```
1  /*This program allows the microcontroller onboard the transmitter board to take infomation from the Nunchuk controller
2    and issue the appropriate commands via the magnetic field */
3
4  #include <C8051F38x.h>
5  #include <stdlib.h>
6  #include <stdio.h>
7
8  #define SYSCLK    48000000L // SYSCLK frequency in Hz
9  #define BAUDRATE  115200L   // Baud rate of UART in bps for I2C not fro Magnetic field transmission
10 #define DEFAULT_F 15500L
11 #define  SMB_FREQUENCY  100000L   // I2C SCL clock rate (10kHz to 100kHz)
12 #define  START_SIGNAL 0b_11111 // Signal that indicates valid inofmation is about to be transmitted
13 #define  RESON_FREQUENCY  0x10000L-(SYSCLK/(2L*14950)) /* The value that must be loaded into timer 2 reload
14                                                          to ensure LC H-bridge is oscillated at resonance frequency fo 14.95 KHz */
15
16
17 #define TRANSMIT_RATE  10 //Length of each bit transmitted in milliseconds
18
19 #define  LED        P2_2
20 #define  LED_ON     0
21 #define  LED_OFF    1
22 #define OUT0 P2_0
23 #define OUT1 P2_1
24
25 #define XMIN 127.0
26 #define XMAX 127.0
27 #define YMIN 127.0
28 #define YMAX 127.0
29 #define GOTO_YX "\x1B[%d;%dH"
30
31 volatile float LF;
32 volatile float LB;
33 volatile float RF;
34 volatile float RB;
35 int convert[12];
36
37 volatile unsigned char pwm_count=0;
38
39 char _c51_external_startup (void)
40 {
41     PCA0MD&=(~0x40) ;    // DISABLE WDT: clear Watchdog Enable bit
42     VDM0CN=0x80; // enable VDD monitor
43     RSTSRC=0x02|0x04; // Enable reset on missing clock detector and VDD
44
45     // CLKSEL&=0b_1111_1000; // Not needed because CLKSEL==0 after reset
46     #if (SYSCLK == 12000000L)
47         //CLKSEL|=0b_0000_0000;  // SYSCLK derived from the Internal High-Frequency Oscillator / 4
48     #elif (SYSCLK == 24000000L)
49         CLKSEL|=0b_0000_0010; // SYSCLK derived from the Internal High-Frequency Oscillator / 2.
50     #elif (SYSCLK == 48000000L)
51         CLKSEL|=0b_0000_0011; // SYSCLK derived from the Internal High-Frequency Oscillator / 1.
52     #else
53         #error SYSCLK must be either 12000000L, 24000000L, or 48000000L
54     #endif
55     OSCICN |= 0x03; // Configure internal oscillator for its maximum frequency
56
57     // Configure UART0
58     SCON0 = 0x10;
59 #if (SYSCLK/BAUDRATE/2L/256L < 1)
60     TH1 = 0x10000-((SYSCLK/BAUDRATE)/2L);
61     CKCON &= ~0x0B;                  // T1M = 1; SCA1:0 = xx
62     CKCON |=  0x08;
63 #elif (SYSCLK/BAUDRATE/2L/256L < 4)
64     TH1 = 0x10000-(SYSCLK/BAUDRATE/2L/4L);
65     CKCON &= ~0x0B; // T1M = 0; SCA1:0 = 01
66     CKCON |=  0x01;
67 #elif (SYSCLK/BAUDRATE/2L/256L < 12)
```

```
68      TH1 = 0x10000-(SYSCLK/BAUDRATE/2L/12L);
69      CKCON &= ~0x0B; // T1M = 0; SCA1:0 = 00
70  #else
71      TH1 = 0x10000-(SYSCLK/BAUDRATE/2/48);
72      CKCON &= ~0x0B; // T1M = 0; SCA1:0 = 10
73      CKCON |=  0x02;
74  #endif
75      TL1 = TH1;      // Init Timer1
76      TMOD &= ~0xf0;  // TMOD: timer 1 in 8-bit autoreload
77      TMOD |=  0x20;
78      TR1 = 1; // START Timer1
79      TI = 1;  // Indicate TX0 ready
80
81      // Configure the pins used for square output
82      P2MDOUT|=0b_0000_0011;
83      P0MDOUT |= 0x10; // Enable UTX as push-pull output
84      P2MDOUT |= 0b0000_0111;   // Make the LED (P2.2) a push-pull output
85      XBR0 = 0b0000_0101;       // Enable SMBus pins and UART pins P0.4(TX) and P0.5(RX)
86      XBR1     = 0x40; // Enable crossbar and weak pull-ups
87
88      // Configure Timer 0 as the I2C clock source
89      CKCON |= 0x04; // Timer0 clock source = SYSCLK
90      TMOD &= 0xf0;  // Mask out timer 1 bits
91      TMOD |= 0x02;  // Timer0 in 8-bit auto-reload mode
