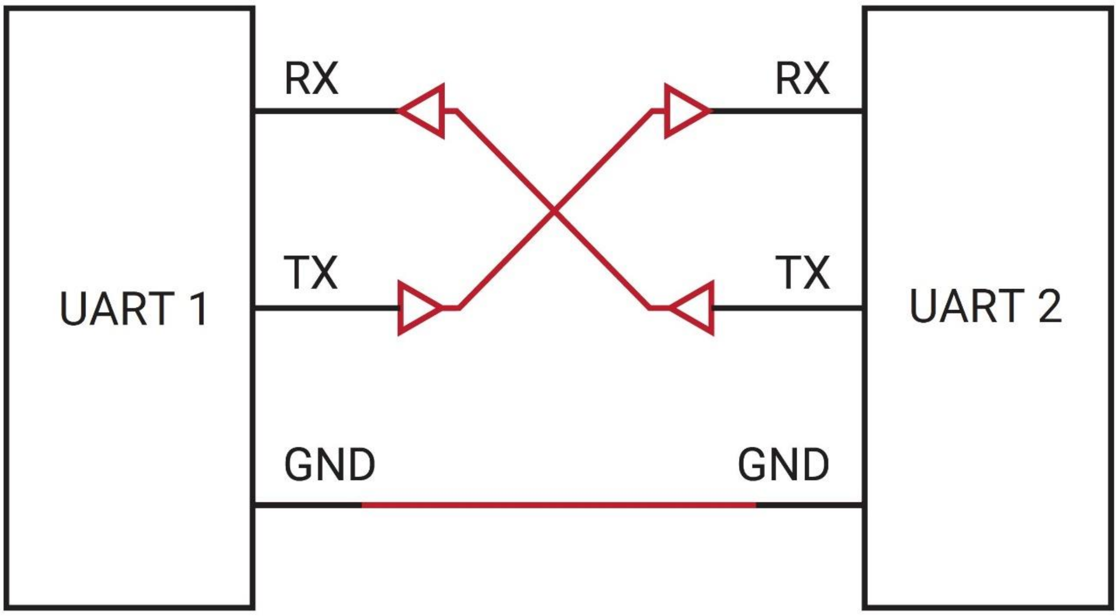
Universal asynchronous receiver-transmitter



Alexandru Malgras

16/01/2022

Table of Contents

Introduction

The goal of this assignment is to understand the basis of the universal asynchronous receiver-transmitter. It is one of the most used device to device communication protocols. In the first part of the assignment, I am covering the pre-built serial communication. The Arduino Uno has one Rx pin (pin 0 – receiver) and one Tx pin (pin 1 – transmitter). The second part will cover building my own UART communication using software. Any of the pins on the Arduino can be used as a Tx or Rx pin.

Research

Literature Study

To start of the assignment, I familiarized myself with UART. To do so I followed an article that explains the basics of UART in a simpler manner.
UART stands for universal asynchronous receiver-transmitter, and it is a communication process between the Arduino and other devices such as a laptop. The communication goes both ways meaning that the Arduino can receive information but can also send information.

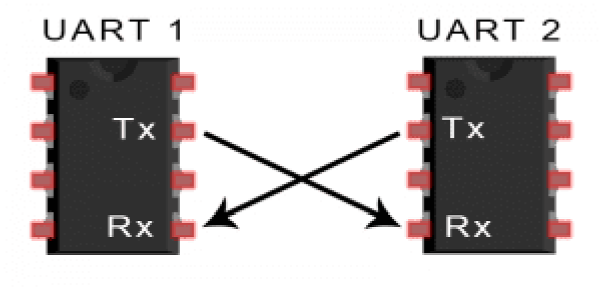


Figure 1 UART communication

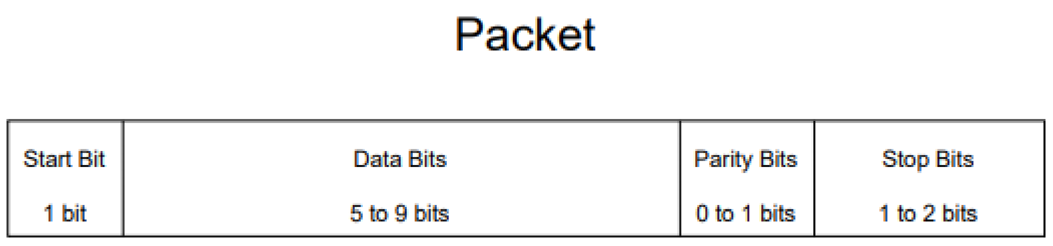
In figure 1 it is shown how the communication works. The Arduino Uno has one transmitter pin (Tx) and one receiver pin (Rx), however using software other pins can also become transmitters or receivers. The data will be transmitted from a device to another using the transmitter pin and it will be intercepted using the receiver pin.

