Hacettepe University

BBM434 Experiment 3

Name and Surname: ZÜBEYDE CİVELEK
Identity Number: 2200356814
Experiment Subject: Architecture - I/O experiment, timers, interrupts experiments
E-Mail: b2200356814@cs.hacettepe.edu.tr

1) Message printing will be done by using serial communication to the port connected via USB

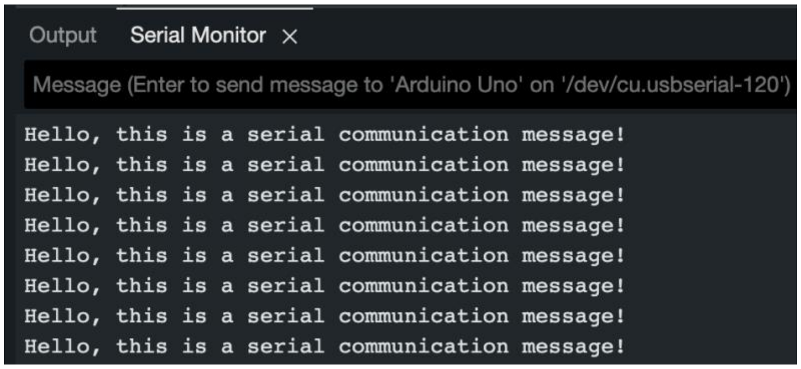
Serial communication is an important feature of embedded systems because it allows data to be sent between a microcontroller and external devices.

I also used serial communication on my model satellite communication for the Teknofest competition last year and in 2022. It allowed me to send and receive data between the satellite and the ground station, enabling real-time monitoring and control of the satellite's functions.

The Arduino sketch in the attached code sample demonstrates serial communication over the USB interface.

```
void setup() {
 // Initialize serial communication
 Serial.begin(9600);
}
void loop() {
 // Send a message over serial communication
 Serial.println("Hello, this is a serial communication message!");

 // Delay for 5 seconds
 delay(5000);
}
```

The setup() function initializes serial communication at 9600 bits per second, establishing a connection between a microcontroller and a computer via USB. The loop() function transmits a message, "Hello, this is a serial communication message!", followed by a newline character for clear separation. A 5000 millisecond delay prevents rapid transmission, ensuring a controlled message output rate.



Message (Enter to send message to 'Arduino Uno' on '/dev/cu.usbserial-120')

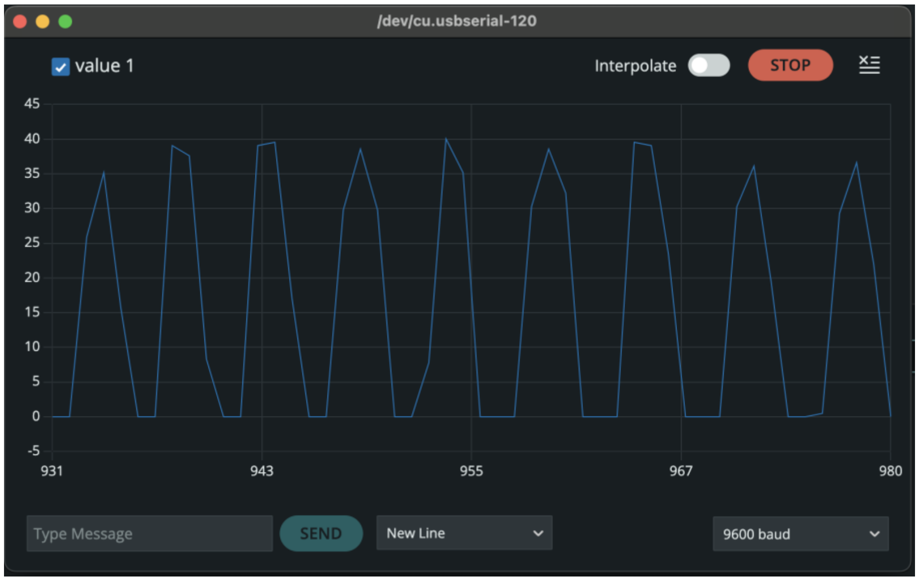
```
Hello, this is a serial communication message!
Hello, this is a serial communication message!
Hello, this is a serial communication message!
Hello, this is a serial communication message!
Hello, this is a serial communication message!
Hello, this is a serial communication message!
Hello, this is a serial communication message!
Hello, this is a serial communication message!
```

2) Using the graphic monitor on the IDE, plot the graph of the data you sent from the microcontrollers.

