MCT 4334

Embedded System Design

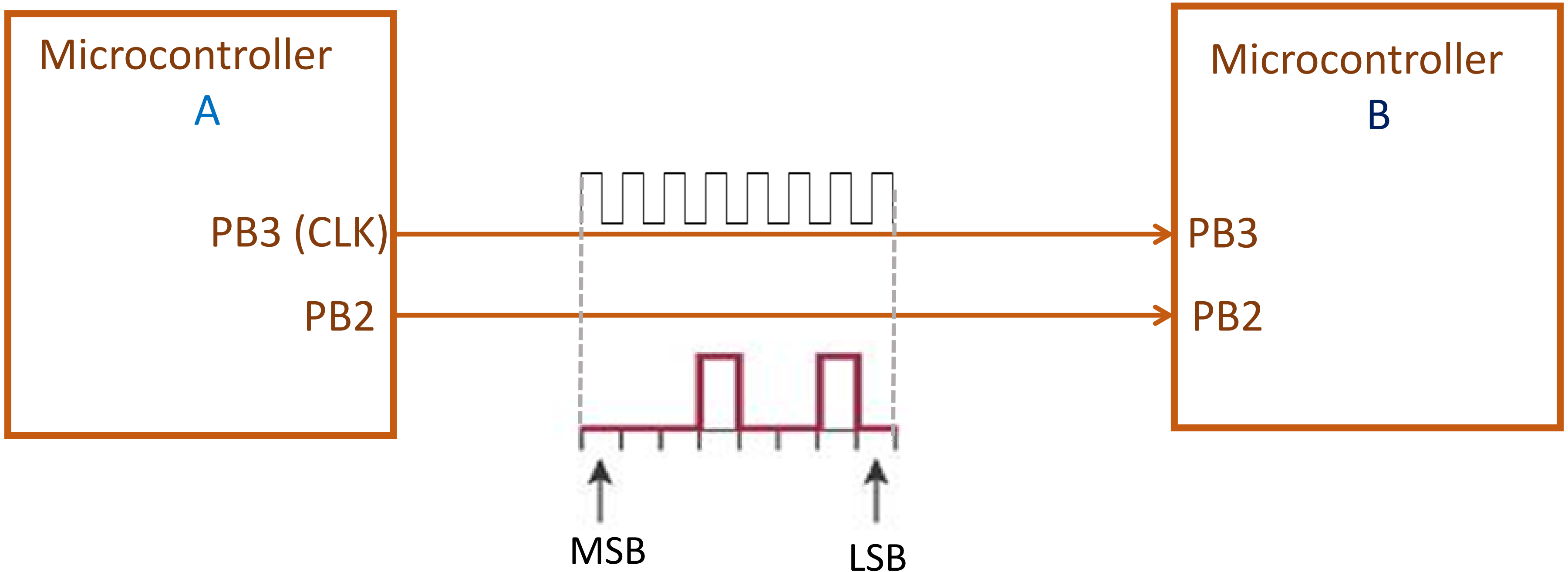
Week 12 Serial Communication

Outline

- Programming examples
- USART
- SPI
- I2C

Synchronous transmission - Review

- The receiver can observe the clock signal and read the individual bits at clock transitions (either PGT or NGT).
- The sender and receiver need to agree on the endianness (MSB first or LSB first?)



Example 1

PB2 and PB3 of two ATmega328p microcontrollers are connected as shown below.

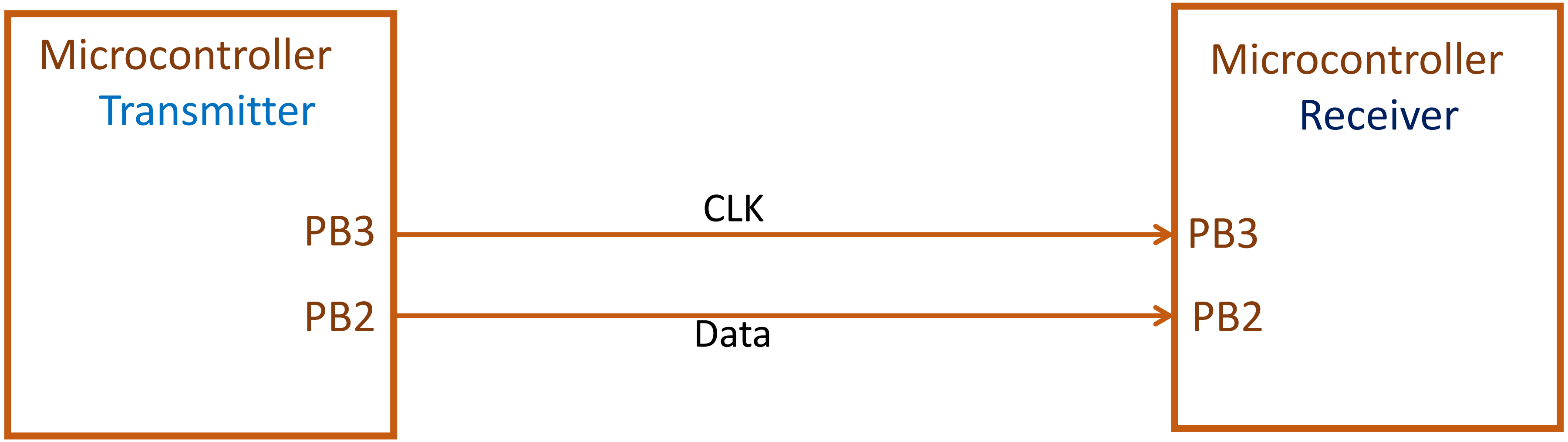
For the transmitter

Write a program that sends a byte of data synchronously through PB2 and PB3. The required bit-rate of the data is 100 Kbps. Send MSB first.

For the receiver

Write a program that keeps waiting for incoming data and prints the received byte through the serial port (as soon as it is received).

Tip:
Baud rate = bit rate
(if nothing else
gets added)



Code for Transmitter

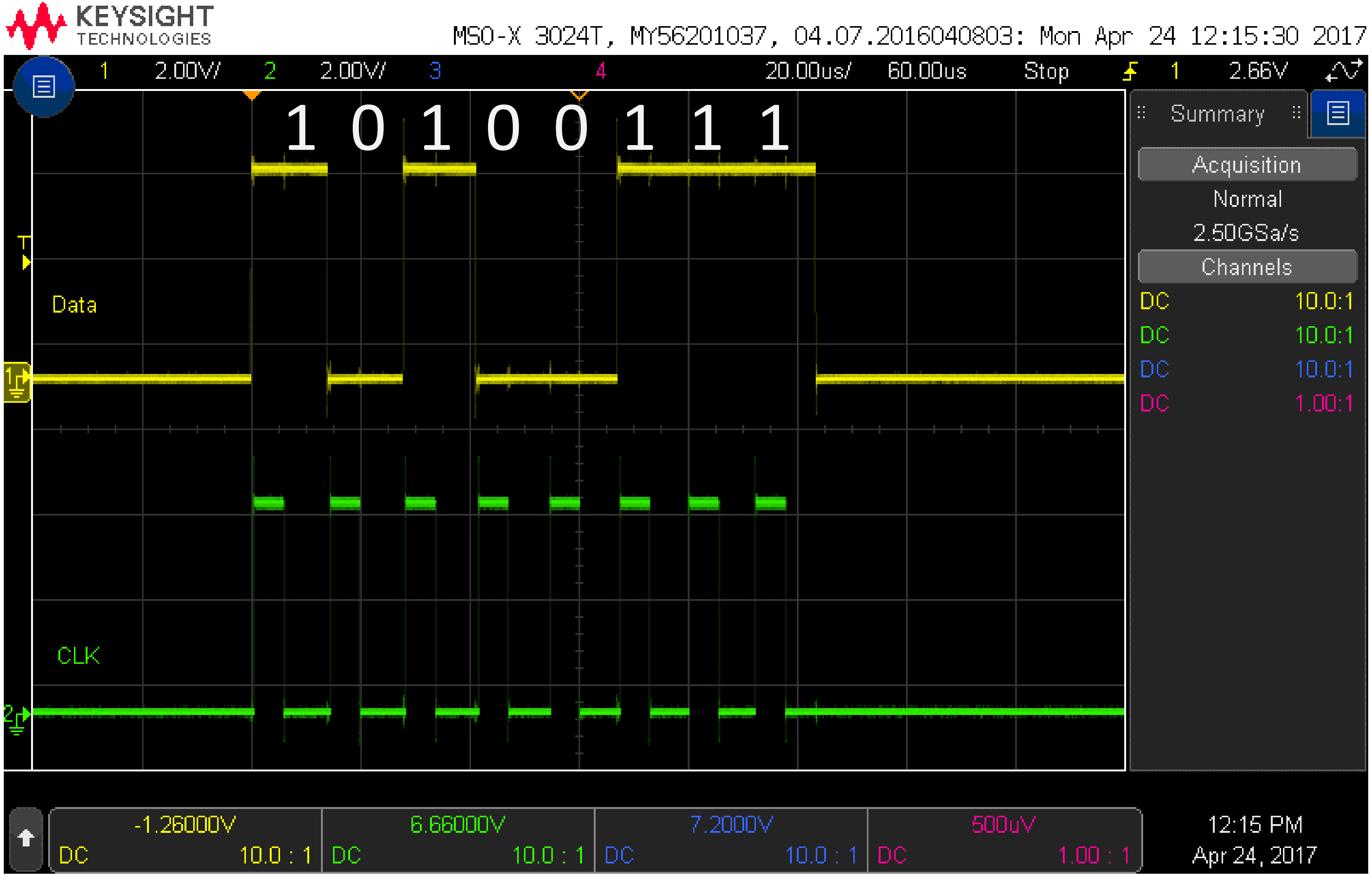
```
char* ddrb = (char*) 0x24;
char* portb = (char*) 0x25;

int main()    //void setup() for Arduino
{
      *ddrb |= (1 << 2);   //PB2 is Data line. This code is same as pinMode(10, OUTPUT);
      *ddrb |= (1 << 3);   //PB3 is clock line. This code is same as pinMode(11, OUTPUT);

      Transmit(167, 100000); //Transmit 1010 0111 at 100kBd.
}
```

```
void Transmit(unsigned char data, unsigned long BaudRate)
{
        unsigned long delay_us = 1000000/BaudRate;   //Delay in microseconds
        unsigned long clock_delay = delay_us / 2;    //Period of CLK is half of signal

        for (int i=7; i>=0; i--)             //Start with 7th bit (MSB) first
        {
                if (data & (1 << i))         //If the ith bit is HIGH
                {
                        *portb |= 1 << 2;            //Make PB2 HIGH
                }
                else
                {
                        *portb &= ~(1 << 2);         //Make PB2 LOW
                }
                *portb |= 1 << 3;                    //Make CLK HIGH
                delayMicroseconds(clock_delay);
                *portb &= ~(1 << 3);                 //Make CLK LOW;
                delayMicroseconds(delay_us - clock_delay);
        }
        *portb &= ~(1 << 2);                         //Done.  Make PB2 back to LOW
}
```

