THE UNIVERSITY OF THE WEST INDIES
ST. AUGUSTINE

ALTERNATIVE ASSESSMENT FOR FINAL EXAMINATIONS OF December 2021

Code and Name of Course: **ECNG3006 Microprocessor Systems** Paper: **Final**

Date and Time: **21 Dec 2021 9AM** Duration: **3 hours**

INSTRUCTIONS TO CANDIDATES: This paper has 6 pages and 3 questions. Max. Marks: **100**

Attempt ALL questions.
Questions 1 and 2 are each worth 25 marks.
Question 3 is worth 50 marks.

Read the ENTIRE question paper before starting.
Please hand-write your answers,
clearly in a legible and readable manner (see Submission Instructions).
Queries regarding this question paper should be addressed to the Examination Invigilator.

The following information was made available to students for use in the fortnight prior to this exam:

- ESP8266 Technical Reference https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf
- Data sheet for the GPIO Expander MCP23017 https://ww1.microchip.com/downloads/en/devicedoc/20001952c.pdf
- RFID-Reader-Documentation-v2.4 https://www.parallax.com/package/rfid-card-reader-documentation
- Mastering the FreeRTOS Real Time Kernel https://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf
- ESP8266 RTOS SDK (ESP-IDF Style) - based on FreeRTOS https://docs.espressif.com/projects/esp8266-rtos-sdk/en/latest/index.html
- Case Study: Conveyor System for a small facility (see pages 5 and 6)

. . . continued

Q1. The ESP8266 RTOS SDK (ESP-IDF style based on FreeRTOS) is an example of a tick-driven open-source real-time operating system (RTOS) kernel, that supports multi-tasking with both pre-emptive and co-operative tasks and jobs.

(a) Differentiate between periodic, aperiodic and sporadic tasks and/or jobs. *5 marks*

(b) A critical real-time system must be implemented using the ESP8266 RTOS SDK. The system has 3 jobs:

- Job A - aperiodic triggered by a high on a GPIO pin
- Job B - periodic with a period of 10 units
- Job C - periodic with a period of 5 units

5 marks

Figure Q1.b on page 3 contains two potential C listings for the implementation. Which listing would you prefer? Justify your choice. Your answer should identify the specific aspects of each listing that would make it preferable in specific contexts.

(c) The ESP8266 RTOS SDK documentation states that:

> The I2C APIs are not thread-safe, if you want to use one I2C port in different tasks, you need to take care of the multi-thread issue.

Explain, in your own words, what is meant by the phrase ”thread-safe”, and describe one strategy to manage the resulting multi-thread issue. *5 marks*

(d) To ensure that critical systems remain “up”, redundant sub-systems are often utilised. Does running two copies of a task, with separate task-control-blocks, on the same processor qualify as a redundant sub-system? Support your answer by referring to historical case studies of embedded system failure. *5 marks*

(e) Describe, in your own words, the role of Trusted Platform Modules (TPM) in embedded systems. Your answer should identify at least one advantage or disadvantage of utilising TPM. *5 marks*

[Q1 Total 25 marks]

Q2. (a) A standard I2C bus (transfer speed: 100 kilo bits/second) is used by an ESP8266 to communicate with a 16-bit ADC. The ADC is configured to sample continuously at 250 samples per second.

i. Explain why ≈ 50 bus-cycles (not 16 bus cycles) are needed to retrieve an ADC reading. *2 marks*

ii. What determines the maximal frequency that 16-bit ADC readings transferred via I2C can be used to reliably detect/track? Explain your reasoning. *3 marks*

(b) Identify one benefit, and one drawback, of utilising coding standards in developing embedded systems that use the ESP8266 RTOS SDK. Support your answer(s) by explaining the rationale for the AUTOSAR Rule A7-5-2:
“Functions shall not call themselves, either directly or indirectly.” [1] . *5 marks*

[1] page 113, Guidelines for the use of the C++14 language in critical and safety-related systems `https://www.autosar.org/fileadmin/user_upload/standards/adaptive/18-10/AUTOSAR_RS_CPP14Guidelines.pdf`

