Name: Nguyen Luong Ngoc Hieu - 544469 - Logbook

The contents of the logbook.

1. The time you spend. (preparation, lessons, and evaluation)(start and stop times)
2. Notations and questions, solutions.
3. PSD or flow diagram of programs
4. Program code with explanation.
5. Explanation of the registers and the bits in the registers, new in your program

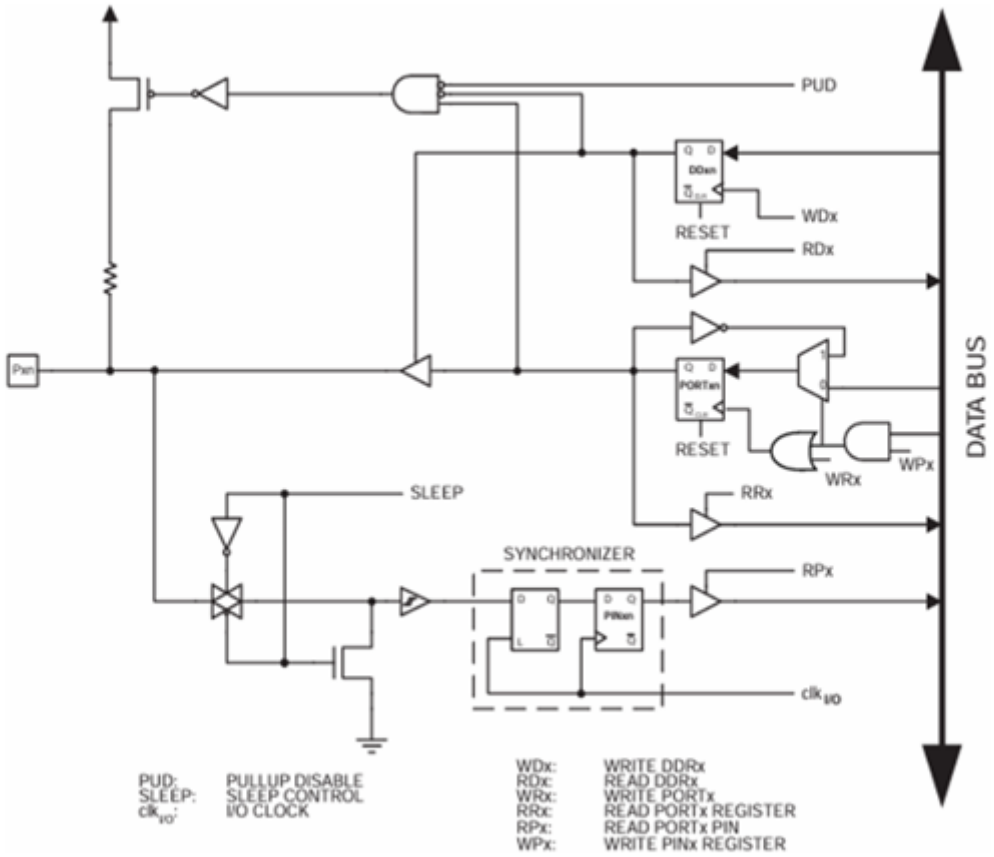
Week 1: IO & Interrupts

Date: 9-7-2025

Step 1: test assignment : "helloworld"

```
This academic year, we will transition from using Microchip Studio back to the
Arduino IDE for embedded systems programming. Please ensure that the Arduino
development environment is installed or reinstalled on your system.

Note that we will not be using .ino sketch files. Instead, we will focus on
writing low-level C code directly. Therefore, remove the default setup() and
loop() functions from your project. Replace them with the provided sample program
below to conduct a basic functionality test. If required, connect an LED to the
designated digital I/O pin.

Carefully study the sample code, annotate it with comments to clarify its purpose
and functionality, and document your understanding in your logbook. Pay particular
attention to the use of C operators and how I/O ports operate internally. For
reference, consult the microcontroller’s datasheet—specifically Figure 13-2—for
details on port structure and behavior.
```

Figure 13-2. General Digital I/O[(1)]



Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Question: what is the difference between a single & and a double && ?

Single & (Bitwise)	**Double && (Logical)**
operate on individual bits of integer data types.	operate on boolean expression and return boolean values (true or false)

Question: why is the F_CPU needed? and what happens if we change it to double or half the value ?

- F_CPU is for defining clock speed of the processor.
- Or, telling the complier what frequency the processor running in
- When we change it to double or half the value the clock cycle will double or half its value.

```
/* MCAexample.c
 * voorbeeld programma voor Microcontrollers
 * C.G.M. Slot (c) 2013
 */
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
```

```
int main(void)
{
  /*
    DDRB - Data Direction Register for Port-B (PIN 8 to PIN 13) (pin is an
    INPUT or OUTPUT)
    PB5 - PORTB5 (pin is HIGH or LOW) - PIN 13 in Arduino Uno R3
    DDRB = (1 << PB5) - Set PIN 13 as OUTPUT
  */
  DDRB = (1 << PB5);
  while(1)
  {
    /*
     Set bit corresponding to PB5 in the PORTB register to 1.
     Also mean, setting pin 5 to Port B to HIGH state (logic 1)
    */
    _delay_ms(100);
    /*
        Clear bit corresponding to PB5 in the PORTB register.
        Also mean, setting pin 5 of Port B to the LOW state (logic 0)
    */
    PORTB |= (1 << PB5);
    _delay_ms(100);
    PORTB &= ~(1 << PB5);

  }
  return 0;
}
```

Step 2: Scanner light

Build and program a scannerlight with at least 5 leds. Use the standard delay function, a while loop and a shift function to blink one led at a time to mimick the motion of travel from left to right and back again. hint: use the skeleton code below as starting point.

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void) {
    DDRB = 0b00011111; // part of port B as output PIN 9 - PIN 13
    /* skeleton code scanner light */
    //_HAS_CHAR16_T_LANGUAGE_SUPPORT
    int position = 0;  // Start at the leftmost LED (PB0)
    int direction = 1; // 1 = right, -1 = left

    while (1) {
        PORTB = (1 << position); // turn on/off led(s)
        // increase or decrease position based on direction
        // check boundaries of position and change direction if needed
        _delay_ms(100);          // Delay for smooth animation
    if (direction == 0)
```

```
    {
        position++;
        if (position >= 5)
        {
            direction = 1;
        }
    }
    else
    {
        position--;
        if (position <= 0)
        {
            direction = 0;
        }
    }
    return 0;
    }
}
```

Step 3: external interrupt example

Try out the external interrupt example: interrupt-test.txt and explain it's working in your logbook. It shows how to turn on/off some leds using a button connected to the external interupt INT0 pin, using an Interrupt Service Routine.

```
/* interrupt test
 * C.Slot Saxion 2021
 */

#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>


volatile int on = 0; // shared volatile variable,
// make sure the compiler does not optimize it away

int main(void) {
    // part of port B as output - set pin from 8 to 13 as OUTPUT mode.
    DDRB = 0b00111111; // part of port B as output
    DDRD = 0;          // port D as inputs ( why? )  all the pins in Port D are
set to INPUT mode.

    EIMSK = (1 << INT0);
    /*
        EICRA - External Interrupt Control Register A
        see data sheet: when does the interrupt react?
        External Interrupt Control Register A (EICRA) define whether the
        external interrupt is activated
        on rising and/or falling edge of the INT0 pin or level sensed. Activity
```

```
        on the pin will cause an
        interrupt request even if INT0 is configured as an output.
    */
    EICRA = 0b00000011;   // see data sheet: when does the interrupt react? At low
level
    sei();                // why ? Sets the global Interrupt flag

    while(1) {
        // waiting for interrupts to make a change
        if (on)
            PORTB = 0b00111111; // turn on leds
        else
            PORTB = 0b00000000; // turn off leds
    }
    return 0;
}

ISR(INT0_vect) {
    _delay_ms(20); // short delay to avoid bouncing
    if (on)        // toggle on
        on = 0;
    else
        on = 1;
}
```

Question: what is the EICRA register used for ?

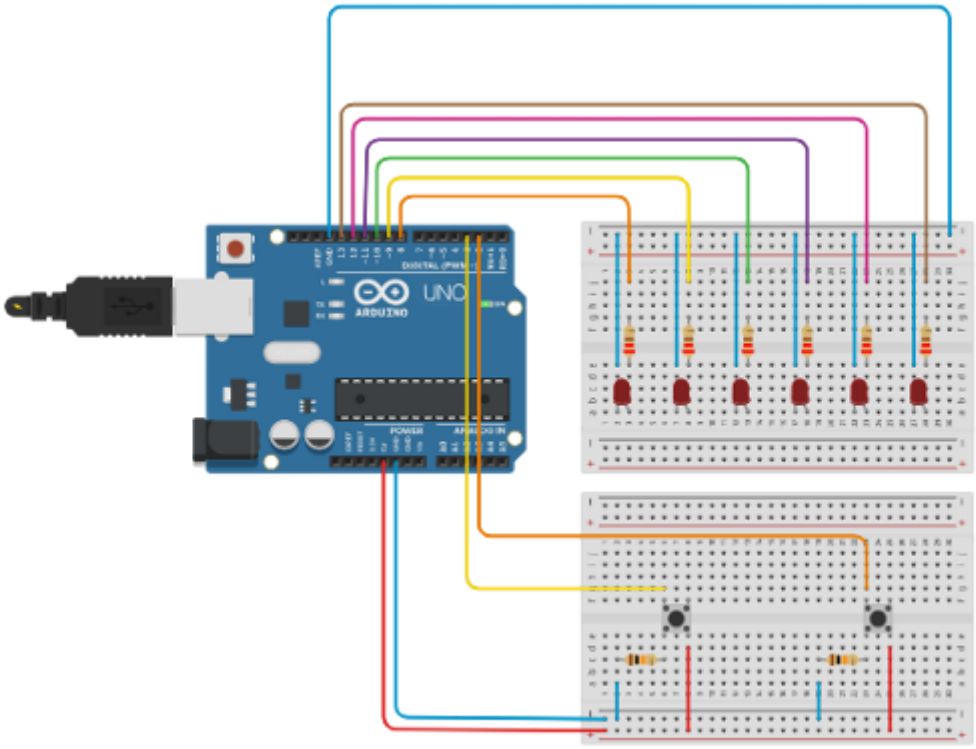
- EICRA register used for controls how external interrupts are triggered.

Question: what is an interrupt vector and where are they located ?

- An interrupt vector is a fixed memory address where the processor jumps when an interrupt occurs.

Step 4: Scanner light interrupt

Now combine step 2 and 3, make the scanner light start "scanning" with one button and stop with another button.



```
/* interrupt test
* C.Slot Saxion 2021
*/
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
volatile int on = 0; // shared volatile variable,
// make sure the compiler does not optimize it away
int main(void)
    {
        // part of port B as output - set pin from 8 to 13 as OUTPUT mode.
        DDRB = 0b00111111;
        // port D as inputs ( why? ) - all the pins in Port D are set to INPUT
mode.
        DDRD = 0;
        /*
        EIMSK - External Interrupt Mask Register
        see data sheet: which interrupt is enabled?
        INT0 - 2 0x0002 INT0 External Interrupt Request 0
        Set HIGH to INT0 - Pin 2
        */
        // Enable both INT0 and INT1
        EIMSK = (1 << INT0) | (1 << INT1);
        /*
        EICRA - External Interrupt Control Register A
        see data sheet: when does the interrupt react?
        External Interrupt Control Register A (EICRA) define whether the external
interrupt is activated on rising and/or falling edge of the INT0 pin or level
```

```
sensed.
        Activity on the pin will cause an interrupt request even if INT0 is
configured as an output.
        */
        // Both INT0 and INT1 have the rising and falling edge
        EICRA = (1 << ISC01) | (1 << ISC00) | (1 << ISC11) | (1 << ISC10);
        /*
            why ?
            sei() - SEts the global interrupt flag
        */
        sei();
        int position = 0; // position of led(s) to turn on
        int direction = 0; // left or right
        while (1){
            if (on){
                PORTB = (1 << position); // turn on/off led(s)
                // increase or decrease position based on direction
                // check boundaries of position and change direction if needed
                _delay_ms(100); // (1000 / n) * F_CPU
                if (direction == 0){
                    position++;
                    if (position >= 5){
                        direction = 1;
                    }
                }
                else{
                    position--;
                    if (position <=0){
                        direction = 0;
                    }
                }
            }
            else{
                PORTB = 0;
            }
            return 0;
        }
    }

ISR(INT0_vect)
{
    _delay_ms(20); // short delay to avoid bouncing
    on = 0;
}

ISR(INT1_vect)
{
    _delay_ms(20); // short delay to avoid bouncing
    on = 1;
}
```

Week 2: Timers

Date: 10-7-2025

Step 1: Timer interrupt

Now build a new programme which toggles/blinks one led with a frequency of 2 Hertz (=half second on, half second off). Use timer 1 with an output compare trigger. Comment and explain the neccessary calculations and registers used in your code and logbook.

Test your programme using an oscilloscope. Measure the acurateness of the blinking, also try to change the value to 4 Hertz blinking (quarter of a second on / quarter off) and measure again, include a screenshot/photo (with the measured delay visible) of your tests on the scope in your logbook.

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
volatile uint8_t overflow_count = 0;

void (setup){
    // Set LED pin as output
    DDRB = 0b00110000;
    /*
        Assuming a clock frequency of 16 MHz and a prescaler of 256:
        Timer frequency = 16,000,000 / 256 = 62,500 Hz
        For 2 Hz blinking, the timer needs to toggle every 0.5 seconds.
        OCR1A = Timer frequency * 0.5 - 1 = 62,500 * 0.5 - 1 = 31249
        For 4 Hz blinking, the timer needs to toggle every 0.25 seconds.
        OCR1A = Timer frequency * 0.5 - 1 = 62,500 * 0.25 - 1 = 15624
    */
    // Configure Timer 1
    TCCR1B |= (1 << WGM12); // CTC mode
    OCR1A = 15624; // Set compare value for 4 Hz
    TIMSK1 |= (1 << OCIE1A); // Enable Timer 1 compare interrupt
    TCCR1B |= (1 << CS12); // Prescaler 256

    /*
        8 - bit => 2^8 - 1 = 255
        Assuming a clock frequency of 16 MHz and a prescaler of 256:
        Timer frequency = 16,000,000 / 256 = 62,500 Hz
        For 4 Hz blinking, the timer needs to toggle every 0.25 seconds.
        TOIE0 = Timer frequency * 0.25 / 255 = 62,500 * 0.25 / 255 = 61
    */
    // Configure Timer 0 for the second LED
    TCCR0A = 0; // Normal mode
    TIMSK0 |= (1 << TOIE0); // Enable Timer 0 overflow interrupt
    TCCR0B |= (1 << CS02); // Prescaler 256

    // Enable global interrupts
    sei();
}

int main(){
```

```
    setup();
    while (1){
    // Main loop
    }
}

ISR(TIMER1_COMPA_vect)
{
    // Toggle LED
    PORTB ^= (1 << PORTB5);
}

ISR(TIMER0_OVF_vect)
{
    overflow_count++;
    if (overflow_count >= 61)
    {
        // Toggle second LED
        PORTB ^= (1 << PORTB4);
        overflow_count = 0;
    }
}
```

Explanation of calculations:

For 16 MHz clock, prescaler of 1024, and 0.5 Hz toggle frequency:

