Lab(FreeRtosBlinky-Toggle Button)

ESE 3025: EmbeddedReal Time Operating Systems

Lambton College in Toronto

Instructor: Takis Zourntos

STUDENT NAME & ID:

Vishal Hasrajani(C0761544)
Parth Patel(C0764929)
Goutham Reddy Alugubelly(C0747981)
Ratnajahnavi rebbapragada(C0762196)

INTRODUCTION :

In this lab, we will integrate GPIO interrupt code with Enhanced RTOS blinky code and then we will adjust the frequency of flashing LEDs using Toggle buttons.Button 1 will increase the frequency of flashing LED and Button 2 will decrease the frequency.

DESCRIPTION :

At first, we checked which Ports and pins are eligible for GPIO interrupts. We found that only port 0 and port 2 are suitable with GPIO interrupts. Among these two ports we selected port 2 . We selected pin 6 (port 2) for Button 1 and pin 7 for Button 2.

```
#define GPIO_INTERRUPT_PIN_A 6 // PIN_A selected for Button 1
#define GPIO_INTERRUPT_PORT  GPIOINT_PORT2 /* GPIO port number
mapped to interrupt */
#define GPIO_INTERRUPT_PIN_B 7 // PIN_A selected for Button 2
```

Now , On the LPC1769, the GPIO interrupt shares the EINT3 vector.

```
#define GPIO_IRQ_HANDLER          EINT3_IRQHandler   /* GPIO
interrupt IRQ function name */
#define GPIO_INTERRUPT_NVIC_NAME      EINT3_IRQn    /* GPIO
interrupt NVIC interrupt name */
```

Now to add an GPIO interrupt, we created the function called GPIO_IRQ_HANDLER(void).

Initially here,we set the count=0 and we are incrementing or decrementing the count depending on which button is pressed.

```
void GPIO_IRQ_HANDLER(void) {
     /* disable interrupt, clear status */
     NVIC_DisableIRQ(GPIO_INTERRUPT_NVIC_NAME);  //
prevents interrupt nesting
     Chip_GPIOINT_ClearIntStatus(LPC_GPIOINT,
GPIO_INTERRUPT_PORT,
              (1 << GPIO_INTERRUPT_PIN_A)|(1 <<
GPIO_INTERRUPT_PIN_B)); //clear the status of pin A and pin
B of the port

     bool Button_State_1 = Chip_GPIO_GetPinState(LPC_GPIO,
GPIO_INTERRUPT_PORT,
              GPIO_INTERRUPT_PIN_A); // Check the state of
the Pin A

     bool Button_State_2 = Chip_GPIO_GetPinState(LPC_GPIO,
GPIO_INTERRUPT_PORT,
                   GPIO_INTERRUPT_PIN_B); //Check the state
of Pin_B

     //Button_State_1 == false means if Button 1 is
pressed.
     if (Button_State_1 == false) {

          if(count>=0 && count <=800)  // Limiting the
frequency to 200
          {
          count = count + 50; //increment the count by 50
          }

     }

     else if(Button_State_2 == false) //if Button 2 is
pressed
     {
          if(count>=0)
          {
          count = count - 50;  // decrementing the count by
50
          }
     }

     /* re-enable interrupt */
     NVIC_EnableIRQ(GPIO_INTERRUPT_NVIC_NAME);
}
```

So here we are disabling the interrupt and clearing the status of GPIO port. After that we are checking the state of the Pin A and Pin B . If Button 1 is pressed,then increase the frequency of the Flashing LED else if Button 2 is pressed, decrease the frequency.

Further, we set up the system hardware by using the function `static void prvSetupHardware(void)`

```
/* Sets up system hardware */
static void prvSetupHardware(void)
{
     SystemCoreClockUpdate();
     Board_Init();

     /* Initial red LED,green LED and blue LED states are
off */
     Board_LED_Set(red, false);
     Board_LED_Set(green, false);
     Board_LED_Set(blue, false);
}
```

Here this function **prvSetupHardware(void)** sets up the system hardware initially.
Then **SystemCoreUpdate()** is used to update the cpu core clock rate.

Board_Init() is used to set up and initialize all required blocks and functions related to board hardware.

Then we set LEDS(that are red,green and blue) to OFF state initially.

LED1(task-1) ,LED2(task-2) and LED3(task-3) threads.

