ECE Lab 4, Spring 2019
Motion Detection Alarm

- Develop a motion detection alarm
- Exercise GPIO programming
  - GPIO: General-Purpose Input and Output
- Continue C and Assembly programming
  - You will write two non-leaf functions

Buzzer and Motion Sensor

- The buzzer makes a simple sound
- Three pins: V_{dd}, Ground, Signal
  - Signal pin, for output: 0 – Off, 1 – On

- The PIR motion sensor detects human movements by passive IR signal detection
- Three pins: V_{dd}, Ground, Data
  - Data pin, for input: 0 – Motion not detected, 1 – Detected

- The signal/data pins are connected to Tiva C's GPIO pins

Tiva C GPIO Ports

Six ports available: A, B, C, D, E, F

- Ports A, B, C, D have 8 pints each
  - Example: PC5 means Port C, Pin 5
- Port E has 6 pins and Port F has 5 pins

Reading: Tiva C Datasheet, Ch. 10

GPIO pins used in our labs so far:

- Sub-LEDs: PF1 (red), PF2 (blue), and PF3 (green)
- Push button: PF0 (SW2), PF4 (SW1)
- 7-Segment: PA6 (clock), PA7 (data)
  - Connected to Grove base J10

GPIO Programming

- Method 1: Directly access I/O registers of GPIO ports
  - To be studied later

- Method 2: Call TivaWare GPIO functions
  - Enable the GPIO Port Peripheral
  - Configure the pins of the port
  - Read/write the pin

GPIO and TivaWare

- Lab 4 involves two GPIO pins
  - Buzzer: PC5, through Grove Base J17, as output pin
  - PIR motion sensor: PC4, through Grove Base J16, as input pin

- How do we program an input pin?
  - Enable the GPIO port as peripheral
    - TivaWare function: SysCtlPeripheralEnable()
  - Configure the pin as **input pin**
    - TivaWare function: GPIOPinTypeGPIOInput()
  - Read the pin
    - TivaWare function: GPIOPinRead()

GPIO and TivaWare

- How do we program an output pin?
  - Enable the GPIO port as peripheral, if not yet done
    - TivaWare function: SysCtlPeripheralEnable()
  - Configure the pin as **output pin**
    - TivaWare function: GPIOPinTypeGPIOOutput()
  - Write to the pin
    - TivaWare function: GPIOPinWrite()

Example: Configure GPIO Pins as Input

```
void pbInitForced()
{
    /// Enable PF and configure PF0 and PF4 to output
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0 |
      GPIO_PIN_4);

      … /// The rest of code
}
```

The code is from pushbutton.c in the Util library.

Example: Read from GPIO Pin

```
int pbRead()
{
    uint8_t pinValue;

    // SW1 and SW2 are active low, so invert the reading
    pinValue = ~GPIOPinRead(GPIO_PORTF_BASE,
             GPIO_PIN_0 | GPIO_PIN_4);

    … // the rest of code
}
```

Example: Initialize GPIO Pins as Output

Initialization for LaunchPad's on-board LED

- Pin usage: Red – PF1, Blue – PF2, Green – PF3

```
void ledInit()
{
    /// Enable the GPIO port peripheral (Port F)
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    /// Configure the three pins used by LED as output
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3);
}
```