The data is sent in form of packets. During the transmitting process the data is converted into a packet and sent to the other device. During the receiving process the data is rebuilt from a received packet.

Sending the data

The transmitting device converts the data bytes into bits. After the conversion is complete the bits are split into packets. When this process is concluded, the packets are sent to the other device.

The components of a packet are, a start bit, data bits, parity bits and stop bits.



Packet

Start Bit	Data Bits	Parity Bits	Stop Bits
1 bit	5 to 9 bits	0 to 1 bits	1 to 2 bits

Figure 2 Packets

In figure 2 the components of a packet are shown. The packet is sent through the transmitter pin (Tx).

Start bit

The UART data transmission line is set to high voltage while it is not transmitting data. In order to start the data transmission, the line is set to low voltage. When the conversion occurs the bit reading starts at the frequency of the baud rate.

Data bits
The data bits or the data frame represent the bits allocated to the data. The data bits vary from five to nine. The maximum of data bits that can be allocated in the packet when a parity bit is set is eight. If the parity bit is not set the amount of data bits can be nine.

Parity bits
The point of setting a parity bit is to check the data integrity. If the parity bit is set to zero it means that the total number of ones in the data frame is even. If the parity bit is set to one it means that the total number of ones in the data frame is odd.

When the UART receives a packet, it reads the data frame and counts the total amount of ones in the data frame and compares it to the parity bit. If the oddness or evenness matches the parity bit, then the data was sent without errors. A common error that affects the data integrity is the mismatched baud rate between the devices.

Stop bits
The UART data transmission line is set back to high voltage from one to two bits duration. The stop bit(s) signal the end of the data packet.

This process can be directly seen on the Arduino Uno. While the data is either transmitted or received the Rx LED will change accordingly.

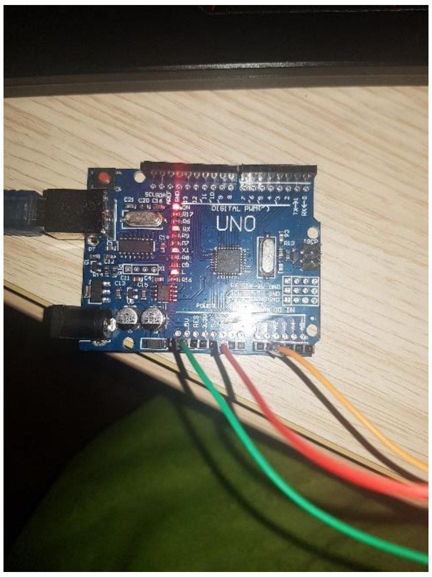


Figure 3 Packets are not received Rx LED is OFF

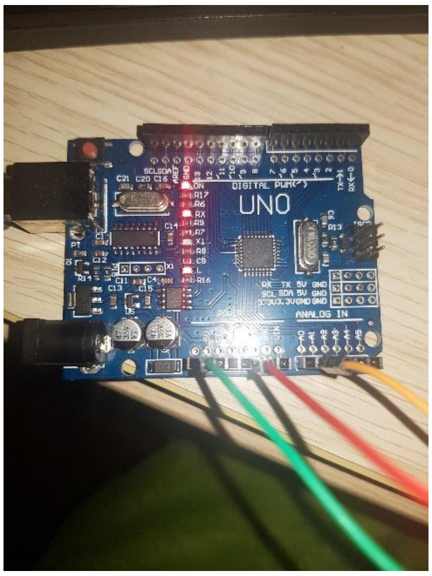


Figure 4 Packets are received Rx LED is ON

Receiving the data

The receiving device checks the packet for possible errors by calculating the number of 1s and comparing the value to the parity bit located in the packet. If the packet has no errors the start bit, parity bits and stop bits are stripped away to get the data bits. A byte is rebuilt using the data bits and stored in the UART buffer.

