Freescale Semiconductor

Lab 5 – Using Events and Mutex

By: Technical Information Center

freescale semiconductor

1 Objective

Create an application that toggles LED1 during 1 second each time SW1 is pressed and toggles LED2 during 1 second each time SW2 is pressed. Each time a SW is pressed a message to a terminal must be sent indicating which SW was pressed and how many times has been pressed.

2 Requirements

- For this Lab CW10.6 and MQX4.1 must be installed in your PC.
- Knowledge about creating a new MQX project or accomplish Lab 2 – Creating and debugging a new MQX project.
- Knowledge about creating and handling tasks or accomplish Lab 3 – Using Tasks
- Knowledge about using LW GPIOs or accomplish Lab 4 – Using LW GPIOs.

3 Implementation

1) Initialize LED1, LED2, SW1 and SW2 on TWR system and create an application that toggles LED1 each time SW1 is pressed, then wait 1 sec using function `_time_delay(1000)`. Task_A must be defined and created for this purpose. Refer to **Lab 4 – Using LW GPIOs**. Debug your application when ready.

2) Define Task_B using the parameters below. Don't forget to declare the prototype and define the task number in main.h.
   a. Task number 3
   b. Task name Task_B
   c. Stack size 1500
   d. Priority must be lower than Main_Task and higher than Task_A
   e. NOT Auto Start.

3) Modify the application according to the following requirements.
   a. Only Task_A will read the button status.
   b. Only Task_B will toggle the LED.

4) To do this you will need to use events. See **Freescale MQX™ RTOS Reference Manual - MQXRM.pdf** for details about events functions and their parameters. You can find it in *<MQX_install_dir>/doc/mqx*.
   a. Create a LWEVENT_STRUCT structure. It can be defined as a global variable.

      Example:

      ```
      LWEVENT_STRUCT lwevent;
      ```

b. Use `_lwevent_create()` in Main_Task to create an event. A message to a terminal must be sent whether the event was created successfully or not and show the error number in case it fails.
c. Use `lwevent_set()` in Task_A to indicate that an event occurred when the button is Pressed.
    i. A message to a terminal must be sent whether the event failed to be set showing the error number.
d. Use `_lwevent_wait_ticks()` in Task_B to wait for the event to occur. After this the LED must toggle.
e. Use `_lwevent_clear()` in Task_B after the code that toggles the LED.

5) In Main_Task use `_task_create()` to create an instance of Task_B. You can refer to **Lab 3 – Using Tasks** or you can see **Freescale MQX™ RTOS Reference Manual - MQXRM.pdf pdf** for details about tasks functions and their parameters. You can find it in *<MQX_install_dir>/doc/mqx*.
    a. The task id number must be saved in a variable of type _task_id.
    b. A message to a terminal must be sent whether the task was created successfully or not and show the error number in case it fails.
    c. After creating Task_B, Main_task must remain blocked using `_task_block()` function.

6) Compile and debug your project. You must be able to toggle LED1 when SW1 is pressed.

7) A local variable must be increased in Task_A each time SW1 is pressed, and a message in terminal must be printed indicating how many times it has been pressed.

8) Compile and debug the project. Look at the output in the terminal. What happens? Why?

9) While the LED is toggling function `_time_delay()` blocks Task_B. During this time Task_A is executed several times. Use a Mutex to avoid that the local variable keeps increasing during the time delay. See **Freescale MQX™ RTOS Reference Manual - MQXRM.pdf** for details about Mutex functions and their parameters. You can find it in *<MQX_install_dir>/doc/mqx*.
    a. Include header `#include <mutex.h>`
    b. Declare an structure of type MUTEX_STRUCT, e.g. `MUTEX_STRUCT my_mutex;`
    c. Initialize mutex in Main_Task using `_mutex_init().`
    d. Use `_mutex_lock()` and `_mutex_unlock()` where needed.

10) Define Task_C which must toggle LED2 when pressing SW2. Use the parameters below for this task. Don't forget to declare the prototype and define the task number in main.h.
    a. Task number 4
    b. Task name Task_C
    c. Stack size 1500
    d. Priority must be lower than Main_Task and higher than Task_A.

