College of Science – Computer Science

CSC368/CSCM68 – Embedded Systems Design PRACTICE EXAM

May/June 2021

Release Time: **10:00 (Time Zone: GMT)**

Deadline: **13:00 (Time Zone: GMT)**

Alternative Assessment Information

- You ***MUST*** use your own copy of this assessment from Canvas. If you obtained this assessment document from a friend or elsewhere then delete this copy and use your own version from Canvas. If you experience difficulties to download the assessment, get in contact via the emails below.
- This is an open-book assessment. This means you may use your notes, textbooks, and other resources, including calculators. Copying from resources other than your notes requires referencing.
- You must submit before the deadline. Allow some spare time for technical submission issues.
- The total time for this assessment is 3 hours. This assessment is designed to be sat in a 2-hour window. An additional hour allows you time for accessing the paper, uploading your submission, and dealing with technical issues.
- It is suggested that you use Microsoft Word (or any other editor of your choice) to type your answers, then save as PDF when you are ready to submit. All submitted text (and code if present) must be word-processed, but you may include images (or photos of hand drawn images) as part of the document.
- This is an individual assessment. Under no circumstances are you to discuss any aspect of this assessment with anyone; nor are you allowed to share this document, ideas or solutions with others using email, social media, instant messaging, websites, or any other means. Your attempts at these questions must be entirely your own work. Those found to have collaborated with others will receive a mark of 0.

Special Instructions

Answer all questions. The total number of marks is 50

Submission Instructions

- Please submit a **single PDF** file **named as your student number followed by the module code** (e.g. 123456-CSC789.pdf) via the submission link located on the module page in Canvas.

By submitting, electronically and/or hardcopy, you state that you fully understand and are complying with the university's policy on Academic Integrity and Academic Misconduct. The policy can be found at https://myuni.swansea.ac.uk/academic-life/academic-misconduct.

Originator(s): ***Shane Fleming***

In case of queries email both: s.t.fleming@swansea.ac.uk and h.elliot@swansea.ac.uk

Part A. General

1. What is meant by the term ”memory-mapped I/O“? How is memory mapped I/O typically interacted with?

[4 marks]

2. What are two advantages of using custom hardware in an embedded system?

[2 marks]

3. Give one example of something that can increase the Worst Case Execution Time (WCET) of a task. Why is it often crucial to analyse a tasks WCET when developing an embedded system?

[2 marks]

4. What are two ways that the power consumption of an embedded system can be improved?

[2 marks]

Part B. Interrupts and GPIO

For this question you have to build a traffic light/pedestrian crossing controller. The traffic lights will tick through their normal sequence, however, the crosswalk button will trigger an interrupt that will cause the traffic light to turn red after a small delay. A finite state machine (FSM) of the traffic light sequence can be seen in Figure 2.

