INSTRUCTIONS

Please attempt all questions. Any type of calculator is allowed. Write your answers in the provided answer booklet ensuring you indicate which question you are answering. Part marks will be given for showing your working. Marks for each question as indicated.

QUESTION 1

In your own words define what is an embedded system and discuss two common design requirements for such systems.

(1 Mark)

QUESTION 2

Define the difference between a soft real-time and hard real-time system and give an example of each.

(1 Mark)

QUESTION 3

Describe the two different models used in Real-Time Operating Systems (RTOS) for thread intercommunication and indicate one freeRTOS kernel object that can be used to implement each model.

(1 Mark)

cont/…

QUESTION 4

Below is an example of a direct register access piece of code within a software driver function that aims to get the direction and control mode for a specific GPIO pin:

```
uint32_t ui32Dir, ui32AFSEL;

//
// Check the arguments.
//
ASSERT(_GPIOBaseValid(ui32Port));
ASSERT(ui8Pin < 8);

//
// Convert from a pin number to a bit position.
//
ui8Pin = 1 << ui8Pin;

//
// Return the pin direction and mode.
//
ui32Dir = HWREG(ui32Port + GPIO_O_DIR);
ui32AFSEL = HWREG(ui32Port + GPIO_O_AFSEL);
return(((ui32Dir & ui8Pin) ? 1 : 0) | ((ui32AFSEL & ui8Pin) ? 2 : 0));
```

Using the code as a guide, complete the software driver model function definition **GPIODirModeGet** by filling in the return and input arguments using the blank spaces below. Make sure to include the correct datatype in your answer:

_____**GPIODirModeGet** ( )

(1 Mark)

cont/…

QUESTION 5

An accelerometer sensor is connected to a microcontroller via an I2C bus and configured to run at a baud rate of 1500 Bits Per Second (bps). The accelerometer measures the acceleration on three axes (X, Y & Z) and features three separate GPIO interrupt lines that are triggered when a new reading is available for each axis of the sensor. The accelerometer has been configured to run in continuous sampling mode at a rate of 120Hz for each axis. The registers within the accelerometer are 16 bits in length. The microcontroller has been configured to have one separate Interrupt Service Routine (ISR) for each GPIO interrupt line of the accelerometer that initiates a read of the accelerometer result register via I2C. It was measured that each ISR adds an additional 50 clock cycles on top of the time taken to perform the I2C read operation. The microcontroller is configured to run with a CPU clock speed of 10Mhz. Assume that the interrupts for each axis are executed consecutively if they are triggered at the same time from the accelerometer. Assume for each I2C transmission the slave address is 8 bits and the register address is 8 bits. You do not need to include the start, stop and acknowledge bits in your calculation for the transmission time.

(a) For this sensor how long will it take to perform one reading from a single axis?

(2 Marks)

(b) Based on your calculation from part (a), indicate if the microcontroller can achieve the sampling rate or not for this sensor and provide your calculations for why this is the case.

(1 Mark)

cont/…

QUESTION 6

The diagram in **Figure 1** below illustrates a simplified single accumulator processor. The current state of the processor is given within the diagram. Assume load, subtract, and store instructions use the value of the accumulator. The processor has just finished executing the first instruction located at memory address 0, as shown by the state of the Current Instruction Register (CIR) and the Accumulator (ACC), and has incremented the Program Counter (PC). A simplified instruction set is shown below for this example which has a 4-bit Op Code and 8-bit Operand.

