```
// GPIO Library
// Jason Losh

//-----------------------------------------------------------------------------
// Hardware Target
//-----------------------------------------------------------------------------

// Target Platform: EK-TM4C123GXL with LCD/Keyboard Interface
// Target uC:       TM4C123GH6PM
// System Clock:    40 MHz

// Hardware configuration:
// GPIO APB ports A-F

//-----------------------------------------------------------------------------
// Device includes, defines, and assembler directives
//-----------------------------------------------------------------------------

#include <stdint.h>
#include <stdbool.h>
#include "tm4c123gh6pm.h"
#include "gpio.h"

// Bit offset of the registers relative to bit 0 of DATA_R at 3FCh
#define OFS_DATA_TO_DIR    1*4*8
#define OFS_DATA_TO_IS     2*4*8
#define OFS_DATA_TO_IBE    3*4*8
#define OFS_DATA_TO_IEV    4*4*8
#define OFS_DATA_TO_IM     5*4*8
#define OFS_DATA_TO_AFSEL  9*4*8
#define OFS_DATA_TO_ODR   68*4*8
#define OFS_DATA_TO_PUR   69*4*8
#define OFS_DATA_TO_PDR   70*4*8
#define OFS_DATA_TO_DEN   72*4*8
#define OFS_DATA_TO_CR    74*4*8
#define OFS_DATA_TO_AMSEL 75*4*8

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Subroutines
//-----------------------------------------------------------------------------

void enablePort(PORT port)
{
    switch(port)
    {
        case PORTA:
            SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R0;
            SYSCTL_GPIOHBCTL_R &= ~1;
            break;
        case PORTB:
            SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R1;
            SYSCTL_GPIOHBCTL_R &= ~2;
            break;
        case PORTC:
            SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R2;
            SYSCTL_GPIOHBCTL_R &= ~4;
            break;
        case PORTD:
            SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R3;
            SYSCTL_GPIOHBCTL_R &= ~8;
            break;
        case PORTE:
            SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R4;
            SYSCTL_GPIOHBCTL_R &= ~16;
            break;
```

```
70          case PORTF:
71              SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R5;
72              SYSCTL_GPIOHBCTL_R &= ~32;
73      }
74      _delay_cycles(3);
75  }
76
77  void disablePort(PORT port)
78  {
79      switch(port)
80      {
81          case PORTA:
82              SYSCTL_RCGCGPIO_R &= ~SYSCTL_RCGCGPIO_R0;
83              break;
84          case PORTB:
85              SYSCTL_RCGCGPIO_R &= ~SYSCTL_RCGCGPIO_R1;
86              break;
87          case PORTC:
88              SYSCTL_RCGCGPIO_R &= ~SYSCTL_RCGCGPIO_R2;
89              break;
90          case PORTD:
91              SYSCTL_RCGCGPIO_R &= ~SYSCTL_RCGCGPIO_R3;
92              break;
93          case PORTE:
94              SYSCTL_RCGCGPIO_R &= ~SYSCTL_RCGCGPIO_R4;
95              break;
96          case PORTF:
97              SYSCTL_RCGCGPIO_R &= ~SYSCTL_RCGCGPIO_R5;
98      }
99      _delay_cycles(3);
100 }
101
102 void selectPinPushPullOutput(PORT port, uint8_t pin)
103 {
104     uint32_t* p;
105     p = (uint32_t*)port + pin + OFS_DATA_TO_ODR;
106     *p = 0;
107     p = (uint32_t*)port + pin + OFS_DATA_TO_DIR;
108     *p = 1;
109     p = (uint32_t*)port + pin + OFS_DATA_TO_DEN;
110     *p = 1;
111 }
112
113 void selectPinOpenDrainOutput(PORT port, uint8_t pin)
114 {
115     uint32_t* p;
116     p = (uint32_t*)port + pin + OFS_DATA_TO_ODR;
117     *p = 1;
