
5575:         0
5576:         Decipoint_Off
5577:         HX 20 0 ?DO
5578:             Write_Led
5579:         LOOP
5580:     Read_Ports
5581:     W_Set @
```

```
5582:     WHILE
5583:     HX 20 MS
5584:     BluLED >LEDS
5585:     1 W_Value +!
5586:         W_Value @
5587: \       0F >
5588: \   In order to make the algorith Base independent:
5589: \   Get base value, and limit the maximum with respect to that.
5590:         BASE @ 1 -
5591:         >
5592:         IF
5593:         0 W_Value !
5594:         THEN
5595:     W_Value @ pat dig2 or 7sdb 2 + !
5596:         0
5597:         Decipoint_On
5598:         HX 60 0 ?DO
5599:             Write_Led
5600:         LOOP
5601:     REPEAT
5602:     THEN
5603:     ;
5604:
5605: : Read_P_Sw_Release
5606:     7SegBlank
5607:     \ Focus on 1st Digit
5608:     Read_Ports
5609:     P_Set @ V_Set @ W_Set @ U_Set @ OR OR INVERT AND
5610:     IF
5611:     DM 10 MS
5612:     BEGIN
5613:     HX 05 MS
5614:     BluLED <LEDS
5615:         1
5616:         Decipoint_Off
5617:         HX 20 0 ?DO
5618:             Write_Led
5619:         LOOP
5620:     Read_Ports
5621:     P_Set @
5622:     WHILE
5623:     HX 40 MS
5624:     BluLED >LEDS
5625:     1 P_Value +!
5626:         P_Value @
5627:         DM 09 >
5628:         \ Run over 0-9
5629:         IF
5630:         0 P_Value !
5631:         THEN
5632:     P_Value @ pat dig1 or 7sdb !
5633:         DM 01
5634:         Decipoint_On
5635:         HX 60 0 ?DO
5636:             Write_Led
5637:         LOOP
5638:     REPEAT
5639:     THEN
5640:     ;
5641:
5642: : WaitToReleasePB
5643:     ShamValue @ .7SEG
5644:     BEGIN
5645:     DM 02 MS
5646:     Read_Ports
5647:     P_Set @
5648:     WHILE
5649:     DM 200 MS
5650:         DM 60 0 ?DO
5651:         Write_Led
5652:         LOOP
5653:     REPEAT
5654:     ;
5655:
5656: : P_Sw_LongPress
5657: \ This P word enables to cancel the operation by longPress.
```

```
5658: \ Nothing to do with saved parameters
5659:     HX 100 Loops !
5660:     false P_Long_Press !
5661:     false P_Press !
5662:     Read_Ports
5663:     P_Set @ V_Set @ W_Set @ U_Set @ OR OR INVERT AND
5664:     IF
5665:     true P_Press !
5666:     DM 10 MS
5667:     BEGIN
5668:     DM 02 MS
5669:     BluLED <LEDS
5670:     Read_Ports
5671:     P_Set @ Loops @ HX 80 = INVERT AND
5672:     WHILE
5673:     1 Loops +!
5674:     DM 20 MS
5675:     BluLED >LEDS
5676:     Loops @ HX 11A >
5677:         IF
5678:         LoSo
5679: \       init_Switches
5680:         false P_Press !
5681:         true P_Long_Press !
5682:         BluLED <LEDS
5683:         HX 80 Loops !
5684:         \ Wait here, until leaving the PB
5685:             BEGIN
5686:             DM 02 MS
5687:             Read_Ports
5688:             P_Set @
5689:             WHILE
5690:             DM 10 MS
5691:             REPEAT
5692:         THEN
5693:     REPEAT
5694:     ELSE
5695:     false P_Press !
5696:     THEN
5697:     ;
5698:
5699: shield Switches\
5700: FREEZE
5701: CHERE SWAP - .
5702:
5703: \ ********************************************************************************
5704:
5705: CHERE
5706: \ //
5707: \ // Previous Stable version: ShamAn_Main_v09_20190115.f
5708: \ // This version differs with commented out monitors
5709: \ // Connection to the serial umbilical cut.
5710: \ // 15Jan2019
5711: \ //
5712: \ // This version is stripped down the commented codes
5713: \ // and removed main words. Those are created in Enigma file.
