Part A

1. [**Conceptual understanding; Easy; 4 marks**]

   - Memory-mapped I/O is where hardware registers used to control hardware peripherals are mapped into the address space of the system. [**2 marks**]
   - Interacting with memory mapped I/O is typically achieved using pointers. [**2 marks**]

2. [**Conceptual understanding; Easy; 2 marks**]

   - One mark for any of the following:
     - Can run concurrently to the rest of the system.
     - Reduced power consumption.
     - Lower latency
     - Predictable latency
     - Computation can be executed in parallel.

     [**2 marks**]

3. [**Conceptual understanding; Moderate; 2 marks**]

   - One mark for one of the following:
     - Memory latency.
     - Operating system task scheduling.
     - Unpredictable I/O Latency
     - Interrupts

     [**1 mark**]
   - One mark for one of the following:
     - Embedded systems are often used in safety-critical applications where a task taking longer than expected can lead to missed deadlines with potentially fatal consequences.
     - For controlling systems it is typically essential to respond to events with predictable latency.

     [**1 mark**]

4. [**Conceptual understanding; Easy; 2 marks**]

   - One mark for any of the following:
     - Enter deep sleep states when the device is idle
     - Reduce the clock frequency of the device
     - Reduce sampling rate of ADC inputs
     - Disable unused modules
     - Use fixed-point arithmetic instead of floating-point
     - General code optimisations
     - Reduce the amount of DRAM

     [**2 marks**]

Part B

5. [**Coding; Moderate/Hard; 15 marks**]

- Example code: **[15 marks]**

```
void IRAM_ATTR crosswalk_button() {
    // ISR body
}

unsigned int * gpio_config = (unsigned int *)(0x30001000);
unsigned int * gpio_write = (unsigned int *)(0x30001000);
unsigned int * gpio_read = (unsigned int *)(0x30001000);

const int R_LED = 0;
const int Y_LED = 9;
const int G_LED = 10;
const int CWG_LED = 13;
const int CWR_LED = 14;

void setup() {
    // calls the ISR crosswalk_button when pin 5 goes high
    attachInterrupt(17, crosswalk_button, RISING);
    state = 0;

    // Set output GPIO pins
    *gpio_config = (1 << R_LED);
    *gpio_config = *gpio_config | (1 << Y_LED);
    *gpio_config = *gpio_config | (1 << G_LED);
    *gpio_config = *gpio_config | (1 << CWG_LED);
    *gpio_config = *gpio_config | (1 << CWR_LED);

    // alternatively
    // *gpio_config = 1 | (1 << 9) | (1 << 10) | (1 << 13) | (1 << 14);
}

unsigned int state = 0;

void loop() {

    if(state == 0) {

        *gpio_write = (1 << R_LED) | (1 << CWG_LED);

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

    if(state == 1) {

        for(int i=0; i<4; i++) {
            *gpio_write = (1 << R_LED) | (1 << Y_LED) | (1 << CWG_LED);
            delay(250);
            *gpio_write = (1 << R_LED) | (1 << Y_LED);
            delay(250);
```

```
        }

        state = 2;
        return;
    }

    if(state == 2) {

        *gpio_write = (1 << G_LED) | (1 << CWR_LED);

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

    if(state == 3) {

        *gpio_write = (1 << Y_LED) | (1 << CWR_LED);

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


    // ERROR we are in an undefined state
    // FLASH ALL THE LIGHTS
    while(true) {
        *gpio_write = 0;
        delay(250);
        *gpio_write = (1 << R_LED) | (1 << Y_LED) |
                      (1 << G_LED) | (1 << CWG_LED) | (1 << CWR_LED);
        delay(250);
    }
}
```


