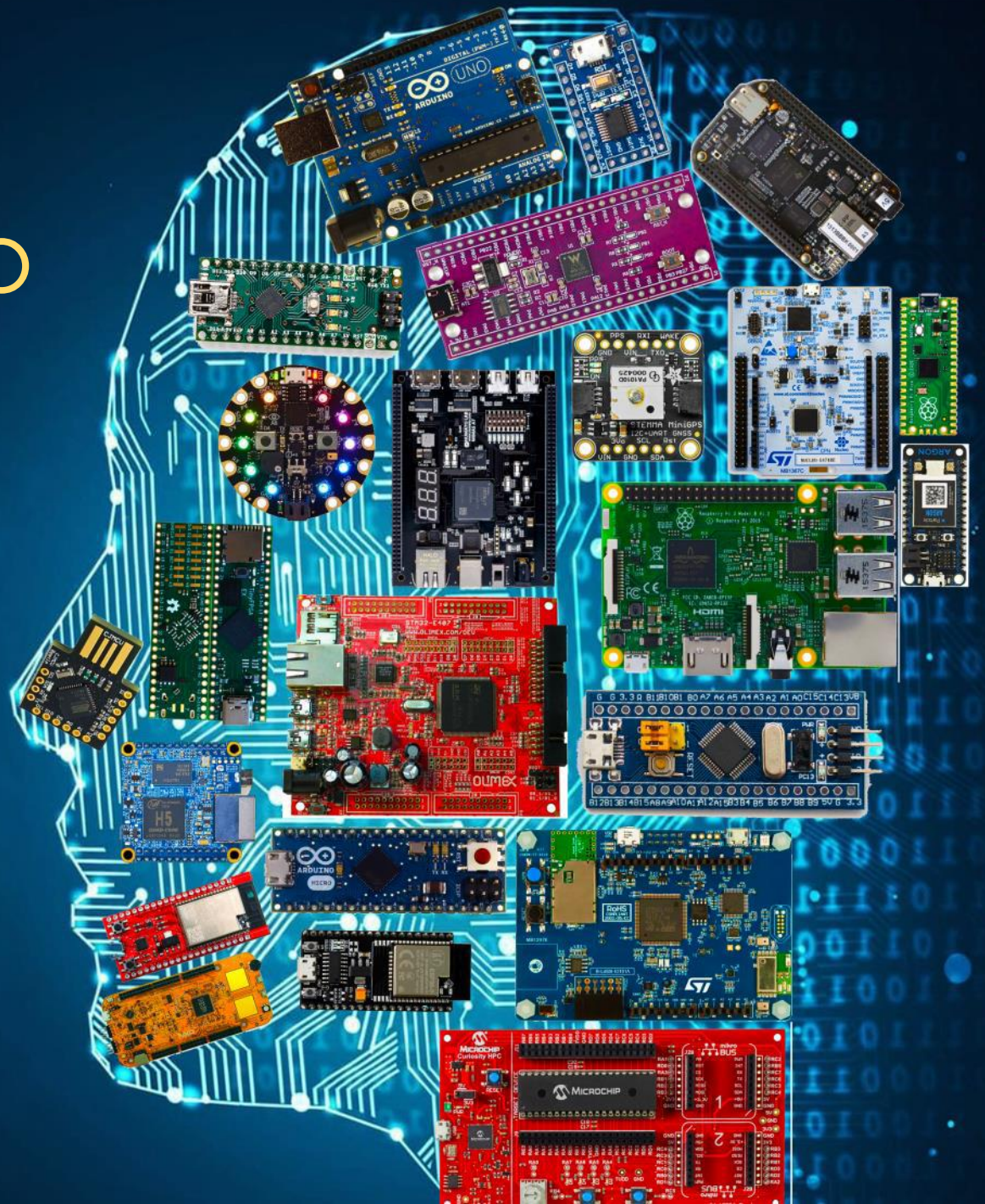
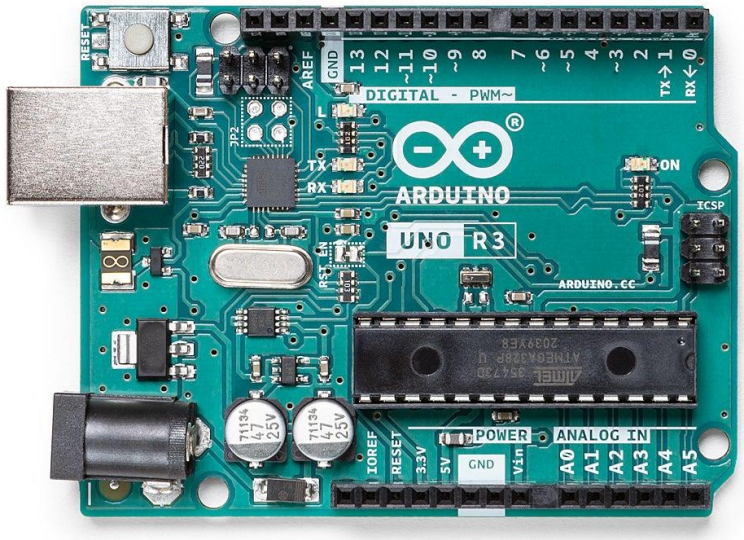
Programming Arduino and Nano 33 BLE (nRF52840)



IE 5995: IoT and Edge AI Programming
Yanchao Liu
Industrial & Systems Engineering
College of Engineering
Wayne State University

What is Arduino

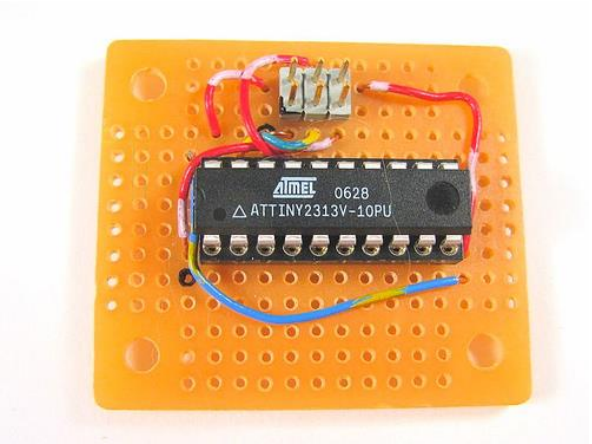


Arduino's purpose is to control things by interfacing with sensors and actuators.

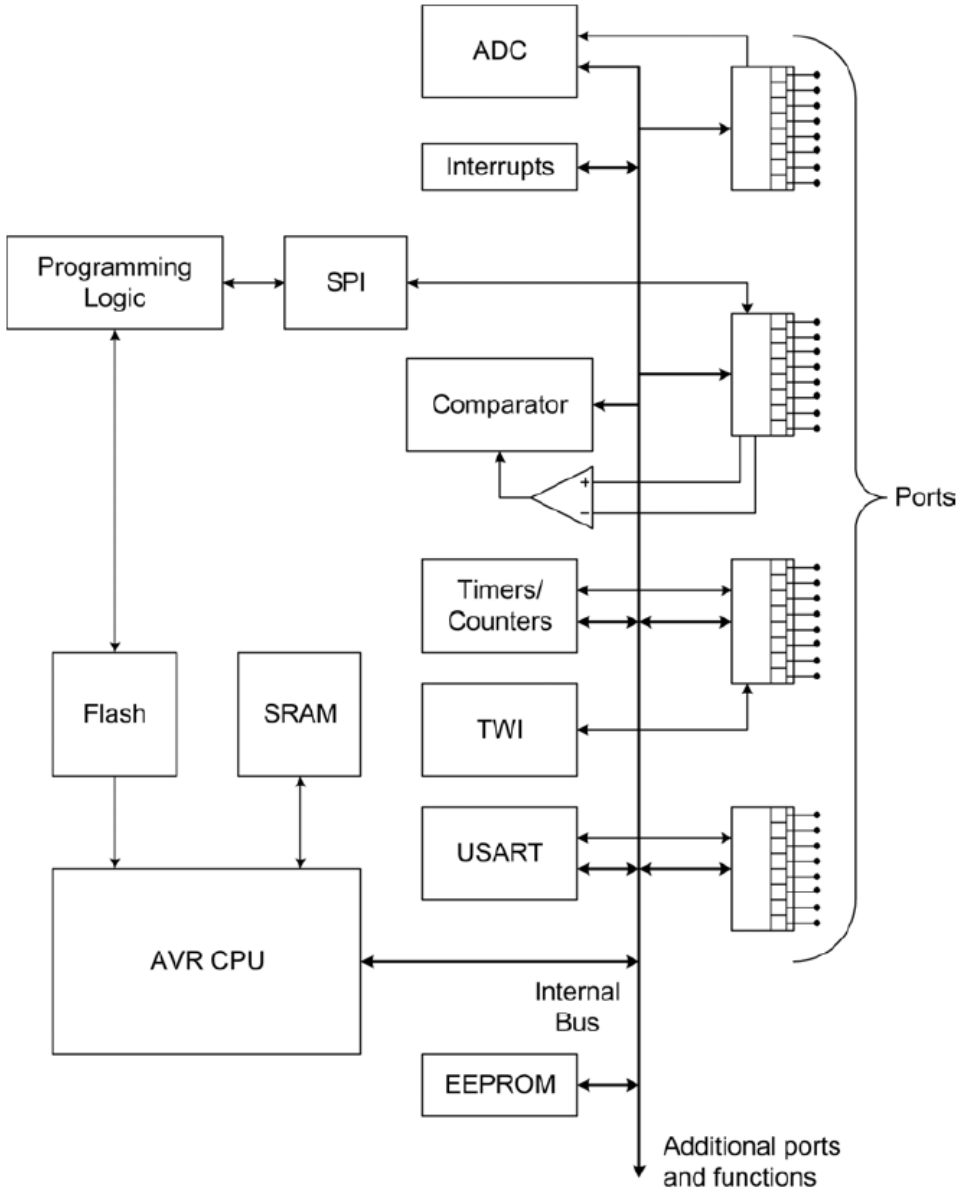
- No keyboard, mouse and screen*
- No operating system, limited memory
- A single program enjoys 100% of CPU time

* Can be attached via "shields"

- Physical Arduino boards
  - Uno
  - Nano
  - Leonardo
  - Mega
  - Pro Mini
  - etc.
- 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, etc.
  - Adafruit
  - SparkFun
  - etc.



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
- etc.

Interfacing with Arduino

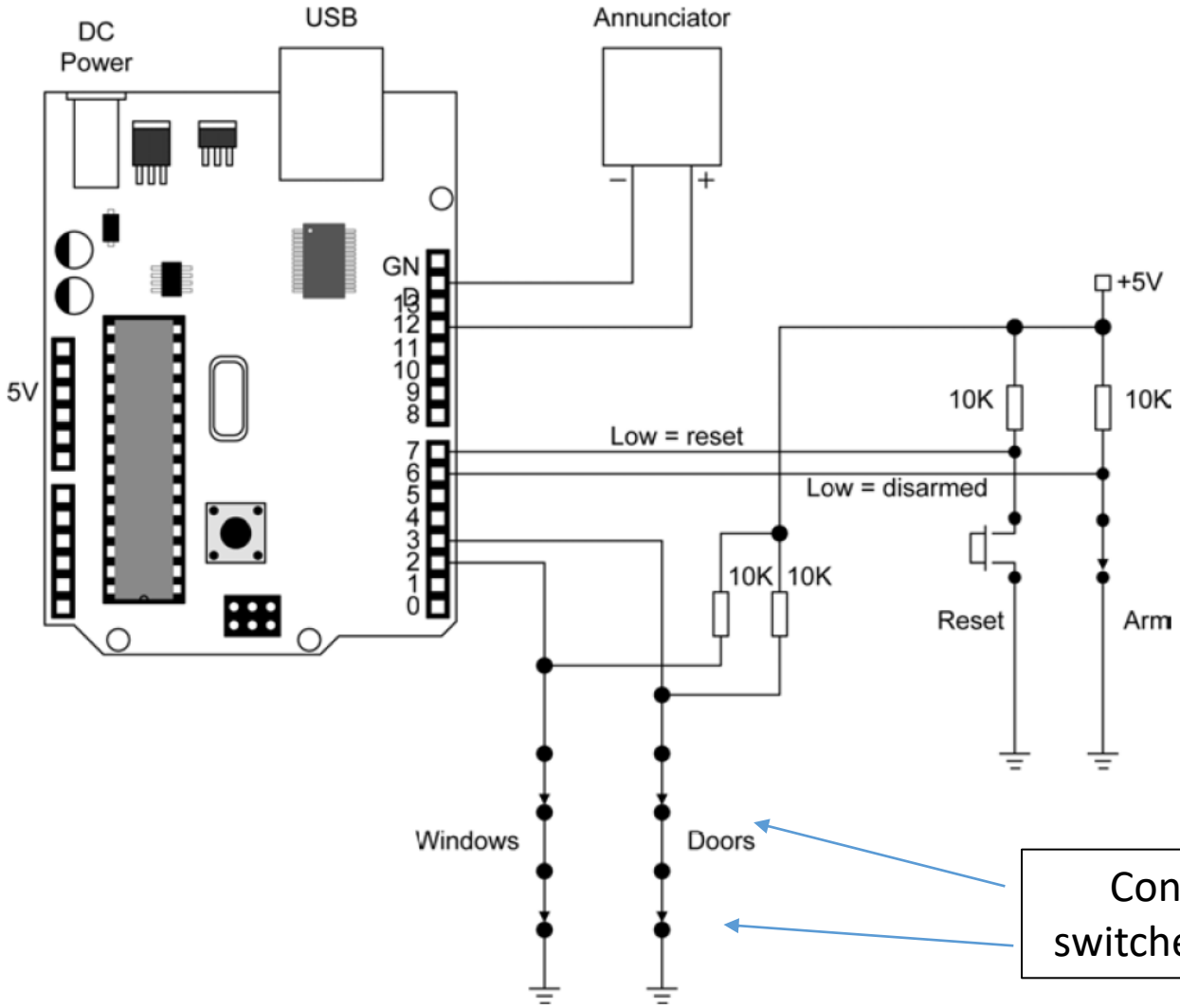
- Temperature sensor
- Pressure sensor
- Switches
- Variable resistor
- Range finder
- PIR (person-in-room) sensor
- Relay
- Motor control
- LED
- 16x2 display
- Graphic display
- Bluetooth shield
- WiFi shield
- Ethernet shield