118     p = (uint32_t*)port + pin + OFS_DATA_TO_DIR;
119     *p = 1;
120     p = (uint32_t*)port + pin + OFS_DATA_TO_DEN;
121     *p = 1;
122 }
123
124 void selectPinDigitalInput(PORT port, uint8_t pin)
125 {
126     uint32_t* p;
127     p = (uint32_t*)port + pin + OFS_DATA_TO_DIR;
128     *p = 0;
129     p = (uint32_t*)port + pin + OFS_DATA_TO_DEN;
130     *p = 1;
131     p = (uint32_t*)port + pin + OFS_DATA_TO_AMSEL;
132     *p = 0;
133 }
134
135 void selectPinAnalogInput(PORT port, uint8_t pin)
136 {
137     uint32_t* p;
138     p = (uint32_t*)port + pin + OFS_DATA_TO_DEN;
```

```
    *p = 0;
    p = (uint32_t*)port + pin + OFS_DATA_TO_AMSEL;
    *p = 1;
    p = (uint32_t*)port + pin + OFS_DATA_TO_AFSEL;
    *p = 1;
}

void setPinCommitControl(PORT port, uint8_t pin)
{
    switch(port)
    {
        case PORTA:
            GPIO_PORTA_LOCK_R = GPIO_LOCK_KEY;
            break;
        case PORTB:
            GPIO_PORTB_LOCK_R = GPIO_LOCK_KEY;
            break;
        case PORTC:
            GPIO_PORTC_LOCK_R = GPIO_LOCK_KEY;
            break;
        case PORTD:
            GPIO_PORTD_LOCK_R = GPIO_LOCK_KEY;
            break;
        case PORTE:
            GPIO_PORTE_LOCK_R = GPIO_LOCK_KEY;
            break;
        case PORTF:
            GPIO_PORTF_LOCK_R = GPIO_LOCK_KEY;
    }
    uint32_t* p;
    p = (uint32_t*)port + pin + OFS_DATA_TO_CR;
    *p = 1;
}

void enablePinPullup(PORT port, uint8_t pin)
{
    uint32_t* p;
    p = (uint32_t*)port + pin + OFS_DATA_TO_PUR;
    *p = 1;
}

void disablePinPullup(PORT port, uint8_t pin)
{
    uint32_t* p;
    p = (uint32_t*)port + pin + OFS_DATA_TO_PUR;
    *p = 0;
}

void enablePinPulldown(PORT port, uint8_t pin)
{
    uint32_t* p;
    p = (uint32_t*)port + pin + OFS_DATA_TO_PDR;
    *p = 1;
}

void disablePinPulldown(PORT port, uint8_t pin)
{
    uint32_t* p;
    p = (uint32_t*)port + pin + OFS_DATA_TO_PDR;
    *p = 0;
}

void setPinAuxFunction(PORT port, uint8_t pin, uint32_t fn)
{
    // call with header file shifted values or 4-bit number
    if (fn <= 15)
        fn = fn << (pin*4);
    else
        fn = fn & (0x0000000F << (pin*4));
```

```
208         switch(port)
209         {
210             case PORTA:
211                 GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R & ~(0x0000000F << (pin*4))) | fn;
212                 break;
213             case PORTB:
214                 GPIO_PORTB_PCTL_R = (GPIO_PORTB_PCTL_R & ~(0x0000000F << (pin*4))) | fn;
215                 break;
216             case PORTC:
217                 GPIO_PORTC_PCTL_R = (GPIO_PORTC_PCTL_R & ~(0x0000000F << (pin*4))) | fn;
218                 break;
219             case PORTD:
220                 GPIO_PORTD_PCTL_R = (GPIO_PORTD_PCTL_R & ~(0x0000000F << (pin*4))) | fn;
221                 break;
222             case PORTE:
223                 GPIO_PORTE_PCTL_R = (GPIO_PORTE_PCTL_R & ~(0x0000000F << (pin*4))) | fn;
224                 break;
225             case PORTF:
226                 GPIO_PORTF_PCTL_R = (GPIO_PORTF_PCTL_R & ~(0x0000000F << (pin*4))) | fn;
227         }
228         // set AFSEL bit only if using aux function, otherwise clear bit
229         uint32_t* p;
230         p = (uint32_t*)port + pin + OFS_DATA_TO_AFSEL;
231         *p = (fn > 0);
232     }
233
234     void selectPinInterruptRisingEdge(PORT port, uint8_t pin)
235     {
236         uint32_t* p;
237         p = (uint32_t*)port + pin + OFS_DATA_TO_IS;
238         *p = 0;
239         p = (uint32_t*)port + pin + OFS_DATA_TO_IBE;