Code for Receiver

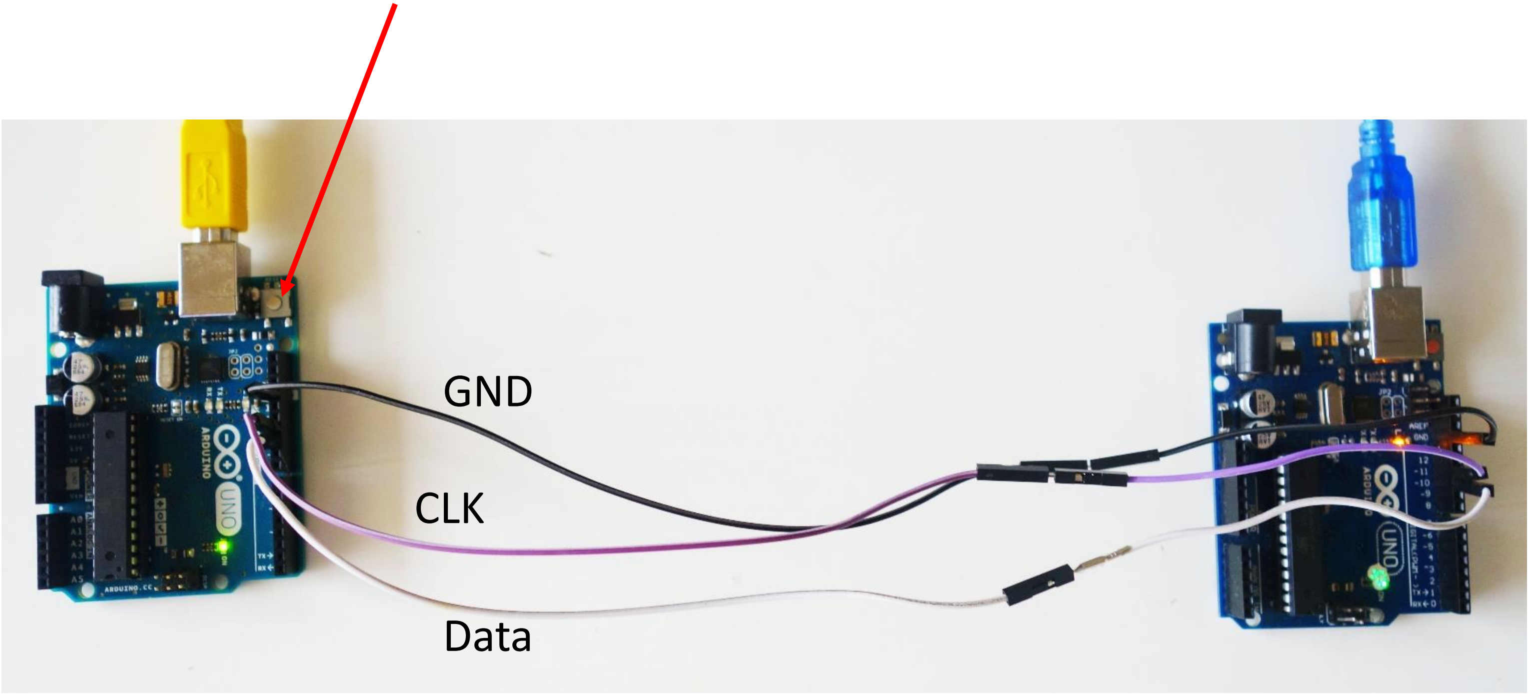
```
volatile char* pinb = (char*) 0x23;

int main()   //void setup() for Arduino
{
      Serial.begin(9600);

      while(1)
      {
              unsigned char data = Receive();  //Get the incoming data
              Serial.println(data);            //Print the incoming data
      }
}
```

```
char Receive()
{
      char ReceivedData;

      bool previous_CLK = ((*pinb) & (1 << 3));                 //previous_CLK = digitalRead(11);

      for (int i=7; i>=0; i--)
      {
            for (;;)
            {
                  bool current_CLK = ((*pinb) & (1 << 3));
                  if (current_CLK && !previous_CLK)               //PGT
                  {
                        previous_CLK = current_CLK;
                        break;                                     //Break statement can be avoided
                  }
                  previous_CLK = current_CLK;

            }
            if ((*pinb) & (1 << 2))                               //If PB2 (data pin) is HIGH
            {
                  ReceivedData |= 1 << i;                               //Set the ith bit
            }
            else
            {
                  ReceivedData &= ~(1 << i);                            //Clear the ith bit
            }
      }
      return ReceivedData;
}
```


KEYSIGHT TECHNOLOGIES
MSO-X 3024T, MY56201037, 04.07.2016040803: Mon Apr 24 12:15:30 2017
1 2.00V/ 2 2.00V/ 3 4 20.00us/ 60.00us Stop 1 2.66V
1 0 1 0 0 1 1 1
Data
CLK
Summary
Acquisition
Normal
2.50GSa/s
Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1
-1.26000V DC 10.0 : 1
6.66000V DC 10.0 : 1
7.2000V DC 10.0 : 1
500uV DC 1.00 : 1
12:15 PM
Apr 24, 2017

Output

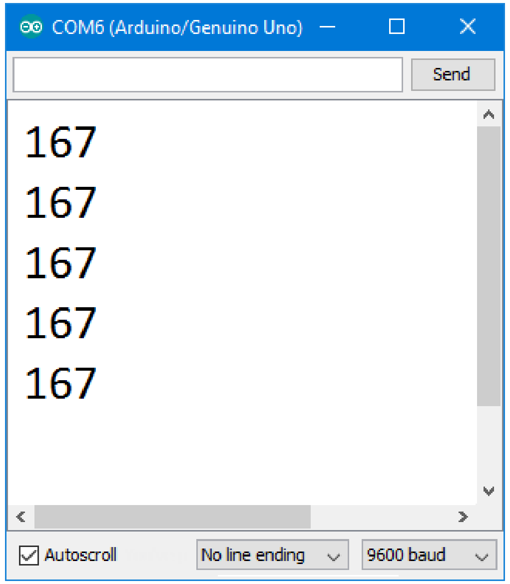
To resend the data, press and release the reset button



Transmitter

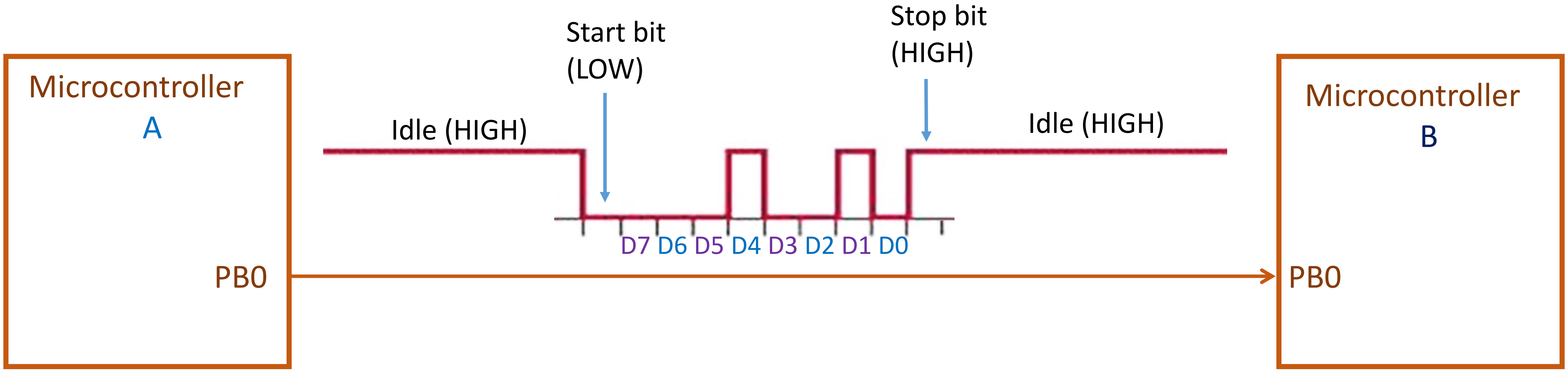
Receiver

After sending 5 times

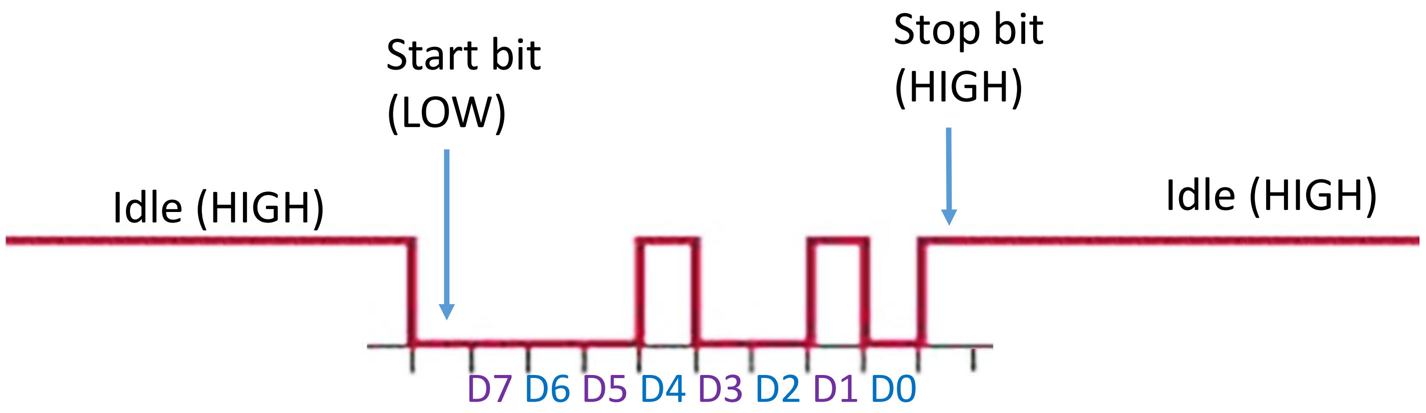


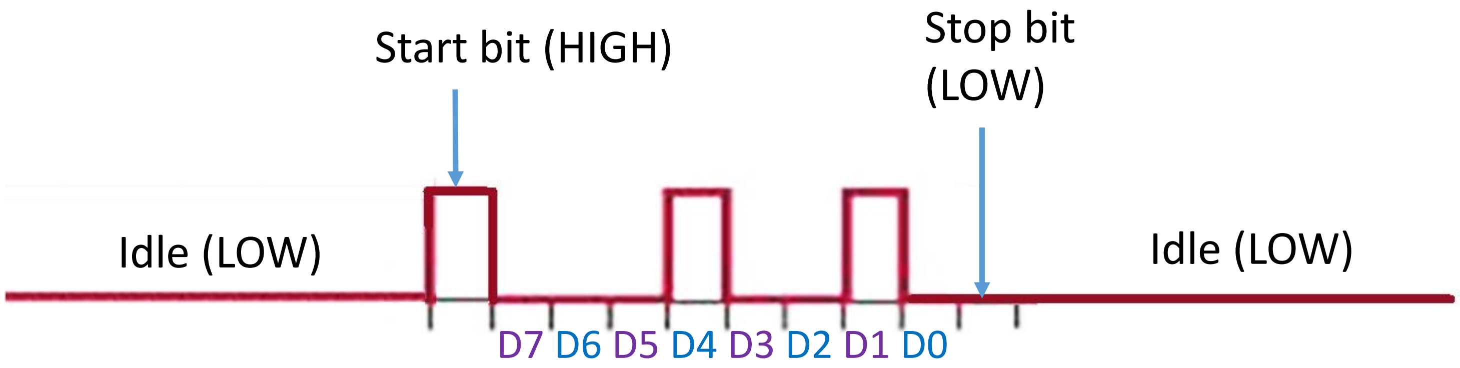
Asynchronous Transmission

- Sender and receiver need to agree on baud rate.
- Sender and receiver need to agree on whether the communication line is idle HIGH or idle LOW.
- Sender and receiver need to agree on the endianness (MSB first or LSB first?)
- Sender and receiver need to agree on whether any other information (such as parity) is to be added.



- Start bit = LOW for idle high and HIGH for idle low.
- End bit = opposite of start bit





Note: even if you are transmitting multiple bytes, every byte needs a start bit and an end bit.

Example 2

PB2 of two ATmega328p microcontrollers are connected as shown below. Assume that communication is idle low.

For the transmitter

Write a program that sends a byte of data asynchronously through PB2. Send MSB first. Required baud rate = 2000 Bd.

For the receiver

Write a program that keeps waiting for incoming data and prints the received byte through the serial port (as soon as it is received).

