Introduction to Arduino Prototyping

Design and Development with Arduino

Nicola Dall'Ora

Sebastiano Gaiardelli

UNIVERSITÀ
di VERONA
Dipartimento
di INFORMATICA

Cyber-Physical & IoT Systems Design

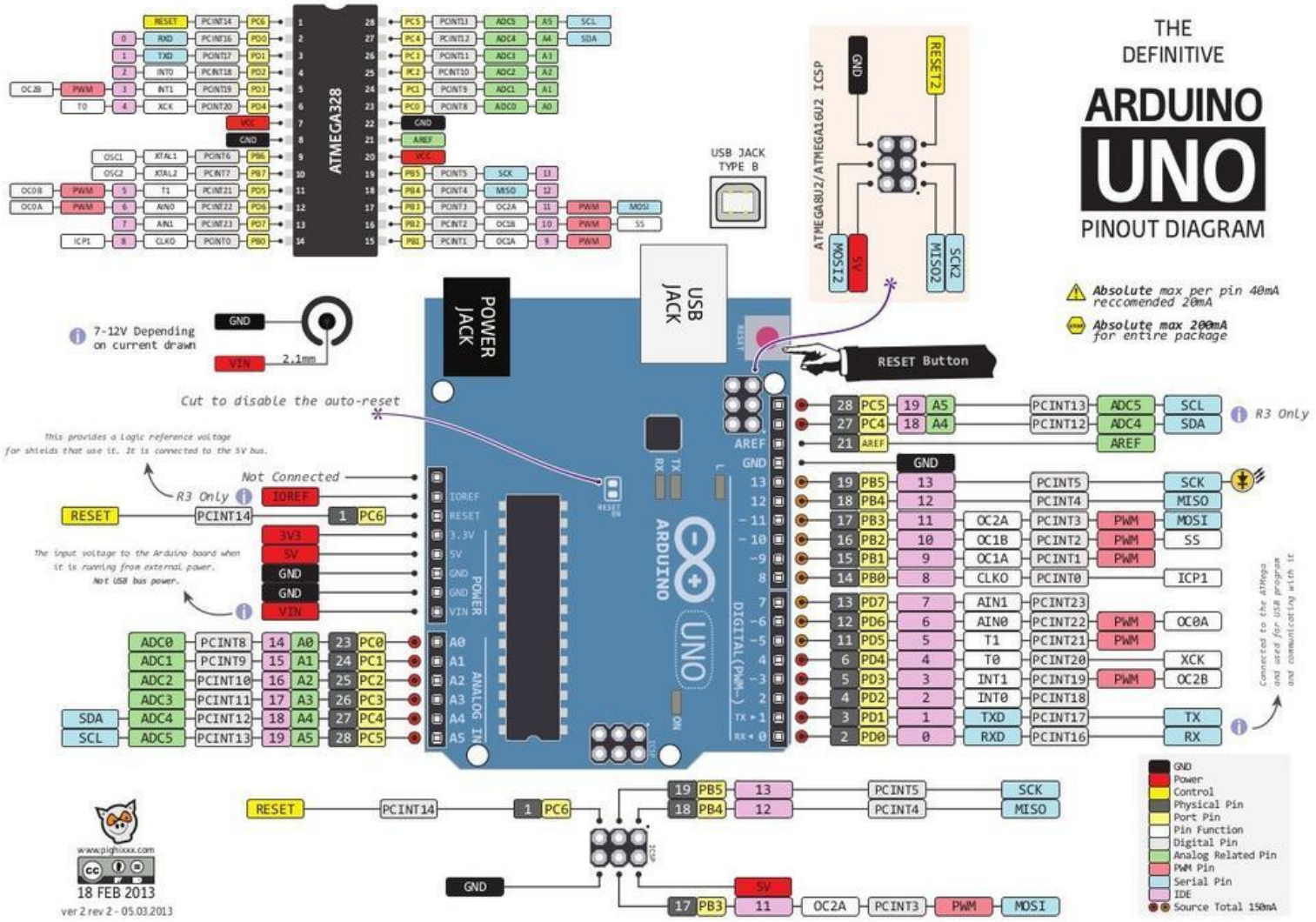
Outline

- Objectives
- Arduino Framework
- Arduino Communication Protocols
- Wireless communication with Arduino – Basics
  - Wireless interface (NRF24L01)
- Wireless communication with Arduino – Scenario 1
  - Mounting instructions
- Wireless communication with Arduino – Scenario 2

Objectives

- Today's lecture has the followings objectives:
  - Learn how to write a correct sketch of code for Arduino;
  - Introduction to wired communication protocols;
  - Wireless interface NRF24L01;
  - Start with a simple circuit for wireless communication.

