**Introduction**

In this hands-on lab, we will use Kinetis Design Studio to 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.

**Resources**

PC running Windows 7 with the following software:

• Kinetis Design Studio (KDS) v2.0.0

• MQX for KSDK v1.1.0

Hardware:

• FRDM-K64F

1. **Implementation**
   1. By now a ‘*New MQX RTOS for KSDK Project Wizard’* does not exists. For this Lab use **‘MQX for KSDK Lab 3 - Base Project’**.

**Note:** If you want to know how to create a base project see the **‘Appendix’** in this document.

* 1. Edit Main\_Task in order to initialize GPIOs. See **‘Kinetis SDK v.1.1 API Reference Manual’** located in ***<KSDK\_install\_dir>/doc*** for details about GPIO functions and their parameters.
     1. Use **GPIO\_DRV\_Init**(switchPins, ledPins) function to initialize onboard LEDs and Buttons.

This function is part of the GPIO driver, it receives the 2 GPIO arrays of structures to initialize the GPIO driver with these parameters.

/\*Init LED and buttons\*/

GPIO\_DRV\_Init(switchPins, ledPins);

* 1. Define Task\_A, it must enter in an endless loop and toggle LED1 during 1 second when SW1 is pressed. Use the parameters below for this task. Don’t forget to declare the prototype and define the task number.
     1. Task number 2
     2. Task name Task\_A
     3. Stack size 1500
     4. Priority must be lower than Main\_Task
     5. NOT Auto Start.

{TASK\_A, Task\_A, 1500, 11, "Task A", 0,0,0},

Add a definition for the task id.

**#define** TASK\_A 2

* 1. Main task should look like this:

**void** **Main\_task**(uint32\_t initial\_data) {

**printf**("\n Start Main Task: Events and Mutex\n");

/\*Init LED and buttons\*/

GPIO\_DRV\_Init(switchPins, ledPins);

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");

}

\_mqx\_exit(0);

}

* 1. To wait 1 sec use the function **\_time\_delay**(1000)
  2. To read SW1 value you can use the following instruction.

**if**(GPIO\_DRV\_ReadPinInput(*kGpioSW1*) == 0)

* 1. To toggle the LED value use **GPIO\_DRV\_TogglePinOutput**(BOARD\_GPIO\_LED\_BLUE). See ‘**Kinetis SDK v.1.1 API Reference Manual’** located in ***<KSDK\_install\_dir>/doc*** for details about GPIO functions and their parameters. Debug your application when ready.
  2. Task A should look like this:  
       
     **void** **Task\_A**(uint32\_t initial\_data) {

**while** (1) {

**if** (GPIO\_DRV\_ReadPinInput(*kGpioSW1*) == 0) { GPIO\_DRV\_TogglePinOutput(BOARD\_GPIO\_LED\_BLUE);

\_time\_delay(1000);

}

}

}

* 1. Define Task\_B using the parameters below. Don’t forget to declare the prototype and define the task number.
     1. Task number 3
     2. Task name Task\_B
     3. Stack size 1500
     4. Priority must be lower than Main\_Task and higher than Task\_A
     5. NOT Auto Start.

{TASK\_B, Task\_B, 1500, 10, "Task B", 0, 0, 0},

Add a definition for the task id.

**#define** TASK\_B 3

* 1. Modify the application according to the following requirements.
     1. Only Task\_A will read the button status.
     2. Only Task\_B will toggle the LED.
  2. To do this you will need to use events. See **Freescale MQX™ RTOS Reference Manual.pdf** for details about events functions and their parameters. You can find it in ***<KSDK\_install\_dir>/rtos/mqx/doc*.**
     1. Create a LWEVENT\_STRUCT structure. It can be defined as a global variable.

Example:

LWEVENT\_STRUCT lwevent;

* + 1. Use **\_lwevent\_create()** in Main\_Taskto 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.

**if** (**\_lwevent\_create**(&lwevent, 0) != MQX\_OK) { //Creation of the event

**printf**("\nMake event failed");

\_task\_block();

} **else** {

**printf**("\nEvent was created successfully");

}

* + 1. Use **lwevent\_set()** in Task\_Ato indicate that an event occurred when the button is Pressed. A message to a terminal must be sent whether the event failed to be set showing the error number.

result = **\_lwevent\_set**(&lwevent, 0x01); //Activation of the event

**if** (result != MQX\_OK) {

**printf**("\nSetting event failed. Error: 0x%X", result);

}

* + 1. Use **\_lwevent\_wait\_ticks**() in Task\_B to wait for the event to occur. After this the LED must toggle.

result = **\_lwevent\_wait\_ticks**(&lwevent, 0x01, FALSE, 0); //wait for event

**if** (result != MQX\_OK) {

**printf**("\nWaiting event failed. Error: 0x%X", result);

}  
GPIO\_DRV\_TogglePinOutput(BOARD\_GPIO\_LED\_BLUE);

\_time\_delay(1000);

* + 1. Use **\_lwevent\_clear**() in Task\_B after the code that toggles the LED.  
        **\_lwevent\_clear**(&lwevent, 0x01); // Clear the event flag
  1. In Main\_Task use **\_task\_create()** to create an instance of Task\_B. You can refer to **Lab – Using Tasks** or you can see **Freescale MQX™ RTOS Reference Manual.pdf** for details about tasks functions and their parameters. You can find it in ***<KSDK\_install\_dir>/rtos/mqx/doc***.
     1. The task id number must be saved in a variable of type \_task\_id.
     2. 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.
     3. After creating Task\_B, Main\_task must remain blocked using **\_task\_block**() function.

