1 - General purpose I/O [40 pts]

In this question, you are going to use the DDR, PORT, and PIN registers from your AVR device (the register definitions are the same as your AVR device). Assume that your AVR has:

- 2 pushbuttons connected to pins 5 (Button 1) and 6 (Button 2) of Port C,
- 2 LEDs connected to pin 2 (LED 1) and 4 (LED 2) of Port D.

Write a **complete** C program, using the skeleton code below, that:

- Runs continuously,
- Sets the pushbuttons as pull-up inputs,
- Turns on LED 1 **when only** Button 1 is pressed,
- Turns on LED 2 **when only** Button 2 is pressed,
- Turns off both LEDs if both buttons are pressed.

You have to implement two separate parts (shown by comments in bold font):

- Initialization of ports and pins (input vs. output, pull-up enable, etc.),
- Main loop that runs continuously and controls the LEDs.

Note: You can ignore button bounce effects.
Note: You should **not** make any assumptions about the initial values of the PORT/PIN/DDR bits.

Pins 5/6 of Port C should be set as pull-up inputs, and Pins 2/4 of Port D should be set as output.

```
#include <avr/io.h>

int main(){

     //Initialization of ports and pins

DDRC &= ~(1<<5) & ~ (1<<6);         //Pins 5 and 6 of Port C are input
PORTC |= (1<<5) | (1<<6);           //Pins 5 and 6 of Port C are pull-up
DDRD  |= (1<<2) | (1<<4);           //Pins 2 and 4 of Port D are output

     while(1){

     //Main loop that runs continuously

         if (!(PINC & (1<<5)) && (PINC & (1<<6))){
         //Check if Button 1 is pressed AND Button 2 is not pressed
            PORTD |= (1<<2);        //Turn on LED 1
         }
         else if ((PINC & (1<<5)) && !(PINC & (1<<6))){
         //Check if Button 1 is not pressed AND Button 2 is pressed
            PORTD |= (1<<4);        //Turn on LED 2
         }

         else if (!(PINC & (1<<5)) && !(PINC & (1<<6))){
         //Check if Button 1 is pressed AND Button 2 is pressed
            PORTD &= ~(1<<2);       //Turn off LED 1
            PORTD &= ~(1<<4);       //Turn off LED 2
         }

     }
     return 0;
}
```

2 – Timer Settings [30 pts]

In this problem, you will create a PWM waveform with a specific duty cycle value using the microcontroller system, described in the file **"Datasheet_for_questions_2_and_3":**

- The microcontroller is an 8-bit system and its clock runs at **8MHz.**
- The timer is also 8-bit.
- The timer uses only the internal clock and there are 3 predefined pre-scale values:
  - 1: No pre-scaling
  - 8: System clock/8
  - 64: System clock/64
- There are two modes of the timer:
  - Normal Mode: The timer counts up to the MAX value and then restarts from the BOTTOM (where MAX=maximum value that an 8-bit counter can go up to and BOTTOM=0).
  - Clear on Compare (CTC) Mode: The timer counts up to the TOP value (defined by the OCRA register) and then restarts from the BOTTOM (=0).
- The registers in the timer unit are as follows:
  - TCNT: Keeps the count value of the timer.
  - OCRA: Keeps the output compare value A.
  - OCRB: Keeps the output compare value B.
  - TCR: Controls the modes and properties of the timer/counter. The layout of the individual bits and how they can be controlled are shown below:

TCR:

7th bit	6th bit	5th bit	4th bit	3rd bit	2nd bit	1st bit	0th bit
-	-	COMPB	COMPA	OVF	TM	PS1	PS0

- The first two bits, PS1 and PS0 are used for pre-scaling:
  - PS1 = 0, PS0 = 0: Timer is off.
  - PS1 = 0, PS0 = 1: No pre-scaling (i.e. pre-scale value = 1).
  - PS1 = 1, PS0 = 0: Pre-scale value = 8.
  - PS1 = 1, PS0 = 1: Pre-scale value = 64.
  - And setting a pre-scalar value starts the timer.
- TM bit is used to set the timer mode.
  - TM = 0: Normal mode.
  - TM = 1: CTC mode.
- OVF bit is used to see if the timer overflows. This bit is set to 1 whenever the TCNT value goes over MAX and becomes 0. This bit does not clear automatically, i.e. it needs to be cleared manually by setting it to 0.
- COMPA bit is used to see if the timer value matches the OCRA register value. If TCNT becomes equal to OCRA, COMPA value becomes 1. This bit does not clear automatically, i.e. it needs to be cleared manually by setting it to 0.
- COMPB bit is used to see if the timer value matches the OCRB register value. If TCNT becomes equal to OCRB, COMPB value becomes 1. This bit does not clear automatically, i.e. it needs to be cleared manually by setting it to 0.

How can you use this timer to create a PWM waveform with 2ms period and 50% duty cycle? Assume that the PWM waveform will be observed on an LED with the following sequence (similar to what we had in Lab 4):

Start→LED ON→Compare B occurs→LED OFF→Compare A occurs→Back to Start

Explain how you should set the timer mode, pre-scale value and Compare Register A/B values.

Since this is an 8-bit timer, the MAX value we can count up to is 256. The first thing we need to do is to set the timer for 2ms.

Assume that pre-scale value is 1. Then, the maximum value we can time is 1/(8MHz)*256 = 32us, which is not enough to time 2ms. So, we increase the pre-scale value to 8, and observe that the maximum value we can time becomes 8*32=256us, which is still not enough to time 2ms. We increase the pre-scale value to 64, and observe that the maximum value we can time becomes 8*256 = 2048us, which is bigger than 2ms.

Thus, this pre-scale value is enough. However, we need to scale the timer since 2048us is bigger than 2ms. When the pre-scale value is 64, 256 time units (maximum value we can count up to with an 8-bit timer) lead to 2048us, the question is, how many time units lead to 2ms → 256*(2ms/2048us) = 250 time units. Thus, to set the timer to 2ms, we need to set the timer:

- In CTC mode
- With pre-scale value 64
- With OCRA value 249 (250-1, since we start counting from 0).

Then, we need to use the OCRB register to create a 25% duty cycle. Since the timer will count 250 time units, to set a 50% duty cycle, we need to set the OCRB value to:
OCRB = (250*0.5) – 1 = 125 – 1 = 124

3 – Timer-based PWM Implementation [30 pts]

Use the following code skeleton to implement the PWM waveform in question 2 <u>with the timer module you have</u>.

- **You should implement the timer_init() function (sets up the timer parameters).**
- **You can assume that there are predefined variables for registers (TCR, TCNT, OCRA and OCRB).**
- **You should use the predefined statements (see the two #define statements) to turn on and off the LED.**

```
#define LEDON     (statement to turn on LED)
#define LEDOFF    (statement to turn off LED)


void timer_init(){

      OCRA = 249;             //set the OCRA value
      OCRB = 124;             //set the OCRB value
      TCR |= (1<<2);          //set the timer mode to CTC
      TCR |= (1<<0) | (1<<1); //set the pre-scaler value to 64
}

int main(){

      timer_init();

      while(1){

            LEDON;                    //Turn on the LED
            while(!(TCR & (1<<5))); //Wait until the COMPB flag is 1
            TCR &= ~(1<<5);         //Clear the COMPB flag
            LEDOFF;                   //Turn off the LED
            while(!(TCR & (1<<4))); //Wait until the COMPA flag is 1
            TCR &= ~(1<<4);         //Clear the COMPA flag

      }

      return 0;
}
```