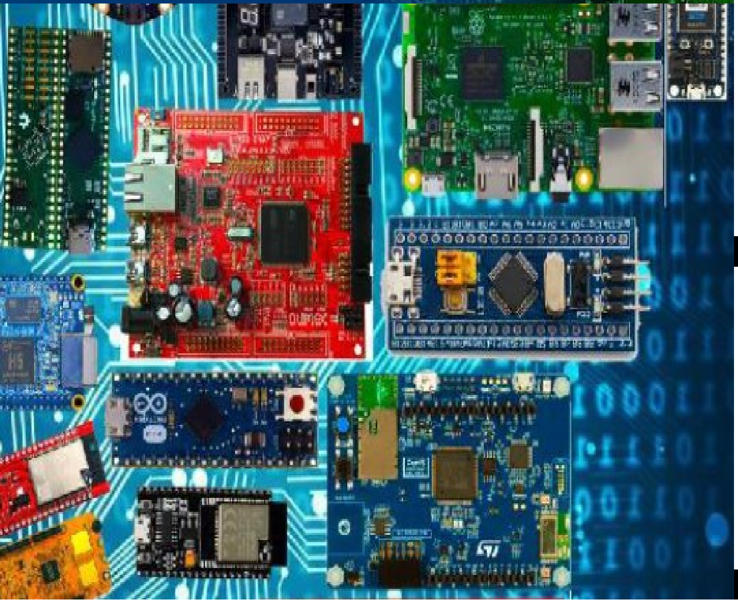


Embedded Systems

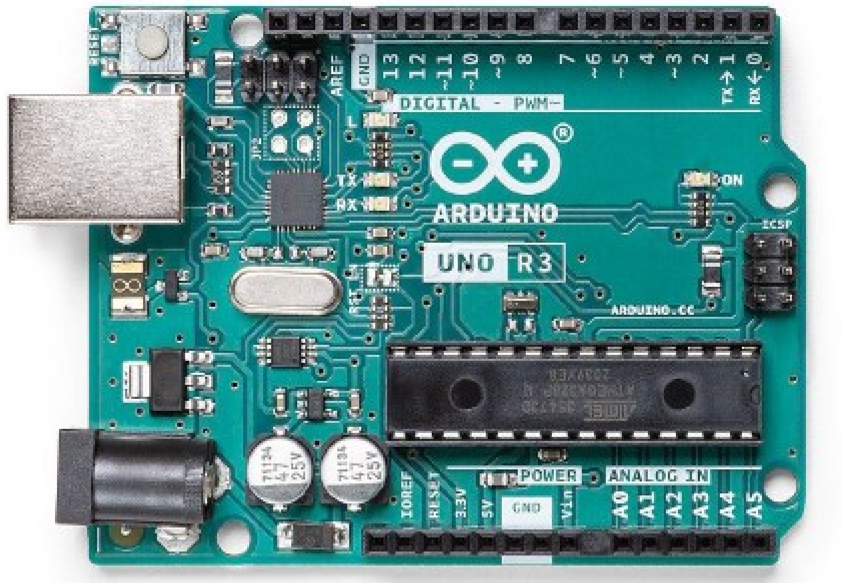
PROGRAMMING ARDUINO AND NANO 33 BLE

Dennis A. N. Gookyi

CONTENTS

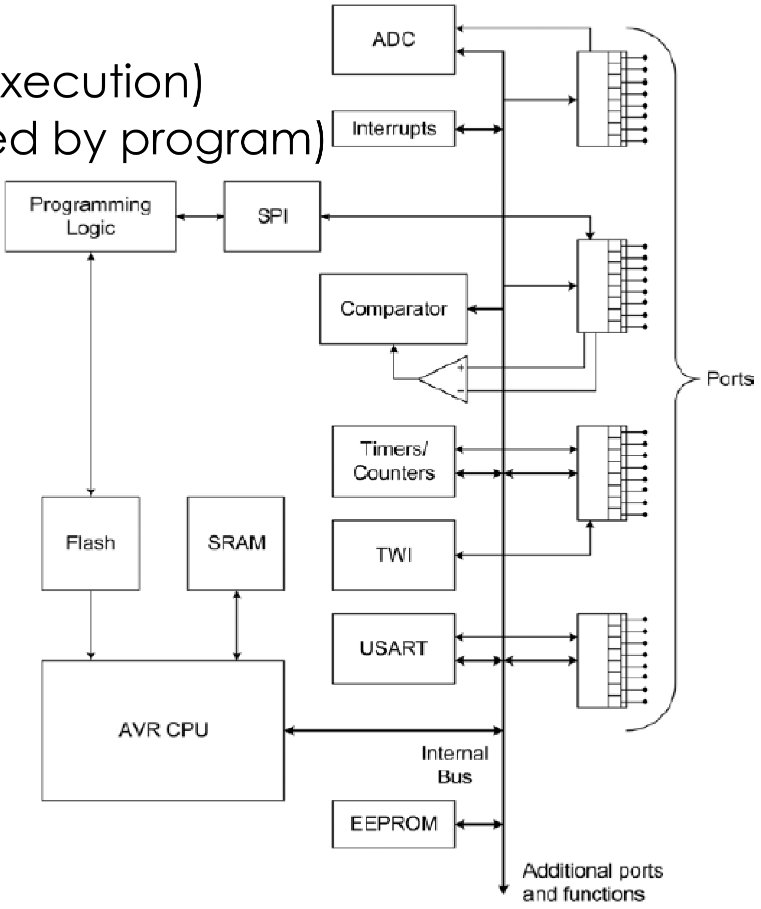
ARDUINO

- Arduino's purpose is to control things by interfacing with sensors and actuators
  - No keyboard, mouse and screen
  - Can be attached via "shields"
  - No operating system, limited memory
  - A single program enjoys 100% of CPU time
- Physical Arduino boards
  - Uno
  - Nano
  - Leonardo
  - Mega
  - Pro Mini
- Arduino IDE
  - Installed on a PC (Windows/Mac/Linux)
  - To develop, install and debug programs on Arduino boards
  - Communicates with Arduino board over USB
- Third party Arduino compatible boards
  - STM32
  - Nucleo / Discovery / Feather,
  - Adafruit
  - SparkFun



ARDUINO

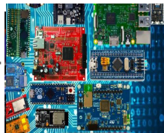
- A generic AVR microcontroller block diagram
  - CPU
  - Internal Memory
    - Flash (stores program code)
    - SRAM (holds data and variable during execution)
    - EEPROM (holds persistent data generated by program)
  - Peripherals
    - A/D converter (ADC)
    - Timers
    - UART
    - SPI
    - DMA
    - GPIO
    - TWI
    - Comparator
    - RTC
    - WDT
    - RNG



ARDUINO

- Interfacing with Arduino
  - Temperature sensor
  - Pressure sensor
  - Switches
  - Variable resistor
  - Range finder
  - PIR (person
  - in room) sensor
  - Relay
  - Motor control
  - LED
  - 16x2 display
  - Graphic display
  - Bluetooth shield
  - WiFi
  - shield
  - Ethernet shield

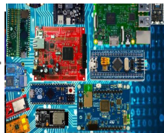
Arduino Libraries:
https://www.arduinolibraries.info/libraries



ARDUINO

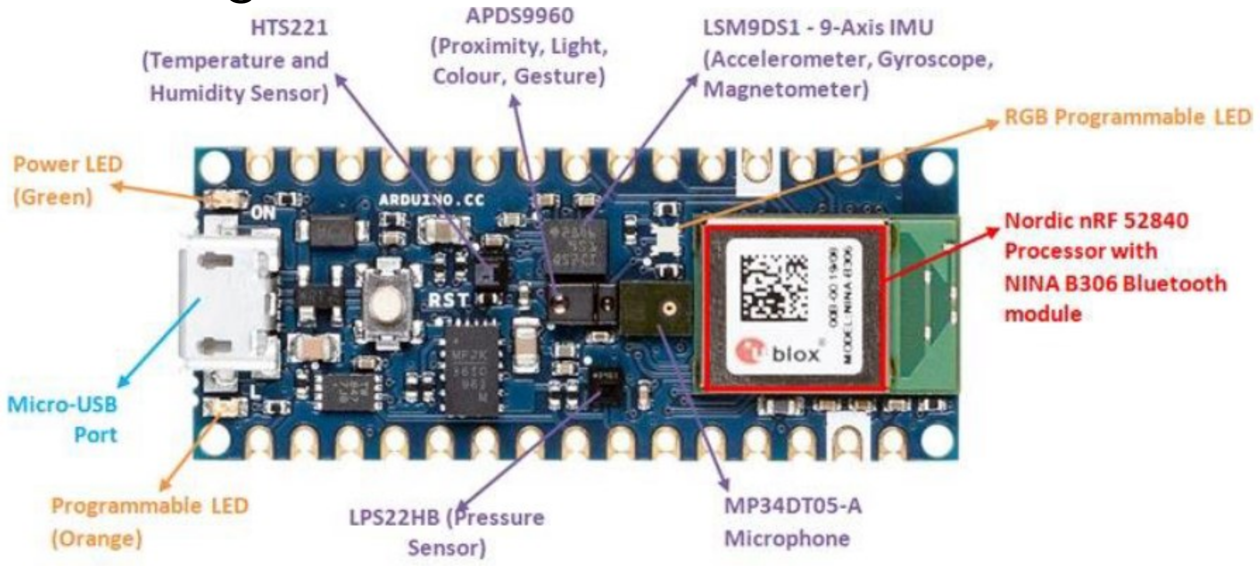
- Uno vs Nano 33 BLE Sense

	Uno R3	Nano 33 BLE Sense
Chip	ATmega328P	nRF52840
Clock	16 MHz	64 MHz
Flash	32 KB	1 MB
SRAM	2 KB	256 KB
EEPROM	1 KB	none
Input Voltage	6 - 20 V	4.5 - 21 V
I/O Voltage	5 V	3.3 V
Pinout	14 digital, 6 PWM, 6 AnalogIn	14 digital (PWM), 8 AnalogIn
Interfaces	USB, SPI, I2C, UART	USB, SPI, I2C, I2S, UART
Connectivity	via shields	BLE 5.0
Weight	25 g	5 g



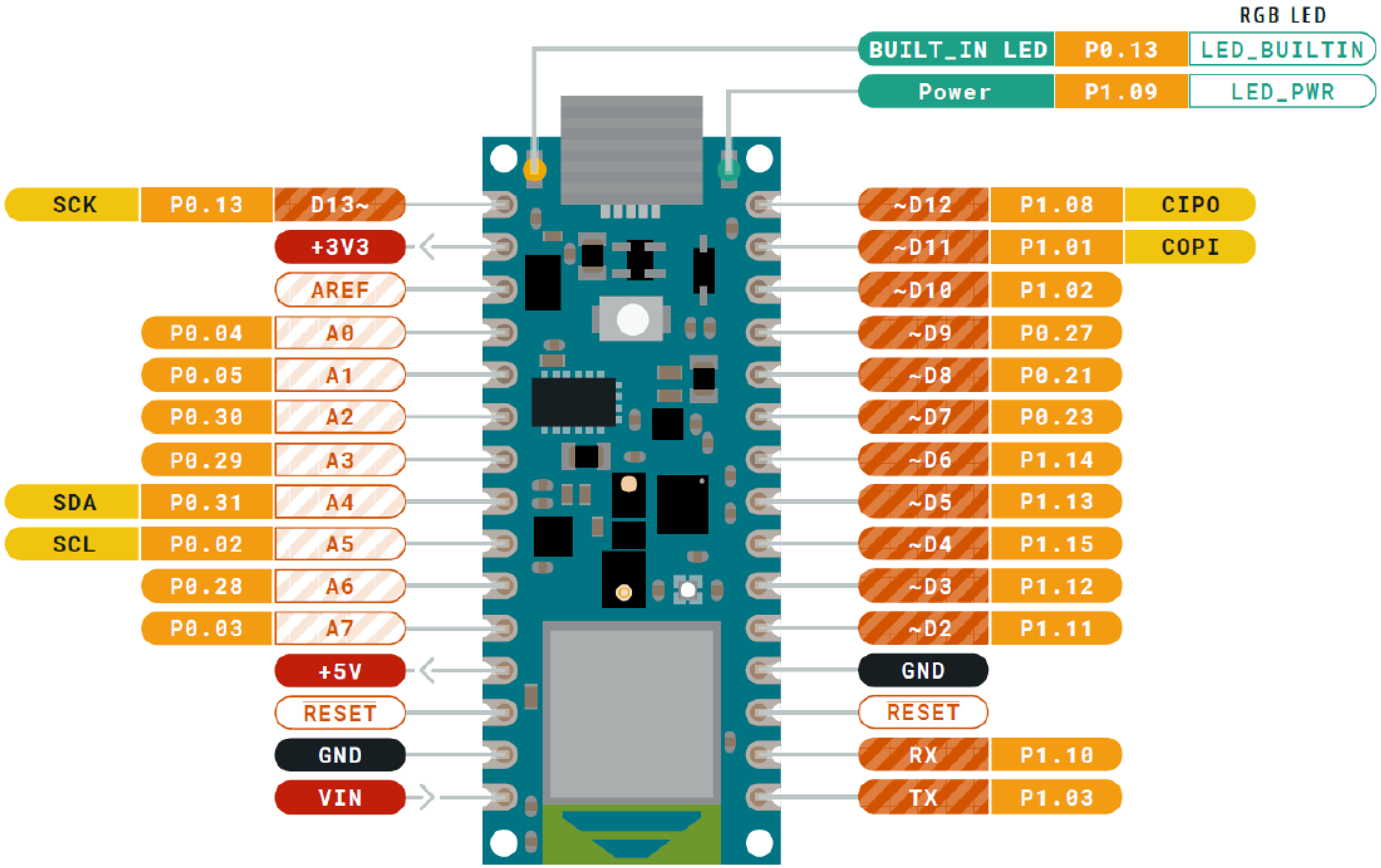
ARDUINO

- Features of Nano 33 Sense
  - 8 Analog Input Pins can provide 12-bit ADC at about 30 kHz
  - Integrated sensors (IMU, Mic, Light, Pressure, Temperature, Humidity)
  - All digital pins can trigger interrupts
  - Only supports 3.3V I/Os and is NOT 5V tolerant so please make sure you are not directly connecting 5V signals to this board or it will be damaged


HTS221 (Temperature and Humidity Sensor)
APDS9960 (Proximity, Light, Colour, Gesture)
LSM9DS1 - 9-Axis IMU (Accelerometer, Gyroscope, Magnetometer)
RGB Programmable LED
Power LED (Green)
Nordic nRF 52840 Processor with NINA B306 Bluetooth module
Micro-USB Port
Programmable LED (Orange)
LPS22HB (Pressure Sensor)
MP34DT05-A Microphone

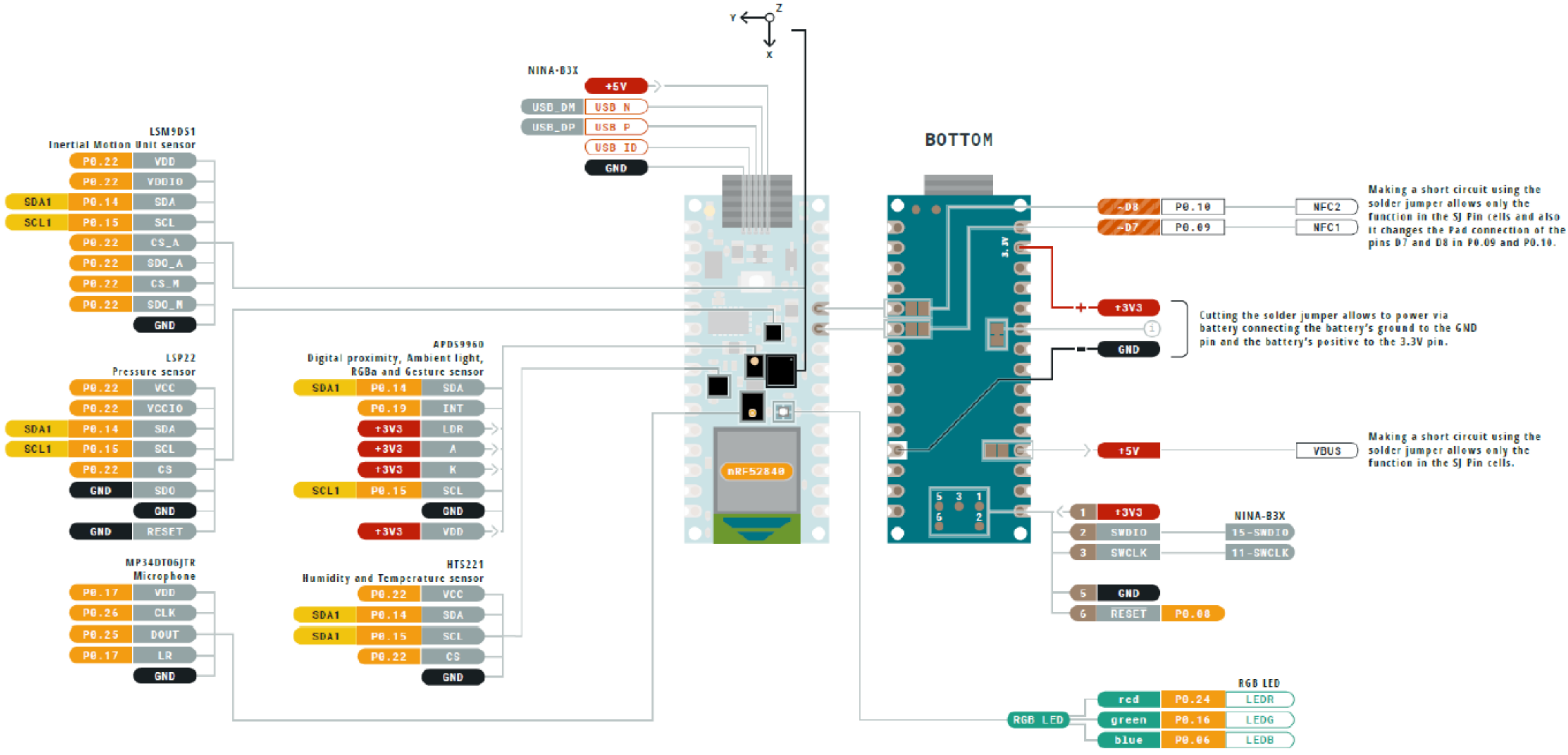
NANO 33 SENSE

- Nano 33 Sense



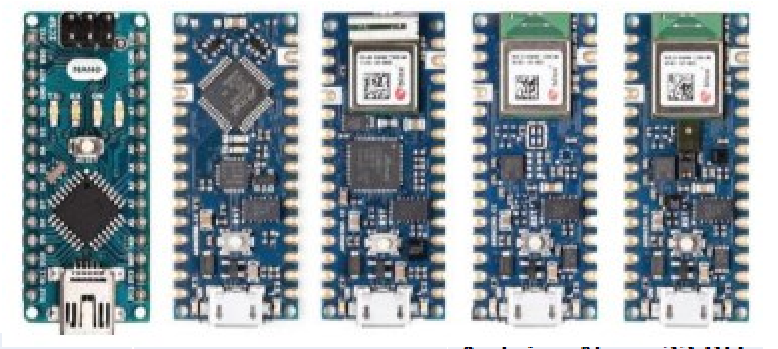
NANO 33 SENSE

- Nano 33 Sense



NANO 33 SENSE

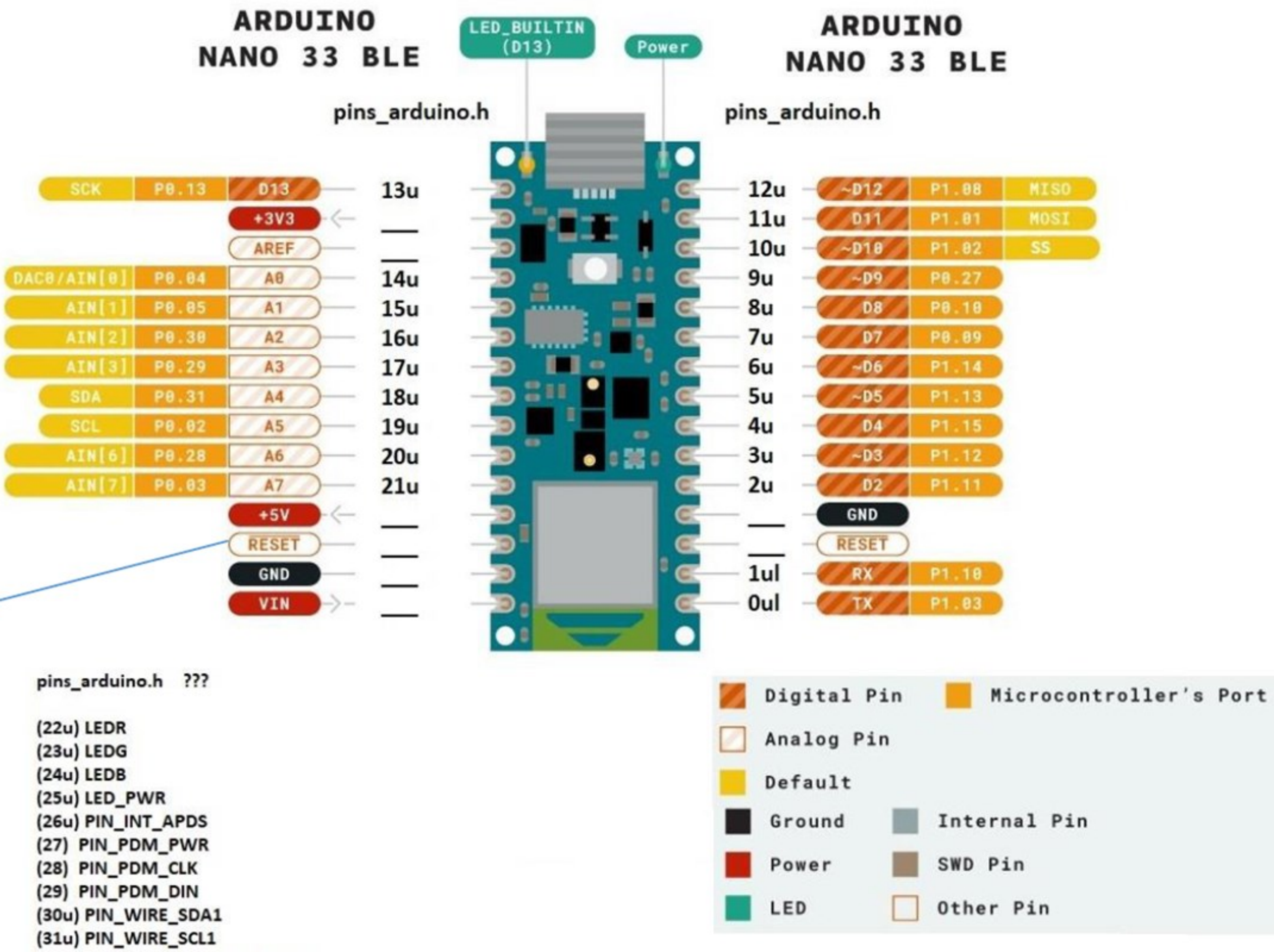
- Arduino Nano comparisons



Property	Arduino Nano	Arduino Nano Every	Arduino Nano 33 IoT	Arduino Nano 33 BLE	Arduino Nano 33 BLE Sense
Microcontroller	ATmega328	ATMega4809	SAMD21 Cortex®-M0+ 32bit low power ARM MCU	nRF52840 (ARM Cortex M4)	nRF52840 (ARM Cortex M4)
Operating voltage	5 V	5 V	3.3 V	3.3 V	3.3 V
Input voltage (VIN)	6-20 V	7-21 V	5-21 V	5-21 V	5-21 V
Clock speed	16 Mhz	20 MHz	48 MHz	64 MHz	64 MHz
Flash	32 KB	48 KB	256 KB	1 MB	1 MB
RAM	2 KB	6 KB	32 KB	256 KB	256 KB
Current per pin	40 mA	40 mA	7 mA	15 mA	15 mA
PWM pins	6	5	11	All	All
IMU	No	No	LSM6DS3 (6-axis)	LSM9DS1 (9-axis)	LSM9DS1 (9-axis)
Other sensors	No	No	No	No	Several
WiFi	No	No	Yes	No	No
Bluetooth	No	No	Yes	Yes	Yes
USB type	Mini	Micro	Micro	Micro	Micro

NANO 33 SENSE

- Pinout



Pins A4 and A5 have an internal pull up and default to be used as an I2C Bus so usage as analog inputs is not recommended.