To ensure data integrity the parity bit is used to check if the state of the bits is the same as when they were transmitted.

UART parameters

In order to have a good communication between the devices some settings must be done. These settings are known as UART parameters. The UART parameters are, baud rate, data length, parity bit, number of stop bits and flow control.

Baud Rate

The baud rate represents the number of bits per second an UART device can transmit or receive. The devices must have the same baud rate otherwise transmitting errors are likely to happen. In the assignment the baud rate used was 9600 bps. An Arduino Uno supports up to 115200 bps.

Data Length

The data length represents the number of bits per byte of data.

Parity Bit

The parity bit represents the evenness or the oddness of the number of ones in the data frame. The parity bit is 0 if the number is even. The parity bit is 1 if the number is odd.

Number of stop bits

The number of stop bits represent the end of a set of bits. A packet can have one to two stop bits.

Flow control

The flow control represents the protection of the data integrity. The UART devices uses special characters as flow control to start or stop the transmission.

The assignment

The assignment is split into three parts.

1. Making a simple ASCII terminal server.
- The terminal must be started only when the character ‘S’ is received.
- When the character ‘D’ is received the levels of digital input must be shown for the pins 8 - 13.
- When the character ‘A’ is received the levels of analog input must be shown for the pins 0 – 5.
- When the character ‘C’ is received the terminal must be cleared.

2. Making a design for the software UART.
3. Implementing the UART module.

Making a simple ASCII terminal server

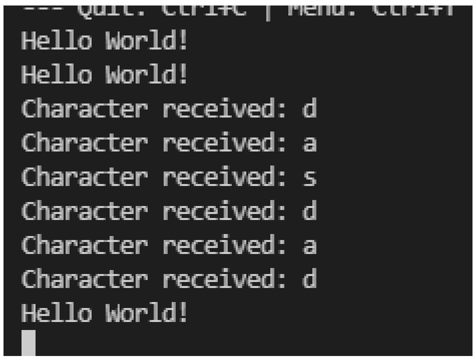
Reading through the assignment I got confused and I approached this solution differently. Instead of making a simple ASCII terminal server I tested the communication between the Arduino Uno and my laptop by creating a serial communication using the tutorial presented in the Arduino Uno ATmega328P datasheet and the information I gathered from the literature study.

This program receives information from the laptop and displays it in the terminal, but also sends a string displayed in the terminal.

The code

```
#include <Arduino.h>
#include <String.h>

#define BAUD 9600
#define MYUBRR ((F_CPU / 16L / BAUD) - 1)

void setup() {
  // put your setup code here, to run once:

  // Set baud rate to 9600
  UBRR0H = (unsigned char)(MYUBRR >> 8);
  UBRR0L = (unsigned char)MYUBRR;

  // Enable transmitter and receiver
  UCSR0B |= (1 << RXEN0);
  UCSR0B |= (1 << TXEN0);

  // Set asynchronous USART
  UCSR0C &= ~(1 << UMSEL01);
  UCSR0C &= ~(1 << UMSEL00);
  // Set frame format to 8data
  UCSR0B &= ~(1 << UCSZ02);
  UCSR0C |= (1 << UCSZ01);
  UCSR0C |= (1 << UCSZ00);

  // Set frame format to 2 stop bit
  UCSR0C |= (1 << USBS0);
}

unsigned long millis_target;

void loop() {
  // put your main code here, to run repeatedly:

  if ((UCSR0A & (1 << RXC0)))
  {
    char input = UDR0;
    String output = "Character received: ";

    for (uint8_t i = 0; i < output.length(); i++)
    {
      // Wait until the data transmit buffer is empty
      while (!(UCSR0A & (1 << UDRE0)));

      // Add new data to the transmit buffer
      UDR0 = output[i];
    }

    // Wait until the data transmit buffer is empty
    while (!(UCSR0A & (1 << UDRE0)));

    // Add new data to the transmit buffer
    UDR0 = input;

    // Wait until the data transmit buffer is empty
    while (!(UCSR0A & (1 << UDRE0)));

    // Add new data to the transmit buffer
    UDR0 = '\n';
  }

  if (millis() >= millis_target)
  {
    millis_target = millis_target + 5000;

    // Output data
    String output = "Hello World!\n";
    for (uint8_t i = 0; i < output.length(); i++)
    {
      // Wait until the data transmit buffer is empty
      while (!(UCSR0A & (1 << UDRE0)));

      // Add new data to the transmit buffer
      UDR0 = output[i];
    }
  }
}
```

