**Semaphores**

Today we are going to go over an example of the use of a semaphore to signal an event between tasks. I will go through the code that debounces the user pushbutton and creates a one-shot to signal the LED blink task each time the button is pressed.

Note that before the scheduler is started, the semaphore needs to be created. To signal an event, the button task “gives” the semaphore and to receive the signal the LED blink task “takes’ the semaphore.

The function xSemaphoreCreateBinary() is described in the AWS FreeRTOS Reference Manual starting on page 212.

The function xSemaphoreGive() is described in the AWS FreeRTOS Reference Manual starting on page 236.

The function xSemaphoreTake() is described in the AWS FreeRTOS Reference Manual starting on page 244. Of particular interest is the second argument, xTicksToWait.

//

// COMP-GENG 422 - Tom Lupfer

//

// Session 9 - Use of a semaphore

//

#include "main.h"

#include <semphr.h>

#define BTN\_DEBOUNCE\_CNT 3

#define BTN\_DELAY 25

#define BTN\_TASK\_PRI 1

#define BTN\_TASK\_STACK 1000

#define BTN\_GPIO\_PRESSED 0

#define BTN\_ST\_RELEASED 0

#define BTN\_ST\_PRESSED 1

#define BTN\_ST\_DEBOUNCE 2

#define LED\_DELAY\_FAST 50

#define LED\_DELAY\_SLOW 500

#define LED\_TASK\_PRI 1

#define LED\_TASK\_STACK 1000

#define SEM\_TIMEOUT\_NONE 0

SemaphoreHandle\_t BtnSemaphore;

void LedBlinkTask(void \*pvParameters);

void ButtonPressTask(void \*pvParameters);

static void SystemClock\_Config(void);

int main(void)

{

// Initialize the hardware and the system clocks

HAL\_Init();

SystemClock\_Config();

// Initialize LED2

BSP\_LED\_Init(*LED2*);

// Initialize the user pushbutton

BSP\_PB\_Init(*BUTTON\_USER*, *BUTTON\_MODE\_GPIO*);

// Create the semaphore to indicate a button press

BtnSemaphore = xSemaphoreCreateBinary();

// Create the button press task

xTaskCreate(ButtonPressTask, // Task function

"ButtonPressTask", // Task name, for debugging

BTN\_TASK\_STACK, // Stack size

NULL, // Task parameter

BTN\_TASK\_PRI, // Task priority

NULL); // Task handle

// Create the LED blink task

xTaskCreate(LedBlinkTask, // Task function

"LedBlinkTask", // Task name, for debugging

LED\_TASK\_STACK, // Stack size

NULL, // Task parameter

LED\_TASK\_PRI, // Task priority

NULL); // Task handle

// Start the scheduler, which never exits

vTaskStartScheduler();

}

// Task to debounce the pushbutton and signal a press

void ButtonPressTask(void \*pvParameters)

{

static int BtnDebounceCnt;

static int BtnState = BTN\_ST\_RELEASED;

while (TRUE)

{

switch (BtnState)

{

case BTN\_ST\_RELEASED:

if (BSP\_PB\_GetState(*BUTTON\_USER*) == BTN\_GPIO\_PRESSED) // pushbutton is pressed

{

BtnDebounceCnt = BTN\_DEBOUNCE\_CNT;

BtnState = BTN\_ST\_DEBOUNCE;

}

break;

case BTN\_ST\_PRESSED:

if (BSP\_PB\_GetState(*BUTTON\_USER*) != BTN\_GPIO\_PRESSED) // pushbutton is released

{

BtnDebounceCnt = BTN\_DEBOUNCE\_CNT;

BtnState = BTN\_ST\_DEBOUNCE;

}

break;

case BTN\_ST\_DEBOUNCE:

if (--BtnDebounceCnt >= 0) // wait debounce period

{

break;