Code for Transmitter

```
char* ddrb = (char*) 0x24;
char* portb = (char*) 0x25;

int main()   //void setup() for Arduino
{
      *ddrb |= (1 << 2);   //PB2 is Data line. This code is same as pinMode(10, OUTPUT);

      Transmit(167, 2000); //Transmit 1010 0111 at 2kBd.
}
```

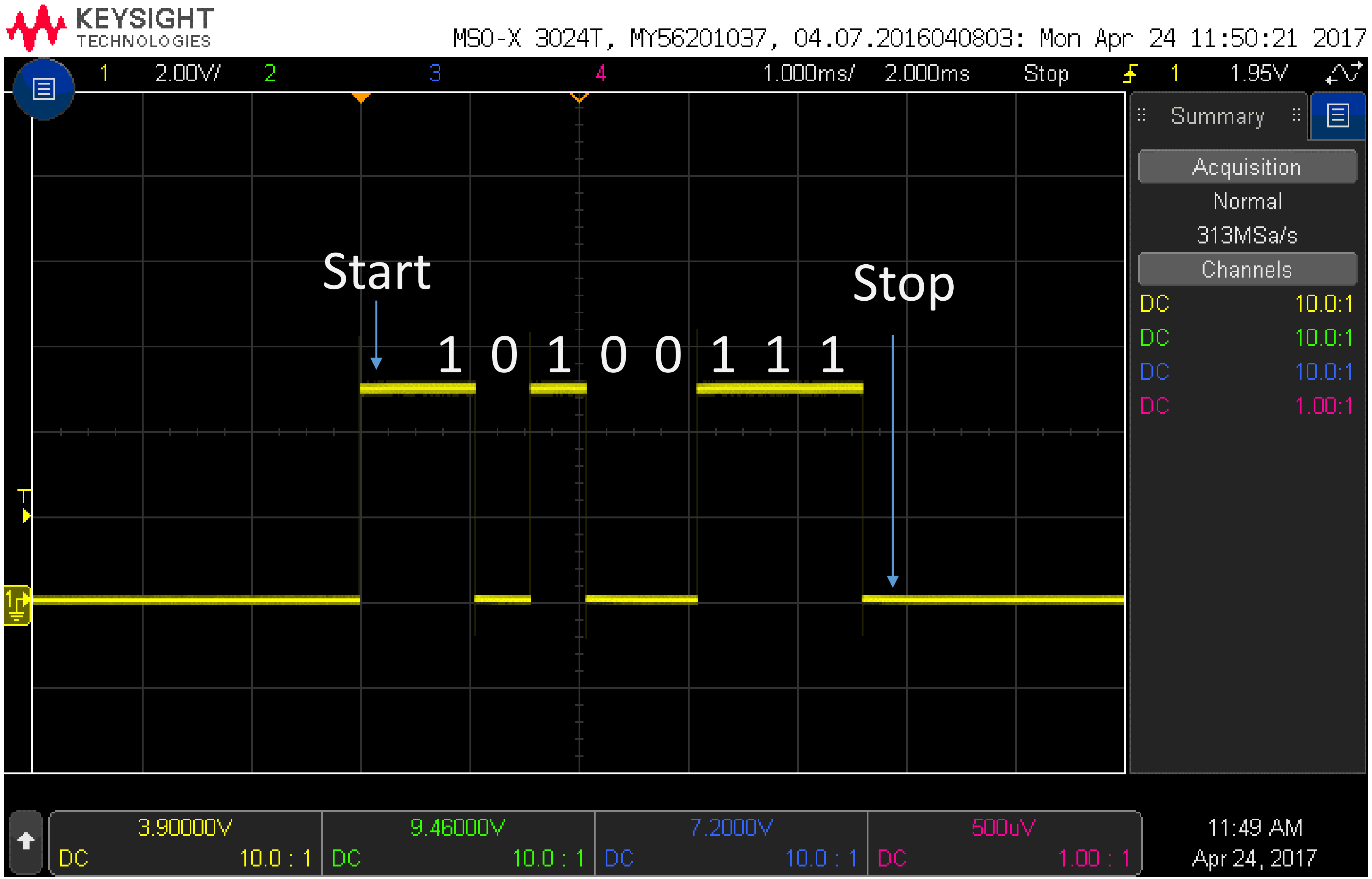
```c
void Transmit(unsigned char data, int BaudRate)
{
	unsigned long delay_us = 1000000/BaudRate;

	*portb |= 1 << 2;                   //Send start bit  (HIGH)
	delayMicroseconds(delay_us);

	for (int i=7; i>=0; i--)            //MSB first (decrementing loop)
	{
		if (data & (1 << i))            //If the ith bit is HIGH
		{
			*portb |= 1 << 2;                //Send HIGH bit
		}
		else
		{
			*portb &= ~(1 << 2);             //Send LOW bit
		}
		delayMicroseconds(delay_us);
	}
	*portb &= ~(1 << 2);                     //Send stop bit (LOW)
	delayMicroseconds(delay_us);
}
```

Code for Receiver

```
volatile char* pinb = (char*) 0x23;

int main()   //void setup() for Arduino
{
      Serial.begin(9600);

      while(1)
      {
            unsigned char data = Receive(2000);  //Get the incoming data
            Serial.println(data);                 //Print the received data
      }
}
```

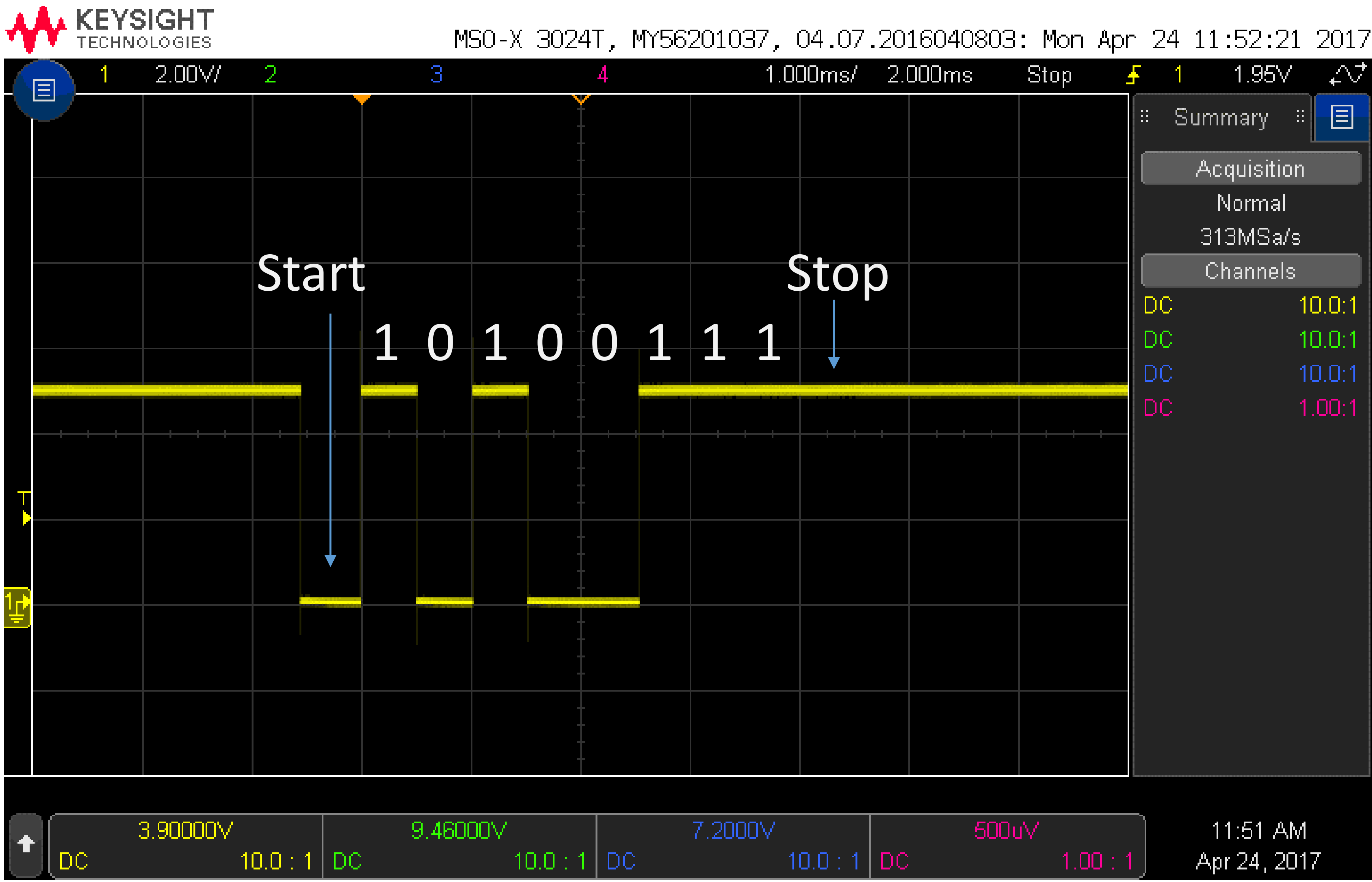
```
char Receive(unsigned long BaudRate)
{
      unsigned long delay_us = 1000000/BaudRate;
      bool previous = ((*pinb) & (1 << 2));         //previous = digitalRead(10);

      while(previous==((*pinb) & (1 << 2)))    //Wait for start-bit
      {
            //As long as there is no change from the idle state, keep waiting.
      }
      delayMicroseconds(delay_us * 1.1);            //Wait a bit longer than required span
                                                    //in order to avoid transition periods
      char ReceivedData;

      for (int i=7; i>=0; i--)                      //MSB first (decrementing loop)
      {
              if ((*pinb) & (1 << 2))               //If PB2 (data pin) is HIGH
              {
                      ReceivedData |= 1 << i;       //Set the ith bit
              }
              else
              {
                      ReceivedData &= ~(1 << i);   //Clear the ith bit
              }
              delayMicroseconds(delay_us);
       }
       return ReceivedData;
}
```

Idle low



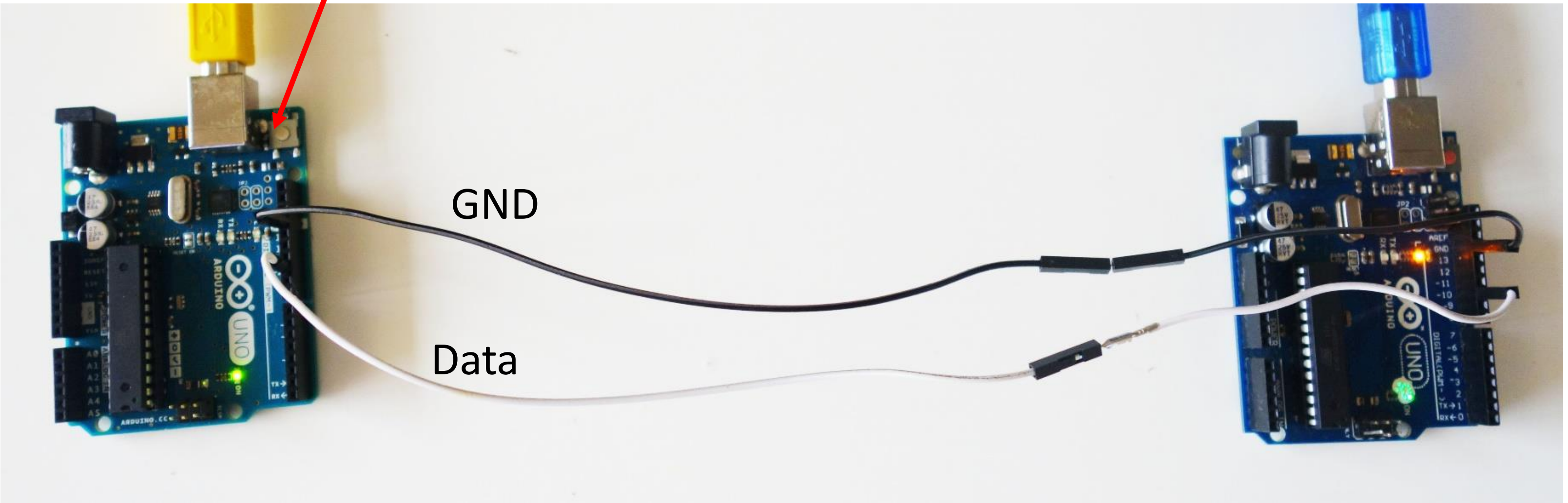
KEYSIGHT
TECHNOLOGIES
MSO-X 3024T, MY56201037, 04.07.2016040803: Mon Apr 24 11:50:21 2017
1 2.00V/ 2 3 4 1.000ms/ 2.000ms Stop 1 1.95V
Summary
Acquisition
Normal
313MSa/s
Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1
Start
Stop
1 0 1 0 0 1 1 1
3.90000V DC 10.0 : 1
9.46000V DC 10.0 : 1
7.2000V DC 10.0 : 1
500uV DC 1.00 : 1
11:49 AM
Apr 24, 2017