5714: \ // 17Jan2019
5715: \ //
5716: \ ****************************************************************
5717:
5718: (*
5719: RECYCLE ( addr -- )
5720: ROMC!  ( b addr -- )
5721: ROM! ( x addr -- )
5722:
5723: The free info blocks are $40 bytes each &
5724: run from $1000 to $103F
5725: and from $1040 to $107F
5726: The other two info blocks are used by noForth or the MSP430 itself.
5727:
5728:     hx 1040 @   ShamValue
5729:     hx 1042 @   expDuration
5730:     hx 1044 @   ShamLength
5731:     hx 1046 @   noShamOp
5732:     hx 1048 @   ShamOp
5733:     hx 104A @   trigAmplitude
```

```
5734:     hx 104C @   ShowType
5735: *)
5736: \ ROM addresses of the Parameters
5737:     hx 1040 CONSTANT    ShamValue_r
5738:     hx 1042 CONSTANT    expDuration_r
5739:     hx 1044 CONSTANT    ShamLength_r
5740:     hx 1046 CONSTANT    noShamOp_r
5741:     hx 1048 CONSTANT    ShamOp_r
5742:     hx 104A CONSTANT    trigAmplitude_r
5743:     hx 104C CONSTANT    ShowType_r
5744: variable Parameter hx 1040 Parameter !
5745: \ Parameter base address initially
5746: variable Parameter_Value hx 1D9 Parameter_Value !
5747: \ create a 20 parameters buffer
5748: create paramsBuff  hx 14 cells allot align
5749: variable Loops2     11 Loops2 !
5750:
5751: \ To Do
5752: \ hex dec clearance
5753: \ variable inits & placements.
5754:
5755: : To7SEG ( n -- )
5756: \ dig1 of 7Seg replaced with P_Value
5757:   4dig
5758:   pat dig1 or  7sdb     !
5759:   pat dig2 or  7sdb 2 + !
5760:   pat dig3 or  7sdb 4 + !
5761:   pat dig4 or  7sdb 6 + !
5762:   P_Value @ pat dig1 or 7sdb !
5763:   ;
5764:
5765: : ReplaceShamValue
5766:     HEX
5767:     ShamValue_r @
5768:     DUP
5769:     ShamValue !
5770:         4DIG
5771:         DROP
5772:         W_Value !
5773:         V_Value !
5774:         U_Value !
5775:     ;
5776:
5777: : Save_Parameter
5778:     P_Press @
5779:     true =
5780:     IF
5781:         Parameter_Value @
5782:         DUP
5783:         To7SEG
5784:         Write_Led
5785: \   : DOWN ( +n -- )
5786:     BEGIN
5787:         DUP
5788:         To7SEG
5789:         1-
5790:         Write_Led
5791:         DUP 0< UNTIL
5792:         DROP
5793:         HX 1A MS
5794:         DM 190 0 ?DO
5795:             Write_Led
5796:         LOOP
5797:     \ Fetch Flash Values to Buffer
5798:     DM 10 0 DO
5799:         hx 1040 I 2 * + @
5800:         paramsBuff I 2 * + !
5801:         LOOP
5802:     hx 1040 RECYCLE
5803:     \ Clears InfoFlash! Make Sure to rescan and save.
5804:     Parameter_Value @
5805:     \ Replace value with fetched component
5806:     Parameter @ hx 1040 INVERT AND
5807:     paramsBuff + !
5808:     \ Write Buffer Values to Flash
5809:     DM 20 0 DO
```

```
5810:         paramsBuff I 2 * + @
5811:         hx 1040 I 2 * +  ROM!
5812:         LOOP
5813:     false P_Press !
5814:     DM 100 MS
5815:     THEN
5816: \       : UP ( +n -- )
5817:         Parameter @ @
5818:         DUP
5819:         HX 1000 >
5820:         SWAP
5821:         HX 0 <
5822:         OR
5823:         IF
5824:             HX 000E
5825:             To7SEG
5826:             Write_Led
5827:             DM 690 0 ?DO
5828:             Write_Led
5829:             LOOP
5830:         ELSE
5831:         Parameter @ @
5832:         1+ 1 ?DO I
5833:         To7SEG
5834:         Write_Led
5835:         LOOP
5836:         HX 1B MS
5837:         DM 190 0 ?DO
5838:             Write_Led
5839:         LOOP
5840:     THEN
5841:     ;
5842:
5843: : count2end
5844:     1 Loops2 +!
5845: \       Loops2 @ .
5846: \       Monitoring
5847:         Loops2 @ HX 01AB >
5848:         IF
5849:         false ModeSet !