Sensors

Reed Switch, Tilt sensor, Flame sensor, Hall effect sensor, Shock (impact) sensor, IR Proximity sensor, IR line following sensor, Conductive contact sensor, Microphone sensor, LED heartbeat sensor, Rotary encoder, Ultrasonic distance sensor, Water sensor, Touch sensor, Soil moisture sensor, Barometric sensor, Light sensor, PIR sensor, Laser transmitter/receiver, RTC module

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

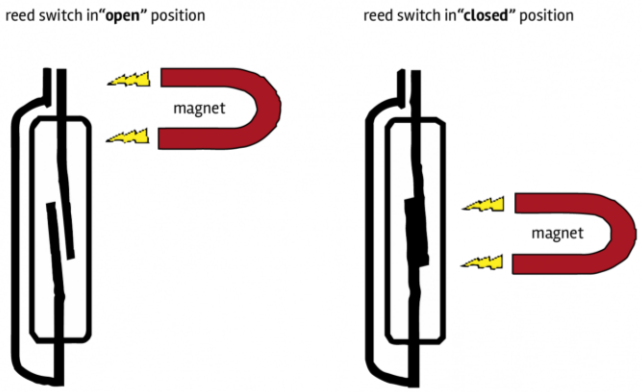
An Arduino-based Intrusion Detector



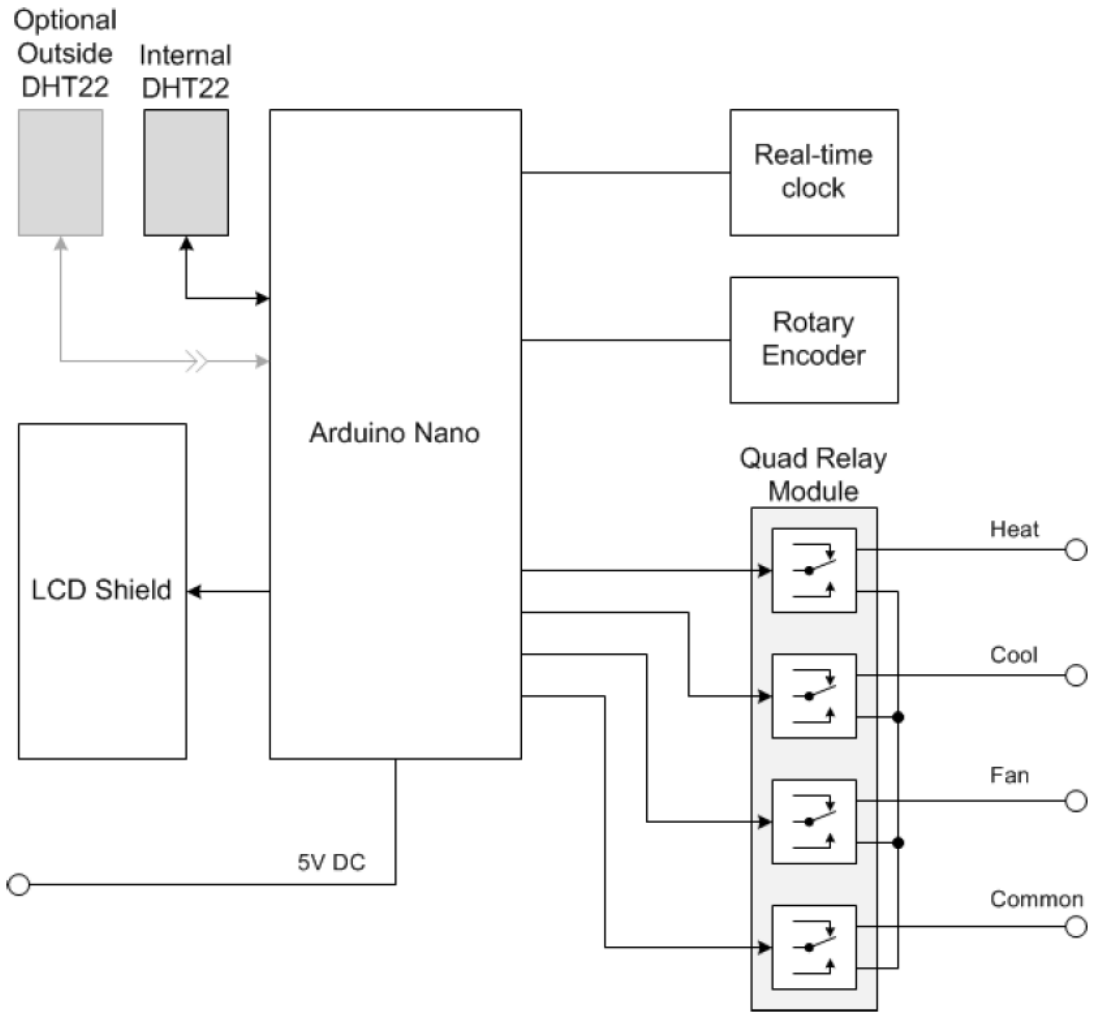
Connecting switches in series

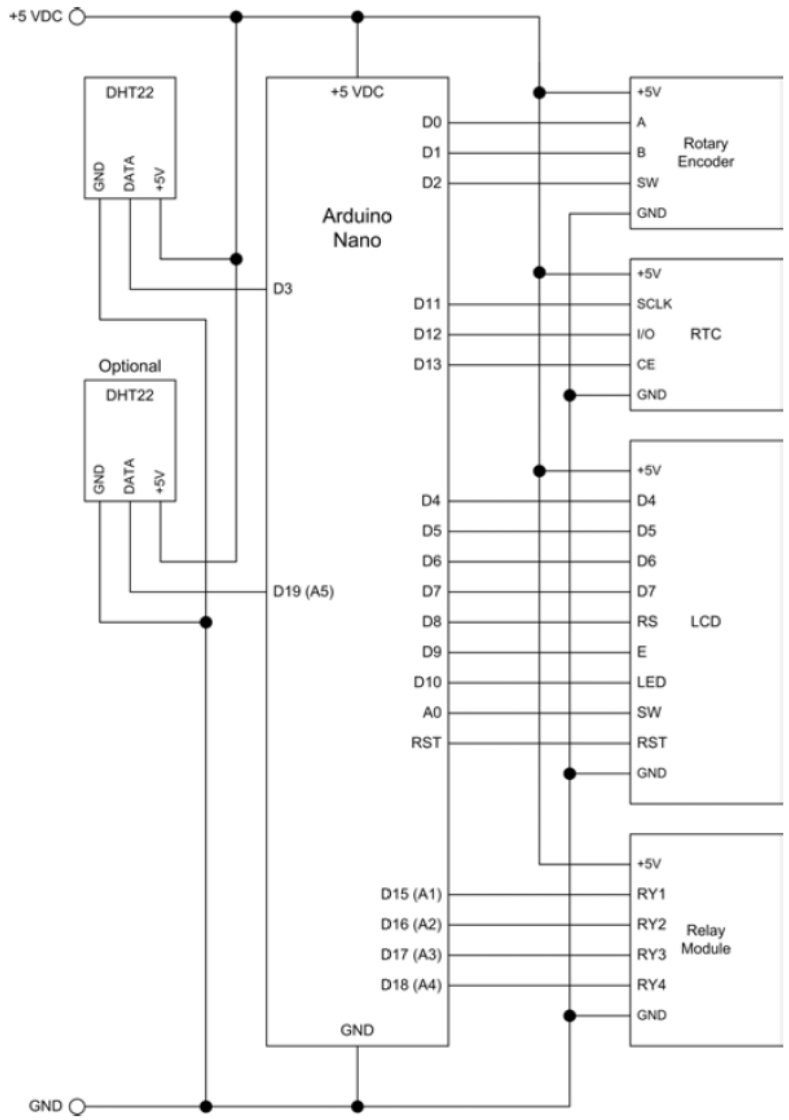
Arduino Sketch:
https://github.com/ArdNut/Miscellaneous/blob/master/SimpleAlarm/SimpleAlarm.ino