Arduino Uno Pinout Diagram



Arduino Framework

- Based on MbedOS
  - Open-source embedded real-time operating system (RTOS);
    - Implements only a subset of the functionalities of MbedOS;
  - Written in C++;
    - Not all the C++ libraries can be used with Arduino (e.g., POSIX Thread, Semaphore);

- Arduino Programming Language: https://www.arduino.cc/reference/en/
- C++ guide: https://github.com/federico-busato/Modern-CPP-Programming
- MbedOS: https://os.mbed.com/mbed-os/
- Write clean code! (https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/)

Arduino Main() Function

```
1 int main () {
2     init ();
3     initVariant ();
4 #if defined(SERIAL_CDC)
5     PluggableUSBD ().begin ();
6     _SerialUSBD.begin (115200);
7 #endif
8
9     setup ();
10    for (;;) {
11        loop ();
12        if (arduino::serialEventRun) arduino::serialEventRun ();
13    }
14
15    return 0;
16}
```

Interrupt

- An interrupt is an **event** that alters the sequence in which the processor executes instructions;
- An interrupt might be:
  - **planned** (specifically requested by the currently running program);
  - **unplanned** (caused by an event);

- The raising behavior of an interrupt can be:
  - LOW: The interrupt event is raised when the signal value is low;
  - CHANGE: The interrupt event is raised each time the signal value changes;
  - RISING: The interrupt event is raised each time the signal value changes from LOW to HIGH
  - FALLING: The interrupt event is raised each time the signal value changes from HIGH to LOW

Interrupt

- The function **attachInterrupt** allows monitoring a pin through an interrupt.
  - The pin of Arduino sensible to signal variations pin 2 and 3, these are specified in the function as 0 and 1;

- The function that handles the interrupt is called Interrupt Service Routine (ISR)

Interrupt

- The function **attachInterrupt** allows monitoring a pin through an interrupt.
  - The pin of Arduino sensible to signal variations pin 2 and 3, these are specified in the function as 0 and 1;
- The function that handles the interrupt is called Interrupt Service Routine (ISR)

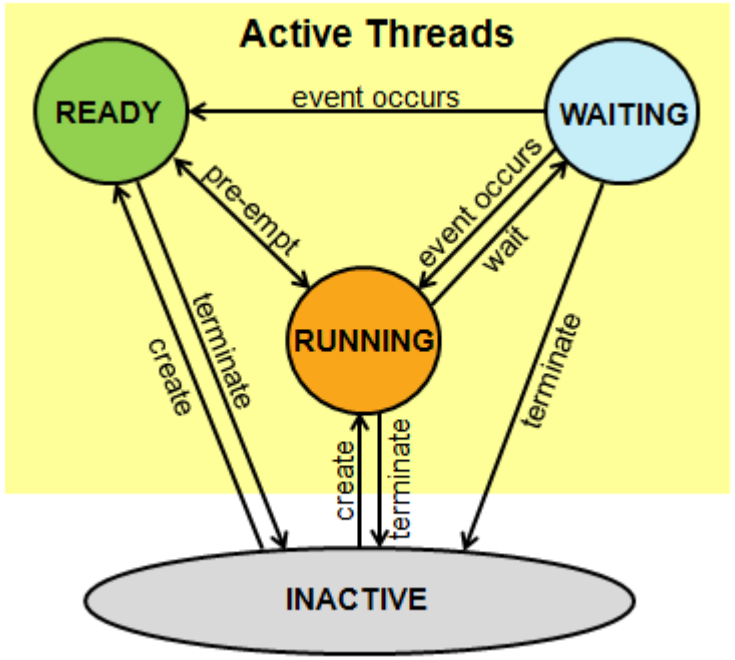
Use the volatile keyword if you have variables shared between the ISR and the loop function

```
1 int pin = 8;
2 volatile int state = LOW;
3
4 void setup(){
5     pinMode(pin, OUTPUT);
6     digitalWrite(pin, LOW);
7     //interrupt associated with the pin 3
8     attachInterrupt(1, ISR, LOW);
9 }
10 …
11 void ISR(){
12     state = !state;
13     digitalWrite(pin, state);
14 }
```

Thread

- Arduino does not support isolated parallel tasks.
  - You can use external libraries to introduce them

```
1 int main () {
2     Thread myThread = Thread ();
3     myThread.enabled = true;
4     myThread.setInterval (10); //10ms
5     myThread.onRun (callback_function);
6     if (myThread.shouldRun ()){
7         myThread.run ();
8     }
9 }
```

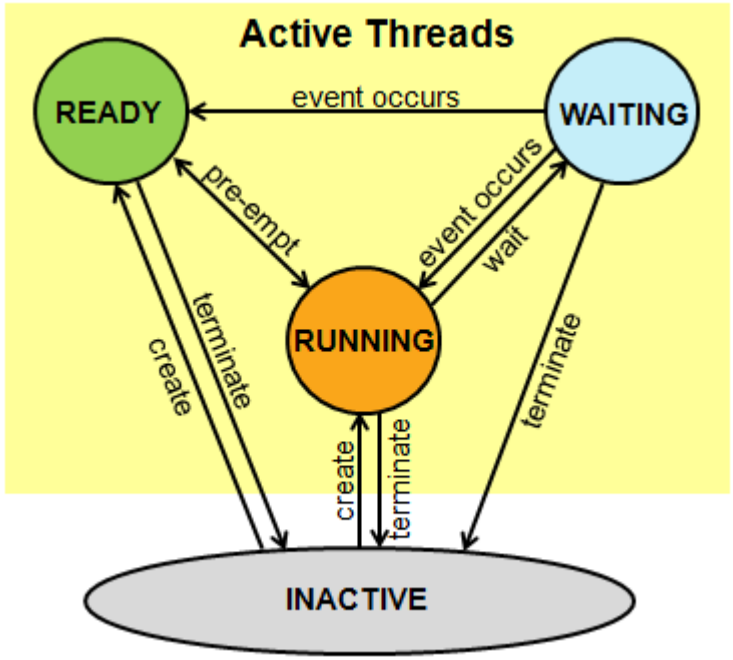


- More info: https://github.com/ivanseidel/ArduinoThread

Thread

- Arduino does not support isolated parallel tasks.
  - You can use external libraries to introduce them