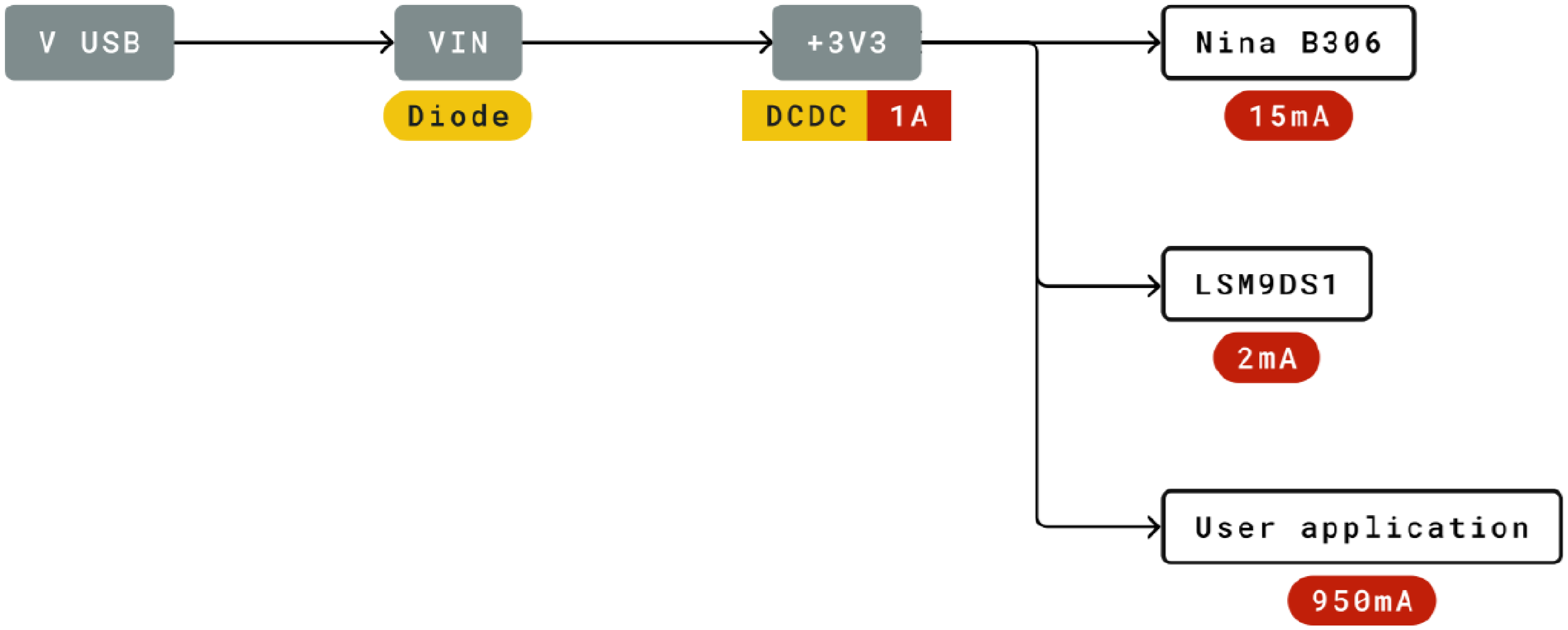
Ground the RESET pin to reset

pins_arduino.h ???

(22u) LEDR
(23u) LEDG
(24u) LEDB
(25u) LED_PWR
(26u) PIN_INT_APDS
(27) PIN_PDM_PWR
(28) PIN_PDM_CLK
(29) PIN_PDM_DIN
(30u) PIN_WIRE_SDA1
(31u) PIN_WIRE_SCL1
(32u) PIN_ENABLE_SENSORS_3V3
(33u) PIN_ENABLE_I2C_PULLUP

NANO 33 SENSE

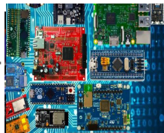
- Nano 33 BLE Power Tree



All Arduino boards have a built-in bootloader which allows flashing the board via USB. In case a sketch locks up the processor and the board is not reachable anymore via USB it is possible to enter bootloader mode by double-tapping the reset button right after power up.

NANO 33 SENSE

- nRF52840
  - Arduino Nano BLE (and BLE Sense) is based on the nRF52840 microprocessor made by Nordic
  - nRF52840 has the ARM Cortex M4 processor with single precision floating point unit (FPU)
  - The nRF52840 contains 1 MB of flash and 256 kB of RAM that can be used for code and data storage
  - The flash can be read an unlimited number of times by the CPU, but it has restrictions on the number of times it can be written and erased (minimum 10,000 times) and also on how it can be written
  - The flash is divided into 256 pages of 4 kB each that can be accessed by the CPU via both the ICODE and DCODE buses

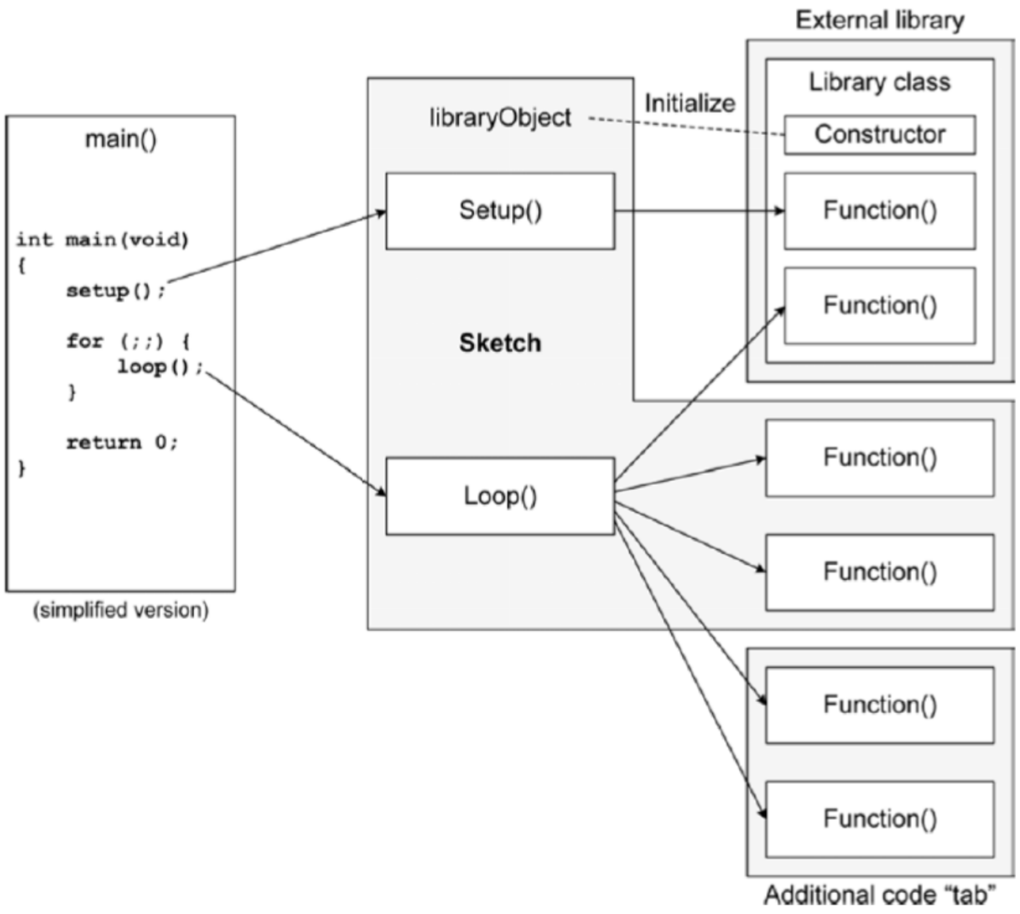


NANO 33 SENSE

❖ Arduino Sketch Structure

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

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

}
```

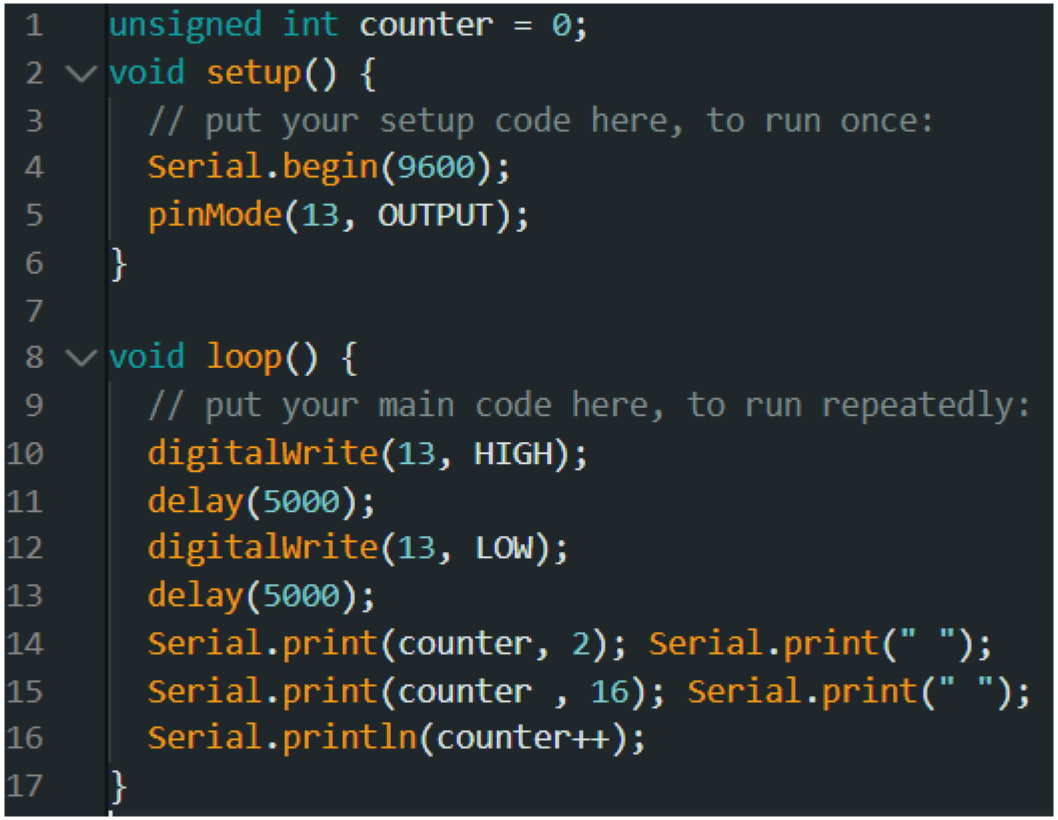
```
int main(void)
{
    init();
    initVariant();

    #ifdefined(USBCON)
    USBDevice.attach();
    #endif

    setup();
    for (;;) {
        loop();
        if (serialEventRun) serialEventRun();
    }
    return 0;
}
```



```
int main(void)
{
     setup();

     for (;;) {
          loop();
     }

     return 0;
}
```

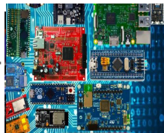
NANO 33 SENSE

❖ Arduino Sketch Example



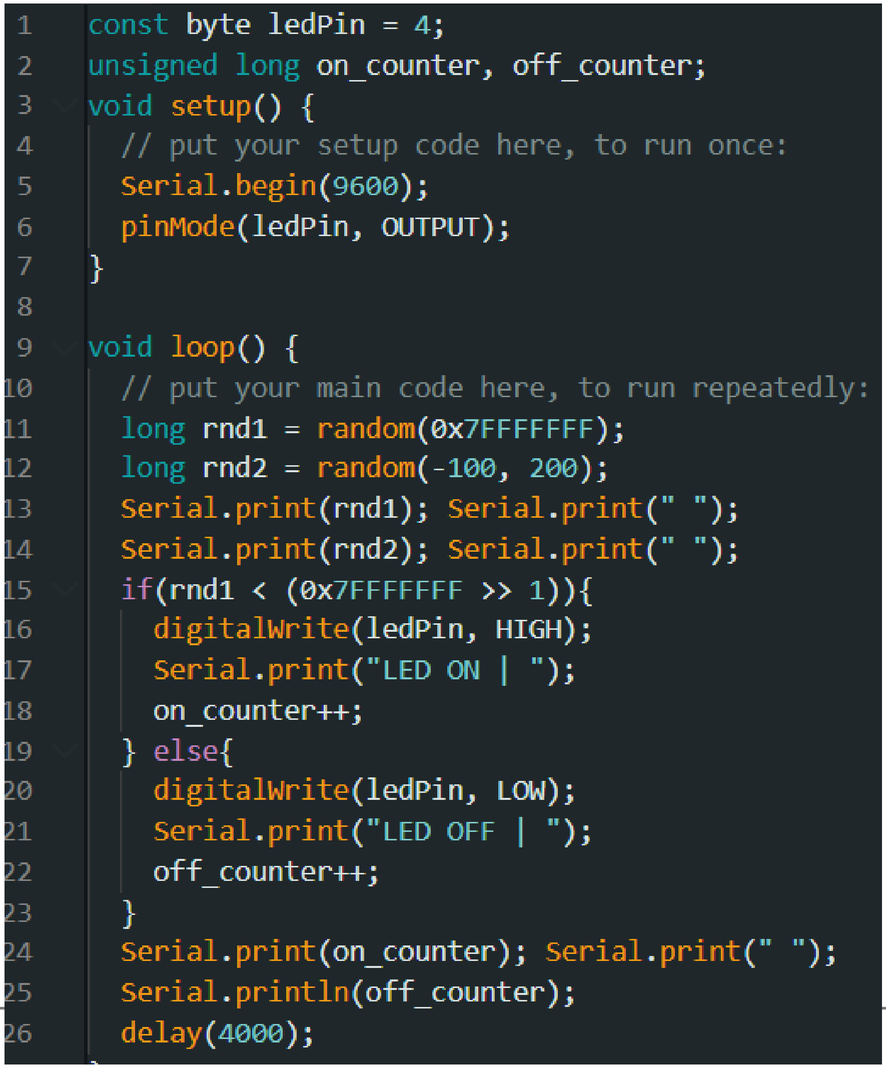
```
unsigned int counter = 0;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(13, HIGH);
  delay(5000);
  digitalWrite(13, LOW);
  delay(5000);
  Serial.print(counter, 2); Serial.print(" ");
  Serial.print(counter , 16); Serial.print(" ");
  Serial.println(counter++);
}
```

0x7FFFFFFF
= 0b01111111111111111111111111111111
= 2,147,483,647
Maximum value of long in 32-bit system.

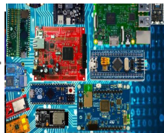


NANO 33 SENSE

- Arduino Sketch Example

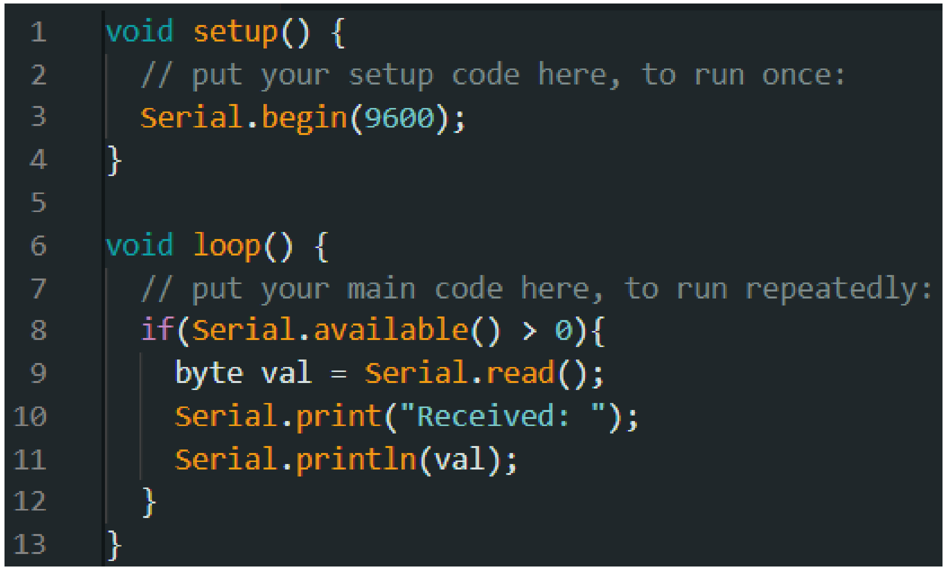


```
const byte ledPin = 4;
unsigned long on_counter, off_counter;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  long rnd1 = random(0x7FFFFFFF);
  long rnd2 = random(-100, 200);
  Serial.print(rnd1); Serial.print(" ");
  Serial.print(rnd2); Serial.print(" ");
  if(rnd1 < (0x7FFFFFFF >> 1)){
    digitalWrite(ledPin, HIGH);
    Serial.print("LED ON | ");
    on_counter++;
  } else{
    digitalWrite(ledPin, LOW);
    Serial.print("LED OFF | ");
    off_counter++;
  }
  Serial.print(on_counter); Serial.print(" ");
  Serial.println(off_counter);
  delay(4000);
```

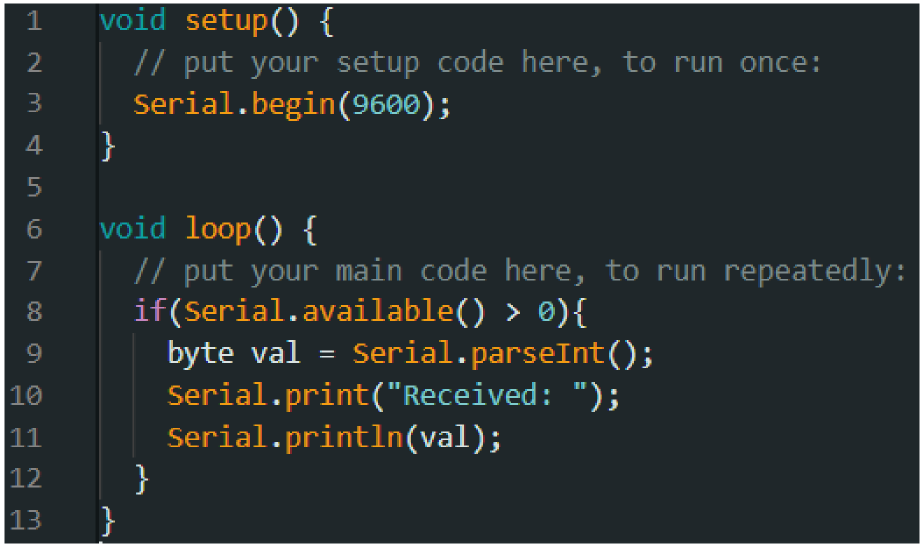


NANO 33 SENSE

❖ Read form serial



```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  if(Serial.available() > 0){
    byte val = Serial.read();
    Serial.print("Received: ");
    Serial.println(val);
  }
}
```



```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  if(Serial.available() > 0){
    byte val = Serial.parseInt();
    Serial.print("Received: ");
    Serial.println(val);
  }
}
```

https://www.arduino.cc/reference/en/

NANO 33 SENSE

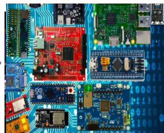
❖ Length of data type may be different on different CPUs

Arduino Data Types (on Arduino Uno)

- 1 byte:
  - boolean (true or false, 0 or 1)
  - char (-128 to +127)
  - byte (0 to 255)
- 2 bytes:
  - int (-32768 to +32768)
  - unsigned int (0 to 65536)
- 4 bytes:
  - long (around -2.1 billion to +2.1 billion, integer)
  - unsigned long (0 to ~4.2 billion, integer)
  - float (-3.4E+38 to 3.4E+38)
  - double (the same as float)

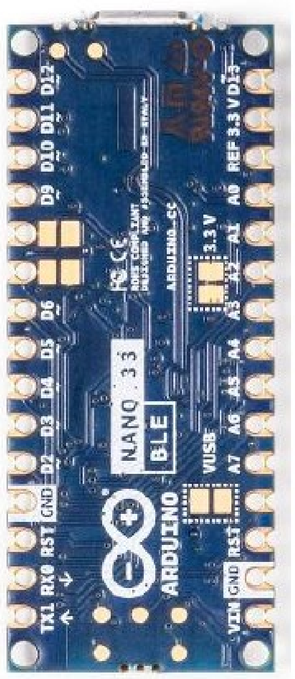
On Arduino Nano 33 BLE (Sense)