In this section of the assignment, the objective is to visualize data received from a temperature sensor on the Arduino board using the Serial Plotter tool provided by the Arduino IDE. The code snippet provided reads data from an LM35 temperature sensor connected to analog pin A0 and transmits the temperature values over serial communication.

```
const int lm35Pin = A0;

void setup() {

 Serial.begin(9600);

}

void loop() {

 int sensorValue = analogRead(lm35Pin);

 float temperature = (sensorValue * 5.0 / 1024.0) * 100.0;

 Serial.println(temperature);

 delay(1000);

}
```

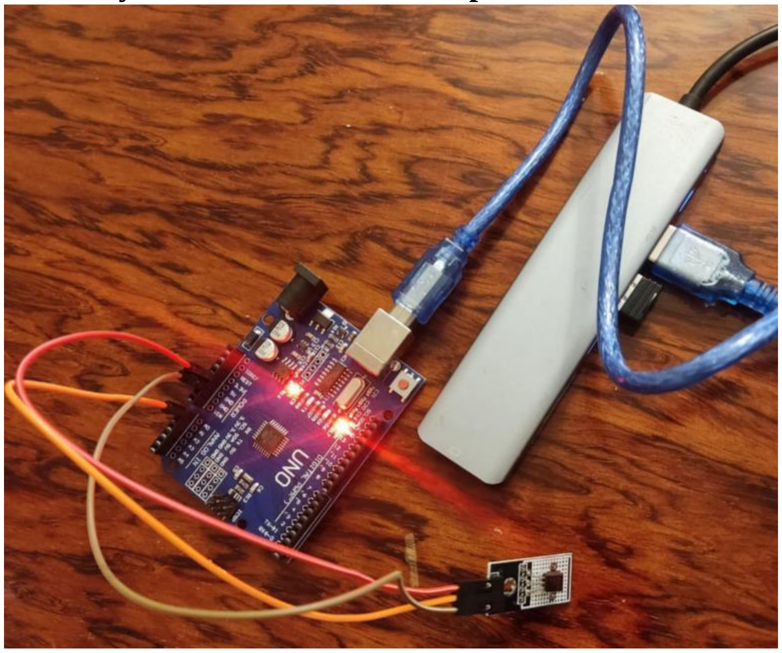
The setup() function initializes serial communication with a 9600 bits per second baud rate. The loop() function reads analog data from the LM35 temperature sensor, converts it to Celsius, and transmits the temperature values over the serial port. The Arduino IDE offers a tool called the Serial Plotter, which automatically detects and plots the data sent over the serial port, providing real-time visualization for monitoring and analyzing temperature sensor behavior.


/dev/cu.usbserial-120

New Line

9600 baud

My Arduino and LM35 temperature sensor:



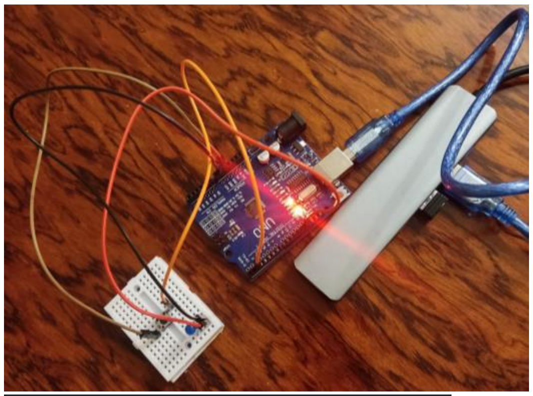
3) The procedure for the cutting mechanism will be used. For this, the software that prints the pressing process on the screen through the interrupt procedure when a key is pressed will be implemented.