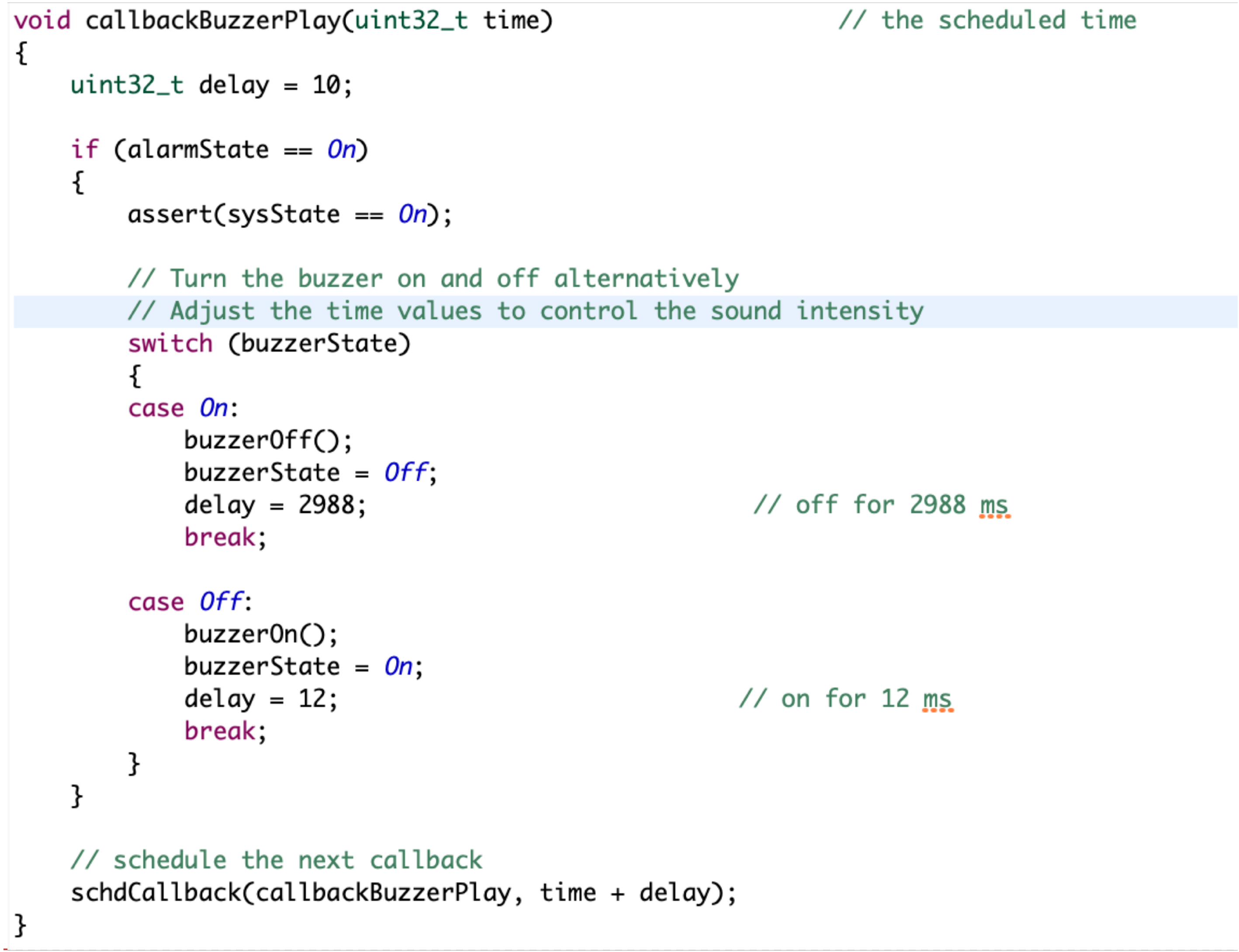
The code is from led.c in the Util library.

Example: Write to GPIO Pins

```
void ledTurnOnOff(bool red, bool green, bool blue)
{
    uint8_t pinValue = 0;
    if (red)
        pinValue |= GPIO_PIN_1;
    if (blue)
        pinValue |= GPIO_PIN_2;
    if (green)
        pinValue |= GPIO_PIN_3;
    GPIOPinWrite(GPIO_PORTF_BASE,
      GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3 /* mask */,
       pinValue);
}
```