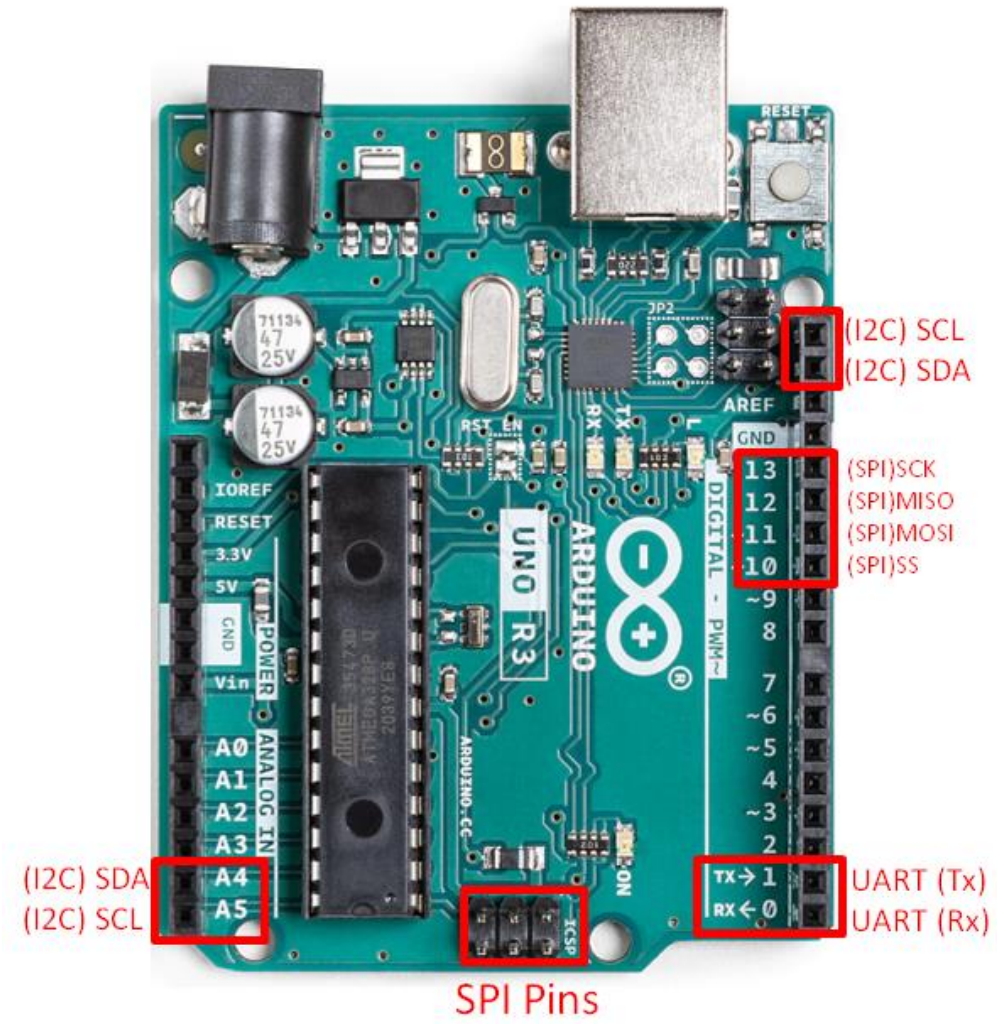
```
1 int main () {
2     ThreadController controller = ThreadController ();
3     // define threads
4     controller.add (&myThread1);
5     controller.add (&myThread1);
6     controller.run ();
7 }
```