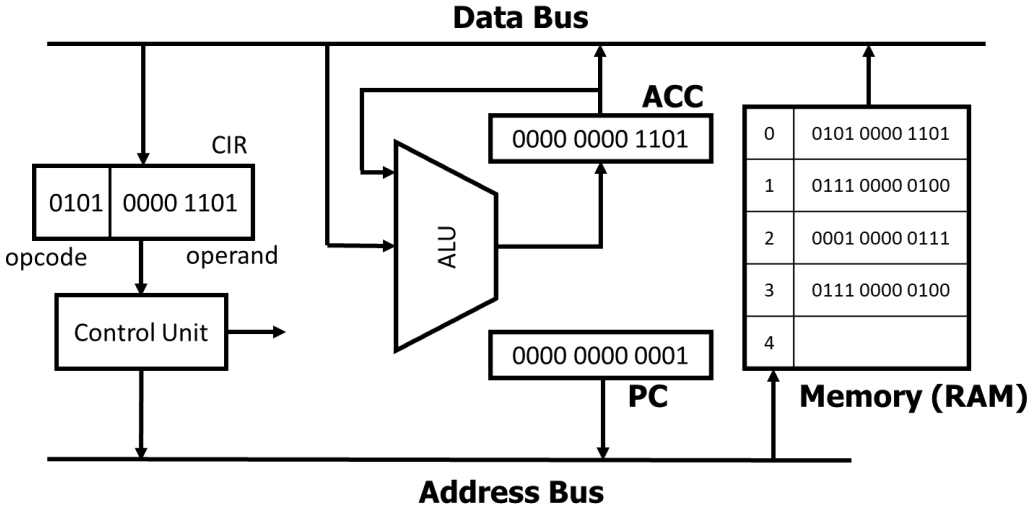

Table 1. Simplified Instruction Set

Instruction Name	Op Code	Operand
Load Operand to ACC	0101	XXXX XXXX
Subtract Operand From ACC, Store Result in ACC	0001	XXXX XXXX
Store ACC at Operand	0111	XXXX XXXX

Figure 1 - A diagram of a simplified single accumulator processor showing the Address and Data bus, memory unit (RAM), Program Counter (PC), Current Instruction Register (CIR), Arithmetic Logic Unit (ALU) and Accumulator register (ACC). A table of instructions for this example accumulator processor is shown below the diagram.

(a) What will be the value in the accumulator after the second instruction (PC = 0000 0000 0001) has been fetched, decoded, and executed?

(1 Mark)

(b) What will be the value in memory location 4 once all instructions have been executed? (Pad your answer with 0's to fill the size of the memory)

(1 Mark)

(c) What is the word size of the example single accumulator processor in **Figure 1**?

(1 Mark)

cont/…

QUESTION 7

The following code snippet was compiled for a Texas Instruments Tiva C series microcontroller (TM4C1294NCPDT) with the Tivaware Peripheral Driver Libraries:

```
extern void I2CHandler(void);

void init(void) {

    uint32_t SysClock;

    SysClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |
                                   SYSCTL_CFG_VCO_480), 120000000);

    SysCtlPeripheralEnable(I2C0_SYSCTL_PERIPH);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);

    GPIOPinTypeI2CSCL(I2C0_GPIO_PORT, I2C0_SCL_PIN);
    GPIOPinTypeI2C(I2C0_GPIO_PORT, I2C0_SDA_PIN);

    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);
    I2CMasterEnable(I2C0_BASE);
    IntEnable(I2C0_BASE);

    IntRegister(INT_I2C0, I2CHandler);

}
```

(a) Describe in your own words what peripheral device this program initialises and what functionality does this create for the program?

(1 Mark)

(b) There is a mistake in this code example. What is the mistake and describe why?

(1 Mark)

cont/…

QUESTION 8

The OPT3001 light sensor is connected to a Tiva C series microcontroller (TM4C1294NCPDT) via a two-wire I2C bus. The OPT3001 sensor is currently configured to have an automatic full-scale range, a conversion time of 800ms and operate in single-shot mode by setting the configuration register to the hexadecimal value of 0xCA10. The OPT3001 sensor has a slave address of 0x51.

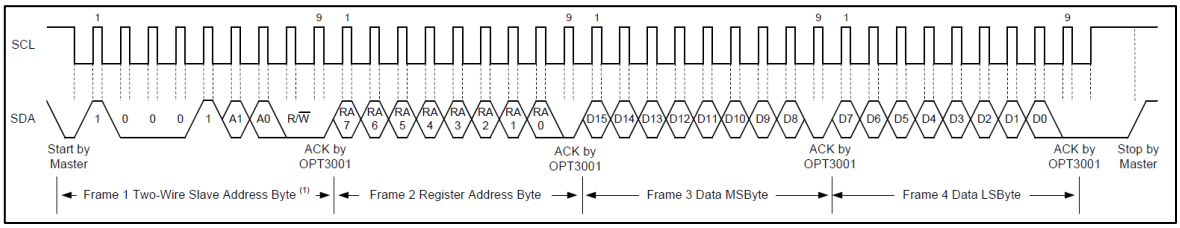

Figure 2 - A diagram of an i2c write transfer from the master to slave device of the OPT3001 light sensor.