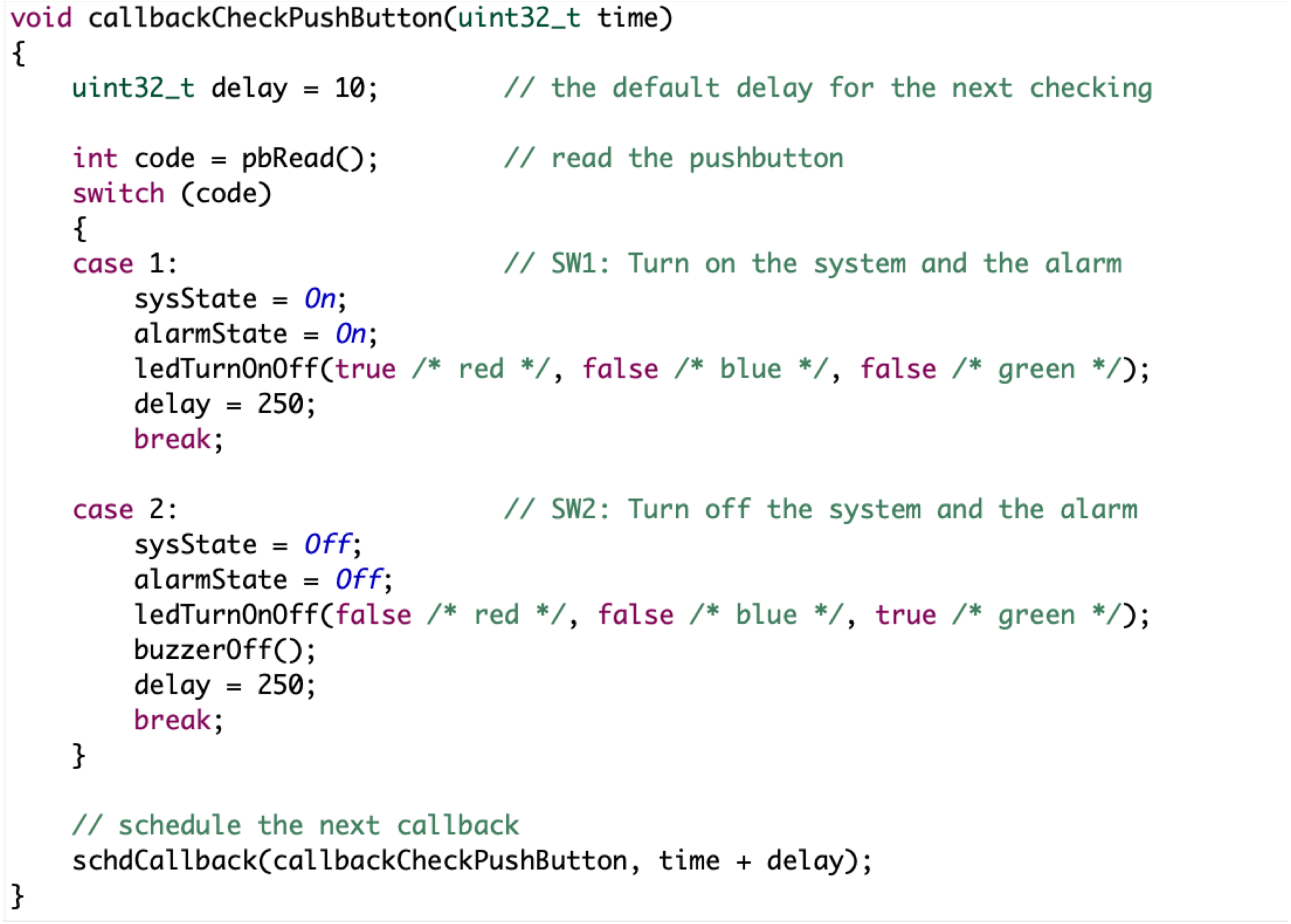
The code is from led.c in the Util library.

