COMPENG 4DS4 Project 1 Report

Aaron Pinto	Raeed Hassan	Jingming Liu	Jeffrey Guo
pintoa9	hassam41	liuj171	guoj69

March 17, 2023

Declaration of Contributions

Task	Contributions
Motor	Raeed, Aaron, Jingming
Position	Raeed, Jingming
LED	Raeed, Jeffrey
RC	Raeed, Aaron, Jingming

Motor Task

The motor task controls the DC motor's speed by calculating the PWM duty cycle and controlling the FTM module. The task will wait for and receive messages from the motor queue, and will convert the values received on the queue to the appropriate PWM duty cycle for the FTM module. The motor queue can contain one single byte integer, and the value sent into the queue from the RC task will be between -100 and 100. When the motor task receives an integer from the motor queue, it will compare the motor value with the previous motor value, and if the value has changed it will convert the value to its corresponding PWM duty cycle, and update the FTM module with this duty cycle. We only trigger the FTM module when the motor value has changed as we experienced jittering with the motor when the task triggers the FTM module every time when the motor value has not changed.

Listing 1: Motor Task

```
void motorTask(void *pvParameters) {
    // Motor task implementation
    BaseType_t status;
    int motorInput;
    float motorDutyCycle;

    while (1) {
        status = xQueueReceive(motor_queue, (void*) &motorInput, portMAX_DELAY);

        if (status != pdPASS) {
            PRINTF("Queue Receive failed!.\r\n");

            while (1)
                ;
        }

        if (prevMotorInput != motorInput) {
            prevMotorInput = motorInput;
            motorDutyCycle = motorInput * 0.025f / 100.0f + 0.0760;
            updatePWM_dutyCycle(FTM_CHANNEL_DC_MOTOR, motorDutyCycle);
            FTM_SetSoftwareTrigger(FTM_MOTOR, true);
        }

        vTaskDelay(1 / portTICK_PERIOD_MS);
    }
}
```

Position Task

The position task controls the servo motor's angle by calculating the PWM duty cycle and controlling the FTM module. The task will wait for and receive messages from the angle queue, and will convert the values received on the queue to the appropriate PWM duty cycle for the FTM module. The angle queue can contain one single byte integer, and the value sent into the queue from the RC task will be between -45 and 45. When the position task receives an integer from the angle queue, it will compare the servo angle value with the previous servo input value, and if the value has changed it will convert the value to its corresponding PWM duty cycle, and update the FTM module with this duty cycle. We only trigger the FTM module when the servo angle value has changed as we experienced jittering with the servo motor when the task triggers the FTM module every time even when the

servo angle value has not changed.

Listing 2: Position Task

```
void positionTask(void *pvParameters) {
    // Position task implementation
    BaseType_t status;
    int servoInput;
    float servoDutyCycle;

    while (1) {
        status = xQueueReceive(angle_queue, (void*) &servoInput, portMAX_DELAY);

        if (status != pdPASS) {
            PRINTF("Queue Receive failed!.\r\n");

            while (1)
                ;
        }

        if (prevServoInput != servoInput) {
            prevServoInput = servoInput;
            servoDutyCycle = servoInput * 0.025f / 45.0f + 0.078;
            updatePWM_dutyCycle(FTM_CHANNEL_SERVO_MOTOR, servoDutyCycle);
            FTM_SetSoftwareTrigger(FTM_MOTOR, true);
        }

        vTaskDelay(1 / portTICK_PERIOD_MS);
    }
}
```

LED Task

Listing 3: LED Task

```
void ledTask(void *pvParameters) {
    // LED task implementation
    BaseType_t status;
    float red;
    float green;
    float blue;

    uint8_t led_input[3];

    while (1) {
        status = xQueueReceive(led_queue, (void*) &led_input, portMAX_DELAY);

        if (status != pdPASS) {
            PRINTF("Queue Receive failed!.\r\n");

            while (1)
                ;
        }

        red = (led_input[0] / 255.0) * 100;
        green = (led_input[1] / 255.0) * 100;
        blue = (led_input[2] / 255.0) * 100;

        FTM_UpdatePwmDutycycle(FTM_LED, FTM_RED_CHANNEL, kFTM_EdgeAlignedPwm, (uint8_t) red)
            ;
        FTM_SetSoftwareTrigger(FTM_LED, true);

        FTM_UpdatePwmDutycycle(FTM_LED, FTM_GREEN_CHANNEL, kFTM_EdgeAlignedPwm, (uint8_t)
            green);
        FTM_SetSoftwareTrigger(FTM_LED, true);
```

```
29
30          FTM_UpdatePwmDutycycle(FTM_LED, FTM_BLUE_CHANNEL, kFTM_EdgeAlignedPwm, (uint8_t)
                blue);
31          FTM_SetSoftwareTrigger(FTM_LED, true);
32
33          vTaskDelay(1 / portTICK_PERIOD_MS);
34      }
35  }
```

RC Task

Listing 4: RC Task

```
1   void rcTask(void *pvParameters) {
2       // RC task implementation
3       BaseType_t status;
4
5       RC_Values rc_values;
6       uint8_t *ptr = (uint8_t*) &rc_values;
7
8       int motor_value;
9       int angle_value;
10      uint8_t led_value[3];
11
12      while (1) {
13          UART_ReadBlocking(RC_UART, ptr, 1);
14          if (*ptr != 0x20)
15              continue;
16          UART_ReadBlocking(RC_UART, &ptr[1], sizeof(rc_values) - 1);
17          if (rc_values.header == 0x4020) {
18
19              if (rc_values.ch8 == 1500) {
20                  motor_value = 0;
21              } else {
22                  switch (rc_values.ch6) {
23                      case 1000:
24                          motor_value = rc_values.ch8 == 1000 ? (rc_values.ch2 - 1000) / 50 :
                                (rc_values.ch2 - 1000) / -50;
25                          led_value[0] = 255;
26                          led_value[1] = 0;
27                          led_value[2] = 0;
28                          break;
29                      case 1500:
30                          motor_value = rc_values.ch8 == 1000 ? (rc_values.ch2 - 1000) / 20 :
                                (rc_values.ch2 - 1000) / -20;
31                          led_value[0] = 255;
32                          led_value[1] = 255;
33                          led_value[2] = 0;
34                          break;
35                      case 2000:
36                          motor_value = rc_values.ch8 == 1000 ? (rc_values.ch2 - 1000) / 10 :
                                (rc_values.ch2 - 1000) / -10;
37                          led_value[0] = 0;
38                          led_value[1] = 255;
39                          led_value[2] = 0;
40                          break;
41                  }
42              }
43
44              angle_value = (rc_values.ch4 - 1500) * 45 / 500;
45
46              status = xQueueSend(motor_queue, (void* ) &motor_value, portMAX_DELAY);
47              if (status != pdPASS) {
48                  PRINTF("Queue Send failed!.\r\n");
49                  while (1)
50                      ;
```

```
            }

            status = xQueueSend(angle_queue, (void* ) &angle_value, portMAX_DELAY);
            if (status != pdPASS) {
                PRINTF("Queue Send failed!.\r\n");
                while (1)
                    ;
            }

            status = xQueueSend(led_queue, (void* ) &led_value, portMAX_DELAY);
            if (status != pdPASS) {
                PRINTF("Queue Send failed!.\r\n");
                while (1)
                    ;
            }

            vTaskDelay(1 / portTICK_PERIOD_MS);
        }
    }
}
```