Question 1: General Output Compare questions (10 pts)

a) For Normal Mode, the two CTC Modes, and the five fast PWM Modes, explain how TOP is set, and how TOP, the OCR registers and TCNT work together to generate waveforms. (6pts)

i) Normal Mode (2pts)

TOP: For Normal Mode TOP is a constant (i.e. 0xFFFF). If cannot be set.

How a waveform is generated: Each time TCNT == OCR:
- the signal is set to a value (0,1,toggle)
- an interrupt is called, whose primary task is to set the OCR to the next time at which the signal should change
- when TCNT reaches TOP it resets to 0.

ii) CTC modes (2pts)

TOP: For CTC Mode TOP is set using OCRA (for mode 4), or ICR (form mode 12)

How a waveform is generated:

Option 1: Same as Normal Mode, except you can program TOP

Option 2: Generating a square wave (i.e. clock wave)
- TOP is set to determine the period (measured in clocks) of the square wave (more correctly half the period)
- when TCNT == OCR, the signal is toggled
- TCNT resets to 0 after reaching TOP
- Note: no interrupts required
iii) Fast PWM modes (2pts)

TOP: For Fast PWM modes 5, 6, 7, TOP is a constant (0xFF, 0x1FF, 0x3FF), thus cannot be programmed. For Fast PWM mode 14 the ICR is used to set TOP, and for mode 15 the OCRA is used to set top.

How a waveform is generated:

Option 1: Same as Normal Mode, except you can program TOP, and you can have an event occur (i.e. the signal can change) when TCNT == TOP (more correctly when TCNT resets to 0).

Option 2: Generating a PWM wave
a) TOP determines the period of the square wave (measured in clocks)
b) OCR (for the appropriate signal) is set to determine the pulse width (measured in clocks)
c) When TCNT = OCR the signal will get set or cleared (depending on if in inverting or non-inverting mode)
d) When TCNT = TOP (more correctly 0) the signal will get set or cleared (depending if in inverting or non-inverting mode)
e) Note: no interrupts required

b) In lecture it is mentioned that the Period of a square wave (using CTC mode) or PWM wave (using Fast PWM mode) is: Square Wave Period = (2*TOP) + 1, PWM Period = TOP + 1. Why do these equations have + 1? (1pts)

The plus 1 is to account for the time (i.e. 1 clock cycle) it take TCNT to reset back to 0 after reaching TOP.
c) More using the Datasheet:

i) Explain the meaning of the “Update of OCRnX at” column of the “Waveform Generation Mode Bit Description” Table. (1 pts)

This column indicates when a new value that has been assigned to the OCR register will take effect (i.e. when it will actually be updated). Immediate means as soon as a new value is provided, and BOTTOM or TOP indicates that the value of OCR will not update until the Timer becomes equal to BOTTOM or TOP.

ii) For the Fast PWM mode rows, why is having an entry of BOTTOM in the “Update of OCRnX at” column instead of Immediate important? (1 pts)

This is important so the PWM does not “glitch” while generating a PWM wave. In other words, a new value of OCR will not be allowed until the current PWM period completes. Any new value is “buffered” until the Timer becomes equal to BOTTOM.

iii) For the Normal node row, why is having an entry of Immediate in the “Update of OCRnX at” instead of TOP or BOTTOM important? (1 pts)

Since in Normal mode you can generate any arbitrary waveform and since TOP is a constant, it is important that OCR can be updated at any time (i.e. immediately when a new value arrives). If this was not the case, then you would only be able to change the output signal once per TCNT counting form 0 to TOP.
Question 2: Don’t Go into the Light! (16 pts)

Complete the program below to have a robot move away from a light source. The robot has two wheels, similar to the robot used in lab, and has a light sensor on each side. See figure.

**Motor Control:** Assume that you are programming Timer/Counter 1 (for left motor) and Timer/Counter 3 (for right motor) of an ATMega128 to generate fast PWM waveforms to control the speed of each wheel’s motor. The speed of the motor is proportional to the percentage of time the PWM signal is high (i.e. PWM duty cycle).