- More info: https://github.com/ivanseidel/ArduinoThread

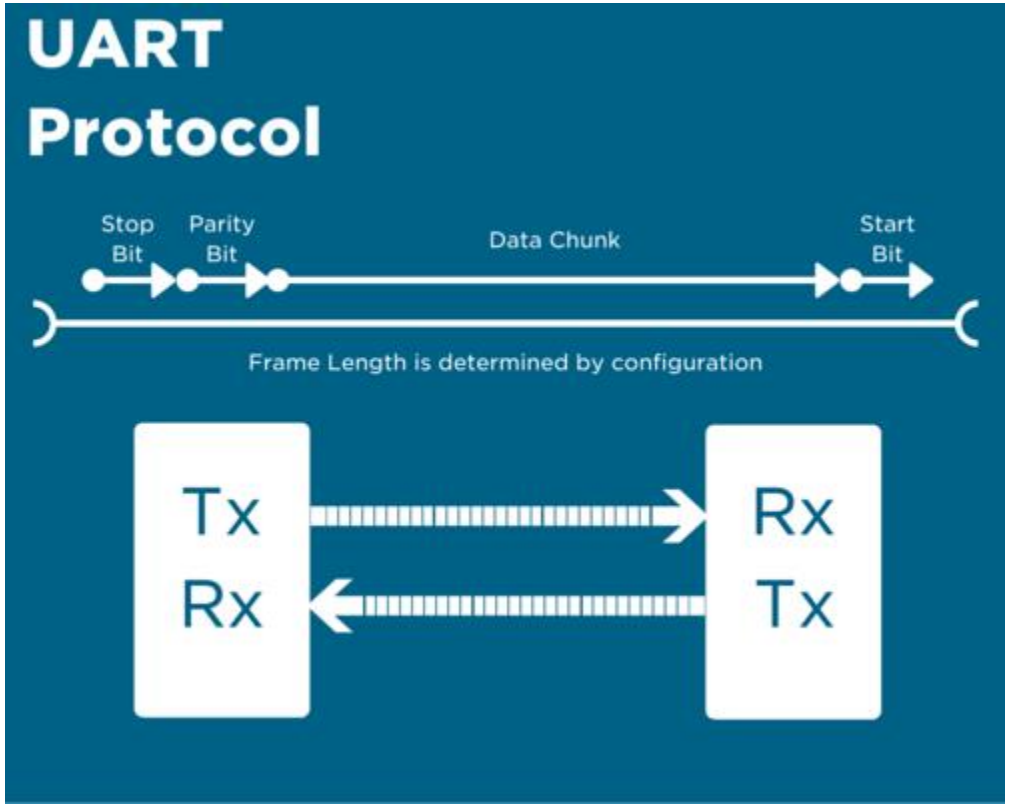
Arduino Communication Protocols

- Communication among different electronic devices like Arduino is standardized among these three protocols; it enables designers to communicate between different devices easily without any compatibility issues
- Arduino have multiple peripherals attached to it; among them there are three communication peripherals used in Arduino boards
  - **UART, SPI, I2C**



Arduino Serial Communication (UART)

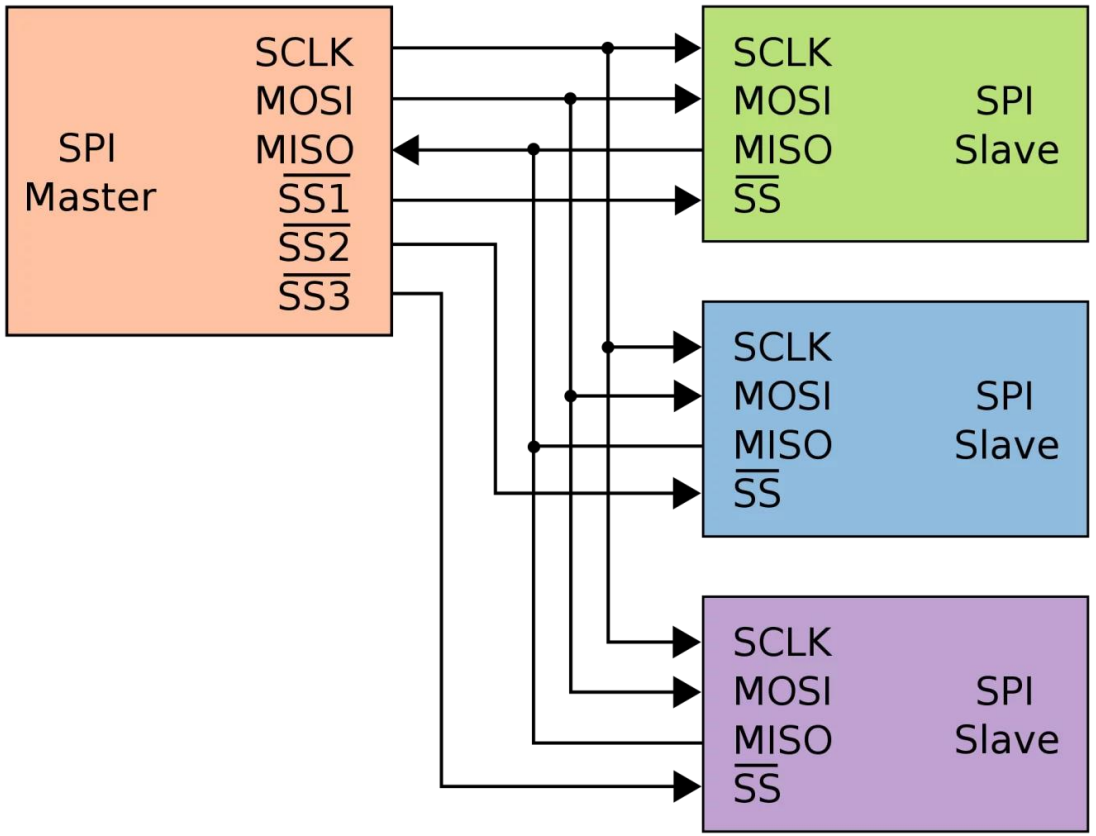
- Serial communication
  - Serial communication is the process of communicating between electronic devices one bit at a time. Serial means one after the other, so we think of serial communications as transferring data one bit at a time, one bit after the other
  - We use serial communications every time we click the upload button in our IDE. If the Arduino is receiving bits the Rx pin will flash. If the Arduino is transferring bits, the Tx pin will flash.



More info: https://core-electronics.com.au/guides/serial-communications-arduino-uno/