Magnetic Reed Switch

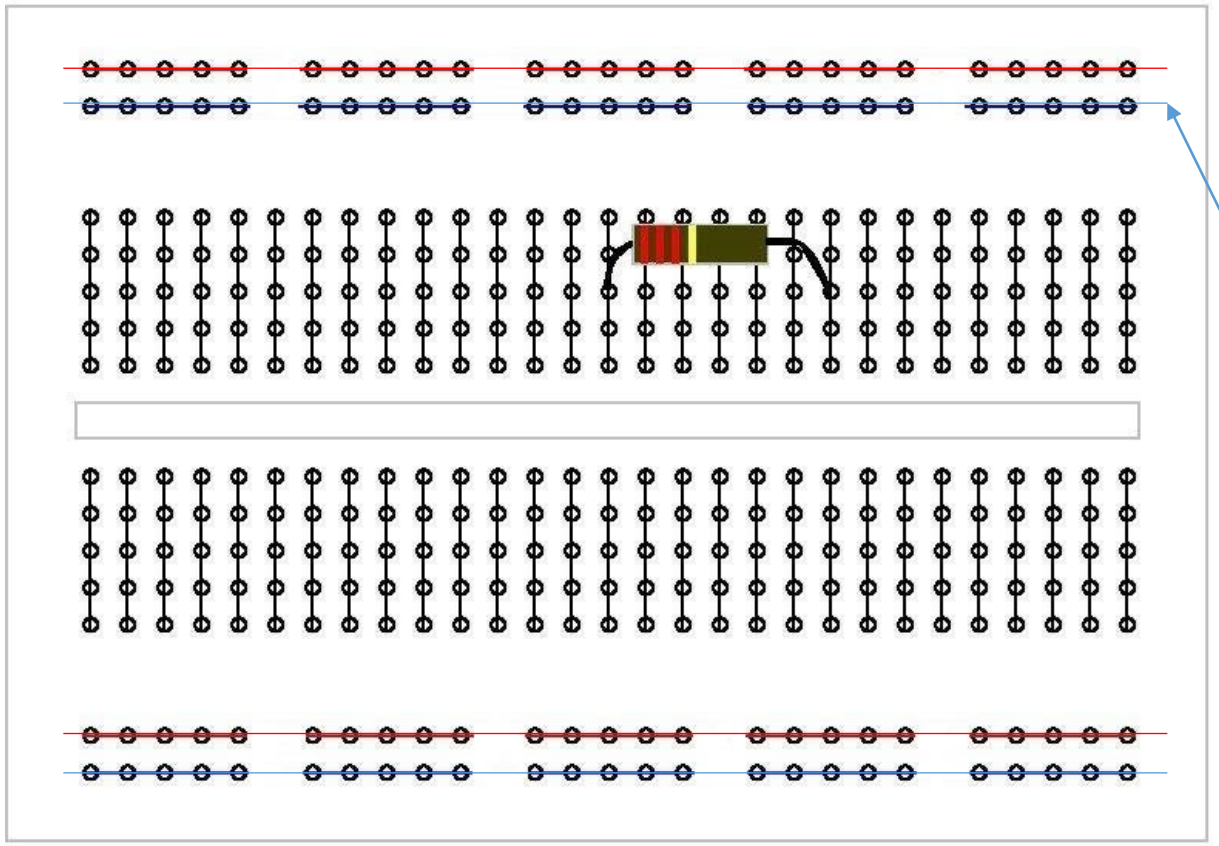


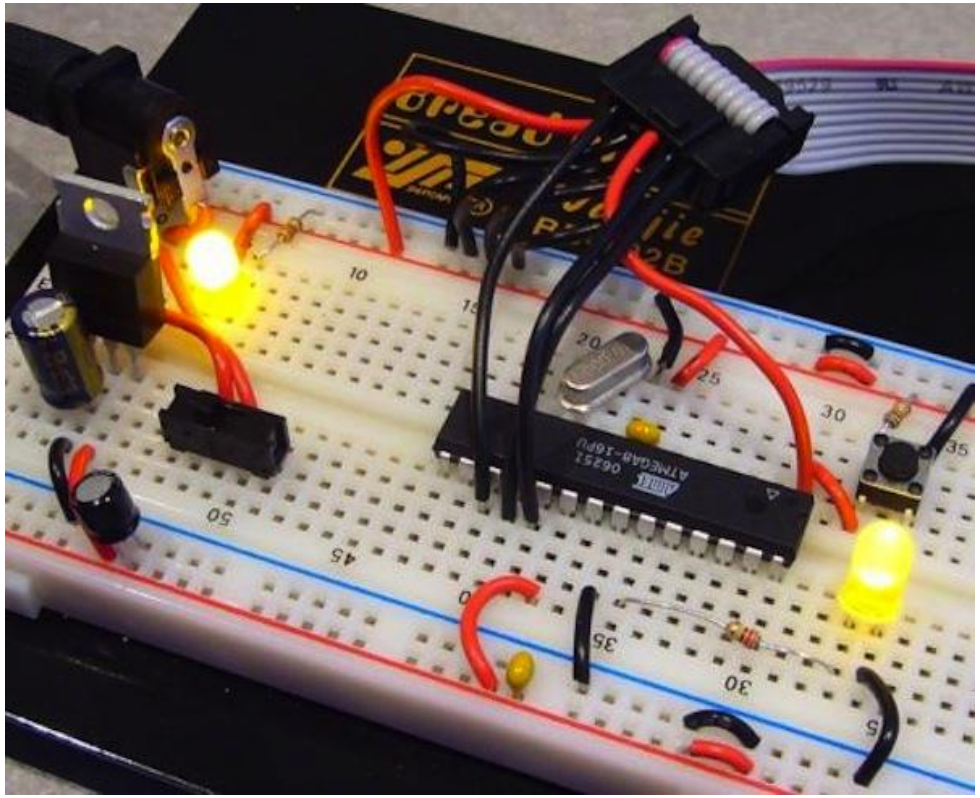
An Arduino-based Thermostat Design



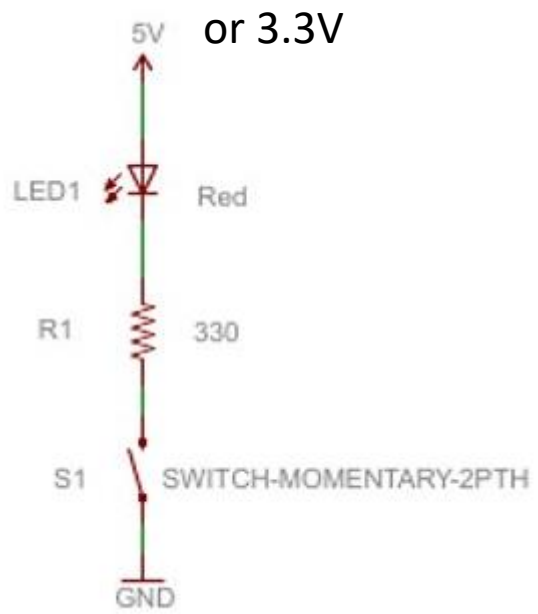


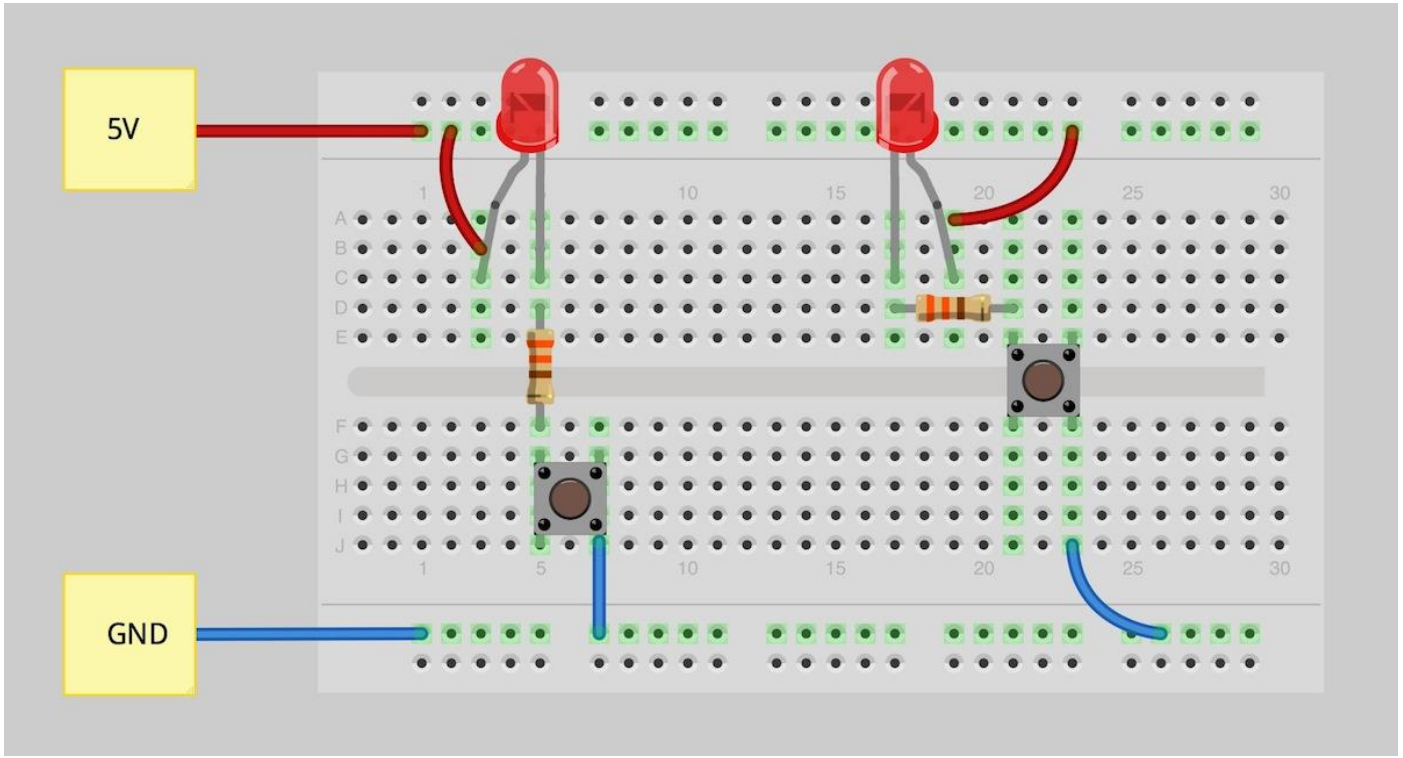
Breadboard





Build a circuit



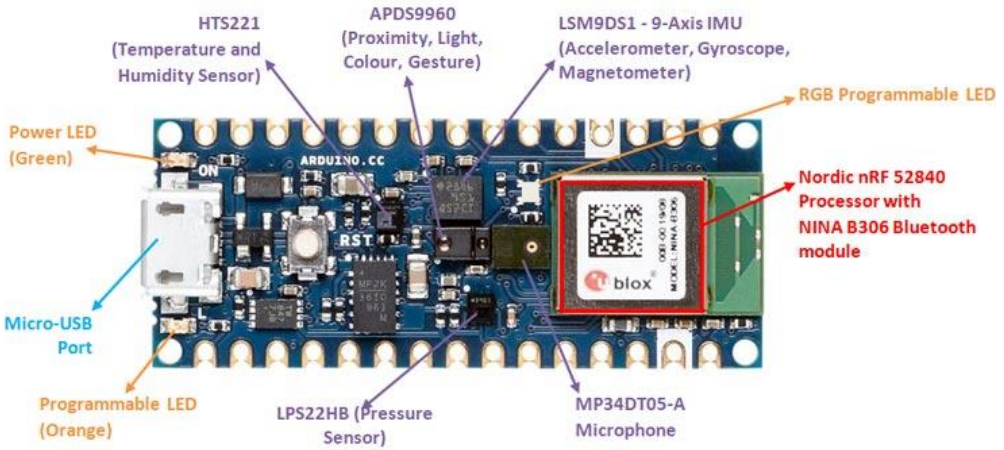


Uno vs Nano 33 BLE Sense

	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

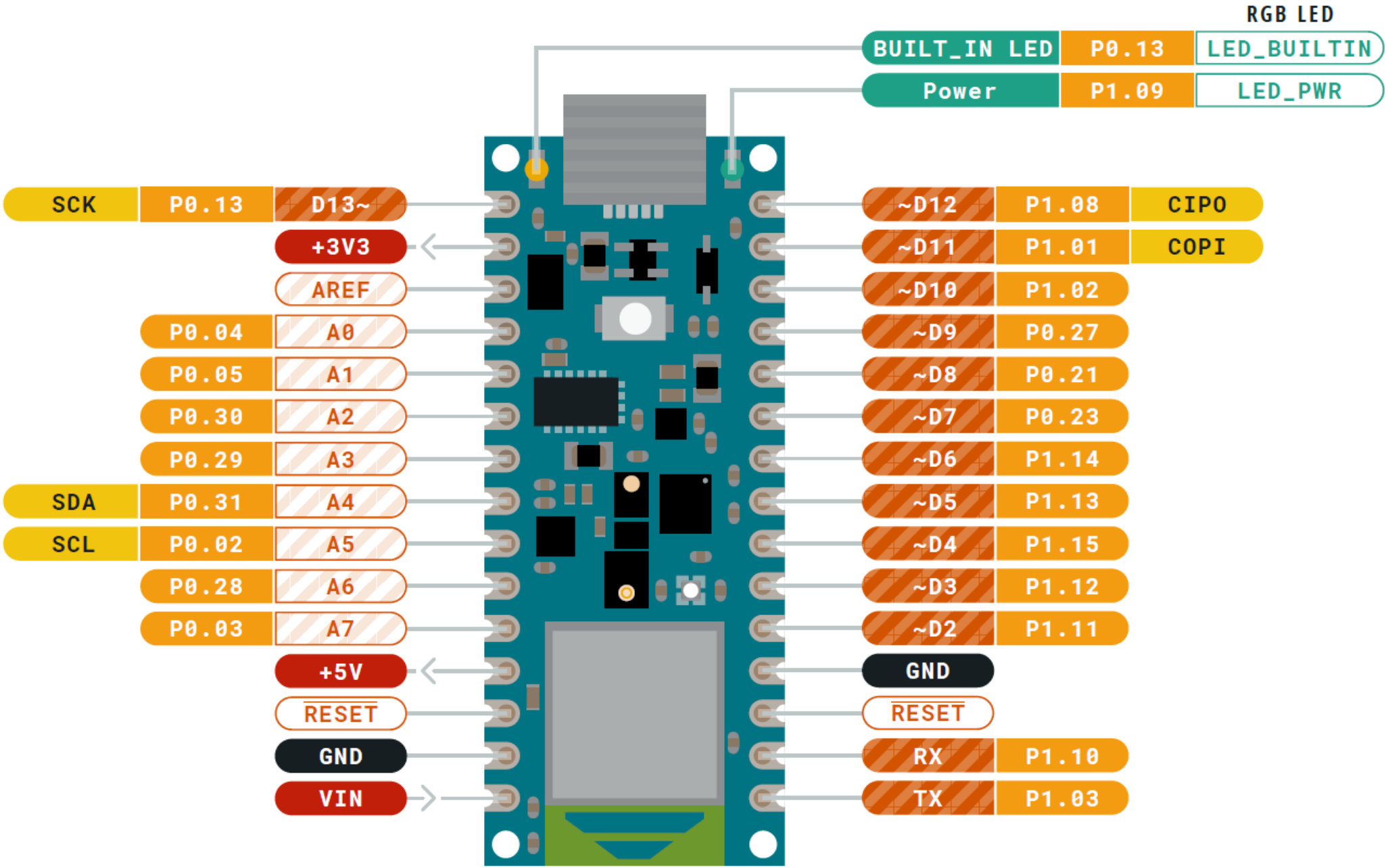
Arduino Nano 33 BLE only supports 3.3V I/Os and is NOT 5V tolerant so please make sure you are not directly connecting 5V signals to this board or it will be damaged.

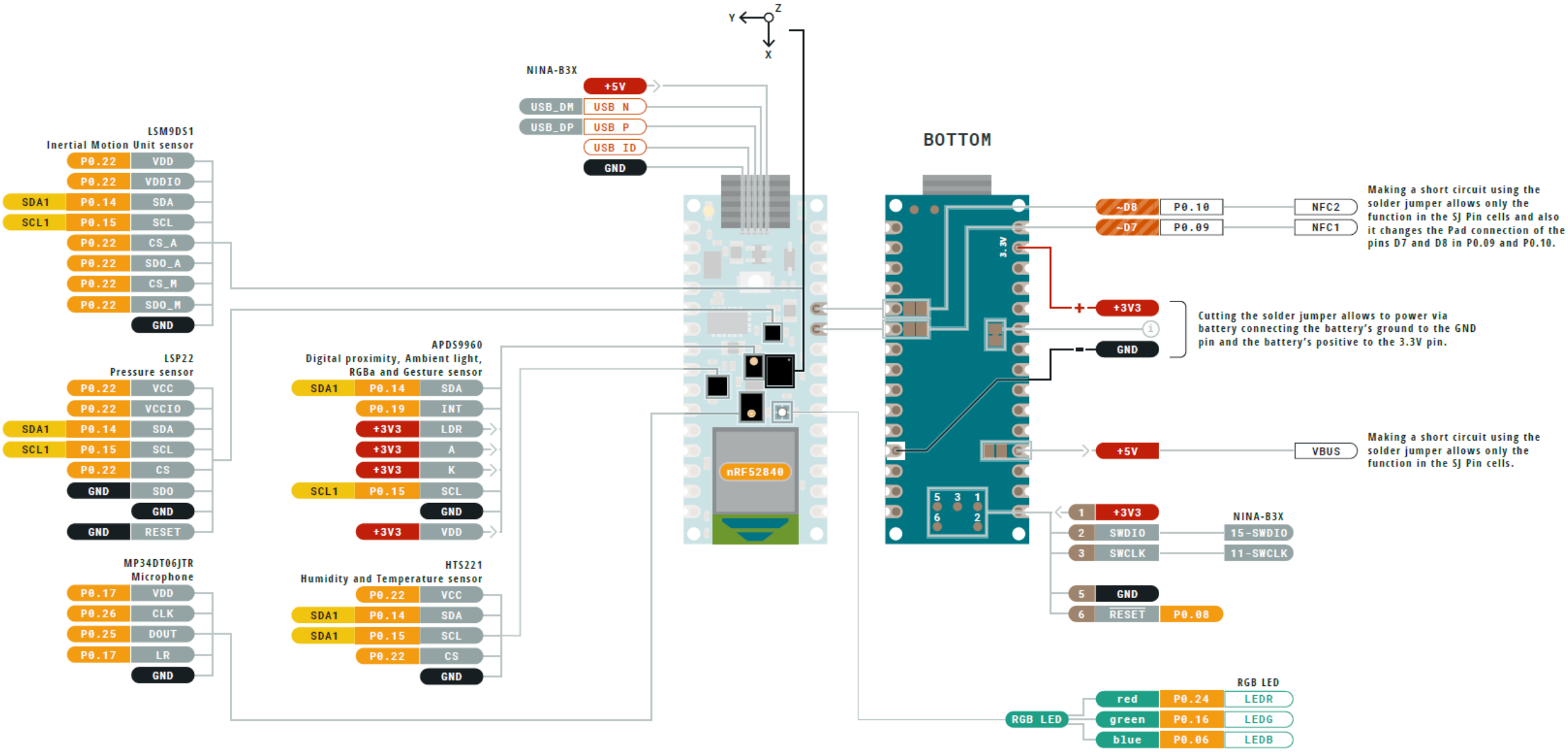
DC current per IO pin is 15 mA (max)