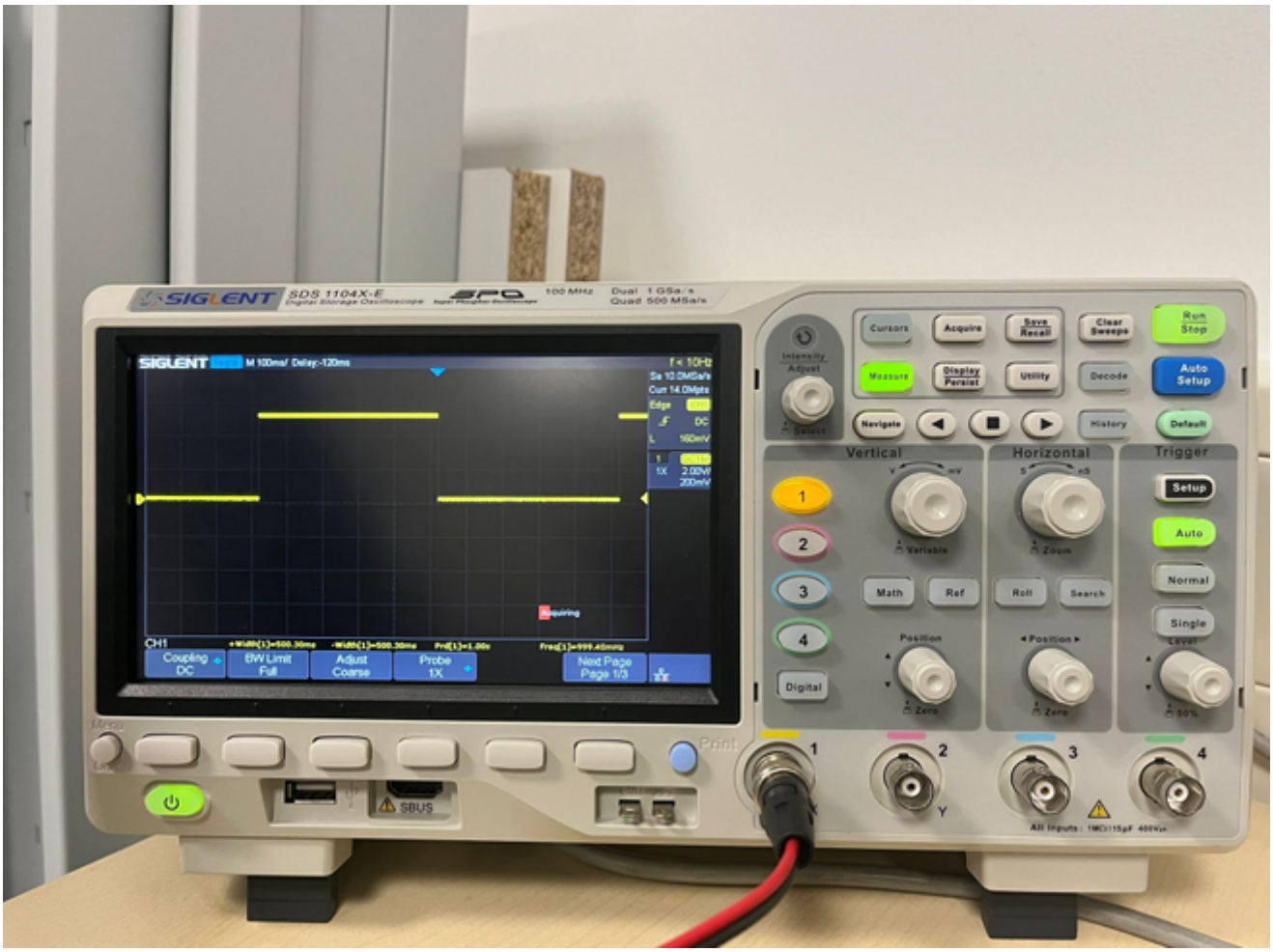
OCR1A = clock frequency(F_CPU) / (prescaler blink frequency)

OCR1A = (16,000,000/ 1024 2) = 7812.5 (rounded to 7812).

Images

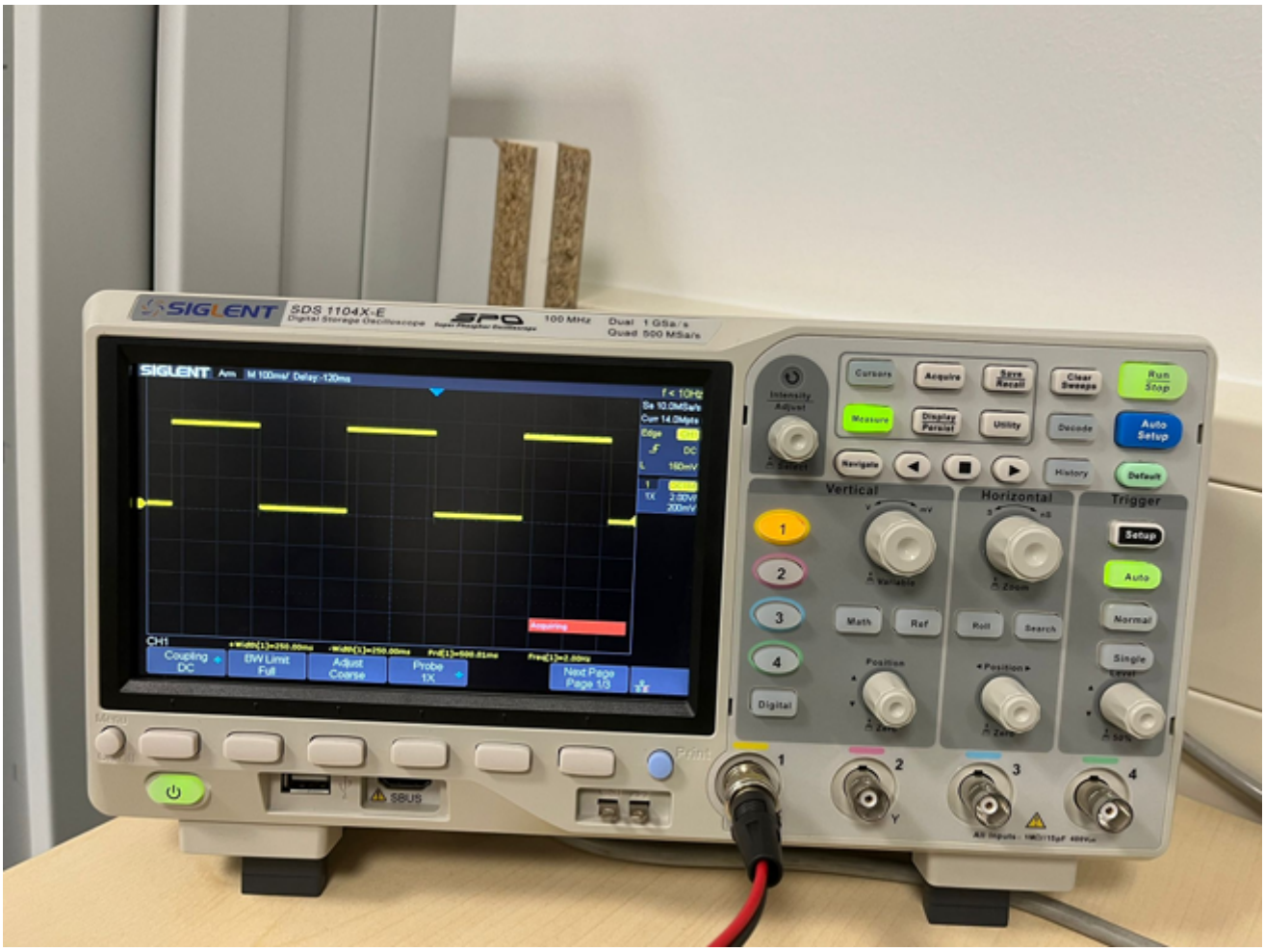
For 2 Hz blinking, the timer needs to toggle every 0.5 seconds.

OCR1A=Timerfrequency∗0.5−1 = 62,500∗0.5−1 = 31249



For 4 Hz blinking, the timer needs to toggle every 0.25 seconds.

$$OCR1A = Timerfrequency * 0.5 - 1 = 62{,}500 * 0.25 - 1 = 15624$$



Question: what is the difference between overflow and output compare mode ?

Output compare mode(CTC):

- In this mode, the timer counter is compared to a predefined value(OCRxA or OCRxB).
- When the counter matches this value, an interrupt is triggered, and the counter can optionally reset.
- This mode allows precise control of the timing and is ideal for generating fixed frequencies or intervals.
- Used here for creating a 0.5-second on/off LED blink.

Overflow mode:

- The counter counts up to its maximum value (255 for 8-bit, 65535 for 16-bit timers).
- An interrupt occurs when the counter overflows back to 0.
- This mode is simpler but less precise than output compare because the interval depends on the timer's resolution and prescaler.

Different modes for timers:

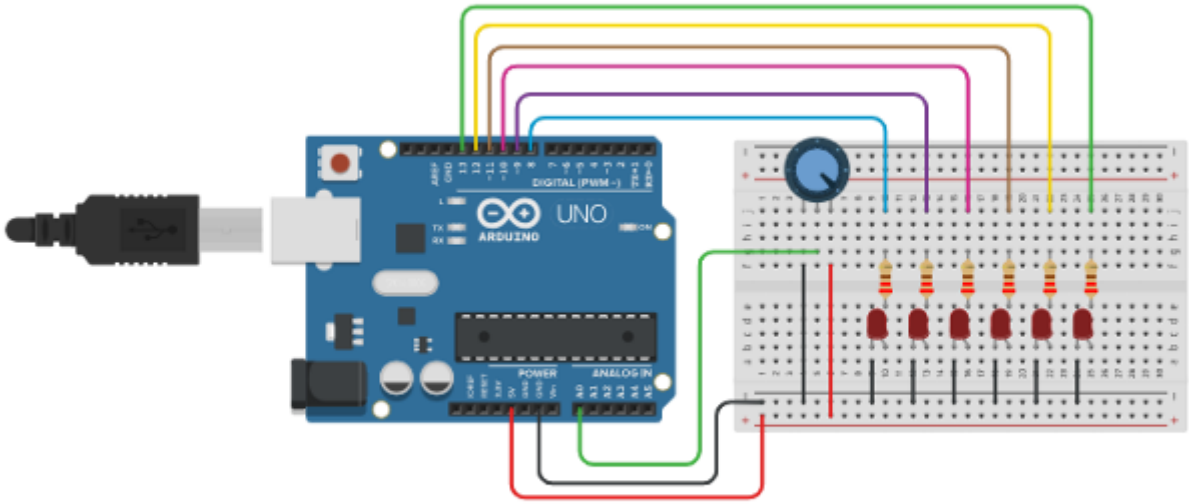
- CTC Mode: The timer counts to a value you set (like 7812 for) and then resets to 0.
- Overflow Mode: The timer counts to its maximum value (like 255 for 8-bit) and then resets.

Week 3: ADC

Date: 10-7-2025

Step 1: AD conversion

Build a program to read a potentiometer and display its MSB value binary on four or more leds of port B. Use the ADC interrupt and the external AREF connected to 5 volts and make good use of the ADLAR bit. Comment your programme and explain in your logbook.




```
/*
* Name: Potentiometer ADC to LED (6 LEDs)
* Description: Reads the MSB of an ADC value from a potentiometer and displays it
in binary on 6 LEDs connected to PORTB using ADC interrupts.
* Configuration: External AREF (5V) and ADLAR enabled for left adjustment.
* Author: Hieu
*/
#include <avr/io.h>
#include <avr/interrupt.h>
// Function to initialize the ADC

void ADC_Init()
{
    // Set ADC prescaler to 64 for a 125kHz ADC clock (16 MHz / 64)
    ADCSRA |= (1 << ADPS2) | (1 << ADPS1);
    // Enable ADC, ADC interrupt, and set ADC reference to external AREF
    ADCSRA |= (1 << ADEN) | (1 << ADIE);
    ADMUX |= (1 << REFS0); // External AREF (connect AREF pin to 5V)
    // Enable left adjustment for ADCH (8 MSBs in ADCH)
    // ADC Left Adjust Result
    // The ADLAR bit affects the presentation of the ADC conversion result in the
```

```
ADC Data Register.
    // Write one to ADLAR to left adjust the result. Otherwise, the result is
right adjusted. Changing the
    // ADLAR bit will affect the ADC Data Register immediately, regardless of any
ongoing conversions.
    ADMUX |= (1 << ADLAR);
    // Select ADC channel 0 (potentiometer connected to PC0/ADC0)
    ADMUX = (ADMUX & 0xF0) | 0x00;
    // Start the first ADC conversion
    ADCSRA |= (1 << ADSC);
}

// ADC Conversion Complete Interrupt Service Routine
ISR(ADC_vect)
{
    // Read the 8 MSBs of ADC result (ADCH)
    uint8_t msb_value = ADCH;
    // Display the 6 MSBs (bits 7 to 2) of the ADC value on PORTB LEDs
    PORTB = msb_value >> 2;
    // Start the next ADC conversion
    ADCSRA |= (1 << ADSC);
}

int main()
{
    // Set lower 6 bits of PORTB as output for LEDs
    DDRB = 0b00111111;
    // Initialize ADC
    ADC_Init();
    // Enable global interrupts
    sei();
    // Infinite loop - All work is handled in the ISR
    while (1)
    {
        // The main loop remains empty as the ISR handles ADC and PORTB updates
    }
    return 0;
}
```

Question 1: What Happens if You Switch AREF from 5V to 3.3V?

The **AREF (Analog Reference Voltage)** determines the range of voltages that the ADC can convert. Switching AREF from 5V to 3.3V affects the ADC behavior as follows:

Reduced Voltage Range:

- With AREF set to 5V:
  - The ADC maps an input voltage range of **0V to 5V** to digital values **0 to 255**.
- With AREF set to 3.3V:
  - The ADC maps a smaller range of **0V to 3.3V** to digital values **0 to 255**.

Higher Resolution:

$$ \text{Resolution} = \frac{\text{Voltage Range}}{256} $$

- For **AREF = 5V**:
  - 5V/256 = 19.53mV/step
- For **AREF = 3.3V**:
  - 3.3V/256 = 12.89mV/step
- Smaller steps mean more precision within the 3.3V range.

Behavior for Input Above 3.3V:

- Voltages above 3.3V will exceed the ADC range and will likely saturate the ADC output at **255**, leading to inaccurate results.

Summary:

- Switching AREF to **3.3V** improves resolution for signals within **0 to 3.3V**, but any input above **3.3V** will be clipped, causing a loss of accuracy.

Question 2: How Can You Set the ADC at a Sampling Rate of 5 kHz?

The ADC sampling rate depends on the **ADC clock frequency** and the number of clock cycles per conversion. Here's how to configure it for 5 kHz:

ADC Conversion Time:

- Each ADC conversion takes **13 clock cycles**.
- To achieve a sampling rate of **5 kHz**, the ADC clock should be: ADCClock = 5,000×13 = 65,000Hz (65kHz)

ADC Prescaler Calculation:

- The ADC clock is derived from the microcontroller clock (16 MHz) using the prescaler.
- Required prescaler: $$ \text{Prescaler} = \frac{\text{Microcontroller Clock}}{\text{ADC Clock}} = \frac{16,000,000}{65,000} \approx 246 $$
- The closest available prescaler is 256, which gives: $$ \text{ADC Clock} = \frac{16,000,000}{256} = 62,500 ,\text{Hz} ; (\text{slightly below } 65 ,\text{kHz}) $$

Code Updated:

Set the ADC prescaler to **256**:

```
ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // Prescaler = 256
```

Sampling Rate Verification:

- With the ADC clock at **62.5 kHz**, the sampling rate is calculated as: $$ \text{Sampling Rate} = \frac{\text{ADC Clock}}{13} = \frac{62,500}{13} \approx 4,807 ,\text{Hz} $$
- This is close to 5 kHz and will work for most applications.

Week 4: UART

Date: 11-7-2025

Question: Explain the added code in your logbook

```
/*
    * uart skeleton programme, fill in missing code yourself
    * Saxion (c) 2023
    *
    * connect four leds on port B 0-3 to display incoming bytes in binary
    *
*/

#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
// calculate and fill in the correct baudrate for 9600 on 16 MHz
// UBRR (USART Baud Rate Register)
// UBRR = 16000000UL / (16 * 9600) - 1 = 103

#define baudrate 103

volatile char latestChar = '*';

void initUART() {
  // set UART baudrate
  UBRR0 = baudrate;
  // Look up & set correct frame format: 8data, 2stop bit, no parity
  UCSR0C = (0 << UPM01) | (0 << UPM01) | (1 << USBS0) | (0 << UCSZ02) |
           (1 << UCSZ01) | (1 << UCSZ00);
  // enable rx & tx & interrupt receive
  // RXEN0: Receiver enable.
  // TXEN0: Transmitter enable.
  // RXCIE0: Receive complete interrupt enable.
  UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0);  // uart runs from now
  on.
};
void UARTsendchar(unsigned char data) {
  while (!(UCSR0A & (1 << UDRE0))) {
    // wait till UDR buffer is empty before filling
  }
  UDR0 = data;
}
ISR(USART_RX_vect) {
  // when receive something, put it on the leds
```

```
    // except when cr or lf
    char temp = UDR0;
    if (!((temp == 0x0D) || (temp == 0x0A))) {  // filter out lf & cr
      PORTB = temp;
      latestChar = temp;  // store received byte
    }
  }
  int main(void) {
    DDRB = 0b00001111;  // leds on portb 0-3
    initUART();
    sei();
    // Enable global interrupts
    while (1) {
      _delay_ms(500);
      UARTsendchar('H');
      UARTsendchar('e');
      UARTsendchar('l');
      UARTsendchar('l');
      UARTsendchar('o');
      UARTsendchar(' ');
      UARTsendchar(latestChar);
      UARTsendchar('\n');
    }
  }
```

Now connect four leds and try out this UART example programme with a terminal programme (for instance Putty or the arduino IDE) on 9600 bauds.

