**Angelica Semenec**

**012179180**

**Lab 7**

**TI-RTOS and MCU’s UART on MSP432 MCU**

**Purpose**

The purpose of this lab is to become familiarized with the TI real-time operating system as well as the MCU’s UART to communicate with an external program.

**Exercise 1**

|  |
| --- |
| **hello.c** |
| .  .  .  Void task1(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P1, GPIO\_PIN0);  while(true)  {  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  Task\_sleep(1000);  }  }  Void task2(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P2, GPIO\_PIN1);  while(true)  {  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P2, GPIO\_PIN1);  Task\_sleep(4000);  }  } |

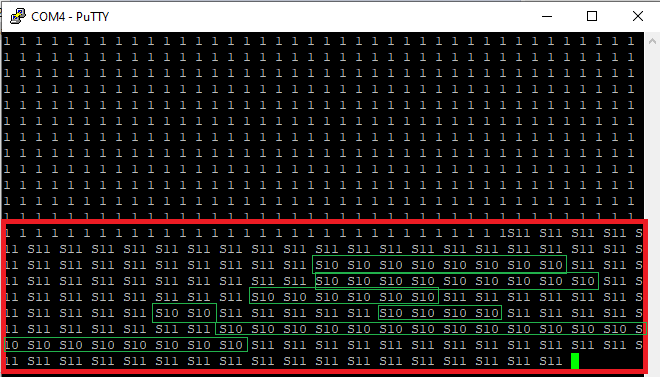
The output pins are configured within each task. Task1 corresponds to the red light and Task2 corresponds to the green light. The function *Task\_sleep()* takes the time in milliseconds to sleep the process. Sleeping Task1 for 1000ms (or 1s) corresponds to a frequency of 1Hz. Sleeping Task2 for 4000ms (or 4s) corresponds to a frequency of 0.25Hz. The function *Task\_sleep()* allows the processor to service other tasks until the sleep time has passed. For each task, the forever loop toggles the output to change the LED to either off or on and then sleeps at the given frequencies.

**Exercise 2**

The default program displays a list of numbers through the terminal. The software tool used to capture the communication is PuTTY.

***Exercise 2.1***

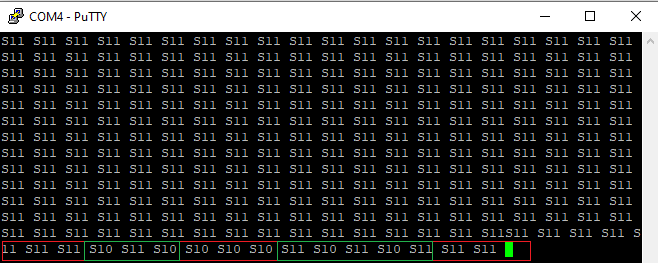
|  |
| --- |
| **hello.c** |
| .  .  .  Void task1(UArg arg0, UArg arg1)  {  //Button S1  MAP\_GPIO\_setAsInputPinWithPullUpResistor(GPIO\_PORT\_P1, GPIO\_PIN1);  .  .  .  unsigned int count = 0;  char buffer[10];  while (1) {  sprintf(buffer, "S1%d ", MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1));  UART\_write(uart, buffer, strlen(buffer));  Task\_sleep(100);  }  }  .  .  . |

****

The terminal output within the red box shows the status of the left-side input pin (S1). When the button is not pushed, “S11” is printed to the terminal (S1 denoting the button and 1 denoting the status as not pushed). Events in which the button is pushed are displayed within the green boxes. The 0 denotes that the button is not pushed. The string is created by reading the status of the push button and printed at 100ms intervals.