. . . continued

Figure Q1.b: Implementation of a Critical System using the ESP8266 RTOS SDK

Listing I - Pre-emptive Priority with Interrupt

```c
static void JobA(){...};
static void JobB(){...};
static void JobC(){...};
const TickType_t xPeriodB = 10;
const TickType_t xPeriodC = 5;

static void isr_hdler(void *)
{
    JobA();
}

static void per_taskB(void *arg)
{
    TickType_t xLastWakeTime;

    for (;;) {
        xLastWakeTime = xTaskGetTickCount();
        JobB();
        vTaskDelayUntil( &xLastWakeTime, xPeriodB );
    }
}

static void per_taskC(void *arg)
{
    TickType_t xLastWakeTime;

    for (;;) {
        xLastWakeTime = xTaskGetTickCount();
        JobB();
        vTaskDelayUntil( &xLastWakeTime, xPeriodC );
    }
}

void app_main(void)
{
    gpio_config_t io_conf;
    io_conf.intr_type = GPIO_INTR_HIGH_LEVEL;
    io_conf.pin_bit_mask = GPIO_PIN_0;
    io_conf.mode = GPIO_MODE_INPUT;
    gpio_config(&io_conf);

    xTaskCreate(per_taskB, "B", 2048, NULL, 10, NULL);
    xTaskCreate(per_taskC, "C", 2048, NULL, 11, NULL);

    gpio_install_isr_service(0);
    gpio_isr_handler_add(GPIO_NUM_0, isr_hdler, NULL);
}
```

Listing II - Cyclic Executive

```c
static void JobA(){...};
static void JobB(){...};
static void JobC(){...};
const TickType_t xSlot = 5;

static void cyclicx(void *arg)
{
    TickType_t xLastWakeTime;

    for (;;) {
        // Slot 1 of 2
        xLastWakeTime = xTaskGetTickCount();
        if (gpio_get_level(GPIO_INPUT_0))
            JobA();
        JobC();
        vTaskDelayUntil( &xLastWakeTime, xSlot);

        //Slot 2 of 2
        xLastWakeTime = xTaskGetTickCount();
        if (gpio_get_level(GPIO_NUM_0))
            JobA();
        JobB();
        JobC();
        vTaskDelayUntil( &xLastWakeTime, xSlot );
    }
}

void app_main(void)
{
    gpio_config_t io_conf;
    io_conf.pin_bit_mask = GPIO_INPUT_IO_0;
    io_conf.mode = GPIO_MODE_INPUT;
    gpio_config(&io_conf);

    for(;;)
    {
        cyclicx();
    }
}
```

. . . continued

(c) The ESP8266 (ESP-01s) comes pre-programmed with a first-stage boot-loader in ROM. The first stage boot-loader allows the SPI Flash to be re-programmed via the serial line. Identify the three different types of information that are typically programmed into the SPI Flash, and explain how (and in what order) this information is accessed during startup. *5 marks*

(d) Identify and describe one technique for determining an **appropriate** task stack size for the ESP8266 RTOS SDK. Is your chosen technique more likely to under-size or over-size the task stack? What are the potential ramifications of using a "bad" task stack size generated using this technique? *5 marks*

(e) Putting a processor to sleep typically reduces CPU power usage. Explain, in your own words, why developers of ESP8266-based critical-systems might decide to forgo using sleep functions, even when limited power is available. *5 marks*

[Q2 Total 25 marks]

Q3. (a) Using the information provided, draw the precedence graphs for Task B, determine the Worst-Case Execution Time for each task, calculate the Worst-Case System Utilization, and comment critically on whether the system is feasible. *15 marks*

(b) Jobs are scheduled using fixed priorities with a pre-emptive priority based scheduler (see page 6). Construct timelines for a single hyper-period from startup where there are no parcels on the conveyor, no sensors are triggered, and the button on station 0 is pressed at time t=2100 units, for a single parcel to be routed to station 3, which arrives at station 3 at t=7300 units. *10 marks*