Step 1: UART basics

1. Look up an ASCII table, what character do you need to send to "light" all four LEDs?

To light all four LEDs connected to PORTB0 to PORTB3, you need to send a byte where the lowest four bits (representing the state of the LEDs) are set to 1. This would be the binary value 1111, which corresponds to the hexadecimal value 0x0F.

Answer: The character you need to send is 0x0F (or the ASCII value for \x0F).

In ASCII, 0x0F corresponds to the "Shift In" control character, but in this program, it's being used to represent the binary value 1111 on the LEDs.

2. Explain how the UDR register "works" so that you can both read from and write to it.

The UDR0 register in the AVR microcontroller is used for both transmitting and receiving data via the UART interface.

Writing to UDR0:

When you send data, you write the byte you want to transmit into the UDR0 register. Once written, the UART module will transmit this byte out of the TX pin (transmit pin) at the configured baud rate.

You can only write to UDR0 when the UDRE0 flag in the UCSR0A register is set, indicating that the data register is empty and ready to be written.

Reading from UDR0:

When data is received over UART, the received byte is stored in the UDR0 register.

You can read from UDR0 to retrieve the incoming byte.

The RXC0 flag in the UCSR0A register is set when new data is available to read from UDR0.

Summary:

Writing: You load data into UDR0 for transmission.

Reading: You read the incoming data from UDR0 when the RXC0 flag is set, which indicates that data has been received.

Step 2: ADC & UART

Combine the ADC from the previous week and the UART in one programme. Once an AD conversion is complete, use the ADC interrupt to store the result in a variable, and in the main while loop send each second that variable as a byte out on the UART. Use an UART interrupt to receive incoming bytes from the UART and display that byte as a binary digit on the leds.

Try this out in three way:

- With a serial monitor on PC.
- With the TX/RX pins connected so you "send" and "receive" your own bytes.
- Connect your Arduino to your neighbour, your ADC is displayed on the other leds and vice versa.

```
/*
 * uart skeleton programme, fill in missing code yourself
 * Saxion (c) 2023
 *
 * connect four leds on port B 0-3 to display incoming bytes in binary
 *
 */
#define F_CPU 16000000UL
#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>
#define baudrate 103
void ADC_init(void);
volatile char latestChar = '*';
volatile int variable;
void initUART() {
  // set UART baudrate
  UBRR0 = baudrate;
  // Look up & set correct frame format: 8data, 2stop bit, no parity
  UCSR0C = (0 << UPM01) | (0 << UPM01) | (1 << USBS0) | (0 << UCSZ02) |
           (1 << UCSZ01) | (1 << UCSZ00);
```

```
  // enable rx & tx & interrupt receive
  UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0);  // uart runs from
  now on.
};
void UARTsendchar(unsigned char data) {
  while (!(UCSR0A & (1 << UDRE0))) {
    // wait till UDR buffer is empty before filling
  }
  UDR0 = data;
}
ISR(USART_RX_vect) {
  // when receive something, put it on the leds
  // except when cr or lf
  char temp = UDR0;
  if (!((temp == 0x0D) || (temp == 0x0A))) {  // filter out lf & cr
    PORTB = temp;
    latestChar = temp;  // store received byte
  }
}
int main(void) {
  DDRB = 0b00001111;  // leds on portb 0-3
  initUART();
  ADC_init();
  sei();  // why ?
  while (1) {
    _delay_ms(500);
    char localVariable = (char)variable;
    UARTsendchar(localVariable);
  }
}
void ADC_init(void) {
  ADMUX = (1 << REFS0) | (1 << ADLAR);  // Select A0 as input and use AREF
  as manual reference point(connect to same input voltage as the circuit)
      ADCSRA = (1 << ADEN) | (1 << ADSC) | (1 << ADATE) | (1 << ADIE) |
               (1 << ADPS2) | (1 << ADPS1);
  /*
  Enable:
  ADEN > Enables ADC (Analog to digital conversion)
  ADSC > Initial conversion
  ADATE > Continious conversions
  ADIE > Enable interrupt (ADC_vect)
  ADPS > Configures the prescalar (f_CPU / 128) > slow down the ADC
  clock > higher resolution
  */
}
// ADC Conversion Complete Interrupt Service Routine
ISR(ADC_vect) {
  // Read MSB of the ADC result and display it on PORTB upper 4 bits
  variable = ADCH >> 4;
  ADCSRA |= 1 << ADSC;
}
```

Week 5: SPI

Date: 11-7-2025

Step 1: SPI (& I2C) example

Ask the teacher for a SPI board. Try out the SPI example programme and study it's behaviour along with the datasheets of the chips involved (see resources). Also have a look at the I2C example programme and answer the questions below in your own words in your logbook.

```
/*
 * SPIexample
 * The example shows how to work with the  SPI Expander chip MCP23s08 as
 output
 * The example also remembers you how to work with the USART
 * Created: 9-10-2015 07:35:40 /april 2016 / februari 2019
 * Author : Tom IJsselmuiden/Hans Stokkink
 */
#define F_CPU 16000000UL
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <util/delay.h>
#define SS 2    // pin 10 arduino uno
#define MOSI 3  // pin 11 arduino uno
#define MISO 4  // pin 12 arduino uno
#define SCK 5   // pin 13 arduino uno
#define TXD 1
#define ENABLE_SS PORTB &= ~(1 << SS)
#define DISABLE_SS PORTB |= (1 << SS)
#define ENABLE_SS2 PORTB &= ~(1 << PORTB1)  // pin 9 arduino = pin portb1 for
other spi chips
#define DISABLE_SS2 PORTB |= (1 << PORTB1)
#define FOSC 16000000
#define BAUD 9600
#define MYUBBR FOSC / 16 / BAUD - 1
#define UART_BUFF_SIZE 10
#define A 0
#define B 1
#define READ 1
#define WRITE 0
#define IODIR 0
#define OUTP_LATCH 10
    // void uart_init(void);
void USART_Init(void);
void init(void);
uint8_t spi_tranceiver(uint8_t data);
void USART_Transmit(uint8_t data);
```

```
int16_t Temp_read(void);
void IOEXP_IODIR(uint8_t data);
void IOEXP_datalatch(uint8_t data);
uint8_t RX_buf[UART_BUFF_SIZE];
uint8_t TX_buf[UART_BUFF_SIZE];
uint8_t RX_index = 0;  // index for RX buffer RX = receive
uint8_t TX_index = 0;  // index for TX buffer TX = transmit

ISR(USART_RX_vect) {
  // store data in buffer
  RX_buf[RX_index++] = UDR0;  // USART data register
  // echo data
  USART_Transmit(RX_buf[RX_index - 1]);
}

int main(void) {
  int x = 0;
  init();
  /* Clearing buffers */
  memset(RX_buf, 0, sizeof(RX_buf));
  memset(TX_buf, 0, sizeof(TX_buf));
  /* Enable interrupts */
  sei();
  /* set all pins of I/0 expander to output */
  IOEXP_IODIR(0);
  /* set lower 4 bits (leds) high*/
  IOEXP_datalatch(0x0f);
  _delay_ms(1000);
  /* set higher 4 bits (leds) high*/
  IOEXP_datalatch(0xf0);
  _delay_ms(1000);
  int16_t data = 0xaa;
  IOEXP_datalatch(data);  // set led
  while (1) {
    data++;
    // int16_t data = 0xaa;
    IOEXP_datalatch(data);  // set led
    USART_Transmit(data);   // give same value to USART
    _delay_ms(100);
  }
}

void init() {
  USART_Init();
  /* Set SS, MOSI and SCK output, all others input */
  DDRB = (1 << SS) | (1 << MOSI) | (1 << SCK) | (1 << PORTB1);
  /* LED PIN */
  DDRB |= (1 << PORTB0);
  /* Set TXD as an output */
  DDRD = (1 << TXD);
  /* Set the slave select pin (Active low) */
  DISABLE_SS;
  DISABLE_SS2;
  /* Enable SPI, Master, set clock rate fosc/16 */
```

```
  SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);
}

// function to initialize UART
void USART_Init(void) {
  // Set baud rate:
  UBRR0 = 103;  // UBRR= Fosc /(16*9600) -1 =103.166= 103
  // enable receiver and transmitter
  UCSR0B |= (1 << RXEN0 | (1 << TXEN0));
  // Set frame format : 8 data 2 stop bit
  UCSR0C = (1 << USBS0) | (3 << UCSZ00);
}

void USART_Transmit(uint8_t data) {
  /* Wait for empty transmit buffer */
  while (!(UCSR0A & (1 << UDRE0)));
  /* Put data into buffer, sends the data */
  UDR0 = data;
}

/* spi data buffer load and send */
uint8_t spi_tranceiver(uint8_t data) {
  // Load data into the buffer
  SPDR = data;
  // Wait until transmission complete
  while (!(SPSR & (1 << SPIF)));
  // Return received data
  return (SPDR);
}


/* IO expander data direction set */
void IOEXP_IODIR(uint8_t data) {
  uint8_t r_data = 0;
  // Make slave select go low
  ENABLE_SS;
  /* Send device address + r/w */
  spi_tranceiver((1 << 6) | WRITE);
  /* Send command */
  spi_tranceiver(IODIR);
  spi_tranceiver(data);
  // Make slave select go high
  DISABLE_SS;
}
/* IO expander latch  set */
void IOEXP_datalatch(uint8_t data) {
  // Make slave select go low
  ENABLE_SS;
  /* Send device address + r/w */
  spi_tranceiver((1 << 6) | WRITE);
  /* Send command */
  spi_tranceiver(OUTP_LATCH);
  spi_tranceiver(data);
  // Make slave select go high
```

```
  DISABLE_SS;
}
//*************************************************************************
*
    //  File Name: i2c - example.c
    //  Version : 1.0
    //  Description  : AVR I2C Bus Master with IO expander
    //  IDE: Atmel AVR Studio
    //  Programmer   : Stokkink
    //
    :
    //  Last Updated : April 2016
    //*************************************************************************
    *
#include <avr/io.h>
#include <compat/twi.h>
#include <util/delay.h>
#define MAX_TRIES 50
#define MCP23008_ID 0x40    // MCP23008 Device Identifier
#define MCP23008_ADDR 0x00  // MCP23008 Device Address
#define IODIR 0x00
#define GPIO 0x09
#define OLAT 0x0A
#define I2C_START 0
#define I2C_DATA 1
#define I2C_DATA_ACK 2
#define I2C_STOP 3
#define ACK 1
#define NACK 0
#define DATASIZE 32
    // MCP23008 I/O Direction Register
    // MCP23008 General Purpose I/O Register
    // MCP23008 Output Latch Register
    /* START I2C Routine */
unsigned char i2c_transmit(unsigned char type) {
  switch (type) {
    case I2C_START:  // Send Start Condition
      TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
      break;
    case I2C_DATA:  // Send Data with No-Acknowledge
      TWCR = (1 << TWINT) | (1 << TWEN);
      break;
    case I2C_DATA_ACK:  // Send Data with Acknowledge
      TWCR = (1 << TWEA) | (1 << TWINT) | (1 << TWEN);
      break;
    case I2C_STOP:  // Send Stop Condition
      TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
      return 0;
  }
  // Wait for TWINT flag set on Register TWCR
  while (!(TWCR & (1 << TWINT)));
  // Return TWI Status Register, mask the prescaler bits (TWPS1,TWPS0)
  return (TWSR & 0xF8);
}
```

```
char i2c_start(unsigned int dev_id, unsigned int dev_addr,
               unsigned char rw_type) {
  unsigned char n = 0;
  unsigned char twi_status;
  char r_val = -1;
i2c_retry:
  if (n++ >= MAX_TRIES) return r_val;
  // Transmit Start Condition
  twi_status = i2c_transmit(I2C_START);
  // Check the TWI Status
  if (twi_status == TW_MT_ARB_LOST) goto i2c_retry;
  if ((twi_status != TW_START) && (twi_status != TW_REP_START)) goto i2c_quit;
  // Send slave address (SLA_W)
  TWDR = (dev_id & 0xF0) | (dev_addr & 0x0E) | rw_type;
  // Transmit I2C Data
  twi_status = i2c_transmit(I2C_DATA);
  // Check the TWSR status
  if ((twi_status == TW_MT_SLA_NACK) || (twi_status == TW_MT_ARB_LOST))
    goto i2c_retry;
  if (twi_status != TW_MT_SLA_ACK) goto i2c_quit;
  r_val = 0;
i2c_quit:
  return r_val;
}
void i2c_stop(void) {
  unsigned char twi_status;
  // Transmit I2C Data
  twi_status = i2c_transmit(I2C_STOP);
}
char i2c_write(char data) {
  unsigned char twi_status;
  char r_val = -1;
  // Send the Data to I2C Bus
  TWDR = data;
  // Transmit I2C Data
  twi_status = i2c_transmit(I2C_DATA);
  // Check the TWSR status
  if (twi_status != TW_MT_DATA_ACK) goto i2c_quit;
  r_val = 0;
i2c_quit:
  return r_val;
}
char i2c_read(char *data, char ack_type) {
  unsigned char twi_status;
  char r_val = -1;
  if (ack_type) {
    // Read I2C Data and Send Acknowledge
    twi_status = i2c_transmit(I2C_DATA_ACK);
    if (twi_status != TW_MR_DATA_ACK) goto i2c_quit;
  } else {
    // Read I2C Data and Send No Acknowledge
    twi_status = i2c_transmit(I2C_DATA);
    if (twi_status != TW_MR_DATA_NACK) goto i2c_quit;
  }
```

```
  // Get the Data
  *data = TWDR;
  r_val = 0;
i2c_quit:
  return r_val;
}
void Write_MCP23008(unsigned char reg_addr, unsigned char data) {
  // Start the I2C Write Transmission
  i2c_start(MCP23008_ID, MCP23008_ADDR, TW_WRITE);
  // Sending the Register Address
  i2c_write(reg_addr);
  // Write data to MCP23008 Register
  i2c_write(data);
  // Stop I2C Transmission
  i2c_stop();
}
unsigned char Read_MCP23008(unsigned char reg_addr) {
  char data;
  // Start the I2C Write Transmission
  i2c_start(MCP23008_ID, MCP23008_ADDR, TW_WRITE);
  // Read data from MCP23008 Register Address
  i2c_write(reg_addr);
  // Stop I2C Transmission
  i2c_stop();
  // Re-Start the I2C Read Transmission
  i2c_start(MCP23008_ID, MCP23008_ADDR, TW_READ);
  i2c_read(&data, NACK);
  // Stop I2C Transmission
  i2c_stop();
  return data;
}
void i2c_init(void) {}
// Initial ATMega328P TWI/I2C Peripheral
TWSR = 0x00;  // Select Prescaler of 1
// SCL frequency = 11059200 / (16 + 2 * 48 * 1) = 98.743 kHz
TWBR = 0x30;  // 48 Decimal
int main(void) {
  int x = 0;
  // Initial Master I2C
  i2c_init();
  // Initial the MCP23008 GP0 to GP7 as Output
  Write_MCP23008(IODIR, 0b00000000);
  Write_MCP23008(GPIO, 0b00000000);  // Reset all the Output Port
  // Loop Forever
  while (1) {
    // Write to MCP23008 GPIO Register
    Write_MCP23008(GPIO, x++);
    _delay_ms(100);
  }
  return 0;
}
// endoffile
```

Question: What is the difference between half-duplex and full duplex? How does this relate to SPI and I2C ?

Half-duplex vs Full-duplex:

- Half-duplex: Communication in both directions but not simultaneously
    - I2C is half-duplex (uses single data line SDA)
- Full-duplex: Communication in both directions simultaneously
    - SPI is full-duplex (separate MOSI/MISO lines)

Question: Why does I2C need resistors? And why doesn't SPI need resistors? I2C vs SPI Resistors:

I2C vs SPI Resistors:

- I2C needs pull-up resistors because:
    - Uses open-drain/open-collector configuration
    - Devices can only pull the line LOW
    - Resistors pull the line HIGH when no device is transmitting
    - Typical values: 4.7kΩ to 10kΩ
- SPI doesn't need pull-up resistors because:
    - Uses push-pull outputs
    - Active driving of both HIGH and LOW states
    - Direct point-to-point connections

Question: When connecting multiple devices to the "bus" how do I2C and SPI "select" the correct device?

Device Selection Methods:

- I2C Device Selection:
    - Uses unique 7-bit device addresses (0x00-0x7F)
    - All devices share same SDA/SCL lines
    - Address sent at start of transmission
    - Only device with matching address responds
    - Supports up to 128 unique devices theoretically
- SPI Device Selection:
    - Uses dedicated Chip Select (CS/SS) line per device
    - Master activates one CS line at a time (active low)
    - Each slave needs separate CS connection
    - No addressing protocol needed
    - Number of devices limited by available GPIO pins

Question: How is decided who the "master" is and who the "slave" in both I2C and SPI?