***Exercise 2.2***

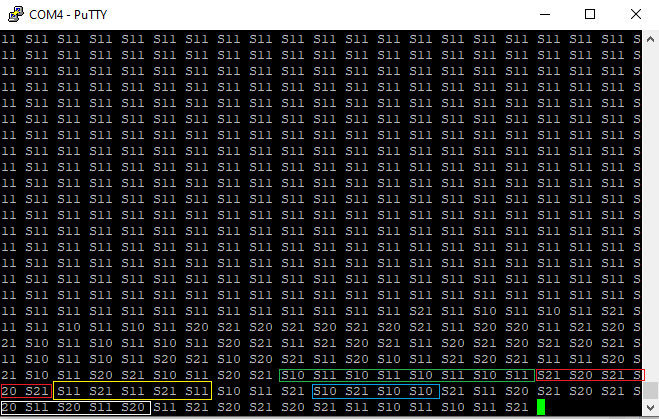
|  |
| --- |
| **hello.c** |
| .  .  .  Void task1(UArg arg0, UArg arg1)  {  .  .  .  time\_t t0;  uint8\_t state;  while (1) {  t0 = time(NULL);  state = MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1);  while (time(NULL) - t0 < 5 && state == MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1))  {  //Spin  }  .  .  .  sprintf(buffer, "S1%d ", MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1));  UART\_write(uart, buffer, strlen(buffer));  }  }  .  .  . |

****

Within the red boxes are instances where the button was either pushed or not pushed for a period of more than 5 seconds. After the state of the button wasn’t changed for 5 seconds, the state of the button was printed to the screen. In the green boxes are instances in which the button was repeatedly pushed and then released. Each time the state changed, the output was immediately printed to the terminal.

***Exercise 2.3***

|  |
| --- |
| **hello.c** |
| .  .  .  Semaphore\_Struct semStruct;  Semaphore\_Handle semHandle;  int main()  {  //NOTE: Code for setting up mutex taken from SDK example code  //mutex\_MSP\_EXP432P401R\_tirtos\_ccs  /\* Construct BIOS objects \*/  Task\_Params taskParams;  Semaphore\_Params semParams;  .  .  .  /\* Construct a Semaphore object to be use as a resource lock, inital count 1 \*/  Semaphore\_Params\_init(&semParams);  Semaphore\_construct(&semStruct, 1, &semParams);  /\* Obtain instance handle \*/  semHandle = Semaphore\_handle(&semStruct);  BIOS\_start(); /\* Does not return \*/  return(0);  }  Void task1(UArg arg0, UArg arg1)  {  //Button S1  MAP\_GPIO\_setAsInputPinWithPullUpResistor(GPIO\_PORT\_P1, GPIO\_PIN1);  printf("Task1\n");  UART\_Handle uart;  UART\_Params uartParams;  UART\_init(); // Driver init  // Set up communication parameters and open the device  UART\_Params\_init(&uartParams);  uartParams.readEcho = UART\_ECHO\_OFF;  char buffer[10];  time\_t t0;  uint8\_t state = 1;  while (1) {  t0 = time(NULL);  while (time(NULL) - t0 < 5 && state == MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1))  {  Task\_yield();  }  state = MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1);  /\* Get access to resource \*/  Semaphore\_pend(semHandle, BIOS\_WAIT\_FOREVER);  uart = UART\_open(CONFIG\_UART\_0, &uartParams);  sprintf(buffer, "S1%d ", state);  UART\_write(uart, buffer, strlen(buffer));  UART\_close(uart);  /\* Unlock resource \*/  Semaphore\_post(semHandle);  }  }  Void task2(UArg arg0, UArg arg1)  {  //Button S2  MAP\_GPIO\_setAsInputPinWithPullUpResistor(GPIO\_PORT\_P1, GPIO\_PIN4);  printf("Task2\n");  UART\_Handle uart;  UART\_Params uartParams;  UART\_init(); // Driver init  // Set up communication parameters and open the device  UART\_Params\_init(&uartParams);  uartParams.readEcho = UART\_ECHO\_OFF;  char buffer[10];  time\_t t0;  uint8\_t state = 1;  while (1) {  t0 = time(NULL);  while (time(NULL) - t0 < 5 && state == MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN4))  {  Task\_yield();  }  state = MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN4);  /\* Get access to resource \*/  Semaphore\_pend(semHandle, BIOS\_WAIT\_FOREVER);  uart = UART\_open(CONFIG\_UART\_0, &uartParams);  sprintf(buffer, "S2%d ", state);  UART\_write(uart, buffer, strlen(buffer));  UART\_close(uart);  /\* Unlock resource \*/  Semaphore\_post(semHandle);  }  } |