5850:         true P_Press !
5851:         THEN
5852:     ;
5853:
5854: : Set_Parameter ( n -- )
5855: \   Save variable Parameter address
5856: \   If still same state use the previous P_Value
5857: \   Uses P_Value [Machine State]
5858: \   n: Parameter@ROM address
5859:     DUP
5860:     Parameter @
5861:     <>
5862:     IF
5863:         Parameter !
5864: \       7SegBlank
5865:     Parameter @ @
5866:     DUP
5867:         4DIG
5868:         DROP
5869: \   P_Value to be set externally
5870:         W_Value !
5871:         V_Value !
5872:         U_Value !
5873: \   Parameter is Flash Address to be set.
5874:     Parameter_Value !
5875:     ELSE
5876:         DROP
5877:     THEN
5878: \   Initialization is done and loop starts
5879:     P_Sw_LongPress
5880:     BEGIN
5881:     count2end
5882:     P_Press @ INVERT
5883: \   Key? INVERT
5884: \   AND
5885: \   Remove KEY? switch later.
```

```
5886:     WHILE
5887:         Read_U_Sw_Release
5888:         Read_V_Sw_Release
5889:         Read_W_Sw_Release
5890:         P_Sw_LongPress
5891:
5892:         U_Value @ \ dup .
5893:         V_Value @ \ dup .
5894:         W_Value @ \ dup .
5895: \   Digits calculation is based on the Base value. So, number base
5896: \   Must not be used:
5897: \   The calculation depends on the environment program loaded.
5898: \   So, better idea is to use Base value composition.
5899:         base @ dup * *
5900:         SWAP
5901:         base @  *
5902:         + +
5903:         Parameter_Value !
5904:         Parameter_Value @ To7SEG
5905:             DM 90 0 ?DO
5906:                 Write_Led
5907:             LOOP
5908:     P_Sw_LongPress
5909:     Parameter @ @
5910:     Parameter_Value @
5911:     <>
5912:     IF
5913:         P_Press @
5914:         true =
5915:         IF
5916:             Save_Parameter
5917:             \ ." saved "
5918:             false P_Press !
5919:             \ ." false P_Press "
5920:         THEN
5921:     THEN
5922:     P_Long_Press @
5923:         IF
5924:             false ModeSet !
5925:             true P_Press !
5926:         THEN
5927:     REPEAT
5928: ;
5929:
5930: : ?Run_Session
5931: \ Correct and Complete the code to run the experiment with
5932: \ Set values -get parameters from info.mem
5933:     P_Press @
5934:     true =
5935:     IF
5936:         ShamValue_r @
5937:         ShamValue @
5938:         =
5939:         IF
5940:         true RunMode !
5941:         false ModeSet !
5942:         false P_Press !
5943: \       CR
5944: \       ." RunningExperiment "
5945: \       CR
5946: \       Monitoring
5947:         ShamValue @ .7SEG
5948:         0 dig1 or 7sdb !
5949: \   Blanks 1st digit.
5950:         DM 90 0 ?DO
5951:             Write_Led
5952:         LOOP
5953:         ELSE
5954:         false RunMode !
5955:         THEN
5956:     THEN
5957:     ;
5958:
5959: : ChangeMode
5960:     P_Press @
5961:     true =
```

```
5962:     IF
5963:         ShamValue_r @
5964:         ShamValue @
5965:         <>
5966:         IF
5967:         true ModeSet !
5968:         false RunMode !
5969:         false P_Press !
5970:         ShamValue_r @
5971:         ShamValue !
5972: \       CR
5973: \       ." ModeSet Active "
5974: \       CR
5975: \       Monitoring
5976:         ELSE
5977:         false ModeSet !
5978:         THEN
5979:     THEN
5980:     ;
5981:
5982: : StandBy
5983: \ Make sure that this works correctly! OK.
5984:     Read_U_Sw_Release
5985:     Read_V_Sw_Release
5986:     Read_W_Sw_Release
5987:     P_Sw_LongPress
5988:
5989:     P_Long_Press @
5990:     IF
5991:         LoSo
5992:         init_Switches
5993:         HX 1040 @
5994:         ShamValue !
5995:         false P_Press !
5996:         false P_Long_Press !
5997:     THEN
5998:     U_Value @ \ dup .
5999:     V_Value @ \ dup .
6000:     W_Value @ \ dup .