Master/Slave Configuration:

- I2C:
    - Devices can be configured as master or slave
    - Multi-master possible but complex
    - Master initiates all communications
    - Slaves only respond when addressed
- SPI:
    - One dedicated master (usually microcontroller)
    - Fixed master/slave roles
    - Master controls clock (SCK)
    - No multi-master capability

Question: How is the byte data sent in I2C and SPI: MSB or LSB first? Can this be changed? Bit Transmission Order:

- I2C:
    - Always MSB first
    - Not configurable
    - Part of protocol specification
- SPI:
    - Usually MSB first but configurable
    - Can be changed in master configuration
    - Some devices support both orders
    - Must match between master and slave
    - Direct drive between master and slaves

Step 2: SPI & ADC sensor - MCP3201 ADC Analysis

Question: What is the input and output range of the ADC chip?

MCP3201 Specifications:

- Input Range: 0V to VDD (typically 5V or 3.3V)
- Output Range: 12-bit resolution (0-4095)
- Single pseudo-differential input
- Sampling rate up to 100 ksps

Question: How can you read out the chip?

MCP3201 Readout Process:

1. Pull CS line LOW to start conversion
2. Clock in 15 bits:
    - Firsts 3 bits are null/dummy
    - Next 12 bits contain ADC value (MSB first)
3. Pull CS line HIGH to end conversion

```
/*
 * SPIexample
 * The example shows how to work with the  SPI Expander chip MCP23s08 as
 output
 * The example also remembers you how to work with the USART
 * Created: 9-10-2015 07:35:40 /april 2016 / februari 2019
 * Author : Tom IJsselmuiden/Hans Stokkink
 */
#define F_CPU 16000000UL
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <util/delay.h>
#define SS 2    // pin 10 arduino uno
#define MOSI 3  // pin 11 arduino uno
#define MISO 4  // pin 12 arduino uno
#define SCK 5   // pin 13 arduino uno
#define TXD 1
#define ENABLE_SS PORTB &= ~(1 << SS)
#define DISABLE_SS PORTB |= (1 << SS)
#define ENABLE_SS2 PORTB &= ~(1 << PORTB1)  // pin 9 arduino = pin portb1 for
other spi chips
#define DISABLE_SS2 PORTB |= (1 << PORTB1)
#define FOSC 16000000
#define BAUD 9600
#define MYUBBR FOSC / 16 / BAUD - 1
#define UART_BUFF_SIZE 10
#define A 0
#define B 1
#define READ 1
#define WRITE 0
#define IODIR 0
#define OUTP_LATCH 10
    // void uart_init(void);
void USART_Init(void);                  // Function to initialize UART
void init(void);                        // Function to initialize the system
uint8_t spi_tranceiver(uint8_t data);   // Function to perform SPI transaction
void USART_Transmit(uint8_t data);      // Function to transmit data via USART
int16_t Temp_read(void);                // Function to read temperature
void IOEXP_IODIR(uint8_t data);         // Function to set IO expander data
direction
void IOEXP_datalatch( uint8_t data);   // Function to set IO expander latch
uint16_t readADC_MCP3201(void);
uint8_t RX_buf[UART_BUFF_SIZE];
```

```
uint8_t TX_buf[UART_BUFF_SIZE];
uint8_t RX_index = 0;  // index for RX buffer RX = receive
uint8_t TX_index = 0;  // index for TX buffer TX = transmit

ISR(USART_RX_vect) {
  // store data in buffer
  RX_buf[RX_index++] = UDR0;  // USART data register
  // echo data
  USART_Transmit(RX_buf[RX_index - 1]);
}

int main(void) {
  int x = 0;
  init();
  /* Clearing buffers */
  memset(RX_buf, 0, sizeof(RX_buf));
  memset(TX_buf, 0, sizeof(TX_buf));
  /* Enable interrupts */
  sei();
  /* set all pins of I/0 expander to output */
  IOEXP_IODIR(0);
  /* set lower 4 bits (leds) high*/
  IOEXP_datalatch(0x0f);
  _delay_ms(1000);
  /* set higher 4 bits (leds) high*/
  IOEXP_datalatch(0xf0);
  _delay_ms(1000);
  int16_t data = 0xaa;
  IOEXP_datalatch(data);  // set led
  while (1) {
    uint16_t adc_value = readADC_MCP3201();  // Read ADC value from
    MCP3201
    // int16_t data = 0xaa;
    IOEXP_datalatch(adc_value);  // set led
    USART_Transmit(adc_value);   // give same value to USART
    _delay_ms(100);
  }
}

void init() {
  USART_Init();
  /* Set SS, MOSI and SCK output, all others input */
  DDRB = (1 << SS) | (1 << MOSI) | (1 << SCK) | (1 << PORTB1);  // Use
  PORTB1 instead of SS2
      /* LED PIN */
      DDRB |= (1 << PORTB0);
  /* Set TXD as an output */
  DDRD = (1 << TXD);
  /* Set the slave select pin (Active low) */
  DISABLE_SS;
  DISABLE_SS2;
  /* Enable SPI, Master, set clock rate fosc/16 */
  SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);
}
```

```
// function to initialize UART
void USART_Init(void) {
  // Set baud rate:
  UBRR0 = 103;  // UBRR= Fosc /(16*9600) -1 =103.166= 103
  // enable receiver and transmitter
  UCSR0B |= (1 << RXEN0 | (1 << TXEN0));
  // Set frame format : 8 data 2 stop bit
  UCSR0C = (1 << USBS0) | (3 << UCSZ00);
}

void USART_Transmit(uint8_t data) {
  /* Wait for empty transmit buffer */
  while (!(UCSR0A & (1 << UDRE0)));
  /* Put data into buffer, sends the data */
  UDR0 = data;
}

/* spi data buffer load and send */
uint8_t spi_tranceiver(uint8_t data) {
  // Load data into the buffer
  SPDR = data;
  // Wait until transmission complete
  while (!(SPSR & (1 << SPIF)));
  // Return received data
  return (SPDR);
}

/* IO expander data direction set */
void IOEXP_IODIR(uint8_t data) {
  uint8_t r_data = 0;
  // Make slave select go low
  ENABLE_SS2;
  /* Send device address + r/w */
  spi_tranceiver((1 << 6) | WRITE);
  /* Send command */
  spi_tranceiver(IODIR);
  spi_tranceiver(data);
  // Make slave select go high
  DISABLE_SS2;
}

/* IO expander latch  set */
void IOEXP_datalatch(uint8_t data) {
  // Make slave select go low
  ENABLE_SS2;
  /* Send device address + r/w */
  spi_tranceiver((1 << 6) | WRITE);
  /* Send command */
  spi_tranceiver(OUTP_LATCH);
  spi_tranceiver(data);
  // Make slave select go high
  DISABLE_SS2;
}
```

```
// Function to perform SPI transaction and read data from MCP3201
uint16_t readADC_MCP3201(void) {
  uint16_t adc_value = 0;
  ENABLE_SS;
  // Start communication by pulling SS low
  // Send and receive 2 bytes (12-bit result from MCP3201) due to the system being
full duplex, send a byte and recieve one back

  uint8_t highByte =
          spi_tranceiver(0x00);            // First 8 bits (high byte)
  uint8_t lowByte = spi_tranceiver(0x00);  // Next 8 bits (low byte)
  // End communication by pulling SS high
  DISABLE_SS;
  // Combine the high and low byte into a 12-bit ADC value
  adc_value = (highByte << 8) | lowByte;  // shift 8 bytes to the left and combine
low and high byte
  adc_value =
      (adc_value >> 1) & 0x0FFF;  // Right shift by 1 to get rid of
      // the non important bits and mask 12 bits we want to read Page21, 6.1
MCP3201 datasheet
  return adc_value >> 4;
}
```

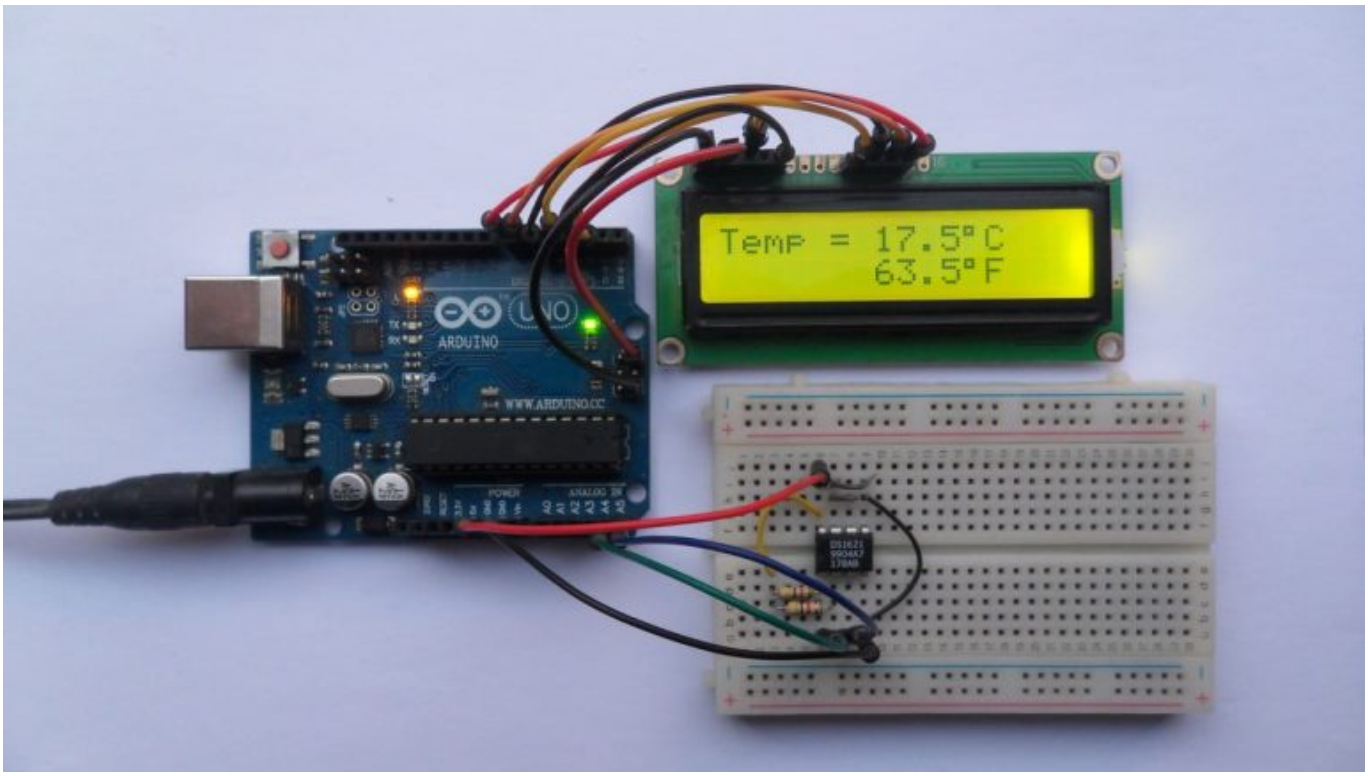
Week 6: I2C Final Assignment

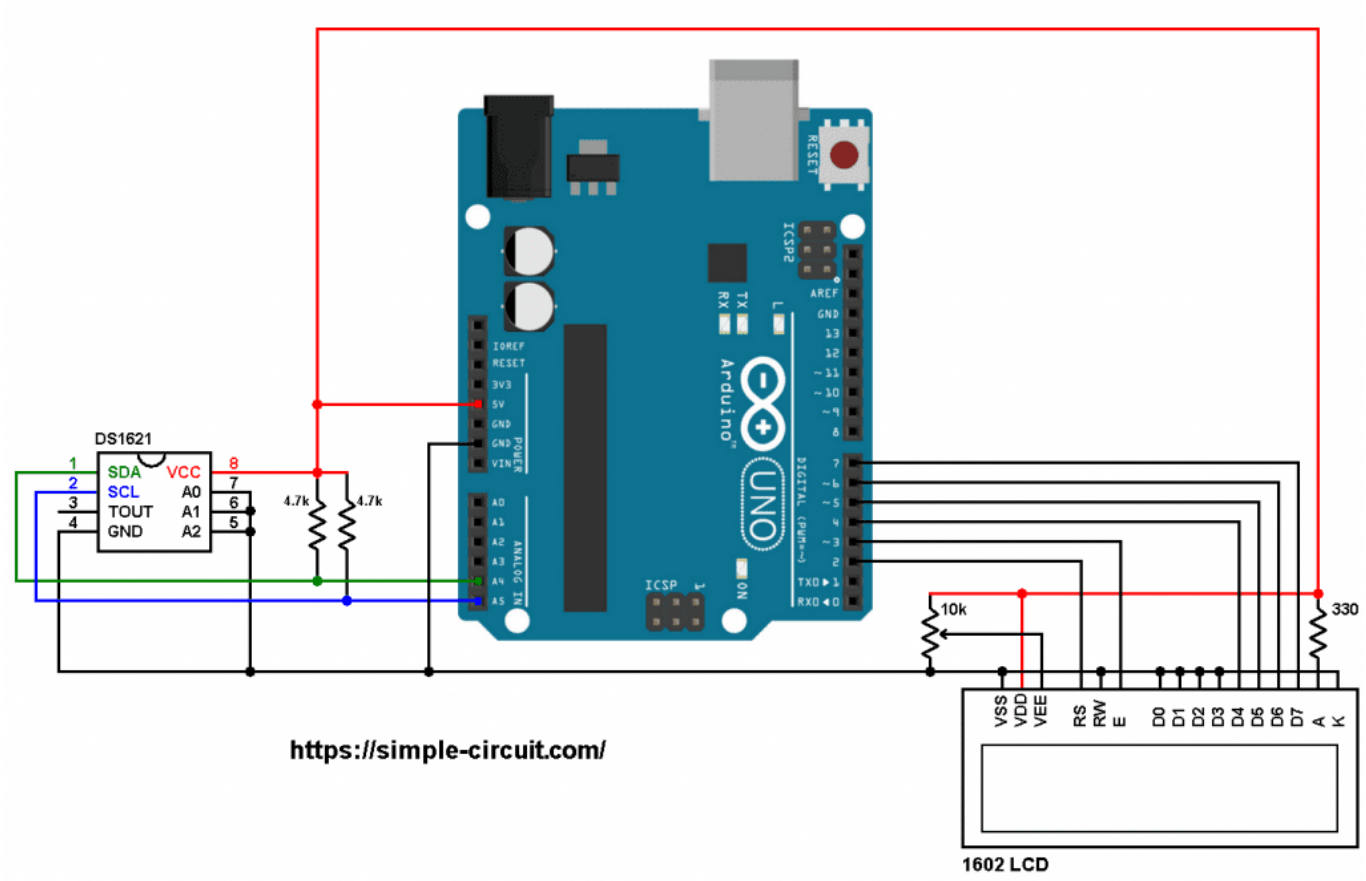
Date: 12-7-2025

DS1621

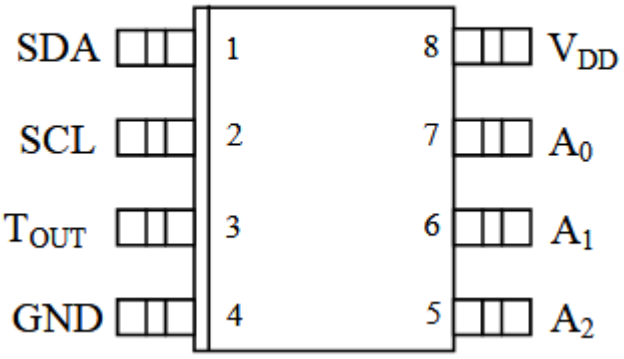
DS1621 Using Wire.h

Tutorial Link


Temp = 17.5°C
63.5°F
ARDUINO
UNO


DS1621
SDA
SCL
TOUT
GND
VCC
A0
A1
A2
4.7k
4.7k
10k
330
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
A
K
1602 LCD
https://simple-circuit.com/

PIN ASSIGNMENT



The **1602 LCD screen** (2 rows and 16 columns) is used to display temperature values in degrees Celsius and degrees Fahrenheit where:

- RS —> Arduino digital pin 2
- E —> Arduino digital pin 3
- D4 —> Arduino digital pin 4
- D5 —> Arduino digital pin 5
- D6 —> Arduino digital pin 6
- D7 —> Arduino digital pin 7
- VSS, RW, D0, D1, D2, D3 and K are connected to Arduino GND, VEE to the 10k ohm variable resistor (or potentiometer) output, VDD to Arduino 5V and A to Arduino 5V through 330 ohm resistor.
- VEE pin is used to control the contrast of the LCD. A (anode) and K (cathode) are the back light LED pins.