The green box shows the output when S1 is pushed and released several times. The red box shows the output when S2 is pushed and released several times. The yellow box shows the output when both buttons remain unpushed for a period of 5 seconds. The blue box shows the output when S1 is pushed and held. The white box shows the output when S2 is pushed and held.

A mutex lock is set up using the semaphore driver library. The count is initialized to 1 so that only 1 task can acquire access to the resource at a time.

Each task sets up its necessary parameters to utilize the UART interface before entering the while loop that polls the buttons and writes to the UART. Within this loop, the current time is recorded. Another loop checks the amount of time passed and the state of the button against the current state. If the state is unchanged or not enough time has passed, the task will yield to another task. The other task can then proceed. When a change is detected or the time limit has been reached, the state of the button at that time is updated. The task then checks to see whether the lock is available or not. If the lock is not available, it will wait until it can acquire the lock. Once the task acquires ownership of the lock, it will open a connection to the UART, transmit the data, then close the connection. After the connection has been closed, it will release the lock, which can then be used by another task that is waiting for the lock.

***Exercise 2.4***

|  |
| --- |
| **hello.c** |
| .  .  .  Void task3(UArg arg0, UArg arg1)  {  UART\_Handle uart;  UART\_Params uartParams;  UART\_init(); // Driver init  // Set up communication parameters and open the device  UART\_Params\_init(&uartParams);  uartParams.readEcho = UART\_ECHO\_OFF;  uartParams.readTimeout = 1; //Read timeout for no key detection  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P2, GPIO\_PIN2); //Blue LED  printf("Task3\n");  char buffer[1];  while (1) {  /\* Get access to resource \*/  Semaphore\_pend(semHandle, BIOS\_WAIT\_FOREVER);  uart = UART\_open(CONFIG\_UART\_0, &uartParams);  UART\_read(uart, buffer, 1);  UART\_close(uart);  /\* Unlock resource \*/  Semaphore\_post(semHandle);  if (buffer[0] == '0')  {  MAP\_GPIO\_setOutputLowOnPin(GPIO\_PORT\_P2, GPIO\_PIN2); //Turn off LED  }  else if (buffer[0] == '1')  {  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P2, GPIO\_PIN2); //Turn on LED  }  Task\_yield();  }  } |

**Appendix**

**Exercise 1**

|  |
| --- |
| **hello.c** |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <string.h>  /\* XDC module Headers \*/  #include <xdc/std.h>  #include <xdc/runtime/System.h>  /\* BIOS module Headers \*/  #include <ti/sysbios/BIOS.h>  #include <ti/sysbios/knl/Clock.h>  #include <ti/sysbios/knl/Task.h>  #include <ti/sysbios/knl/Semaphore.h>  #include <ti/drivers/Board.h>  #define \_\_MSP432P4XX\_\_  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  #define TASKSTACKSIZE 2048  Void task1(UArg arg0, UArg arg1);  Void task2(UArg arg0, UArg arg1);  Task\_Struct task1Struct, task2Struct;  Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];  int main()  {  /\* Construct BIOS objects \*/  Task\_Params taskParams;  /\* Call driver init functions \*/  Board\_init();  /\* Construct task threads \*/  Task\_Params\_init(&taskParams);  taskParams.stackSize = TASKSTACKSIZE;  taskParams.stack = &task1Stack;  Task\_construct(&task1Struct, (Task\_FuncPtr)task1, &taskParams, NULL);  taskParams.stack = &task2Stack;  Task\_construct(&task2Struct, (Task\_FuncPtr)task2, &taskParams, NULL);  BIOS\_start(); /\* Does not return \*/  return(0);  }  Void task1(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P1, GPIO\_PIN0);  while(true)  {  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P1, GPIO\_PIN0);  Task\_sleep(1000);  }  }  Void task2(UArg arg0, UArg arg1)  {  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P2, GPIO\_PIN1);  while(true)  {  MAP\_GPIO\_toggleOutputOnPin(GPIO\_PORT\_P2, GPIO\_PIN1);  Task\_sleep(4000);  }  } |