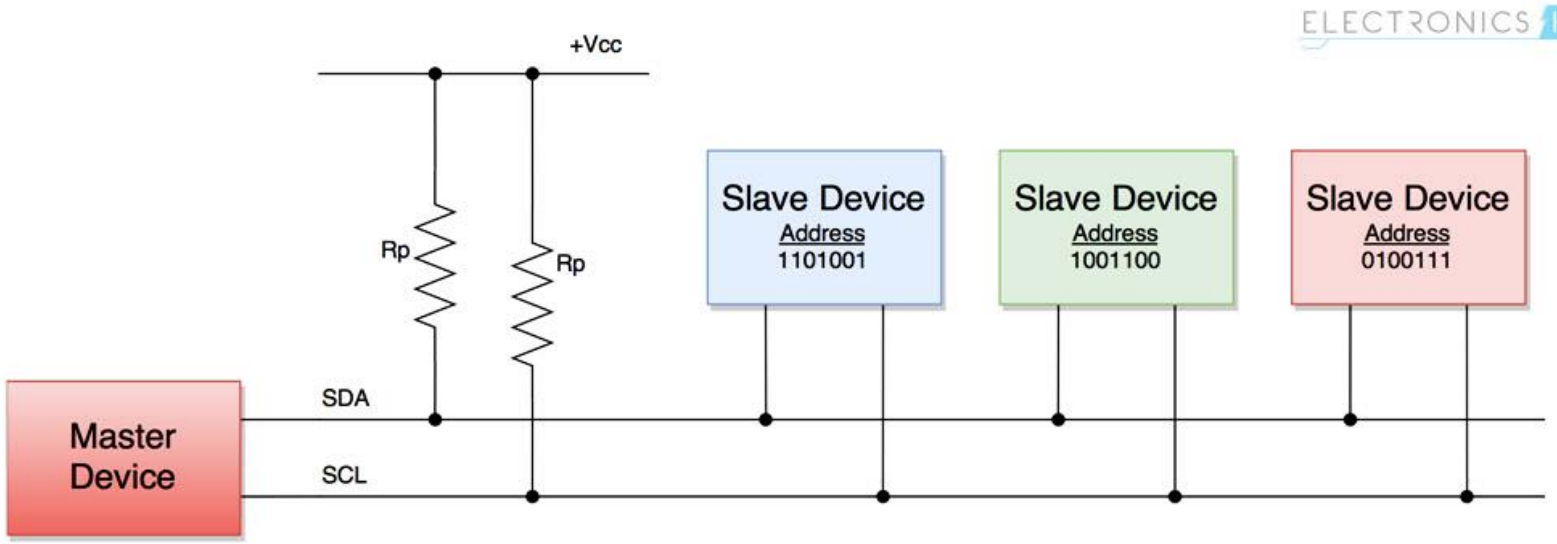
SPI Communication

- **Arduino & Serial Peripheral Interface (SPI)**
- Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances.
- https://docs.arduino.cc/learn/communication/spi

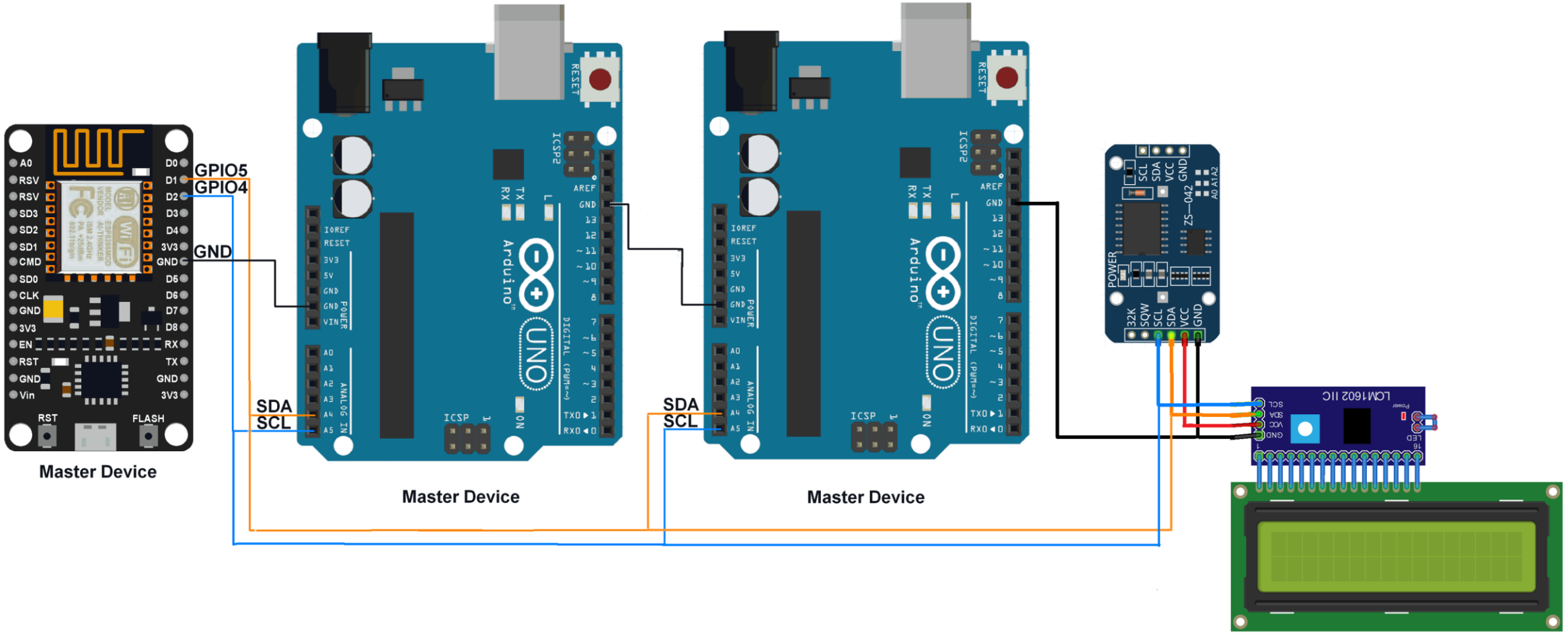


I2C Communication

- I2C is short for Inter-Integrated Circuit, a synchronous serial communication protocol developed by Phillips for communication between a fast Microcontroller and relatively slow peripherals (like Memory or Sensors) using just two wires
- Using I2C, you can transmit data at rates 100 kbit/s (clock 100 kHz – Standard Mode) to 3.4 Mbit/s (clock 3.4 MHz – High Speed Mode).