92      // Timer 0 configured to overflow at 1/3 the rate defined by SMB_FREQUENCY
93      TL0 = TH0 = 256-(SYSCLK/SMB_FREQUENCY/3);
94      TR0 = 1; // Enable timer 0
95
96      // Configure and enable SMBus
97      SMB0CF = INH | EXTHOLD | SMBTOE | SMBFTE ;
98      SMB0CF |= ENSMB;  // Enable SMBus
99
100     // Initialize timer 2 for periodic interrupts
101     TMR2CN=0x00;   // Stop Timer2; Clear TF2;
102     CKCON|=0b_0001_0000;
103     TMR2RL= RESON_FREQUENCY; // Initialize reload value
104     TMR2=0xffff;   // Set to reload immediately
105     ET2=1;         // Enable Timer2 interrupts
106     TR2=0;         //Don't Start Timer2
107
108     EA=1; // Enable interrupts
109
110
111     LED = LED_OFF;
112
113     EA=1; // Enable interrupts
114
115     return 0;
116 }
117 void Timer4ms(unsigned char ms) // Makes program wait specified amount of milliseconds
118 {
119     unsigned char i;// usec counter
120     unsigned char k;
121
122     k=SFRPAGE;
123     SFRPAGE=0xf;
124     // The input for Timer 4 is selected as SYSCLK by setting bit 0 of CKCON1:
125     CKCON1|=0b_0000_0001;
126
127     TMR4RL = 65536-(SYSCLK/1000L); // Set Timer4 to overflow in 1 ms.
128     TMR4 = TMR4RL;                 // Initialize Timer4 for first overflow
129
130     TMR4CN = 0x04;                 // Start Timer4 and clear overflow flag
131     for (i = 0; i < ms; i++)       // Count <ms> overflows
132     {
133         while (!(TMR4CN & 0x80));  // Wait for overflow
134         TMR4CN &= ~(0x80);         // Clear overflow indicator
135     }
136     TMR4CN = 0 ;                   // Stop Timer4 and clear overflow flag
```

```
137     SFRPAGE=k;
138 }
139 void I2C_write (unsigned char output_data)
140 {
141     SMB0DAT = output_data; // Put data into buffer
142     SI0 = 0;
143     while (!SI0); // Wait until done with send
144 }
145 unsigned char I2C_read (void)
146 {
147     unsigned char input_data;
148 
149     SI0 = 0;
150     while (!SI0); // Wait until we have data to read
151     input_data = SMB0DAT; // Read the data
152 
153     return input_data;
154 }
155 void I2C_start (void)
156 {
157     ACK0 = 1;
158     STA0 = 1;     // Send I2C start
159     STO0 = 0;
160     SI0 = 0;
161     while (!SI0); // Wait until start sent
162     STA0 = 0;     // Reset I2C start
163 }
164 void I2C_stop(void)
165 {
166     STO0 = 1;   // Perform I2C stop
167     SI0 = 0;    // Clear SI
168     //while (!SI0);    // Wait until stop complete (Doesn't work???)
169 }
170 void nunchuck_init(bit print_extension_type)
171 {
172     unsigned char i, buf[6];
173     // Newer initialization format that works for all nunchucks
174     I2C_start();
175     I2C_write(0xA4);
176     I2C_write(0xF0);
177     I2C_write(0x55);
178     I2C_stop();
179     Timer4ms(1);
180 
181     I2C_start();
182     I2C_write(0xA4);
183     I2C_write(0xFB);
184     I2C_write(0x00);
185     I2C_stop();
186     Timer4ms(1);
187 
188     // Read the extension type from the register block.  For the original Nunchuk it should be
189     // 00 00 a4 20 00 00.
190     I2C_start();
191     I2C_write(0xA4);
192     I2C_write(0xFA); // extension type register
193     I2C_stop();
194     Timer4ms(3); // 3 ms required to complete acquisition
195 
196     I2C_start();
197     I2C_write(0xA5);
198 
199     // Receive values
200     for(i=0; i<6; i++)
201     {
202         buf[i]=I2C_read();
203     }
204     ACK0=0;
205     I2C_stop();
```

```
206     Timer4ms(3);
207
208     if(print_extension_type)
209     {
210         printf("Extension type: %02x  %02x  %02x  %02x  %02x  %02x\n",
211             buf[0],  buf[1], buf[2], buf[3], buf[4], buf[5]);
212     }
213
214     // Send the crypto key (zeros), in 3 blocks of 6, 6 & 4.