**Exercise 2.1**

|  |
| --- |
| **hello.c** |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <string.h>  /\* XDC module Headers \*/  #include <xdc/std.h>  #include <xdc/runtime/System.h>  /\* BIOS module Headers \*/  #include <ti/sysbios/BIOS.h>  #include <ti/sysbios/knl/Clock.h>  #include <ti/sysbios/knl/Task.h>  #include <ti/sysbios/knl/Semaphore.h>  #include <ti/drivers/Board.h>  #define \_\_MSP432P4XX\_\_  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  #include <ti/drivers/UART.h>  #include "ti\_drivers\_config.h"  #define TASKSTACKSIZE 2048  Void task1(UArg arg0, UArg arg1);  Void task2(UArg arg0, UArg arg1);  Task\_Struct task1Struct, task2Struct;  Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];  int main()  {  /\* Construct BIOS objects \*/  Task\_Params taskParams;  /\* Call driver init functions \*/  Board\_init();  /\* Construct task threads \*/  Task\_Params\_init(&taskParams);  taskParams.stackSize = TASKSTACKSIZE;  taskParams.stack = &task1Stack;  Task\_construct(&task1Struct, (Task\_FuncPtr)task1, &taskParams, NULL);  taskParams.stack = &task2Stack;  Task\_construct(&task2Struct, (Task\_FuncPtr)task2, &taskParams, NULL);  BIOS\_start(); /\* Does not return \*/  return(0);  }  Void task1(UArg arg0, UArg arg1)  {  //Button S1  MAP\_GPIO\_setAsInputPinWithPullUpResistor(GPIO\_PORT\_P1, GPIO\_PIN1);  printf("Task1\n");  UART\_Handle uart;  UART\_Params uartParams;  UART\_init(); // Driver init  // Set up communication parameters and open the device  UART\_Params\_init(&uartParams);  uartParams.readEcho = UART\_ECHO\_OFF;  uart = UART\_open(CONFIG\_UART\_0, &uartParams);  if (uart == NULL) {  printf("Failed to open UART.\n");  while (1);  }  unsigned int count = 0;  char buffer[10];  while (1) {  sprintf(buffer, "S1%d ", MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1));  UART\_write(uart, buffer, strlen(buffer));  Task\_sleep(100);  }  }  Void task2(UArg arg0, UArg arg1)  {  printf("Task2\n");  } |