REGISTER	ADDRESS (Hex)(1)	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Result	00h	E3	E2	E1	E0	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0
Configuration	01h	RN3	RN2	RN1	RN0	CT	M1	M0	OVF	CRF	FH	FL	L	POL	ME	FC1	FC0
Low Limit	02h	LE3	LE2	LE1	LE0	TL11	TL10	TL9	TL8	TL7	TL6	TL5	TL4	TL3	TL2	TL1	TL0
High Limit	03h	HE3	HE2	HE1	HE0	TH11	TH10	TH9	TH8	TH7	TH6	TH5	TH4	TH3	TH2	TH1	TH0
Manufacturer ID	7Eh	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
Device ID	7Fh	DID15	DID14	DID13	DID12	DID11	DID10	DID9	DID8	DID7	DID6	DID5	DID4	DID3	DID2	DID1	DID0

Table 1 – Map of registers within the OPT3001 sensor indicating the address and corresponding bit fields in each register.

RN3	RN2	RN1	RN0	FULL-SCALE RANGE (lux)	LSB SIZE (lux per LSB)
0	0	0	0	40.95	0.01
0	0	0	1	81.90	0.02
0	0	1	0	163.80	0.04
0	0	1	1	327.60	0.08
0	1	0	0	655.20	0.16
0	1	0	1	1310.40	0.32
0	1	1	0	2620.80	0.64
0	1	1	1	5241.60	1.28
1	0	0	0	10483.20	2.56
1	0	0	1	20966.40	5.12
1	0	1	0	41932.80	10.24
1	0	1	1	83865.60	20.48

Table 2 – Full Scale Range and LSB size as a function of Exponent Level or Range Number Field in Configuration Register.

Question 8 continued overleaf

cont/…

Question 8 continued

Bit	Field	Type	Reset	Description
15:12	RN[3:0]	R/W	1100b	**Range number field (read or write).** The range number field selects the full-scale lux range of the device. The format of this field is the same as the result register exponent field (E[3:0]); When RN[3:0] is set to 1100b (0Ch), the device operates in automatic full-scale setting mode, as described in the *Automatic Full-Scale Setting Mode* section. In this mode, the automatically chosen range is reported in the result exponent (register 00h, E[3:0]). The device powers up as 1100 in automatic full-scale setting mode. Codes 1101b, 1110b, and 1111b (0Dh, 0Eh, and 0Fh) are reserved for future use.
11	CT	R/W	1b	**Conversion time field (read or write).** The conversion time field determines the length of the light to digital conversion process. The choices are 100 ms and 800 ms. A longer integration time allows for a lower noise measurement. Range 0101b reduces by one bit. Ranges 0100b, 0011b, 0010b, and 0001b reduces by two bits. Range 0000b reduces by three bits. The result register format and associated LSB weight does not change as a function of the conversion time. 0 = 100 ms 1 = 800 ms
10:9	M[1:0]	R/W	00b	**Mode of conversion operation field (read or write).** The mode of conversion operation field controls whether the device is operating in continuous conversion, single-shot, or low-power shutdown mode. The default is 00b (shutdown mode), such that upon power-up, the device only consumes operational level power after appropriately programming the device.. 00 = Shutdown (default) 01 = Single shot 10, 11 = Continuous conversions
8	OVF	R	0b	**Overflow flag field (read-only).**
7	CRF	R	0b	**Conversion ready field (read-only).** The conversion ready field indicates when a conversion completes. The field is set to 1 at the end of a conversion and is cleared (set to 0) when the configuration register is subsequently read or written with any value except one containing the shutdown mode (mode of operation field, M[1:0] = 00b).
6	FH	R	0b	**Flag high field (read-only).** The flag high field (FH) identifies that the result of a conversion is larger than a specified level of interest. FH is set to 1 when the result is larger than the level in the high-limit register (register address 03h) for a consecutive number of measurements defined by the fault count field (FC[1:0]). See the *Interrupt Reporting Mechanism Modes* section for more details on clearing and other behaviors of this field.
5	FL	R	0b	**Flag low field (read-only).** The flag low field (FL) identifies that the result of a conversion is smaller than a specified level of interest. FL is set to 1 when the result is smaller than the level in the low-limit register (register address 02h) for a consecutive number of measurements defined by the fault count field (FC[1:0]). See the *Interrupt Reporting Mechanism Modes* section for more details on clearing and other behaviors of this field.
4	L	R/W	1b	**Latch field (read or write).** The latch field controls the functionality of the interrupt reporting mechanisms: the INT pin, the flag high field (FH), and flag low field (FL). This bit selects the reporting style between a latched window-style comparison and a transparent hysteresis-style comparison. 0 = The device functions in transparent hysteresis-style comparison operation, where the three interrupt reporting mechanisms directly reflect the comparison of the result register with the high- and low-limit registers with no user-controlled clearing event. See the *Interrupt Operation, INT Pin, and Interrupt Reporting Mechanisms* section for further details. 1 = The device functions in latched window-style comparison operation, latching the interrupt reporting mechanisms until a user-controlled clearing event.
3	POL	R/W	0b	**Polarity field (read or write).** The polarity field controls the polarity or active state of the INT pin. 0 = The INT pin reports active low, pulling the pin low upon an interrupt event. 1 = Operation of the INT pin is inverted, where the INT pin reports active high, becoming high impedance and allowing the INT pin to be pulled high upon an interrupt event.
2	ME	R/W	0b	**Mask exponent field (read or write).** The mask exponent field forces the result register exponent field (register 00h, bits E[3:0]) to 0000b when the full-scale range is manually set, which can simplify the processing of the result register when the full-scale range is manually programmed.
1:0	FC[1:0]	R/W	00b	**Fault count field (read or write).** 00 = One fault count (default) 01 = Two fault counts 10 = Four fault counts 11 = Eight fault counts