LED1 TOGGLE THREAD

```
/* LED1 toggle thread */
static void vLEDTask1(void *pvParameters) {
      xLastWakeTime1 = xTaskGetTickCount (); //calculate
last wake time

     while (1) {
          Board_LED_Set(red, true); //red led ON
          vTaskDelayUntil( &xLastWakeTime1, 1000 -
count);//block for 1 sec
          Board_LED_Set(red, false);//red led OFF
          vTaskDelayUntil( &xLastWakeTime1, (5000 -
(5*count)));//block for 5 seconds.


     }
}
```

Here we have used vTaskDelayUntil as it is mostly preferred for the periodic tasks.It starts counting the time from when the task first entered the running state.

Then we added (1000-count)so that as the count changes the frequency also changes.
Similarly for every other case.

LED 2 TOGGLE THREAD

```
/* LED2 toggle thread */
static void vLEDTask2(void *pvParameters) {

     xLastWakeTime2 = xTaskGetTickCount (); //calculate
last wake time

// Count changes the frequency of Leds.
//initially the count is zero
     vTaskDelayUntil( &xLastWakeTime2, (2000 -
(2*count)));//Block for 2 seconds
     while (1) {

          Board_LED_Set(green, true); //green led ON
           vTaskDelayUntil( &xLastWakeTime2, 1000 -
count);//Block for 1 sec
          Board_LED_Set(green, false); //green led OFF
           vTaskDelayUntil( &xLastWakeTime2, (5000 -
(5*count)));//block for 5 sec



     }
}
```

LED 3 TOGGLE THREAD

```
static void vLEDTask3(void *pvParameters) {
     xLastWakeTime3 = xTaskGetTickCount (); //calculate
last wake time

     vTaskDelayUntil( &xLastWakeTime3, (4000 -
(4*count)));//Block for 4sec
```

```
    while (1) {

        Board_LED_Set(blue,true); //blue led ON
         vTaskDelayUntil( &xLastWakeTime3, 1000 -
count); //Block for 1 sec
        Board_LED_Set(blue,false);//blue led OFF
         vTaskDelayUntil( &xLastWakeTime3, (5000 -
(5*count)));//Block for 5 sec
    }
}
```

Now moving on to the main function ,
Here we are setting up the hardware and then we created 3 tasks in the real time with the same priorities.
To activate the interrupt, here we chose pin A(i.e pin 6) and pin B(i.e pin 7) of port 2 as an input port
Then , GPIO pins are configured in a way that falling edge will activate the state of pins.
Finally we enabled the Interrupt and started the scheduler.

```
int main(void)
{
    prvSetupHardware();

    /* LED1 toggle thread */
    xTaskCreate(vLEDTask1, (signed char *) "vTaskLed1",
                configMINIMAL_STACK_SIZE, NULL,
(tskIDLE_PRIORITY + 1UL),
                (xTaskHandle *) NULL);
```

```
     /* LED2 toggle thread */
     xTaskCreate(vLEDTask2, (signed char *) "vTaskLed2",
                    configMINIMAL_STACK_SIZE, NULL,
(tskIDLE_PRIORITY + 1UL),
                    (xTaskHandle *) NULL);
     /* LED3 toggle thread */
     xTaskCreate(vLEDTask3, (signed char *) "vTaskLed3",
                         configMINIMAL_STACK_SIZE, NULL,
(tskIDLE_PRIORITY + 1UL),
                         (xTaskHandle *) NULL);


     Chip_GPIO_SetPinDIRInput(LPC_GPIO,
GPIO_INTERRUPT_PORT,
               GPIO_INTERRUPT_PIN_A); //Set pin A of port as
input
     Chip_GPIO_SetPinDIRInput(LPC_GPIO,
GPIO_INTERRUPT_PORT,
                    GPIO_INTERRUPT_PIN_B); //Set pin B of
port as input

     /* Configure the GPIO interrupt */
     Chip_GPIOINT_SetIntFalling(LPC_GPIOINT,
GPIO_INTERRUPT_PORT,
               (1 << GPIO_INTERRUPT_PIN_A)|(1 <<
GPIO_INTERRUPT_PIN_B)); //Set the Gpio pin for Falling Edge
for Pin A or Pin B

     /* Enable interrupt in the NVIC */
     NVIC_ClearPendingIRQ(GPIO_INTERRUPT_NVIC_NAME);
     NVIC_EnableIRQ(GPIO_INTERRUPT_NVIC_NAME);
```

```
    /* Start the scheduler */
    vTaskStartScheduler();
    /* Should never arrive here */
    return 1;
}
```

OUTPUT :

Youtube Link :

https://www.youtube.com/watch?v=vbiXyAHV7Zc

CONCLUSION :

Overall , we learned how to change the frequency of the LEDs in the FreeRtosBlinky code using the Toggle Buttons .We also learned the working of vTaskDelayUntil() function and we used it in each threads to adjust the frequency at a particular time. To the end ,we executed it successfully and the experience we got here was amazing.