**Exercise 2.2**

|  |
| --- |
| **hello.c** |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <string.h>  /\* XDC module Headers \*/  #include <xdc/std.h>  #include <xdc/runtime/System.h>  /\* BIOS module Headers \*/  #include <ti/sysbios/BIOS.h>  #include <ti/sysbios/knl/Clock.h>  #include <ti/sysbios/knl/Task.h>  #include <ti/sysbios/knl/Semaphore.h>  #include <ti/drivers/Board.h>  #define \_\_MSP432P4XX\_\_  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  #include <ti/drivers/UART.h>  #include "ti\_drivers\_config.h"  #define TASKSTACKSIZE 2048  Void task1(UArg arg0, UArg arg1);  Void task2(UArg arg0, UArg arg1);  Task\_Struct task1Struct, task2Struct;  Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];  int main()  {  /\* Construct BIOS objects \*/  Task\_Params taskParams;  /\* Call driver init functions \*/  Board\_init();  /\* Construct task threads \*/  Task\_Params\_init(&taskParams);  taskParams.stackSize = TASKSTACKSIZE;  taskParams.stack = &task1Stack;  Task\_construct(&task1Struct, (Task\_FuncPtr)task1, &taskParams, NULL);  taskParams.stack = &task2Stack;  Task\_construct(&task2Struct, (Task\_FuncPtr)task2, &taskParams, NULL);  BIOS\_start(); /\* Does not return \*/  return(0);  }  Void task1(UArg arg0, UArg arg1)  {  //Button S1  MAP\_GPIO\_setAsInputPinWithPullUpResistor(GPIO\_PORT\_P1, GPIO\_PIN1);  printf("Task1\n");  UART\_Handle uart;  UART\_Params uartParams;  UART\_init(); // Driver init  // Set up communication parameters and open the device  UART\_Params\_init(&uartParams);  uartParams.readEcho = UART\_ECHO\_OFF;  uart = UART\_open(CONFIG\_UART\_0, &uartParams);  if (uart == NULL) {  printf("Failed to open UART.\n");  while (1);  }  unsigned int count = 0;  char buffer[10];  time\_t t0;  uint8\_t state;  while (1) {  t0 = time(NULL);  state = MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1);  while (time(NULL) - t0 < 5 && state == MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1))  {  //Spin  }  sprintf(buffer, "S1%d ", MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1));  UART\_write(uart, buffer, strlen(buffer));  //Task\_sleep(100);  }  }  Void task2(UArg arg0, UArg arg1)  {  printf("Task2\n");  } |

**Exercise 2.3**

|  |
| --- |
| **hello.c** |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <string.h>  /\* XDC module Headers \*/  #include <xdc/std.h>  #include <xdc/runtime/System.h>  /\* BIOS module Headers \*/  #include <ti/sysbios/BIOS.h>  #include <ti/sysbios/knl/Clock.h>  #include <ti/sysbios/knl/Task.h>  #include <ti/sysbios/knl/Semaphore.h>  #include <ti/drivers/Board.h>  #define \_\_MSP432P4XX\_\_  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  #include <ti/drivers/UART.h>  #include "ti\_drivers\_config.h"  #define TASKSTACKSIZE 2048  Void task1(UArg arg0, UArg arg1);  Void task2(UArg arg0, UArg arg1);  Task\_Struct