6. **[Coding; Hard; 10 marks]**

- Example code: **[10 marks]**

```
  unsigned int isr_fired = 0;

  void IRAM_ATTR crosswalk_button() {
    // ISR body
    if( (state == 1) || (state == 2) ) {
        isr_fired = 1;
    }
}

unsigned int * gpio_config = (unsigned int *)(0x30001000);
unsigned int * gpio_write = (unsigned int *)(0x30001000);
unsigned int * gpio_read = (unsigned int *)(0x30001000);

const int R_LED = 0;
const int Y_LED = 9;
```

```
const int G_LED = 10;
const int CWG_LED = 13;
const int CWR_LED = 14;

void setup() {
    // calls the ISR crosswalk_button when pin 5 goes high
    attachInterrupt(17, crosswalk_button, RISING);
    state = 0;

    // Set output GPIO pins
    *gpio_config = (1 << R_LED);
    *gpio_config = *gpio_config | (1 << Y_LED);
    *gpio_config = *gpio_config | (1 << G_LED);
    *gpio_config = *gpio_config | (1 << CWG_LED);
    *gpio_config = *gpio_config | (1 << CWR_LED);

    // alternatively
    // *gpio_config = 1 | (1 << 9) | (1 << 10) | (1 << 13) | (1 << 14);
}

unsigned int state = 0;

void mod_delay(unsigned int in) {
    unsigned int start = millis();
    while( ~isr_fired || (start + in) > millis() ) { }
}

void loop() {

    if(state == 0) {

        *gpio_write = (1 << R_LED) | (1 << CWG_LED);

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

    if(state == 1) {

        for(int i=0; i<4; i++) {
            *gpio_write = (1 << R_LED) | (1 << Y_LED) | (1 << CWG_LED);
            mod_delay(250);
            *gpio_write = (1 << R_LED) | (1 << Y_LED);
            mod_delay(250);
        }

        if(isr_fired) {
            state = 4;
            isr_fired = 0;
        } else {
            state = 2;
        }
```

```
        return;
    }

    if(state == 2) {

        *gpio_write = (1 << G_LED) | (1 << CWR_LED);

        mod_delay(30000);

        if(isr_fired) {
            state = 4;
            isr_fired = 0;
        } else {
            state = 3;
        }
        return;
    }

    if(state == 3) {

        *gpio_write = (1 << Y_LED) | (1 << CWR_LED);

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

    if(state == 4) {
        *gpio_write = *gpio_read & ~(1 << CWG_LED) | (1 << CWR_LED);
        state = 3;
        return;
    }


    // ERROR we are in an undefined state
    // FLASH ALL THE LIGHTS
    while(true) {
        *gpio_write = 0;
        delay(250);
        *gpio_write = (1 << R_LED) | (1 << Y_LED) |
                      (1 << G_LED) | (1 << CWG_LED) | (1 << CWR_LED);
        delay(250);
    }
}
```


Part C

7. **[Conceptual understanding; Moderate; 3 marks]**

- The highest resolution the timer can have is when the clock is running at it's fastest rate, $40MHz$. In this case the counter will increment every clock period $\frac{1}{40*10^6} = 2.5*10^{-8}s = 25ns$. [**3 marks**]

8. **[Conceptual understanding; Moderate; 3 marks]**

- To get the maximum period of time that this timer can measure we need to divide the clock by the highest amount that we can.

In this case **dival** is an 8-bit number meaning the maximum value we can divide the clock by is $2^8 - 1 = 255$. Setting this divider value gives us a *divclk* of $\frac{40MHz}{255} = 156862.75Hz$.

The counter is a 16-bit counter meaning that it can count to a maximum of $2^16 - 1 = 65535$ cycles. Since our *divclk* has a period of $1/156862.75 = 6.374\mu s$ the maximum value that this timer can count to is $(6.374 * 10^{-6}) * 65535 = 0.42seconds$. **[3 marks]**

9. **[Coding; Hard; 10 marks]**

- Example code: **[10 marks]**

```
unsigned int* div_setup = (unsigned int*)(0xFFFF1000);
unsigned int* counter_setup = (unsigned int*)(0xFFFF1004);

void setup() {
        // counter increments every 2uS
        *counter_setup = 10000; // Set the maximum count to 10000

        // Set the alarm to be at the maximum counter value
        *counter_setup = *counter_setup | (10000 << 16);

        // Setup the clock divider and enable
        *div_setup = (1 << 31); // enables the timer
        *div_setup = *div_setup | (1 << 30); // enables the interrupt
        *div_setup = *div_setup | 80; // Divides the clock by 80
}
```


End of Paper