UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

ECE342 – Computer Hardware
Midterm Examination

February 13, 2023
Duration: 100 minutes

Examiner: Prof. B. Korst

Exam Type X: Open Book and open notes. **Calculator Type 1:** Any type allowed

Enter your name below **EXACTLY** as it appears on Quercus.

Exam guidelines:

(a) This booklet consists of **FOUR** questions. You must answer all of them.
(b) The exam booklet consists of **8** pages.
(c) You are permitted to consult your own notes, including those you have saved on the ECF `W:` drive.
(d) No communication with another person either in the room or electronically is permitted.
(e) Short, clear and to the point answers will get more marks than long, unclear answers.
(f) You do not need to use all the space provided for each question. Filling up space because it is there will result in 0 marks.
(g) If you make a mistake, clearly cross out your incorrect version and indicate which version you would like marked. If this is unclear, the first answer will be marked.
(h) For all questions in this booklet, you must provide answers for the `ECE342 microcontroller`.
(i) For all questions in this booklet, the most efficient answers will get maximum marks. Please think about the best solution for each question before answering.

Question 1 [4 Marks]

In this question, you must use the ECE342 microcontroller to control a robot that moves in one dimension. Sensor S produces an analog voltage between $0-5V$ representing the distance to the nearest obstacle. $0V$ means the object is $0m$ away and 5V means the robot is 10 meters away. You do not need to consider objects further than $10m$. The relationship between distance and voltage is linear.

Motor M is controlled by an analog voltage between $0-5V$ representing the speed at which the robot moves. $0V$ means the robot is moving at $-5m/s$ and $5V$ means the robot moves at $+5m/s$. The relationship between voltage and speed is linear.

(a) [1 Mark] If sensor S is connected to ADC1, what is the smallest interval in meters that you can detect.

$(10m - 0m)/2^{10} = 0.00977m$

(b) [1 Mark] If motor M is connected to DAC1, what is the resolution in m/s that you can control.

$(5m/s - (-5m/s))/2^{12} = 0.00244m/s$

(c) [1 Mark] When the robot is moving at maximum speed (5m/s) towards an obstacle, what is the distance travelled between two consecutive ADC samples?

$1/1KHz \times 5m/s = 0.005m$

(d) [1 Mark] To save battery, we want to lower the ADC sampling frequency. Assuming that the robot can go from 5m/s to 0m/s instantly, what is the lowest sampling frequency in samples per second that guarantees it will not crash into an obstacle that suddenly appears 2m in front of the robot?

$Time = 2m/(5m/s) = 0.4s$
$Frequency = 1/0.4s = 2.5Hz$

Question 2 [6 Marks]

In this question, you must write code to adjust the temperate of a liquid stored in a rectangular container. The container is equipped with four sensors. Three sensors measure the length, width, and height of the container. You also need to know the liquid's *volumetric heat capacity (k)*, which is the amount of energy (in Joules) to add to each cubic meter of the solution. The fourth sensors provides you with this k value. Therefore, the energy to be added is calculated as: $energy = k \times length \times width \times height$

The Table below lists these sensors and the format in which they produce output.

	Purpose	Output format
Sensor A	Measures height (m)	UQ2.6
Sensor B	Measures length (m)	UQ4.4
Sensor C	Measures width (m)	UQ0.4
Sensor D	Specifies the amount of energy to add (J)	UQ6.1

NOTE: You should not declare any additional variables in your code for this question.

(a) [3 Marks] In this part, the box is heated using a heating element which you can control using the `AddHeat` function. This function accepts the `EnergyToAdd` parameter in $UQ16.16$ format. Complete the code below to calculate the amount of energy to be added.

```
#include "main.h"
#define SensorA_Address         0x80000000
#define SensorB_Address         0x80000004
#define SensorC_Address         0x80000008
#define SensorD_Address         0x8000000C
#define HeatingElementAddress   0x80000010

void fillContainer() {
    uint32_t SensorAVal = Read_Sensor(SensorA_Address);
    uint32_t SensorBVal = Read_Sensor(SensorB_Address);
    uint32_t SensorCVal = Read_Sensor(SensorC_Address);
    uint32_t SensorDVal = Read_Sensor(SensorD_Address);
    uint32_t EnergyToAdd;

    // SOLUTION:
    // The result is in UQ12.15 Format (2+4+0+6=12 and 6+4+4+1=15)
    EnergyToAdd = SensorAVal * SensorBVal * SensorCVal * SensorDVal

    // We need to left shift by 1 to make this UQ16.16.
    EnergyToAdd  = EnergyToAdd << 1


    AddHeat(HeatingElementAddress, EnergyToAdd);
}
```