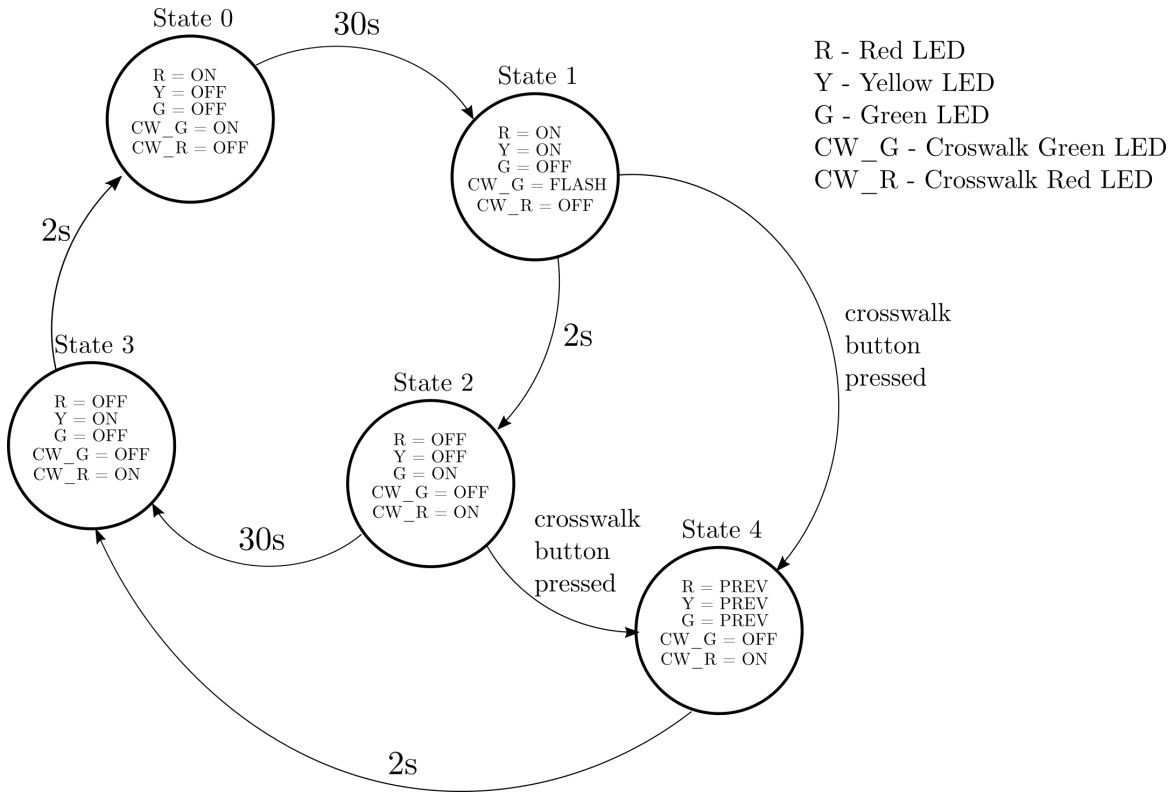

Figure 1: Finite State Machine (FSM) for the traffic light application

The micro-controller that we are programming has five LEDs connected for the traffic lights and crosswalk lights, and one button (active high). They are connected to the GPIO pins as follows:

Table 1: GPIO connections to LEDs and crosswalk button

GPIO Pin	**Connected**	**Direction**
0	Red LED	*Output*
9	Yellow LED	*Output*
10	Green LED	*Output*
13	Crosswalk Green LED	*Output*
14	Crosswalk Red LED	*Output*
17	Crosswalk Button	*Input*

The following 32-bit registers can be used to interact with the GPIO pins.

Table 2: GPIO Registers

Addr	**Name**	**R/W**	**Info**
`0x30001000`	GPIO_CONFIG	R/W	Sets direction of each GPIO. 1=output; 0=input
`0x30001004`	GPIO_WRITE	W	Sets the value of each GPIO pin
`0x30001008`	GPIO_READ	R	Reads the value of each GPIO pin

Each bit in each of these registers corresponds to each GPIO pin, i.e. bit 5 corresponds to GPIO 5. For example, if we wanted to set the value of pin 22, we would first configure that pin to be an output with `GPIO_CONFIG`, by setting bit 22, and then write to bit 22 of `GPIO_WRITE` to set it's value.

```
// Incomplete traffic light code
// please complete this in the questions below
// ( Feel free to use enums for the state variable.
// however, I wont penalise you if you don't. )

 void IRAM_ATTR crosswalk_button() {
       // ISR body
 }

 void setup() {
     // calls the ISR crosswalk_button when pin 5 goes high
     attachInterrupt(5, crosswalk_button, RISING);
     state = 0;
 }

 unsigned int state = 0;

 void loop() {

     if(state == 0) {

         delay(30000); // delay(N) delays execution by N milliseconds
         state = 1;
         return;
     }

     if(state == 1) {

         delay(2000);
         state = 2;
         return;
     }

     if(state == 2) {

         delay(30000);
         state = 3;
         return;
     }

    if(state == 3) {

         delay(2000);
         state = 0;
         return;
    }

    // ERROR we are in an undefined state
    // FLASH ALL THE LIGHTS
 }
```