I2C Communication (2/2)



Comparison Between UART, SPI, I2C

Protocol	UART	SPI	I2C
Speed	Slowest	Fastest	Faster than UART
Number of devices	Up to 2	4 devices	Up to 128 devices
Wires required	2(Tx,Rx)	4(SCK,MOSI,MISO,SS)	2(SDA,SCL)
Duplex Mode	Full Duplex Mode	Full Duplex Mode	Half Duplex
Number of Master-Slaves possible	Single Master-Single Slave	Single Master-Multiple Slaves	Multiple Masters-Multiple Slaves
Complexity	Simple	Can easily control multiple devices	Complex with increase in devices
Acknowledgment bit	No	No	Yes

Wireless communication with Arduino

Wireless interface (NRF24L01)

Wireless interface (NRF24L01) (1/2)

- The nRF24L01 is a single chip 2.4GHz transceiver with an embedded baseband protocol engine (Enhanced ShockBurst), designed for ultra low power wireless applications:
  - works on SPI Protocol;
  - used for sending and receiving data at an operating radio frequency of 2.4 to
  - 2.5 GHz ISM band

1	GND
2	VCC
3	CE
4	CSN
5	SCK
6	MOSI
7	MISO
8	IRQ

Figure: Interface of NRF24L01.

Wireless interface (NRF24L01) (1/2)

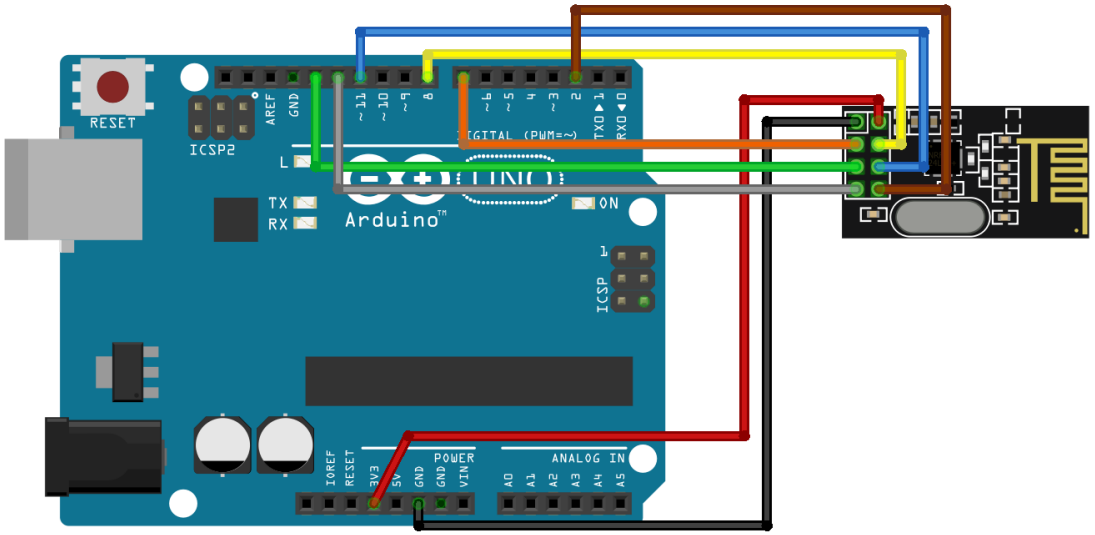
- The NRF24L01 module is powered by a voltage between 1.9V and 3.6V, while the other pins can be controlled with 5V

GND	The ground pin.
VCC	The power supply.
CE	The **Chip Enable**. Determines whether the module should be placed in receiving or transmitting state.
CSN	SPI **Chip Select**.
SCK	SPI **Clock**. A clock signal used to synchronize the data transfer through the serial bus.
MOSI	SPI **Master Out Slave In**. A line leaving the master and entering the slaves.
MISO	SPI **Master In Slave Out**. An input line leaving the slave and entering the master.
IRQ	Interrupt.

Serial Peripheral Interface (SPI) (1/3)

- The Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by micro-controllers for communicating with one or more peripheral devices quickly over short distances
  - It is used for sending and receiving data at an operating radio frequency of
  - 2.4 to 2.5 GHz ISM band;
  - Also used for communication between two micro-controllers
  - SPI connections always have one master device (usually a micro-controller) which controls the peripheral devices

Serial Peripheral Interface (SPI) (2/3)



The GND and VCC pin of the module should be connected respectively to the GND and 3.3V pins of the Arduino.

Be Careful

Do not connect the module to the 5V voltage supply, it is too high.

Serial Peripheral Interface (SPI) (3/3)

- The CE pin and the CSN pin can be connected to any digital pin as they are configured via software, in this case the digital pins 7 and 8.

- The SCK, MOSI and MISO pins are default as they are used for **SPI Communication**: pin 11 (MOSI), pin 12 (MISO), pin 13 (SCK).

- These support pins are used by the SPI library, used by the RF module.

- IRQ is not used in this scenario.

