HOCHSCHULE
RHEIN-WAAL
Rhine-Waal University
of Applied Sciences

LABORATORY

Microcontroller

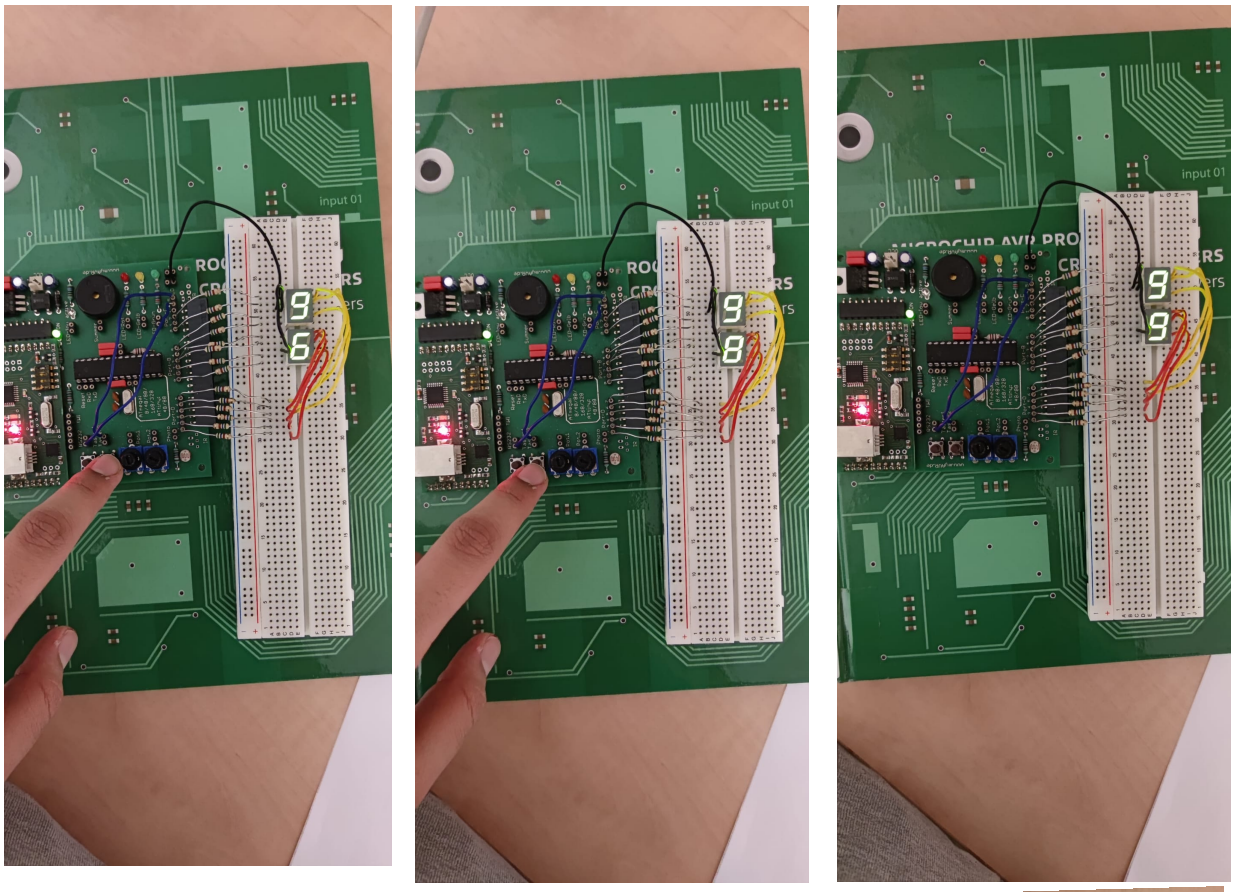
3

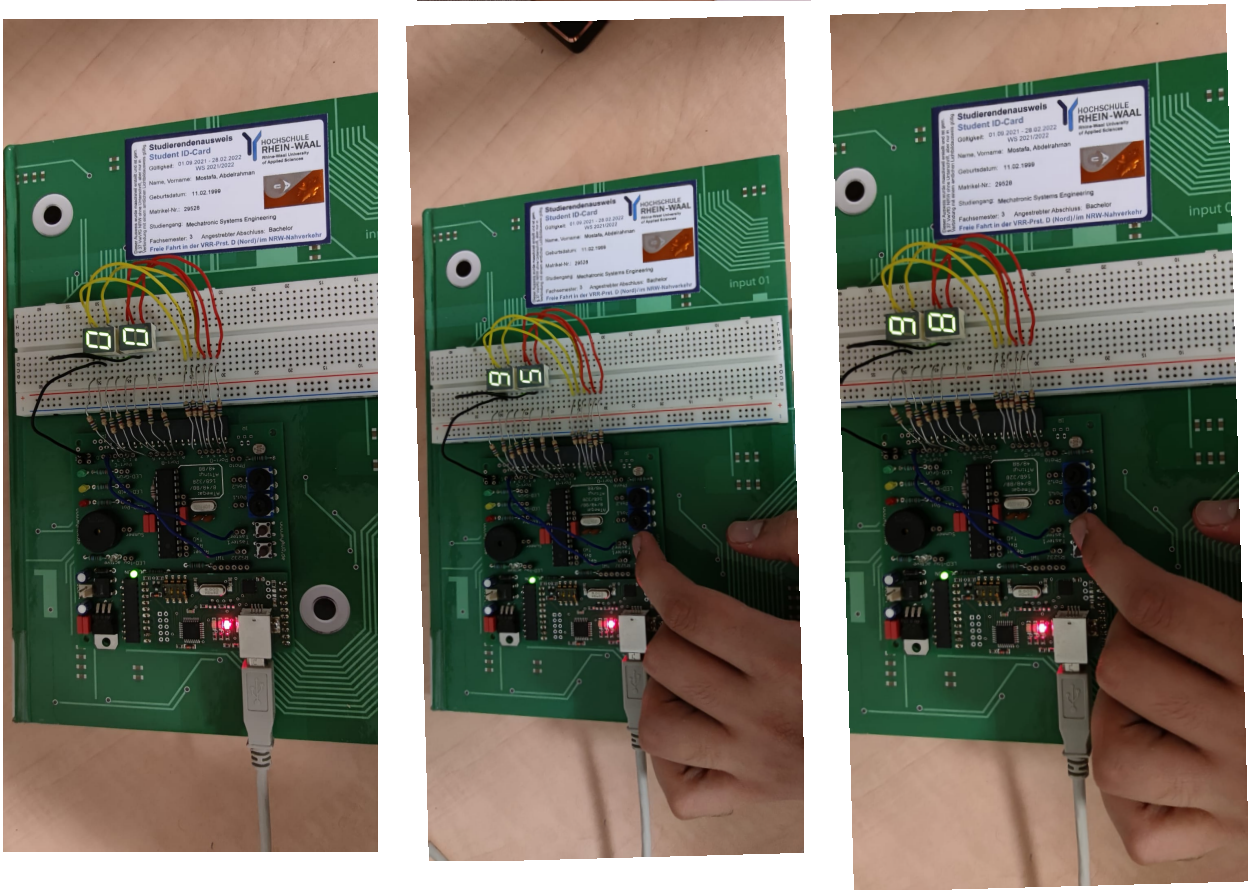
EXPERIMENT:

Seven Segment Counter & I²C Bus: EEPROM

Write your name in every sourcefile you edit and compile the code with your matriculation number (variable in the template). All files should contain all names of the persons who made changes. Do use the @author tag for this (as available in the headline of template files).

Task 1:
What we want at the end of this task is a working “counter”, see Figure 5.





We have 3 functions other than the main and init:
1 - **sevensegment**: which takes the an unsigned integer as an input "value", which is represented in 2 digits.
2 - **sevensegment1**: which take the 2nd digit of the value (value%10). the reminder when dividing the number with 10.
Then we use switch condition to turn the LEDs on or off according to the 2nd digit of the value.
3 - **sevensegment10**: Takes the 1st digit of the value (value/10).
Then we use switch condition to turn the LEDs on or off according to the 1nd digit of the value.

Task 3:
Your program should do the following:

- Show current potentiometer value (0 to 1023) on the LCD,
- Enable the buttons to save and load the values to/from the EEPROM,
- Handle the potentiometer value as a 16 bit value.

Since the address of the memory is 0b1010(0000), where the first 4 bits are always the same, but the last 4, 3 of them are determined from the position of the jumpers, and the last one indicates Read/Write.