Table 2 – A description of the configuration register bit fields (extracted from the OPT3001 datasheet)

Question 8 continued overleaf

cont/…

Question 8 continued

(a) Using **Figure 2** and the tables above as a guide, indicate the bytes in order of first to last byte (left to right) being transmitted to the sensor to perform a write operation of the configuration register of the OPT3001 to configure the sensor to operate at a fixed scale range if maximum values can be up to 500 Lux. Also configure the sensor to operate in continuous conversion mode with a conversion time of 100ms and have the interrupt (INT) pin as active high. All other settings of the configuration register should be preserved and unchanged.

(2 Marks)

(b) Using **Figure 2** and the tables above as a guide, if the configuration register was read as 0xCE13, what does this indicate has occurred within the sensor?

(1 Mark)

QUESTION 9

The following sub-questions relate to using multiple threads within a freeRTOS program.

(a) In a real-time operating system, when all threads have the same priority, give an example of two scenarios that can cause a thread to switch to another one?

(1 Mark)

(a) Name two different types of freeRTOS kernel objects that can be used to share data between threads and describe how they differ?

(1 Mark)

QUESTION 10

Answer the following questions below considering the timing diagram shown in **Figure 3** below and the definition of the worst case schedulable bound (W_N) given in equation 1, where *N* is the number of processes. Assume the CPU time to execute a context switch between threads is negligible.