Task for Buzzer Playing



```
void callbackBuzzerPlay(uint32_t time)                    // the scheduled time
{
    uint32_t delay = 10;

    if (alarmState == On)
    {
        assert(sysState == On);

        // Turn the buzzer on and off alternatively
        // Adjust the time values to control the sound intensity
        switch (buzzerState)
        {
        case On:
            buzzerOff();
            buzzerState = Off;
            delay = 2988;                                  // off for 2988 ms
            break;

        case Off:
            buzzerOn();
            buzzerState = On;
            delay = 12;                                   // on for 12 ms
            break;
        }
    }

    // schedule the next callback
    schdCallback(callbackBuzzerPlay, time + delay);
}
```

Task for Checking Pushbuttons

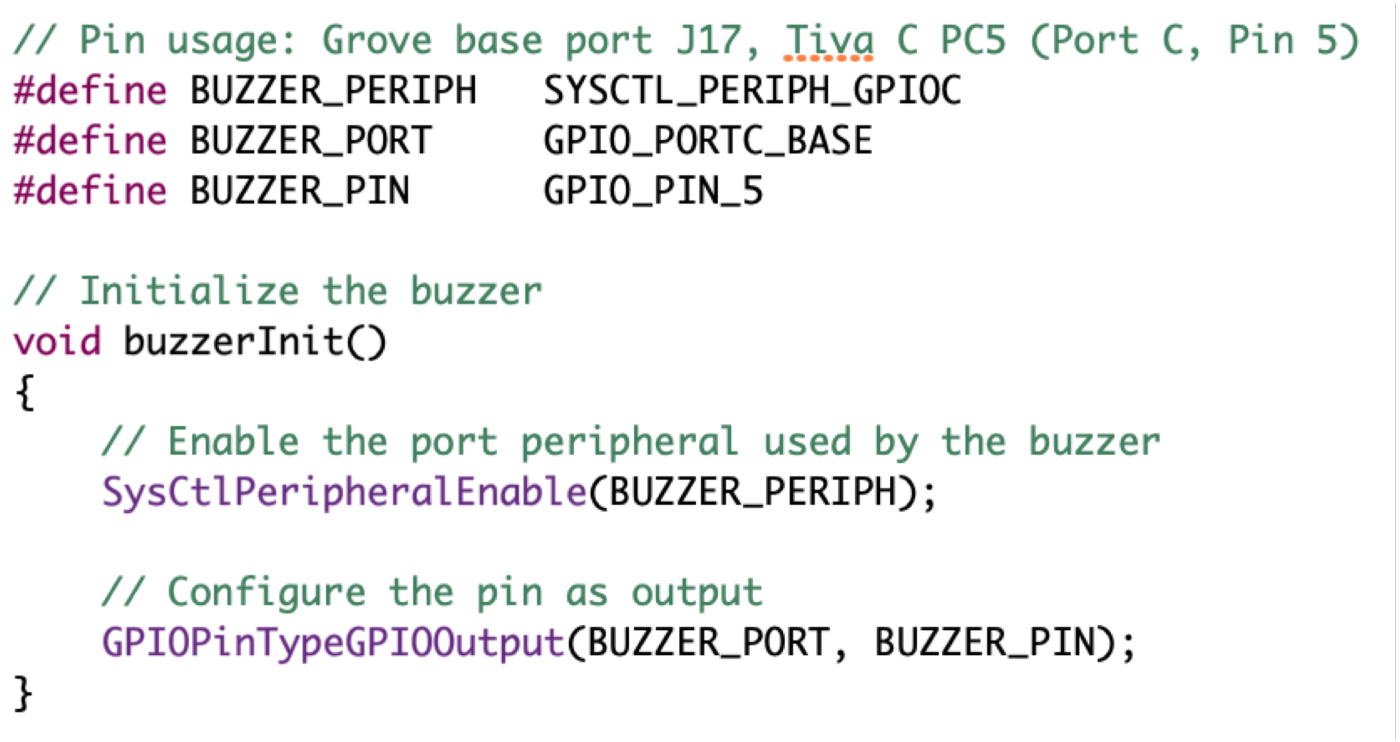


```
void callbackCheckPushButton(uint32_t time)
{
    uint32_t delay = 10;        // the default delay for the next checking

    int code = pbRead();        // read the pushbutton
    switch (code)
    {
    case 1:                     // SW1: Turn on the system and the alarm
        sysState = On;
        alarmState = On;
        ledTurnOnOff(true /* red */, false /* blue */, false /* green */);
        delay = 250;
        break;

    case 2:                     // SW2: Turn off the system and the alarm
        sysState = Off;
        alarmState = Off;
        ledTurnOnOff(false /* red */, false /* blue */, true /* green */);
        buzzerOff();
        delay = 250;
        break;
    }

    // schedule the next callback
    schdCallback(callbackCheckPushButton, time + delay);
}
```