In this section of the assignment, an interrupt-driven approach is employed to implement the procedure for the cutting mechanism. My code utilizes a touch button connected to pin 2, and an LED connected to pin 13 to simulate the pressing and cutting process.

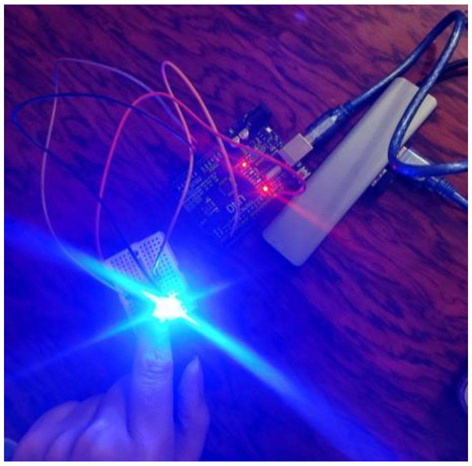
```
const int buttonPin = 2; // The pin to which the touch button is connected
const int ledPin = 13;   // The pin to which the LED is connected
volatile bool buttonPressed = false; // Flag for interrupt processing

void setup() {
 Serial.begin(9600);
 pinMode(buttonPin, INPUT); // Set the touch button pin as input
 pinMode(ledPin, OUTPUT); // Set the LED pin as output

 // Attach a function to the falling edge interrupt
 attachInterrupt(digitalPinToInterrupt(buttonPin), buttonInterrupt, FALLING);
}

void loop() {
 if (buttonPressed) {
   noInterrupts(); // Disable interrupts to ensure atomic operation
```

```
    Serial.println("Touch button pressed! Interrupt processing occurred.");
    digitalWrite(ledPin, HIGH); // Turn on the LED
    interrupts(); // Re-enable interrupts
    buttonPressed = false; // Reset the flag
  }
}

void buttonInterrupt() {
 // Interrupt function
 noInterrupts(); // Disable interrupts to ensure atomic operation
 Serial.println("Interrupt function called.");
 buttonPressed = true;
 digitalWrite(ledPin, LOW); // Turn off the LED
 interrupts(); // Re-enable interrupts
}
```

The attachInterrupt function associates the buttonInterrupt function with the falling edge of the touch button signal. When the button is pressed, an interrupt is triggered, and the associated buttonInterrupt function is executed. Interrupts are temporarily disabled to ensure atomic operation. The function prints a message, sets the buttonPressed flag to true, turns off the LED, and re-enables interrupts. If the buttonPressed flag is true, the LED is turned on, and the flag is reset to false.



```
Interrupt function called.
Interrupt function called.
```