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>

// I2C Bit Rate
#define SCL_CLOCK 100000L

// DS1621 I2C Address
#define DS1621_ADDRESS 0x48

// I2C operation types
#define I2C_WRITE 0
#define I2C_READ  1

// Function Prototypes
void i2c_init(void);
void i2c_start(void);
void i2c_stop(void);
void i2c_write_byte(uint8_t data);
uint8_t i2c_read_byte(uint8_t ack);

void ds1621_init(void);
int16_t get_temperature(void);
```

```
void uart_init(unsigned int baud);
void uart_transmit(unsigned char data);
void uart_print(const char* str);

int main(void) {
    // Initialize UART for serial output
    uart_init(9600);

    // Initialize I2C
    i2c_init();

    // Initialize DS1621
    ds1621_init();

    char buffer[100];

    while(1) {
        // Get temperature
        int16_t c_temp = get_temperature();
        int16_t f_temp = c_temp * 9 / 5 + 320;

        // Format Celsius
        snprintf(buffer, sizeof(buffer), "Temp: %c%02d.%1d C, %c%02d.%1d F",
            (c_temp < 0 ? '-' : ' '), abs(c_temp/10), abs(c_temp%10),
            (f_temp < 0 ? '-' : ' '), abs(f_temp/10), abs(f_temp%10));

        // Print temperature
        uart_print(buffer);
        uart_print("\n");

        // Wait 1 second
        _delay_ms(1000);
    }

    return 0;
}

// Initialize I2C
void i2c_init(void) {
    TWSR = 0x00;  // Prescaler = 1
    TWBR = ((F_CPU / SCL_CLOCK) - 16) / 2;
}

// Send I2C start condition
void i2c_start(void) {
    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
    while (!(TWCR & (1<<TWINT)));
}

// Send I2C stop condition
void i2c_stop(void) {
    TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN);
}
```

```
// Write a byte to I2C
void i2c_write_byte(uint8_t data) {
    TWDR = data;
    TWCR = (1<<TWINT) | (1<<TWEN);
    while (!(TWCR & (1<<TWINT)));
}

// Read a byte from I2C
uint8_t i2c_read_byte(uint8_t ack) {
    TWCR = (1<<TWINT) | (1<<TWEN) | (ack ? (1<<TWEA) : 0);
    while (!(TWCR & (1<<TWINT)));
    return TWDR;
}

// Initialize DS1621
void ds1621_init(void) {
    i2c_start();
    i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
    i2c_write_byte(0xAC);  // Access Configuration Register
    i2c_write_byte(0x00);  // Continuous conversion
    i2c_stop();

    // Start temperature conversion
    i2c_start();
    i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
    i2c_write_byte(0xEE);  // Start conversion
    i2c_stop();
}

// Read temperature from DS1621
int16_t get_temperature(void) {
    i2c_start();
    i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
    i2c_write_byte(0xAA);  // Read temperature

    // Repeated start
    i2c_start();
    i2c_write_byte((DS1621_ADDRESS << 1) | I2C_READ);

    uint8_t t_msb = i2c_read_byte(1);  // Read MSB, send ACK
    uint8_t t_lsb = i2c_read_byte(0);  // Read LSB, send NACK
    i2c_stop();

    // Calculate temperature in tenths of a degree
    int16_t raw_t = (int8_t)t_msb << 1 | t_lsb >> 7;
    return raw_t * 10 / 2;
}

// UART Initialization
void uart_init(unsigned int baud) {
    unsigned int ubrr = F_CPU/16/baud-1;
    UBRR0H = (unsigned char)(ubrr>>8);
    UBRR0L = (unsigned char)ubrr;
    UCSR0B = (1<<TXEN0);
```

```
    UCSR0C = (1<<USBS0)|(3<<UCSZ00);
}

// UART Transmit
void uart_transmit(unsigned char data) {
    while (!(UCSR0A & (1<<UDRE0)));
    UDR0 = data;
}

// UART Print String
void uart_print(const char* str) {
    while (*str) {
        uart_transmit(*str++);
    }
}
```

DS1621 Using i2c function

```
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <util/delay.h>
// I2C Bit Rate
#define SCL_CLOCK 100000L
// DS1621 I2C Address
#define DS1621_ADDRESS 0x48
// I2C operation types
#define I2C_WRITE 0
#define I2C_READ 1
// Function Prototypes
void i2c_init(void);
void i2c_start(void);
void i2c_stop(void);
void i2c_write_byte(uint8_t data);
uint8_t i2c_read_byte(uint8_t ack);
void ds1621_init(void);
int16_t get_temperature(void);
void uart_init(unsigned int baud);
void uart_transmit(unsigned char data);
void uart_print(const char* str);
int main(void) {
  // Initialize UART for serial output
  uart_init(9600);
  // Initialize I2C
  i2c_init();
  // Initialize DS1621
  ds1621_init();
  char buffer[100];
  while (1) {
    // Get temperature
    int16_t c_temp = get_temperature();
```

```
    int16_t f_temp = c_temp * 9 / 5 + 320;
    // Format Celsius
    snprintf(buffer, sizeof(buffer), "Temp: %c%02d.%1d C, %c%02d.%1d F",
             (c_temp < 0 ? '-' : ' '), abs(c_temp / 10), abs(c_temp % 10),
             (f_temp < 0 ? '-' : ' '), abs(f_temp / 10), abs(f_temp % 10));
    // Print temperature
    uart_print(buffer);
    uart_print("\n");
    // Wait 1 second
    _delay_ms(1000);
  }
  return 0;
}

// Initialize I2C
void i2c_init(void) {
  TWSR = 0x00;  // Prescaler = 1
  TWBR = ((F_CPU / SCL_CLOCK) - 16) / 2;
}

// Send I2C start condition
void i2c_start(void) {
  TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
  while (!(TWCR & (1 << TWINT)));
}

// Send I2C stop condition
void i2c_stop(void) { TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN); }
// Write a byte to I2C
void i2c_write_byte(uint8_t data) {
  TWDR = data;
  TWCR = (1 << TWINT) | (1 << TWEN);
  while (!(TWCR & (1 << TWINT)));
}

// Read a byte from I2C
uint8_t i2c_read_byte(uint8_t ack) {
  TWCR = (1 << TWINT) | (1 << TWEN) | (ack ? (1 << TWEA) : 0);
  while (!(TWCR & (1 << TWINT)));
  return TWDR;
}

// Initialize DS1621
void ds1621_init(void) {
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
  i2c_write_byte(0xAC);  // Access Configuration Register
  i2c_write_byte(0x00);  // Continuous conversion
  i2c_stop();
  // Start temperature conversion
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
  i2c_write_byte(0xEE);  // Start conversion
  i2c_stop();
```

```
}

// Read temperature from DS1621
int16_t get_temperature(void) {
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
  i2c_write_byte(0xAA);  // Read temperature
  // Repeated start
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_READ);
  uint8_t t_msb = i2c_read_byte(1);  // Read MSB, send ACK
  uint8_t t_lsb = i2c_read_byte(0);  // Read LSB, send NACK
  i2c_stop();
  // Calculate temperature in tenths of a degree
  int16_t raw_t = (int8_t)t_msb << 1 | t_lsb >> 7;
  return raw_t * 10 / 2;
}

// UART Initialization
void uart_init(unsigned int baud) {
  unsigned int ubrr = F_CPU / 16 / baud - 1;
  UBRR0H = (unsigned char)(ubrr >> 8);
  UBRR0L = (unsigned char)ubrr;
  UCSR0B = (1 << TXEN0);
  UCSR0C = (1 << USBS0) | (3 << UCSZ00);
}

// UART Transmit
void uart_transmit(unsigned char data) {
  while (!(UCSR0A & (1 << UDRE0)));
  UDR0 = data;
}

// UART Print String
void uart_print(const char* str) {
  while (*str) {
    uart_transmit(*str++);
  }
}
```

HD44780

HD44780 with Wire.h

```
#include <Wire.h>  // For I2C communication
#define LCD_ADDR 0x27
#define LCD_ROWS 2
#define LCD_COLS 16
// I2C address of the LCD (0x27)
// LCD commands
#define LCD_CLEAR 0x01
```

```
#define LCD_HOME 0x02
#define LCD_ENTRY_MODE 0x04
#define LCD_DISPLAY_CONTROL 0x08
#define LCD_FUNCTION_SET 0x20  // HD44780 datasheet p24
// Display on/off control
#define LCD_DISPLAY_ON 0x04  // HD44780 datasheet p24
// Set DDRAM address
#define LCD_DDRAM_ADDRESS 0x80  // HD44780 datasheet p24
// Function set flags
#define LCD_4BIT_MODE 0x00  // HD44780 datasheet p27
#define LCD_2LINE 0x08      // HD44780 datasheet p27
#define LCD_5x8DOTS 0x00    // HD44780 datasheet p29
// Control bits
#define LCD_ENABLE_BIT 0x04  // Enable bit
#define LCD_BACKLIGHT 0x08   // Backlight control bit
// I2C functions
void i2c_write_byte(uint8_t data) {
  Wire.beginTransmission(LCD_ADDR);
  Wire.write(data | LCD_BACKLIGHT);  // Keep backlight on
  Wire.endTransmission();
}

void pulse_enable(uint8_t data) {
  i2c_write_byte(data | LCD_ENABLE_BIT);
  delayMicroseconds(1);
  // Enable pulse must be >450ns according to
  datasheet HD44780U p25 i2c_write_byte(data & ~LCD_ENABLE_BIT);
  delayMicroseconds(50);
  // Commands need >37us to settle according
  to datasheet HD44780U
}

// Send a command to the LCD  using 4 bit interface
void lcd_send_command(uint8_t command) {
  uint8_t high_nibble = command & 0xF0;
  uint8_t low_nibble = (command << 4) & 0xF0;
  pulse_enable(high_nibble);
  pulse_enable(low_nibble);
}

// Send data to the LCD
void lcd_send_data(uint8_t data) {
  uint8_t high_nibble = (data & 0xF0) | 0x01;  // RS = 1 for data
  uint8_t low_nibble = ((data << 4) & 0xF0) | 0x01;
  pulse_enable(high_nibble);
  pulse_enable(low_nibble);
}

// Initialize LCD
void lcd_init() {
  delay(50);
  // HD44780U p46
  // Wait for LCD to power up
  lcd_send_command(0x03);  // Initialize in 4-bit mode
```

```
  delay(5);
  lcd_send_command(0x03);
  delayMicroseconds(150);
  lcd_send_command(0x03);
  lcd_send_command(0x02);  // Set to 4-bit mode p46
  lcd_send_command(LCD_FUNCTION_SET | LCD_4BIT_MODE | LCD_2LINE | LCD_5x8DOTS);
  lcd_send_command(LCD_DISPLAY_CONTROL |LCD_DISPLAY_ON);  // Display on, cursor
off, no blink
  lcd_send_command(LCD_CLEAR);
  // Clear display
  lcd_send_command(LCD_ENTRY_MODE | 0x00);
  // Entry mode set: no shift
  delay(2);
}
// Clear LCD
void lcd_clear() {
  lcd_send_command(LCD_CLEAR);
  delay(2);
}
// Set cursor position
void lcd_set_cursor(uint8_t col, uint8_t row) {
  uint8_t row_offsets[] = {0x00, 0x40};
  // Entry mode
  // DDRAM addressHD44780U p12
  // lcd_send_command(LCD_DDRAM_ADDRESS | (col + row_offsets[row]));
}

// Print a string on the LCD
void lcd_print(const char *str) {
  while (*str) {
    lcd_send_data(*str++);
  }
}

void setup() {
  Wire.begin();
  lcd_init();
}

// Initialize I2C
// Initialize LCD
void loop() {
  char buffer[5];  // Temporary buffer to hold the integer as a string
  for (int i = 0; i < 20; i++) {
    lcd_clear();
    lcd_set_cursor(3, 0);
    lcd_print("Change code ");
    lcd_set_cursor(1, 1);
    lcd_print("World");
    lcd_set_cursor(8, 1);
    itoa(i, buffer, 10);
    lcd_print(buffer);
    delay(1000);
```