**Note:** You must use 9-bit fast PWM mode, and use a TIMER prescaler of 8.

**Light Sensors:** The light sensors are connected to Channel 2 (left sensor) and 4 (right sensor) of the ATMEGA128 ADC as single channel inputs (i.e. not differential)

**Robot behavior:** The Robot should move away from the light in the following way.
- Speed of left motor = Intensity of left sensor / (Intensity of left sensor + Intensity of right sensor)
- Speed of right motor = Intensity of right sensor / (Intensity of left sensor + Intensity of right sensor)
a. Initialize TIMER1 and TIMER3 to meet the above requirements and so that both wheels initially move at 50% their maximum speed. (3pts)

```c
void init_TIMER1_TIMER3()
{

    //Set up TIMER1 and TIMER3 for 9-bit Fast PWM
    //Choosing to use channel B for both Timers
    //Bits 7-2: COMA1 COMA0 COMB1 COMB0 COMC1 COMC0 = 001000 (Non inv)
    // Bits 1-0: WGM1 WGM0 = 10 (9-bit Fast PWM Mode)
    TCCR1A = 0b00100010;
    TCCR3A = 0b00100010

    // bits 7: ICNC = 0 (not doing input capture)
    // bits 6: ICES = 0 (not doing input capture)
    // bits 5: Reserved
    // bits 4-3: WGM3 WGM2 = 01 (Want 9-bit Fast PWM MODE = 0110)
    // bits 2-0: CS2 CS1 CS0 = 010 (prescale 8 give 2MHz tick rate)
    TCCR1B = 0b00001010;
    TCCR3B = 0b00001010;

    //Set for 50% max speed which corresponds to 50% duty cycle
    // 9-bit Fast PWM has a period 512 so set pulse width to
    // 256 or 255
    OCR1B = 255;
    OCR3B = 255;

    Enable Channel B pins for output on Timer 1 and 3:
    DDRB |= 01000000; // Wire 6 of Port B (Timer 1 OC1B)
    DDRE |= 00010000; // Wire 4 of Port E (Timer 3 OC3B)
}
```

b. Initialize the ADC to meet the specification above, and meets the DataSheet requirements for ADC (3 pts)

```c
void init_ADC()
{
    // Enable ADC, set prescaler to 128 -> 16MHz / 128 = 125 KHz
    ADCSRA = 0b10000111
    // Chose Internal Ref and to initialize to channel 4
    ADMUX = 11000100;
}
```
c. Complete the following API function to read in the light sensor values. Use single shot mode and polling. (3 pts)

```c
void get_sensor_reading(int *left_sensor, int *right_sensor)
{
    // Set to channel 2
    ADMUX = 11000010;

    // Initiate an ADC conversion
    ADCSRA |= 0x40; // or 0b01000000

    // Wait for the conversion to finish
    while(ADCSRA & 0x40)
    {}

    // Read the value
    *left_sensor = ADC;

    // Set to channel 4
    ADMUX = 11000100;

    // Initiate an ADC conversion
    ADCSRA |= 0x40; // or 0b01000000

    // Wait for the conversion to finish
    while(ADCSRA & 0x40)
    {}

    // Read the value
    *right_sensor = ADC;
}
```
d. Complete the following API function to set the speed of each motor. The inputs should be specified on a 100-point scale (e.g. 50 means 50% speed). Assume the input parameters are no less than 1 and no greater than 99. Also rounding errors are acceptable (i.e. do NOT use floating-point calculation) (5pts)

```c
void set_motor_speed(int left_motor, int right_motor)
{
    unsigned long temp_left = left_motor;
    unsigned long temp_right = right_motor;

    OCR1B = (temp_left * 512)/100;
    OCR3B = (temp_right * 512)/100;
}
```

Note: Technically there should be a -1
Note: Do not deduct for not have unsigned long type

e. Complete main() (2pts)

```c
// Don’t go into the light program
main()
{
    int left_sensor;
    int right_sensor;
    int left_motor;
    int right_motor;

    init_TIMER1_TIMER3();
    init_ADC();

    while(1)
    {
        get_sensor_reading(&left_sensor, &right_sensor);

        // Computed motor speed commands
        left_motor = (100 * (unsigned long) left_sensor) / (left_sensor + right_sensor);
        right_motor = (100 * (unsigned long) right_sensor) / (left_sensor + right_sensor);

        Note: Do not deduct for not have unsigned long type

        set_motor_speed(left_motor, right_motor);
    }
}
```
**Question 3: Square Waves (10 pts)**