Output



```
--- Quit: Ctrl+C | Menu: Ctrl+T
Hello World!
Hello World!
Character received: d
Character received: a
Character received: s
Character received: d
Character received: a
Character received: d
Hello World!
```

Figure 5 Receiver-transmitter output

In figure 5 the output shown in the console can be seen. The transmitter sends the string “Hello World!”, the receiver receives different characters from the laptop.

Making a design for the software UART

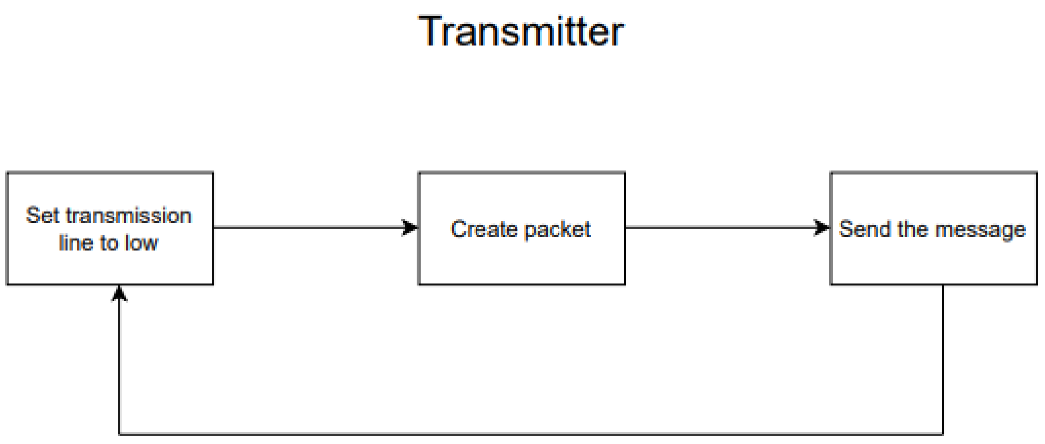


Figure 6 Transmitter design

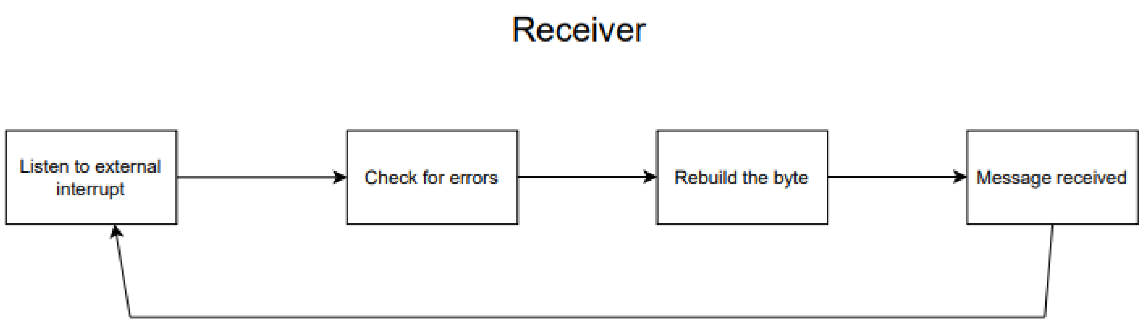


Figure 7 Receiver design

Implementing the UART module

The program below contains three methods. One method for the transmitter (send), one for the receiver (receive) and one printing method that prints a string to the terminal using the send method.

The program follows on the literature study and the UART design.

The send method is creating a packet. When the packet is created, it is sent to the receiver.

The receive method is rebuilding the received packet into two bytes to be able to read the information.

