

```

1  /*****
2  * Lab3.c - Function generator designed with uC/OS. User can set amplitude, frequency,
3  *           and type of wave via the K65 Tower Board's touch sensing electrodes and keypad.
4  *
5  * 02/01/2018, Created project. Brian Willis
6  * 02/07/2018, Rod Mesecar added to project
7  * 02/07/2018, Anthony Needles added to project
8  *****/
9  #include "MCUType.h"
10 #include "app_cfg.h"
11 #include "os.h"
12 #include "K65TWR_GPIO.h"
13 #include "uCOSKey.h"
14 #include "LcdLayered.h"
15 #include "DMA.h"
16 #include "TSI.h"
17 #include "Wave.h"
18
19 /*****
20 * Defined Constants
21 *****/
22 #define CURSORSTART 10
23 #define UI_TASK_MSG_Q_SIZE 5
24
25 /*****
26 * Allocate task control blocks
27 *****/
28 static OS_TCB AppTaskStartTCB;
29 static OS_TCB UITaskTCB;
30 static OS_TCB UITSISrvTaskTCB;
31 static OS_TCB UIKeySrvTaskTCB;
32
33 /*****
34 * Allocate task stack space
35 *****/
36 static CPU_STK AppTaskStartStk[APP_CFG_TASK_START_STK_SIZE];
37 static CPU_STK UITaskStk[APP_CFG_UI_TASK_STK_SIZE];
38 static CPU_STK UITSISrvTaskStk[APP_CFG_UITSISRV_TASK_STK_SIZE];
39 static CPU_STK UIKeySrvTaskStk[APP_CFG_UIKEYSRV_TASK_STK_SIZE];
40
41 /*****
42 * Task Function Prototypes
43 *****/
44 static void AppStartTask(void *p_arg);
45 static void UITask(void *p_arg);
46 static void UITSISrvTask(void *p_arg);
47 static void UIKeySrvTask(void *p_arg);
48
49 /*****
50 * Private Resources
51 *****/
52 static WAVE_W dispWave;           //Local wave that displays waveform from Wave.c

```

```

53 static WAVE_W setWave;           //Local wave that gets adjusted by UI Task
54 static INT8U cursorLoc = CURSORSTART;
55
56 /*****
57 * main()
58 *****/
59 void main(void) {
60     OS_ERR os_err;
61     CPU_IntDis();                 //Disable all interrupts, OS will enable them
62     OSInit(&os_err);              //Initialize uC/OS-III
63     while(os_err != OS_ERR_NONE) {} //Error Trap
64
65     OSTaskCreate(&AppTaskStartTCB, //Address of TCB assigned to task
66                 "Start Task",      //Name you want to give the task
67                 AppStartTask,      //Address of the task itself
68                 (void *) 0,         //p_arg is not used so null ptr
69                 APP_CFG_TASK_START_PRIO, //Priority you assign to the task
70                 &AppTaskStartStk[0], //Base address of task's stack
71                 (APP_CFG_TASK_START_STK_SIZE/10u), //Watermark limit for stack growth
72                 APP_CFG_TASK_START_STK_SIZE, //Stack size
73                 0,                  //Size of task message queue
74                 0,                  //Time quanta for round robin
75                 (void *) 0,         //Extension pointer is not used
76                 (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR), //Options
77                 &os_err);           //Ptr to error code destination
78     while(os_err != OS_ERR_NONE) {} //Error Trap
79
80     OSStart(&os_err);              //Start multitasking (i.e. give control to uC/OS)
81     while(os_err != OS_ERR_NONE) {} //Error Trap
82 }
83
84 /*****
85 * This should run once and be suspended. Could restart everything by resuming.
86 * (Resuming not tested)
87 * Todd Morton, 01/06/2016
88 * Modified for Lab 3: Brian Willis, 02/16/2018
89 *****/
90 static void AppStartTask(void *p_arg) {
91     OS_ERR os_err;
92     (void)p_arg;                  //Avoid compiler warning for unused variable
93     OS_CPU_SysTickInitFreq(DEFAULT_SYSTEM_CLOCK);
94
95     /* Initialize StatTask. This must be called when there is only one task running.
96      * Therefore, any function call that creates a new task must come after this line.
97      * Or, alternatively, you can comment out this line, or remove it. If you do, you
98      * will not have accurate CPU load information */
99     // OSStatTaskCPUUsageInit(&os_err);
100
101     //Initialize peripherals
102     LcdInit();
103     KeyInit();
104     TSIIInit();