6001: \   Digits calculation is based on the Base value. So, number base
6002: \   Must not be used:
6003: \   The calculation depends on the environment program loaded.
6004: \   So, better idea is to use Base value composition.
6005:     base @ dup * *
6006:     SWAP
6007:     base @  *
6008:     + +
6009:     ShamValue !
6010:     ShamValue @ .7SEG
6011:     0 dig1 or 7sdb !
6012: \   Blanks 1st digit.
6013:         DM 90 0 ?DO
6014:             Write_Led
6015:         LOOP
6016: \   ." StandBy"
6017: \   Uncomment to Monitor
6018: \   Monitoring
6019:     ;
6020:
6021: : Exit_Check
6022: \   ." Exit_Check"
6023: \   Monitoring
6024:     0000 To7SEG
6025:     P_Sw_LongPress
6026:     BEGIN
6027:     P_Press @ P_Long_Press @ OR INVERT
6028:     WHILE
6029:     count2end
6030:     DM 90 0 ?DO
6031:     Write_Led
6032:     LOOP
6033:     P_Sw_LongPress
6034:     REPEAT
6035:     P_Press @
6036:     IF
6037:     01 P_Value !
```

```
6038: \   CR
6039: \   ." P_Press"
6040: \   Monitoring
6041:     THEN
6042:     P_Long_Press @
6043:     IF
6044:     false ModeSet !
6045: \   CR
6046: \   ." P_Long_Press"
6047: \   Monitoring
6048:     THEN
6049:     ReplaceShamValue
6050:     ;
6051:
6052: : SelectParameters
6053:             \   Returns to main!
6054:     DM 11 Loops2 !
6055:     0 P_Value !
6056:             \ Arbitrarization
6057:     BEGIN
6058: \   ModeSet @ KEY? INVERT AND TRUE =
6059:             \   Remove KEY? switch later.
6060:     ModeSet @
6061:     WHILE
6062:     P_Value @
6063:         CASE
6064:         00 OF Exit_Check    ENDOF
6065:         01 OF ( Set_ShamCode )
6066:             HEX
6067:             ShamValue_r
6068:             Set_Parameter
6069:             02 P_Value !
6070:         ENDOF
6071:         02 OF ( Set_expDuration )
6072:             DECIMAL
6073:             expDuration_r
6074:             Set_Parameter
6075:             03 P_Value !
6076:         ENDOF
6077:         03 OF ( Set_ShamLength )
6078:             DECIMAL
6079:             ShamLength_r
6080:             Set_Parameter
6081:             04 P_Value !
6082:         ENDOF
6083:         04 OF ( Set_noShamOp )
6084:             HEX
6085:             noShamOp_r
6086:             Set_Parameter
6087:             05 P_Value !
6088:         ENDOF
6089:         05 OF ( Set_ShamOp )
6090:             HEX
6091:             ShamOp_r
6092:             Set_Parameter
6093:             06 P_Value !
6094:         ENDOF
6095:         06 OF ( Set_trigAmplitude )
6096:             DECIMAL
6097:             trigAmplitude_r
6098:             Set_Parameter
6099:             07 P_Value !
6100:         ENDOF
6101:         07 OF ( Set_ShowType )
6102:             HEX
6103:             ShowType_r
6104:             Set_Parameter
6105:             00 P_Value !
6106:         ENDOF
6107:         ENDCASE
6108:         ReplaceShamValue
6109:         \ If waits inactively, after a delay will return
6110:     REPEAT
6111:     ;
6112:
6113: shield Main\
```

```
6114: FREEZE
6115: CHERE SWAP - .
6116: \ ********************************************************************************
6117:
6118: CHERE
6119:
6120: \ ================================================================
6121: \ //
6122: \ //Reads ShamCodes and Runs Related Procedure- Sham or True tDCS
6123: (*
6124: When ShamRoutine runs, Parallel Leads are shorted to a 50 Ohms Load. So, subject receives a tenth or less of a
      stimulation current. Sham Codes are supplied by a separate SW running on a distant PC and code is delivered to
      the experiment site over phone. Internal Decoder, selects Sham or True stimulation, referring to this code.
6125:
6126: When Sham stimulation is active, a Relay, connected to p2.5, is activated. In order to mask the relay chatter,
      a synhronised buzz is generated using the piezo buzzer or vibration motor. In order to eliminate any
      estimation/ recognition, buzzing is supplied every 5 minutes after the session is activated.