$$W_N = N\left(2^{1/N} - 1\right) \quad (1)$$

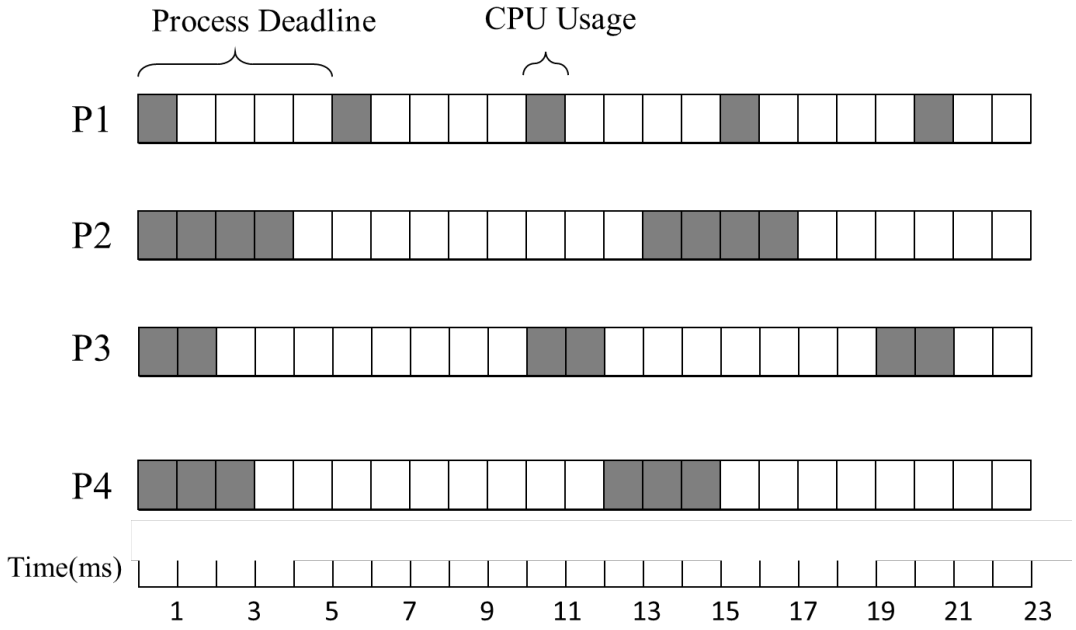

Figure 3 - A timing diagram for four different threads/processes (P) within a multithreaded program. Greyed blocks represent time that the thread/process is utilising the CPU. Each process has its own deadline indicated by the repeating pattern.

(a) If a scheduler was using a rate monotonic scheduling method, show whether all four processes can be scheduled or not to meet their own deadlines using the concept of the worst-case schedulable bound.

(2 Marks)

(b) What scheduling method could be used for this example to schedule all four processes to meet their deadlines and indicate whether there is any disadvantage of using your chosen method.

(1 Mark)

cont/…

QUESTION 11

Consider the following piece of code that has been compiled for a Texas Instruments Tiva C Series microcontroller (TM4C1294NCPDT) and assume that all freeRTOS libraries and include files are added.

```
SemaphoreHandle_t xSemaphoreA, xSemaphoreB;
uint8_t sensor1_reading, sensor2_reading;

void Task1(void *pvParameters){
    while (1){
        xSemaphoreTake(xSemaphoreA, portMAX_DELAY);
       sensor1_reading = readSensor1();

        xSemaphoreTake(xSemaphoreB, portMAX_DELAY);
       sensor2_reading = readSensor2();

        xSemaphoreGive(xSemaphoreB);
        xSemaphoreGive(xSemaphoreA);
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

void Task2(void *pvParameters) {
    while (1) {
        xSemaphoreTake(xSemaphoreB, portMAX_DELAY);
       sensor1_reading = readSensor1();

        xSemaphoreTake(xSemaphoreA, portMAX_DELAY);
       sensor2_reading = readSensor2();

        xSemaphoreGive(xSemaphoreA);
        xSemaphoreGive(xSemaphoreB);
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

void main(){
    xSemaphoreA = xSemaphoreCreateBinary();
    xSemaphoreB = xSemaphoreCreateBinary();

    xTaskCreate(Task1, "Task1", 1000, NULL, 1, NULL);
    xTaskCreate(Task2, "Task2", 1000, NULL, 1, NULL);

    vTaskStartScheduler();
}
```

(a) What is a potential problem in this example piece of code which reads two sensors from two tasks?

(2 Marks)

cont/…

QUESTION 12

Answer the following questions below regarding concurrent access within a multithreaded RTOS program.

(a) What is the difference between a bounded and unbounded priority inversion and give an example of how both can occur.

(1 Mark)

(b) If you can modify the priorities of two tasks that require access to the same shared resource, what is a simple method that is not the priority inheritance or ceiling protocol to ensure they can never conflict and cause issues with concurrent access to a shared resource?

(1 Mark)

QUESTION 13

A freeRTOS program has multiple tasks running with different priority levels assigned to each Interrupt Service Routine (ISR), Task and Idle type. Assume ISR's return to tasks of the highest priority that are ready to run. The context switches that occurred between each thread when the program was executed is shown in **Figure 4**.