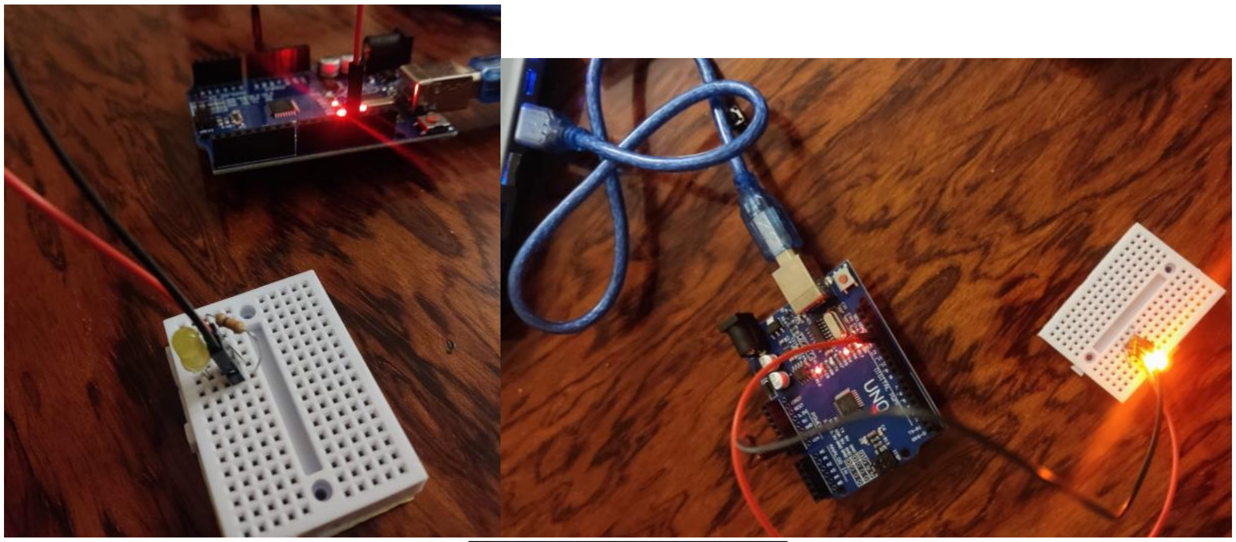
```
Interrupt function called.
Touch button pressed! Interrupt processing occurred.
```

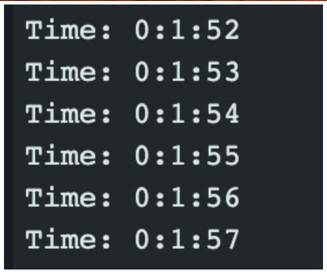
4) **Perform the real-time clock software on the microcontroller you use by using the clock interrupt procedure.** Again on this software, write and run the main program section required for updating and querying the clock.

In this section of the experiment, a real-time clock (RTC) is implemented on the Arduino Uno using the clock interrupt procedure. The code leverages Timer1 to generate interrupts at regular intervals, allowing for the accurate tracking of time.

```
#include <avr/io.h>
#include <avr/interrupt.h>
const int ledPin = 13; // Pin connected to the LED on Arduino Uno
volatile int hours = 0;
volatile int minutes = 0;
volatile int seconds = 0;
void setup() {
 pinMode(ledPin, OUTPUT);  // initialize the digital pin as an output
 Serial.begin(9600);       // initialize serial communication
 // Set up Timer1 for interrupt every second
 cli(); // Disable interrupts
 TCCR1A = 0; // Set entire TCCR1A register to 0
 TCCR1B = 0; // Set entire TCCR1B register to 0
 // Set compare match register to desired timer count:
 OCR1A = 15624;
  (Clear Timer on Compare Match)
 TCCR1B |= (1 << WGM12);// Enable CTC mode
 TCCR1B |= (1 << CS12) | (1 << CS10);// Set CS12, CS10 bits for 1024 prescaler
 TIMSK1 |= (1 << OCIE1A);// Enable timer compare interrupt
 sei(); // Enable interrupts
}
void loop() {
 Serial.print("Time: ");
 Serial.print(hours);
 Serial.print(":");
 Serial.print(minutes);
 Serial.print(":");
 Serial.println(seconds);
 delay(1000);
}
ISR(TIMER1_COMPA_vect) {
 // Update the clock values
 seconds++;
 if (seconds == 60) {
    seconds = 0;
    minutes++;
    if (minutes == 60) {
      minutes = 0;
      hours++;
      if (hours == 24) {
        hours = 0;
      }
    }
  }
  digitalWrite(ledPin, !digitalRead(ledPin));
}
```

The setup() function in Timer1 generates interrupts every second using the CTC mode, updating clock values and toggling the LED. The loop() function can be used for additional tasks in the main program while the clock updates in the background through interrupts. The loop example prints the current time to the Serial Monitor every second.





```
Time:  0:1:52
Time:  0:1:53
Time:  0:1:54
Time:  0:1:55
Time:  0:1:56
Time:  0:1:57
```