(b) [3 Marks] In part A, the heating element is only able to **add** energy to the liquid. In this part, the temperature is adjusted using a *Peltier* element, which can **add** or **subtract** energy from the liquid. To enable this, two changes have been made to the setup:
1) Sensor D now provides its output in $Q6.1$ format.
2) The `AddHeat` function now accepts input in $Q15.16$ format.
The format of the other sensor outputs is unchanged from part A.

Complete the program shown below, using the new formats for Sensor D and `AddHeat`. Think carefully about how changing the number format for Sensor D will change your code.

```
#include "main.h"
#define SensorA_Address  0x80000000
#define SensorB_Address  0x80000004
#define SensorC_Address  0x80000008
#define SensorD_Address  0x8000000C
#define PeltierAddress   0x80000014

void fillContainer() {
    uint32_t SensorAVal = Read_Sensor(SensorA_Address);
    uint32_t SensorBVal = Read_Sensor(SensorB_Address);
    uint32_t SensorCVal = Read_Sensor(SensorC_Address);
    uint32_t SensorDVal = Read_Sensor(SensorD_Address);
    uint32_t EnergyToChange;

    // SOLUTION:

    // Sign Extend SensorDVal from 8-bits to 32-bits.
    If ((Sensor_D_Val & 0x40)!=0) //7th bit is a 1
        Sensor_D_Val = Sensor_D_Val | FFFFFF80; //Fill rest of bits with 1

    //This is in Q15.15 Format (3+5+1+7=16 and 6+4+4+1=15) so no overflow
    EnergyToAdd = SensorAVal * SensorBVal * SensorCVal * SensorDVal

    //This converts it to Q15.16 Format
    EnergyToAdd  = EnergyToAdd << 1









    AddHeat(PeltierAddress, EnergyToChange);
}
```

Question 3 [10 Marks]

In this question, you must write code to calculate the average of 100 values, sampled from ADC1. You must then output the average of these 100 values on DAC1. On the `ECE342 Microcontroller` platform, ADC1 triggers an interrupt when a new sample is ready.

(a) [5 Marks] Complete the following code to implement the functionality described above.

```
#include ``main.h''

int ADC_Values[100];
int n_samples = 0;
void ADC1_InterruptHandler(void) {
    ADC_Values[n_samples++] = ADC1_ReadValue();
    ADC1_ClearInterrupt();
}
int main(void) {
    System_Init();

    int avg;

    DAC1_Init();
    DAC1_WriteValue(0);

    while(1) {
        if (n_samples == 100) {
            ADC1_StopInterrupt();
            avg = 0;
            for (int i = 0; i < 100; i++) {
                // ADC_Values are 10-bit vals so no overflow
                avg += ADC_Values[i];
            }
            avg /= 100;
            DAC1_WriteValue(avg);
            n_samples = 0;
            ADC1_StartInterrupt();
        }
    }
}
```