a) For TIMER1 using CTC **mode 12**, generate a square wave with a **10 ms** period on **channel B**, using a **prescaler of 64**. Assume the Date Direction Register has already been set for you. (5 pts)

```c
void init_TIMER1()
{
    TCCR1A = 0b00010000; // Channel B toggle mode
    TCCR1B = 0b00011011; // Mode 12 and 64 prescaler
    ICR1 = 1250; // Set Top to 5 ms = 1250 * .004 ms per tick
    OCRB = 1250; // OCRB just needs to be less than or equal to TOP
}
```

b) Now assume you have to use Normal Mode with a prescaler of 1 to generate a square wave, and the time to setup and execute the code in an ISR takes 20 µs. What CPU utilize (i.e. percent of the CPU time) would be spent handing interrupts for: (3 pts)

i) Generating a square wave with a 10ms period

Overhead = 2*20us / 10ms ; // Two interrupts per period.
    = 40*10^-6 / 10*10^-3
    = .004 = .4%

ii) Generating a square wave with 100 µs period

Overhead = 2*20us / 100us ; // Two interrupts per period.
    = 40*10^-6 / 100*10^-6
    = .4 = 40%

iii) Generating a square wave with a 50 µs period.

Overhead = 2*20us / 100us ; // Two interrupts per period.
    = 40*10^-6 / 50*10^-6
    = .8 = 80%

c) **What is the key trade-off between using CTC vs. Normal Mode for generating a square wave?** (2 pts)

The key tradeoff is that Normal mode incurs Interrupt overhead while, CTC mode does not for generating a square wave.
Question 4: Software implemented Input Capture (10 pts)

a) Assume the ATMEGA128 does not have Input Capture hardware or Interrupts. Write a C program to save TIMER1’s count value (i.e. TCNT1) when a positive edge event occurs on PortD, pin4. Note: you may want to do 4c first. (4 pts)

```c
int main(void)
{
    unsigned int rising_edge_time;

    // For grading do not mark off for not having this
    DDRD &= 0b11101111; // set wire 4 of PortD for input

    while(1)
    {
        // Check if the current value of wire 4 is 0
        if( ( PIND & 0b00010000) == 0)
        {
            // wait for wire 4 to become a 1
            while( !(PIND & 0b00010000))
            {
            }
            rising_edge_time = TCNT1; // Store counter value
        }
    
    return 0;
}
```

b) Describe two disadvantages of your software-implemented input capture program, as compared to using Input Capture hardware with Interrupts. (3 pts)

1. Does not allow other parts of the code to run, while it is running.

2. With the "software only" approach an interrupt could occur after detecting a positive edge, but before the TIMER value is read. Thus adding to the error of when the positive edge occurred.

3. The Input Capture hardware stores the value of TCNT as soon as the event occurs, while the software implementation will have error related to where in the polling loop the code is when the positive edge occurs.
c) Complete the bubble diagram below to implement a Mealy State Machine that implements a positive edge detector (i.e. the state machine outputs 1 each time a positive edge occurs on the input). Assume at Start the input to the state machine is initially 0 (3 pts)
Question 5: Edge Detection (15 pts)
The following simple block diagram illustrates how one may implement an Input Capture circuit.

An example timing diagram for capturing the rising edge of Input_signal is given below. In summary, on the occurrence of a positive edge on Input_signal: 1) Write_EN pulses ‘1’ for one sys_clk cycle, and 2) the value in the Timer/Counter Register (TCNT) is loaded into the Input Capture Register (ICR).
Assume: Any time Input_signal transitions is will hold its value for at least 1 sys_clk cycle.
a) Draw out a digital circuit to implement an Edge Detector that creates a 1-sys_clk-wide pulse on Write_EN to load the TCNT register into the ICR register when a rising edge occurs on Input_signal. The only components you can use are D-Flip Flops, and AND, OR, NOT gates (7 pts)

b) Repeat a) for an edge detector that creates a Write_EN pulse for detecting the falling edge of Input_signal. (3 pts)
c) The ATMEGA128 allows for detecting either the positive edge or negative edge of an input by configuring the “Edge Select” bit of a configuration register. Draw the digital circuit to allow the Edge Detector to detect a positive edge when ES=1, and a negative edge when ES=0. You may now use multiplexers in addition to D-Flip Flops, and AND, OR, NOT gates (5 pts)