The code

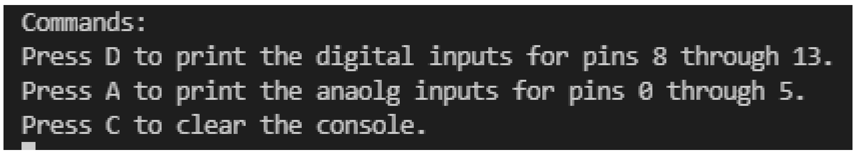
```
#include <Arduino.h>

#define BAUD_RATE 9600
#define BIT_DURATION_MICROS (1000000UL / BAUD_RATE)
#define PACKET_START_BITS 1
#define PACKET_DATA_BITS 8
#define PACKET_PARITY_BITS 1
#define PACKET_END_BITS 1
#define PACKET_BITS (PACKET_START_BITS + PACKET_DATA_BITS + PACKET_PARITY_BITS
+ PACKET_END_BITS)

/*
Simple send method without the use of timers.
Blocks until the full message is sent.
*/
void send(uint16_t data)
{
    unsigned long micros_target = micros();

    // Send the start bits
    PORTD &= ~(1 << PORTD1);
    micros_target += BIT_DURATION_MICROS * PACKET_START_BITS;
    while (micros() < micros_target);

    // Send data bits
    uint8_t parity = 0;
    for (uint8_t i = 0; i < PACKET_DATA_BITS; i++)
    {
        uint16_t bit = (data >> i) & 1;

        PORTD = (PORTD & ~(1 << PORTD1)) | (bit << PORTD1);
        parity ^= bit;

        micros_target += BIT_DURATION_MICROS;
        while (micros() < micros_target);
    }

    // Send parity bits
    for (uint8_t i = 0; i < PACKET_PARITY_BITS; i++)
    {
        uint16_t bit = parity & 1;

        PORTD = (PORTD & ~(1 << PORTD1)) | (bit << PORTD1);
        parity ^= bit;

        micros_target += BIT_DURATION_MICROS;
        while (micros() < micros_target);
    }

    // Send end bits
    if (PACKET_END_BITS > 1)
    {
        PORTD &= ~(1 << PORTD1);
        micros_target += BIT_DURATION_MICROS * (PACKET_END_BITS - 1);
        while (micros() < micros_target);
    }

    PORTD |= (1 << PORTD1);
    if (PACKET_END_BITS)
    {
        micros_target += BIT_DURATION_MICROS;
        while (micros() < micros_target);
    }
}

/*
Simple print method without the use of timers.
Blocks until the full message is sent.
*/
void print(String text)
{
    for (unsigned int i = 0; i < text.length(); i++)
    {
        send(text.charAt(i));
    }
}

/*
Simple receive method without the use of interrupts or timers.
Blocks until a character is received.
*/
uint16_t receive()
```

```
{
    unsigned long micros_target;

    // Wait until start bit is received and start timing
    while (PIND & (1 << PIND0));
    micros_target = micros();

    // Wait until start bit is over
    micros_target += BIT_DURATION_MICROS * PACKET_START_BITS;
    while (micros() < micros_target);

    // Record data bits
    uint8_t parity = 0;
    uint16_t data = 0;
    for (uint8_t i = 0; i < PACKET_DATA_BITS; i++)
    {
        uint16_t bit = ((PIND & (1 << PIND0)) >> PIND0);

        data |= (bit << i);
        parity ^= bit;

        // Wait until this databit is over
        micros_target += BIT_DURATION_MICROS;
        while (micros() < micros_target);
    }

    // Record parity bits
    for (uint8_t i = 0; i < PACKET_PARITY_BITS; i++)
    {
        parity ^= ((PIND & (1 << PIND0)) >> PIND0);

        // Wait until this paritybit is over
        micros_target += BIT_DURATION_MICROS;
        while (micros() < micros_target);
    }

    // Wait until end bits are over
    micros_target += BIT_DURATION_MICROS * PACKET_END_BITS;
    while (micros() < micros_target);

    // Validate parity
    if (parity)
    {
        return 0UL;
    }

    return data;
}
```

```
void setup()
{
  //Set pin modes to input for power reduction.
    DDRB = 0;
    DDRC = 0;
    DDRD = 0;

  //Enable pull-up on all pins for power reduction
    PORTB = 255;
    PORTC = 255;
    PORTD = 255;

    //Set up send pin (D1)
    DDRD |= (1 << DDD1);        //Output
    PORTD |= (1 << PORTD1);     //High

    //Set up receive pin (D0)
    DDRD &= ~(1 << DDD0);       //Input
    PORTD |= (1 << PORTD0);     //Enable internal pullup

    /*Setting up adc to read the analog input*/

    //Set voltage refference to AVcc
    ADMUX &= ~(1 << REFS1);
    ADMUX |= (1 << REFS0);

    //Left adjust the result
    ADMUX |= (1 << ADLAR);

    //Disable digital input for analog pins
    DIDR0 |= (1 << ADC0D) | (1 << ADC1D) | (1 << ADC2D) | (1 << ADC3D) | (1 <<
ADC4D) | (1 << ADC5D);
}

void loop()
{
    //Assigning the received key
    char received = receive();

    //Start the terminal
    if(received != 'S')
    {
        return;
    }

    print("Commands:\n");
    print("Press D to print the digital inputs for pins 8 through 13.\n");
    print("Press A to print the anaolg inputs for pins 0 through 5.\n");
    print("Press C to clear the console.\n");

    while(true)
    {
        received = receive();
        //Printing the digital inputs for pins 8 through 13
        if(received == 'D')
        {
            print("Printing the digital inputs for pins 8 through 13\n");
            for (uint8_t i = 0; i < 6; i++)
            {
                print("DI");
                if(i < 2)
                {
                    print("0");
                    print((String)(i+8));
                }
                else
                {
                    print((String)(i+8));
                }
                print(": ");

                if((PINB & (1 << (0+i))))
                {
                    send('1');
                }
                else
                {
                    send('0');
                }
                print("\n");
            }
        }

        //Printing the analog inputs for pins 0 through 5
        if(received == 'A')
        {
            print("Printing the analog inputs for pins 0 through 5...\n");

            for (uint8_t i = 0; i < 6; i++)
            {
                if( i == 0)
                {
                    //Enable ADC
                    ADCSRA |= (1 << ADEN);
                }
```