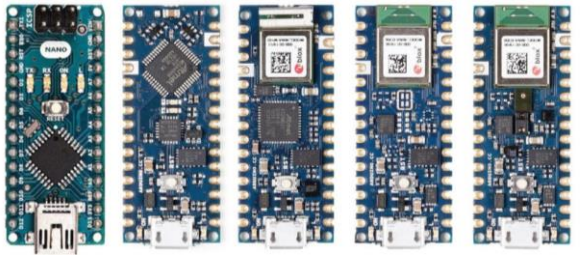
Features of Nano 33 Sense

- 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


RGB LED
BUILT_IN LED
P0.13
LED_BUILTIN
Power
P1.09
LED_PWR
SCK
P0.13
D13~
+3V3
AREF
P0.04
A0
P0.05
A1
P0.30
A2
P0.29
A3
SDA
P0.31
A4
SCL
P0.02
A5
P0.28
A6
P0.03
A7
+5V
RESET
GND
VIN
~D12
P1.08
CIPO
~D11
P1.01
COPI
~D10
P1.02
~D9
P0.27
~D8
P0.21
~D7
P0.23
~D6
P1.14
~D5
P1.13
~D4
P1.15
~D3
P1.12
~D2
P1.11
GND
RESET
RX
P1.10
TX
P1.03

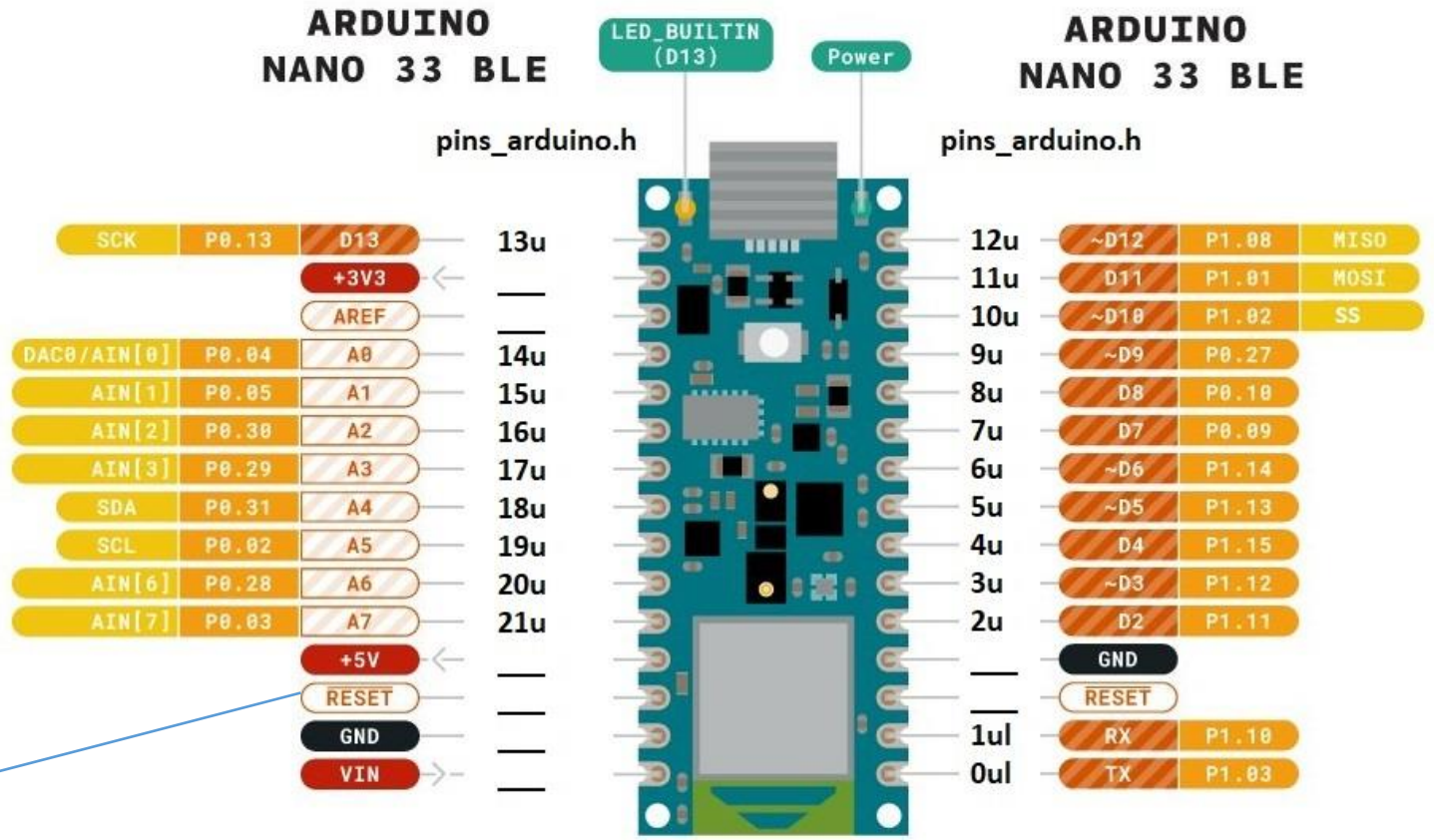

Y
Z
X
NINA-B3X
+5V
USB_DM
USB N
USB_DP
USB P
USB ID
GND
LSM9DS1
Inertial Motion Unit sensor
P0.22 VDD
P0.22 VDDIO
SDA1 P0.14 SDA
SCL1 P0.15 SCL
P0.22 CS_A
P0.22 SDO_A
P0.22 CS_M
P0.22 SDO_M
GND
LSP22
Pressure sensor
P0.22 VCC
P0.22 VCCIO
SDA1 P0.14 SDA
SCL1 P0.15 SCL
P0.22 CS
GND SDO
GND
GND RESET
MP34DT06JTR
Microphone
P0.17 VDD
P0.26 CLK
P0.25 DOUT
P0.17 LR
GND
APDS9960
Digital proximity, Ambient light,
RGBa and Gesture sensor
SDA1 P0.14 SDA
P0.19 INT
+3V3 LDR
+3V3 A
+3V3 K
SCL1 P0.15 SCL
GND
+3V3 VDD
HTS221
Humidity and Temperature sensor
P0.22 VCC
SDA1 P0.14 SDA
SDA1 P0.15 SCL
P0.22 CS
GND
nRF52840
BOTTOM
3.3V
~D8 P0.10 NFC2
~D7 P0.09 NFC1
Making a short circuit using the solder jumper allows only the function in the SJ Pin cells and also it changes the Pad connection of the pins D7 and D8 in P0.09 and P0.10.
+3V3
GND
Cutting the solder jumper allows to power via battery connecting the battery's ground to the GND pin and the battery's positive to the 3.3V pin.
+5V VBUS
Making a short circuit using the solder jumper allows only the function in the SJ Pin cells.
1 +3V3
2 SWDIO 15-SWDIO
3 SWCLK 11-SWCLK
NINA-B3X
5 GND
6 RESET P0.08
RGB LED
RGB LED
red P0.24 LEDR
green P0.16 LEDG
blue P0.06 LEDB

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
Price*	$20	$12.50	$20	$23	$33
	Amazon	Amazon	Amazon	Amazon	Amazon

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.

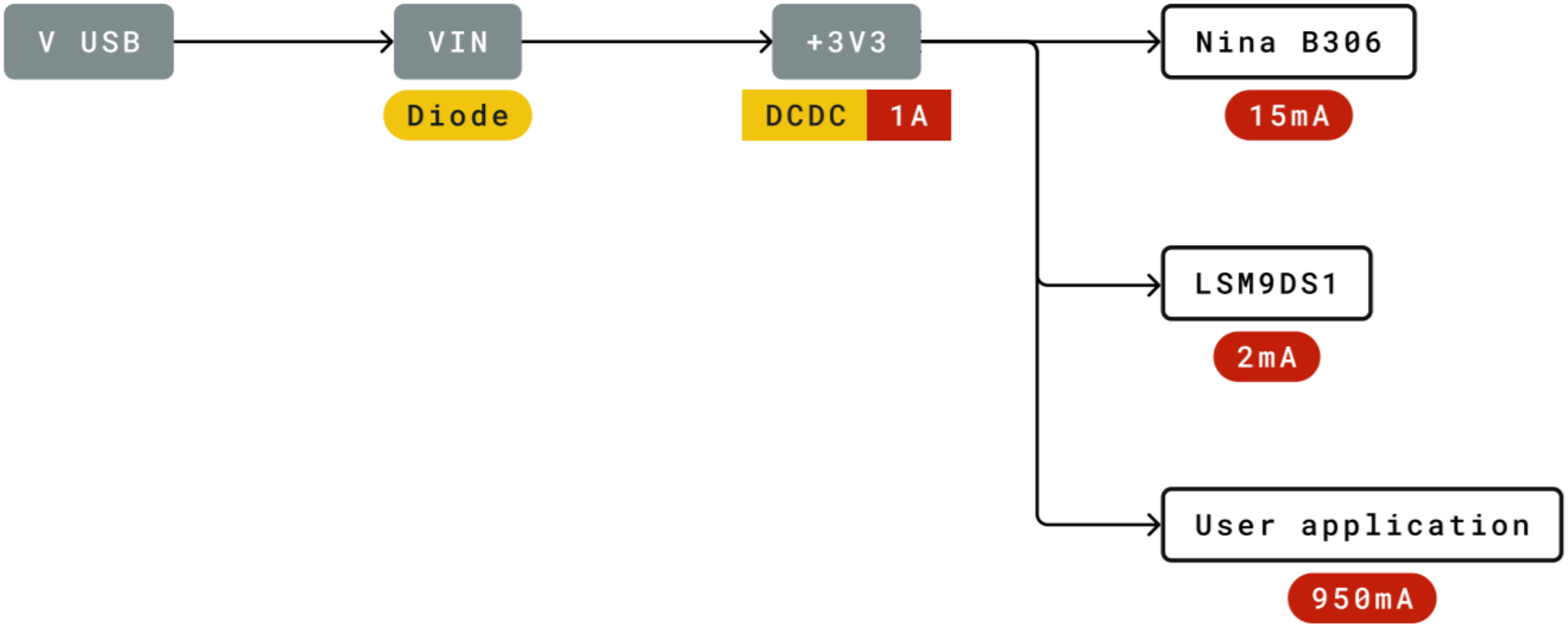
Ground the RESET pin to reset

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

https://forum.arduino.cc/t/fully-understand-pins_arduino-h-for-the-nano-33-ble/628994

Nano 33 BLE Power Tree



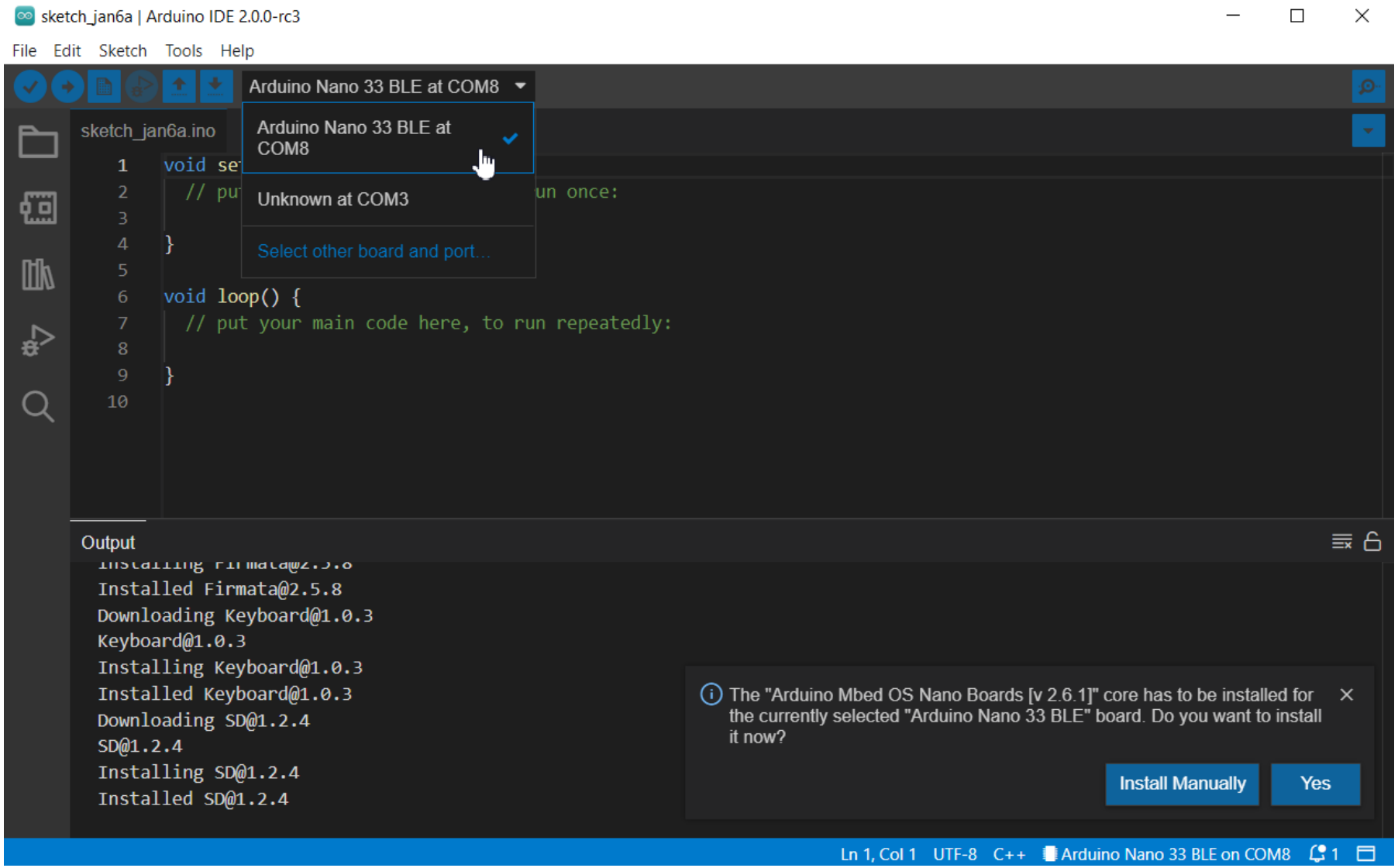
All Arduino boards have a built-in bootloader which allows flashing the board via USB. In case a sketch locks up the processor and the board is not reachable anymore via USB it is possible to enter bootloader mode by double-tapping the reset button right after power up.

nRF52840

- Arduino Nano BLE (and BLE Sense) is based on the nRF52840 microprocessor made by Nordic Semiconductor.
- 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.