6127:
6128: Stimulation duration is set and stored in Flash, when the system is started with a hidden key. (Or, this could
      be activated after system started, and a DurationSet code is given to the unit.)
6129:
6130: Stimulation start detection is done with sampling the stimulation lead voltage. After about 5 minutes,
      stimulation current will be shorted if ShamCode is active.Then, the stimulation will be reverted during the
      last 5 minutes, calculated using the stimulation duration data.
6131:
6132: Adnan Kurt at 1/28/2019 7:49 PM
6133: ShamAn dissipates about 20mA at standby. When buzzers, LED and vibration motor is operating, it gets about 70mA.
6134:     *)
6135: \ //
6136: \ // Process CODE.
6137: \ //
6138: (*
6139:     Process Code controls the process, decodes keys and manages the experiment.
6140:     Timing and triggering, experiment duration are handled by this file.
6141: *)
6142:
6143: (*
6144: \ When PB_LongPress Leave the Process anytime
6145: \ RunTime: Periodic Relay and Vibration Chatter
6146: \ RunTime: Read and Acknowledge tDCS operation
6147:     Select Modes with PB_Press and DisplayChange
6148:     0. Run Experiment
6149:     1. Set ShamCode
6150:     2. Set Experiment Duration
6151:     3. Set Sham Duration
6152:     4. Set noSham Operation with Code
6153:     5. Set Sham Operation with Code
6154:     6. Set tDCS trigger amplitude
6155:     7. Show Experiment Type with "Secret Code"
6156:     Save Parameters with Short Press
6157:     Don't Save if noChange
6158: How to Access Modes?
6159: When displayed code is changed (one digit for example) and PB pressed run into ModeSelect. PB rolls through 8
      states.
6160:
6161: Typical parameters are:
6162:     0. Run Experiment              N/A
6163:     1. Set ShamCode             HX  0EDA
6164:     2. Set Experiment Duration          DM  0010 ( min )
6165:     3. Set Sham Duration                DM  0002 ( min )
6166:     4. Set noSham Operation with Code       HX  03B5
6167:     5. Set Sham Operation with Code     HX  0CCA
6168:     6. Set tDCS trigger amplitude           DM  0060 ( in Counts )
6169:     7. Show Exp. "Secret Code"          HX  0076 ( noShamOp_r 3 RSHIFT )
6170: Typical Parameters are saved to the Flash initially, just for reference.
6171: Voltage divider is 1:2 10k+10k||100nF; I will replace with 1:10 divider 100k+10k||100nF
6172: ADC reference is 1V5, and 1024 counts max. Calculate accordingly.
6173: \   15 * 1000 * 1024 /
6174: \   Measured voltage in mV
6175: *)
6176:
6177: \ // ADNAN KURT
6178: \ // MAKELAB
6179: \ // 16 Jan. 2019
6180: \ // Etiler, ISTANBUL
6181: \ //
```

```
6182: \ //
6183: \ ****************************************************************
6184:
6185:     hx AA CONSTANT  Sham
6186:     hx FF CONSTANT  noSham
6187: \   Sample Values:
6188: \   hx CCA CONSTANT ShamOp
6189: \   hx 3B5 CONSTANT noShamOp
6190: \   AND returns hx 80 -need for selection action
6191: \   3 << returns hx 400 :) -arbitrarily higher bit selected
6192: \   These values could be different for other units. So,
6193: \   use info.mem area to fetch.
6194: \   Select comparision must be 2^n for even distribution.
6195: \   Take care of overflow!
6196: \
6197: \   Decoding Algorithm:
6198: (*
6199:     noShamOp_r @ ShamOp_r @ 2 LSHIFT XOR
6200:     XOR
6201:     noShamOp_r @ ShamOp_r @ AND 3 LSHIFT AND
6202:     hx 400 =
6203:
6204: : Decode
6205:     HEX
6206:     noShamOp_r @ ShamOp_r @ 2 LSHIFT XOR
6207:     XOR
6208:     CR DUP . CR
6209:     noShamOp_r @ ShamOp_r @ AND 3 LSHIFT
6210:     TUCK AND =
6211:     ;
6212: *)
6213:
6214: DECIMAL
6215:
6216:     variable Tick 0 Tick !
6217:     variable Secs 0 Secs !
6218: \   variable Mins 0 Mins !
6219: \   TimeKeeping
6220: : Clock_Secs
6221:     DM 10 MS
6222:     Write_LED
6223:     1 Tick +!