215
216     I2C_start();
217     I2C_write(0xA4);
218     I2C_write(0xF0);
219     I2C_write(0xAA);
220     I2C_stop();
221     Timer4ms(1);
222
223     I2C_start();
224     I2C_write(0xA4);
225     I2C_write(0x40);
226     I2C_write(0x00);
227     I2C_write(0x00);
228     I2C_write(0x00);
229     I2C_write(0x00);
230     I2C_write(0x00);
231     I2C_write(0x00);
232     I2C_stop();
233     Timer4ms(1);
234
235     I2C_start();
236     I2C_write(0xA4);
237     I2C_write(0x40);
238     I2C_write(0x00);
239     I2C_write(0x00);
240     I2C_write(0x00);
241     I2C_write(0x00);
242     I2C_write(0x00);
243     I2C_write(0x00);
244     I2C_stop();
245     Timer4ms(1);
246
247     I2C_start();
248     I2C_write(0xA4);
249     I2C_write(0x40);
250     I2C_write(0x00);
251     I2C_write(0x00);
252     I2C_write(0x00);
253     I2C_write(0x00);
254     I2C_stop();
255     Timer4ms(1);
256
257 }
258 int round_to_ten( int num) // Actually assigns power values between 1-100 to values between 0-7 so they can ↙
        be transmitted as 3-bit binary
259 {
260     return num/14;
261 }
262 void nunchuck_getdata(unsigned char * s) // Retrieves current values from nunchuk
263 {
264     unsigned char i;
265
266     // Start measurement
267     I2C_start();
268     I2C_write(0xA4);
269     I2C_write(0x00);
270     I2C_stop();
271     Timer4ms(3);    // 3 ms required to complete acquisition
272
273     // Request values
```

```
274     I2C_start();
275     I2C_write(0xA5);
276 
277     // Receive values
278     for(i=0; i<6; i++)
279     {
280         s[i]=(I2C_read()^0x17)+0x17; // Read and decrypt
281     }
282     ACK0=0;
283     I2C_stop();
284 }
285 void send_bit(int b) // Sends a single bit via the magnetic field
286 {
287     if(b == 1) // If sending a logic 1
288     {
289         OUT1 = 0; //Ensure the pins to each H-bridge side are opposite
290         OUT0 = 1;
291         TR2 = 1; //Enable timer 2 which causes the two pins above to oscillate
292         Timer4ms(TRANSMIT_RATE-2); // Cut short to account for RC lag
293         TR2 = 0;
294         Timer4ms(2);
295     }
296     else //If sending a logic 0
297     {
298         OUT1 = 0; // Ensure no voltage across LC
299         OUT0 = 0;
300         Timer4ms(TRANSMIT_RATE);
301     }
302     TR2 = 0; // Disables off timer 2
303     OUT1 = 0;
304     OUT0 = 0;
305 }
306 void send_int(int num, int size) // Converts an integer to binary, places it in an array and send the ↙
    contents of the array bit by bit via the magnetic field
307 {
308     int i;
309     for(i = size; i>0; i--)
310     {
311         convert[i-1] = num%2;
312         num = num / 2;
313     }
314     for(i = 0;i<size;i++)
315     {
316         send_bit(convert[i]);
317     }
318 }
319 void Timer2_ISR (void) interrupt 5 // Allows program to turn magnetic field on and off
320 {
321     unsigned char page_save;
322     page_save=SFRPAGE;
323     SFRPAGE=0;
324     TF2H = 0; // Clear Timer2 interrupt flag
325     OUT0=!OUT0;//Invert the pin to one side of the LC H-Bridge
326     OUT1=!OUT1;// Invert the pin other H-bridge pin
327     SFRPAGE=page_save;
328 }
329 void main (void)
330 {
331     unsigned char rbuf[6];
332     int joy_x, joy_y, off_x, off_y, acc_x, acc_y, acc_z;
333     bit but1, but2;
334     bit l_dir, r_dir;
335     int left_power, right_power;
336     int power=0;
337 
338     EA = 1;
339 
340     Timer4ms(200);
341     nunchuck_init(1);
```

```
342     Timer4ms(100);
343
344     nunchuck_getdata(rbuf);
345
346     off_x=(int)rbuf[0]-128;
347     off_y=(int)rbuf[1]-128;
348
349     while(1)
350     {
351
352         nunchuck_getdata(rbuf); // Retrieves data from nunchuk
353         joy_x=(int)rbuf[0]-128-off_x; // MOves all data from nunchuk buffer to respective variables
354         joy_y=(int)rbuf[1]-128-off_y;
355         acc_x=rbuf[2]*4;
356         acc_y=rbuf[3]*4;
357         acc_z=rbuf[4]*4;
358         but1=!((rbuf[5] & 0x01)?1:0);