Get started with the Arduino IDE

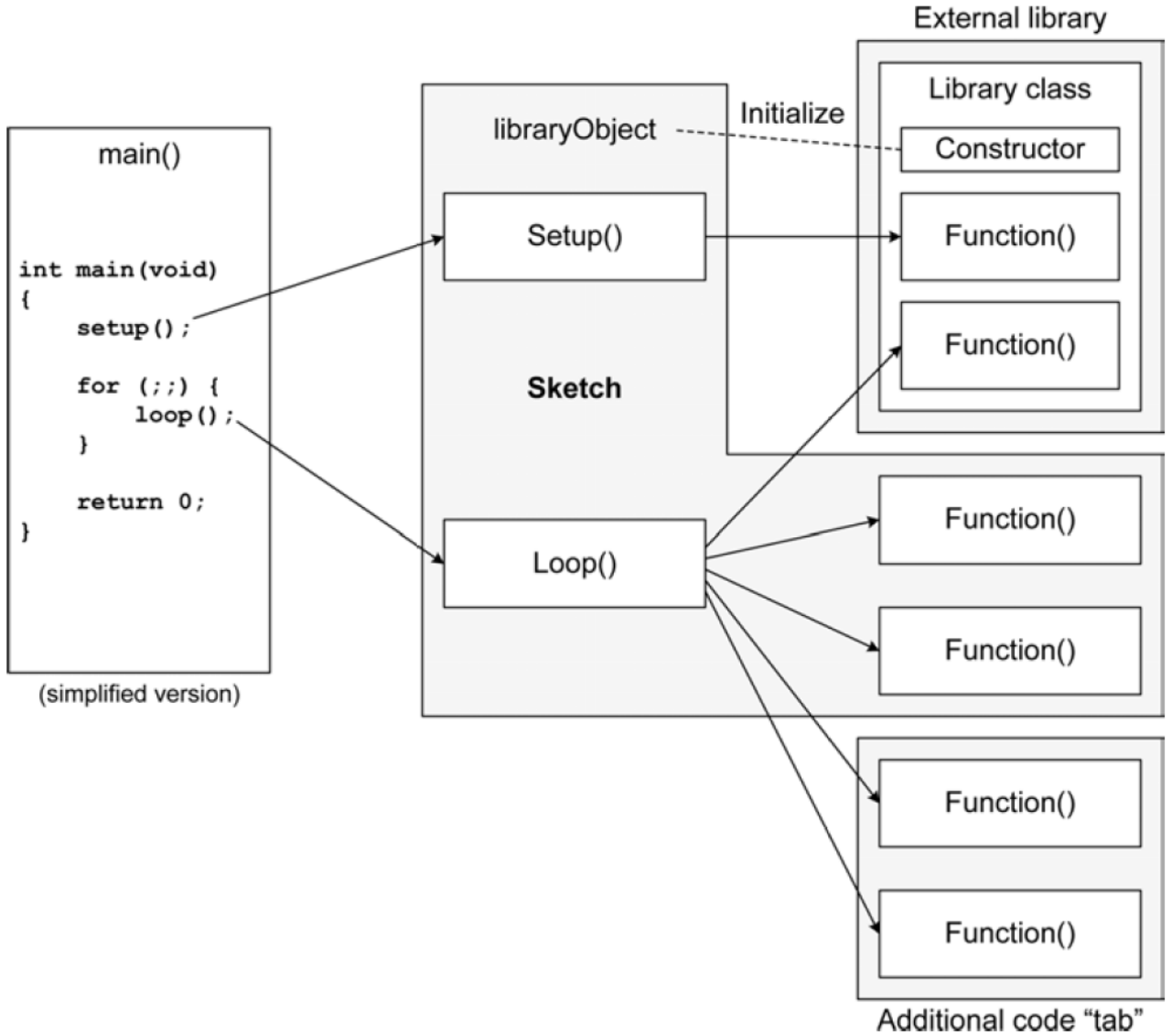
- Install Arduino IDE
- Open the IDE
- Connect the Arduino Nano 33 BLE board to PC using USB
- Select the board (port number may be different on different computers)
- Click “Yes” to install the suggested add-on



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;
}
```



Programming the Arduino (exercise)

- Connect Nano 33 to PC using USB
- Open Arduino IDE
  - Select Board
  - Select Port
- Try and modify the blink sketch in the example
  - pinMode()
  - digitalWrite()
  - digitalRead()
  - delay()
  - Serial.begin(), .println(), .print(), .available()
  - random()
  - analogRead()
  - analogWrite() (actually, PWM)
- Use Serial Monitor

Try these examples

Example 1

```
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(500);
  digitalWrite(13, LOW);
  delay(500);
  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.

Example 2 (use pin 4 to drive a LED)

```
const byte ledPin = 4;
unsigned long on_counter, off_counter;
void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  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(500);
}

```

Note: standard C does not have syntax for binary literal, the “0b” prefix is a non-standard compiler extension. In Arduino, the prefix “B” can also be used to denote binary literals (see Binary.h)

Read from Serial

```
void setup(){
  Serial.begin(9600);
}
void loop(){
  if(Serial.available() > 0){
    byte val = Serial.read();
    Serial.print("Received: ");
    Serial.println(val);
  }
}
```

```
void setup(){
  Serial.begin(9600);
}
void loop(){
  if(Serial.available() > 0){
    int val = Serial.parseInt();
    Serial.print("Received: ");
    Serial.println(val);
  }
}
```

Exercise: Create a program that generates ten, one at a time, two-number addition questions, e.g., 4+7=? After printing out each question, wait for the user (e.g., a kindergartener) to input an answer, then present the next question. After 10 questions are answered, print out the number of correct answers and the total time taken to complete the 10-question test.

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

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)

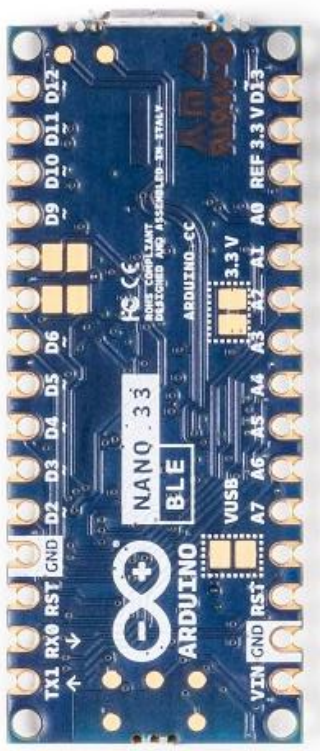
On Arduino Nano 33 BLE (Sense)

```
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)
```

Arduino Pin Names

```
// 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
```

C:\Users\gn0061\AppData\Local\Arduino15\packages\arduino\hardware\mbed_nano\2.6.1\variants\ARDUINO_NANO33BLE\pins_arduino.h



- 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

```
// 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);
```

Try the examples

```
const byte analogPin = A2;
char msg[30];
unsigned long on_counter, off_counter;
void setup() {
  Serial.begin(9600);
}

void loop() {
  int val = analogRead(analogPin);
  float volt = val*3.3/1023;
  sprintf(msg, "ADC: %d Volt: %0.2f", val, volt);
  Serial.println(msg);
  delay(500);
}

```

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.

Also, try changing the expression val*3.3/1023 to val/1023*3.3, see what happens. Why?

```
#define BUFLEN 8192
int buf[BUFLEN];
unsigned long start_time, duration;
void setup() {
  Serial.begin(9600);
}

void loop() {
  start_time = micros();
  for(int i=0; i<BUFLEN; i++){
    buf[i] = analogRead(A3);
  }
  duration = micros() - start_time;
  Serial.println(duration);
  delay(500);
}

```

- How much time (in microseconds) does each invocation of analogRead() take?
- Take 30 samples (of duration) and construct a confidence interval for the mean time of the analogRead() invocation.

Integer vs Floating-point arithmetic

```
#define BUFLEN 8192
int val;
float buf[BUFLEN];
unsigned long start_time, duration;
void setup() {
 Serial.begin(9600);
}
void loop() {
 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(500);
}
```

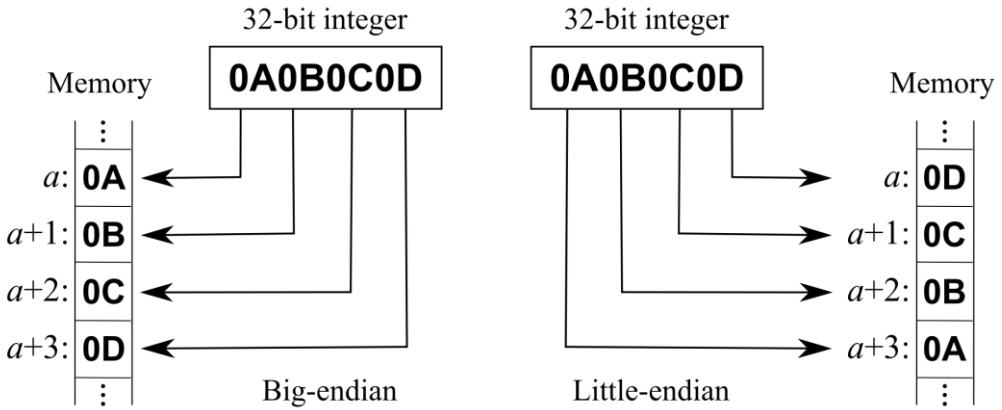
```
#define BUFLEN 8192
int val;
unsigned long buf[BUFLEN];
unsigned long start_time, duration;
void setup() {
 Serial.begin(9600);
}
void loop() {
 start_time = micros();
 for(int i=0; i<BUFLEN; i++){
  val = analogRead(A3);
    buf[i] = val*3300000/1023;  // micro-volt
 }
 duration = micros() - start_time;
 Serial.println(duration);
 delay(500);
}
```

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
```

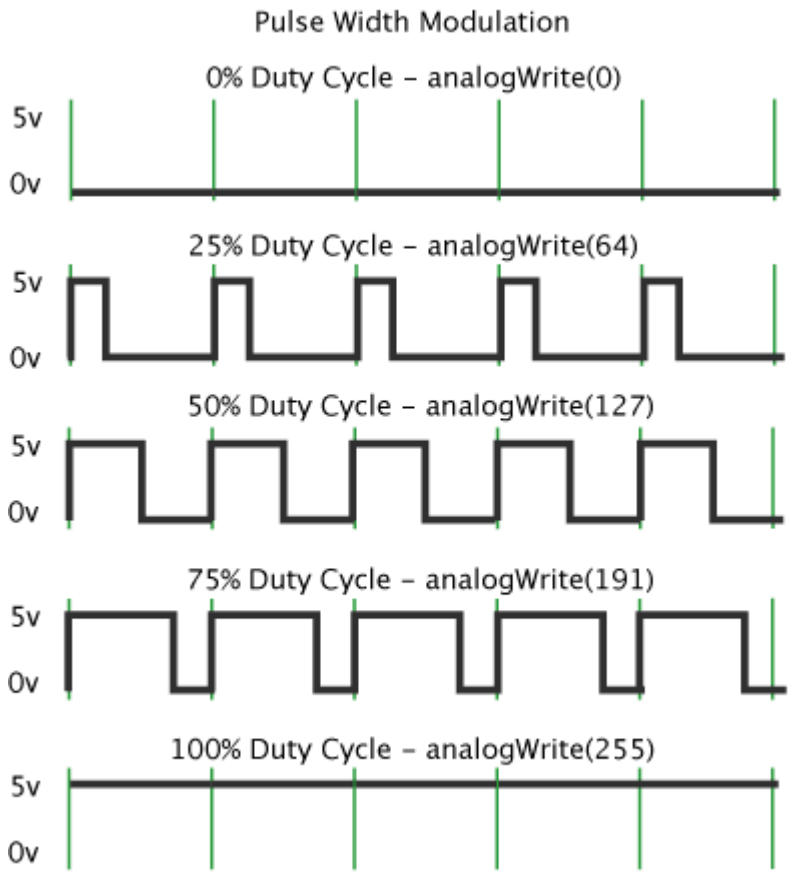
Endianness and byte addressing



- 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