6224:     ;
6225:
6226: \   Counting Down from initial duration value.
6227: \   Clock does not tick correctly. It is slower to a timer.
6228: \   300 seconds takes 234 secs. Check the factor after the final version.
6229: \   600 set seconds takes 847 secs. Not consistent yet :(
6230: \   That could relate to housekeeping and comms.
6231: : downClock
6232:     DECIMAL
6233: \   BEGIN-UNTIL Loop is commented out to use with the main routine loop.
6234:     BEGIN
6235:     Clock_Secs
6236: \   Tick @ DM 101 =
6237: \   Correction for timing with measured data, Ratio is 1.41
6238:     Tick @ DM 71 =
6239:     IF
6240:         -1 Secs +!
6241:         01 Tick !
6242: \       60 / Mins !
6243: \       60 MOD Secs !
6244: \       CR
6245: \       Mins @ . ." : " Secs @ .
6246:         Secs @ DUP
6247: \       DM 60 / . ." : " DM 60 MOD .
6248: \       Secs @ DUP
6249:         DM 60 / DM 100 * SWAP DM 60 MOD + .7Seg
6250:         Secs @ DUP
6251:         BN 01 AND BN 01 =
6252:         IF
6253:             1 Decipoint_On
6254: \           ." * "
6255:         THEN
6256:         BN 01 AND BN 00 =
6257:         IF
```

```
6258:             1 Decipoint_Off
6259: \           ." - "
6260:         THEN
6261:         Write_LED
6262:     THEN
6263: \   Tick @ DM 100 =
6264:     Tick @ DM 70 =
6265:     UNTIL
6266:     ;
6267:
6268: \   Needs DOUBLE numbers to work with. Final value
6269: \   averaged and placed into single variable Stimulus.
6270: \   Read Katano (Cathode - Anode potential )
6271: Variable Stimulus
6272: 0 Stimulus !
6273: : KatAno
6274:     InitADC
6275:     DN 0
6276: \   Double number initialization!
6277:     DM 100 0 DO
6278: \   HX 4000 ADC@ Stimulus +! DM 5 MS
6279:     HX 4000 ADC@
6280:     DUP
6281:     .7Seg
6282:     Write_LED
6283:     M+ DM 5 MS
6284:     LOOP
6285:     DM 100
6286:     DU/S DROP DROP
6287: \   DUP .
6288: \   Stimulus @ DM 100 /
6289: \   15 * 1000 * 1024 /
6290: \   Measured voltage in mV
6291: \   Could be hard to evaluate, not necessary.
6292: \   Instead use raw counts. 10Bit ADC, 1.5V ref.
6293:     Stimulus !
6294:     ;
6295:
6296: \   Chatter word is used to generate masking noise for relay.
6297: \   VibMotor control generates long inertial duration.
6298: : Chatter
6299:     buzout p1 1 + *BIS
6300:     HX 130 DM 40 ton
6301:     buzout p1 1 + *BIC
6302: \   CR
6303: \   ." Chatted here. "
6304: \   Secs @ DUP
6305: \   DM 60 / . ." : " DM 60 MOD .
6306: \   CR
6307: \   Periodic Chatting debug lines above (CR to CR)
6308: \   buzout direction register disables vibmotor operation as well.
6309: \   So, when calling vibmotor, make sure to redirect p1 port.
6310:     ;
6311:
6312: Variable ShamFlag false ShamFlag !
6313: : Experiment_Control
6314:         Secs @
6315:         DM 300 MOD 0 =
6316:         IF
6317:             Chatter
6318:         THEN
6319:         expDuration_r @ DM 60 *
6320:         Secs @ -
6321:         ShamLength_r @ DM 60 * =
6322:         IF
6323:             ShamFlag @
6324:             IF
6325:                 Chatter
6326:                 Relay_on
6327:                 \ ." on "
6328:                 \ Shorts/ Decouples Output Leads
6329:                 \ BluLED >LEDS
6330:                 \ Remove after test !
6331:             ELSE
6332:                 Chatter
6333:                 Relay_off
```

```
6334:                 \ ." off "
6335:                 \ tDCS Leads are intact, routed to the Subject
6336:                 \ BluLED <LEDS
6337:                 \ Remove after test !