5. Complete the code above to implement the traffic light **without** the crosswalk button. The LEDs must correctly change state according to the FSM in Figure 2, you may ignore state 4 for this question. You will need to use the memory-mapped registers to configure and control the GPIO pins, you are not permitted to use the Arduino `digitalWrite()` functions. In state 1, the green crosswalk LED should flash with a period of 250ms. If we are in an unknown state make all the LEDs flash forever with a period of 250ms.

[15 marks]

6. Add the functionality for the crosswalk button according to the FSM in Figure 2. When the crosswalk button is pressed the system should move into state 4 with minimal delay.

[10 marks]

Part C. Configuring a Hardware Timer

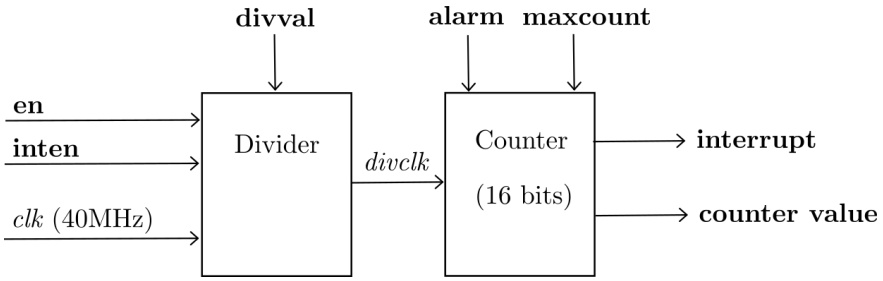

Figure 2: Block diagram of the hardware timer

Figure 3 shows a block diagram of the timer module within a µC (Microcontroller). *NOTE: this is not the same as the timer modules on the ESP32.*

- The **en** signal is used to enable or disable the entire module, when it is set to `1` the divided clock signal, *divclk*, is passed on to the counter submodule. Whenever this value changes the counter is also reset.
- The **inten** signal enables the generation of interrupts from the timer, when it is set to `1` interrupts are enabled when it is `0` they are disabled.
- The divider submodule divides the frequency of the input clock, *clk*, to produce a divided clock, *divclk*. The amount it divides by is set by the signal **divval**, this is an `8-bit` value meaning it can divide the input clock by any value between `1 - 255`. If **divval** is `0` then the input clock is divided by `1`.
- The counter counts every time *divclk* changes from `LOW` to `HIGH`. It is a `16-bit` counter that will count up to the `16-bit` value **maxcount**, where it then automatically starts again from `0`. The signal **counter value** is the current value of the counter.
- The **alarm** value causes the interrupt signal, **interrupt**, to fire whenever **counter value** is equal to **alarm** and **inten** is high.

Table 3: Table showing the memory-map for the timer module registers

Address	**Info**	**Bit functions**	**Read-Only(RO)/Read-Write(RW)**
`0xFFFF1000`	Divider Setup	Bit 31: sets **en** Bit 30: sets **inten** Bits 30-8: unused Bits 7-0: sets **divval**	RW
`0xFFFF1004`	Counter Setup	Bits 31 - 16: sets **alarm** Bits 15 - 0: sets **maxcount**	RW
`0xFFFF1008`	Counter Value	Bits 31 - 16: unused Bits 15 - 0: read **counter value**	RO

7. What is the highest resolution (smallest unit of time) that this timer can measure? Justify your answer.

[2 marks]

8. What is the maximum period of time that this timer can be used to time? Justify your answer.

[3 marks]

9. Write code that will configure the timer to generate an interrupt every 20 milliseconds. You should use pointers to do this.

[10 marks]

End of Paper