(c) The system as described, assigned specific priorities to the tasks. Alternatively, priorities could be assigned using **rate-monotonic** static scheduling, or **shortest job first** dynamic priority scheduling. State, on the basis of three (or more) performance criteria, which priority assignment scheme you would recommend, and justify your recommendation in terms of the application context, and your predictions for system/task operation under the same conditions as in Q3.b. *15 marks*

(d) The emergency stop function is activated by a single human action and should be available and operational at all times ... After an individual activates it, the machinery must not be able to operate again until they reset the function. [2]

You have been asked to modify the conveyor system firmware, so that an emergency stop function switches off all transfer relays. The proposed modification will share the same GPIO pin as the RFID Serial RX function i.e. when no RFID tag reader is active, an interrupt on the RX pin will trigger the emergency stop function. Identify two operational issues and/or concerns with the suggested modification. How can your concerns be adequately addressed in firmware?. *10 marks*

[Q3 Total 50 marks]

END OF QUESTIONS

[2] ANSI Blog: Safety of Machinery – Emergency Stop Function `https://blog.ansi.org/?p=7300`

. . . continued

Case Study: Conveyor System for a small facility

A small processing facility makes use of a conveyor system to route items from one station to another. The conveyor system is arranged so that items will circulate around the facility by moving from station to station. The controller for the system is an ESP-01s with firmware based on ESP8266 RTOS SDK . The controller is connected, via a standard I2C bus, to four GPIO expanders, and an LCD display.The controller is configured to receive TTL-level RS232-type messages from RFID tag readers via the Serial RX line at 2400 baud; this line defaults to LOGIC_LOW when no RFID readers are activated. Only one RFID reader can be active at a time. The controller is also configured to respond to a rising edge interrupt signal from one of the GPIO expanders.

The system can accommodate up to 16 distinct stations (numbered 0 - 15) along the conveyor. At each station, there is a serial RFID tag reader (that is normally de-activated), a sensor that indicates whether an item on the conveyor is passing the station, a momentary push-button for the human operator to press when they wish to place an item on the conveyor, and a relay to control the mechanism that transfers items off of the conveyor. There is also a set of RFID destination station tags used to indicate which station an item should be routed to next.

Items arriving from outside the facility are manually processed (at station 0) before being placed on the conveyor - specifically, each item has an RFID tag attached, the station push-button is pressed, the item is then scanned along with a destination station tag before being placed on the moving conveyor. When an item is detected in front of a station, the RFID code for the item is scanned, and if necessary, the item is transferred off the conveyor to the station by activating the associated relay. An item is only transferred off of the conveyor if the item has been routed to the specified station, AND there is not already an item on that station. A human operator will process the item on their station as needed, and when finished, will press their station push-button, scan the item along with a destination station tag, then manually place the item on the moving conveyor. Items that are complete are routed to station 0, where the RFID tag is manually removed, before the item is packed to leave the facility.

The number of items in circulation on the conveyor at any time is displayed on the LCD display (I2C address 0x34). The station transfer relays are controlled by individual outputs on a GPIO expander (I2C address 0x42). The station push-buttons are connected to individual inputs on a GPIO expander (I2C address 0x44). The RFID tag readers are connected to a single serial RX line on the ESP8266, and are activated via individual outputs on a GPIO expander (I2C address 0x46). The station sensors are connected to individual inputs on a GPIO expander (I2C address 0x48) - in addition the expander is configured to send an interrupt signal if any of the sensor signals goes low. Items associated with a station are connected to the associated GPIO line on the GPIO expanders e.g. the push-button on station #5 is connected to GPIO5 on the expander with I2C address 0x46.

The system is implemented using four tasks, a binary semaphore, a global data structure and a message queue.