11) In Main_Task use `_task_create()` to create an instance of Task_C. You can refer to **Lab 3 – Using Tasks** or you can see **Freescale MQX™ RTOS Reference Manual - MQXRM.pdf pdf** for details about tasks functions and their parameters. You can find it in *<MQX_install_dir>/doc/mqx*.
    a. The task id number must be saved in a variable of type _task_id.
    b. A message to a terminal must be sent whether the task was created successfully or not and show the error number in case it fails.
    c. After creating Task_C, Main_Task must remain blocked using `_task_block()` function.

12) Each time a button is pressed, Task_A must print in terminal a message indicating which button was pressed and how many times it has been pressed.

This now completes Lab 5.

4 Code

main.c

```
#include "main.h"
#include <mutex.h>


#if !BSPCFG_ENABLE_IO_SUBSYSTEM
#error This application requires BSPCFG_ENABLE_IO_SUBSYSTEM defined non-zero in user_config.h. Please recompile BSP with this option.
#endif


#ifndef BSP_DEFAULT_IO_CHANNEL_DEFINED
#error This application requires BSP_DEFAULT_IO_CHANNEL to be not NULL. Please set corresponding BSPCFG_ENABLE_TTYx to non-zero in user_config.h and recompile BSP with this option.
#endif


TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
/*  Task number, Entry point, Stack, Pri, String, Auto? */
  {MAIN_TASK,   Main_task,   1500,  9,   "main", MQX_AUTO_START_TASK},
  {TASK_A,   Task_A,       1500,  11,   "task_a", 0},
  {TASK_B,   Task_B,       1500,  10,   "task_b", 0}, //Priority must be lower than Main_Task and higher than Task_A
  {TASK_C,   Task_C,       1500,  10,   "task_b", 0},
  {0,         0,         0,     0,   0,     0,           }
};

uint32_t result;
_task_id t1,t2,t3;
LWGPIO_STRUCT led1, led2, btn1, btn2;
LWEVENT_STRUCT lwevent;
MUTEX_STRUCT my_mutex;

/*TASK*-----------------------------------------------
*
* Task Name    : Main_task
* Comments     :
*    This task prints " Hello World "
*
*END*-----------------------------------------------*/

//LAB5

void Main_task(uint32_t initial_data)
{
   printf("\n Start Main Task: Events and Mutex\n");

   /*Init LED*/
```

```
   lwgpio_init(&led1, BSP_LED1, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_NOCHANGE);
   lwgpio_init(&led2, BSP_LED2, LWGPIO_DIR_OUTPUT, LWGPIO_VALUE_NOCHANGE);

   /*Init button*/
   lwgpio_init(&btn1, BSP_SW1, LWGPIO_DIR_INPUT, LWGPIO_VALUE_NOCHANGE);
   lwgpio_init(&btn2, BSP_SW2, LWGPIO_DIR_INPUT, LWGPIO_VALUE_NOCHANGE);

   /*Set GPIO functionality*/
   lwgpio_set_functionality(&led1, BSP_LED1_MUX_GPIO);
   lwgpio_set_functionality(&led2, BSP_LED2_MUX_GPIO);
   lwgpio_set_functionality(&btn1, BSP_SW1_MUX_GPIO);
   lwgpio_set_functionality(&btn2, BSP_SW2_MUX_GPIO);

   lwgpio_set_attribute(&btn1, LWGPIO_ATTR_PULL_UP, LWGPIO_AVAL_ENABLE);
   lwgpio_set_attribute(&btn2, LWGPIO_ATTR_PULL_UP, LWGPIO_AVAL_ENABLE);

   lwgpio_set_value(&led1,LWGPIO_VALUE_HIGH); //Turn off led1
   lwgpio_set_value(&led2,LWGPIO_VALUE_HIGH); //Turn off led2

   t1 = _task_create(0,TASK_A,0);//in this moment, Task A was added to the queue

   if(t1 == MQX_NULL_TASK_ID){
      printf("\nCould not create Task A. \n");
   }else{
      printf("\nTask A was created. \n");
   }

   t2 = _task_create(0, TASK_B, 0);//in this moment, Task B was added to the queue

   if (t2 == MQX_NULL_TASK_ID) {
      printf("\nCould not create Task B. \n");
   } else {
      printf("\nTask B was created. \n");
   }