Idle high


KEYSIGHT TECHNOLOGIES
MSO-X 3024T, MY56201037, 04.07.2016040803: Mon Apr 24 11:52:21 2017
1 2.00V/ 2 3 4 1.000ms/ 2.000ms Stop 1 1.95V
Start
1 0 1 0 0 1 1 1
Stop
Summary
Acquisition
Normal
313MSa/s
Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 1.00:1
3.90000V DC 10.0 : 1
9.46000V DC 10.0 : 1
7.2000V DC 10.0 : 1
500uV DC 1.00 : 1
11:51 AM
Apr 24, 2017

Output

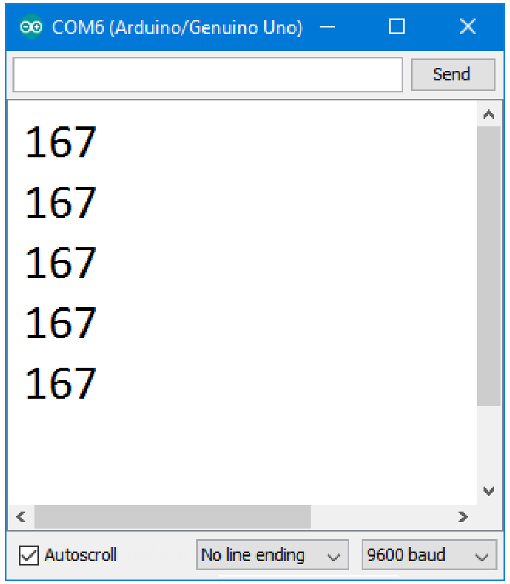
To resend the data, press and release the reset button



Transmitter

Receiver

After sending 5 times



Example 3 (Sending text)

Same set up as in example 2. This time the transmitter must transmit a string at 2kBd.

Answer: There are two ways.

First way

Call transmit() function for every character.

```
Transmit('Y', 2000);
Transmit('U', 2000);
Transmit('S', 2000);
Transmit('O', 2000);
Transmit('F', 2000);
Transmit('\n', 2000);
```

Second way

Write a function that loops though the string

```
void Transmit(char* str, unsigned long BaudRate)
{
      for (int i=0; str[i]!=0; i++)
      {
            Transmit(str[i], BaudRate);
      }
}
```

and then call that function

```
Transmit("YUSOF\n", 2000);
```

Transmitter

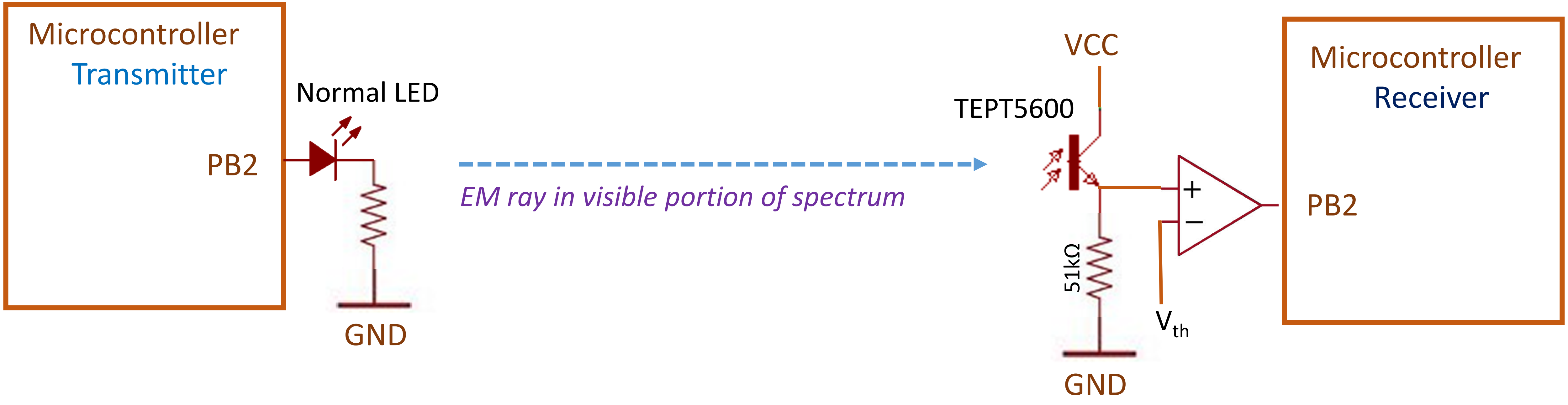
```
int main()   //void setup() for Arduino
{
      *ddrb |= (1 << 2);

      Transmit("YUSOF\n", 2000);
}
```

Receiver

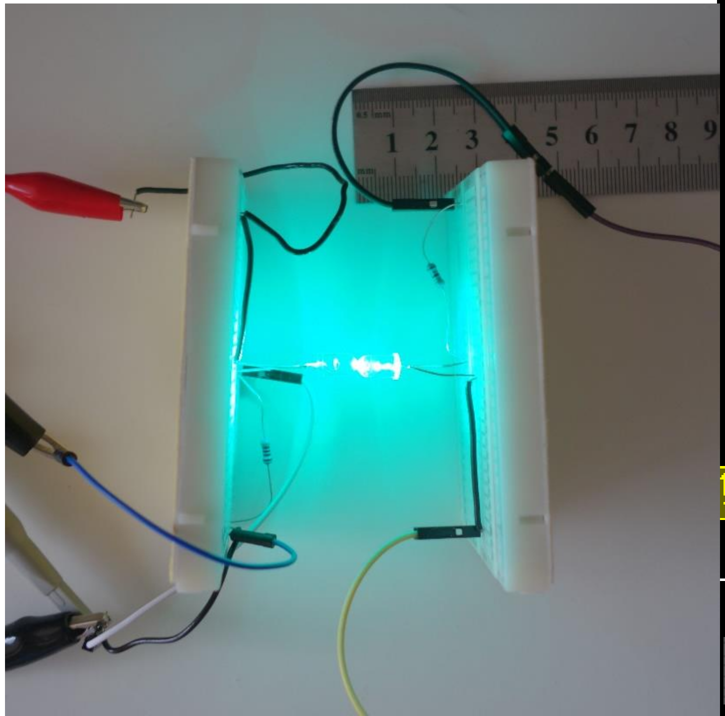
```
int main()   //void setup() for Arduino
{
      Serial.begin(9600);

      while(1)
      {
              char data = Receive(2000);
              Serial.print(data);
      }
}
```

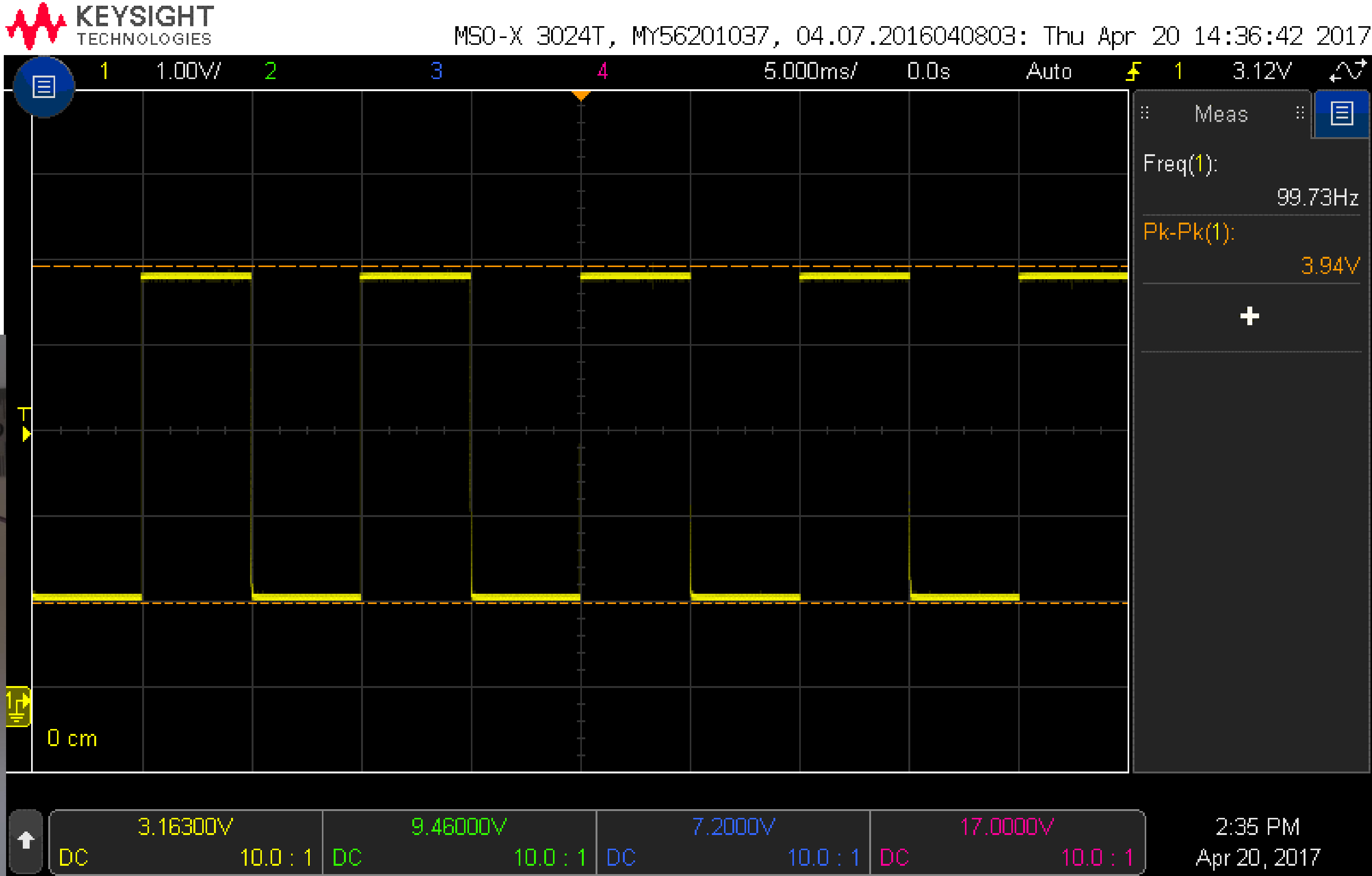
Baseband wireless transmission (proof-of-concept)

- Broadband transmission (with modulation) is often used in wireless communication.
- However, baseband wireless transmission is also possible in theory.