task1Struct, task2Struct;  Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE];  Semaphore\_Struct semStruct;  Semaphore\_Handle semHandle;  int main()  {  //NOTE: Code for setting up mutex taken from SDK example code  //mutex\_MSP\_EXP432P401R\_tirtos\_ccs  /\* Construct BIOS objects \*/  Task\_Params taskParams;  Semaphore\_Params semParams;  /\* Call driver init functions \*/  Board\_init();  /\* Construct task threads \*/  Task\_Params\_init(&taskParams);  taskParams.stackSize = TASKSTACKSIZE;  taskParams.stack = &task1Stack;  Task\_construct(&task1Struct, (Task\_FuncPtr)task1, &taskParams, NULL);  taskParams.stack = &task2Stack;  Task\_construct(&task2Struct, (Task\_FuncPtr)task2, &taskParams, NULL);  /\* Construct a Semaphore object to be use as a resource lock, inital count 1 \*/  Semaphore\_Params\_init(&semParams);  Semaphore\_construct(&semStruct, 1, &semParams);  /\* Obtain instance handle \*/  semHandle = Semaphore\_handle(&semStruct);  BIOS\_start(); /\* Does not return \*/  return(0);  }  Void task1(UArg arg0, UArg arg1)  {  //Button S1  MAP\_GPIO\_setAsInputPinWithPullUpResistor(GPIO\_PORT\_P1, GPIO\_PIN1);  printf("Task1\n");  UART\_Handle uart;  UART\_Params uartParams;  UART\_init(); // Driver init  // Set up communication parameters and open the device  UART\_Params\_init(&uartParams);  uartParams.readEcho = UART\_ECHO\_OFF;  char buffer[10];  time\_t t0;  uint8\_t state = 1;  while (1) {  t0 = time(NULL);  while (time(NULL) - t0 < 5 && state == MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1))  {  Task\_yield();  }  state = MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1);  /\* Get access to resource \*/  Semaphore\_pend(semHandle, BIOS\_WAIT\_FOREVER);  uart = UART\_open(CONFIG\_UART\_0, &uartParams);  sprintf(buffer, "S1%d ", state);  UART\_write(uart, buffer, strlen(buffer));  UART\_close(uart);  /\* Unlock resource \*/  Semaphore\_post(semHandle);  }  }  Void task2(UArg arg0, UArg arg1)  {  //Button S2  MAP\_GPIO\_setAsInputPinWithPullUpResistor(GPIO\_PORT\_P1, GPIO\_PIN4);  printf("Task2\n");  UART\_Handle uart;  UART\_Params uartParams;  UART\_init(); // Driver init  // Set up communication parameters and open the device  UART\_Params\_init(&uartParams);  uartParams.readEcho = UART\_ECHO\_OFF;  char buffer[10];  time\_t t0;  uint8\_t state = 1;  while (1) {  t0 = time(NULL);  while (time(NULL) - t0 < 5 && state == MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN4))  {  Task\_yield();  }  state = MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN4);  /\* Get access to resource \*/  Semaphore\_pend(semHandle, BIOS\_WAIT\_FOREVER);  uart = UART\_open(CONFIG\_UART\_0, &uartParams);  sprintf(buffer, "S2%d ", state);  UART\_write(uart, buffer, strlen(buffer));  UART\_close(uart);  /\* Unlock resource \*/  Semaphore\_post(semHandle);  }  } |