6338:             THEN
6339:         THEN
6340:         Secs @
6341:         ShamLength_r @ DM 60 * =
6342:         IF
6343:             Chatter
6344:             Relay_off
6345:         THEN
6346:         ;
6347:
6348: : Experiment ( n -- )
6349: \   n: Sham or noSham code
6350: (*
6351: Experiment to be run with various parameters:
6352: 1. Sham or noSham parameter is fed to the word.
6353: 2. ShamLength (min) is used to deliver sham stimulus.
6354: 3. trigAmplitude is used to start downClock.
6355: 4. expDuration is total stimulation duration (min). Secs to be initialized.
6356: 5. ShowType is used to relieve stimulus type.
6357: 6. Periodic masking to be implemented.
6358: 7. Need to implement Long_Press exit.
6359: *)
6360:     DUP
6361:     Sham    = IF true   ShamFlag ! THEN
6362:     noSham  = IF false  ShamFlag ! THEN
6363:     expDuration_r @ DM 60 * Secs !
6364:     ShowType_r @
6365:     noShamOp_r @ 3 RSHIFT =
6366:     ShamFlag @
6367:     AND
6368:     IF
6369:         BluLED >LEDS
6370:     ELSE
6371:         BluLED <LEDS
6372:     THEN
6373:     0 Tick !
6374:     KatAno
6375:     false P_Press !
6376:     false P_Long_Press !
6377:     BEGIN
6378: \   Take care of TimeOut! 100 could be low.
6379: \   Key? INVERT Tick @ DM 100 > INVERT AND
6380: \   Removed Key? word.
6381:         Tick @ DM 100 > INVERT
6382:         Stimulus @ trigAmplitude_r @ <
6383:         AND
6384:         P_Press @ INVERT
6385:         AND
6386:     WHILE
6387:         KatAno
6388:         DM 10 MS
6389:         1 Tick +!
6390: \   Enables exit during initial interelectrode measurement delay.
6391: \   Returns with error, then should press PB again to exit.
6392:         P_Sw_LongPress
6393:     REPEAT
6394:     Tick @ DM 99 >
6395:     P_Press @
6396:     OR
6397:     IF
6398:         false P_Press !
6399:         false P_Long_Press !
6400:         HEX
6401:         BEGIN
6402:         P_Sw_LongPress
6403: \       HX 100 0 ?DO
6404:         HX E01 .7Seg
6405:         Write_LED
6406: \       LOOP
6407:         P_Press @
6408:         UNTIL
6409:         EXIT
```

```
6410: \   Write EXIT Procedure
6411:     THEN
6412:     false P_Press !
6413:     false P_Long_Press !
6414:     \ BluLED <LEDS
6415:     \ Remove after Experiment_Control test !
6416:     BEGIN
6417:     P_Sw_LongPress
6418:     Stimulus @ trigAmplitude_r @ >
6419:     IF
6420:         downClock
6421:         Experiment_Control
6422:     THEN
6423: \   Key? Secs @ 1 <  OR
6424: \   P_Sw_LongPress Leaves BluLED Off
6425: \   In order to have correct acknowledgement
6426: \   Check the condition that affects the process not.
6427:     P_Press @
6428:     IF
6429:         ShowType_r @
6430:         noShamOp_r @ 3 RSHIFT =
6431:         IF
6432:             BluLED >LEDS
6433:         ELSE
6434:             BluLED <LEDS
6435:         THEN
6436:     THEN
6437:     P_Long_Press @
6438:     Secs @ 1 <
6439:     OR
6440:     UNTIL
6441:     BluLED <LEDS
6442: \   End of Session Acknowledgement
6443:     VibMotor_on
6444:     DM 300 MS
6445:     VibMotor_off
6446:     BluLED >LEDS
6447:     DM 330 MS
6448:     BluLED <LEDS
6449:     VibMotor_on
6450:     DM 300 MS
6451:     VibMotor_off
6452:     BluLED >LEDS
6453:     DM 330 MS
6454:     BluLED <LEDS
6455:     Relay_off
6456:     false   ShamFlag !
6457: \   CR
6458:     DM 1000 MS
6459: \   If terminal is active. Otherwise comment out the previous word.
6460:     ;
6461:
6462: : Decoded_Sham
6463:     HEX
6464: \   Returns True or False on the stack
6465:     noShamOp_r @ ShamOp_r @ 2 LSHIFT XOR
6466:     XOR
6467: \   To display HEX data properly over terminal, HEX environment set
6468: \   CR DUP . CR
6469: \   Displays for evaluation.