APPENDIX:

Frequency-Increase-Toggle-FreeBlinky.c

```
/*
 * @brief FreeRTOS Blinky example
 *
 * @note
 * Copyright(C) NXP Semiconductors, 2014
 * All rights reserved.
 *
 * @par
 * Software that is described herein is for illustrative
purposes only
 * which provides customers with programming information
regarding the
 * LPC products.  This software is supplied "AS IS" without any
warranties of
 * any kind, and NXP Semiconductors and its licensor disclaim
any and
 * all warranties, express or implied, including all implied
warranties of
 * merchantability, fitness for a particular purpose and
non-infringement of
 * intellectual property rights.  NXP Semiconductors assumes no
responsibility
 * or liability for the use of the software, conveys no license
or rights under any
 * patent, copyright, mask work right, or any other intellectual
property rights in
 * or to any products. NXP Semiconductors reserves the right to
make changes
 * in the software without notification. NXP Semiconductors also
makes no
 * representation or warranty that such application will be
```

```
suitable for the
 * specified use without further testing or modification.
 *
 * @par
 * Permission to use, copy, modify, and distribute this software
and its
 * documentation is hereby granted, under NXP Semiconductors'
and its
 * licensor's relevant copyrights in the software, without fee,
provided that it
 * is used in conjunction with NXP Semiconductors
microcontrollers.  This
 * copyright, permission, and disclaimer notice must appear in
all copies of
 * this code.
 */

#include "board.h"
#include "FreeRTOS.h"
#include "task.h"

#define GPIO_INTERRUPT_PIN_A   6
#define GPIO_INTERRUPT_PORT    GPIOINT_PORT2
#define GPIO_INTERRUPT_PIN_B   7


/* On the LPC1769, the GPIO interrupts share the EINT3 vector.
*/
#define GPIO_IRQ_HANDLER           EINT3_IRQHandler/* GPIO
interrupt IRQ function name */
#define GPIO_INTERRUPT_NVIC_NAME        EINT3_IRQn/* GPIO
interrupt NVIC interrupt name */

int count = 0;
portTickType xLastWakeTime1;
portTickType xLastWakeTime2;
portTickType xLastWakeTime3;
portTickType xLastWakeTime4;
portTickType xLastWakeTime5;
portTickType xLastWakeTime6;

void GPIO_IRQ_HANDLER(void) {
     /* disable interrupt, clear status */
     NVIC_DisableIRQ(GPIO_INTERRUPT_NVIC_NAME);  // prevents
interrupt nesting
     Chip_GPIOINT_ClearIntStatus(LPC_GPIOINT,
GPIO_INTERRUPT_PORT,
               (1 << GPIO_INTERRUPT_PIN_A)|(1 <<
GPIO_INTERRUPT_PIN_B)); //clear the status of pin A and pin B of
the port

     bool Button_State_1 = Chip_GPIO_GetPinState(LPC_GPIO,
GPIO_INTERRUPT_PORT,
               GPIO_INTERRUPT_PIN_A); // Check the state of the
Pin A

     bool Button_State_2 = Chip_GPIO_GetPinState(LPC_GPIO,
GPIO_INTERRUPT_PORT,
                    GPIO_INTERRUPT_PIN_B); //Check the state of
Pin_B

     //As we are not using the pull up resistor,so
Button_State_1 == false means if Button 1 is pressed.
     if (Button_State_1 == false) {

          if(count>=0 && count <=800)  // Limiting the
frequency to 200
          {
          count = count + 50; //increment the count by 50
          }

     }
     else if(Button_State_2 == false) //if Button 2 is pressed
     {
          if(count>=0)
          {
          count = count - 50;  // decrementing the count by 50
          }
     }

     /* re-enable interrupt */
     NVIC_EnableIRQ(GPIO_INTERRUPT_NVIC_NAME);
}

/***************************************************************
**************
 * Private types/enumerations/variables

****************************************************************
************/

/***************************************************************