```
  }
}
```

HD44780 with DS1621

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <util/delay.h>
// Convert i to string with base 10
// Print the converted string
#define F_CPU 16000000UL   // 16 MHz
#define SCL_CLOCK 100000L  // I2C clock in Hz
// DS1621 working with 100kHz I2C clock page ... of the datasheet
// I2C operation types
#define I2C_WRITE 0
#define I2C_READ 1
// DS1621
#define DS1621_ADDRESS 0x48  // I2C address of the DS1621 (0x48)
// I2C Bit Rate
// LCD
#define LCD_ADDR 0x27  // I2C address of the LCD (0x27)
#define LCD_ROWS 2
// Number of rows on the LCD
#define LCD_COLS 16  // Number of columns on the LCD
// LCD commands
#define LCD_CLEAR 0x01
#define LCD_HOME 0x02
#define LCD_ENTRY_MODE 0x04
#define LCD_DISPLAY_CONTROL 0x08
#define LCD_FUNCTION_SET 0x20  // HD44780 datasheet p24 // 0b00100000
                               // Display on/off control
#define LCD_DISPLAY_ON 0x04    // HD44780 datasheet p24 // 0b00000100
// Set DDRAM address
#define LCD_DDRAM_ADDRESS 0x80  // HD44780 datasheet p24 // 0b10000000
// Function set flags
#define LCD_4BIT_MODE 0x00  // HD44780 datasheet p27
#define LCD_2LINE 0x08
// HD44780 datasheet p27 // 0b00001000
#define LCD_5x8DOTS 0x00  // HD44780 datasheet p29
// Control bits
#define LCD_ENABLE_BIT 0x04  // Enable bit
#define LCD_BACKLIGHT 0x08   // Backlight control bit
// Function Prototypes
// UART functions
void uart_init(unsigned int baud);
// Initialize UART
void uart_transmit(unsigned char data);  // Transmit data via UART
void uart_print(const char *str);
// Print string via UART
```

```
// I2C functions
void i2c_init(void);
void i2c_start(void);
void i2c_stop(void);
// Initialize I2C
// Start I2C communication
// Stop I2C communication
void i2c_write_byte(uint8_t data);    // Write byte to I2C
uint8_t i2c_read_byte(uint8_t ack);  // Read byte from I2C
void i2c_write_byte_lcd(uint8_t data);
void pulse_enable(uint8_t data);
void lcd_send_command(uint8_t command);
void lcd_send_data(uint8_t data);
void lcd_init(void);
void lcd_clear(void);
void lcd_set_cursor(uint8_t col, uint8_t row);
void lcd_print(const char *str);
// DS1621 functions
void ds1621_init(void);
int16_t get_temperature(void);
// Initialize I2C
void i2c_init(void) {
  TWSR = 0x00;  // Prescaler = 1
  TWBR = ((F_CPU / SCL_CLOCK) - 16) / 2;
}

// Send I2C start condition
void i2c_start(void) {
  // TWCR: TWI Control Register
  // TWINT: TWI Interrupt Flag
  // TWSTA: TWI Start Condition Bit
  TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
  // Wait for TWINT Flag set. This indicates that the start condition has been
transmitted.
  while (!(TWCR & (1 << TWINT)));
}

// Send I2C stop condition
void i2c_stop(void) {
  // TWCR: TWI Control Register
  // TWINT: TWI Interrupt Flag
  // TWSTO: TWI Stop Condition Bit
  // TWEN: TWI Enable Bit
  TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
}

// Write a byte to I2C
void i2c_write_byte(uint8_t data) {
  // TWDR: TWI Data Register
  TWDR = data;
  // TWCR: TWI Control Register
  // TWINT: TWI Interrupt Flag
  // TWEN: TWI Enable Bit
  TWCR = (1 << TWINT) | (1 << TWEN);
```

```
  // Wait for TWINT Flag set. This indicates that the data has been transmitted,
and ACK / NACK has been
  received.while (!(TWCR & (1 << TWINT)));
}
// Read a byte from I2C
uint8_t i2c_read_byte(uint8_t ack) {
  // TWCR: TWI Control Register
  // TWINT: TWI Interrupt Flag
  // TWEA: TWI Enable Acknowledge Bit
  // TWEN: TWI Enable Bit
  TWCR = (1 << TWINT) | (1 << TWEN) | (ack ? (1 << TWEA) : 0);
  while (!(TWCR & (1 << TWINT)));
  return TWDR;
}

// Initialize DS1621
void ds1621_init(void) {
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
  i2c_write_byte(0xAC);  // Access Configuration Register
  i2c_write_byte(0x00);  // Continuous conversion
  i2c_stop();
  // Start temperature conversion
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
  i2c_write_byte(0xEE);  // Start conversion
  i2c_stop();
}

// Read temperature from DS1621
int16_t get_temperature(void) {
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
  i2c_write_byte(0xAA);  // Read temperature
  // Repeated start
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_READ);
  uint8_t t_msb = i2c_read_byte(1);  // Read MSB, send ACK
  uint8_t t_lsb = i2c_read_byte(0);  // Read LSB, send NACK
  i2c_stop();
  // Calculate temperature in tenths of a degree
  int16_t raw_t = (int8_t)t_msb << 1 | t_lsb >> 7;
  return raw_t * 10 / 2;
}

// UART Initialization
void uart_init(unsigned int baud) {
  unsigned int ubrr = F_CPU / 16 / baud - 1;
  UBRR0H = (unsigned char)(ubrr >> 8);
  UBRR0L = (unsigned char)ubrr;
  UCSR0B = (1 << TXEN0);
  UCSR0C = (1 << USBS0) | (3 << UCSZ00);
}
```

```
// UART Transmit
void uart_transmit(unsigned char data) {
  while (!(UCSR0A & (1 << UDRE0)));
  UDR0 = data;
}

// UART Print String
void uart_print(const char *str) {
  while (*str) {
    uart_transmit(*str++);
  }
}

void i2c_write_byte_lcd(uint8_t data) {
  i2c_start();
  i2c_write_byte((LCD_ADDR << 1) | I2C_WRITE);
  i2c_write_byte(data | LCD_BACKLIGHT);
  i2c_stop();
}

void pulse_enable(uint8_t data) {
  i2c_write_byte_lcd(data | LCD_ENABLE_BIT);
  _delay_ms(1);  // Enable pulse must be >450ns according to datasheet HD44780U
p25
  i2c_write_byte_lcd(data & ~LCD_ENABLE_BIT);
  _delay_ms(50);  // Commands need >37us to settle according to datasheet HD44780U
}

// Send a command to the LCD  using 4 bit interface
void lcd_send_command(uint8_t command) {
  uint8_t high_nibble = command & 0xF0;
  uint8_t low_nibble = (command << 4) & 0xF0;
  pulse_enable(high_nibble);
  pulse_enable(low_nibble);
}

// Send data to the LCD
void lcd_send_data(uint8_t data) {
  uint8_t high_nibble = (data & 0xF0) | 0x01;  // RS = 1 for data
  uint8_t low_nibble = ((data << 4) & 0xF0) | 0x01;
  pulse_enable(high_nibble);
  pulse_enable(low_nibble);
}

// Initialize LCD
void lcd_init() {
  _delay_ms(50);
  // HD44780U p46
  // Wait for LCD to power up
  lcd_send_command(0x03);  // Initialize in 4-bit mode p46
  _delay_ms(5);
  lcd_send_command(0x03);
  _delay_ms(150);
  lcd_send_command(0x03);
```

```
  lcd_send_command(0x02);  // Set to 4-bit mode
  lcd_send_command(LCD_FUNCTION_SET | LCD_4BIT_MODE | LCD_2LINE | LCD_5x8DOTS);
  lcd_send_command(LCD_DISPLAY_CONTROL | LCD_DISPLAY_ON);  // Display on, cursor
off, no blink
  lcd_send_command(LCD_CLEAR); // Clear display
  lcd_send_command(LCD_ENTRY_MODE | 0x00); // Entry mode set: no shift
  _delay_ms(2);
}

// Clear LCD
void lcd_clear() {
  lcd_send_command(LCD_CLEAR);
  _delay_ms(2);
}

// Entry mode
// Set cursor position
void lcd_set_cursor(uint8_t col, uint8_t row) {
  uint8_t row_offsets[] = {0x00, 0x40};  // DDRAM address HD44780U p12
  lcd_send_command(LCD_DDRAM_ADDRESS | (col + row_offsets[row]));
}
// Print a string on the LCD
void lcd_print(const char *str) {
  while (*str) {
    lcd_send_data(*str++);
  }
}

int main(void) {
  // Initialize UART for serial output
  uart_init(9600);
  // Initialize I2C
  i2c_init();
  // Initialize DS1621
  ds1621_init();
  lcd_init();  // Initialize LCD
  lcd_clear();
  int16_t c_temp = get_temperature();
  int16_t f_temp = c_temp * 9 / 5 + 320;
  char buffer_F[20];
  char buffer_C[20];
  char temperature[10];
  // Format Celsius
  snprintf(buffer_C, sizeof(buffer_C), "Temp: %c%02d.%1d C",
           (c_temp < 0 ? '-' : ' '), abs(c_temp / 10), abs(c_temp % 10));
  // Format F
  snprintf(buffer_F, sizeof(buffer_F), "Temp: %c%02d.%1d F",
           (f_temp < 0 ? '-' : ' '), abs(f_temp / 10), abs(f_temp % 10));
  snprintf(temperature, sizeof(temperature), "%c%02d.%1d",
           (c_temp < 0 ? '-' : ' '), abs(c_temp / 10), abs(c_temp % 10));
  // Print temperature
  uart_print(temperature);
  uart_print("\n");
  lcd_set_cursor(1, 0);
```

```
    lcd_print(buffer_C);
    lcd_set_cursor(1, 1);
    lcd_print(buffer_F);
    while (1) {
      unsigned long currentMillis = millis();
      // Get temperature
      c_temp = get_temperature();
      f_temp = c_temp * 9 / 5 + 320;
      i2c_start();
      i2c_write_byte(MCP23017_ADDR << 1);  // Address with write bit
      i2c_write_byte(GPIOA);
      // Register address
      if (on) {
        i2c_write_byte(
            digitMap[(int)abs(f_temp / 10) - (f_temp / 100 * 10)]);  // Data A
        i2c_write_byte(digitMap[(int)abs(f_temp / 100)]);
        // Data B
      } else {
        i2c_write_byte(
            digitMap[(int)abs(c_temp / 10) - (c_temp / 100 * 10)]);  // Data A
        i2c_write_byte(digitMap[(int)abs(c_temp / 100)]);
        // Data B
      }
      i2c_stop();
      // Format Celsius
      snprintf(buffer_C, sizeof(buffer_C), "Temp: %c%02d.%1d C",
               (c_temp < 0 ? '-' : ' '), abs(c_temp / 10), abs(c_temp % 10));
      // Format F
      snprintf(buffer_F, sizeof(buffer_F), "Temp: %c%02d.%1d F",
               (f_temp < 0 ? '-' : ' '), abs(f_temp / 10), abs(f_temp % 10));
      snprintf(temperature, sizeof(temperature), "%c%02d.%1d",
               (c_temp < 0 ? '-' : ' '), abs(c_temp / 10), abs(c_temp % 10));
      if (currentMillis - previousMillis >= period) {
        lcd_clear();
        // Print temperature
        uart_print(temperature);
        uart_print("\n");
        lcd_set_cursor(1, 0);
        lcd_print(buffer_C);
        lcd_set_cursor(1, 1);
        lcd_print(buffer_F);
        previousMillis = currentMillis;
      }
    }
    return 0;
}
```

MCP23017 with 7 segments

```
#include "Wire.h"
#define MCP23017_ADDR 0x20  // I2C address of the MCP23017
```

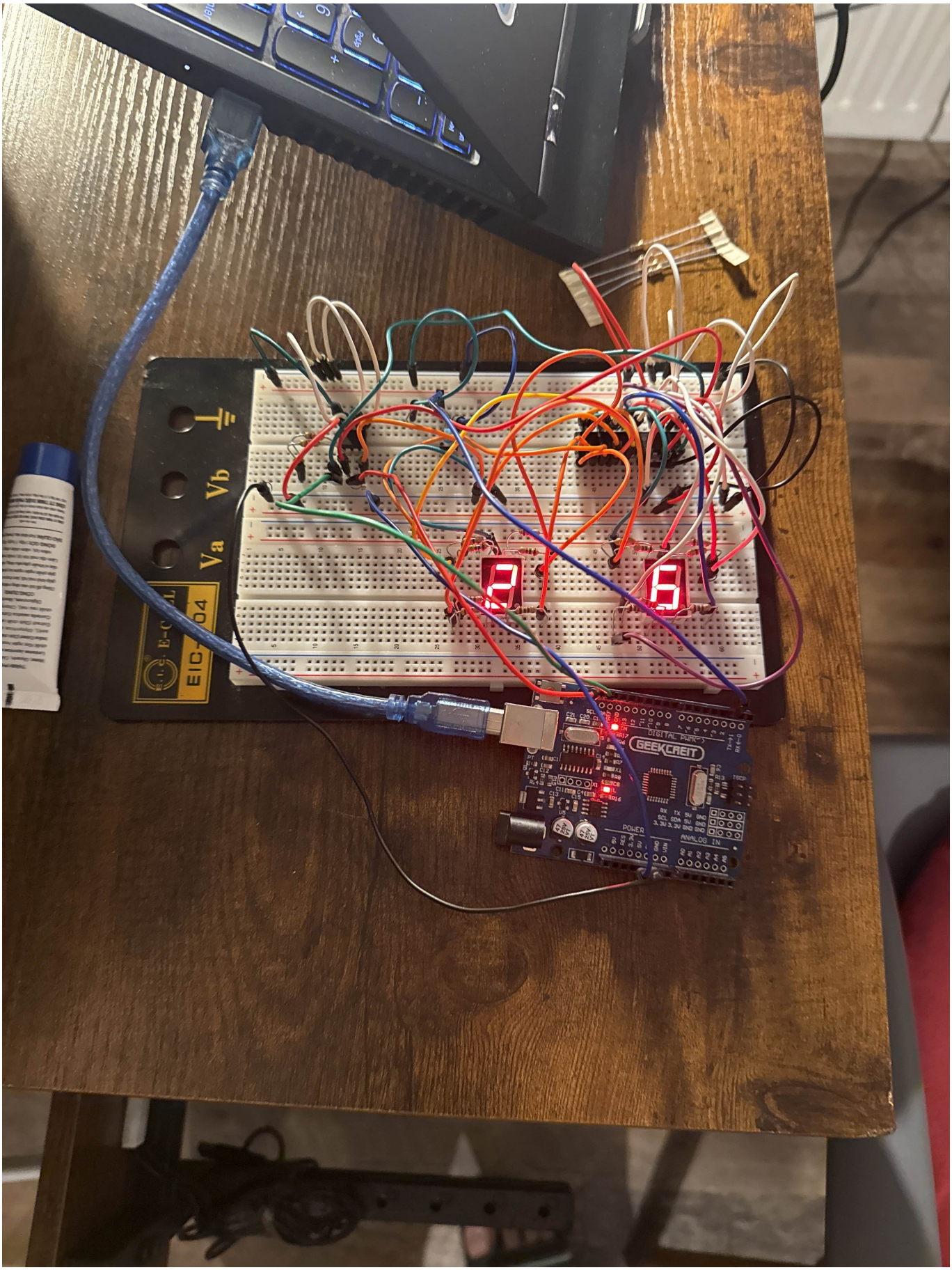
```
#define GPIOA 0x12
#define GPIOB 0x13
// Define bit patterns for digits 0-9 (assuming common cathode)
const uint8_t digitMap[10] = {
    0b00111111,                                          // 0
    0b00000110,                                          // 1
    Final assigment Combine all components 0b01011011,  // 2
    0b01001111,                                          // 3
    0b01100110,                                          // 4
    0b01101101,                                          // 5
    0b01111101,                                          // 6
    0b00000111,                                          // 7
    0b01111111,                                          // 8
    0b01101111                                           // 9
};
void displayDigit(uint8_t digit, uint8_t port) {
  if (digit > 9) return;  // Invalid digit, ignore
  Wire.beginTransmission(MCP23017_ADDR);
  Wire.write(port);             // GPIOA register (output for Port A)
  Wire.write(digitMap[digit]);  // Write segment pattern
  Wire.endTransmission();
}
void setup() {
  Wire.begin();  // wake up I2C bus
  // set I/O pins to outputs
  Wire.beginTransmission(0x20);
  Wire.write(0x00);  // IODIRA register
  Wire.write(0x00);  // set all of port A to outputs
  Wire.endTransmission();
  Wire.beginTransmission(0x20);
  Wire.write(0x01);  // IODIRB register
  Wire.write(0x00);  // set all of port B to outputs
  Wire.endTransmission();
}
void loop() {
  for (uint8_t i = 0; i < 10; i++) {
    displayDigit(i, GPIOA);  // Show digit on 7-segment display
    displayDigit(i, GPIOB);  // Show digit on 7-segment display
    delay(1000);             // Hold the digit for 1 second
  }
}
```

Final asignment

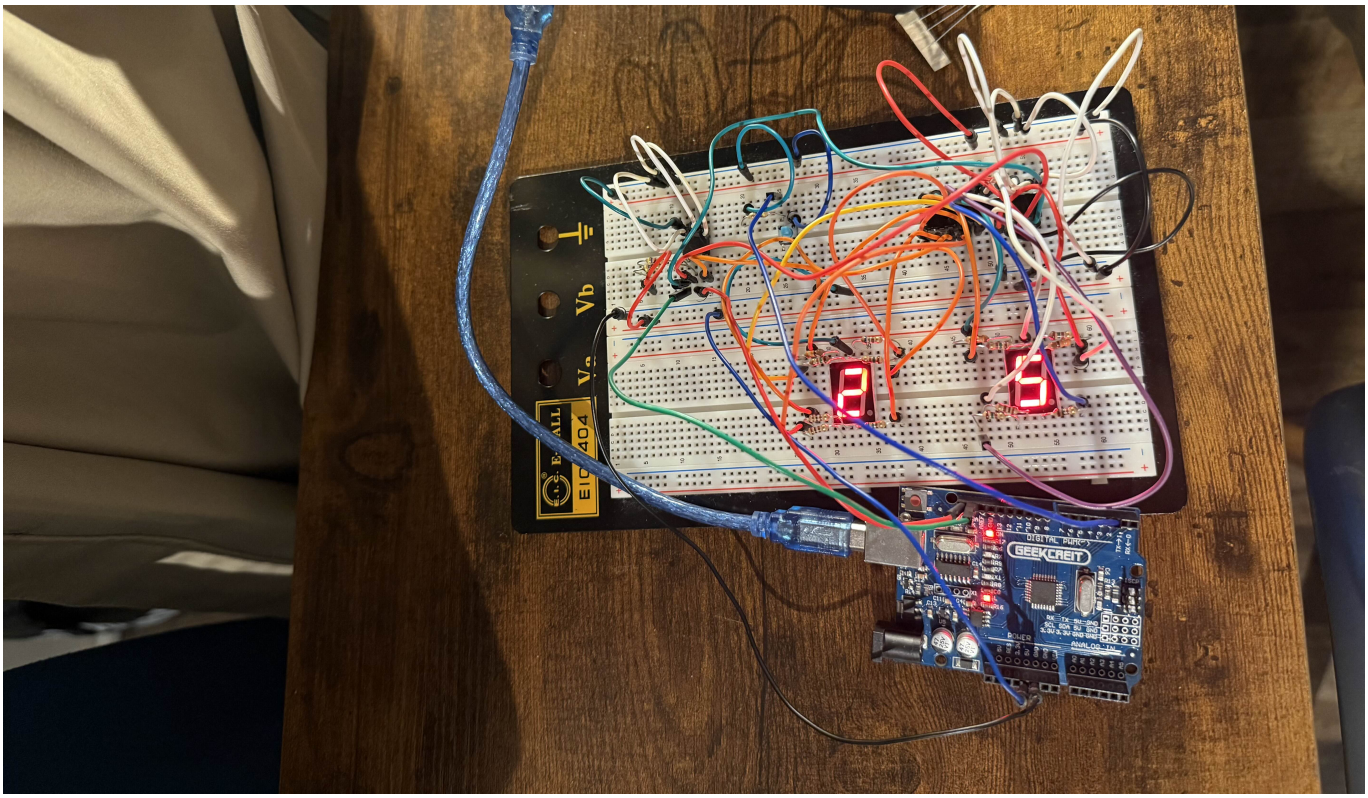
Combine all components

- Arduino Uno R3
- DS1621 x1
- MCP23017 x1
- HD44780 x1
- 4k7 Ohm Resistors x3
- 220 Ohm Resistors x14

- 7 Segments x2
- Button x1


Vb
Va
GEEKCREIT
POWER
ANALOG IN


Lenovo



final_code.ino

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <avr/interrupt.h>

#define F_CPU 16000000UL    // 16 MHz
#define SCL_CLOCK 100000L  // I2C clock in Hz
// DS1621 working with 100kHz I2C clock page ... of the datasheet

// I2C operation types
#define I2C_WRITE 0
#define I2C_READ 1

// DS1621
#define DS1621_ADDRESS 0x48  // I2C address of the DS1621 (0x48)
// I2C Bit Rate

// MCP23017
#define MCP23017_ADDR 0x20  // I2C address of the MCP23017 (shifted left by 1 for
R/W bit)
#define IODIRA 0x00         // Port A I/O direction register
#define IODIRB 0x01         // Port B I/O direction register
#define GPIOA 0x12          // Port A register