359         but2=!((rbuf[5] & 0x02)?1:0);
360         if (rbuf[5] & 0x04) acc_x+=2;
361         if (rbuf[5] & 0x08) acc_x+=1;
362         if (rbuf[5] & 0x10) acc_y+=2;
363         if (rbuf[5] & 0x20) acc_y+=1;
364         if (rbuf[5] & 0x40) acc_z+=2;
365         if (rbuf[5] & 0x80) acc_z+=1;
366         LF = 0;
367         RF = 0;
368         LB = 0;
369         RB = 0;
370
371       if(but1) //Activates field to enter Tracking Mode
372         {
373
374             printf("\nTracking Mode!\n");
375             Timer4ms(50);
376             while(but1) // Debounces Button 1
377             {
378                 nunchuck_getdata(rbuf);
379                 but1 = !((rbuf[5] & 0x01)?1:0);
380                 Timer4ms(50);
381             }
382             OUT1 = 0;
383             OUT0 = 1;
384             TR2 = 1;
385             Timer4ms(50);
386             while(!but1) // Waits for command from button 1 to return to nunchuk mode
387             {
388             nunchuck_getdata(rbuf);
389             but1 = !((rbuf[5] & 0x01)?1:0);
390             Timer4ms(50);
391             }
392             Timer4ms(50);
393             while(but1) // Debounces Button 1
394             {
395                 nunchuck_getdata(rbuf);
396                 but1 = !((rbuf[5] & 0x01)?1:0);
397                 Timer4ms(50);
398             }
399             OUT1 =0;
400             OUT0 = 0;
401             TR2 = 0;
402             printf("\nControl!\n");
403         }
404     else if(but2) // Disables field and enters IR mode
405         {
406
407             printf("\nIR Mode!\n");
408             Timer4ms(50);
409             while(but2) // Debounces Button 2
410             {
```

```
411                 nunchuck_getdata(rbuf);
412                 but2=!((rbuf[5] & 0x02)?1:0);
413                 Timer4ms(50);
414             }
415             OUT1 = 0; //Ensures field is disabled
416             OUT0 = 0;
417             TR2 = 0;
418             Timer4ms(50);
419             while(!but2) // Waits for command from button 2 to return to nunchuk mode
420             {
421             nunchuck_getdata(rbuf);
422             but2=!((rbuf[5] & 0x02)?1:0);
423             Timer4ms(50);
424             }
425             Timer4ms(50);
426             while(but2) // Debounces Button 2
427             {
428                 nunchuck_getdata(rbuf);
429                 but2=!((rbuf[5] & 0x02)?1:0);
430                 Timer4ms(50);
431             }
432             OUT1 =0;
433             OUT0 = 0;
434             TR2 = 0;
435             printf("\nControl!\n");
436         }
437     else // Assumes it is in Nunchuk mode if it gets to here
438         //Converts numbers received from nunchuk to values for transmission protocol
439     {
440
441         l_dir = 1;
442         r_dir = 1;
443         power = 0;
444
445         if(joy_x > 5)
446         {
447             l_dir = 1;
448             r_dir = 0;
449             power = (int) 1*((float)joy_x/127)*100/14;
450         }
451         else if(joy_x < -5)
452         {
453             l_dir = 0;
454             r_dir = 1;
455             power = (int) -1*((float)joy_x/127)*100/14;
456         }
457
458         if(joy_y > 5)
459         {
460             l_dir = 1;
461             r_dir = 1;
462             power = (int) 1*((float)joy_y/127)*100/14;
463         }
464         else if(joy_y < -5)
465         {
466             l_dir = 0;
467             r_dir = 0;
468             power = (int) -1*((float)joy_y/127)*100/14;
469         }
470
471         send_int(START_SIGNAL, 5); // Transmitted data per transmission protocol
472         send_bit(0);
473         send_bit(but1);
474         send_bit(but2);
475         send_bit(l_dir);
476         send_bit(r_dir);
477         send_bit(0);
478         send_bit(0);
479         send_int(power,3);
```

```
480         send_int(0, 4);
481     }
482 
483   }
484 }
```