**************
 * Public types/enumerations/variables

****************************************************************
************/

/***************************************************************
**************
 * Private functions

****************************************************************
************/

/* Sets up system hardware */
static void prvSetupHardware(void)
```

```
{
     SystemCoreClockUpdate();
     Board_Init();

     /* Initial red LED,green LED and blue LED states are off
*/
     Board_LED_Set(red, false);
     Board_LED_Set(green, false);
     Board_LED_Set(blue, false);
}


/* LED1 toggle thread */
static void vLEDTask1(void *pvParameters) {
      xLastWakeTime1 = xTaskGetTickCount ();

     while (1) {
          Board_LED_Set(red, true); //red led ON
          vTaskDelayUntil( &xLastWakeTime1, 1000 -
count);//1sec delay
          Board_LED_Set(red, false);//red led OFF
          vTaskDelayUntil( &xLastWakeTime1, (5000 -
(5*count));//5sec delay


     }
}

/* LED2 toggle thread */
static void vLEDTask2(void *pvParameters) {

     xLastWakeTime2 = xTaskGetTickCount ();
     vTaskDelayUntil( &xLastWakeTime2, (2000 -
(2*count)));//2sec delay
     while (1) {
          Board_LED_Set(green, true); //green led ON
           vTaskDelayUntil( &xLastWakeTime2, 1000 - 
count);//1sec delay
          Board_LED_Set(green, false); //green led OFF
           vTaskDelayUntil( &xLastWakeTime2, (5000 - 
(5*count)));//5sec delay


     }
}

static void vLEDTask3(void *pvParameters) {
     xLastWakeTime3 = xTaskGetTickCount ();
     vTaskDelayUntil( &xLastWakeTime3, (4000 - 
(4*count)));//4sec delay
     while (1) {

          Board_LED_Set(blue,true); //blue led ON
           vTaskDelayUntil( &xLastWakeTime3, 1000 - count);
//1sec delay
          Board_LED_Set(blue,false);//blue led OFF
           vTaskDelayUntil( &xLastWakeTime3, (5000 - 
(5*count)));//5sec delay



     }
}





/* UART (or output) thread */
```

```
/*****************************************************************
**************
 * Public functions

*****************************************************************
************/

/**
 * @brief main routine for FreeRTOS blinky example
 * @returnNothing, function should not exit
 */
int main(void)
{
     prvSetupHardware();

     /* LED1 toggle thread */
     xTaskCreate(vLEDTask1, (signed char *) "vTaskLed1",
                    configMINIMAL_STACK_SIZE, NULL,
(tskIDLE_PRIORITY + 1UL),
                    (xTaskHandle *) NULL);

     /* LED2 toggle thread */
     xTaskCreate(vLEDTask2, (signed char *) "vTaskLed2",
                    configMINIMAL_STACK_SIZE, NULL,
(tskIDLE_PRIORITY + 1UL),
                    (xTaskHandle *) NULL);
     /* LED3 toggle thread */
     xTaskCreate(vLEDTask3, (signed char *) "vTaskLed3",
                         configMINIMAL_STACK_SIZE, NULL,
(tskIDLE_PRIORITY + 1UL),
                         (xTaskHandle *) NULL);
     Chip_GPIO_SetPinDIRInput(LPC_GPIO, GPIO_INTERRUPT_PORT,
               GPIO_INTERRUPT_PIN_A);
     Chip_GPIO_SetPinDIRInput(LPC_GPIO, GPIO_INTERRUPT_PORT,
                    GPIO_INTERRUPT_PIN_B);



     /* Configure the GPIO interrupt */
     Chip_GPIOINT_SetIntFalling(LPC_GPIOINT,
GPIO_INTERRUPT_PORT,
               (1 << GPIO_INTERRUPT_PIN_A)|(1 <<
GPIO_INTERRUPT_PIN_B));



     /* Enable interrupt in the NVIC */
     NVIC_ClearPendingIRQ(GPIO_INTERRUPT_NVIC_NAME);
     NVIC_EnableIRQ(GPIO_INTERRUPT_NVIC_NAME);



     /* Start the scheduler */
     vTaskStartScheduler();

     /* Should never arrive here */
     return 1;
}

/**
 * @}
 */
```