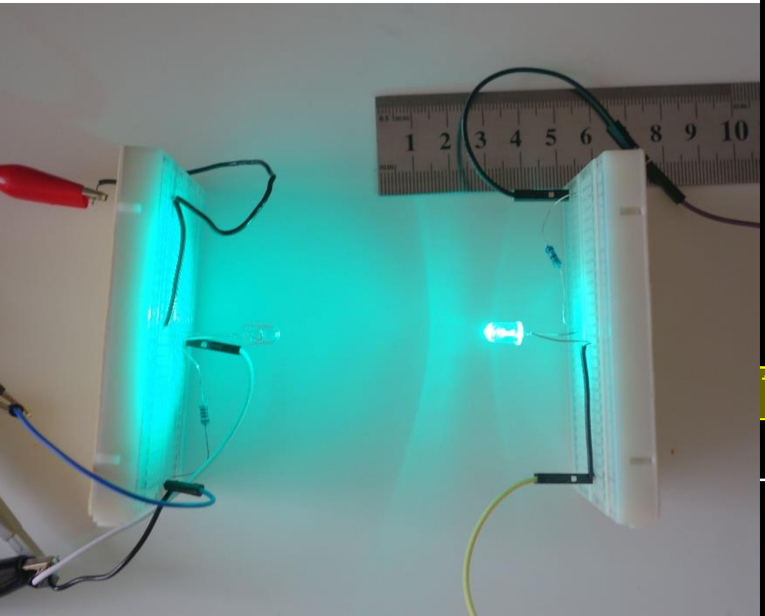


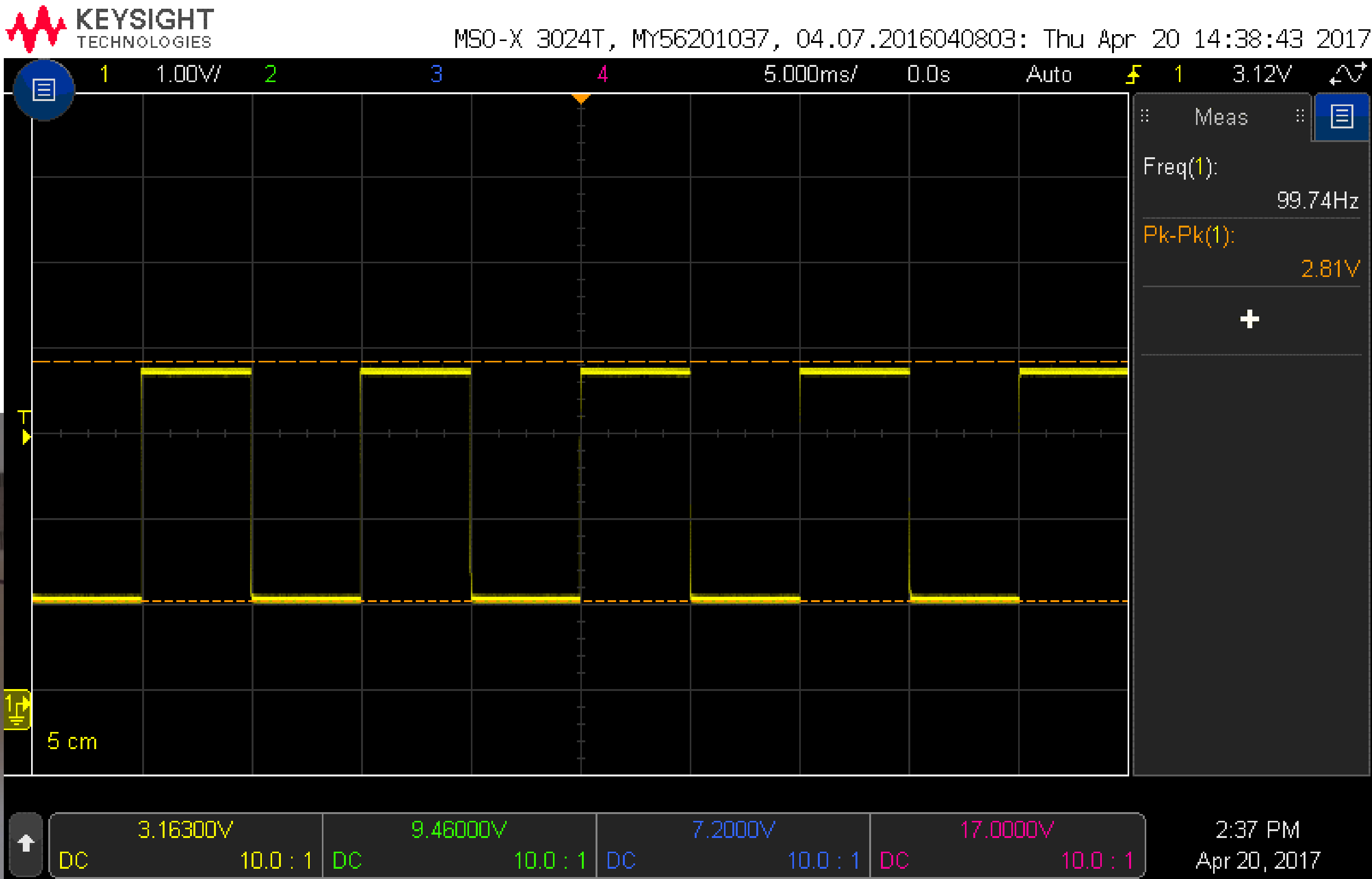
- Sender is generating 100Hz pulses at PB2. The receiver is placed 0cm away from the sender.
- The yellow waveform is experienced by the receiver at PB2.



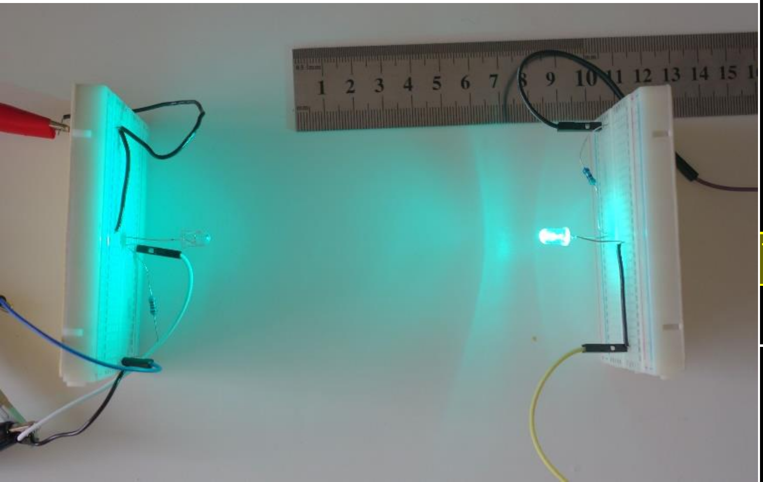


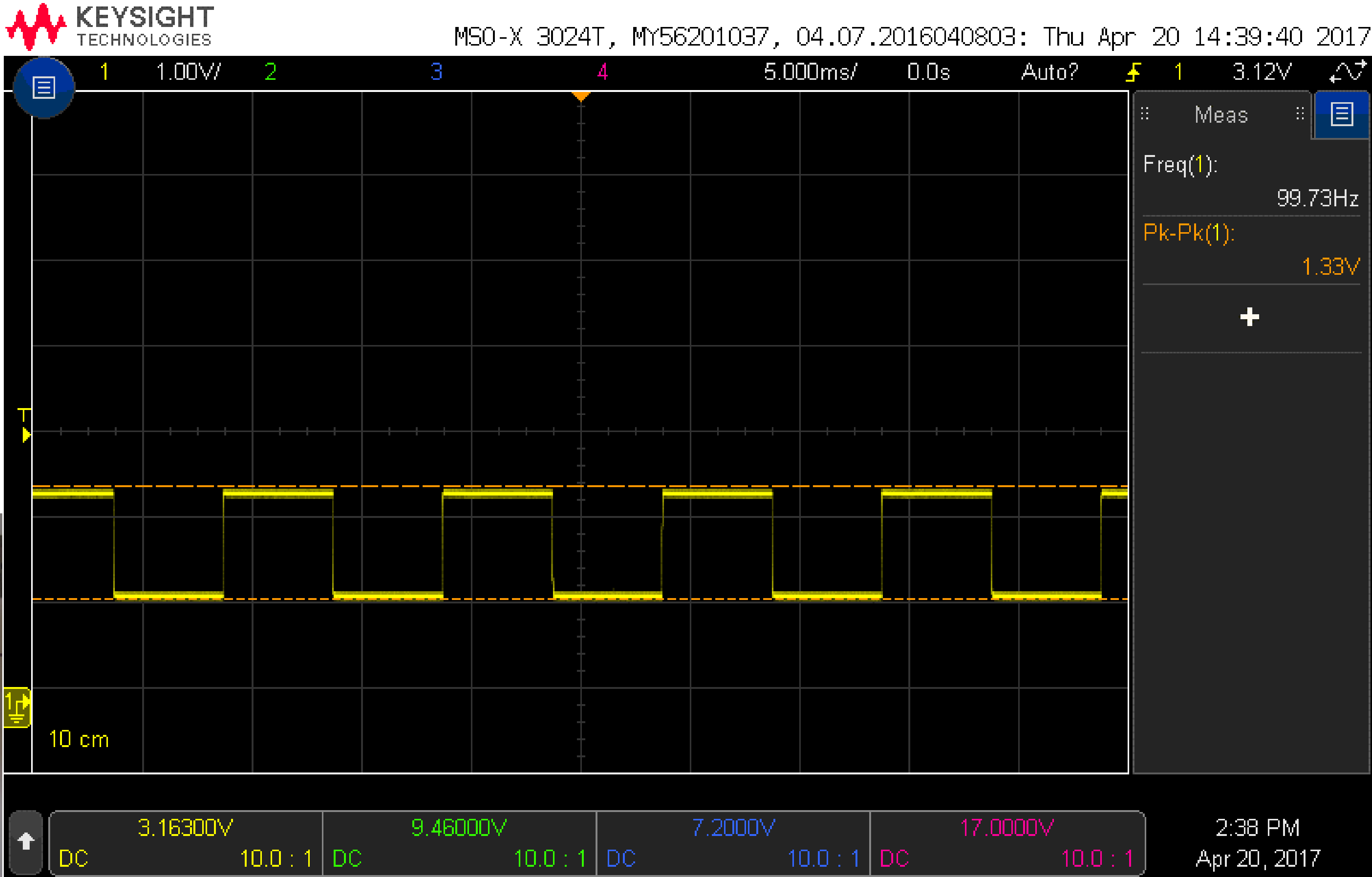
- The receiver is placed 5cm away from the sender.
- Amplitude is lower.



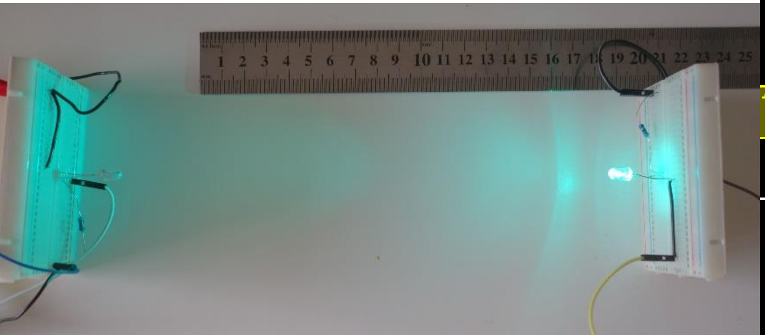


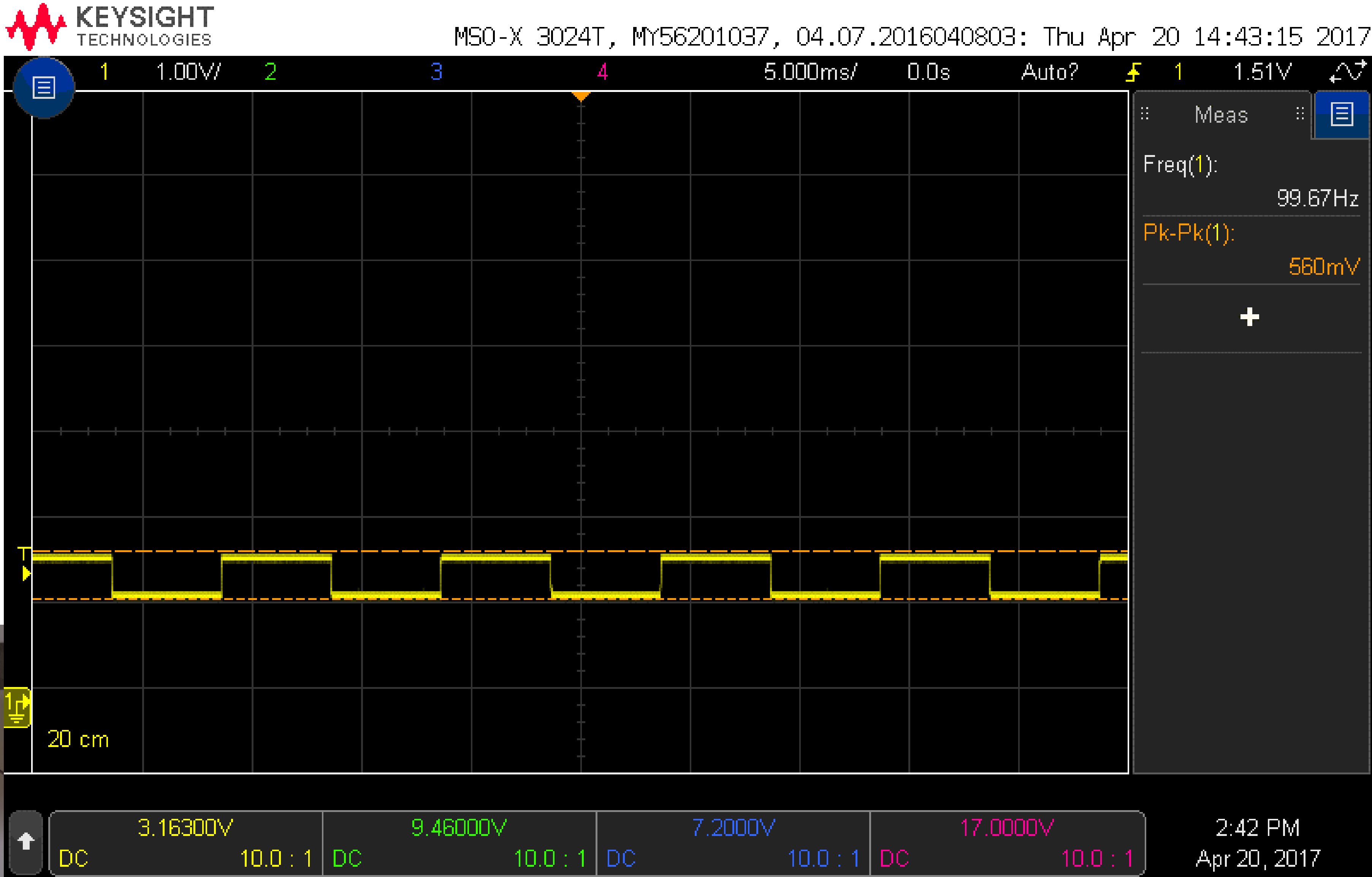
- The receiver is placed 10cm away from the sender.