(b) [5 Marks] In part A, you are storing 10-bit values in 32-bit memory locations, which wastes 22 bits. You must now improve efficiency by storing one ADC value in bits[31:16] and a second ADC value in bits[15:0] in each 32-bit memory location. You must still calculate and output the average of 100 values on DAC1. **NOTE:** Think carefully about computing the average when two 10-bit values are stored together. You must still output just a single average value via the DAC at the end.

```
#include "main.h"
int ADC_Values[50];
int n_samples = 0;

void ADC1_InterruptHandler(void) {
    if (n_samples % 2 == 0) {
        ADC_Values[n_samples / 2] = ADC1_ReadValue() & 0x00FF;
    }
    else {
        ADC_Values[n_samples / 2] |= ADC1_ReadValue() << 16;
    }
    n_samples++;
}

int main(void) {
    System_Init();

    short int avg;

    DAC1_Init();
    DAC1_WriteValue(0);

    while(1) {
        if (n_samples == 100) {
            ADC1_StopInterrupt();
            avg = 0;
            for (int i = 0; i < 50; i++) {
                short int v1, v2;
                v1 = ADC_Values[i];
                v2 = ADC_Values[i] >> 16;
                avg += (v1 + v2) / 100;
            }
            DAC1_WriteValue(avg);
            n_samples = 0;
            ADC1_StartInterrupt();
        }
    }
}
```

Question 4 [10 Marks]

You must write code to control a system consisting of a: 1) ECE342 microcontroller, 2) an infrared camera (IC) and 3) an EEPROM. Specifically, your code must copy one captured image from the IC to the EEPROM. All three devices are connected on the same I2C bus.

The IC captures frames of up to 256×256 8-bit pixels and stores the image in an internal memory. The image is captured in row-major order, similar to how 2D arrays are stored in C. The register map for the IC is shown to the right.

To use the camera, you must do the following:

(a) Write a '1' to the `Capture` register to capture an image.
(b) **After 10ms**, the captured image is saved to internal memory and stored until `Capture` is set to '1' again.
(c) You can then read the image pixel-by-pixel by setting the `Row` and `Column` registers. The IC will then put the value at that pixel in the `PixelValue` register. You must do this for every pixel in the captured image.

Address	8-bit Registers
0x0	Capture
0x8	Row
0x10	Column
0x18	PixelValue

NOTE: The `PixelValue` register will be set in 1 cycle after setting the row and column, so no extra delay is needed between setting the registers and reading `PixelValue`.

Assume you have the following `main` function provided to you.

```
#include ''main.h''
#define ADDR_IC         (uint8_t)0x12
#define ADDR_EEPROM     (uint8_t)0x14
#define IC_CAPTURE_REG  (uint8_t)0x00
#define IC_ROW_REG      (uint8_t)0x08
#define IC_COL_REG      (uint8_t)0x10
#define IC_PIXEL_REG    (uint8_t)0x18

uint8_t FrameData[256][256];

int main(void){
    System_Init();
    I2C1_Init();

    while(1){
        CaptureImage(uint8_t * FrameData);
        SaveImageToEEPROM(uint8_t * FrameData);
        Delay(1000); // Delay until the next image must be captured
    }
}
```

(a) [5 Marks] Implement the `CaptureImage()` function to capture one 256×256 image from the IC and store it in the `FrameData` array. **NOTE:** The I2C has a limit of 32 bytes in a single transfer. Fill in the body of the function given below.

```
void CaptureImage(uint8_t *FrameData) {
    I2C_Write(ADDR_IC, 0x0, 1, 1);
    Delay(10);
    I2C_Write(ADDR_IC, 0x0, 1, 0);
    for (int i = 0; i < 256; i++) {
        for (int j = 0; j < 256; j++) {
            I2C_Write(ADDR_IC, 0x8, 1, &i);
            I2C_Write(ADDR_IC, 0x10, 1, &j);
            I2C_Read(ADDR_IC, 0x18, 1, &FrameData[i][j]);
        }
    }
}




}
```

(b) [5 Marks] Implement the `SaveImageToEEPROM()` function to copy the image in the `FrameData` array to the EEPROM. Similar to Lab 4, the EEPROM only has memory locations. You can write to each location in this memory by providing the appropriate address. You should save the image in the EEPROM using the same row-major order, starting at memory address `0x0000`. Fill in the body of the function given below.

```
void SaveImageToEEPROM(uint8_t * FrameData) {
    for (int i = 0; i < 256; i++) {
        for (int j = 0; j < 256; j++) {
            I2C_Write(ADDR_EEPROM, i*256 + j, 1, &FrameData[i][j]);
        }
    }
}
```