   t3 = _task_create(0, TASK_C, 0);//in this moment, Task C was added to the queue

   if (t3 == MQX_NULL_TASK_ID) {
      printf("\nCould not create Task C. \n");
   } else {
      printf("\nTask C was created. \n");
   }
   if (_lwevent_create(&lwevent,0) != MQX_OK) { //Creation of the event
         printf("\nMake event failed");
         _task_block();
      }else{
         printf("\nEvent was created successfully");
      }

   _mutex_init(&my_mutex, NULL);
   _task_block();//block this Task and continue with the Task list


 _mqx_exit(0);
}
```

```
void Task_A(uint32_t initial_data){

   uint32_t count1 = 0;
   uint32_t count2 = 0;

   while(1){
      _mutex_lock(&my_mutex);
      if(lwgpio_get_value(&btn1) == LWGPIO_VALUE_LOW){ //LOW VALUE = PRESS BUTTON
            result = _lwevent_set(&lwevent,0x01); //Activation of the event
            if(result != MQX_OK){
                  printf("\nSetting event failed. Error: 0x%X", result);
            }else{
                  count1++;
                  printf("\nButton 1 was pressed %d times.", count1);
            }
      }
      if(lwgpio_get_value(&btn2) == LWGPIO_VALUE_LOW){ //LOW VALUE = PRESS BUTTON
            result = _lwevent_set(&lwevent,0x01); //Activation of the event
            if(result != MQX_OK){
                  printf("\nSetting event failed. Error: 0x%X", result);
            }else{
                  count2++;
                  printf("\nButton 2 was pressed %d times.", count2);
                  }
            }

            _mutex_unlock(&my_mutex);
      }

}

void Task_B(uint32_t initial_data)
{
      while(1)
      {

            result = _lwevent_wait_ticks(&lwevent,0x01,FALSE,0); //wait for event
            if(result != MQX_OK)
            {
                  printf("\nWaiting event failed. Error: 0x%X", result);
            }
            _mutex_lock(&my_mutex);
            lwgpio_toggle_value(&led1);
            _time_delay(1000);
            _mutex_unlock(&my_mutex);
            _lwevent_clear(&lwevent,0x01);// Clear the event flag
      }

}

void Task_C(uint32_t initial_data){
      while(1){

            result = _lwevent_wait_ticks(&lwevent,0x02,FALSE,0);
            if(result != MQX_OK){
                  printf("\nWaiting event failed. Error: 0x%X", result);
```

```
            }
            _mutex_lock(&my_mutex);
            lwgpio_toggle_value(&led2);
            _time_delay(1000);
            _mutex_unlock(&my_mutex);
            _lwevent_clear(&lwevent,0x02);
        }
}


/* EOF */
```

Main.h

```
#ifndef __main_h_
#define __main_h_
#include <mqx.h>
#include <bsp.h>

#define MAIN_TASK 1
#define TASK_A 2
#define TASK_B 3
#define TASK_C 4

extern void Main_task(uint32_t);
extern void Task_A(uint32_t);
extern void Task_B(uint32_t);
extern void Task_C(uint32_t);

/* PPP device must be set manually and
** must be different from the default IO channel (BSP_DEFAULT_IO_CHANNEL)
*/
#define PPP_DEVICE      "ittyb:"

/*
** Define PPP_DEVICE_DUN only when using PPP to communicate
** to Win9x Dial-Up Networking over a null-modem
** This is ignored if PPP_DEVICE is not #define'd
*/
#define PPP_DEVICE_DUN  1

/*
** Define the local and remote IP addresses for the PPP link
** These are ignored if PPP_DEVICE is not #define'd
*/
#define PPP_LOCADDR     IPADDR(192,168,0,216)
#define PPP_PEERADDR    IPADDR(192,168,0,217)

/*
** Define a default gateway
*/
#define GATE_ADDR       IPADDR(192,168,0,1)

#endif /* __main_h_ */
```