```
byte is 1 byte(s)
short is 2 byte(s)
int is 4 byte(s)
unsigned int is 4 byte(s)
long is 4 byte(s)
unsigned long is 4 byte(s)
long long is 8 byte(s)
unsigned long long is 8 byte(s)
float is 4 byte(s)
double is 8 byte(s)
uint8_t is 1 byte(s)
uint16_t is 2 byte(s)
uint32_t is 4 byte(s)
```



NANO 33 SENSE

- Arduino Pin Names
  - Pin name aliases are defined in the pins_arduino.h
  - As a convention
  - Digital pins can be referred by the number (e.g., 3) or D+number (e.g. D3)
  - Analog pins must be referred by A+number , e.g., A3

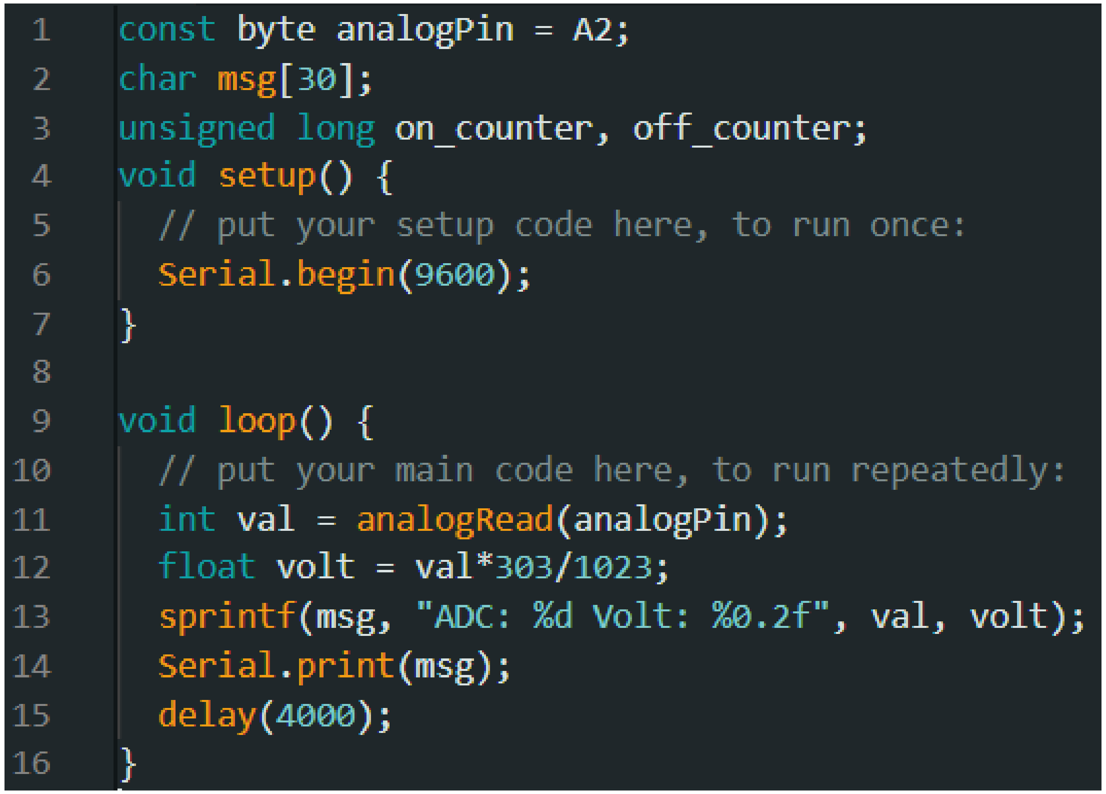


```
// LEDs
// ----
#define PIN_LED     (13u)
#define LED_BUILTIN PIN_LED
#define LEDR        (22u)
#define LEDG        (23u)
#define LEDB        (24u)
#define LED_PWR     (25u)

// Analog pins
// -----------
#define PIN_A0 (14u)
#define PIN_A1 (15u)
#define PIN_A2 (16u)
#define PIN_A3 (17u)
#define PIN_A4 (18u)
#define PIN_A5 (19u)
#define PIN_A6 (20u)
#define PIN_A7 (21u)
static const uint8_t A0  = PIN_A0;
static const uint8_t A1  = PIN_A1;
static const uint8_t A2  = PIN_A2;
static const uint8_t A3  = PIN_A3;
static const uint8_t A4  = PIN_A4;
static const uint8_t A5  = PIN_A5;
static const uint8_t A6  = PIN_A6;
static const uint8_t A7  = PIN_A7;
#define ADC_RESOLUTION 12
```

```
// set pin D3 to ON
digitalWrite(3, HIGH);
digitalWrite(D3, HIGH);  // both works

// read voltage from pin A3, and convert to a
number between 0 and 4095 (12-bit ADC)
analogRead(A3);
```

NANO 33 SENSE

- Arduino Sketch Example

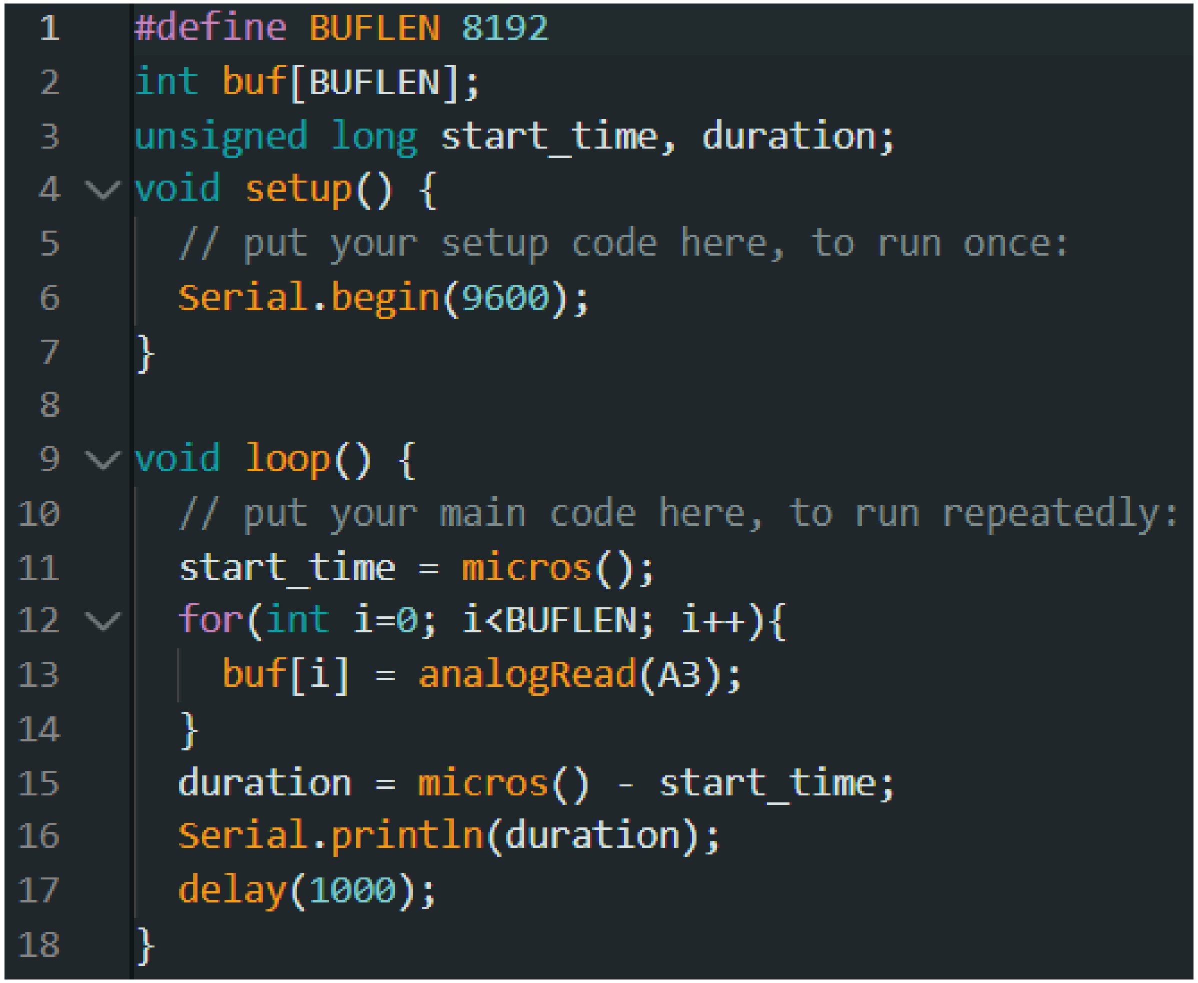


```
const byte analogPin = A2;
char msg[30];
unsigned long on_counter, off_counter;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  int val = analogRead(analogPin);
  float volt = val*303/1023;
  sprintf(msg, "ADC: %d Volt: %0.2f", val, volt);
  Serial.print(msg);
  delay(4000);
}
```

- To test, connect a jumper cable to A2, and try connecting the other end to 3.3 V or GND, and observe the output in the Serial Monitor

NANO 33 SENSE

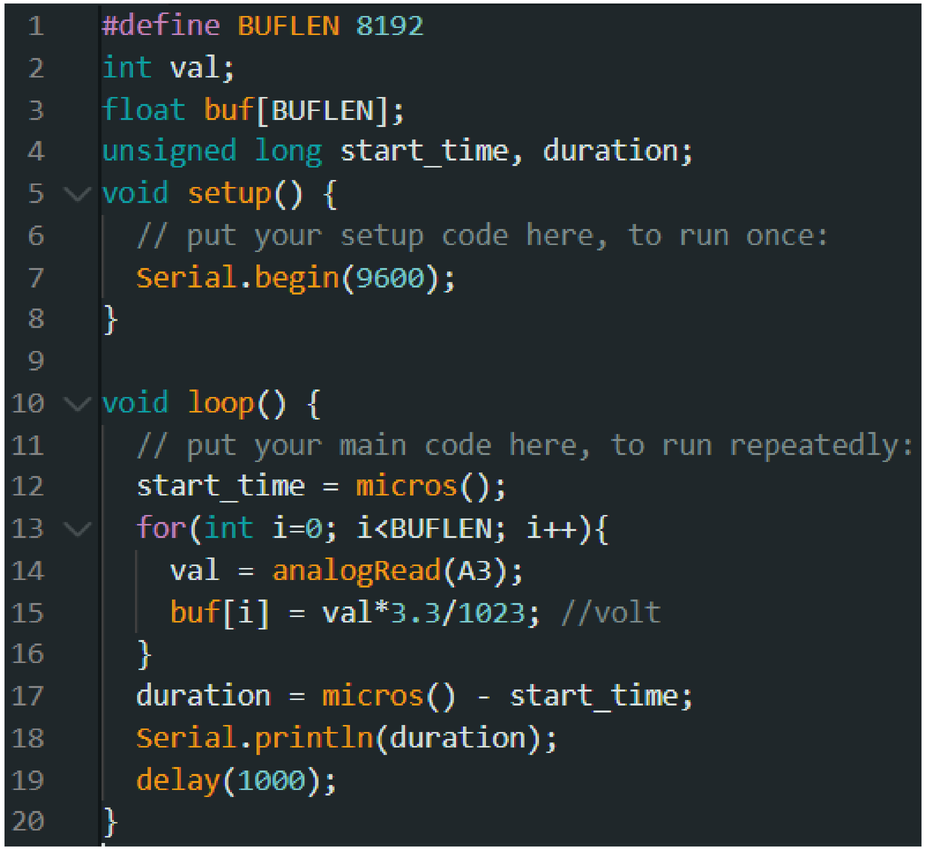
- Arduino Sketch Example



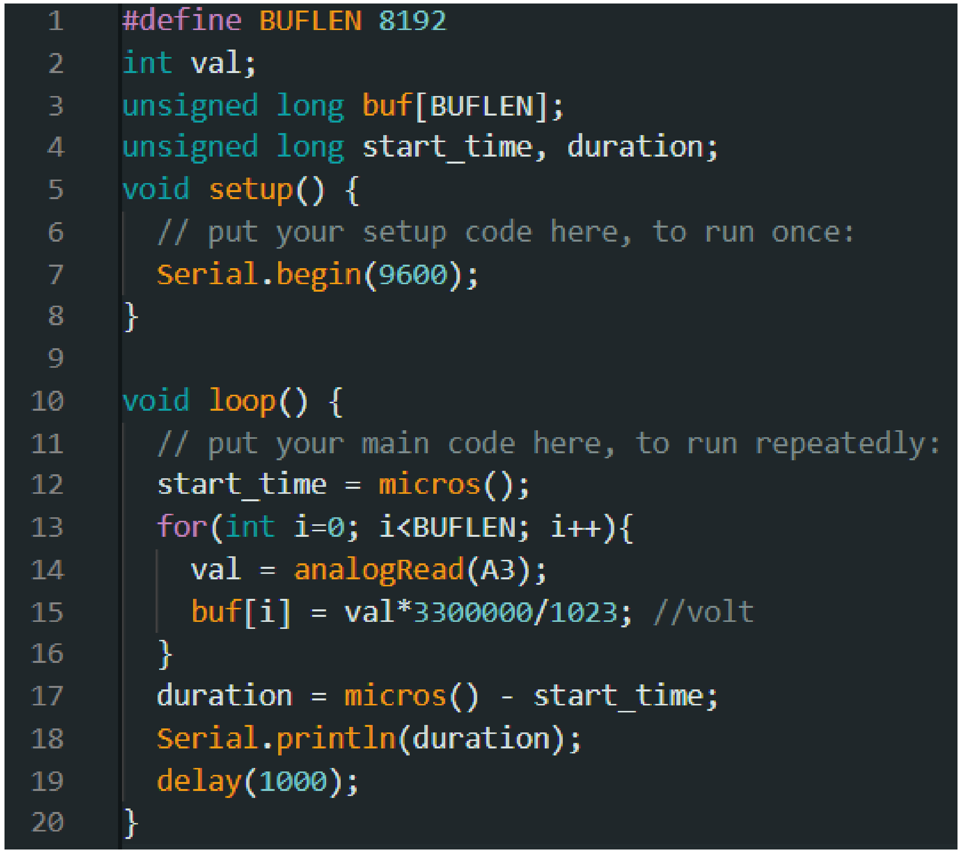
```
#define BUFLEN 8192
int buf[BUFLEN];
unsigned long start_time, duration;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  start_time = micros();
  for(int i=0; i<BUFLEN; i++){
    buf[i] = analogRead(A3);
  }
  duration = micros() - start_time;
  Serial.println(duration);
  delay(1000);
}
```

NANO 33 SENSE

- Integer vs Floating point arithmetic



```
#define BUFLEN 8192
int val;
float buf[BUFLEN];
unsigned long start_time, duration;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  start_time = micros();
  for(int i=0; i<BUFLEN; i++){
    val = analogRead(A3);
    buf[i] = val*3.3/1023; //volt
  }
  duration = micros() - start_time;
  Serial.println(duration);
  delay(1000);
}
```

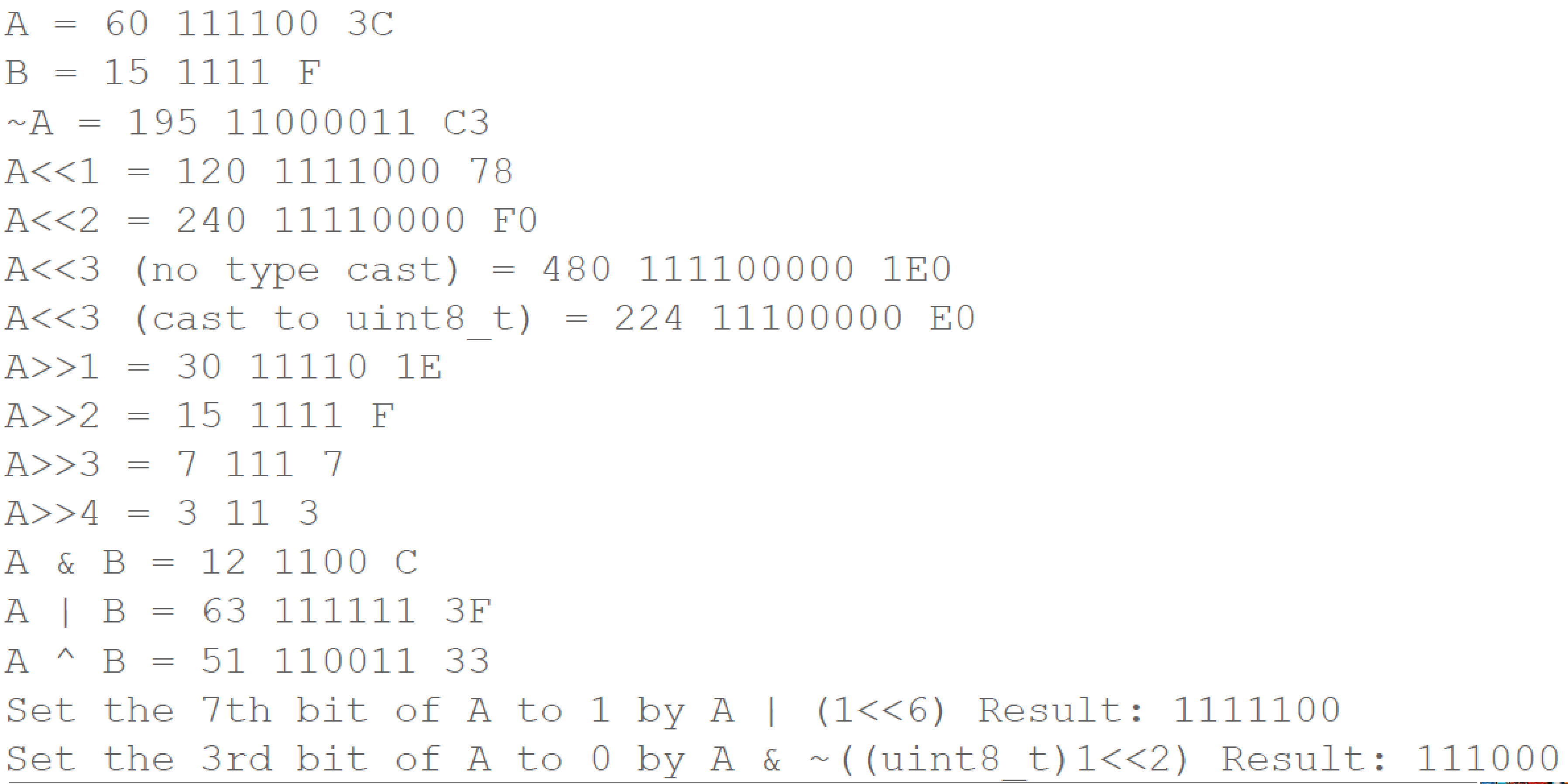


```
#define BUFLEN 8192
int val;
unsigned long buf[BUFLEN];
unsigned long start_time, duration;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  start_time = micros();
  for(int i=0; i<BUFLEN; i++){
    val = analogRead(A3);
    buf[i] = val*3300000/1023; //volt
  }
  duration = micros() - start_time;
  Serial.println(duration);
  delay(1000);
}
```

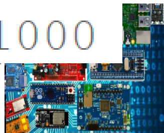
NANO 33 SENSE

- Bitwise operation

Decimal Binary Hexadecimal

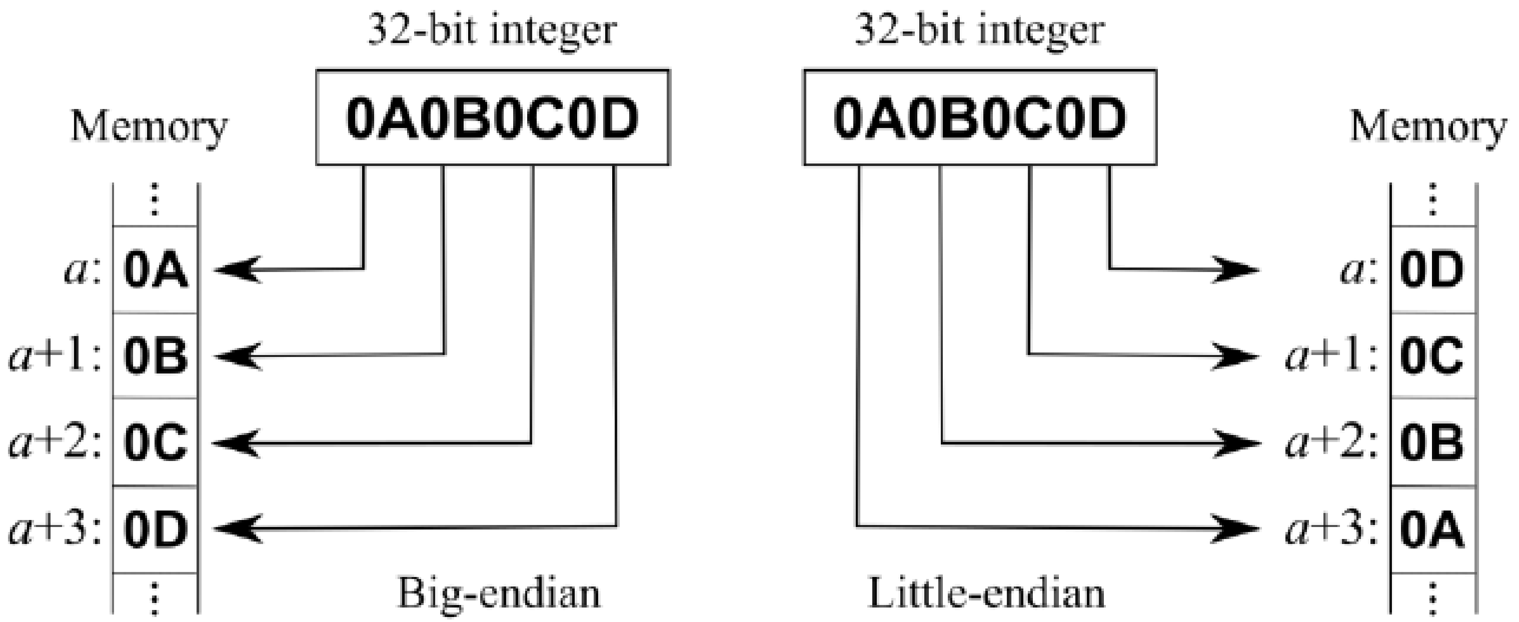