6470: \   To return to proper flow, DECIMAL environment reset
6471:     noShamOp_r @ ShamOp_r @ AND 3 LSHIFT
6472:     TUCK AND =
6473: \   Compares with noShamOp_r and ShamOp_r, so that
6474: \   check bit could be highly flexible.
6475: \   HX 400 =
6476: \   Returns true if, decoded ShamValue has 10th bit set.
6477: \   Return ERROR otherwise! So, could authenticate
6478: \   noShamOp_r and ShamOp_r
6479:     DECIMAL
6480:     ;
6481:
6482: : RunSession
6483:     ShamValue_r @
6484:     noShamOp_r @ =
6485:     IF
```

```
6486:         noSham Experiment
6487:     THEN
6488:     ShamValue_r @
6489:     ShamOp_r =
6490:     IF
6491:         Sham Experiment
6492:     THEN
6493:     ShamValue_r @
6494:     Decoded_Sham
6495:     \ Returns true if key is Sham else false for noSham
6496:     IF
6497:         Sham Experiment
6498:     ELSE
6499:         noSham Experiment
6500:     THEN
6501:     false RunMode !
6502: \ Need to return error if not satisfied. Complete later.
6503: \ Make it rugged.
6504: \ Buzzer-VibMotor-GrnLED activates during ADC@ OK.
6505: \ Correct it! OK.
6506: \ Relay/ VibMotor chatter duration is long. Check it! OK.
6507: \ Make sure that Sham Decoder works correctly. Simulate with LabView as well.
6508: \ HEX DEC ambiguity must be resolved. OK.
6509: \ Need to EXIT Word implemented. OK.
6510: \ What to do?
6511: \ Hex Ambuigity clearance OK.
6512: \ Exit - Leave procedure OK.
6513: \ Robust and Graceful ending
6514: \ Timing accuracy OK.
6515: \ Decoding proof
6516: \ Decoding caused problems. Returned FALSE for any input.
6517: \ Then I've tested with Parameter_Init_Tests.f
6518: \ And moved DECIMAL word to the end of code, seems to be OK.
6519: \ 100 Ohms connection validation OK.
6520: \ 100k replacement to 10k OK.
6521: \ A known issue -did not & will not- corrected:
6522: \ While running experiment, if PB is pressed shortly,
6523: \ BlueLED will light and, resulting in turning off the BluLED,
6524: \ If ShowCode option was on.
6525: \ Remove the Terminal feedback, such as clock. OK.
6526: \ Make APP OK.
6527: \ APP could run with a KEY? Switch. OK.
6528: \ Timing correction could be implemented with MS delays. OK.
6529: \ Implement Error Propagation with code, to display actions.
6530: \ HEX environment is forced at many parts of the program. Not appealing
6531: \ but must force robustness. Later, simplify the code. OK.
6532: \ When updating the code, I've forced number types. However, SW
6533: \ selection delay is a bit short now. Need to update it for more
6534: \ friendly use.
6535:     ;
6536:
6537: : ShamAn_Main
6538:     init_ports
6539:     ini7Seg
6540:     init_Switches
6541:     7SegBlank
6542:     ReplaceShamValue
6543: \   init_Switches returns HEX stack.
6544:     HEX
6545:     CR CR CR CR
6546:     ." agnostic S h a m A n "
6547:     CR
6548:     ." Blind tDCS Sham Generator and Controller "
6549:     CR
6550:     ." Jan. 2019 Adnan Kurt "
6551:     CR
6552:     ." @Tekno F i l  #make L a b  #ISU c a n  "
6553:     CR
6554:     ." Istanbul "
6555:     CR CR
6556:     ." any keypress will return to interpreter. "
6557:     CR
6558:     ." refer to guide and design documents for forth words. "
6559:     CR CR CR
6560:     BEGIN
6561:     HEX
```

```
6562:         StandBy
6563:         ?Run_Session
6564:         ChangeMode
6565:     RunMode @ IF HEX RunSession         THEN
6566:     ModeSet @ IF HEX SelectParameters   THEN
6567:         Key?
6568: \       Remove KEY? switch later.
6569: \       Key could be useful to revoke the interpreter
6570: \       When APP is activated. Leave it.
6571:     UNTIL
6572: \   AGAIN
6573:     ;
6574:
6575: shield Enigma\
6576: ' ShamAn_Main TO APP
6577: FREEZE
6578:
6579: CHERE SWAP - .
6580:
6581: \ ******************************************************************************************
6582:
6583:
6584:
6585:
```