```c
#include <stdio.h>
#include <stdint.h>
int main()
{
    uint32_t a = 0xF86A81F3;
    uint8_t * p = (uint8_t*)&a;
    printf("%x %u\n", a, a);
    printf("%x %x %x %x\n", *p, *(p+1), *(p+2), *(p+3));
    printf("%x\n", (uint8_t)a);
    a >>= 8;
    printf("%x %u\n", a, a);
    printf("%x %x %x %x\n", *p, *(p+1), *(p+2), *(p+3));
}
```

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.

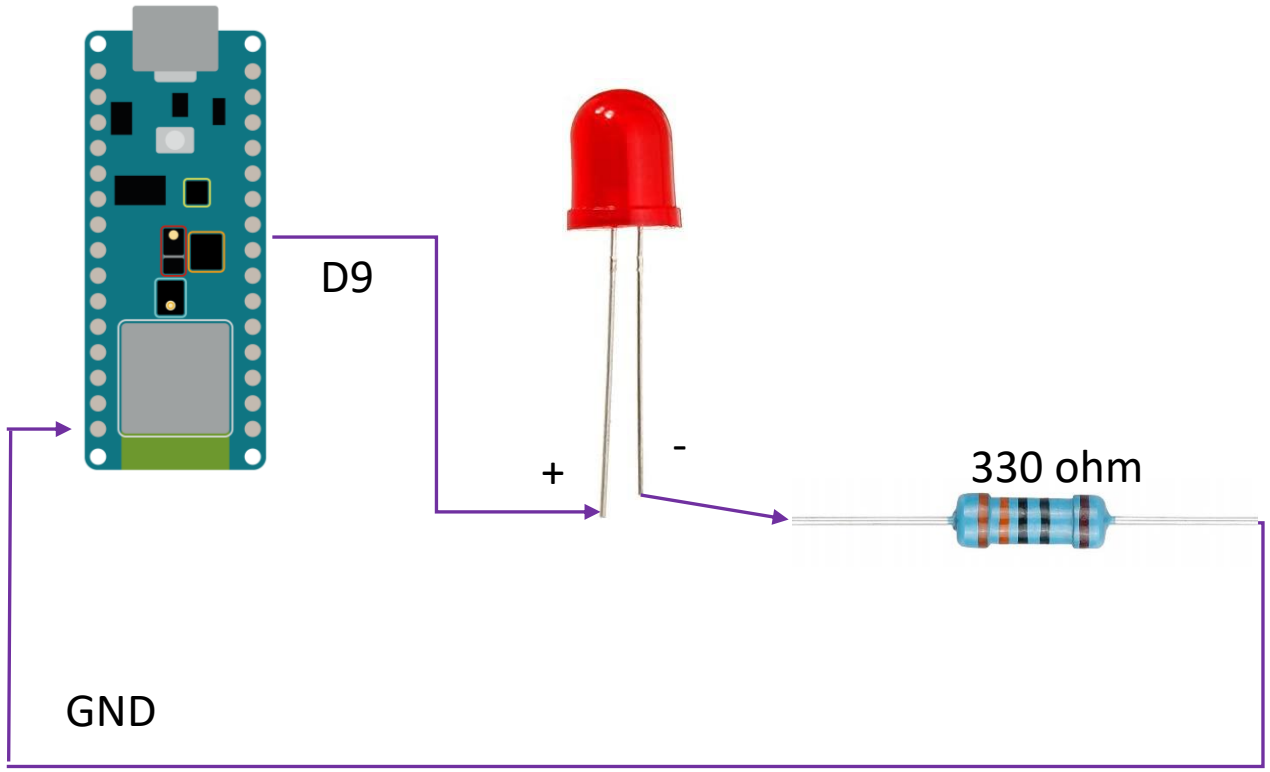
https://docs.arduino.cc/learn/microcontrollers/analog-output

PWM Frequency and Pins

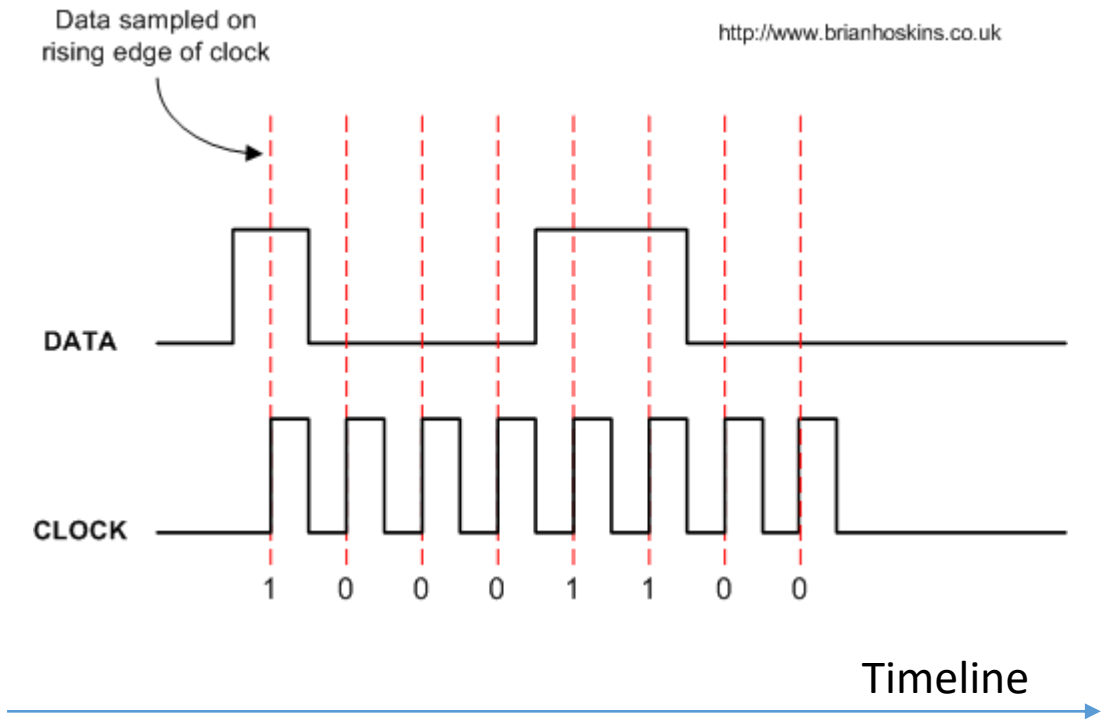
BOARD	PWM PINS	PWM FREQUENCY
Uno, Nano, Mini	3, 5, 6, 9, 10, 11	490 Hz (pins 5 and 6: 980 Hz)
Mega	2 - 13, 44 - 46	490 Hz (pins 4 and 13: 980 Hz)
Leonardo, Micro, Yún	3, 5, 6, 9, 10, 11, 13	490 Hz (pins 3 and 11: 980 Hz)
Uno WiFi Rev2, Nano Every	3, 5, 6, 9, 10	976 Hz
MKR boards *	0 - 8, 10, A3, A4	732 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

Try the example (use PWM to fade LED)

- Examples -> 01. Basics -> Fade



Two-wire serial communication toy example



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

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);
}
```

Sending number 42, 8-bit binary representation: 00101010

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);
}
```

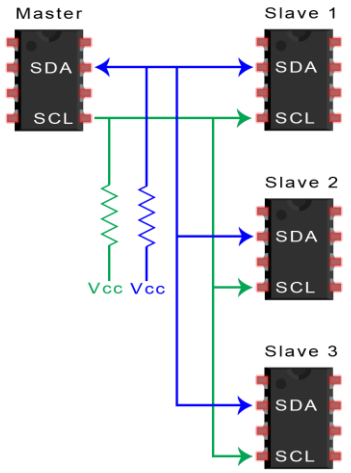
- Upload the Tx code to the sender board, upload Rx code to the receiver board
  - Make sure two boards have the same I/O voltage
  - e.g., don't connect Uno (5v) with Nano (3.3v)
- Connect the sender and receiver
  - sender's dataPin -> receiver's dataPin
  - sender's clockPin -> receiver's clockPin
- 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

Exercise

- Design and implement a one-wire communication protocol, i.e., the two devices are connected by only one line
- Write both the sender and receiver programs and demonstrate the how it works
- Encrypt / decrypt the messages being sent / received, e.g., using the AES algorithm

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).
- 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.

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.

I2C Library

- On Arduino, the Wire library provides functions for I2C communication.

Functions
begin()
end()
requestFrom()
beginTransmission()
endTransmission()
write()
available()
read()
setClock()
onReceive()
onRequest()
setWireTimeout()
clearWireTimeoutFlag()
getWireTimeoutFlag()

I2C experiments

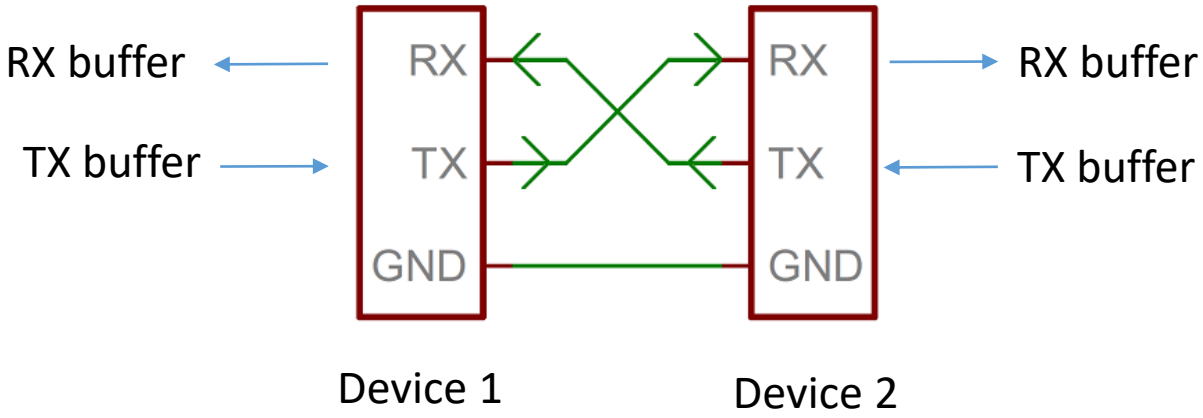
- Connect two Arduino Nano 33 BLE boards via I2C. One acts as master and the other acts as slave.

```
// 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() {
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, led_state);
  Wire.begin(my_addr); // Initialize I2C as master
  Wire.onReceive(recv_handler);
  Wire.onRequest(request_handler);
  Serial.begin(9600);
}
void recv_handler(int nbytes){
  led_state = !led_state;
}
void request_handler(){
  Wire.write(my_addr);
}

void loop() {
  digitalWrite(LED_BUILTIN, led_state);
  delay(50);
}
```

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)
- 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

https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html
https://learn.sparkfun.com/tutorials/serial-communication/rules-of-serial

Serial Bridge

```
void setup() {
  Serial.begin(115200);
  Serial1.begin(115200);
}

void loop() {
  while(Serial.available()){
    Serial1.write(Serial.read());
  }
  while(Serial1.available()){
    Serial.write(Serial1.read());
  }
}
```

- Serial is the USB Serial Port through which the Arduino board connects to the PC
- Serial1 works through the TX / RX pins on the board
- Connect two boards and message each other
  - Board1's TX connects to Board2's RX
  - Board1's RX connects to Board2's TX
  - Board1's GND connects to Board2's GND
- **Caution: Do not connect a 5 V board with a 3.3 V board**
  - e.g., Do not connect Arduino Uno / Mega (5 V) with Arduino Nano 33 BLE or any Adafruit board (3.3 V)

Experiment: use the serial bridge to read data from a GPS module and dump it to the computer's serial monitor

Wireless Serial

- In class experiments
  - Two boards talk to each other wirelessly through the HC-12 transceivers
  - Connect Serial1 (TX / RX pins on board) to HC-12
  - Set Serial1's baud rate to 6900, the default setting on HC-12
  - The Serial Bridge code (with modified baud rate on Serial1) should work
- Now two (or multiple) MCUs can talk wirelessly, think of some project ideas leveraging this capability.

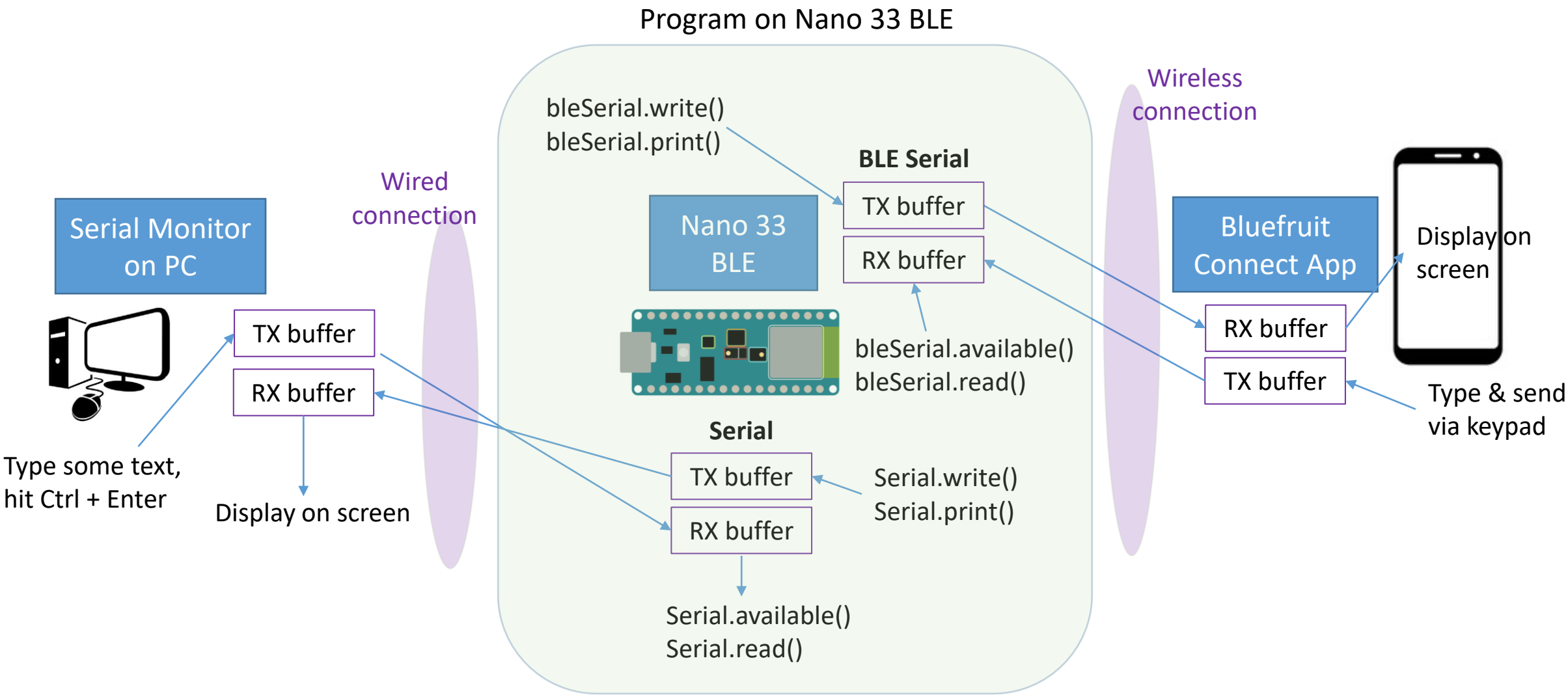
BLESerial on Nano 33 BLE