**Exercise 2.4**

|  |
| --- |
| **hello.c** |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <string.h>  /\* XDC module Headers \*/  #include <xdc/std.h>  #include <xdc/runtime/System.h>  /\* BIOS module Headers \*/  #include <ti/sysbios/BIOS.h>  #include <ti/sysbios/knl/Clock.h>  #include <ti/sysbios/knl/Task.h>  #include <ti/sysbios/knl/Semaphore.h>  #include <ti/drivers/Board.h>  #define \_\_MSP432P4XX\_\_  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>  #include <ti/drivers/UART.h>  #include "ti\_drivers\_config.h"  #define TASKSTACKSIZE 2048  Void task1(UArg arg0, UArg arg1);  Void task2(UArg arg0, UArg arg1);  Void task3(UArg arg0, UArg arg1);  Task\_Struct task1Struct, task2Struct, task3Struct;  Char task1Stack[TASKSTACKSIZE], task2Stack[TASKSTACKSIZE], task3Stack[TASKSTACKSIZE];  Semaphore\_Struct semStruct;  Semaphore\_Handle semHandle;  int main()  {  //NOTE: Code for setting up mutex taken from SDK example code  //mutex\_MSP\_EXP432P401R\_tirtos\_ccs  /\* Construct BIOS objects \*/  Task\_Params taskParams;  Semaphore\_Params semParams;  /\* Call driver init functions \*/  Board\_init();  /\* Construct task threads \*/  Task\_Params\_init(&taskParams);  taskParams.stackSize = TASKSTACKSIZE;  taskParams.stack = &task1Stack;  Task\_construct(&task1Struct, (Task\_FuncPtr)task1, &taskParams, NULL);  taskParams.stack = &task2Stack;  Task\_construct(&task2Struct, (Task\_FuncPtr)task2, &taskParams, NULL);  taskParams.stack = &task3Stack;  Task\_construct(&task3Struct, (Task\_FuncPtr)task3, &taskParams, NULL);  /\* Construct a Semaphore object to be use as a resource lock, inital count 1 \*/  Semaphore\_Params\_init(&semParams);  Semaphore\_construct(&semStruct, 1, &semParams);  /\* Obtain instance handle \*/  semHandle = Semaphore\_handle(&semStruct);  BIOS\_start(); /\* Does not return \*/  return(0);  }  Void task1(UArg arg0, UArg arg1)  {  //Button S1  MAP\_GPIO\_setAsInputPinWithPullUpResistor(GPIO\_PORT\_P1, GPIO\_PIN1);  printf("Task1\n");  UART\_Handle uart;  UART\_Params uartParams;  UART\_init(); // Driver init  // Set up communication parameters and open the device  UART\_Params\_init(&uartParams);  uartParams.readEcho = UART\_ECHO\_OFF;  char buffer[10];  time\_t t0;  uint8\_t state = 1;  while (1) {  t0 = time(NULL);  while (time(NULL) - t0 < 5 && state == MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1))  {  Task\_yield();  }  state = MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN1);  /\* Get access to resource \*/  Semaphore\_pend(semHandle, BIOS\_WAIT\_FOREVER);  uart = UART\_open(CONFIG\_UART\_0, &uartParams);  sprintf(buffer, "S1%d ", state);  UART\_write(uart, buffer, strlen(buffer));  UART\_close(uart);  /\* Unlock resource \*/  Semaphore\_post(semHandle);  }  }  Void task2(UArg arg0, UArg arg1)  {  //Button S2  MAP\_GPIO\_setAsInputPinWithPullUpResistor(GPIO\_PORT\_P1, GPIO\_PIN4);  printf("Task2\n");  UART\_Handle uart;  UART\_Params uartParams;  UART\_init(); // Driver init  // Set up communication parameters and open the device  UART\_Params\_init(&uartParams);  uartParams.readEcho = UART\_ECHO\_OFF;  char buffer[10];  time\_t t0;  uint8\_t state = 1;  while (1) {  t0 = time(NULL);  while (time(NULL) - t0 < 5 && state == MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN4))  {  Task\_yield();  }  state = MAP\_GPIO\_getInputPinValue(GPIO\_PORT\_P1, GPIO\_PIN4);  /\* Get access to resource \*/  Semaphore\_pend(semHandle, BIOS\_WAIT\_FOREVER);  uart = UART\_open(CONFIG\_UART\_0, &uartParams);  sprintf(buffer, "S2%d ", state);  UART\_write(uart, buffer, strlen(buffer));  UART\_close(uart);  /\* Unlock resource \*/  Semaphore\_post(semHandle);  }  }  Void task3(UArg arg0, UArg arg1)  {  UART\_Handle uart;  UART\_Params uartParams;  UART\_init(); // Driver init  // Set up communication parameters and open the device  UART\_Params\_init(&uartParams);  uartParams.readEcho = UART\_ECHO\_OFF;  uartParams.readTimeout = 1; //Read timeout for no key detection  MAP\_GPIO\_setAsOutputPin(GPIO\_PORT\_P2, GPIO\_PIN2); //Blue LED  printf("Task3\n");  char buffer[1];  while (1) {  /\* Get access to resource \*/  Semaphore\_pend(semHandle, BIOS\_WAIT\_FOREVER);  uart = UART\_open(CONFIG\_UART\_0, &uartParams);  UART\_read(uart, buffer, 1);  UART\_close(uart);  /\* Unlock resource \*/  Semaphore\_post(semHandle);  if (buffer[0] == '0')  {  MAP\_GPIO\_setOutputLowOnPin(GPIO\_PORT\_P2, GPIO\_PIN2); //Turn off LED  }  else if (buffer[0] == '1')  {  MAP\_GPIO\_setOutputHighOnPin(GPIO\_PORT\_P2, GPIO\_PIN2); //Turn on LED  }  Task\_yield();  }  } |