```
A = 60 111100 3C
B = 15 1111 F
~A = 195 11000011 C3
A<<1 = 120 1111000 78
A<<2 = 240 11110000 F0
A<<3 (no type cast) = 480 111100000 1E0
A<<3 (cast to uint8_t) = 224 11100000 E0
A>>1 = 30 11110 1E
A>>2 = 15 1111 F
A>>3 = 7 111 7
A>>4 = 3 11 3
A & B = 12 1100 C
A | B = 63 111111 3F
A ^ B = 51 110011 33
Set the 7th bit of A to 1 by A | (1<<6) Result: 1111100
Set the 3rd bit of A to 0 by A & ~((uint8_t)1<<2) Result: 111000
```



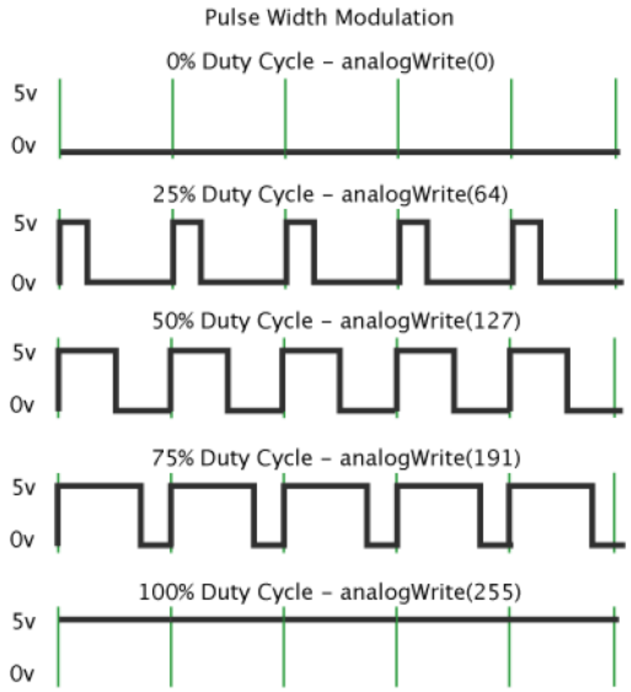
NANO 33 SENSE

- Endianness and byte addressing
  - Beware of the endianness when you need to manipulate or transmit data by bytes, or when memory contents are transmitted between computers with different endianness
  - Casting an uint32_t integer to uint8_t takes the low 8-bit of the integer



NANO 33 SENSE

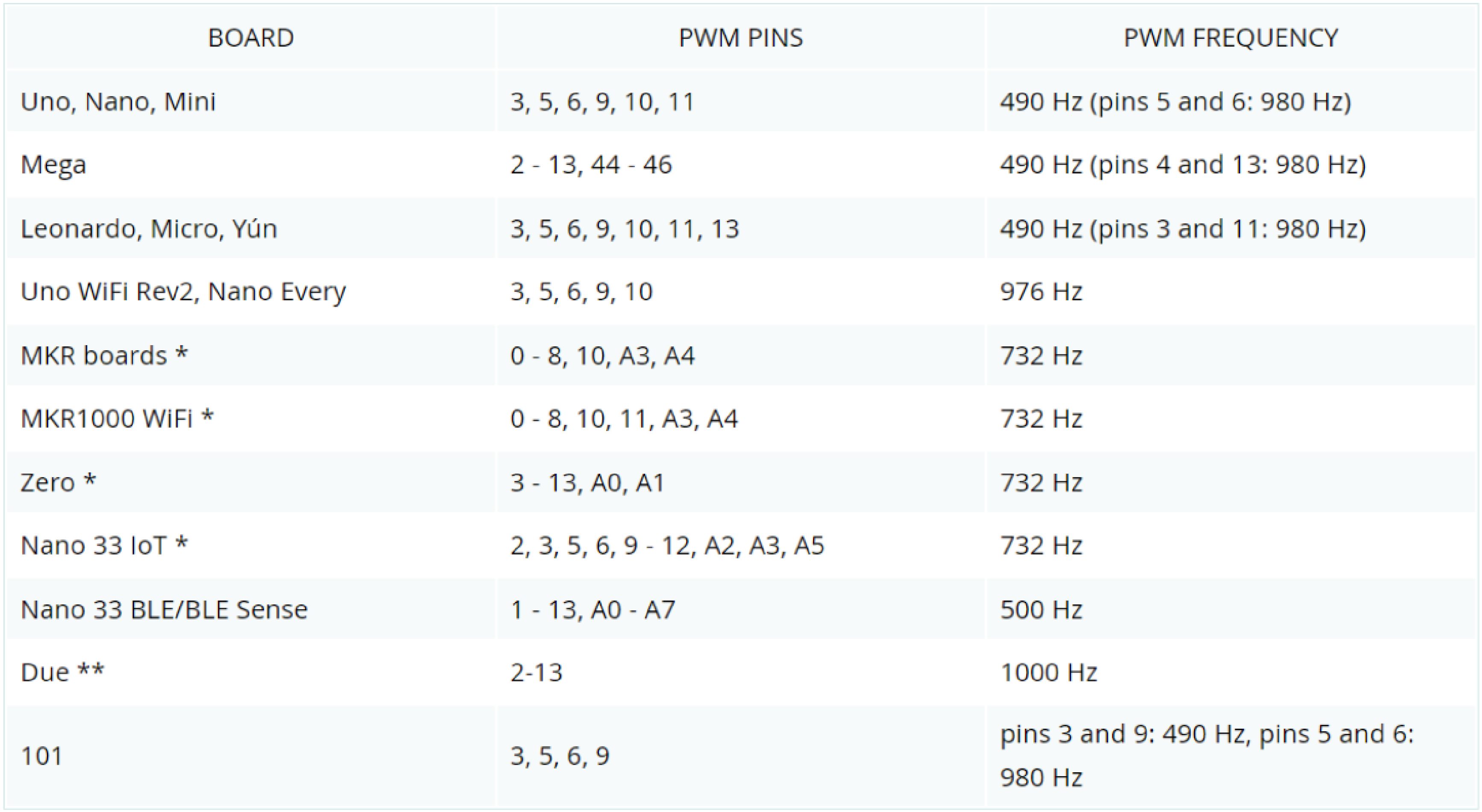
- Pulse Width Modulation (PWM)
  - A technique for getting analog results with digital means
  - Digital control is used to create a square wave, a signal switched between ON and OFF
  - The simulated analog voltage is the portion of the time the signal spends ON versus the time that the signal spends OFF


Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
25% Duty Cycle – analogWrite(64)
50% Duty Cycle – analogWrite(127)
75% Duty Cycle – analogWrite(191)
100% Duty Cycle – analogWrite(255)
5v
0v

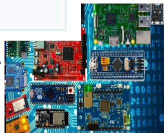
https://docs.arduino.cc/learn/microcontrollers/analog output

NANO 33 SENSE

- PWM Frequency and Pins

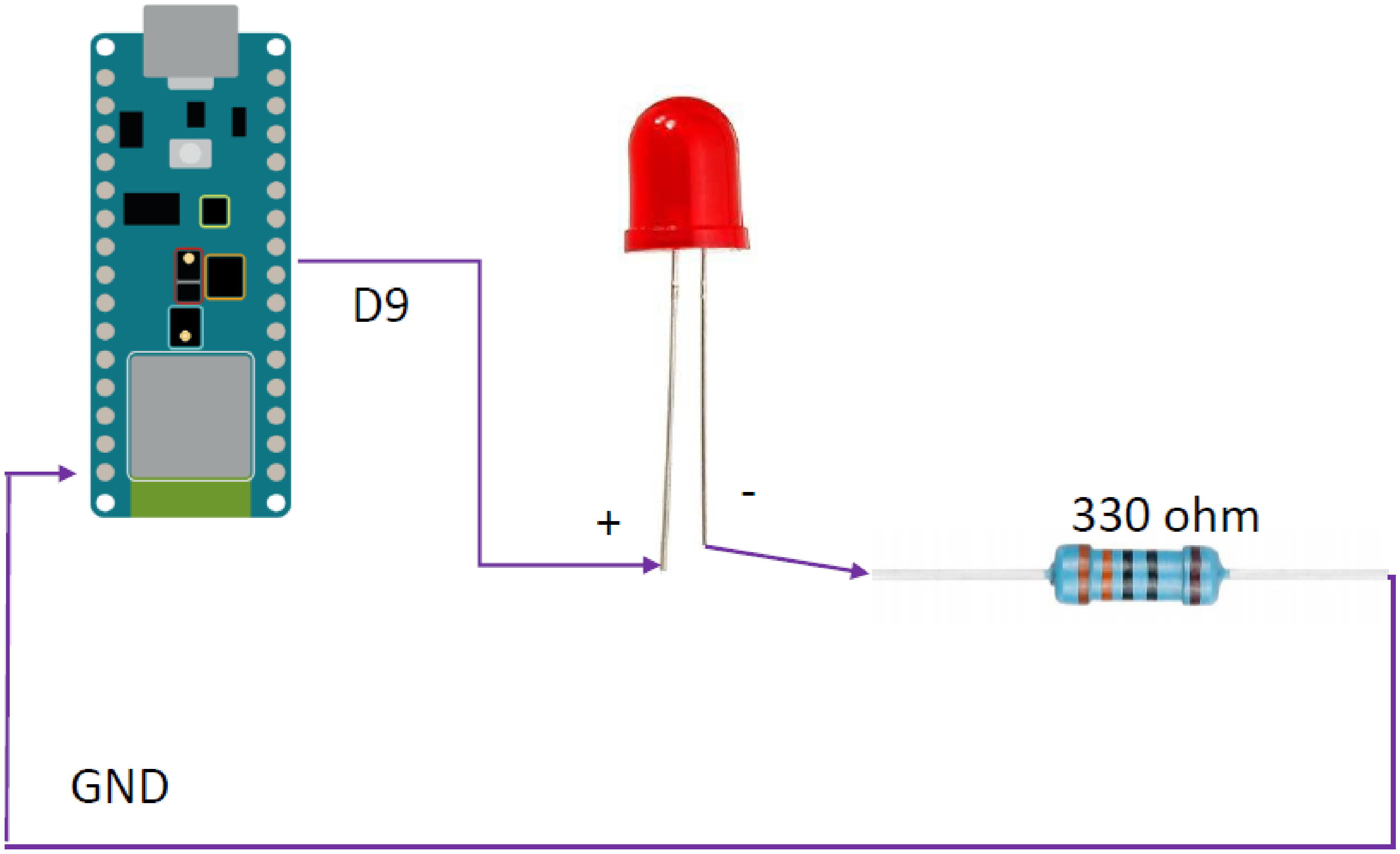


BOARD	PWM PINS	PWM FREQUENCY
Uno, Nano, Mini	3, 5, 6, 9, 10, 11	490 Hz (pins 5 and 6: 980 Hz)
Mega	2 - 13, 44 - 46	490 Hz (pins 4 and 13: 980 Hz)
Leonardo, Micro, Yún	3, 5, 6, 9, 10, 11, 13	490 Hz (pins 3 and 11: 980 Hz)
Uno WiFi Rev2, Nano Every	3, 5, 6, 9, 10	976 Hz
MKR boards *	0 - 8, 10, A3, A4	732 Hz
MKR1000 WiFi *	0 - 8, 10, 11, A3, A4	732 Hz
Zero *	3 - 13, A0, A1	732 Hz
Nano 33 IoT *	2, 3, 5, 6, 9 - 12, A2, A3, A5	732 Hz
Nano 33 BLE/BLE Sense	1 - 13, A0 - A7	500 Hz
Due **	2-13	1000 Hz
101	3, 5, 6, 9	pins 3 and 9: 490 Hz, pins 5 and 6: 980 Hz



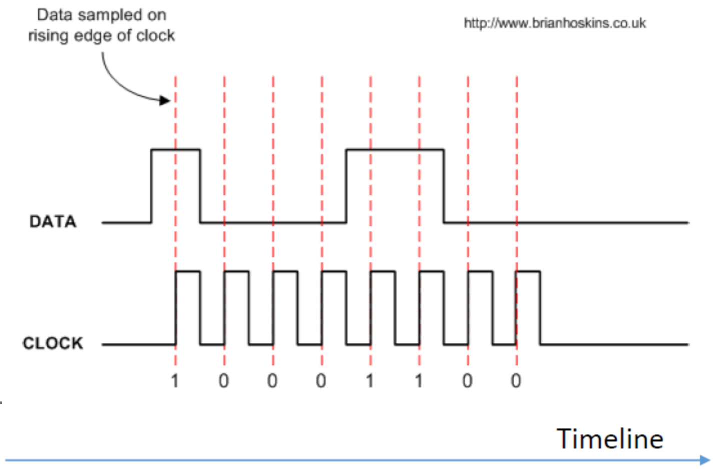
NANO 33 SENSE

- PWM example (use PWM to fade LED)
  - Examples--> 01. Basics --> Fade



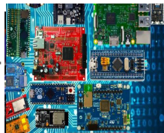
NANO 33 SENSE

- Two-wire serial communication toy example
  - Serial interface is common in device communications
    - Send / Receive data one bit at a time
    - Need a clock signal to synchronize timing between sending and receiving ends
  - Sender does this to send a byte:
    - Set the data pin to the first bit (MSB) of the byte to be sent
    - Wait for the signal to be stable
    - Pulse the clock pin
    - Repeat step 1 for the next bit until all 8 bits are sent


Data sampled on rising edge of clock
http://www.brianhoskins.co.uk
DATA
CLOCK
1 0 0 0 1 1 0 0
Timeline

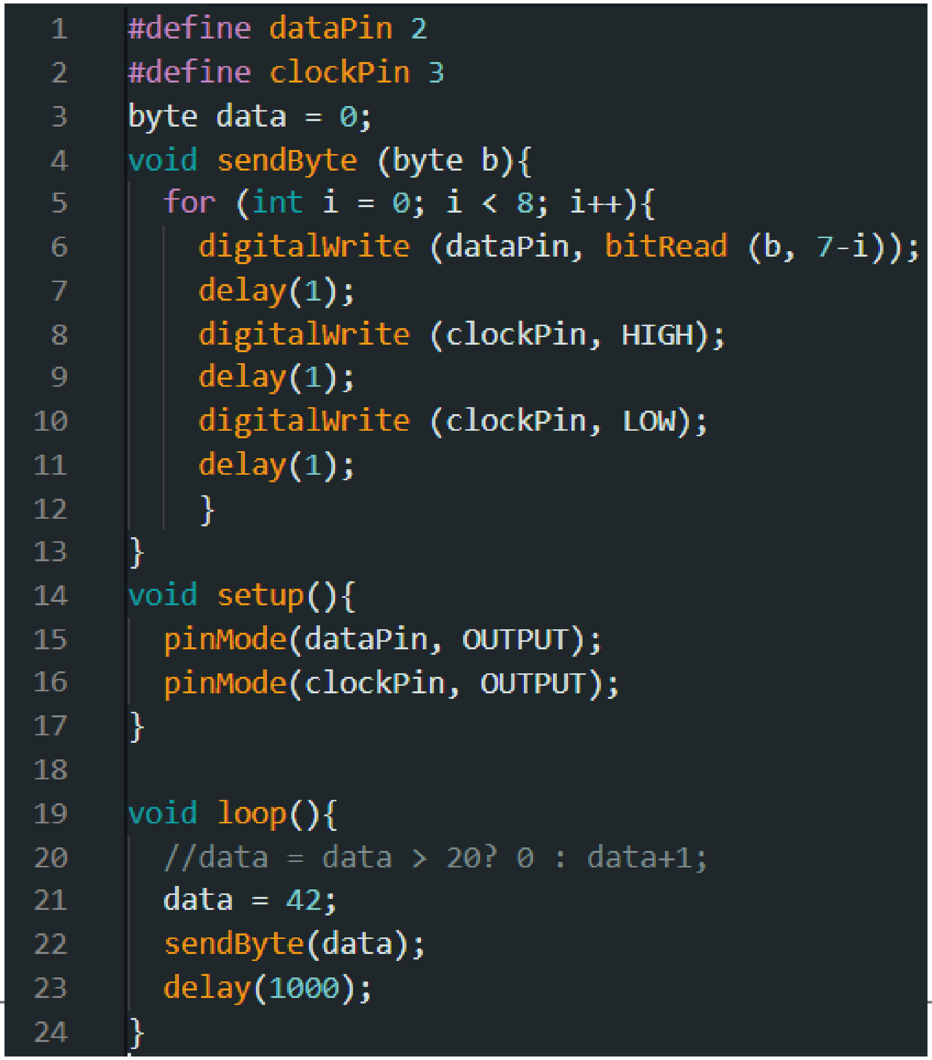
NANO 33 SENSE

- Two-wire serial communication toy example
  - Serial interface is common in device communications
    - Send / Receive data one bit at a time
    - Need a clock signal to synchronize timing between sending and receiving ends
  - Sender does this to send a byte:
    - Set the data pin to the first bit (MSB) of the byte to be sent
    - Wait for the signal to be stable
    - Pulse the clock pin
    - Repeat step 1 for the next bit until all 8 bits are sent



NANO 33 SENSE

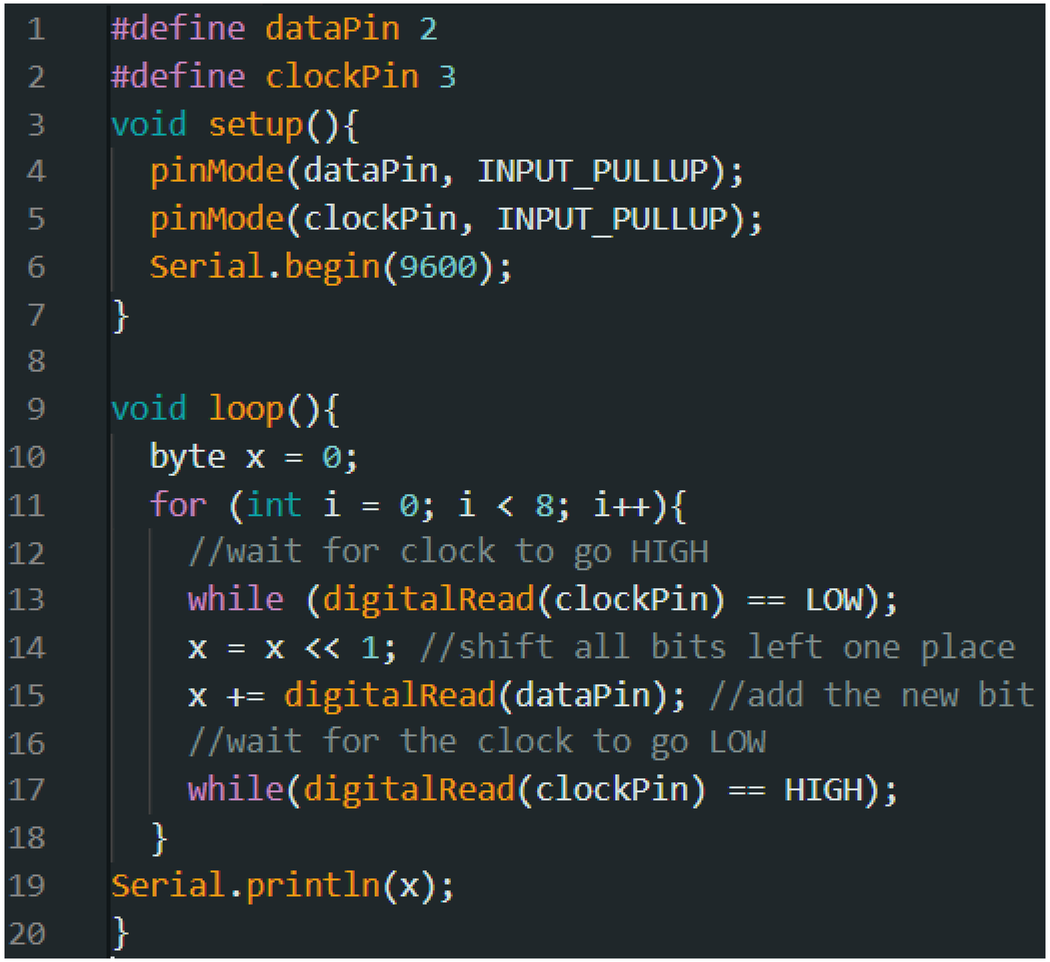
- Two-wire serial send (tx) example



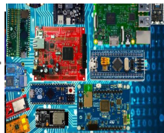
```
#define dataPin 2
#define clockPin 3
byte data = 0;
void sendByte (byte b){
  for (int i = 0; i < 8; i++){
    digitalWrite (dataPin, bitRead (b, 7-i));
    delay(1);
    digitalWrite (clockPin, HIGH);
    delay(1);
    digitalWrite (clockPin, LOW);
    delay(1);
    }
}
void setup(){
  pinMode(dataPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
}

void loop(){
  //data = data > 20? 0 : data+1;
  data = 42;
  sendByte(data);
  delay(1000);
}
```

NANO 33 SENSE

❖ Two-wire serial receive (rx) example

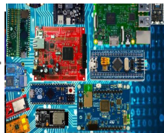