240         *p = 0;
241         p = (uint32_t*)port + pin + OFS_DATA_TO_IEV;
242         *p = 1;
243     }
244
245     void selectPinInterruptFallingEdge(PORT port, uint8_t pin)
246     {
247         uint32_t* p;
248         p = (uint32_t*)port + pin + OFS_DATA_TO_IS;
249         *p = 0;
250         p = (uint32_t*)port + pin + OFS_DATA_TO_IBE;
251         *p = 0;
252         p = (uint32_t*)port + pin + OFS_DATA_TO_IEV;
253         *p = 0;
254     }
255
256     void selectPinInterruptBothEdges(PORT port, uint8_t pin)
257     {
258         uint32_t* p;
259         p = (uint32_t*)port + pin + OFS_DATA_TO_IS;
260         *p = 0;
261         p = (uint32_t*)port + pin + OFS_DATA_TO_IBE;
262         *p = 1;
263     }
264
265     void selectPinInterruptHighLevel(PORT port, uint8_t pin)
266     {
267         uint32_t* p;
268         p = (uint32_t*)port + pin + OFS_DATA_TO_IS;
269         *p = 1;
270         p = (uint32_t*)port + pin + OFS_DATA_TO_IEV;
271         *p = 1;
272     }
273
274     void selectPinInterruptLowLevel(PORT port, uint8_t pin)
275     {
276         uint32_t* p;
```

```
277         p = (uint32_t*)port + pin + OFS_DATA_TO_IS;
278         *p = 1;
279         p = (uint32_t*)port + pin + OFS_DATA_TO_IEV;
280         *p = 0;
281     }
282
283     void enablePinInterrupt(PORT port, uint8_t pin)
284     {
285         uint32_t* p;
286         p = (uint32_t*)port + pin + OFS_DATA_TO_IM;
287         *p = 1;
288     }
289
290     void disablePinInterrupt(PORT port, uint8_t pin)
291     {
292         uint32_t* p;
293         p = (uint32_t*)port + pin + OFS_DATA_TO_IM;
294         *p = 0;
295     }
296
297     void setPinValue(PORT port, uint8_t pin, bool value)
298     {
299         uint32_t* p;
300         p = (uint32_t*)port + pin;
301         *p = value;
302     }
303
304     bool getPinValue(PORT port, uint8_t pin)
305     {
306         uint32_t* p;
307         p = (uint32_t*)port + pin;
308         return *p;
309     }
310
311     void setPortValue(PORT port, uint8_t value)
312     {
313         switch(port)
314         {
315             case PORTA:
316                 GPIO_PORTA_DATA_R = value;
317                 break;
318             case PORTB:
319                 GPIO_PORTB_DATA_R = value;
320                 break;
321             case PORTC:
322                 GPIO_PORTC_DATA_R = value;
323                 break;
324             case PORTD:
325                 GPIO_PORTD_DATA_R = value;
326                 break;
327             case PORTE:
328                 GPIO_PORTE_DATA_R = value;
329                 break;
330             case PORTF:
331                 GPIO_PORTF_DATA_R = value;
332         }
333     }
334
335     uint8_t getPortValue(PORT port)
336     {
337         uint8_t value;
338         switch(port)
339         {
340             case PORTA:
341                 value = GPIO_PORTA_DATA_R;
342                 break;
343             case PORTB:
344                 value = GPIO_PORTB_DATA_R;
345                 break;
```

```
346             case PORTC:
347                 value = GPIO_PORTC_DATA_R;
348                 break;
349             case PORTD:
350                 value = GPIO_PORTD_DATA_R;
351                 break;
352             case PORTE:
353                 value = GPIO_PORTE_DATA_R;
354                 break;
355             case PORTF:
356                 value = GPIO_PORTF_DATA_R;
357         }
358         return value;
359     }
360
```