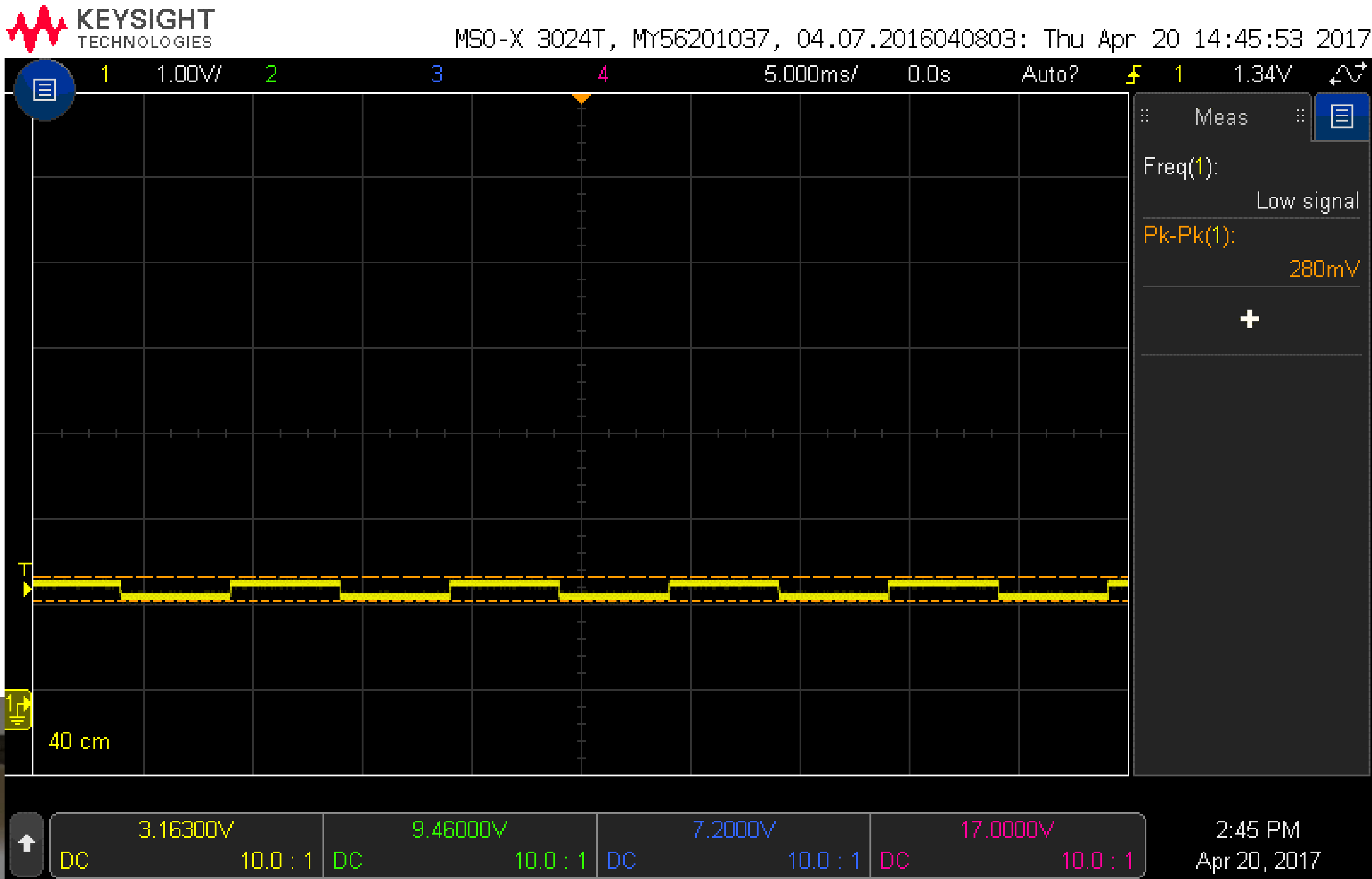


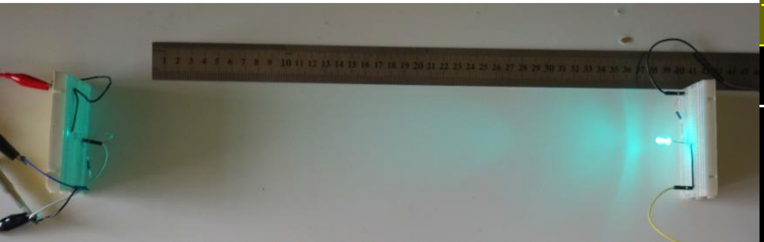
- The receiver is placed 20cm away from the sender.





- The receiver is placed 40cm away from the sender.

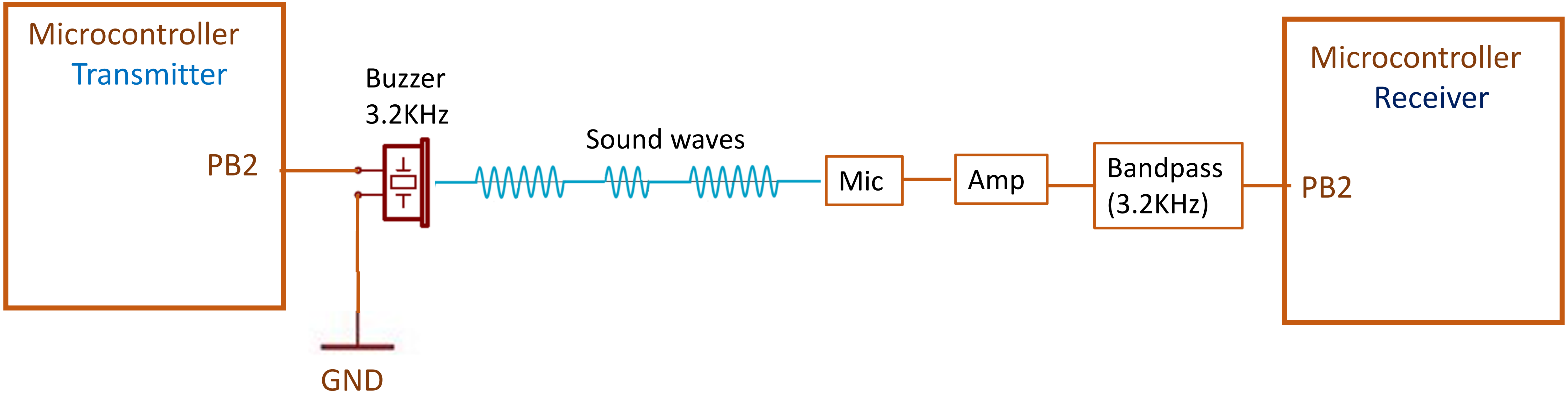


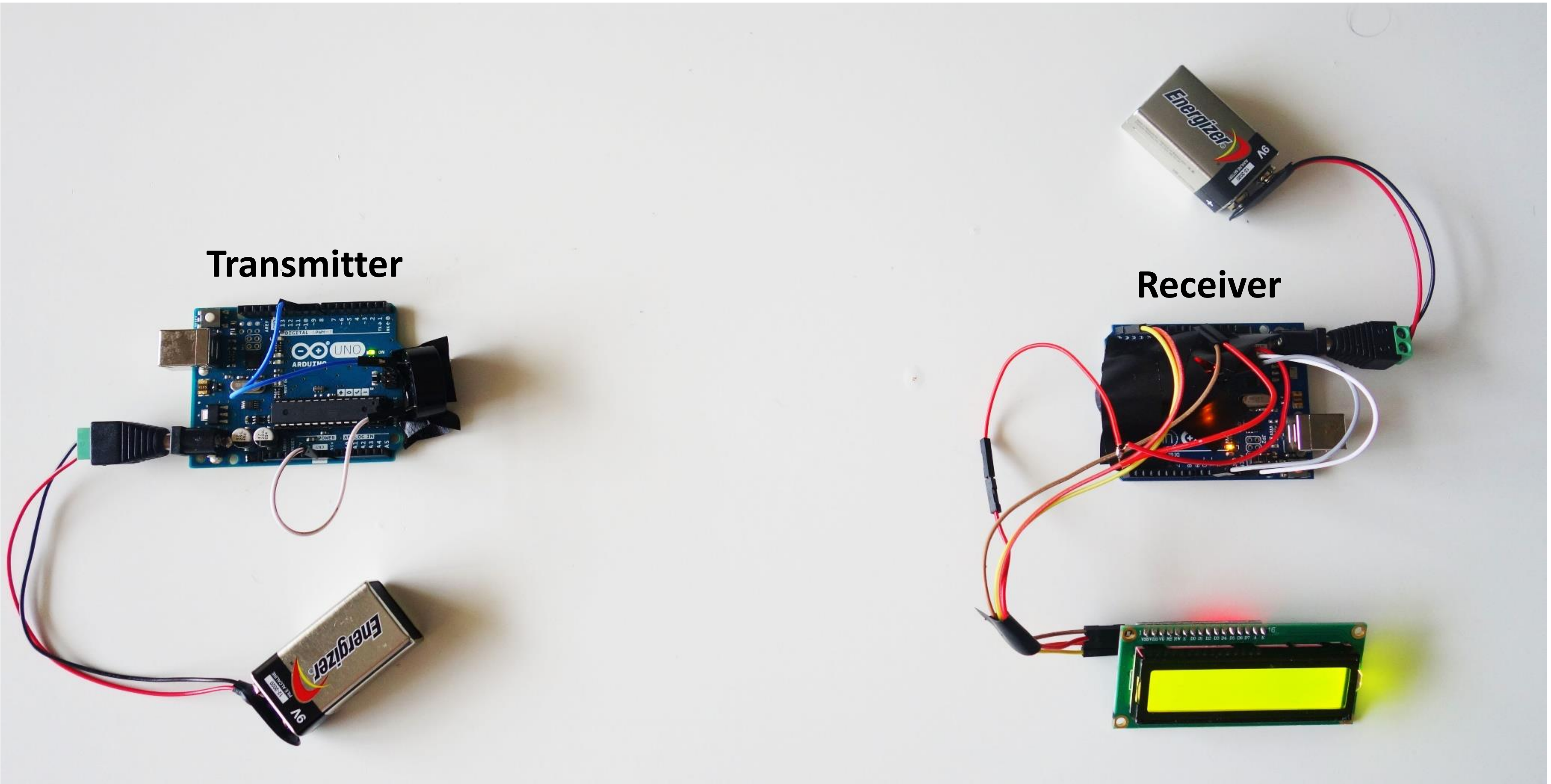



Receiver
Transmitter

Broadband transmission (Sound)