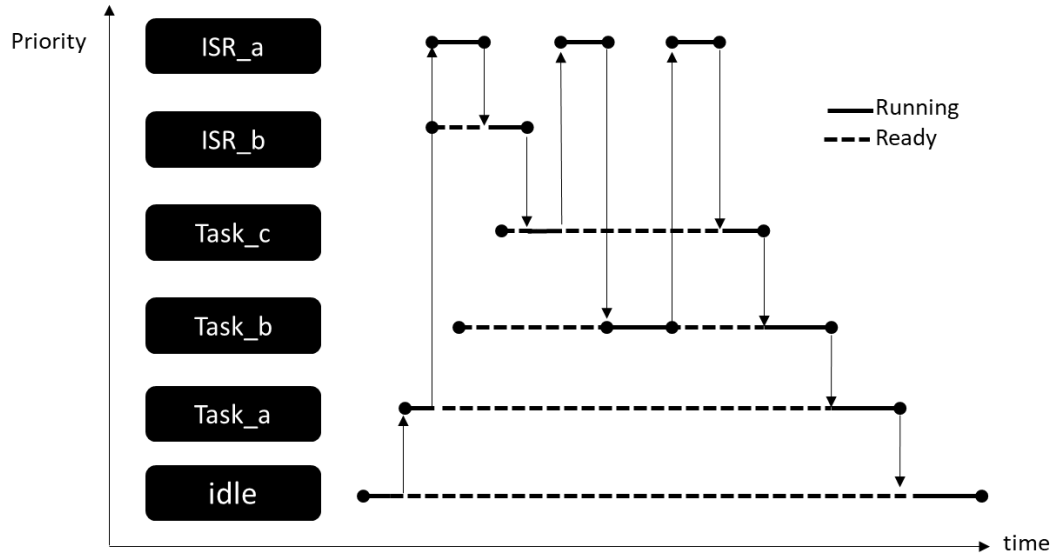

Figure 4 – Task priority and scheduling diagram for a freeRTOS based program.

(a) Using **Figure 4** as a guide, identify one potential problem that has occurred in the execution of this multithreaded program using FreeRTOS terminology. Explain the problem and propose a method, if possible, to solve it.

(2 Marks)

QUESTION 14

Consider the following piece of code that has been compiled for a Texas Instruments Tiva C Series microcontroller (TM4C1294NCPDT) and assume that all freeRTOS libraries and include files are added and the program compiles and runs without any errors.

```
#define Event_Id_00 (1 << 0)
#define Event_Id_01 (1 << 1)

typedef struct MsgObj {
    int         id;
    char        val;
} MsgObj;

SemaphoreHandle_t semHandle;
QueueHandle_t queueHandle;
TaskHandle_t task0Handle, task1Handle;
EventGroupHandle_t evtHandle;

void task0(void *pvParameters)
{
    MsgObj msg;
    EventBits_t evBits;

    while (1)
    {
        UARTprintf("Receiving Msg: ");

        evBits = xEventGroupWaitBits(evtHandle, Event_Id_00 | Event_Id_01, pdTRUE, pdFALSE,
                                     portMAX_DELAY);

        if ((evBits & Event_Id_00) != 0){
            UARTprintf("Got Event: %d ",Event_Id_00);
            if (xSemaphoreTake(semHandle, portMAX_DELAY) == pdPASS){
                xSemaphoreGive(semHandle);
                xQueueReceive(queueHandle, &msg, portMAX_DELAY);
                xEventGroupSetBits(evtHandle, Event_Id_01);
                UARTprintf("Got Message: %d%c ", msg.id, msg.val);
            }
        }

        if ((evBits & Event_Id_01) != 0){
            if (xSemaphoreTake(semHandle, portMAX_DELAY) == pdPASS){
                UARTprintf("Got Event 2");
                xSemaphoreGive(semHandle);
                vTaskDelete(NULL);
            }
        }
    }
}

void task1(void *pvParameters)
{
    MsgObj msg;
    msg.id = 5;
    msg.val = 'k';

    if (xSemaphoreTake(semHandle, portMAX_DELAY) == pdPASS){
        UARTprintf("Task 1 Sending Message: ");
        xQueueSend(queueHandle, &msg, portMAX_DELAY);
        xEventGroupSetBits(evtHandle, Event_Id_00);
        xSemaphoreGive(semHandle);
    }

    vTaskDelete(NULL);
}
```

Question 14 continued overleaf

cont/…

Question 14 continued

```
#define MAX_COUNT 10
#define INITIAL_COUNT 2

int main()
{
    semHandle = xSemaphoreCreateCounting(MAX_COUNT,INITIAL_COUNT);
    queueHandle = xQueueCreate(1, sizeof(MsgObj));
    evtHandle = xEventGroupCreate();

    ConfigureUART();

    xTaskCreate(task0, "Task0", 1000, NULL, 6, &task0Handle);
    xTaskCreate(task1, "Task1", 1000, NULL, 4, &task1Handle);

    xSemaphoreGive(semHandle);

    vTaskStartScheduler();

    for(;;);
}
```

(a) What is the value of the count field within the Semaphore after this code is run (assume all necessary include files and libraries are used)?

(2 Marks)

(b) What string is printed to the serial console when this section of code is run (assume all necessary include files and libraries are used)?

(2 Marks)

END OF PAPER