What To Do in Lab 4

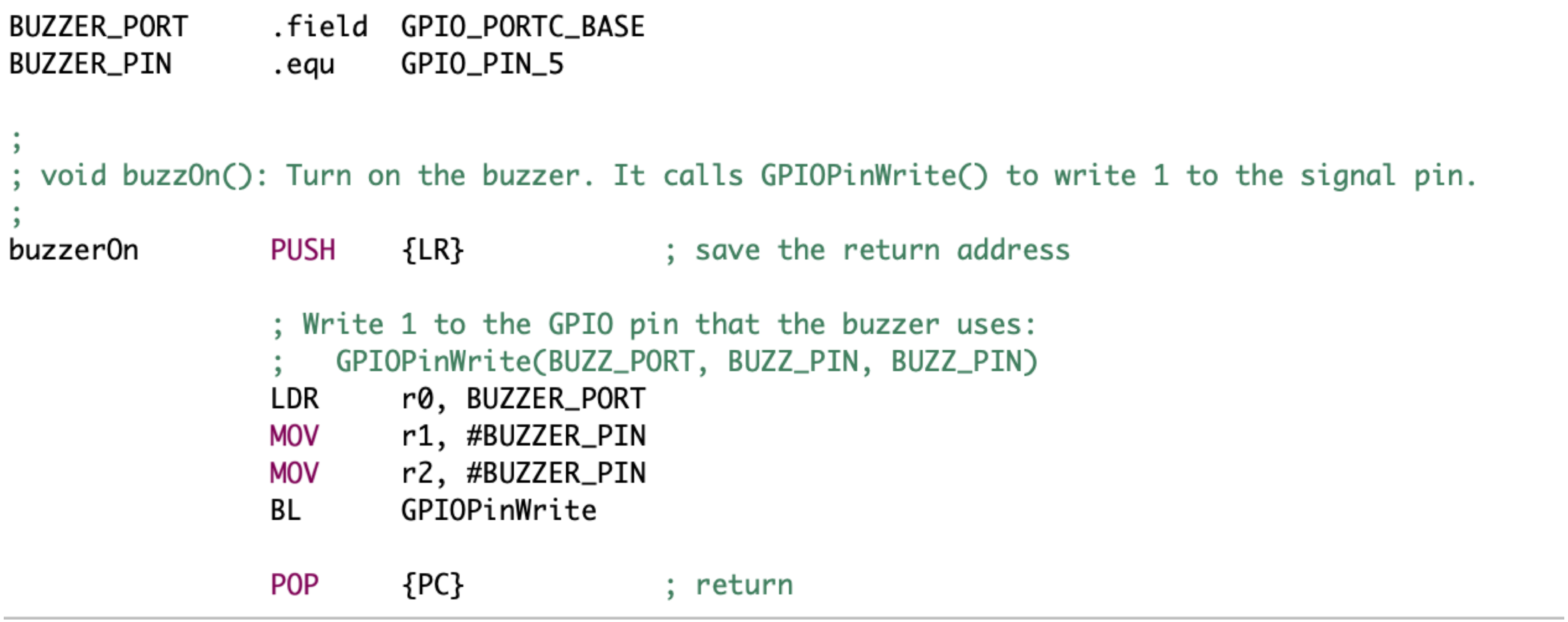
- Create Lab4 project and add the starter program files
- Test and run the starter code, then read through the program files
- Add the following files to the project
  - motion.c: Write a function to initialize for the PIR motion sensor
  - motion_asm.asm: Write a function to read the input from the motion sensor
  - motion.h: Contains the prototypes of the above functions
- Use buzzer.c, buzzer_asm.asm, and buzzer.h as templates