```

```

105 GpioDBugBitsInit();
106 DMAInit(*DMAWavCurSamples);
107 DMADAC0Init();
108 DMAPIT0Init();
109 WaveInit();
110
111 WaveGet(&setWave); //Initialize local waves
112 WaveGet(&dispWave);
113
114 LcdDispClear(WAVE_LAYER);
115
116 OSTaskCreate(&UITaskTCB, //Create UITask
117             "UI Task",
118             UITask,
119             (void *) 0,
120             APP_CFG_UI_TASK_PRIO,
121             &UITaskStk[0],
122             (APP_CFG_UI_TASK_STK_SIZE / 10u),
123             APP_CFG_UI_TASK_STK_SIZE,
124             UI_TASK_MSG_Q_SIZE, // Task Msg Queue
125             0,
126             (void *) 0,
127             (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
128             &os_err);
129 while(os_err != OS_ERR_NONE){} //Error Trap
130
131 OSTaskCreate(&UITSISrvTaskTCB, //Create UITSISrvTask
132             "UI Touch Service Task",
133             UITSISrvTask,
134             (void *) 0,
135             APP_CFG_UITSISRV_TASK_PRIO,
136             &UITSISrvTaskStk[0],
137             (APP_CFG_UITSISRV_TASK_STK_SIZE / 10u),
138             APP_CFG_UITSISRV_TASK_STK_SIZE,
139             0,
140             0,
141             (void *) 0,
142             (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
143             &os_err);
144 while(os_err != OS_ERR_NONE){} //Error Trap
145
146 OSTaskCreate(&UIKeySrvTaskTCB, //Create UITSISrvTask
147             "UI Key Service Task",
148             UIKeySrvTask,
149             (void *) 0,
150             APP_CFG_UIKEYSRV_TASK_PRIO,
151             &UIKeySrvTaskStk[0],
152             (APP_CFG_UIKEYSRV_TASK_STK_SIZE / 10u),
153             APP_CFG_UIKEYSRV_TASK_STK_SIZE,
154             0,
155             0,
156             (void *) 0,

```

```

157         (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
158         &os_err);
159     while(os_err != OS_ERR_NONE){}           //Error Trap
160
161     OSTaskSuspend((OS_TCB *)0, &os_err);
162     while(os_err != OS_ERR_NONE){}           //Error Trap
163 }
164
165 /*****
166 * UITask() - Controls the user interface. Displays current waveform to LCD and
167 *           updates waveform based on user inputs.
168 *
169 * 02/16/2018, Brian Willis
170 *****/
171 static void UITask(void *p_arg){
172     OS_ERR os_err;
173     INT8U *msgp;
174     OS_MSG_SIZE msg_size;
175     (void)p_arg;
176
177     while(1){
178         (void)LcdCursor(2, cursorLoc, WAVE_LAYER, TRUE, TRUE); //Display cursor
179         WaveGet(&dispWave); //Get current waveform to display its properties
180
181         //Display current waveform to LCD
182         LcdDispDecByte(1, 2, WAVE_LAYER, dispWave.amp, 1);
183         LcdDispString(1, 1, WAVE_LAYER, "A:");
184         //Display lower 2 chars of frequency
185         LcdDispDecByte(1, CURSORSTART+2, WAVE_LAYER, (INT8U)(dispWave.freq-((dispWave.freq/100)*100)), 1);
186         //Middle 2 chars
187         LcdDispDecByte(1, CURSORSTART, WAVE_LAYER, (INT8U)((((dispWave.freq-((dispWave.freq/10000)*10000))
188                                                                -(dispWave.freq-((dispWave.freq/100)*100)))/100), 1);
189         //Upper char
190         LcdDispDecByte(1, CURSORSTART-2, WAVE_LAYER, (INT8U)((dispWave.freq)/10000), 1);
191         LcdDispString(1, 8, WAVE_LAYER, "F:");
192         LcdDispString(1, 15, WAVE_LAYER, "Hz");
193         if(dispWave.waveshape == SIN){
194             LcdDispString(2, 1, WAVE_LAYER, "SINE");
195         } else{
196             LcdDispString(2, 1, WAVE_LAYER, "TRI ");
197         }
198
199         //Frequency being changed by user
200         LcdDispDecByte(2, CURSORSTART+2, WAVE_LAYER, (INT8U)(setWave.freq-((setWave.freq/100)*100)), 1);
201         LcdDispDecByte(2, CURSORSTART, WAVE_LAYER, (INT8U)((((setWave.freq-((setWave.freq/10000)*10000))
202                                                                -(setWave.freq-((setWave.freq/100)*100)))/100), 1);
203         LcdDispDecByte(2, CURSORSTART-2, WAVE_LAYER, (INT8U)((setWave.freq)/10000), 1);
204         LcdDispString(2, 6, WAVE_LAYER, "Set:");
205         LcdDispString(2, 15, WAVE_LAYER, "Hz");
206
207         //Turn off debug bit while waiting
208         DB4_TURN_OFF();