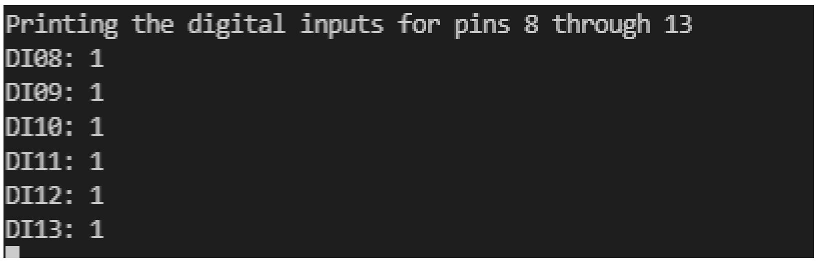
```
                //Select ADC pin
                ADMUX &= ~(1 << MUX0) & ~(1 << MUX1) & ~(1 << MUX2) & ~(1 <<
MUX3);
                ADMUX |= i;

                //Start conversion and wait for completion
                ADCSRA |= (1 << ADSC);
                while (ADCSRA & (1 << ADSC));

                print("AI");
                print((String)i);
                print(": ");
                print((String)ADCH);
                print("\n");

                if(i == 5)
                {
                    // Disable ADC
                    ADCSRA &= ~(1 << ADEN);
                }
            }
        }

        //Clearing the console
        if(received == 'C')
        {
            print("Clearing the console...\n");
            delay(1000);
            //Clearing the console and adjusting the text to reset the cursor.
            print("\e[0H\e[0J");
            return;
        }
    }
}
```

Output



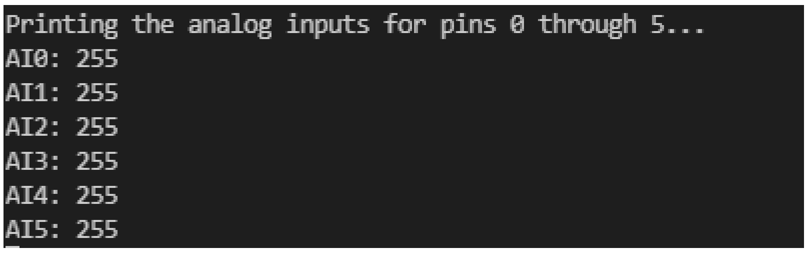
```
Commands:
Press D to print the digital inputs for pins 8 through 13.
Press A to print the anaolg inputs for pins 0 through 5.
Press C to clear the console.
```