- The kernel will be configured to
  - allow interrupt events to preempt the running task;
  - perform pre-emptive priority-based scheduling;
  - prevent pre-emption of tasks accessing the I2C bus;
  - incur negligible overhead at system reset, and scheduler task switch.
- The following tasks will be used to implement the system:
  - Task A a periodic task responsible for updating the LCD display
  - Task B a periodic task responsible for routing an item when a station button is pressed;
  - Task C an aperiodic (interrupt-driven) task that signals Task D when an item is detected;
  - Task D a sporadic task responsible for transferring items off the conveyor.
- Task A updates the LCD display based on messages sent to a message queue Q by Tasks B and D.
- Task C uses a semaphore S to signal Task D.
- A globally accessible data structure G is used to store each item's RFID tag, and the station number it has been routed to. The structure is read from Task D and written to from Task B.

. . . continued

Task	Operation, Jobs, and Estimated Times	
A	`Display Task (periodic with period of 5 seconds = 5000 units)`	
	`Manually Assigned Priority Level 10 - LO`	
	`Static variable count`	
	`  Repeat while Q is not empty`	(JAa, e=1 unit)
	`    Retrieve message m from Q`	(JAb, e=1 unit)
	`    If m came from Task D`	(JAc, e=1 unit)
	`      decrement count`	(JAd, e=1 unit)
	`    Else If m came from Task B then`	(JAe, e=1 unit)
	`      increment count`	(JAf, e=1 unit)
	`  Display count on LCD (via I2C).`	(JAg, e=1 unit)
B	`Routing Task (periodic with period of 1 seconds = 1000 units)`	
	`Manually Assigned Priority Level 11 - MID`	
	`Local variables i,j[12],k[12],s`	
	`  read 16-bit pushbutton GPIO expander status into variable i`	(JBa, e=2 units)
	`  for the first station s (0...15) whose bit in i is SET`	(JBb, e=1 unit)
	`    activate RFID reader (SET bit s on GPIO expander)`	(JBc, e=101 unit)
	`    retrieve 12-byte RFID code from serial port into j`	
	`    if j is a valid parcel code`	(JBd, e=5 units)
	`      retrieve 12-byte RFID code from serial port into k`	(JBe, e=100 units)
	`      if k is a valid station code`	(JBf, e=5 units)
	`        update route in G for parcel j to k`	(JBg, e=11 units)
	`        update status in G for station s to AVAILABLE`	
	`        place a message on Q for Task A`	
	`    deactivate RFID reader (CLR bit s on GPIO expander)`	(JBh, e=2 unit)
	`    clear signal from button s (CLR bit s on GPIO expander)`	
C	`Interrupt Task (aperiodic)`	
	`  send semaphore S to Task D`	(JCa, e=2 units)
	`  disable interrupts`	
D	`Transfer task (periodic with period of 2 seconds = 2000 units)`	
	`Manually Assigned Priority Level 12 - HI`	
	`Local variables i,j[12],s`	
	`  CLR all bits in 16-bit transfer relay GPIO expander`	(JDa, e=1 units)
	`  If semaphore S is available`	(JDb, e=1 units)
	`    retrieve semaphore S`	(JDc, e=4 units)
	`    read 16-bit sensor GPIO expander status into variable i`	
	`    enable interrupts`	
	`    for the first station s (0...15) whose bit in i is CLR`	(JDd, e=1 unit)
	`      if status in G for station s is AVAILABLE`	(JDe, e=5 units)
	`        activate RFID reader (SET bit s on RFID GPIO expander)`	(JDf, e=101 units)
	`        retrieve 12-byte RFID code from serial port into j`	
	`        if j is a valid parcel code`	(JDg, e=5 units)
	`          if route in G for parcel j is RFID code for s`	(JDh, e=10 units)
	`            SET bit s in 16-bit transfer relay GPIO expander`	(JDi, e=7 units)
	`            update status in G for station s to BUSY`	
	`            place a message on Q for Task A`	
	`        deactivate RFID reader (CLR bit s on RFID GPIO expander)`	(JDj, e=1 unit)

END OF EXAM PAPER