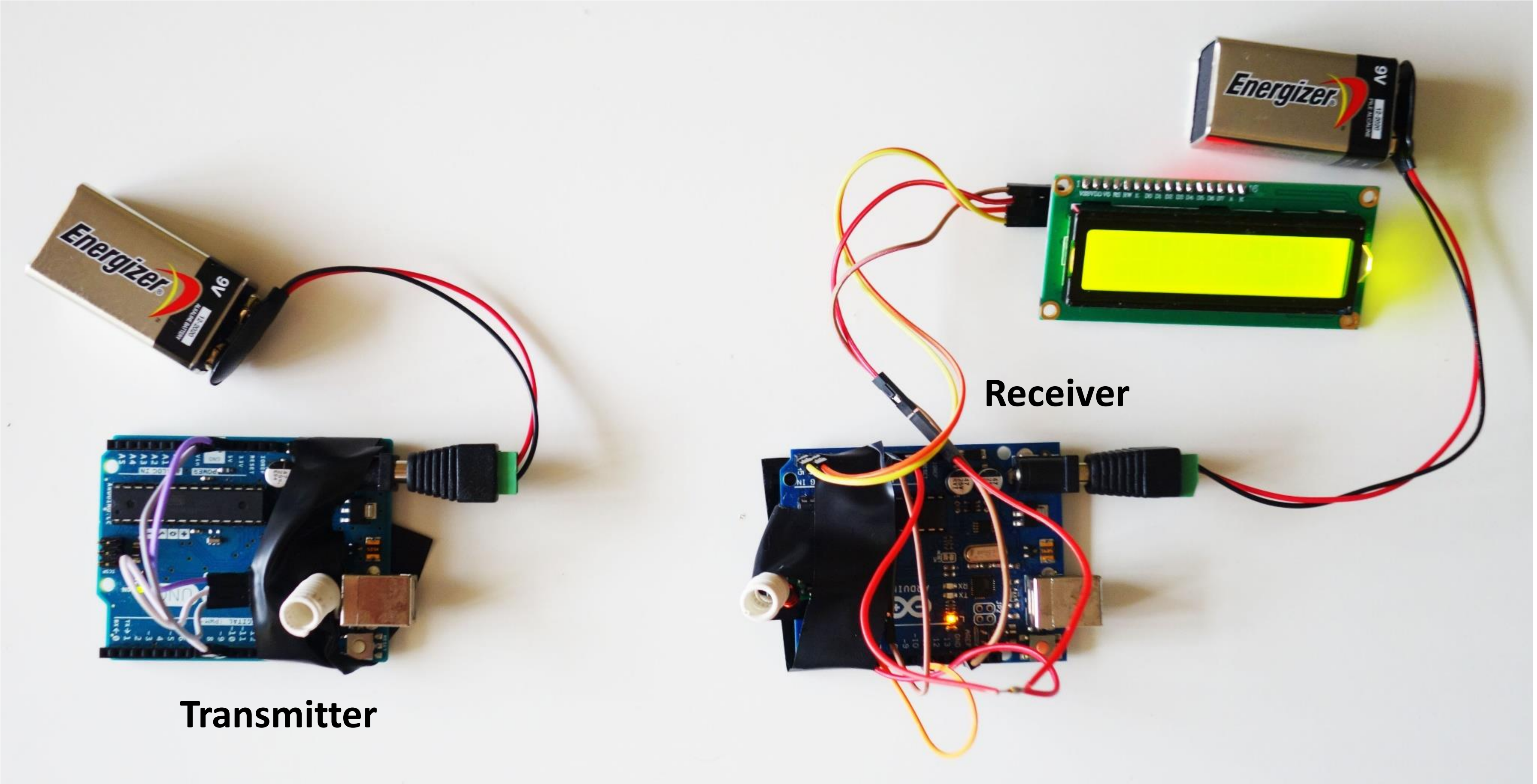
- When PB2 is HIGH, the buzzer creates a 3.2KHz sinusoidal sound waves. The receiver needs to demodulate it.




Transmitter
Receiver

Broadband transmission (RF)

- RF modulator and demodulator circuit form an RF link.
- Ideally, when PB2 of transmitter is HIGH, PB2 of receiver becomes HIGH (and vice versa).
- In reality, the receiver receives a great deal of noise.


Energizer 9V
Energizer 9V
Receiver
Transmitter

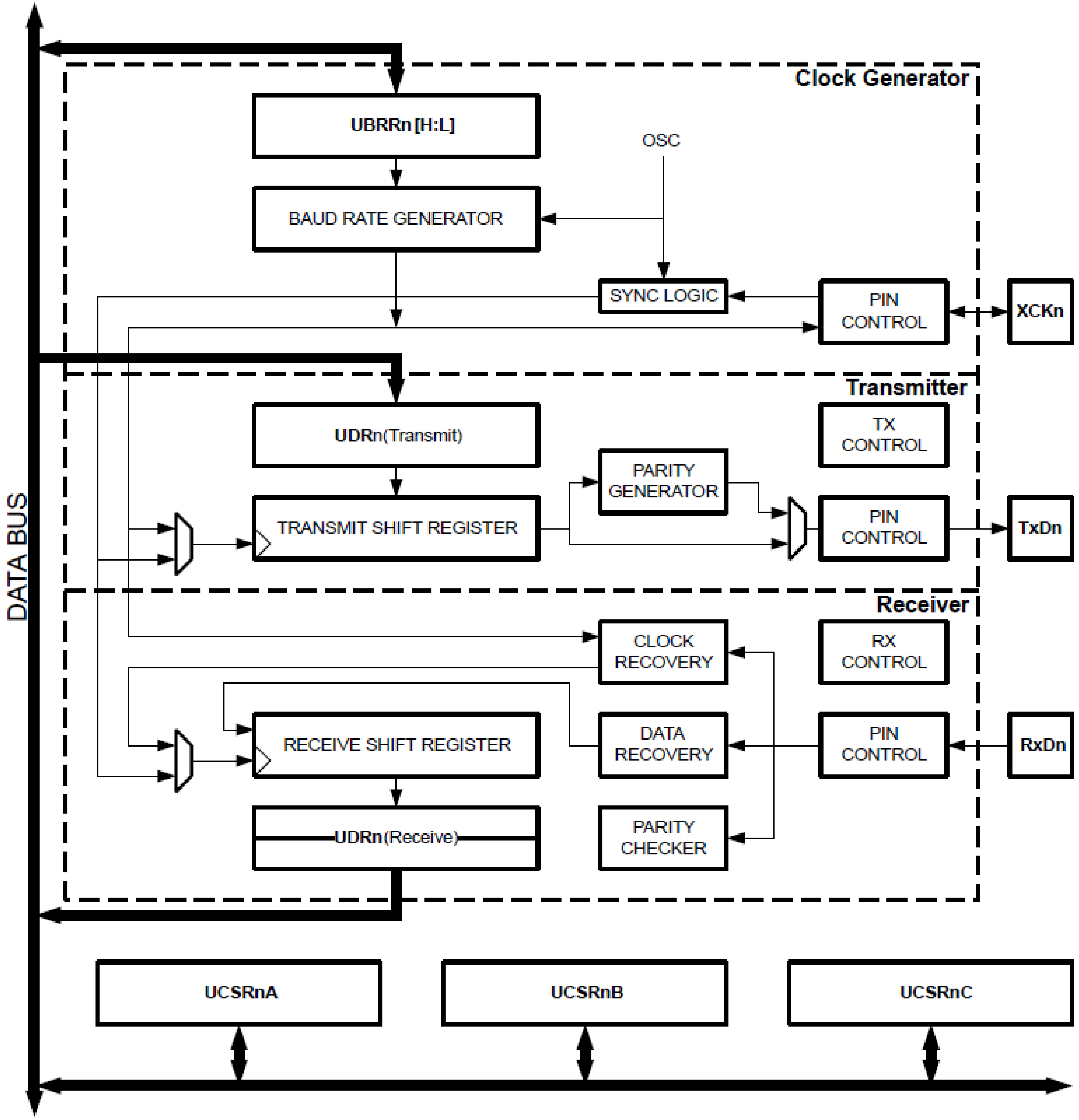
Serial communication on ATmega328p

ATmega328p contains hardware for three serial communication protocols:

- USART (Universal Synchronous Asynchronous Receiver Transmitter)
- I^2C (Inter-integrated Circuit)
- SPI (Serial Peripheral Interface)

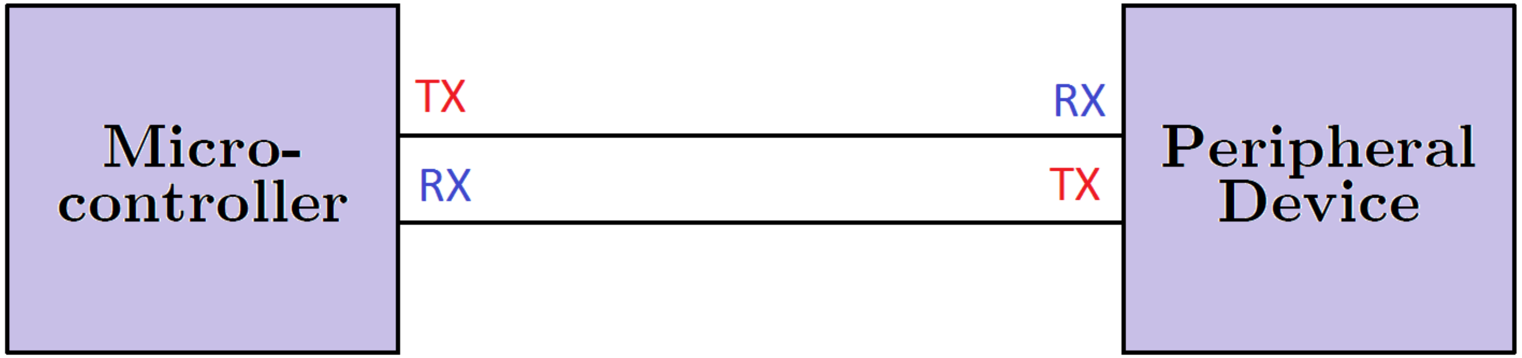
USART

The USART hardware on ATmega38p can be configured as either asynchronous (UART) or synchronous.



UART

- Full duplex asynchronous serial communication
- RX is shared with PB0
- TX is shared with PB1
- Arduino Serial class uses UART
- Stop bit length (1 or 2), parity (even, odd, none), data frame size (5 to 9 bits), baud-rate (pre-scaler) can be changed via registers.