```

```

209 //Wait for either key press or TSI
210 msgp = OSTaskQPend(0, OS_OPT_PEND_BLOCKING, &msg_size, (CPU_TS *)0, &os_err);
211 //Error Trap
212 while(os_err != OS_ERR_NONE){}
213 //Turn on debug bit while ready/running
214 DB4_TURN_ON();
215
216 switch(*msgp){
217     case(1): //Left electrode touched
218         if(setWave.amp < 20){
219             setWave.amp++;
220             WaveSet(&setWave, 1);
221         } else{}
222         break;
223     case(2): //Right electrode touched
224         if(setWave.amp > 0){
225             setWave.amp--;
226             WaveSet(&setWave, 1);
227         } else{}
228         break;
229     default: //Key press
230         if(*msgp == 0x11){ // 'A' - Set to sinewave
231             setWave.waveshape = SIN;
232             WaveSet(&setWave, 3);
233         } else if(*msgp == 0x12){ // 'B' - Set to triangle wave
234             setWave.waveshape = TRI;
235             WaveSet(&setWave, 3);
236         } else if((*msgp >= 48) && (*msgp <= 57)){ //0-9
237             *msgp = *msgp - 48; //Convert ASCII to decimal
238             switch(cursorLoc){ //Current location of cursor determines digit to update
239                 case(CURSORSTART): //Update 10,000s place
240                     setWave.freq = ((*msgp)*10000)+(setWave.freq-((setWave.freq/10000)*10000));
241                     cursorLoc++;
242                     break;
243                 case(CURSORSTART+1): //1,000s
244                     setWave.freq =
245                         *msgp*1000+((setWave.freq/10000)*10000)+(setWave.freq-((setWave.freq/1000)*1000));
246                     cursorLoc++;
247                     break;
248                 case(CURSORSTART+2): //100s
249                     setWave.freq =
250                         *msgp*100+((setWave.freq/1000)*1000)+(setWave.freq-((setWave.freq/100)*100));
251                     cursorLoc++;
252                     break;
253                 case(CURSORSTART+3): //10s
254                     setWave.freq = *msgp*10+((setWave.freq/100)*100)+(setWave.freq-((setWave.freq/10)*10));
255                     cursorLoc++;
256                     break;
257                 case(CURSORSTART+4): //1s
258                     setWave.freq = *msgp+((setWave.freq/10)*10);
259                     break;

```

```

259         } else if(*msgp == 0x14){                                //'D' - Backspace
260             if(cursorLoc > CURSORSTART){
261                 cursorLoc--;
262             } else{}
263         } else if(*msgp == 0x23){                                //'#' - Enter
264             if(setWave.freq > 10000){
265                 setWave.freq = 10000;
266             } else if(setWave.freq < 10){
267                 setWave.freq = 10;
268             } else{
269                 WaveSet(&setWave, 2);
270                 dispWave = setWave;
271             }
272             cursorLoc = CURSORSTART;
273         } else{}
274         break;
275     }
276 }
277 }
278
279 /*****
280 * UITSISrvTask() - Pends on TouchPend and updates UIInputQ. Gives TSI data to UITask().
281 *
282 * 02/09/2018, Rod Mesecar
283 * 02/16/2018, Modified for UITask(). Brian Willis
284 *****/
285 static void UITSISrvTask(void *p_arg){
286     OS_ERR os_err;
287     (void)p_arg;
288     INT8U tsipress;
289     OS_MSG_SIZE msg_size = sizeof(tsipress);
290
291     while(1){
292         DB5_TURN_OFF();
293         tsipress = TSITouchPend(&os_err);
294         while(os_err != OS_ERR_NONE){}          //Error Trap
295         DB5_TURN_OFF();
296
297         OSTaskQPost(&UITaskTCB, &tsipress, msg_size, OS_OPT_POST_FIFO, &os_err);    //Place TSI data into queue
298         while(os_err != OS_ERR_NONE){}          //Error Trap
299     }
300 }
301
302 /*****
303 * UIKeySrvTask() - Pends on KeyPend and updates UIInputQ. Gives keypad data to UITask().
304 *
305 * 02/09/2018, Rod Mesecar
306 * 02/16/2018, Modified for UITask(). Brian Willis
307 *****/
308 static void UIKeySrvTask(void *p_arg){
309     OS_ERR os_err;
310     (void)p_arg;

```

```
311     INT8U keypress = 0;
312     OS_MSG_SIZE msg_size = sizeof(keypress);
313
314     while(1){
315         DB6_TURN_OFF();
316         keypress = KeyPend(0, &os_err);      //Wait for key press
317         while(os_err != OS_ERR_NONE){}      //Error Trap
318         DB6_TURN_OFF();
319
320         OSTaskQPost(&UITaskTCB, &keypress, msg_size, OS_OPT_POST_FIFO, &os_err);    //Place keypress into queue
321         while(os_err != OS_ERR_NONE){}      //Error Trap
322     }
323 }
324
```