// Define bit patterns for digits 0-9 (assuming common cathode)
const uint8_t digitMap[10] = {
  0b00111111,  // 0 (segments a, b, c, d, e, f on)
  0b00000110,  // 1 (segments b, c on)
```

```
  0b01011011,  // 2 (segments a, b, d, e, g on)
  0b01001111,  // 3 (segments a, b, c, d, g on)
  0b01100110,  // 4 (segments b, c, f, g on)
  0b01101101,  // 5 (segments a, c, d, f, g on)
  0b01111101,  // 6 (segments a, c, d, e, f, g on)
  0b00000111,  // 7 (segments a, b, c on)
  0b01111111,  // 8 (all segments on)
  0b01101111   // 9 (segments a, b, c, d, f, g on)
};

// Function Prototypes
// UART functions
void uart_init(unsigned int baud);       // Initialize UART
void uart_transmit(unsigned char data);  // Transmit data via UART
void uart_print(const char *str);        // Print string via UART

// Timer functions
void init_timer0();  // Initialize Timer0
unsigned long millis();

// I2C functions
void i2c_init(void);                 // Initialize I2C
void i2c_start(void);                // Start I2C communication
void i2c_stop(void);                 // Stop I2C communication
void i2c_write_byte(uint8_t data);   // Write byte to I2C
uint8_t i2c_read_byte(uint8_t ack);  // Read byte from I2C

void i2c_write_byte_lcd(uint8_t data);
void pulse_enable(uint8_t data);


// MCP23017 functions
void mcp23017_write(uint8_t reg, uint8_t data);
void displayDigit(uint8_t digit, uint8_t port);
void init_seven_segment(void);

// DS1621 functions
void ds1621_init(void);
int16_t get_temperature(void);

// Initialize I2C
void i2c_init(void) {
  TWSR = 0x00;  // Prescaler = 1
  TWBR = ((F_CPU / SCL_CLOCK) - 16) / 2;
}

// Send I2C start condition
void i2c_start(void) {
  // TWCR: TWI Control Register
  // TWINT: TWI Interrupt Flag
  // TWSTA: TWI Start Condition Bit
  TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
  // Wait for TWINT Flag set. This indicates that the start condition has been
transmitted.
```

```
  while (!(TWCR & (1 << TWINT)))
    ;
}

// Send I2C stop condition
void i2c_stop(void) {
  // TWCR: TWI Control Register
  // TWINT: TWI Interrupt Flag
  // TWSTO: TWI Stop Condition Bit
  // TWEN: TWI Enable Bit
  TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
}

// Write a byte to I2C
void i2c_write_byte(uint8_t data) {
  // TWDR: TWI Data Register
  TWDR = data;
  // TWCR: TWI Control Register
  // TWINT: TWI Interrupt Flag
  // TWEN: TWI Enable Bit
  TWCR = (1 << TWINT) | (1 << TWEN);
  // Wait for TWINT Flag set. This indicates that the data has been transmitted,
and ACK/NACK has been received.
  while (!(TWCR & (1 << TWINT)))
    ;
}

// Read a byte from I2C
uint8_t i2c_read_byte(uint8_t ack) {
  // TWCR: TWI Control Register
  // TWINT: TWI Interrupt Flag
  // TWEA: TWI Enable Acknowledge Bit
  // TWEN: TWI Enable Bit
  TWCR = (1 << TWINT) | (1 << TWEN) | (ack ? (1 << TWEA) : 0);
  while (!(TWCR & (1 << TWINT)))
    ;
  return TWDR;
}

// Initialize DS1621
void ds1621_init(void) {
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
  i2c_write_byte(0xAC);  // Access Configuration Register
  i2c_write_byte(0x00);  // Continuous conversion
  i2c_stop();

  // Start temperature conversion
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
  i2c_write_byte(0xEE);  // Start conversion
  i2c_stop();
}
```

```
// Read temperature from DS1621
int16_t get_temperature(void) {
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_WRITE);
  i2c_write_byte(0xAA);  // Read temperature

  // Repeated start
  i2c_start();
  i2c_write_byte((DS1621_ADDRESS << 1) | I2C_READ);

  uint8_t t_msb = i2c_read_byte(1);  // Read MSB, send ACK
  uint8_t t_lsb = i2c_read_byte(0);  // Read LSB, send NACK
  i2c_stop();

  // Calculate temperature in tenths of a degree
  int16_t raw_t = (int8_t)t_msb << 1 | t_lsb >> 7;
  return raw_t * 10 / 2;
}

// UART Initialization
void uart_init(unsigned int baud) {
  unsigned int ubrr = F_CPU / 16 / baud - 1;
  UBRR0H = (unsigned char)(ubrr >> 8);
  UBRR0L = (unsigned char)ubrr;
  UCSR0B = (1 << TXEN0);
  UCSR0C = (1 << USBS0) | (3 << UCSZ00);
}

// UART Transmit
void uart_transmit(unsigned char data) {
  while (!(UCSR0A & (1 << UDRE0)))
    ;
  UDR0 = data;
}

// UART Print String
void uart_print(const char *str) {
  while (*str) {
    uart_transmit(*str++);
  }
}

void i2c_write_byte_lcd(uint8_t data) {
  i2c_start();
  i2c_write_byte(I2C_WRITE);
  i2c_write_byte(data);
  i2c_stop();
}

void pulse_enable(uint8_t data) {
  i2c_write_byte_lcd(data);
  _delay_ms(1);  // Enable pulse must be >450ns according to datasheet HD44780U
p25
  i2c_write_byte_lcd(data);
```

```
  _delay_ms(50);  // Commands need >37us to settle according to datasheet HD44780U
}

// Write to MCP23017 register
void mcp23017_write(uint8_t reg, uint8_t data) {
  i2c_start();
  i2c_write_byte(MCP23017_ADDR << 1);  // Address with write bit
  i2c_write_byte(reg);                 // Register address
  i2c_write_byte(data);                // Data
  i2c_stop();
}

// Display digit on specified port
void displayDigit(uint8_t digit, uint8_t port) {
  if (digit > 9)
    return;  // Invalid digit, ignore
  mcp23017_write(port, digitMap[digit]);
}

// Initialize the MCP23017 for 7-segment displays
void init_seven_segment(void) {
  // Configure both ports as outputs
  mcp23017_write(IODIRA, 0x00);
  _delay_ms(20);
  mcp23017_write(IODIRB, 0x00);
  _delay_ms(20);
  i2c_start();
  i2c_write_byte(MCP23017_ADDR << 1);  // Address with write bit
  i2c_write_byte(GPIOA);               // Register address
  i2c_write_byte(digitMap[0]);         // Data A
  i2c_write_byte(digitMap[0]);         // Data B
  i2c_stop();
}

// Interrupt Service Routine for Timer0 compare match
unsigned long previousMillis;
volatile int on = 0;  // shared volatile variable,
// make sure the compiler does not optimize it away

volatile unsigned long timer0_millis = 0;
unsigned long period = 60000;  // 60 second


void init_timer0() {
  // Set Timer0 to CTC mode
  TCCR0A = (1 << WGM01);

  // Set prescaler to 64
  TCCR0B = (1 << CS01) | (1 << CS00);

  // Set compare value for 1ms interrupt at 16MHz
  OCR0A = 249;

  // Enable Timer0 compare match interrupt
```

```
  TIMSK0 = (1 << OCIE0A);
}

ISR(TIMER0_COMPA_vect) {
  timer0_millis++;
}

unsigned long millis() {
  unsigned long m;
  uint8_t oldSREG = SREG;

  // Disable interrupts while reading timer0_millis
  cli();
  m = timer0_millis;
  SREG = oldSREG;

  return m;
}

int main(void) {
  // Initialize UART for serial output
  uart_init(9600);

  // Initialize I2C
  i2c_init();

  // Initialize DS1621
  ds1621_init();

  // Configure MCP23017 ports as outputs
  init_seven_segment();

  // port D as inputs ( why? ) - all the pins in Port D are set to INPUT mode.
  DDRD = 0;
  /*
      EIMSK - External Interrupt Mask Register
      see data sheet: which interrupt is enabled?
      INT0 - 2 0x0002 INT0 External Interrupt Request 0
      Set HIGH to INT0 - Pin 2
    */
  EIMSK = (1 << INT0);
  /*
      EICRA - External Interrupt Control Register A
      see data sheet: when does the interrupt react?
      External Interrupt Control Register A (EICRA) define whether the external
interrupt is activated
      on rising and/or falling edge of the INT0 pin or level sensed. Activity on
the pin will cause an
      interrupt request even if INT0 is configured as an output.
    */
  EICRA = 0b00000011;
  /*
      why ?
      sei() - SEts the global interrupt flag
```

```
    */
  init_timer0();  // Add this line
  sei();

  int16_t c_temp = get_temperature();
  int16_t f_temp = c_temp * 9 / 5 + 320;

  char buffer_F[20];
  char buffer_C[20];
  char temperature[10];

  // Format Celsius
  snprintf(buffer_C, sizeof(buffer_C), "Temp: %c%02d.%1d C",
           (c_temp < 0 ? '-' : ' '), abs(c_temp / 10), abs(c_temp % 10));
  // Format F
  snprintf(buffer_F, sizeof(buffer_F), "Temp: %c%02d.%1d F",
           (f_temp < 0 ? '-' : ' '), abs(f_temp / 10), abs(f_temp % 10));

  snprintf(temperature, sizeof(temperature), "%c%02d.%1d", (c_temp < 0 ? '-' : '
'), abs(c_temp / 10), abs(c_temp % 10));

  // Print temperature
  uart_print(temperature);
  uart_print("\n");

  while (1) {

    unsigned long currentMillis = millis();
    // Get temperature

    c_temp = get_temperature();
    f_temp = c_temp * 9 / 5 + 320;

    i2c_start();
    i2c_write_byte(MCP23017_ADDR << 1);  // Address with write bit
    i2c_write_byte(GPIOA);               // Register address
    if (on) {
      i2c_write_byte(digitMap[(int)abs(f_temp / 10) - (f_temp / 100 * 10)]);  //
Data A
      i2c_write_byte(digitMap[(int)abs(f_temp / 100)]);                       //
Data B
    } else {
      i2c_write_byte(digitMap[(int)abs(c_temp / 10) - (c_temp / 100 * 10)]);  //
Data A
      i2c_write_byte(digitMap[(int)abs(c_temp / 100)]);                       //
Data B
    }
    i2c_stop();

    // Format Celsius
    snprintf(buffer_C, sizeof(buffer_C), "Temp: %c%02d.%1d C", (c_temp < 0 ? '-' :
' '), abs(c_temp / 10), abs(c_temp % 10));
    // Format F
    snprintf(buffer_F, sizeof(buffer_F), "Temp: %c%02d.%1d F", (f_temp < 0 ? '-' :
```

```
' '), abs(f_temp / 10), abs(f_temp % 10));

    snprintf(temperature, sizeof(temperature), "%c%02d.%1d", (c_temp < 0 ? '-' : '
'), abs(c_temp / 10), abs(c_temp % 10));

    if (currentMillis - previousMillis >= period) {
      // Print temperature
      uart_print(temperature);
      uart_print("\n");
      previousMillis = currentMillis;
    }
  }

  return 0;
}


ISR(INT0_vect) {
  _delay_ms(20);  // short delay to avoid bouncing
  if (on)         // toggle on
    on = 0;
  else
    on = 1;
}
```

Database

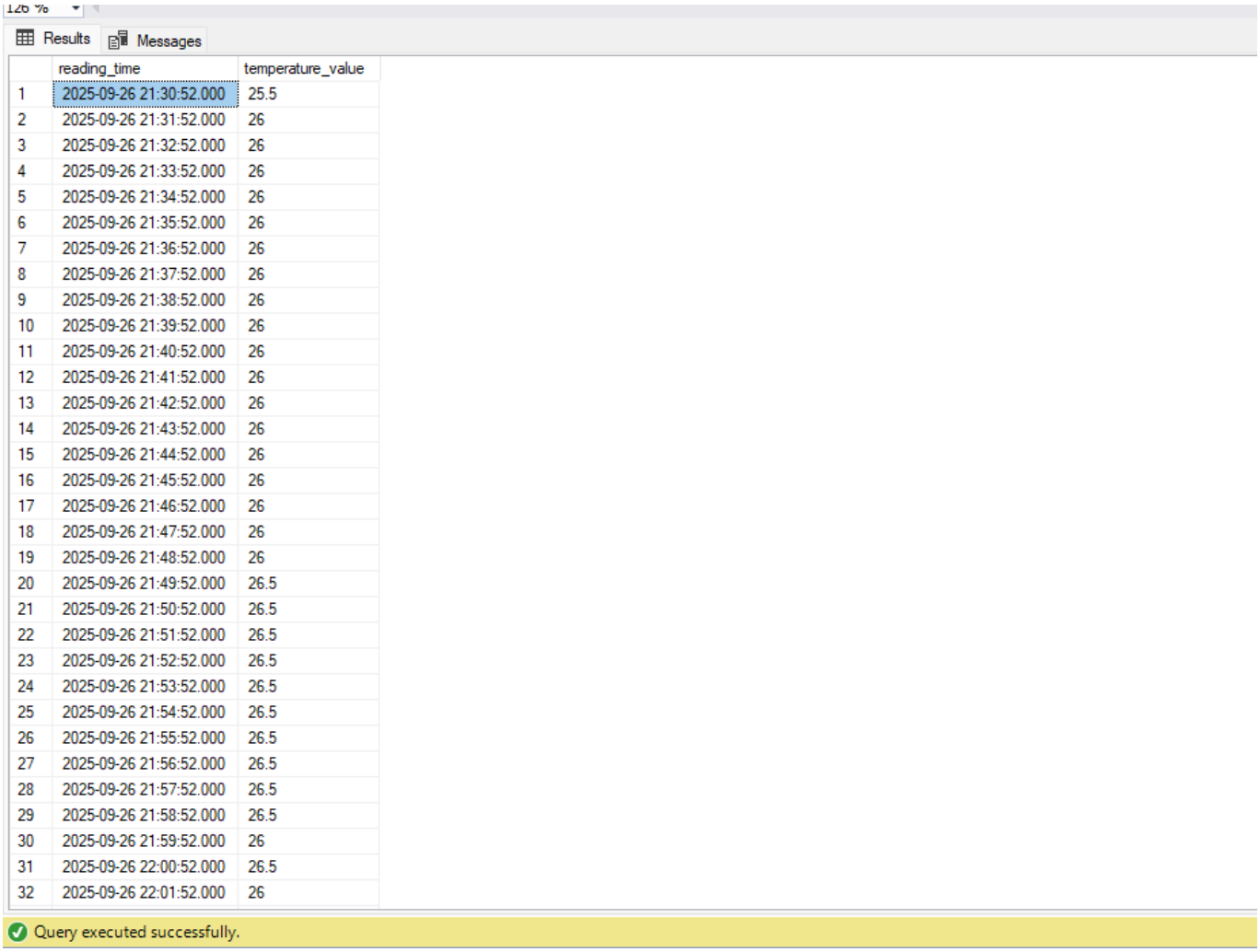
Use Microsoft SQL to create database and table temperature

```
 USE master;
 GO
IF NOT EXISTS (SELECT name FROM sys.databases WHERE name =
 'final_project_microcontroller')
 BEGIN
 CREATE DATABASE final_project_microcontroller;
 END
 GO
 USE final_project_microcontroller;
 GO
 CREATE TABLE temperature (
 reading_time DATETIME PRIMARY KEY,
 temperature_value FLOAT NOT NULL
 );
 GO

 use final_project_microcontroller
 Go
 Select * from temperature
 truncate table temperature
```

And I use python code to insert the data from the sensor to the database

```
import pyodbc
import serial
from datetime import datetime

server = 'Felix-Nguyen\SQLEXPRESS'
database = 'final_project_microcontroller'


conn = pyodbc.connect(
    f'DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATABASE=
{database};Trusted_Connection=yes;'
)
cursor = conn.cursor()

ser = serial.Serial('COM7', 9600, timeout=1)

print("Listening for data...")

while True:
    try:
        data = ser.readline().decode().strip()
        if data:
            reading_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            temperature = float(data)
            cursor.execute("INSERT INTO dbo.temperature (reading_time,
temperature_value) VALUES (?, ?)",
    (reading_time, temperature))
            conn.commit()
            print(f"Inserted reading time: {reading_time}")
            print(f"Inserted temperature: {temperature}")

    except Exception as e:
        print(f"Error: {e}")
        break

cursor.close()
conn.close()
ser.close()
```

Result



Results Messages

	reading_time	temperature_value
1	2025-09-26 21:30:52.000	25.5
2	2025-09-26 21:31:52.000	26
3	2025-09-26 21:32:52.000	26
4	2025-09-26 21:33:52.000	26
5	2025-09-26 21:34:52.000	26
6	2025-09-26 21:35:52.000	26
7	2025-09-26 21:36:52.000	26
8	2025-09-26 21:37:52.000	26
9	2025-09-26 21:38:52.000	26
10	2025-09-26 21:39:52.000	26
11	2025-09-26 21:40:52.000	26
12	2025-09-26 21:41:52.000	26
13	2025-09-26 21:42:52.000	26
14	2025-09-26 21:43:52.000	26
15	2025-09-26 21:44:52.000	26
16	2025-09-26 21:45:52.000	26
17	2025-09-26 21:46:52.000	26
18	2025-09-26 21:47:52.000	26
19	2025-09-26 21:48:52.000	26
20	2025-09-26 21:49:52.000	26.5
21	2025-09-26 21:50:52.000	26.5
22	2025-09-26 21:51:52.000	26.5
23	2025-09-26 21:52:52.000	26.5
24	2025-09-26 21:53:52.000	26.5
25	2025-09-26 21:54:52.000	26.5
26	2025-09-26 21:55:52.000	26.5
27	2025-09-26 21:56:52.000	26.5
28	2025-09-26 21:57:52.000	26.5
29	2025-09-26 21:58:52.000	26.5
30	2025-09-26 21:59:52.000	26
31	2025-09-26 22:00:52.000	26.5
32	2025-09-26 22:01:52.000	26

Query executed successfully.

API

I use FastAPI from Python to create the API for this assessment.