```
#define dataPin 2
#define clockPin 3
void setup(){
  pinMode(dataPin, INPUT_PULLUP);
  pinMode(clockPin, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop(){
  byte x = 0;
  for (int i = 0; i < 8; i++){
    //wait for clock to go HIGH
    while (digitalRead(clockPin) == LOW);
    x = x << 1; //shift all bits left one place
    x += digitalRead(dataPin); //add the new bit
    //wait for the clock to go LOW
    while(digitalRead(clockPin) == HIGH);
  }
Serial.println(x);
}
```



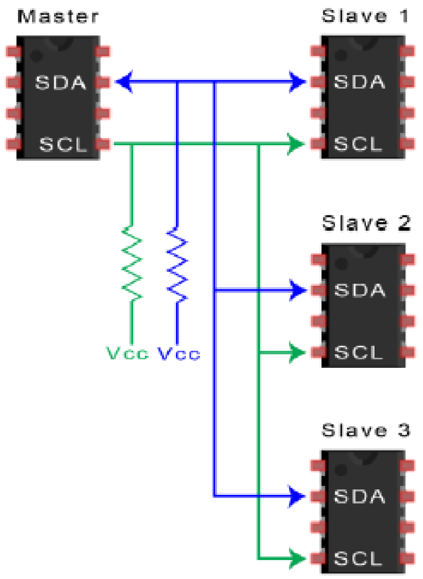
NANO 33 SENSE

- Two-wire serial communication
  - Upload the Tx code to the sender board, upload Rx code to the receiver board
    - Make sure two boards have the same I/O voltage
    - e.g., don’t connect Uno (5v) with Nano (3.3v)
  - Connect the sender and receiver
    - sender’s dataPin --> receiver’s dataPin
    - sender’s clockPin --> receiver’s clockPin
  - Open the Serial Monitor on the receiver end to see data received
  - CAUTION
    - Don’t upload the sender code to both boards and connect them
    - Can damage boards when two digital output pins connect



NANO 33 SENSE

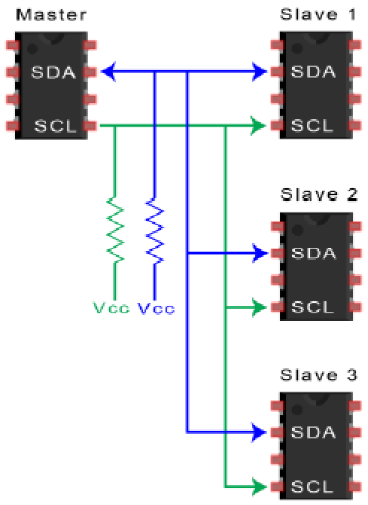
- I2C Interface
  - I2C is a serial communication protocol, commonly used for connecting MCU and sensors
  - It is a "bus", meaning multiple devices can use the same two wires (one for timing, one for data)
  - In other words, the MCU can use only two wires to communicate with multiple (up to 127) devices (i.e., sensors)



NANO 33 SENSE

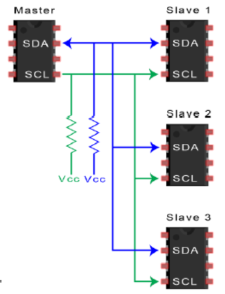
I2C Interface

- The device that provides the clock signal is the “master”, all others are “slaves”
  - Typically, the MCU is the master, sensor units are slaves
  - Each slave connected on the bus should have a unique address, which is a number between 1 and 127
    - The value 0 is the broadcast address
  - The master identifies the slave by the slave’s address
  - For a sensor unit, its I2C address is typically hardcoded, and is explained in the datasheet



NANO 33 SENSE

- I2C hardware
  - I2C uses two wires (thus, also called two wire interface, or TWI): Serial Clock (SCL) line and Serial Data (SDA) line
  - When no data is transmitted, SCL and SDA lines are in a tri state or free state, i.e., neither HIGH nor LOW, a floating value
  - When there is data to be transmitted, the sender (master or slave) takes the SDA line out of tri state and sends data as logic highs and lows in time with the clock signal
  - When transmission is complete, the clock signal can stop, and the SDA line is returned to tri state
  - Usually the slave only sends data upon the master's request, so the clock signal is guaranteed available

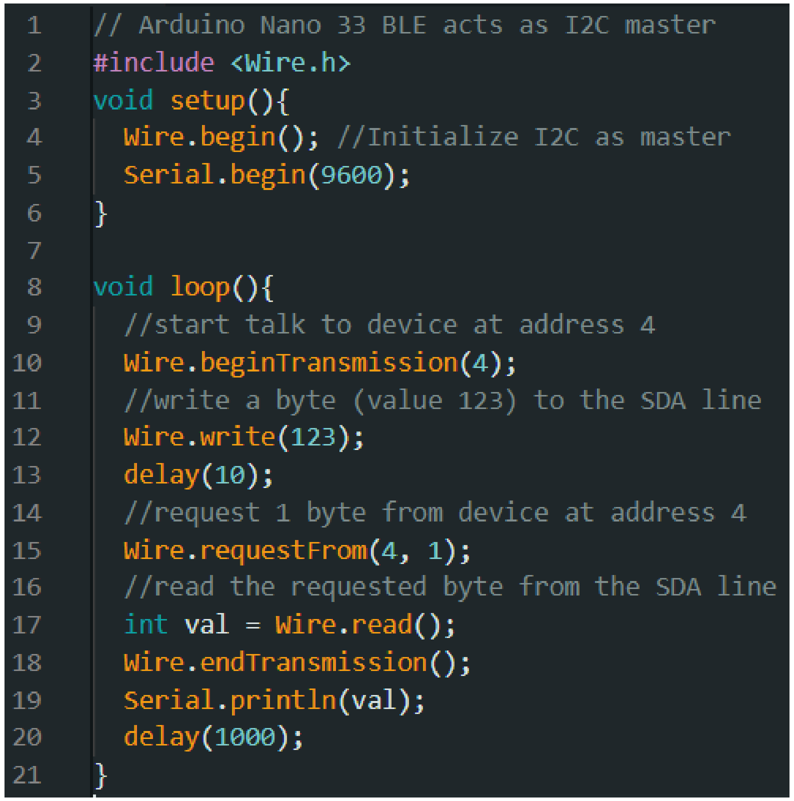


NANO 33 SENSE

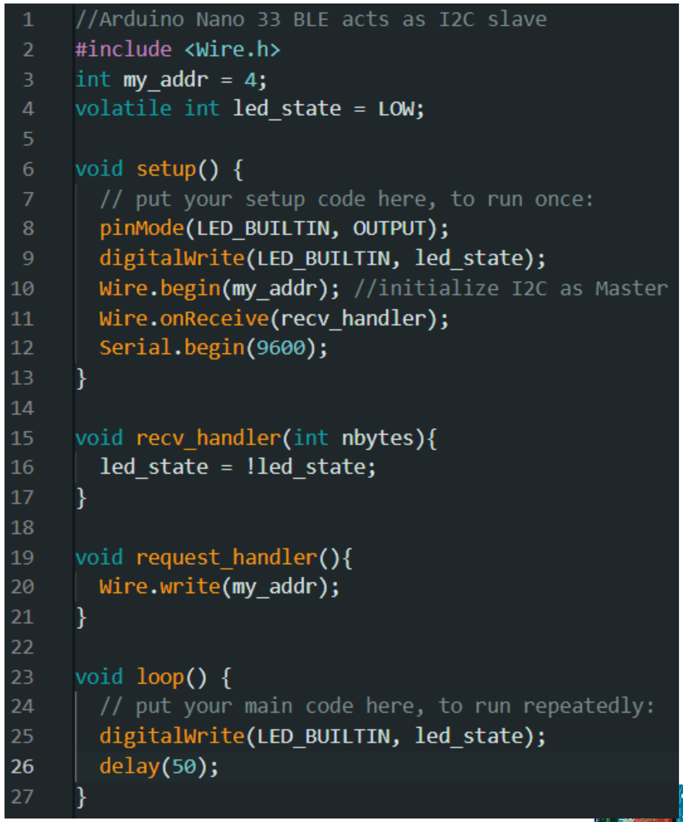
- I2C Library
  - On Arduino, the Wire library provides functions for I2C communication
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/begin
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/end
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/requestfrom
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/begintransmission
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/endtransmission
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/write
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/available
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/read
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/setclock
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/onreceive
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/onrequest
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/setwiretimeout
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/clearwiretimeoutflag
    - https://www.arduino.cc/reference/en/language/functions/communication/wire/getwiretimeoutflag

NANO 33 SENSE

- I2C experiments
  - Connect two Arduino Nano 33 BLE boards via I2C
  - One acts as master and the other acts as slave



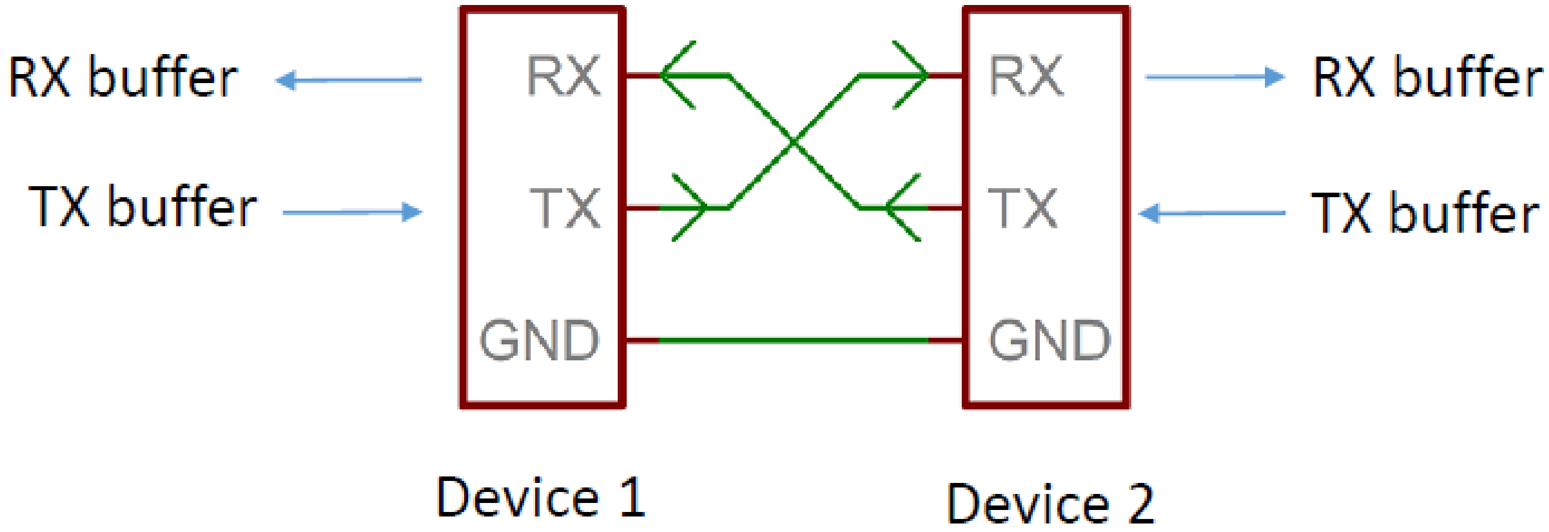
```
// Arduino Nano 33 BLE acts as I2C master
#include <Wire.h>
void setup(){
  Wire.begin(); //Initialize I2C as master
  Serial.begin(9600);
}

void loop(){
  //start talk to device at address 4
  Wire.beginTransmission(4);
  //write a byte (value 123) to the SDA line
  Wire.write(123);
  delay(10);
  //request 1 byte from device at address 4
  Wire.requestFrom(4, 1);
  //read the requested byte from the SDA line
  int val = Wire.read();
  Wire.endTransmission();
  Serial.println(val);
  delay(1000);
}
```



```
//Arduino Nano 33 BLE acts as I2C slave
#include <Wire.h>
int my_addr = 4;
volatile int led_state = LOW;

void setup() {
  // put your setup code here, to run once:
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, led_state);
  Wire.begin(my_addr); //initialize I2C as Master
  Wire.onReceive(recv_handler);
  Serial.begin(9600);
}

void recv_handler(int nbytes){
  led_state = !led_state;
}

void request_handler(){
  Wire.write(my_addr);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED_BUILTIN, led_state);
  delay(50);
}
```

NANO 33 SENSE

- UART Protocol
  - UART (Universal asynchronous receiver transmitter) is
    - A device to device communication protocol
    - A block of circuitry responsible for implementing serial communication (hardware UART)

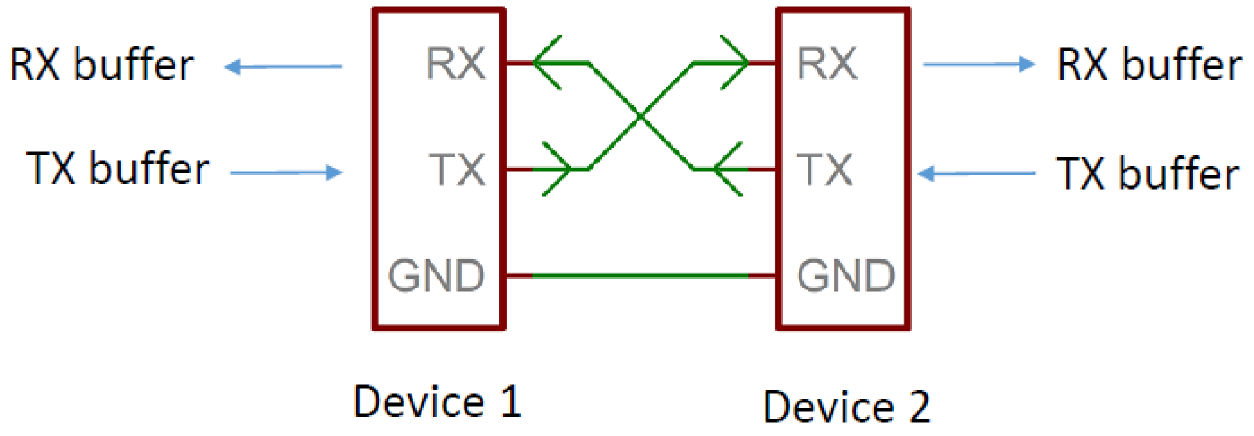


https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

https://learn.sparkfun.com/tutorials/serial-communication/rules-of-serial

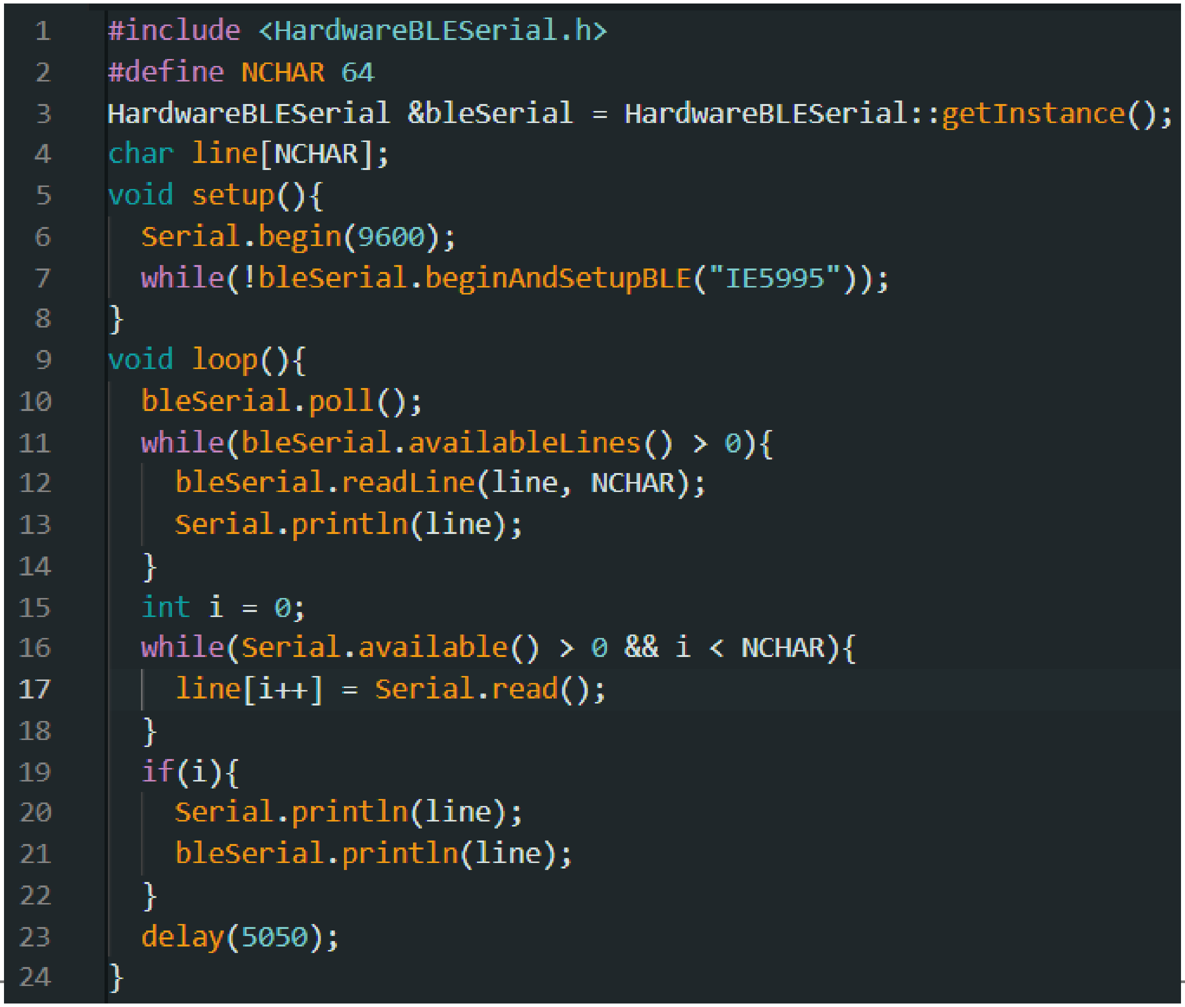
NANO 33 SENSE

- UART Protocol
  - The interface can be bit banged, i.e., directly controlled by the processor, called Software UART
  - Processor intensive, not preferred but does the job when hardware UART is not available
  - Baud rate must be set the same on both ends of the communication channel
  - Most typical Baud Rates: 9600 , 19200, 38400, 57600, 115200 , 230400, 460800, 921600, 1000000, 1500000



NANO 33 SENSE

❖ BLESerial on Nano 33 BLE



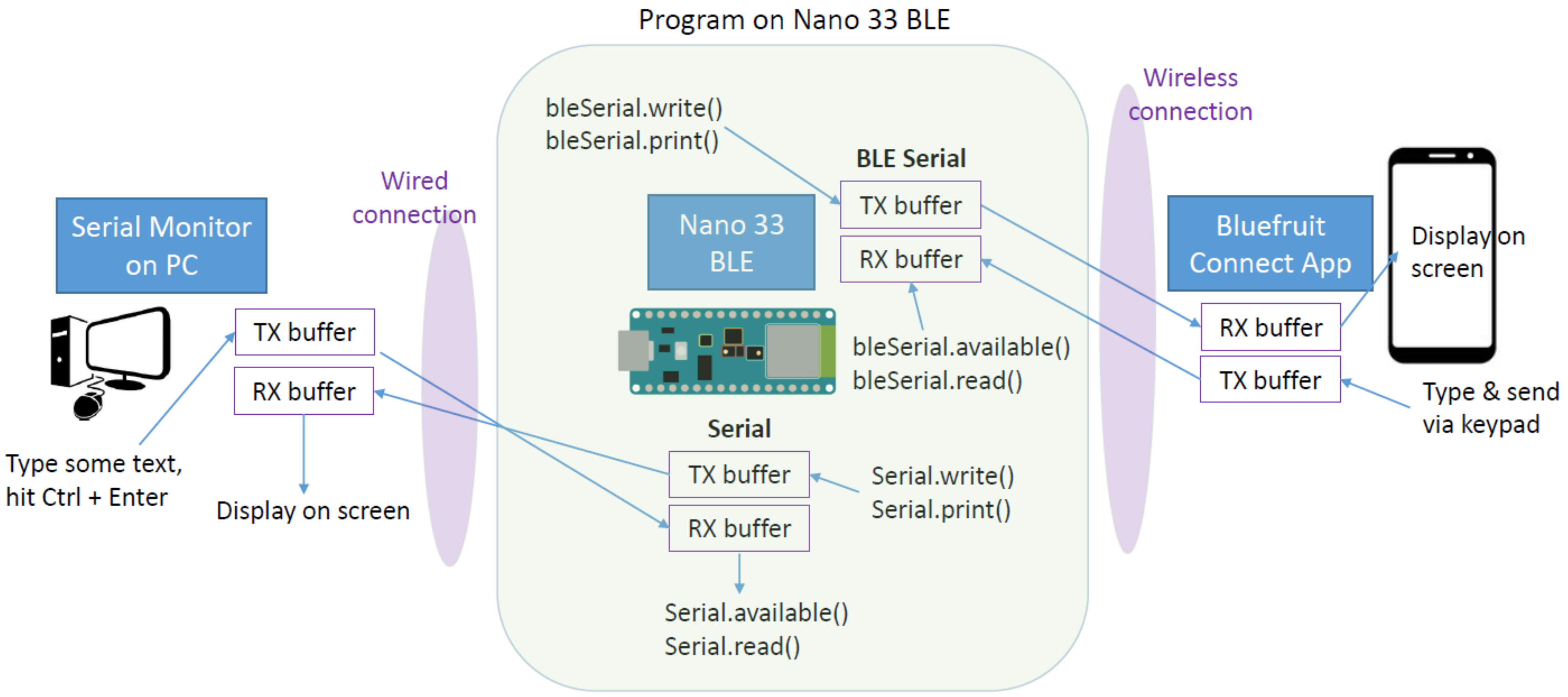
```
#include <HardwareBLESerial.h>
#define NCHAR 64
HardwareBLESerial &bleSerial = HardwareBLESerial::getInstance();
char line[NCHAR];
void setup(){
  Serial.begin(9600);
  while(!bleSerial.beginAndSetupBLE("IE5995"));
}
void loop(){
  bleSerial.poll();
  while(bleSerial.availableLines() > 0){
    bleSerial.readLine(line, NCHAR);
    Serial.println(line);
  }
  int i = 0;
  while(Serial.available() > 0 && i < NCHAR){
    line[i++] = Serial.read();
  }
  if(i){
    Serial.println(line);
    bleSerial.println(line);
  }
  delay(5050);
}
```

NANO 33 SENSE

- BLESerial on Nano 33 BLE
  - Install the HardwareBLESerial library
  - Upload the code to Nano 33 BLE
    - Modify “IE5995” to something else before uploading, to prevent conflicts
  - Download the phone app “Bluefruit Connect”
    - Via App Store or Google Play
  - Connect to the device via Bluefruit Connect
  - Open UART in Bluefruit Connect
  - Open Serial Monitor of the PC that’s connected to Nano 33 BLE
  - You can exchange text messages between the Serial Monitor on PC and the UART console in Bluefruit Connect on the cellphone

NANO 33 SENSE

- BLESerial on Nano 33 BLE



NANO 33 SENSE

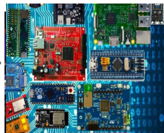
- Interrupts
  - Interrupts allow microcontrollers to respond to events without having to repeatedly poll to see if the event has occurred
  - Interrupts can be triggered by a Pin or by a Timer

Polling method:

```
void loop{
   if (digitalRead(inputPin) == LOW){
      // do something
   }
}
```

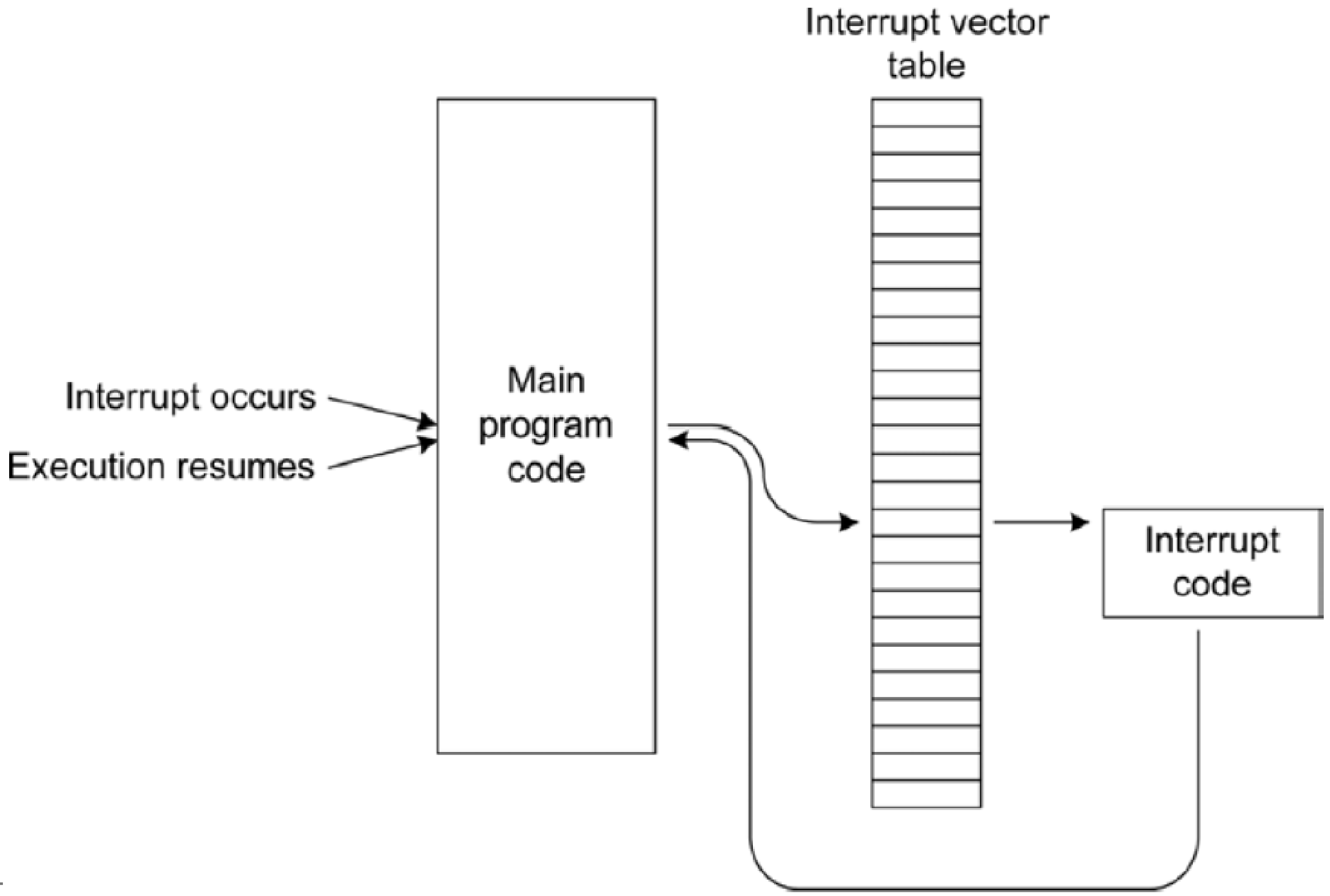
Interrupt method:

```
void setup{
   attachInterrupt(digitalPinToInterrupt(interruptPin), myISR, FALLING);
}
void loop(){}
void myISR(){
   // do something to respond to the event
}
```



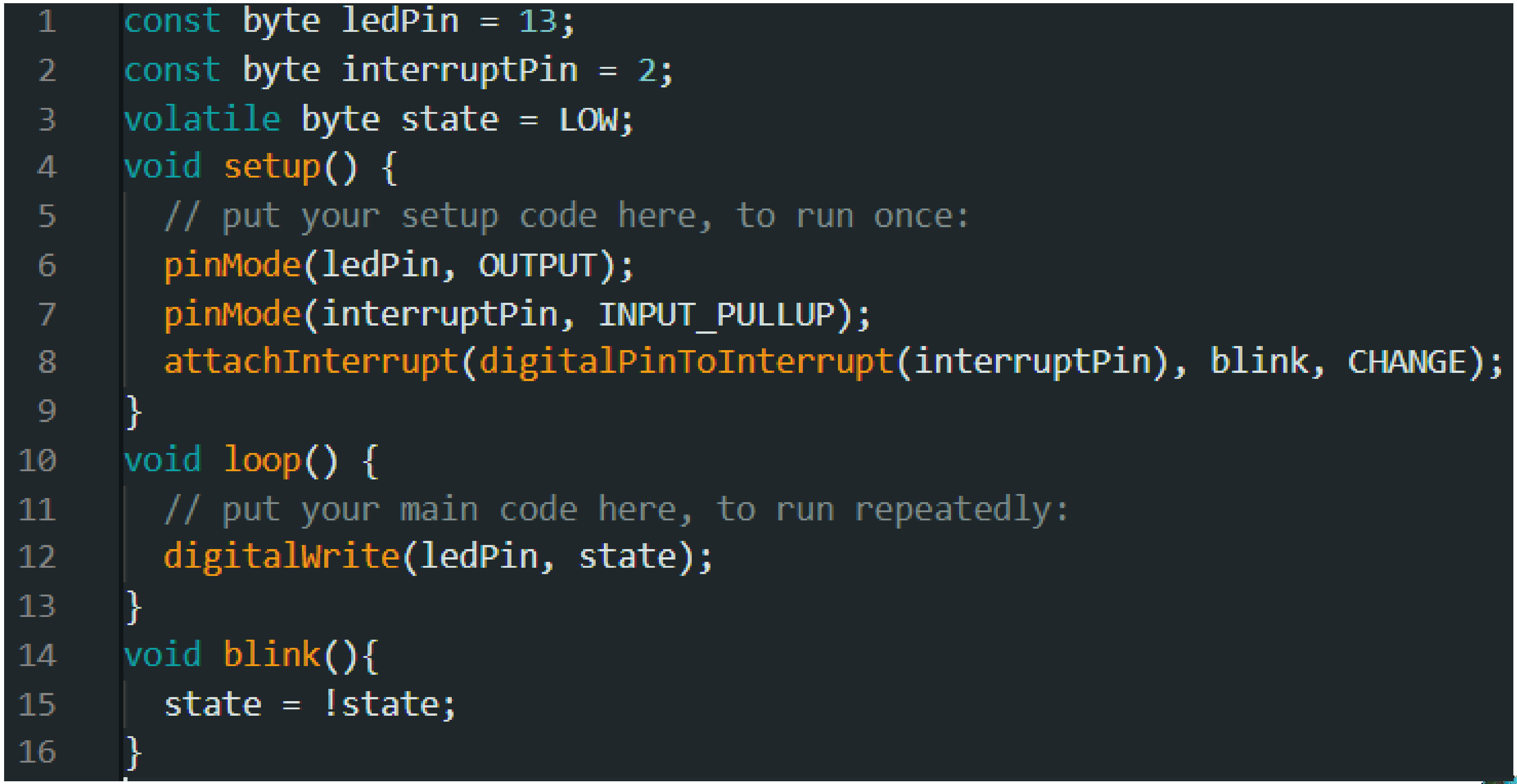
NANO 33 SENSE

- Interrupts
  - Interrupt Handling Process



NANO 33 SENSE

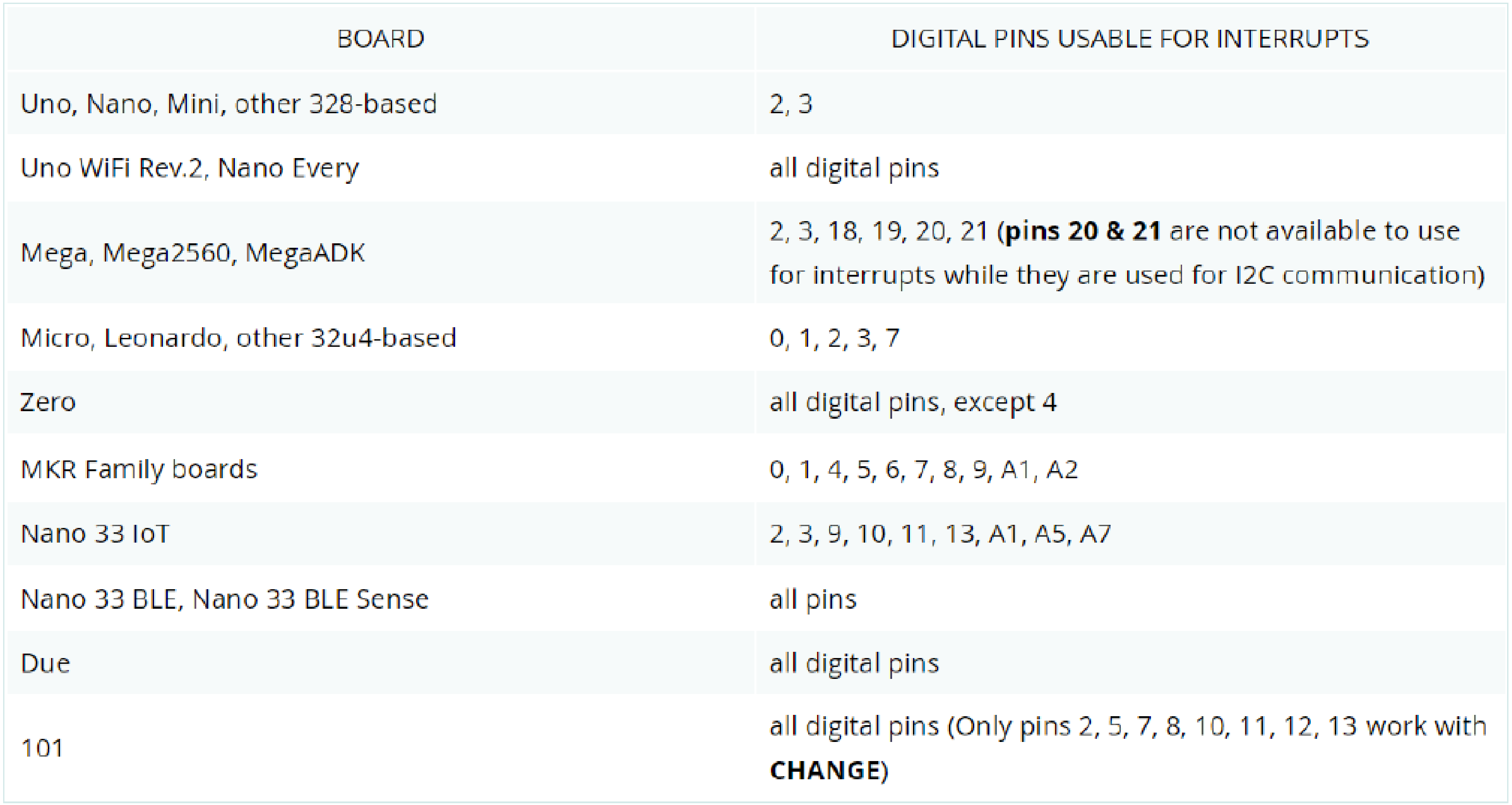
- Interrupts
  - Example



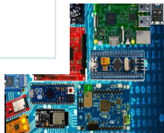
```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;
void setup() {
  // put your setup code here, to run once:
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}
void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(ledPin, state);
}
void blink(){
  state = !state;
}
```

NANO 33 SENSE

- Interrupts
  - Pins available for interrupts

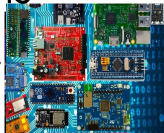


BOARD	DIGITAL PINS USABLE FOR INTERRUPTS
Uno, Nano, Mini, other 328-based	2, 3
Uno WiFi Rev.2, Nano Every	all digital pins
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21 (**pins 20 & 21** are not available to use for interrupts while they are used for I2C communication)
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Zero	all digital pins, except 4
MKR Family boards	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Nano 33 IoT	2, 3, 9, 10, 11, 13, A1, A5, A7
Nano 33 BLE, Nano 33 BLE Sense	all pins
Due	all digital pins
101	all digital pins (Only pins 2, 5, 7, 8, 10, 11, 12, 13 work with **CHANGE**)



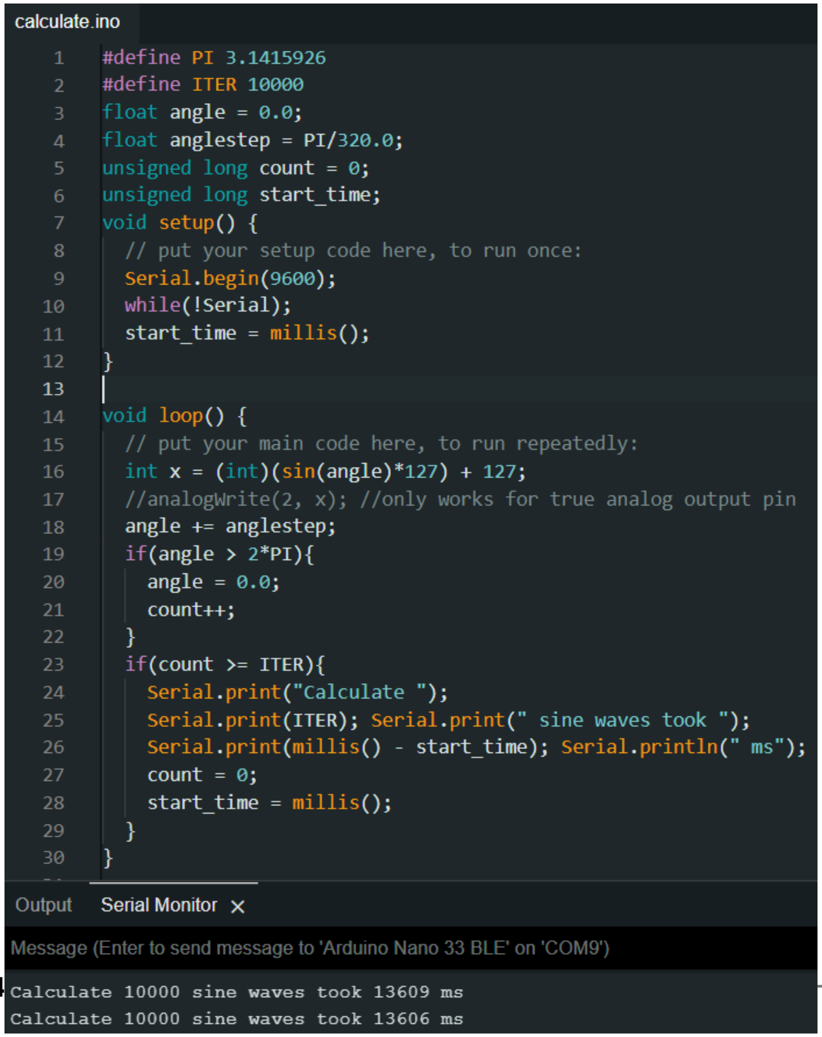
NANO 33 SENSE

- Arduino code optimization for speed
  - Avoid using float
    - Floating point arithmetic is slow on hardware that does not support it
    - Use long instead, which can retain more digits of precision, and faster
  - Lookup rather than calculate
  - Use const byte instead of int for small constants such as Pin names
  - Move looping code into the setup function
    - The loop() function checks for Serial communication which takes time
  - Speeding up Digital IO
    - Directly manipulate the pin bit in the port register
  - Speeding up Analog Inputs
    - Reducing the prescaler of the Timer that triggers ADC conversions



NANO 33 SENSE

- Arduino code optimization for speed



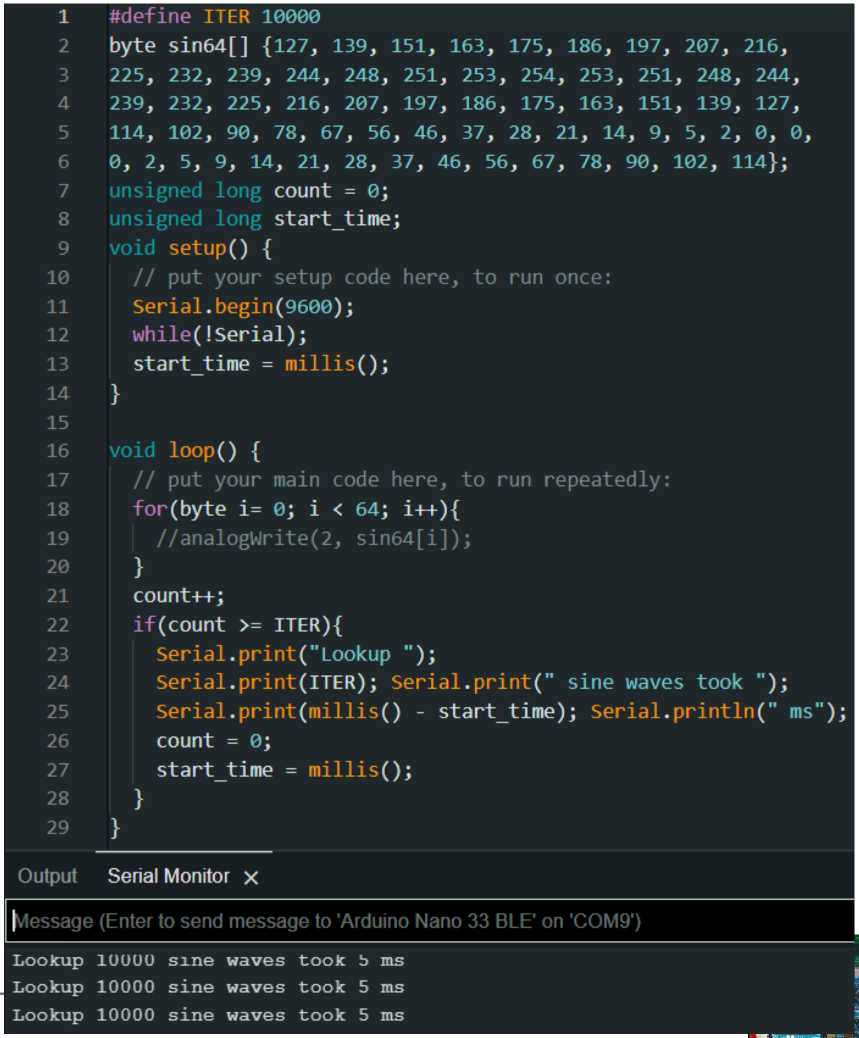
calculate.ino

```
#define PI 3.1415926
#define ITER 10000
float angle = 0.0;
float anglestep = PI/320.0;
unsigned long count = 0;
unsigned long start_time;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  while(!Serial);
  start_time = millis();
}

void loop() {
  // put your main code here, to run repeatedly:
  int x = (int)(sin(angle)*127) + 127;
  //analogWrite(2, x); //only works for true analog output pin
  angle += anglestep;
  if(angle > 2*PI){
    angle = 0.0;
    count++;
  }
  if(count >= ITER){
    Serial.print("Calculate ");
    Serial.print(ITER); Serial.print(" sine waves took ");
    Serial.print(millis() - start_time); Serial.println(" ms");
    count = 0;
    start_time = millis();
  }
}
```

Output Serial Monitor ×

Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM9')

```
Calculate 10000 sine waves took 13609 ms
Calculate 10000 sine waves took 13606 ms
```



```
#define ITER 10000
byte sin64[] {127, 139, 151, 163, 175, 186, 197, 207, 216,
225, 232, 239, 244, 248, 251, 253, 254, 253, 251, 248, 244,
239, 232, 225, 216, 207, 197, 186, 175, 163, 151, 139, 127,
114, 102, 90, 78, 67, 56, 46, 37, 28, 21, 14, 9, 5, 2, 0, 0,
0, 2, 5, 9, 14, 21, 28, 37, 46, 56, 67, 78, 90, 102, 114};
unsigned long count = 0;
unsigned long start_time;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  while(!Serial);
  start_time = millis();
}

void loop() {
  // put your main code here, to run repeatedly:
  for(byte i= 0; i < 64; i++){
    //analogWrite(2, sin64[i]);
  }
  count++;
  if(count >= ITER){
    Serial.print("Lookup ");
    Serial.print(ITER); Serial.print(" sine waves took ");
    Serial.print(millis() - start_time); Serial.println(" ms");
    count = 0;
    start_time = millis();
  }
}
```

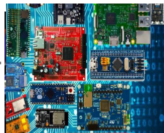
Output Serial Monitor ×

Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM9')