Advantage

- Simple
- Bidirectional

Disadvantage

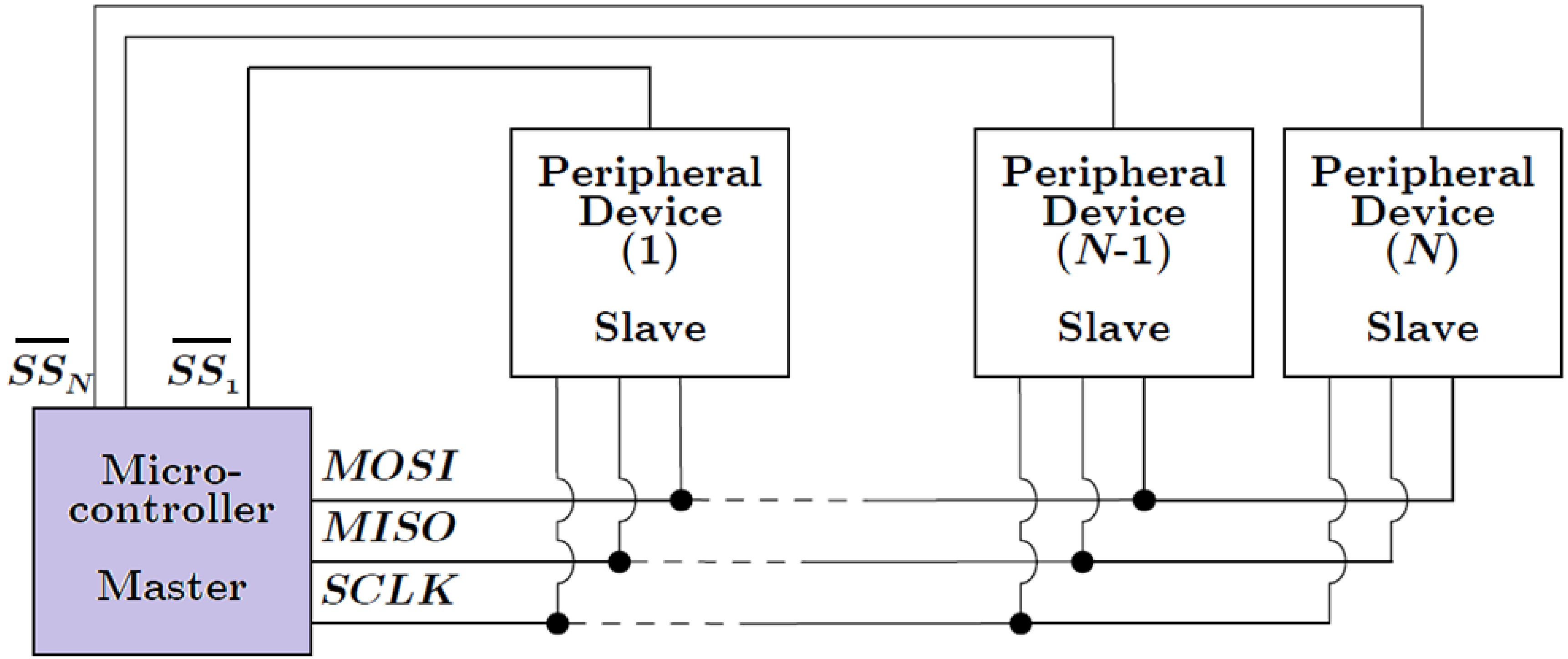
- Supports one to one communication only

SPI

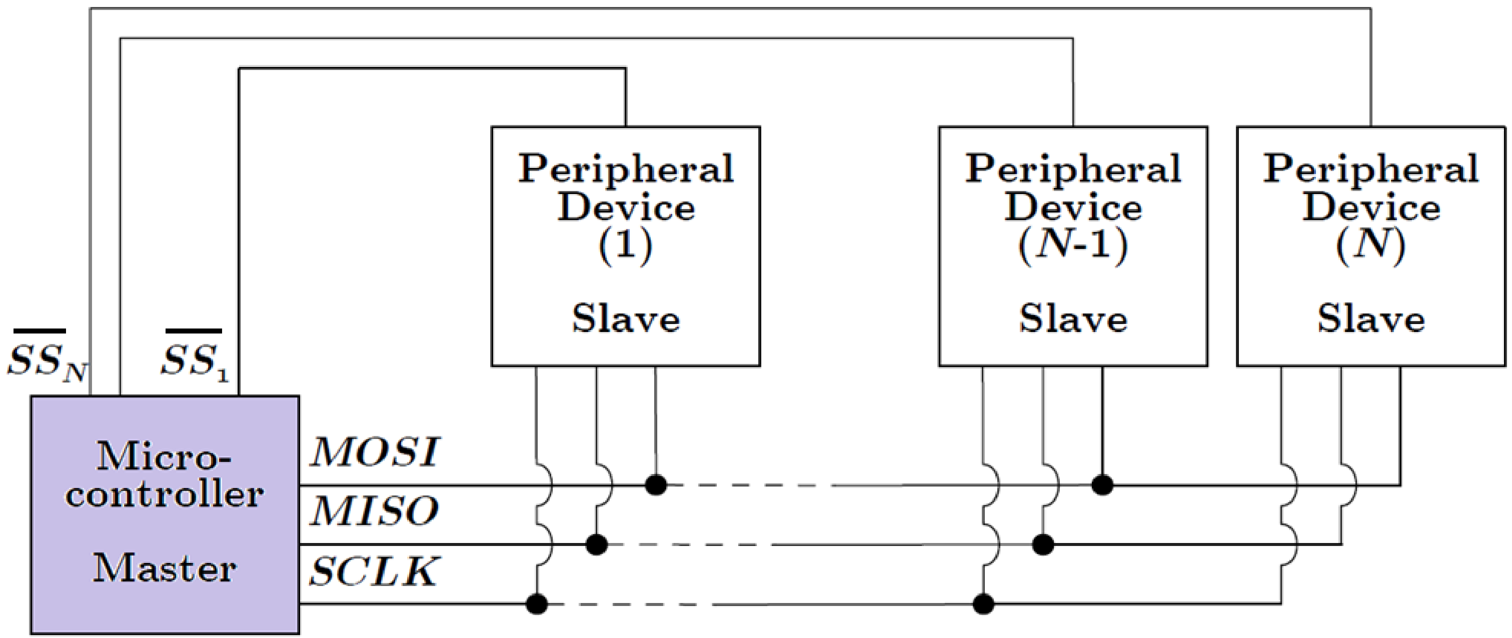
- Developed by Motorola
- Requires at least 3 wires
  1. MOSI (Master out slave in)
  2. MISO (Master in slave out)
  3. SCLK (Serial clock)

$\overline{SS}$ = Slave select line (need if there are multiple slaves)

For example
If there are 10 slaves, 10 $\overline{SS}$ lines are required.



- Only the master controls the clock.
- SS is active low (HIGH by default and when the master wants to communicate with a particular slave, the respective SS line must be made LOW)



Advantage

- Supports multiple slaves
- Bidirectional

Disadvantage

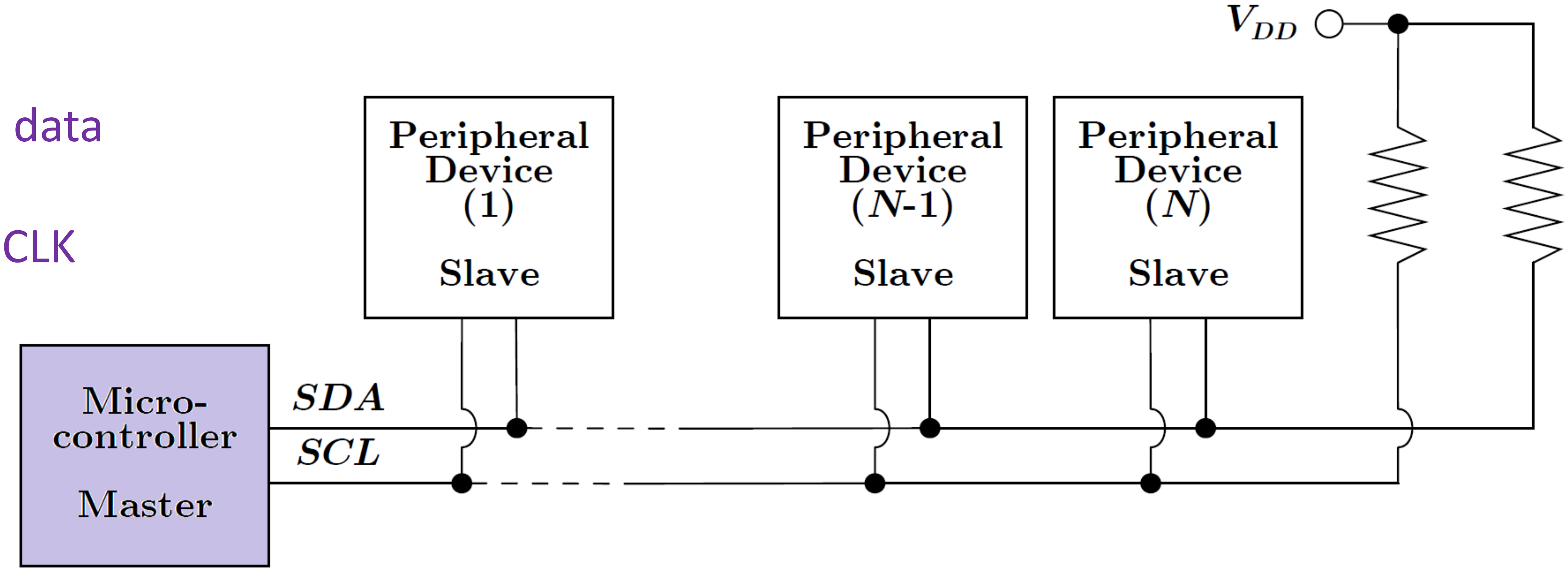
- A slave can only respond to the master when the master requests
- Slaves cannot talk to each other
- Separate SS line is required for each slave (Not really scalable)

I^2C

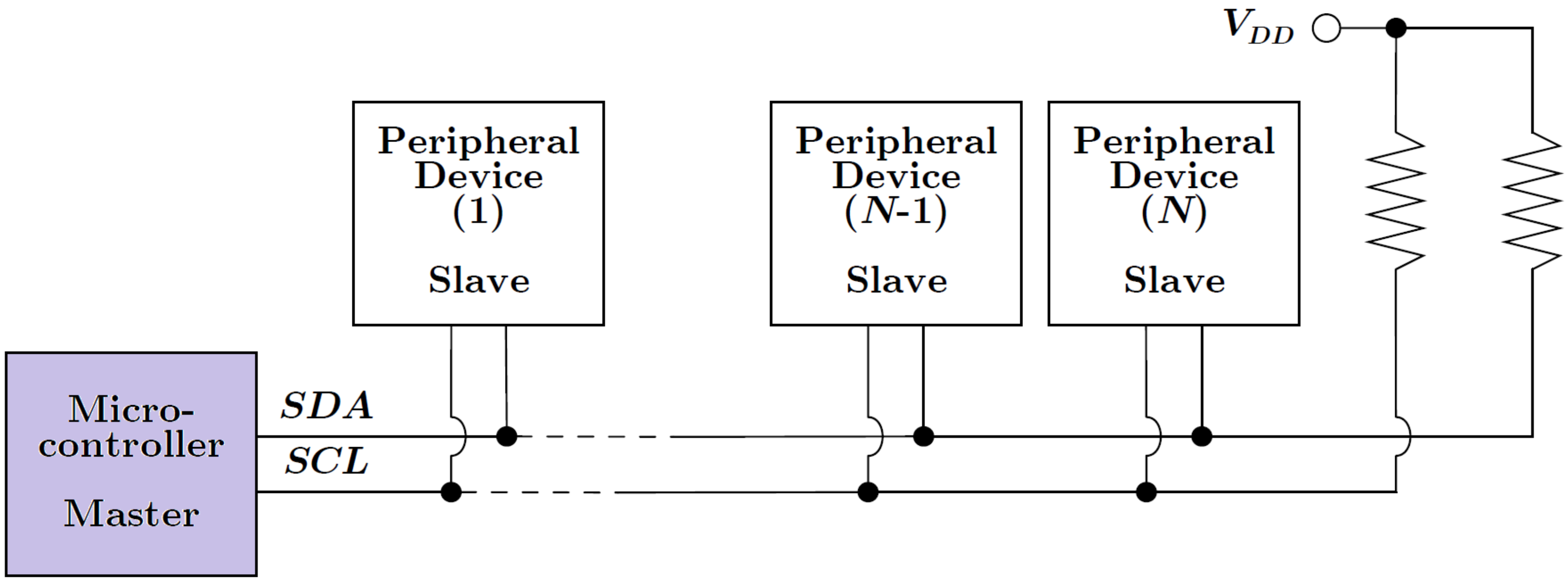
- Developed by NXP Semiconductors.
- Every slave has an address.
- Requires only two wires for up to 2^N slaves using N-bit slave addressing.
- Requires two external pull up resistors

SDA = Serial data

SCL = Serial CLK



- The Master beings communications by transmitting a single start bit followed by the address of slave device (which the master is trying to access)
- The slave responds with an acknowledgement bit if it is present on the serial bus.
- The master continues by either writing data to the slave or listing for data from the slave.



Advantage

- Supports multiple slaves

Disadvantage

- Half-duplex

I²C vs SPI comparison

- SPI is faster (CLK can go MHz range)
- SPI requires more wires
- SPI is full-duplex
- SPI is simpler to implement

- I²C is slower (around 100-400kHz*)
- I²C requires less wires
- I²C is half-duplex
- I²C is more difficult to implement

*Note:

Recent versions of I²C claimed to support of MHz clock rate