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");

}

* 1. Task A and Task B should look like this:

**void** **Task\_A**(uint32\_t initial\_data) {

**while** (1) {

**if** (GPIO\_DRV\_ReadPinInput(*kGpioSW1*) == 0) { //0 = PRESS BUTTON

result = **\_lwevent\_set**(&lwevent, 0x01); //Activation of the event

**if** (result != MQX\_OK) {

**printf**("\nSetting event failed. Error: 0x%X", result);

}

}

}

}

**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);

}

GPIO\_DRV\_TogglePinOutput(BOARD\_GPIO\_LED\_BLUE);

\_time\_delay(1000);

**\_lwevent\_clear**(&lwevent, 0x01); // Clear the event flag

}

}

* 1. Compile and debug your project. You must be able to toggle LED1 when SW1 is pressed.
  2. 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.

uint32\_t count1 = 0;

* 1. Compile and debug the project. Look at the output in the terminal. What happens? Why?

* 1. 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.pdf** for details about Mutex functions and their parameters. You can find it in ***<KSDK\_install\_dir>/rtos/mqx/doc***.
     1. Include header **#include** <mutex.h>
     2. Declare an structure of type MUTEX\_STRUCT, e.g. MUTEX\_STRUCT my\_mutex;
     3. Initialize mutex in Main\_Task using **\_mutex\_init().**
     4. Use **\_mutex\_lock()** and **\_mutex\_unlock()** where needed.
  2. 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.
     1. Task number 4
     2. Task name Task\_C
     3. Stack size 1500

Priority must be lower than Main\_Task and higher than Task\_A.  
{TASK\_C, Task\_C, 1500, 10, "Task C", 0, 0, 0},

Add a definition for the task id.

**#define** TASK\_C 4

* 1. In Main\_Task use **\_task\_create()** to create an instance of Task\_C. You can refer to **Lab - Using Tasks** or you can see **Freescale MQX™ RTOS Reference Manual.pdf** for details about tasks functions and their parameters. You can find it in ***<KSDK\_install\_dir>/rtos/mqx/doc***.
     1. The task id number must be saved in a variable of type \_task\_id.
     2. 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.
     3. After creating Task\_C, Main\_Task must remain blocked using **\_task\_block**() function.

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");

}

* 1. 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.**

### Code

### main.c

**#include** <stdio.h>

**#include** <mqx.h>

**#include** <bsp.h>

**#include** <mutex.h>

/\* Task IDs \*/

**#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);

\_task\_id ta, tb, tc, td;

//Priority of task B and C must be lower than Main\_Task and higher than Task\_A

**const** TASK\_TEMPLATE\_STRUCT MQX\_template\_list[] =

{

/\* Task Index, Function, Stack, Priority, Name, Attributes, Param, Time Slice \*/

{MAIN\_TASK, Main\_task, 1500, 9, "main", MQX\_AUTO\_START\_TASK, 0, 0},

{TASK\_A, Task\_A, 1500, 11, "Task A", 0, 0, 0},

{TASK\_B, Task\_B, 1500, 10, "Task B", 0, 0, 0},//Priority must be lower than Main\_Task and higher than Task\_A

{TASK\_C, Task\_C, 1500, 10, "Task C", 0, 0, 0},

{ 0 }

};

uint32\_t result;

\_task\_id t1,t2,t3;

LWEVENT\_STRUCT lwevent;

MUTEX\_STRUCT my\_mutex;

**void** **Main\_task**(uint32\_t initial\_data) {

**printf**("\n Start Main Task: Events and Mutex\n");

/\*Init LED and buttons\*/

GPIO\_DRV\_Init(switchPins, ledPins);

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** (GPIO\_DRV\_ReadPinInput(*kGpioSW1*) == 0) { //0 = 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** (GPIO\_DRV\_ReadPinInput(*kGpioSW2*) == 0) { //LOW VALUE = PRESS BUTTON

result = **\_lwevent\_set**(&lwevent, 0x02); //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);

GPIO\_DRV\_TogglePinOutput(BOARD\_GPIO\_LED\_BLUE);

\_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);

GPIO\_DRV\_TogglePinOutput(BOARD\_GPIO\_LED\_RED);

\_time\_delay(1000);

**\_mutex\_unlock**(&my\_mutex);

**\_lwevent\_clear**(&lwevent, 0x02);

}

}

/\* EOF \*/