```
Lookup 10000 sine waves took 5 ms
Lookup 10000 sine waves took 5 ms
Lookup 10000 sine waves took 5 ms
```

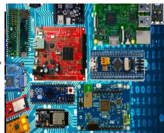
NANO 33 SENSE

- Minimize power usage
  - If the product is battery operated, reducing power usage is important
  - Ways to reduce power consumption
    - Put the microcontroller in sleep when it is not doing anything
    - Turn off (disable) unused peripherals
    - Use lowest clock frequency sufficient
    - Use lower input voltage if allowable
    - Provide power to sensor only when taking a reading



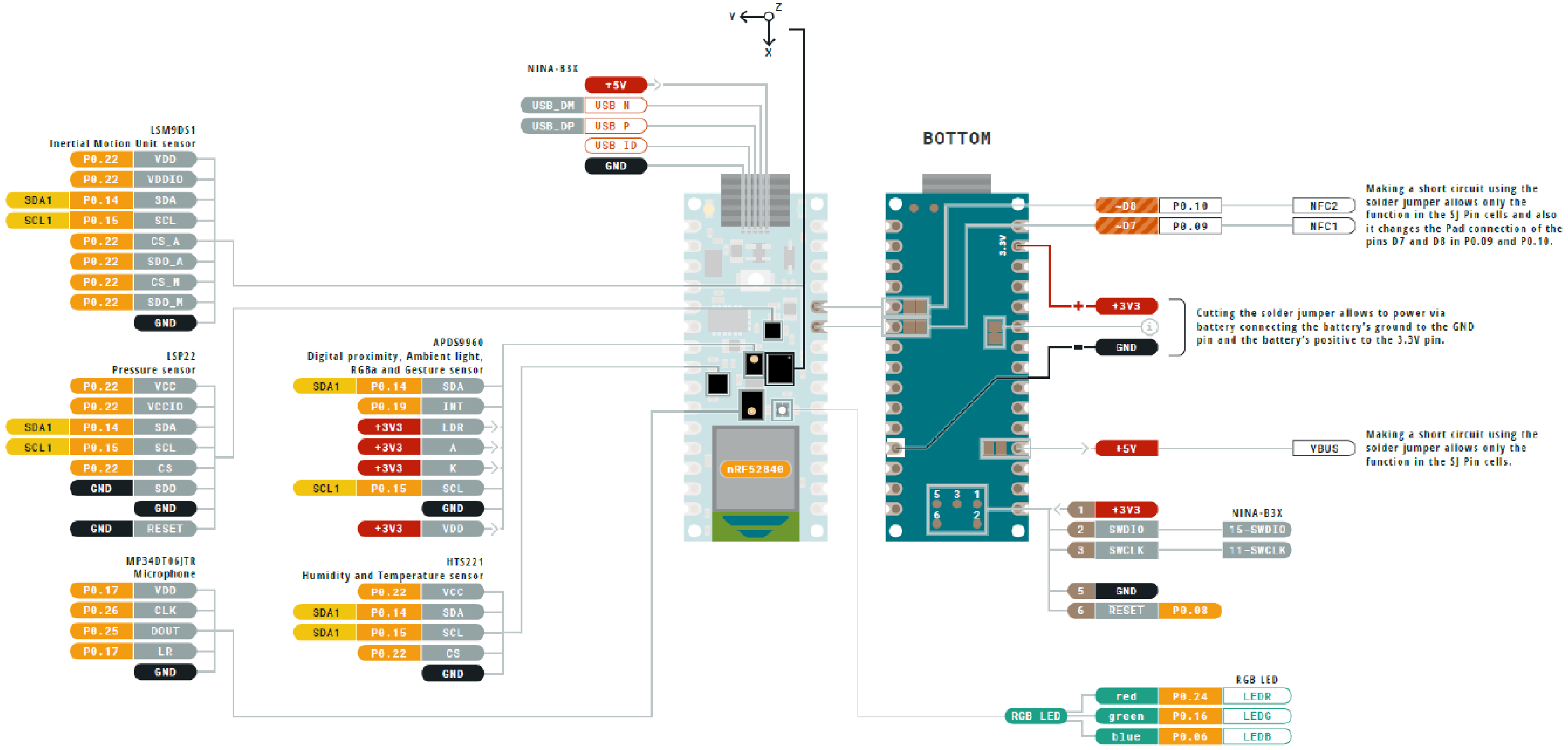
NANO 33 SENSE

- Minimize RAM usage
  - Reduce the amount of memory used by data and variables
    - Choose data type based on need, especially for large arrays
      - Int is at least 2 byte, values range in [32,768, 32767]
      - If the value range of [0, 255] suffices, use byte , which is 1 byte
    - Store string constants in Flash Memory
      - Serial.println("Some message to print"); // Stored both in flash memory and in RAM
      - Serial.println(F ("Some message to print")); // Fetched from flash memory, no RAM use
    - Use PROGMEM directive to store constant arrays in Flash
      - #include <avr/pgmspace.h>
    - Use const if the variable value do not change while the sketch is running
      - "const int v = 5000;" uses less memory than "int v = 5000;" and runs faster when v is used in computation
    - Bypass the bootloader
      - The Nano 33 BLE bootloader takes 35 KB of Flash Memory
      - There is a way to program the chip without using a bootloader, thus saving RAM and reducing start-up time
    - Length of variable names does not help save memory



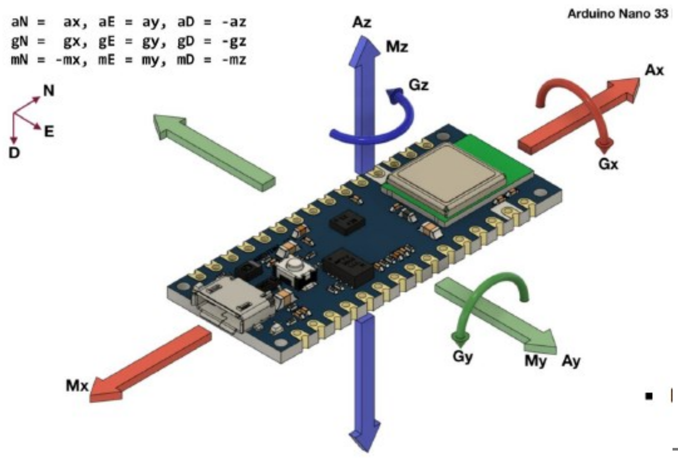
NANO 33 SENSE

- Integrated sensors on Nano 33 BLE Sense

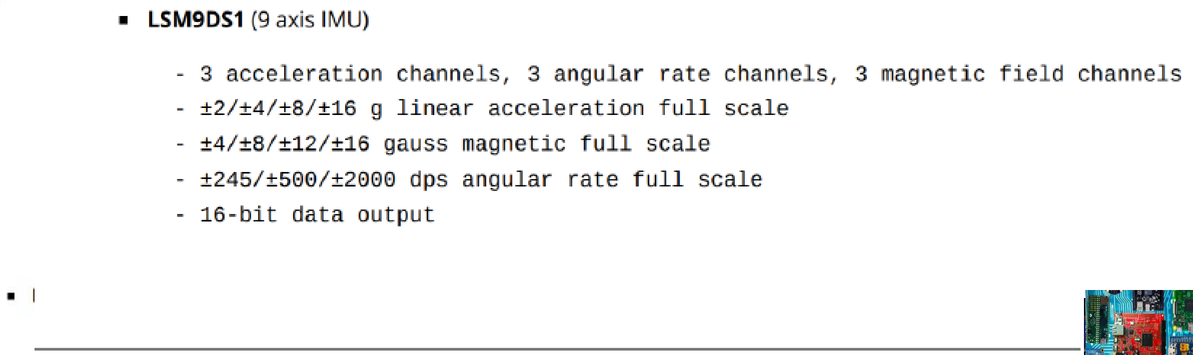


NANO 33 SENSE

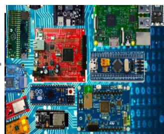
- Integrated sensors on Nano 33 BLE Sense
  - LSM9DS1
    - The LSM9DS1 inertial measurement unit (IMU)
      - accelerometer
      - gyroscope
      - magnetometer
    - Useful for detecting orientation, motion or vibrations
    - Datasheet: https://content.arduino.cc/assets/Nano_BLE_Sense_lsm9ds1.pdf
    - Library "Arduino_LSM9DS1"



aN = ax, aE = ay, aD = -az
gN = gx, gE = gy, gD = -gz
mN = -mx, mE = my, mD = -mz
N
E
D
Arduino Nano 33
Az
Mz
Gz
Ax
Gx
Mx
Gy
My
Ay

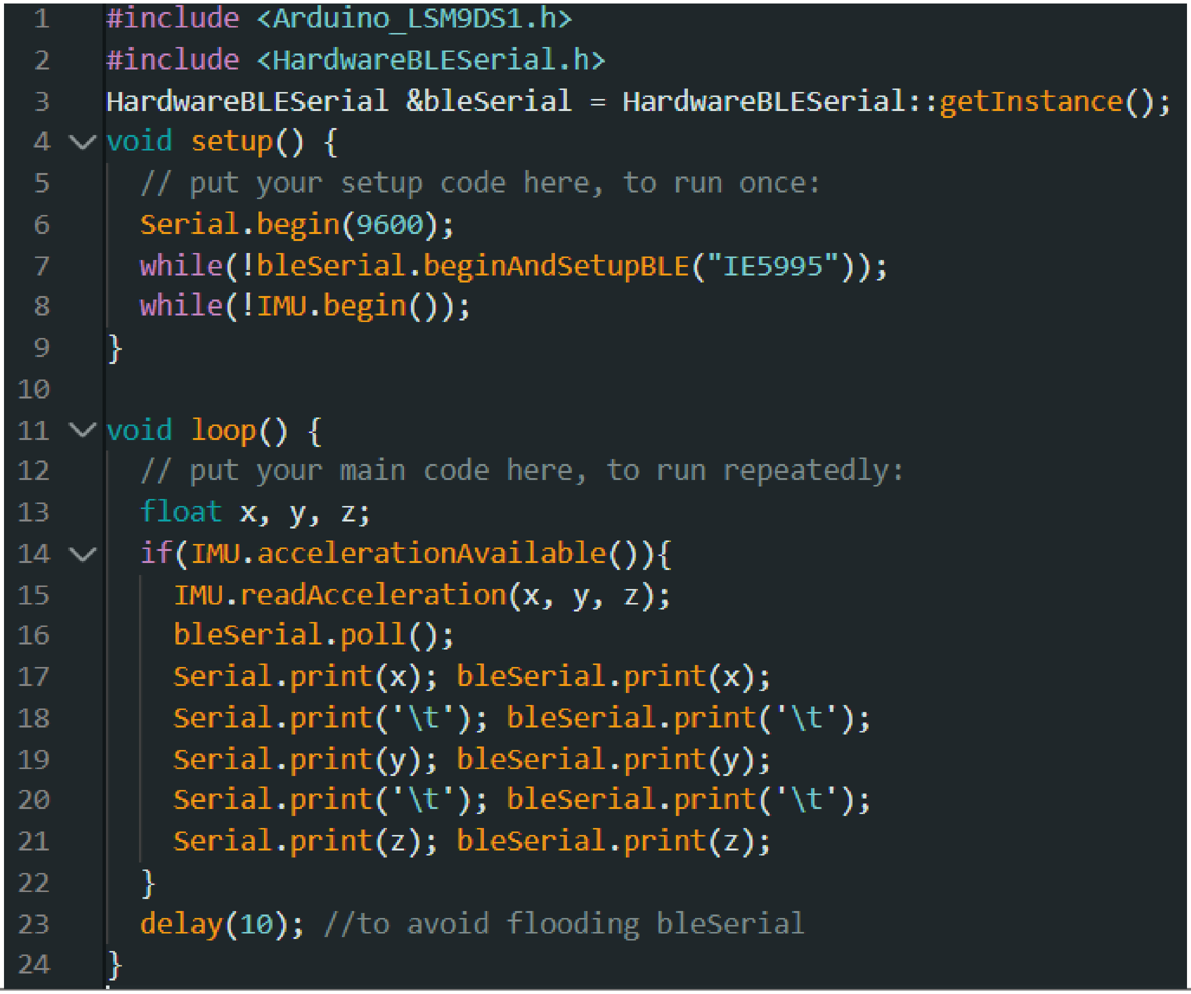


- **LSM9DS1** (9 axis IMU)
  - 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels
  - ±2/±4/±8/±16 g linear acceleration full scale
  - ±4/±8/±12/±16 gauss magnetic full scale
  - ±245/±500/±2000 dps angular rate full scale
  - 16-bit data output



NANO 33 SENSE

- Integrated sensors on Nano 33 BLE Sense
  - LSM9DS1



```
#include <Arduino_LSM9DS1.h>
#include <HardwareBLESerial.h>
HardwareBLESerial &bleSerial = HardwareBLESerial::getInstance();
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  while(!bleSerial.beginAndSetupBLE("IE5995"));
  while(!IMU.begin());
}

void loop() {
  // put your main code here, to run repeatedly:
  float x, y, z;
  if(IMU.accelerationAvailable()){
    IMU.readAcceleration(x, y, z);
    bleSerial.poll();
    Serial.print(x); bleSerial.print(x);
    Serial.print('\t'); bleSerial.print('\t');
    Serial.print(y); bleSerial.print(y);
    Serial.print('\t'); bleSerial.print('\t');
    Serial.print(z); bleSerial.print(z);
  }
  delay(10); //to avoid flooding bleSerial
}
```

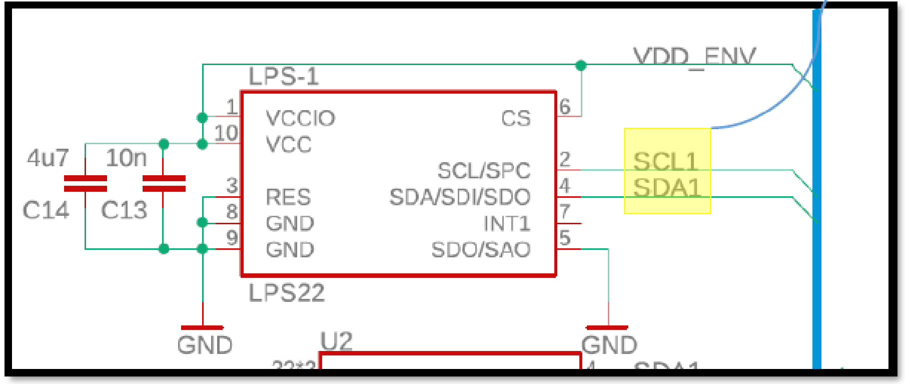
NANO 33 SENSE

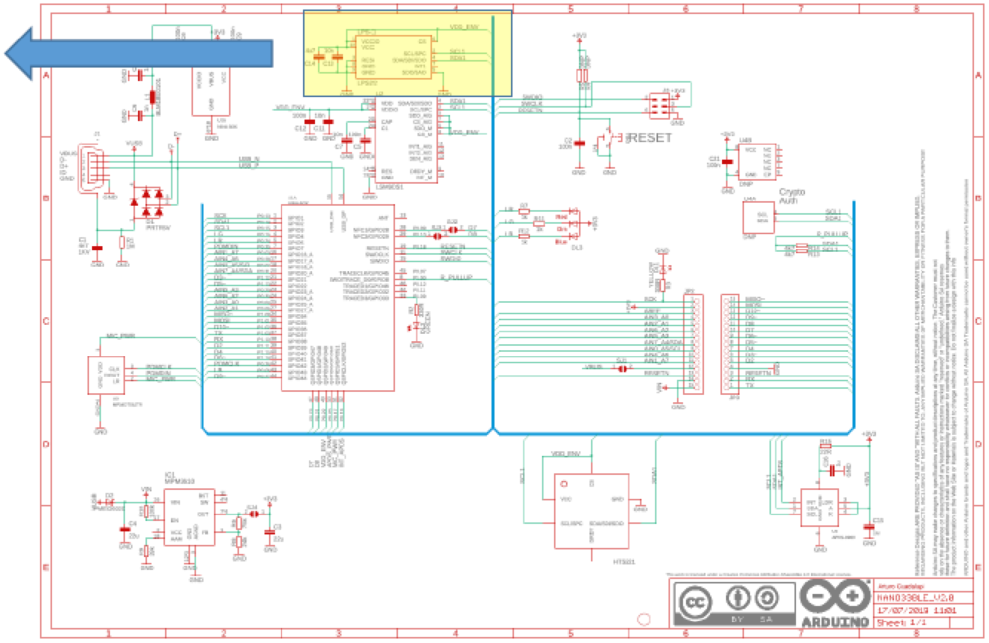
- Interact with sensors on the I2C bus
  - There are ready made libraries to read sensor data from the integrated sensors on the BLE Sense board
  - For the sake of learning, let us implement our own “device driver” to obtain sensor data
  - We will start with the barometric pressure sensor LPS22HB

NANO 33 SENSE

- Interact with sensors on the I2C bus
  - Write your own device driver for LPS22HB
    - First, find out how LPS22HB is physically connected to the nrf52840 MCU on the BLE Sense Rev2 board

Note: SCL1 and SDA1 = Wire1 object in the Wire library; whereas SCL and SDA = Wire.

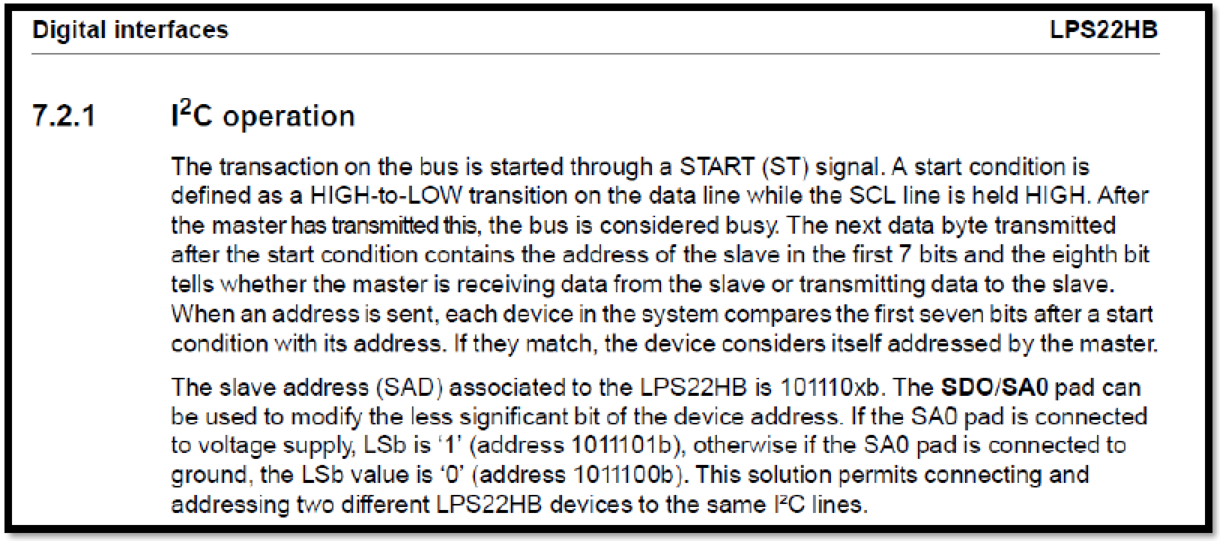




We can see that it is connected to the I2C bus through nrf52840's SCL1 and SDA1 pins. Also, note that the SDO/SA0 pin on LPS22HB is connected to GND. This information will be useful for determining the I2C address of the LPS22HB, according to its dataset.

NANO 33 SENSE

- Interact with sensors on the I2C bus
  - Write your own device driver for LPS22HB
    - Next, find the datasheet of LPS22HB and go to the I2C operation section



Digital interfaces LPS22HB

7.2.1 I²C operation

The transaction on the bus is started through a START (ST) signal. A start condition is defined as a HIGH-to-LOW transition on the data line while the SCL line is held HIGH. After the master has transmitted this, the bus is considered busy. The next data byte transmitted after the start condition contains the address of the slave in the first 7 bits and the eighth bit tells whether the master is receiving data from the slave or transmitting data to the slave. When an address is sent, each device in the system compares the first seven bits after a start condition with its address. If they match, the device considers itself addressed by the master.

The slave address (SAD) associated to the LPS22HB is 101110xb. The **SDO/SA0** pad can be used to modify the less significant bit of the device address. If the SA0 pad is connected to voltage supply, LSb is '1' (address 1011101b), otherwise if the SA0 pad is connected to ground, the LSb value is '0' (address 1011100b). This solution permits connecting and addressing two different LPS22HB devices to the same I²C lines.

Given that SDO/SA0 is connected to GND, we know that the device's I2C address should be 1011100b, which is 0x5C.

```
#define LPS22HB_ADDRESS  0x5C
```

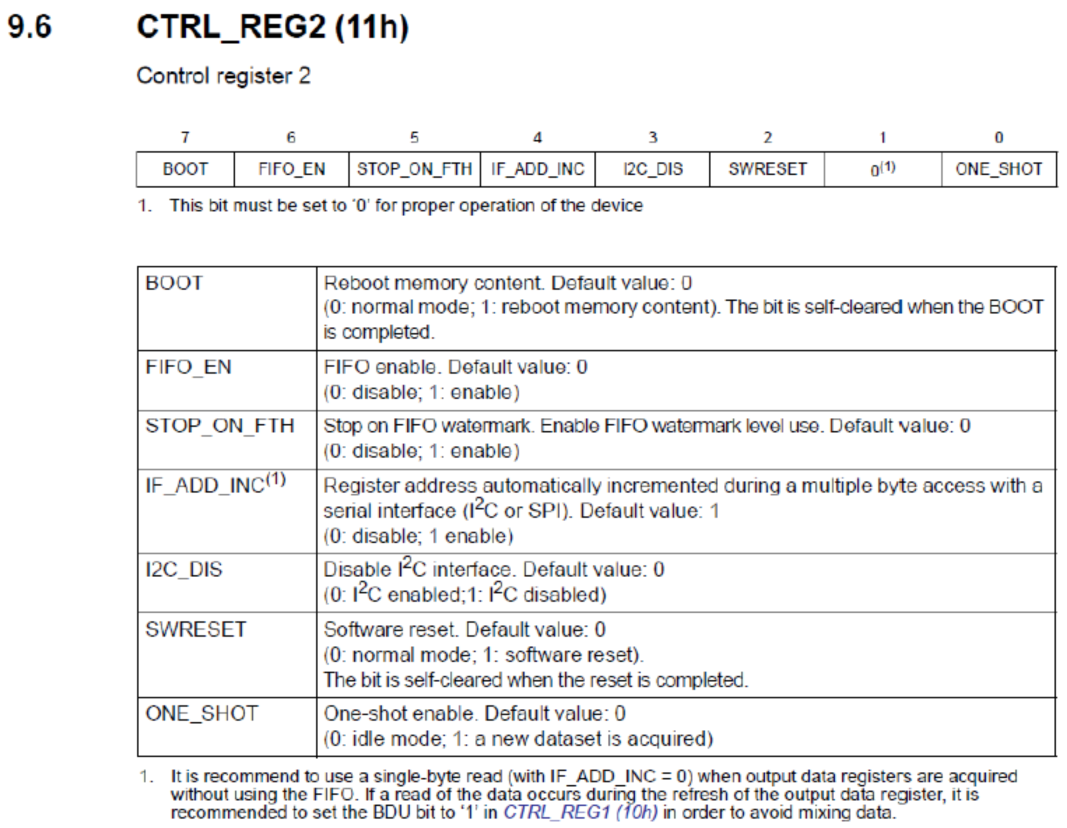
We will use Arduino's Wire library to perform I2C communication with the sensor.