```
#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(50);
}
```

- 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

Note: if you use an Adafruit board (feather, playground, etc.) having the nRF52840 MCU, you should use the BLESerial library instead of the HardwareBLESerial library. The usage is slightly different but similar.

What's happening



Exercise (homework)

1. Make BLE Serial work between your Nano 33 BLE board (or other nRF52 board) and your smart phone's Bluefruit Connect App (or similar app)
2. Turn on/off an LED connected to the board by the phone app
3. Bring a project idea leveraging what's learned so far

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:

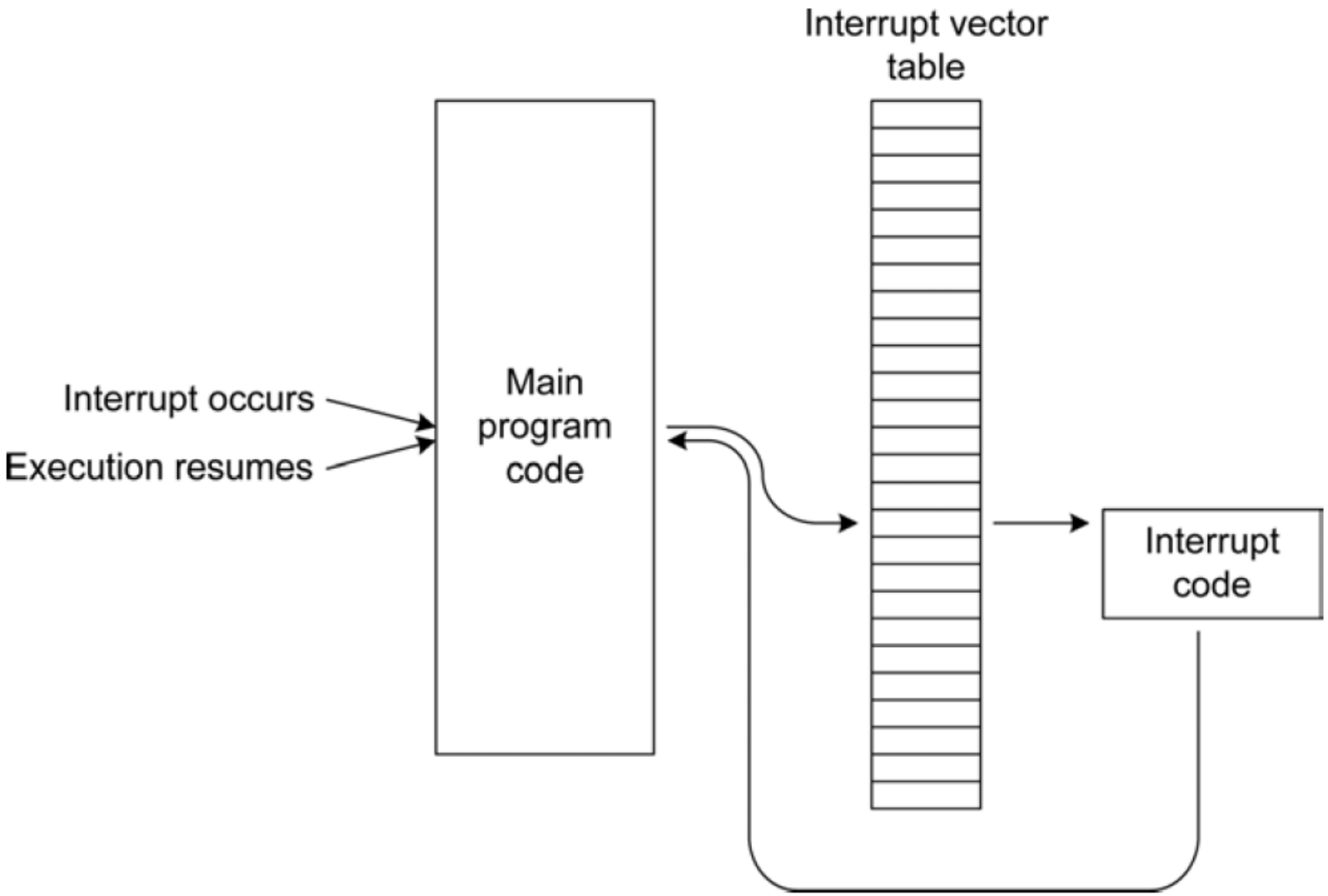
```
void setup{
   attachInterrupt(digitalPinToInterrupt(interruptPin), myISR, FALLING);
}
void loop(){}
void myISR(){
   // do something to respond to the event
}
```

Exercises:

Toggle the LED (connected to D13) whenever pin D3 goes from HIGH to LOW

- Try both the Polling and the Interrupt methods

Interrupt Handling Process



Interrupt Example

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

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**)

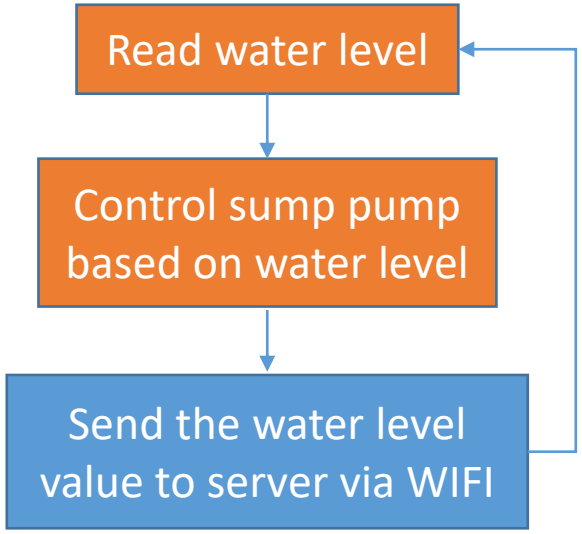
Interrupt Service Routines (ISR)

- ISR is also called callback function
  - Cannot take parameters and cannot return a value
  - Use volatile global variables to pass data between ISR and the rest of the program
- Keep ISR short and fast
  - Don't perform heavy-duty computations in ISR
  - Interrupts are automatically turned off while inside an ISR*
    - delay() and millis() will not work in an ISR; delayMicroseconds() will work
    - Serial.print() will not work because Serial communication uses interrupts
  - While ISR is running, nothing happens in the main loop until the ISR is finished
- In program code, we can explicitly turn interrupts on/off using
  - interrupts(); and noInterrupts();
  - Useful when we do not wish an area of code to be interrupted
    - e.g., when using delayMicroseconds() to generate pulses with accurate timing

This is just for Arduino. It is configured this way to keep things simple. Many processors (e.g., STM32) allow interrupt to occur within an ISR.

Timer Interrupt

- The interrupt is triggered by a hardware timer event
  - Whenever a given period of time has elapsed, an interrupt is triggered and the associated ISR is run, regardless of what the CPU is doing at the moment
- Why use timer interrupt?
  - To have the mission-critical task (which is implemented in the ISR) not blocked by ill-behaving functions / tasks in the main loop



What if the WIFI module encounter a problem connecting to the server?

Use Timer Interrupts

- Use the attachInterrupt(function, period) function from the TimerOne library
- This library only works for the AVR architecture (i.e., Atmega MCUs)
- This library is not available for nRF52840
  - User needs to write lower-level code to realize Timer-triggered Interrupts

```
#include <TimerOne.h>
void setup(){
   Timer1.initialize(500);  // 500 microseconds per trigger
   Timer1.attachInterrupt(callback);
}
void callback(){
   // Implement the ISR
}
```

Exercise: finish this code to

- blink the LED at 2 Hz
- generate a 1000 Hz square wave
- generate a 1000 Hz PWM signal at 10% duty cycle to drive the LED

Other useful methods of the Timer1 library

```
void initialize(long microseconds=1000000);
void start();
void stop();
void restart();
unsigned long read();
void setPeriod(long microseconds);
void pwm(char pin, int duty, long microseconds=-1);
void setPwmDuty(char pin, int duty);
void disablePwm(char pin);
void attachInterrupt(void (*isr)(), long microseconds=-1);
void detachInterrupt();
```

See detailed documentation here: https://playground.arduino.cc/Code/Timer1/

Timer Interrupt on Nano 33 BLE

- Install the library NRF52_MBED_TimerInterrupt
- Basic usage:

```
#include <NRF52_MBED_TimerInterrupt.h>
#include <NRF52_MBED_ISR_Timer.h>

NRF52_MBED_Timer ITimer(NRF_TIMER_3);
volatile byte led_val;
void callback(){
  led_val = !led_val;
}
void setup(){
        ITimer.attachInterruptInterval(500000, callback);
}
void loop(){
  digitalWrite(13, led_val);
}
```

Exercise: finish this code to

- blink the LED at 2 Hz
- generate a 1000 Hz square wave
- generate a 1000 Hz PWM signal at 10% duty cycle to drive the LED

https://github.com/khoih-prog/NRF52_MBED_TimerInterrupt

```ino
volatile int flag = 0;
extern "C" {
  void TIMER4_IRQHandler_v(){
    if (NRF_TIMER4->EVENTS_COMPARE[0] == 1){
        flag++;
        NRF_TIMER4->EVENTS_COMPARE[0] = 0;
    }
  }
}

void setup() {
  Serial.begin(115200);
  Serial.println("Configuring timer");
  // Refer to definitions in nrf52840.h and nrf52840_bitfield.h
  // 16-bit timer, max count is 65535
  NRF_TIMER4->BITMODE = TIMER_BITMODE_BITMODE_16Bit << TIMER_BITMODE_BITMODE_Pos;
  // Timer frequency = 16 MHz/2^PRESCALER (pp 460 of nRF52840 Product Specification v1.1)
  // So this sets the timer frequency to 16 MHz/2^4 = 1 MHz
  NRF_TIMER4->PRESCALER = 4 << TIMER_PRESCALER_PRESCALER_Pos;
  // Sets the capture compare value, cannot exceed 65536
  // 20000 counts will take 20 ms
  NRF_TIMER4->CC[0] = 20000;
  // Enable interrupt for event Compare[0], see pp 464
  NRF_TIMER4->INTENSET = TIMER_INTENSET_COMPARE0_Enabled << TIMER_INTENSET_COMPARE0_Pos;
  // Sets the shortcuts between event Compare[0] and task CLEAR (i.e., clear time)
  // That is reset the timer count to 0 when the event occurs
  NRF_TIMER4->SHORTS = TIMER_SHORTS_COMPARE0_CLEAR_Enabled << TIMER_SHORTS_COMPARE0_CLEAR_Pos;
  // Function is defined in core_cm4.h
  // TIMER4_IRQn is a nrf52840 Specific Interrupt Number, defined in nrf52840.h
  // Enables a device specific interrupt in the NVIC interrupt controller.
  NVIC_EnableIRQ(TIMER4_IRQn);
  // Start the timer, pp 461
  NRF_TIMER4->TASKS_START = 1;
}

void loop() {
  Serial.println(flag);
  delay(500);
}
```

nRF52840 Timer Interrupt Example

- Uses nRF MDK

C:\Users\gn0061\AppData\Local\Arduino15\packages\arduino\hardware\mbed_nano\2.6.1\cores\arduino\mbed\targets\TARGET_NORDIC\TARGET_NRF5x\TARGET_SDK_15_0\modules\nrfx\mdk\nrf52840.h

C:\Users\gn0061\AppData\Local\Arduino15\packages\arduino\hardware\mbed_nano\2.6.1\cores\arduino\mbed\targets\TARGET_NORDIC\TARGET_NRF5x\TARGET_SDK_15_0\modules\nrfx\mdk\ nrf52840_bitfield.h

C:\Users\gn0061\AppData\Local\Arduino15\packages\arduino\hardware\mbed_nano\2.6.1\cores\arduino\mbed\cmsis\CMSIS_5\CMSIS\TARGET_CORTEX_M\Include\core_cm4.h

https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf

This example: Directly access the NRF API and implement your own timer interrupt.

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

* Involves bypassing Arduino functions

Calculate vs Lookup

```ino
#define PI 3.1415926
#define ITER 10000
float angle = 0.0;
float angleStep = PI/32.0;
unsigned long count = 0;
unsigned long start_time;
void setup() {
  Serial.begin(9600);
  while(!Serial);
  start_time = millis();
}
void loop() {
  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 ×

Not connected. Select a board and a port to connect automatica | No Line Ending | 9600 baud

```
Calculate 10000 sine waves took 1411 ms
Calculate 10000 sine waves took 1411 ms
Calculate 10000 sine waves took 1411 ms
Calculate 10000 sine waves took 1410 ms
Calculate 10000 sine waves took 1409 ms
Calculate 10000 sine waves took 1410 ms
Calculate 10000 sine waves took 1410 ms
Calculate 10000 sine waves took 1410 ms
```

```ino
#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() {
  Serial.begin(9600);
  while(!Serial);
  start_time = millis();
}
void loop() {
  for(byte i=0; i < 64; i++){
    // analogWrite(2, sin64[i]);
  }
  count++;
  if(count >= ITER){
    Serial.print("Look up ");
    Serial.print(ITER); Serial.print(" sine waves took ");
    Serial.print(millis() - start_time); Serial.println(" ms");
    count = 0;
    start_time = millis();
  }
}