```
from fastapi import FastAPI, HTTPException, Query, Depends
from sqlalchemy import Column, Float, DateTime, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, Session
from typing import Optional
import math
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

# --- CORS middleware ---
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://127.0.0.1:5500", "http://localhost:5500"],  # frontend
origins
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

```
# --- Database setup (SQL Server, pyodbc) ---
DATABASE_URL = (
    "mssql+pyodbc://@FELIX-NGUYEN\\SQLEXPRESS/final_project_microcontroller"
    "?driver=ODBC Driver 17 for SQL Server&Trusted_Connection=yes"
)

engine = create_engine(DATABASE_URL, pool_pre_ping=True)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

class Temperature(Base):
    __tablename__ = "temperature"
    reading_time = Column(DateTime, primary_key=True, nullable=False)
    temperature_value = Column(Float, nullable=False)

# Dependency: get DB session
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()


# GET latest temperature
@app.get("/api/temperature/latest")
def get_latest_temperature(db: Session = Depends(get_db)):
    latest = (
        db.query(Temperature)
        .order_by(Temperature.reading_time.desc())
        .first()
    )

    if not latest:
        raise HTTPException(status_code=404, detail="No temperature readings
found.")

    return {
        "reading_time": latest.reading_time,
        "temp_c": latest.temperature_value,
        "temp_f": latest.temperature_value * 9 / 5 + 32,
    }


# GET temperature list with filters + pagination
@app.get("/api/temperature")
def get_temperatures(
    start_time: Optional[str] = Query(None),
    end_time: Optional[str] = Query(None),
    page: int = Query(1, gt=0),
    page_size: int = Query(10, gt=0),
    db: Session = Depends(get_db),
):
    query = db.query(Temperature)
```

```
    # Apply filters
    if start_time and not end_time:
        query = query.filter(Temperature.reading_time == start_time)

    if end_time and not start_time:
        query = query.filter(Temperature.reading_time == end_time)

    if start_time and end_time:
        query = query.filter(
            Temperature.reading_time >= start_time,
            Temperature.reading_time <= end_time,
        )

    # Pagination
    total_items = query.count()
    total_pages = math.ceil(total_items / page_size) if total_items else 0

    if page > total_pages and total_items > 0:
        raise HTTPException(
            status_code=404,
            detail=f"Page {page} does not exist. Total pages: {total_pages}.",
        )

    rows = (
        query.order_by(Temperature.reading_time.desc())
        .offset((page - 1) * page_size)
        .limit(page_size)
        .all()
    )

    return {
        "total_items": total_items,
        "total_pages": total_pages,
        "current_page": page,
        "page_size": page_size,
        "data": [
            {
                "reading_time": r.reading_time,
                "temperature_value": r.temperature_value,
            }
            for r in rows
        ],
    }
```

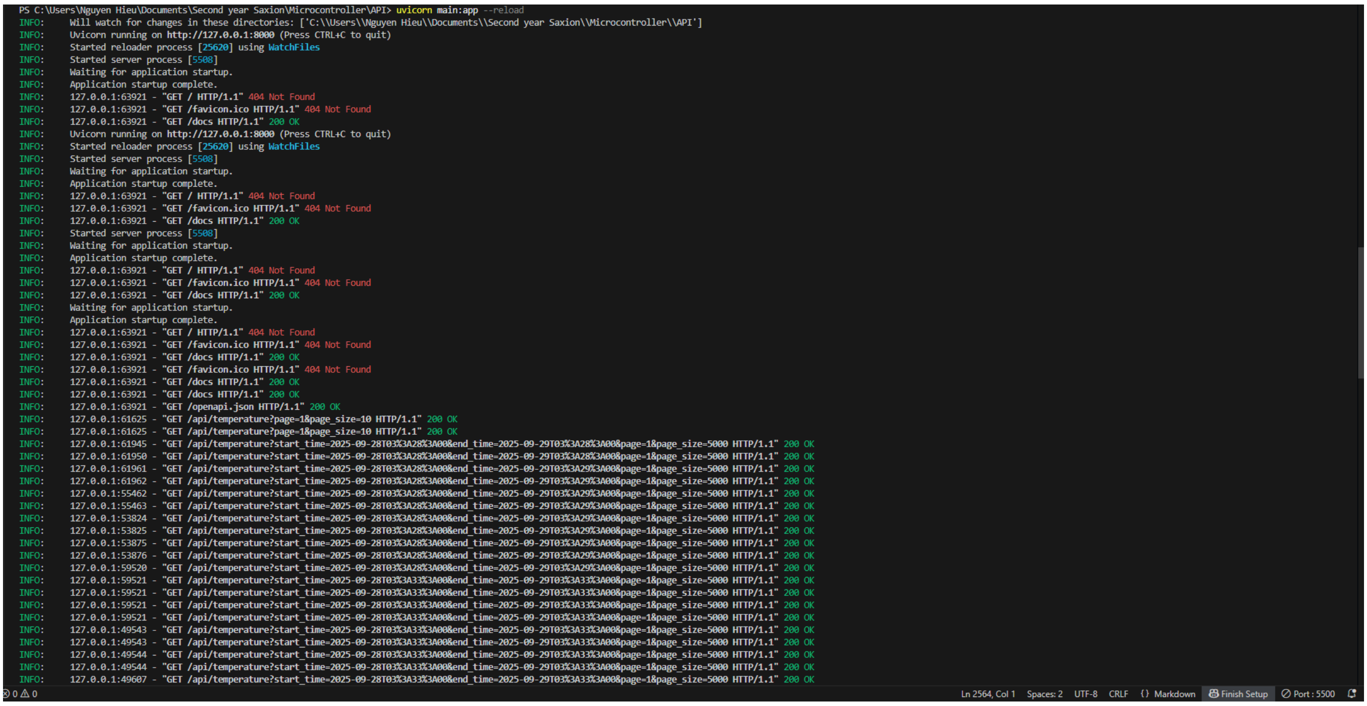
And this is how it get the API

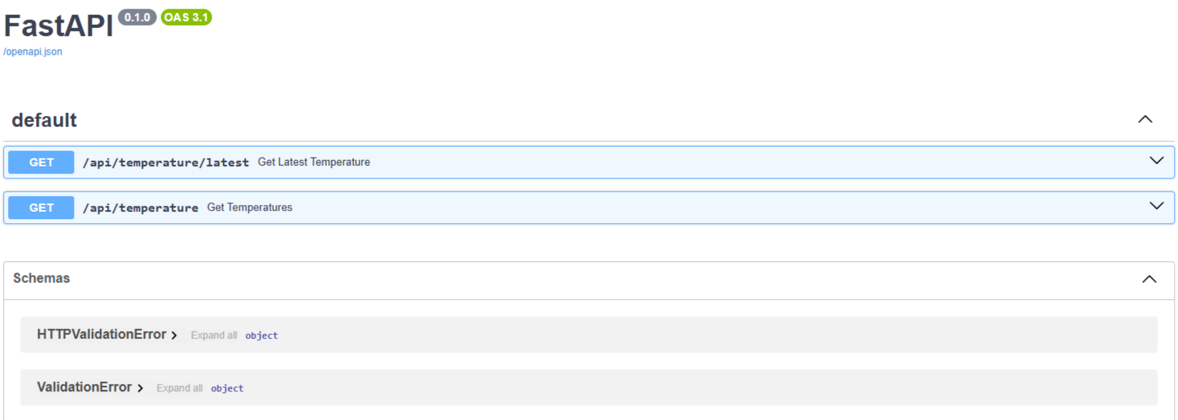
```
from fastapi import FastAPI, HTTPException, Query, Depends
from sqlalchemy import Column, Float, DateTime, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, Session
from typing import Optional
```

```
import math
from fastapi.middleware.cors import CORSMiddleware

# --- FastAPI app (only once!) ---
app = FastAPI()

# --- CORS middleware ---
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://127.0.0.1:5500", "http://localhost:5500"],  # frontend
origins
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# --- Database setup (SQL Server, pyodbc) ---
DATABASE_URL = (
    "mssql+pyodbc://@FELIX-NGUYEN\\SQLEXPRESS/final_project_microcontroller"
    "?driver=ODBC Driver 17 for SQL Server&Trusted_Connection=yes"
)

engine = create_engine(DATABASE_URL, pool_pre_ping=True)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

# --- SQLAlchemy model (match your real table: no id column) ---
class Temperature(Base):
    __tablename__ = "temperature"
    reading_time = Column(DateTime, primary_key=True, nullable=False)
    temperature_value = Column(Float, nullable=False)

# Dependency: get DB session
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()


# GET latest temperature
@app.get("/api/temperature/latest")
def get_latest_temperature(db: Session = Depends(get_db)):
    latest = (
        db.query(Temperature)
        .order_by(Temperature.reading_time.desc())
        .first()
    )

    if not latest:
        raise HTTPException(status_code=404, detail="No temperature readings
found.")
```

```
    return {
        "reading_time": latest.reading_time,
        "temp_c": latest.temperature_value,
        "temp_f": latest.temperature_value * 9 / 5 + 32,
    }


# GET temperature list with filters + pagination
@app.get("/api/temperature")
def get_temperatures(
    start_time: Optional[str] = Query(None),
    end_time: Optional[str] = Query(None),
    page: int = Query(1, gt=0),
    page_size: int = Query(10, gt=0),
    db: Session = Depends(get_db),
):
    query = db.query(Temperature)

    # Apply filters
    if start_time and not end_time:
        query = query.filter(Temperature.reading_time == start_time)

    if end_time and not start_time:
        query = query.filter(Temperature.reading_time == end_time)

    if start_time and end_time:
        query = query.filter(
            Temperature.reading_time >= start_time,
            Temperature.reading_time <= end_time,
        )

    # Pagination
    total_items = query.count()
    total_pages = math.ceil(total_items / page_size) if total_items else 0

    if page > total_pages and total_items > 0:
        raise HTTPException(
            status_code=404,
            detail=f"Page {page} does not exist. Total pages: {total_pages}.",
        )

    rows = (
        query.order_by(Temperature.reading_time.desc())
        .offset((page - 1) * page_size)
        .limit(page_size)
        .all()
    )

    return {
        "total_items": total_items,
        "total_pages": total_pages,
        "current_page": page,
        "page_size": page_size,
        "data": [
```

```
            {
                "reading_time": r.reading_time,
                "temperature_value": r.temperature_value,
            }
            for r in rows
        ],
    }
```





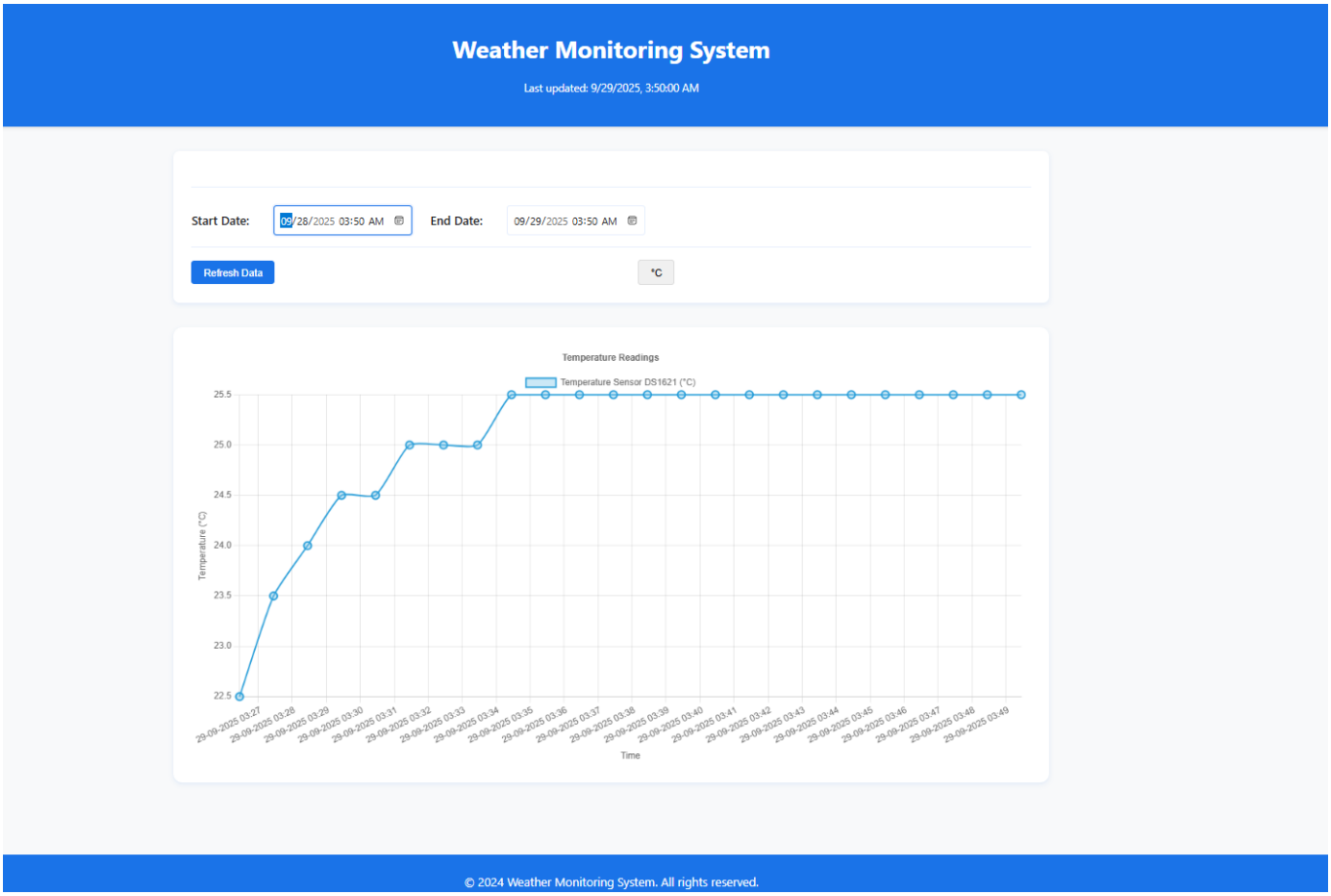
127.0.0.1:8000/api/temperature?page=1&page_size=10

Pretty-print


```
{
  "total_items": 146,
  "total_pages": 15,
  "current_page": 1,
  "page_size": 10,
  "data": [
    {
      "reading_time": "2025-09-29T03:27:27",
      "temperature_value": 23.5
    },
    {
      "reading_time": "2025-09-29T03:26:27",
      "temperature_value": 22.5
    },
    {
      "reading_time": "2025-09-26T23:53:52",
      "temperature_value": 25.5
    },
    {
      "reading_time": "2025-09-26T23:52:52",
      "temperature_value": 25.5
    },
    {
      "reading_time": "2025-09-26T23:51:52",
      "temperature_value": 25.5
    },
    {
      "reading_time": "2025-09-26T23:50:52",
      "temperature_value": 25.5
    },
    {
      "reading_time": "2025-09-26T23:49:52",
      "temperature_value": 25.5
    },
    {
      "reading_time": "2025-09-26T23:48:52",
      "temperature_value": 25.5
    },
    {
      "reading_time": "2025-09-26T23:47:52",
      "temperature_value": 25.5
    },
    {
      "reading_time": "2025-09-26T23:46:52",
      "temperature_value": 25.5
    }
  ]
}
```


Website (HTML, CSS, JS)

Please check the GitHub for information



Github: Felixnguyen05