NRF24L01

- **Instantiate** the Radio Frequency (RF) module manager:
  - **RF24(uint8_t cePin, uint8_t csPin)**
    - cePin: Pin connected to the RF Module Chip Enable;
    - csPin: Pin connected to Chip Select;

- **Initialize** the module before calling any other commands:
  - **void RF24::begin(void);**

- Configure the pipe in **reading/writing mode:**
  - **void RF24::openWritingPipe(uint64_t address);**
  - **void RF24::openReadingPipe(uint8_t number, uint64_t address);**
    - number: the number of the pipe ranges from 0 to 5 (0 usually used ad writing pipe)
    - address: the 40 bit address of the pipe to open

NRF24L01

- **Start/Stop listening**:
  - **void RF24::startListening(void)**
    - Start listening to open pipes in read mode. Make sure that the openReadingPipe() function is called. You cannot call the write function if you have not called the stopListening() function before.
  - **void RF24::stoptListening(void)**
    - Stops listenings for incoming messages
- **Write Data:**
  - **bool RF24::write(const void* buf, uint_8 len)**
    - Writes on the open writing pipe. First, make sure that the openWritingPipe() function has been called. Return True if the data has been sent, False otherwise.
    - buf: pointer to the data to send
    - len: number of bytes to send

NRF24L01

- **Check for incoming messages**:
  - **bool RF24::available(void)**
    - Checks if there are bytes available to read. It returns True if there is an incoming stream of data, False otherwise;
  - **bool RF24::read(void* buf, uint8_t len)**
    - Reads incoming data and returns the last received data. It returns True if the data has been sent, False otherwise;
    - buf: pointer of the buffer in which data will be placed;
    - len: maximum bytes to read from the buffer;

- **Change the communication channel**
  - **void RF24::setChannel(uint8_t channel)**
    - Sets the communication channel;
    - channel: Which RF channel is used to communicate. It allows to select a value from 0 to 127;

NRF24L01

- **Change the payload size**:
  - **bool RF24::setPayloadSize(uint8_t len)**
    - If this method is not called the device transmits to the maximum payload size, or 32 btyes;

- **Change the power amplifier level**
  - **void RF24::setPALevel(rf24_pa_dbm_e level)**
    - Sets the power amplifier in one of four levels:
      - RF24_PA_MIN = -18dBm
      - RF24_PA_LOW = -12dBm
      - RF24_PA_MED = -6dBm
      - RF24_PA_HIGH = 0dBm

NRF24L01

- **Change the transmission speed**
  - **void RF24::setChannel(rf24_datae_e speed)**
    - Sets the transmission speed at one of the three speeds:
      - RF24_250KBPS = 250Kbs
      - RF24_1MBPS = 1Mbs
      - RF24_2MBPS = 2Mbs

- In order to work, all the communications must have the same:
  - Transmission speed;
  - Channel;
  - Payload Size;

Wireless communication with Arduino

Scenario 1

Introduction

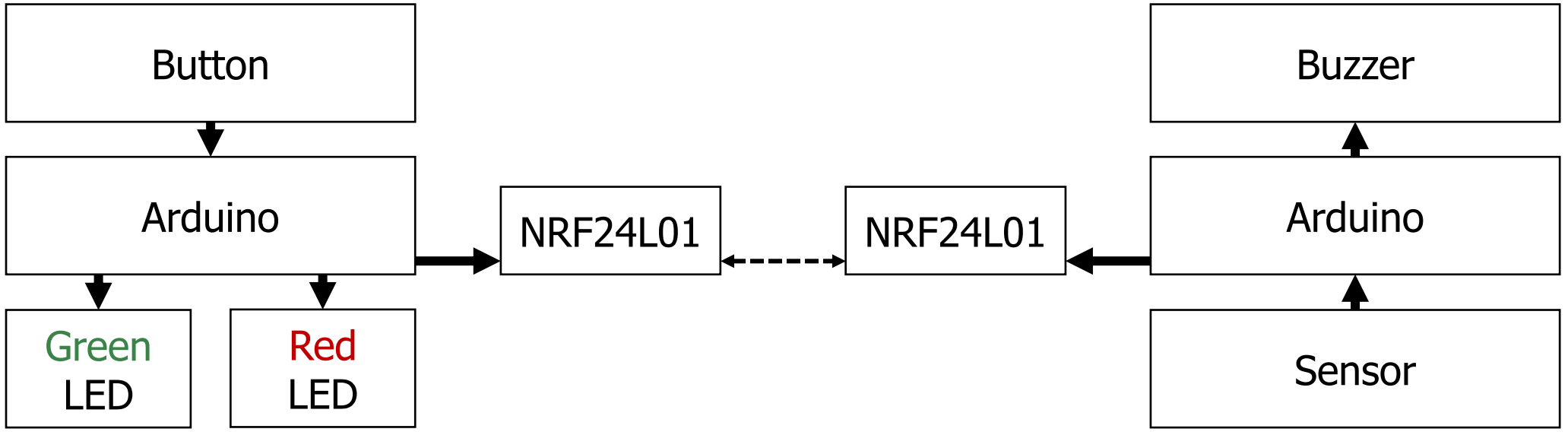
- This scenario concerns the use of two Arduino boards to transmit packets through the use of a Radio-Frequency (RF) board (NRF24L01)
- The project is a temperature measurement system which notify with an acoustic alarm if the temperature levels cross an upper-bound
- Two LEDs (i.e., Red and Green) shows if the temperature is getting near the upper-bound:

GREEN LED	the temperature is low.
RED LED	the temperature is rising.
SOUND	the temperature has crossed the upper-bound.