}

if (BSP\_PB\_GetState(*BUTTON\_USER*) == BTN\_GPIO\_PRESSED) // pushbutton is pressed

{

xSemaphoreGive(BtnSemaphore); // signal that the pushbutton has been pressed

BtnState = BTN\_ST\_PRESSED;

}

else // pushbutton is released

{

BtnState = BTN\_ST\_RELEASED;

}

break;

}

vTaskDelay(BTN\_DELAY);

}

}

// Task to blink LED2

void LedBlinkTask(void \*pvParameters)

{

static Bool LedBlinkIsSlow = TRUE;

while (TRUE)

{

BSP\_LED\_Toggle(*LED2*);

if (xSemaphoreTake(BtnSemaphore, SEM\_TIMEOUT\_NONE) == pdTRUE)

{

LedBlinkIsSlow = !LedBlinkIsSlow;

}

vTaskDelay(LedBlinkIsSlow ? LED\_DELAY\_SLOW : LED\_DELAY\_FAST);

}

}

//

// System Clock Configuration

//

// System Clock source = PLL (MSI)

// SYSCLK(Hz) = 80000000

// HCLK(Hz) = 80000000

// AHB Prescaler = 1

// APB1 Prescaler = 1

// APB2 Prescaler = 1

// MSI Frequency(Hz) = 4000000

// PLL\_M = 1

// PLL\_N = 40

// PLL\_R = 2

// PLL\_P = 7

// PLL\_Q = 4

// Flash Latency(WS) = 4

//

static void SystemClock\_Config(void)

{

RCC\_ClkInitTypeDef RCC\_ClkInitStruct;

RCC\_OscInitTypeDef RCC\_OscInitStruct;

// MSI is enabled after System reset, activate PLL with MSI as source

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_MSI;

RCC\_OscInitStruct.MSIState = RCC\_MSI\_ON;

RCC\_OscInitStruct.MSIClockRange = RCC\_MSIRANGE\_6;

RCC\_OscInitStruct.MSICalibrationValue = RCC\_MSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_MSI;

RCC\_OscInitStruct.PLL.PLLM = 1;

RCC\_OscInitStruct.PLL.PLLN = 40;

RCC\_OscInitStruct.PLL.PLLR = 2;

RCC\_OscInitStruct.PLL.PLLP = 7;

RCC\_OscInitStruct.PLL.PLLQ = 4;

if(HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != *HAL\_OK*)

{

while (TRUE) // hang if there is an initialization error

;

}

// Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2 clock dividers

RCC\_ClkInitStruct.ClockType = (RCC\_CLOCKTYPE\_SYSCLK | RCC\_CLOCKTYPE\_HCLK |  
 RCC\_CLOCKTYPE\_PCLK1 | RCC\_CLOCKTYPE\_PCLK2);

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if(HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_4) != *HAL\_OK*)

{

while (TRUE) // hang if there is an initialization error

;

}

}

**Final Project – Initial Discussion**

Here’s a summary of the peripherals available on the ST Micro B-L475E-IOT01A development board:

* STM32L475 MCU – 80MHz Cortex-M4 core, FPU, MPU, 1MB flash, 128KB RAM
* Serial flash – 64Mb (8MB)
* WiFi – 802.11 b/g/n
* Bluetooth Low Energy (BLE) – V4.1
* Sub-GHz RF – 915MHz ISM band
* Near Field Communications (NFC) Tag – ISO/IEC 14443 Type A and NFC Forum Type 4
* Digital microphones (2)
* Temperature and relative humidity sensor
* Barometric pressure sensor
* 3D magnetometer
* 3D accelerometer + 3D gyroscope
* ToF (time-of-flight) sensor
* Green LED
* Pushbutton
* Potential feature via an Arduino-compatible shield – Graphical LCD

We will spend some time discussing possible functionality for the final project and we will take a look at the documentation for the ST Micro VL53L0x, which is located in the lower left of the development board and has the reference designator U4.

**Class Notes**