Thats is why I made the address of the memory 0xA0 = 0b10100000

```
uint16_t load_value(void){
    uint8_t highbyte, lowbyte;

    i2c_master_open_write(0xA0);
    i2c_master_write(15);
    i2c_master_open_read(0xA0);
    highbyte = i2c_master_read_next();
    lowbyte = i2c_master_read_last();
    i2c_master_close();

    display_showload();

    return highbyte * 256 + lowbyte;
}
```

The reason for using highbyte and lowbyte is because we have learned in the first lab that ADC value is 10 bits, so in order to save or load (write or read) them with an 8-bits memory, we have to divide our value and store it in two positions, and do the opposite when loading (reading) it.

save_value function takes the ADCW value and write it to position 15 and 16 in the memory.

load_value function takes no input, reads the values from position 15, and 16, assigns them to the variables highbyte, and lowbyte, and then return the whole value of the stored ADCW again.

```
void save_value(uint16_t tosave){
    uint8_t highbyte, lowbyte;

    highbyte = tosave/256;
    lowbyte = tosave%256;

    i2c_master_open_write(0xA0); //Device address, open to write
    i2c_master_write(15); //Set the position pointer of the device
    i2c_master_write(highbyte); //Write a byte
    i2c_master_write(lowbyte); //Write a byte
    i2c_master_close(); //Close the device access

    display_showsaved();
}
```