Required Material

- 2 x Arduino Uno
- 2 x Prototyping board
- 2 x NRF24L01 Module
- 1 x Red LED & 1 x Green LED
- 1 x Switch
- 1 x Buzzer
- 1 x Temperature Sensor (DALLAS1820)
- 3 x Resistor 390 Ω & 1 x Resistor 4.7 kΩ

Structure


Button
Arduino
Green LED
Red LED
NRF24L01
NRF24L01
Buzzer
Arduino
Sensor

Functionality

- The structure comprises two setups:
  - Master: Arduino, NRF24L01, two LEDs, and button;
  - Slave: Arduino, NRF24L01, DALLAS18B20, temperature sensor, and buzzer;
- The node to which the button is connected is considered as a master
- Whenever the user presses the master's button, a request is sent to the slave
- The request is the temperature measured at the place where the slave is deployed

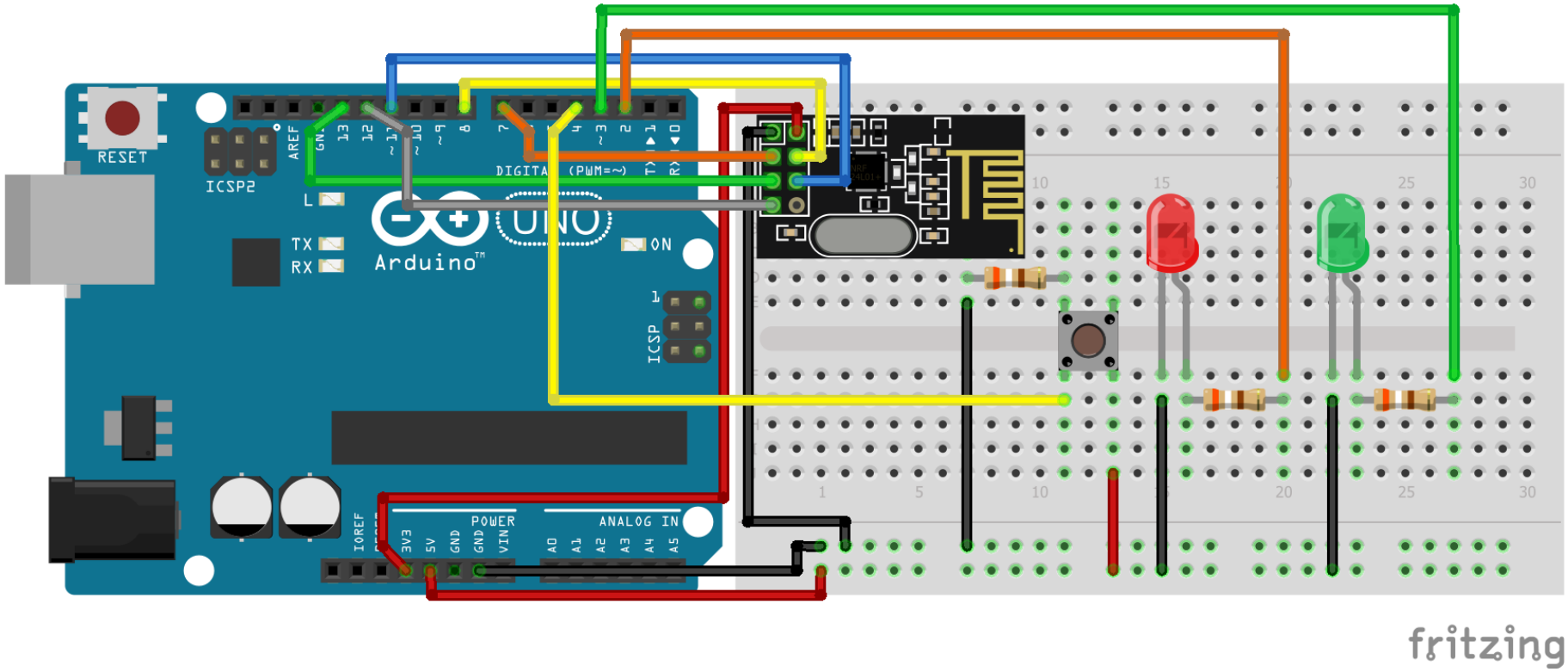
Functionality

- Once the temperature is received by the master, it is displayed on the serial monitor of the Arduino
- The time elapsed from the request to the response is shown along with the temperature in milliseconds
- However, if the time elapsed exceeds a threshold (i.e., it takes too long to be received), the value is discarded since it cannot be considered as valid and an error message is shown.

Wireless communication with Arduino

Scenario 1

Master (1/3)


RESET
ICSP2
Arduino
UNO
TX
RX
ON
ICSP
POWER
ANALOG IN
fritzing

Master (2/3)

- Whenever the button is pressed, the master node sends a request to the slave node for the current temperature value.
- Based on the temperature value supplied by the slave, the LEDs turn on for about 2 seconds.
- On each transmission, along with the request/response, the time at which the packet is send is provided.
- Thus, a structure with both time and temperature values is created and used during the communication.

Master (3/3)

- The current time value is acquired by means of the micros() function, which returns the number of microseconds since the Arduino started running the program.

- Once the slave has received the request and the time-stamp, it updates the acquired temperature value inside the packet and then send it back to the master.

- The master receives the packet and evaluates the elapsed time by using the stored value inside the packet and the aforementioned function.

- If the elapsed time is not above a pre-determined threshold, the packet is displayed on the serial monitor.

Slave

- The slave node waits the requests from the master. However, if the temperature is too high, a buzzer is activated every 2 seconds, until the temperature returns to normal.