Figure 8 Terminal after pressing S



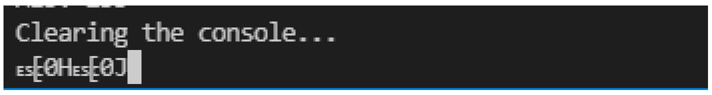
```
Printing the digital inputs for pins 8 through 13
DI08: 1
DI09: 1
DI10: 1
DI11: 1
DI12: 1
DI13: 1
```

Figure 9 Terminal after pressing D



```
Printing the analog inputs for pins 0 through 5...
AI0: 255
AI1: 255
AI2: 255
AI3: 255
AI4: 255
AI5: 255
```

Figure 10 Terminal after pressing A



```
Clearing the console...
␛[0H␛[0J
```

Figure 11 Terminal after pressing C

In order to close the terminal, I used ANSI escape sequences. In figure 11 we can see the sequence is printed but it has no effect. In order to see the sequence in effect a PUTY terminal must be used.

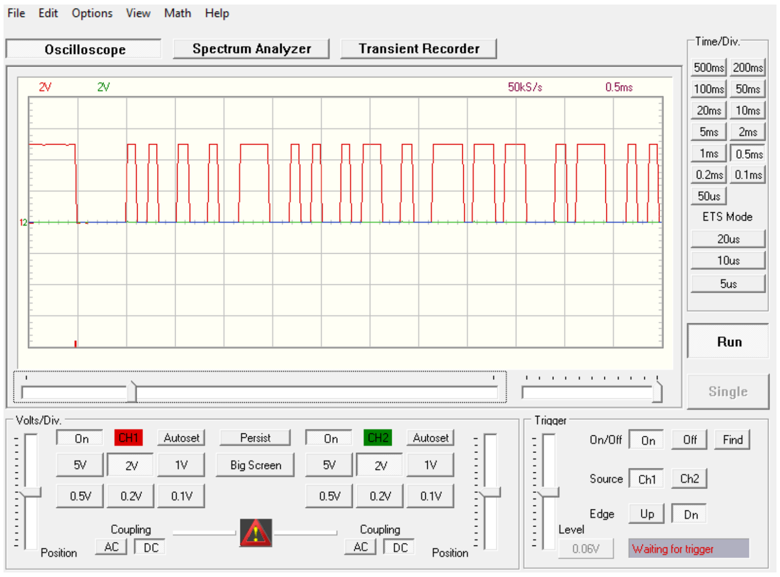


Figure 12 Logic analyzer result while transmitting pin states to the laptop

Conclusion

This assignment finalizes the Embedded Systems Minor. Since the first year of university the focus was very little on embedded/hardware since I am following the software path. Throughout this minor I was able to refresh my memory on topics I learned long ago, such as mathematics and ICT fundamentals. I learned a lot about programming in low level and how software and hardware are linked.

During this assignment I learned about how the information is processed by the CPU, how a keyboard works, understanding how to use serial communication between devices, how to create a serial communication using software.

While the assignment is not near a perfect state, I think it taught me significant information that will for sure be useful in the future.

If I could improve something about this assignment that would be lowering the risk of losing data integrity by checking for different kind of errors in the transmission and overall improvements to the performance. Sometimes I was debating if enabling/disabling some of the components rather than checking if they are enabled/disabled to proceed with some processes would slightly improve the performance.

References

Mallari, J. (2022). How to Set Up UART Communication on the Arduino. Retrieved 16 January 2022, from https://www.circuitbasics.com/how-to-set-up-uart-communication-for-arduino/

Peňa, E., & Legaspi, M. (2022). UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter | Analog Devices. Retrieved 17 January 2022, from https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

ANSI Escape Codes. (2018). Retrieved 17 January 2022, from https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797

Arduino Uno datasheet ATmega328P. (2022). Retrieved 17 January 2022, from https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf