```
//*****************************************************************************
//
// gpio.c - Simple interrupt-driven GPIO example.
//
// Copyright (c) 2006-2011 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 7243 of the EK-LM3S6965 Firmware Package.
//
//*****************************************************************************

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"


//*****************************************************************************
//
// The error routine that is called if the driver library encounters an error.
//
//*****************************************************************************
#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif

volatile int flag = 1;

void handle_gpio(void);

//*****************************************************************************
//
// GPIO setup
//
//*****************************************************************************
void GPIO_setup(void)
{
    // Enable the GPIO peripheral used by this
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    // Register interrupt function
    GPIOPortIntRegister(GPIO_PORTF_BASE, handle_gpio);
    // Configure push-button as INPUT
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_1);
    // Enable pin to interrupt on rising or falling edge (YOU DETERMINE)
    GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_FALLING_EDGE);
    // Set the PAD configuration for the button
        // --> 2mA drive strength, Pull-up or Pull-Down (YOU DETERMINE)
```

```
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD
_WPU);
    // Configure LED as OUPTUT
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0);
    // Enable interrupts for the specified pin
    GPIOPinIntEnable(GPIO_PORTF_BASE, GPIO_PIN_1);
    // Enable interrupt in the Interrupt controller
}

// ****************************************************************************
//
// The interrupt handler for the PF1 pin interrupt.  When triggered, this will
// toggle the LED. PF1 is connected to SELECT Button.
//
// ****************************************************************************
void handle_gpio(void)
{
    //Clear interrupt source
    GPIOPinIntClear(GPIO_PORTF_BASE, GPIO_PIN_1);
    //Toggle LED
    flag ^=1;
    if(flag) {
        //LED ON
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 1);
    } else {
        //LED OFF
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, 0);
    }
}



//****************************************************************************
//
// Main()
//
//****************************************************************************
int
main(void)
{
    // Set the clocking to run directly from the crystal.
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                   SYSCTL_XTAL_8MHZ);
    // Call the GPIO setup function
    GPIO_setup();
    // Enable processor interrupts.
    IntMasterEnable();

    while(1)        // Infinite loop
    {
        // Code to acknowledge interrupt and toggle LED
    }
}
```