```
#include <Wire.h>
```

On Nano 33 BLE Sense, integrated sensors are connected to Wire1.

NANO 33 SENSE

- Interact with sensors on the I2C bus
  - Write your own device driver for LPS22HB
    - When we need the sensor to acquire a new reading, we need to set the least significant bit (bit 0) of the CTRL_REG2 register



9.6 CTRL_REG2 (11h)

Control register 2

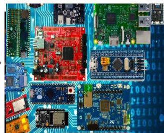
7	6	5	4	3	2	1	0
BOOT	FIFO_EN	STOP_ON_FTH	IF_ADD_INC	I2C_DIS	SWRESET	0(1)	ONE_SHOT

1. This bit must be set to '0' for proper operation of the device

BOOT	Reboot memory content. Default value: 0 (0: normal mode; 1: reboot memory content). The bit is self-cleared when the BOOT is completed.
FIFO_EN	FIFO enable. Default value: 0 (0: disable; 1: enable)
STOP_ON_FTH	Stop on FIFO watermark. Enable FIFO watermark level use. Default value: 0 (0: disable; 1: enable)
IF_ADD_INC(1)	Register address automatically incremented during a multiple byte access with a serial interface (I^{2}C or SPI). Default value: 1 (0: disable; 1 enable)
I2C_DIS	Disable I^{2}C interface. Default value: 0 (0: I^{2}C enabled;1: I^{2}C disabled)
SWRESET	Software reset. Default value: 0 (0: normal mode; 1: software reset). The bit is self-cleared when the reset is completed.
ONE_SHOT	One-shot enable. Default value: 0 (0: idle mode; 1: a new dataset is acquired)

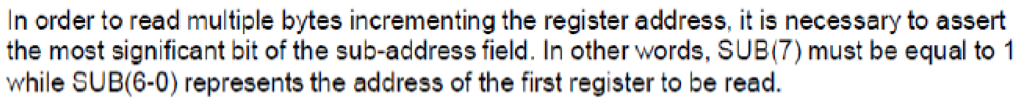
1. It is recommend to use a single-byte read (with IF_ADD_INC = 0) when output data registers are acquired without using the FIFO. If a read of the data occurs during the refresh of the output data register, it is recommended to set the BDU bit to '1' in *CTRL_REG1 (10h)* in order to avoid mixing data.

```
#define CTRL_REG2 0x11

void trigger(){
  Wire1.beginTransmission(LPS22HB_ADDRESS);
  Wire1.write(CTRL_REG2);
  Wire1.write(0x01);
  Wire1.endTransmission();
}
```

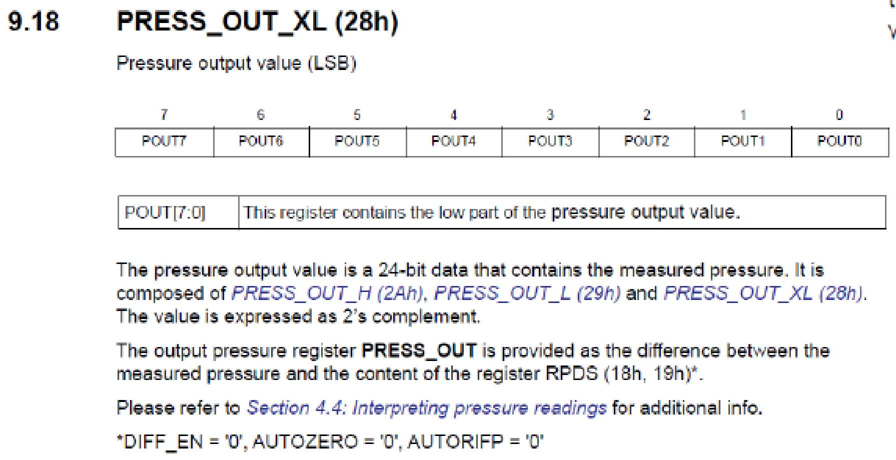


NANO 33 SENSE

- Interact with sensors on the I2C bus
  - Write your own device driver for LPS22HB
    - The pressure data is 24 bit; thus we need to read the three 8 bit registers (at addresses 0x28, 0x29 and 0x2A)
    - If IF_ADD_INC bit is set in CTRL_REG2, then register address will be automatically incremented in repeated read of multi byte data



In order to read multiple bytes incrementing the register address, it is necessary to assert the most significant bit of the sub-address field. In other words, SUB(7) must be equal to 1 while SUB(6-0) represents the address of the first register to be read.



9.18 PRESS_OUT_XL (28h)

Pressure output value (LSB)

7	6	5	4	3	2	1	0
POUT7	POUT6	POUT5	POUT4	POUT3	POUT2	POUT1	POUT0

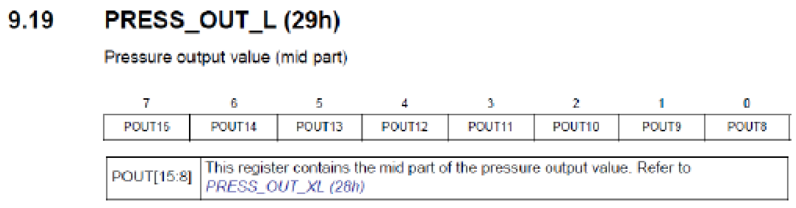
POUT[7:0]	This register contains the low part of the pressure output value.

The pressure output value is a 24-bit data that contains the measured pressure. It is composed of *PRESS_OUT_H (2Ah)*, *PRESS_OUT_L (29h)* and *PRESS_OUT_XL (28h)*. The value is expressed as 2's complement.

The output pressure register **PRESS_OUT** is provided as the difference between the measured pressure and the content of the register RPDS (18h, 19h)*.

Please refer to *Section 4.4: Interpreting pressure readings* for additional info.

*DIFF_EN = '0', AUTOZERO = '0', AUTORIFP = '0'

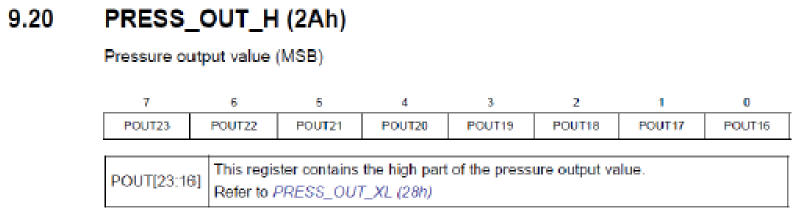


9.19 PRESS_OUT_L (29h)

Pressure output value (mid part)

7	6	5	4	3	2	1	0
POUT15	POUT14	POUT13	POUT12	POUT11	POUT10	POUT9	POUT8

POUT[15:8]	This register contains the mid part of the pressure output value. Refer to *PRESS_OUT_XL (28h)*



9.20 PRESS_OUT_H (2Ah)

Pressure output value (MSB)

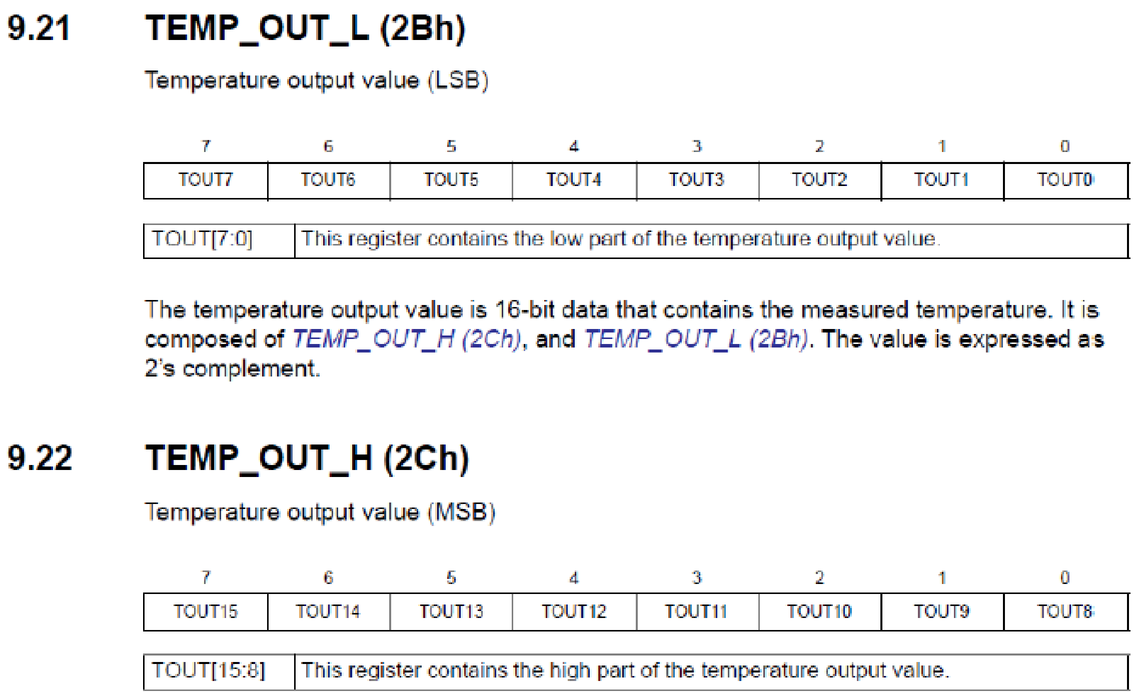
7	6	5	4	3	2	1	0
POUT23	POUT22	POUT21	POUT20	POUT19	POUT18	POUT17	POUT16

POUT[23:16]	This register contains the high part of the pressure output value. Refer to *PRESS_OUT_XL (28h)*

```
#define PRESS_OUT_XL 0x28
#define PRESS_OUT_L 0x29
#define PRESS_OUT_H 0x2A
```

NANO 33 SENSE

- Interact with sensors on the I2C bus
  - Write your own device driver for LPS22HB
    - The temperature is 16 bit value, contained in two registers



9.21 TEMP_OUT_L (2Bh)

Temperature output value (LSB)

7	6	5	4	3	2	1	0
TOUT7	TOUT6	TOUT5	TOUT4	TOUT3	TOUT2	TOUT1	TOUT0

TOUT[7:0]	This register contains the low part of the temperature output value.

The temperature output value is 16-bit data that contains the measured temperature. It is composed of *TEMP_OUT_H (2Ch)*, and *TEMP_OUT_L (2Bh)*. The value is expressed as 2's complement.

9.22 TEMP_OUT_H (2Ch)

Temperature output value (MSB)

7	6	5	4	3	2	1	0
TOUT15	TOUT14	TOUT13	TOUT12	TOUT11	TOUT10	TOUT9	TOUT8

TOUT[15:8]	This register contains the high part of the temperature output value.

```
#define TEMP_OUT_L 0x2B
#define TEMP_OUT_H 0x2C
```

NANO 33 SENSE

- Interact with sensors on the I2C bus
  - Write your own device driver for LPS22HB
    - Trigger a conversion

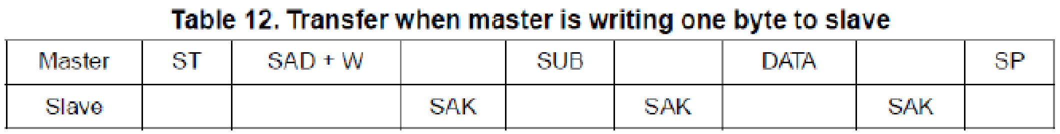
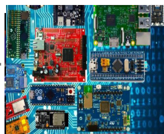


Table 12. Transfer when master is writing one byte to slave

Master	ST	SAD + W		SUB		DATA		SP
Slave			SAK		SAK		SAK	

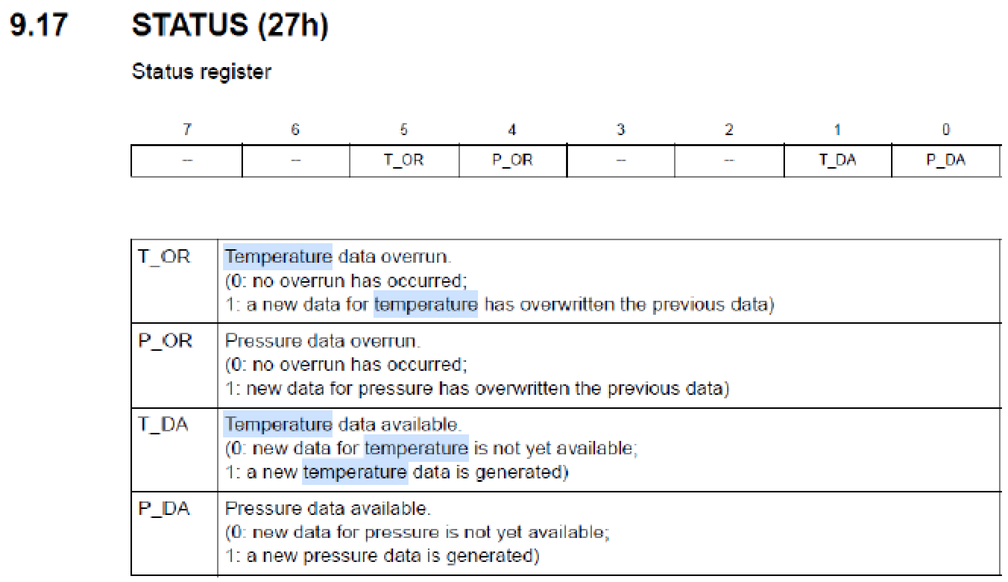
Write to the CTRL_REG2 register – set bit 0 of this register.

```
#define CTRL_REG2 0x11

void trigger(){
  Wire1.beginTransmission(LPS22HB_ADDRESS);
  Wire1.write(CTRL_REG2);
  Wire1.write(0x01);
  Wire1.endTransmission();
}
```



NANO 33 SENSE

- Interact with sensors on the I2C bus
  - Write your own device driver for LPS22HB
    - Check if data is ready, before reading



9.17 STATUS (27h)

Status register

7	6	5	4	3	2	1	0
--	--	T_OR	P_OR	--	--	T_DA	P_DA

T_OR	Temperature data overrun. (0: no overrun has occurred; 1: a new data for temperature has overwritten the previous data)
P_OR	Pressure data overrun. (0: no overrun has occurred; 1: new data for pressure has overwritten the previous data)
T_DA	Temperature data available. (0: new data for temperature is not yet available; 1: a new temperature data is generated)
P_DA	Pressure data available. (0: new data for pressure is not yet available; 1: a new pressure data is generated)

```
#define LPS22HB_STATUS 0x27

bool presReady(){
  uint8_t status = read1(LPS22HB_STATUS);
  return(status & 0x1);
}

bool tempReady(){
  uint8_t status = read1(LPS22HB_STATUS);
  return(status & 0x2);
}
```

NANO 33 SENSE

- Interact with sensors on the I2C bus
  - Write your own device driver for LPS22HB
    - Read data

```
uint32_t getPressure(){
  return read1(PRESS_OUT_XL) |
         read1(PRESS_OUT_L) << 8 |
         read1(PRESS_OUT_H) << 16;
}

uint16_t getTemperature(){
  return read1(TEMP_OUT_L) |
         read1(TEMP_OUT_H) << 8;
}
```

For pressure, we have to perform the below sequence three time, one for each byte of the pressure data.

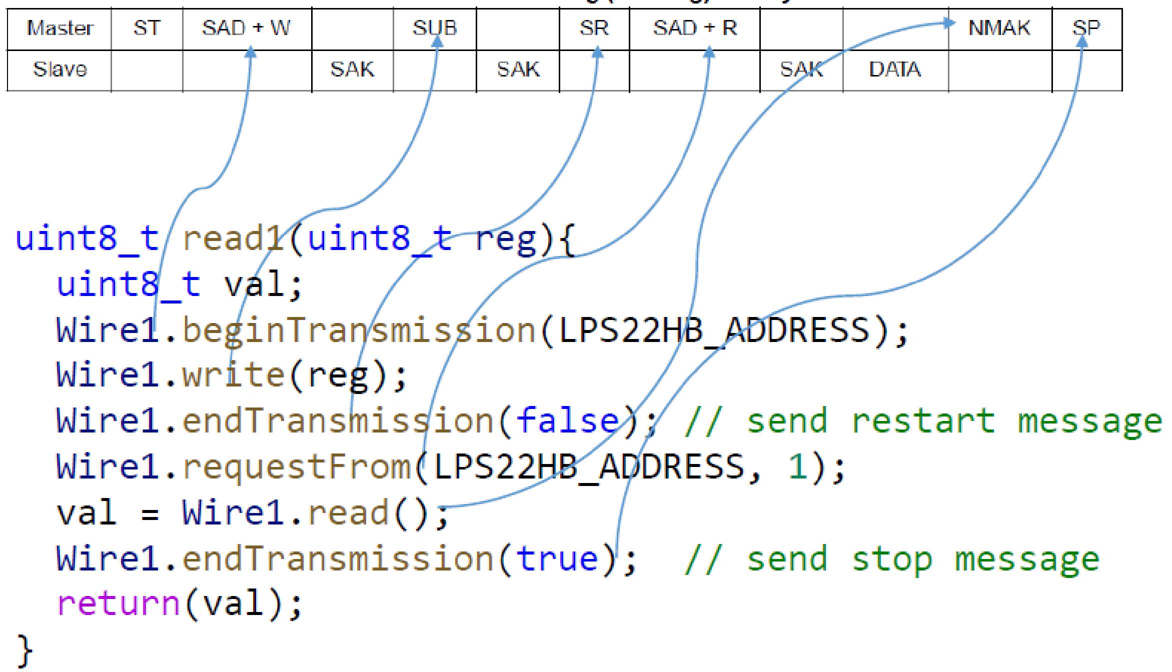


Table 14. Transfer when master is receiving (reading) one byte of data from slave

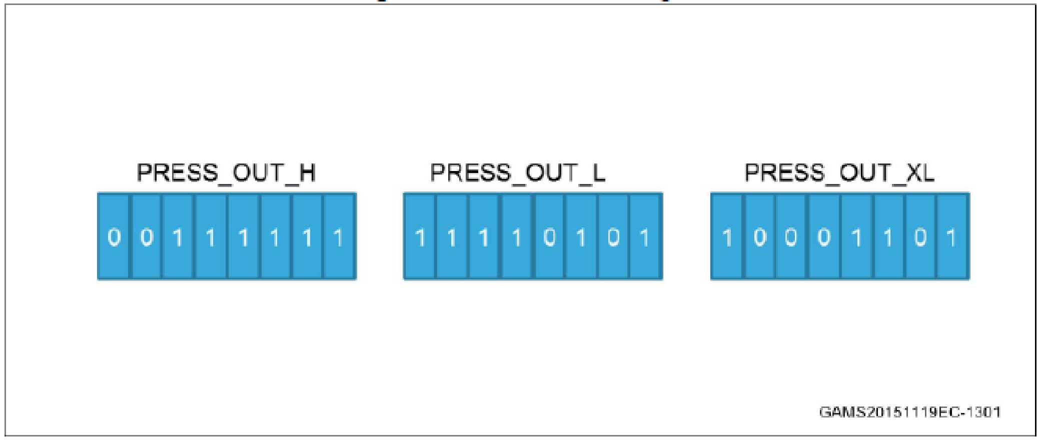
Master	ST	SAD + W		SUB		SR	SAD + R			NMAK	SP
Slave			SAK		SAK			SAK	DATA		

```
uint8_t read1(uint8_t reg){
  uint8_t val;
  Wire1.beginTransmission(LPS22HB_ADDRESS);
  Wire1.write(reg);
  Wire1.endTransmission(false); // send restart message
  Wire1.requestFrom(LPS22HB_ADDRESS, 1);
  val = Wire1.read();
  Wire1.endTransmission(true);  // send stop message
  return(val);
}
```

NANO 33 SENSE

- Interact with sensors on the I2C bus
  - Write your own device driver for LPS22HB
    - Read data

Figure 6. Pressure readings



Equation 1

Pressure Value (LSB) = PRESS_OUT_H (2Ah) & PRESS_OUT_L (29h) & PRESS_OUT_XL (28h)
= 3FF58Dh = 4191629 LSB (decimal signed)

Equation 2

$$\text{Pressure (hPa)} = \frac{\text{Pressure Value (LSB)}}{\text{Scaling Factor}} = \frac{4191629\ \text{LSB}}{4096\ \text{LSB/hPa}} = 1023.3\text{hPa}$$

```
trigger();
uint32_t raw_pres;
uint16_t raw_temp;
while(!presReady());
raw_pres = getPressure();
while(!tempReady());
raw_temp = getTemperature();
float pres_hPa = raw_pres / 4096.0;
float temp_F = raw_temp / 100.0;
```

NANO 33 SENSE

- Interact with sensors on the I2C bus
  - Write your own device driver for LPS22HB
    - Complete code:

```
#include <Wire.h>
#define LPS22HB_ADDRESS  0x5C
#define CTRL_REG2 0x11
#define LPS22HB_STATUS 0x27
#define PRESS_OUT_XL 0x28
#define PRESS_OUT_L 0x29
#define PRESS_OUT_H 0x2A
#define TEMP_OUT_L 0x2B
#define TEMP_OUT_H 0x2C
char msg[40];

void trigger(){
  Wire1.beginTransmission(LPS22HB_ADDRESS);
  Wire1.write(CTRL_REG2);
  Wire1.write(0x01);
  Wire1.endTransmission();
}

// Perform single-byte read outlined in Table 14 of
LPS22HB datasheet
uint8_t read1(uint8_t reg){
  uint8_t val;
  Wire1.beginTransmission(LPS22HB_ADDRESS);
  Wire1.write(reg);
  Wire1.endTransmission(false); // send restart message
  Wire1.requestFrom(LPS22HB_ADDRESS, 1);
  val = Wire1.read();
  Wire1.endTransmission(true);  // send stop message
  return(val);
}
```

```
bool presReady(){
  uint8_t status = read1(LPS22HB_STATUS);
  return(status & 0x1);
}

bool tempReady(){
  uint8_t status = read1(LPS22HB_STATUS);
  return(status & 0x2);
}

uint32_t getPressure(){
  return read1(PRESS_OUT_XL) | read1(PRESS_OUT_L) << 8 | read1(PRESS_OUT_H) << 16;
}

uint16_t getTemperature(){
  return read1(TEMP_OUT_L) | read1(TEMP_OUT_H) << 8;
}

void setup() {
  Wire1.begin();
  Serial.begin(9600);
}

void loop() {
  trigger();
  uint32_t raw_pres;
  uint16_t raw_temp;
  while(!presReady());
  raw_pres = getPressure();
  while(!tempReady());
  raw_temp = getTemperature();
  float pres_hPa = raw_pres / 4096.0;
  float temp_F = raw_temp / 100.0;
  sprintf(msg, "Pressure %0.1f, Temperature %0.2f\n", pres_hPa, temp_F);
  Serial.println(msg);
  delay(1000);
}
```