C Function for Initialize the Buzzer (buzzer.c)



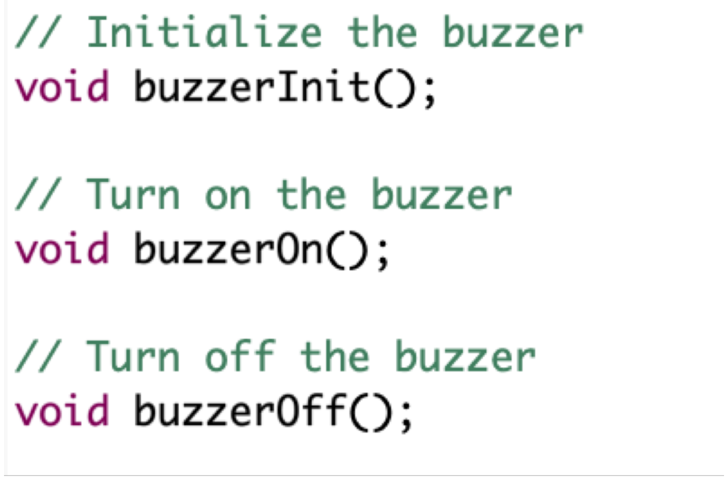
```c
// Pin usage: Grove base port J17, Tiva C PC5 (Port C, Pin 5)
#define BUZZER_PERIPH   SYSCTL_PERIPH_GPIOC
#define BUZZER_PORT     GPIO_PORTC_BASE
#define BUZZER_PIN      GPIO_PIN_5

// Initialize the buzzer
void buzzerInit()
{
    // Enable the port peripheral used by the buzzer
    SysCtlPeripheralEnable(BUZZER_PERIPH);

    // Configure the pin as output
    GPIOPinTypeGPIOOutput(BUZZER_PORT, BUZZER_PIN);
}
```

Assembly Function for Operating the Buzzer (buzzer_asm.asm)



```
BUZZER_PORT      .field  GPIO_PORTC_BASE
BUZZER_PIN       .equ    GPIO_PIN_5

;
; void buzzOn(): Turn on the buzzer. It calls GPIOPinWrite() to write 1 to the signal pin.
;
buzzerOn         PUSH    {LR}            ; save the return address

                 ; Write 1 to the GPIO pin that the buzzer uses:
                 ;   GPIOPinWrite(BUZZ_PORT, BUZZ_PIN, BUZZ_PIN)
                 LDR     r0, BUZZER_PORT
                 MOV     r1, #BUZZER_PIN
                 MOV     r2, #BUZZER_PIN
                 BL      GPIOPinWrite

                 POP     {PC}            ; return
```

Buzzer Function Porotypes (buzzer.h)



```
// Initialize the buzzer
void buzzerInit();

// Turn on the buzzer
void buzzerOn();

// Turn off the buzzer
void buzzerOff();
```

What To Do in Lab 4 (cont.)

- In Part 1, write assembly code to
  1. Enable Port C and configure PC4 as output pin
  2. Read the input from the PIR motion sensor
  3. Add a callback function to print a message on terminal periodically

  Recall PC4 is the signal pin of the PIR motion sensor
- You are suggested to write three functions
  - void pirInit(), in motion.c
  - bool pirDetect(), in motion_asm.asm
  - a callback function for checking the PIR motion sensor, in main.c

What To Do in Lab 4 (cont.)

- In Part 2, revise the C code in main.c such that:
  - Initially, the system is deactivated
  - Push SW1: The system should be activated (sysState = On)
  - Push SW2: The is de-activated
  - **Deactivated:** LED turned off, no buzzer sound
  - **Activated and NO motion**: LED in green color, no buzzer sound
  - **Activated and motion detected**: LED in red color, buzzer beeps periodically
  - ***Optional***: The alarm may be on and off. Fix the problem.