```

Output Serial Monitor ×

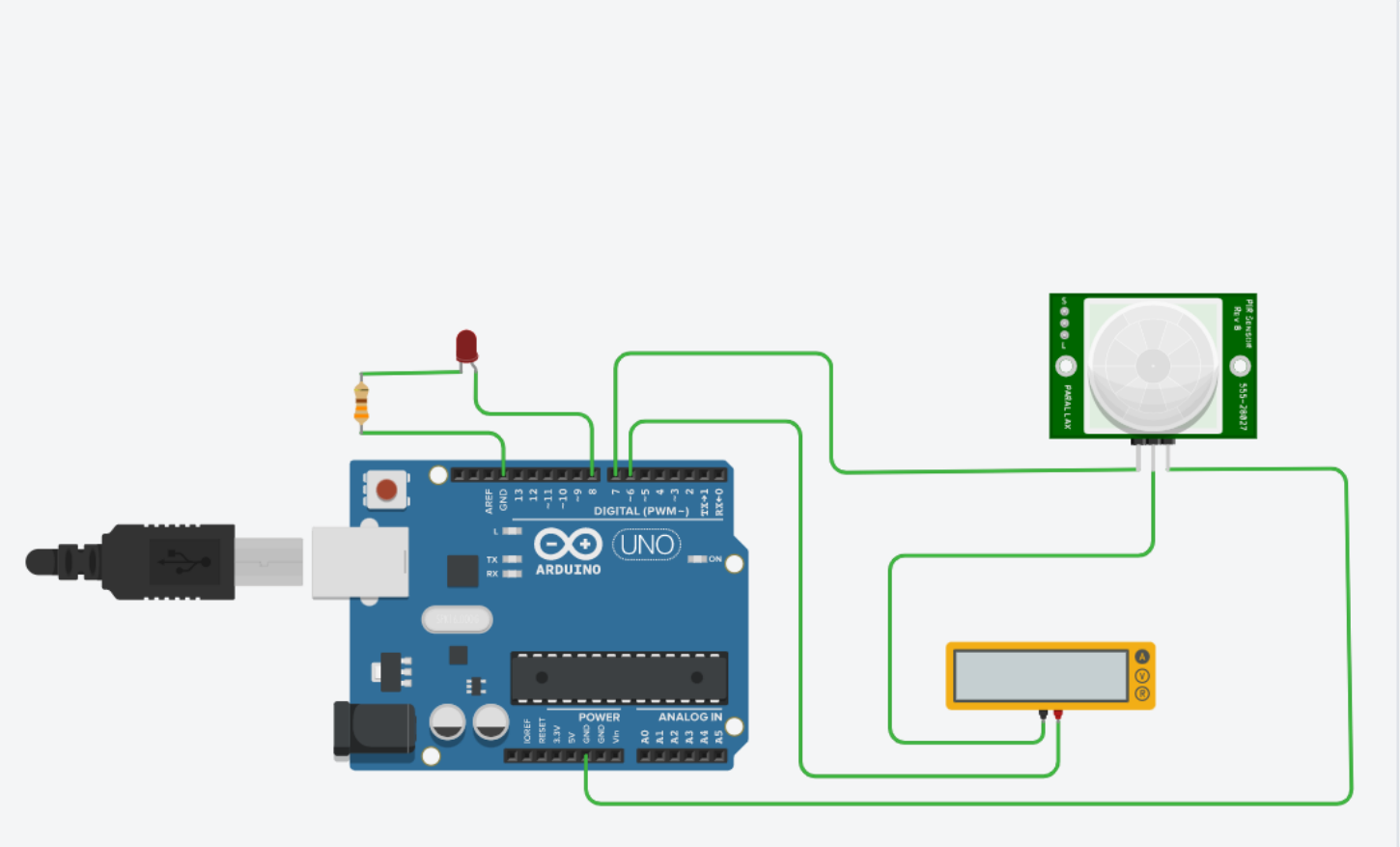
Not connected. Select a board and a port to connect automatically. | No Line Ending | 9600 baud

```
Look up 10000 sine waves took 6 ms
Look up 10000 sine waves took 6 ms
Look up 10000 sine waves took 6 ms
Look up 10000 sine waves took 5 ms
Look up 10000 sine waves took 5 ms
Look up 10000 sine waves took 6 ms
Look up 10000 sine waves took 6 ms
Look up 10000 sine waves took 6 ms
```

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

CONNECTIONS make a 'POWERFUL' difference



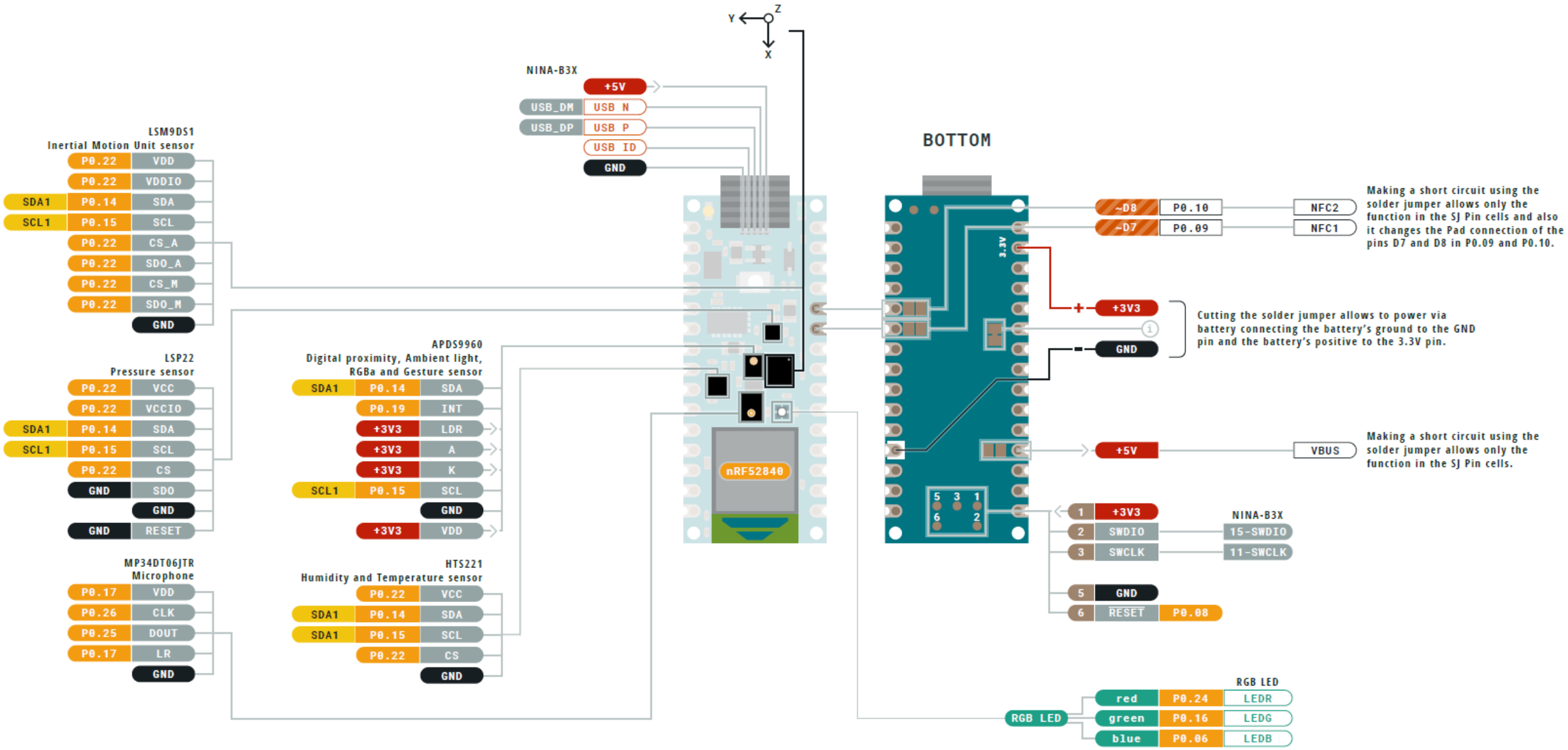

```
// C++ code
//
void setup()
{
  pinMode(7, INPUT);
  pinMode(8, OUTPUT);
  pinMode(6, OUTPUT);
}

void loop()
{
    digitalWrite(6,HIGH);
    delay(50);
    digitalWrite(8, digitalRead(7));
    delay(50);
    digitalWrite(6,LOW);
    delay(1000);
}
```

Power-saving modification: supply power to sensor (red line) from a digital pin, which is set to HIGH only when taking a sensor reading.

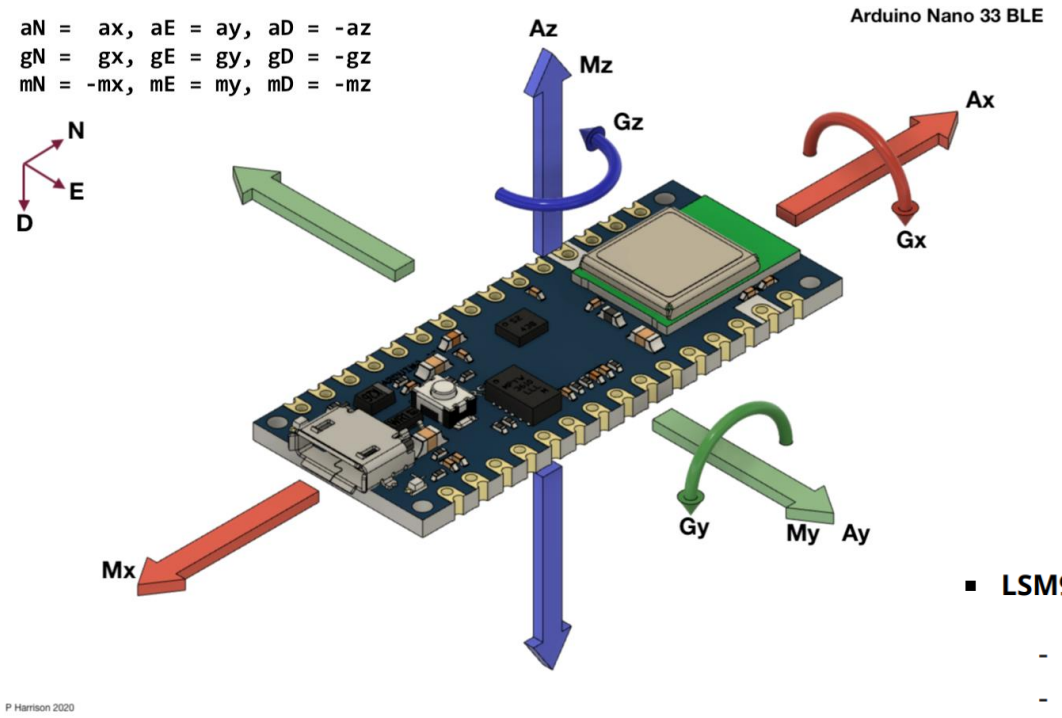
Minimize RAM usage

- 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

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
- Library "Arduino_LSM9DS1"

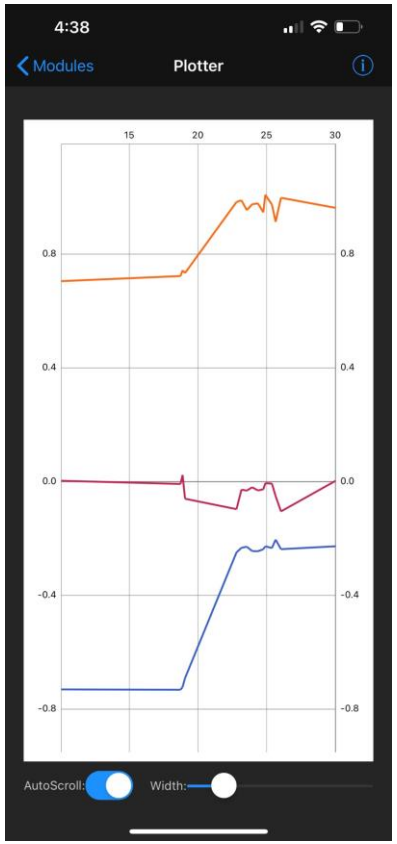
- **LSM9DS1** (9 axis IMU)

```
- 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
```

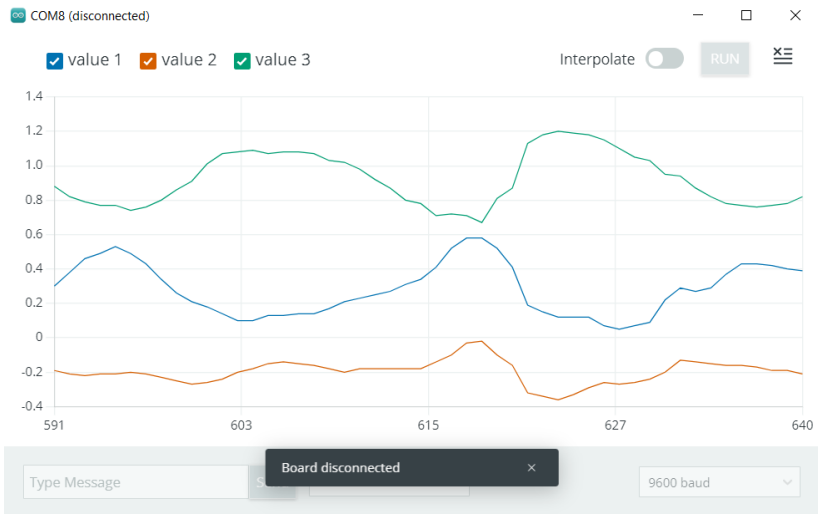
https://github.com/kriswiner/LSM9DS1/blob/master/LSM9DS1_MS5611_BasicAHRS_t3.ino
https://axodyne.com/2020/06/arduino-nano-33-ble-ahrs/
https://github.com/kriswiner/MPU6050/wiki/Simple-and-Effective-Magnetometer-Calibration

LSM9DS1

```
#include <Arduino_LSM9DS1.h>
#include <HardwareBLESerial.h>
HardwareBLESerial &bleSerial = HardwareBLESerial::getInstance();
void setup() {
  Serial.begin(9600);
  while(!bleSerial.beginAndSetupBLE("IE5995"));
  while(!IMU.begin());
}
void loop() {
  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.println(z); bleSerial.println(z);
  }
  delay(10);  // to avoid flooding bleSerial
}
```



Acceleration in G: x, y, z



Exercise:

- Write programs to read the gyroscope and magnetometer values
- Study the source code LSM9DS1.cpp
  - Use Wire.h to communicate with sensor device via I2C
  - The same method can be used to read other onboard sensors that has I2C bus

PDM MEMS Microphone

- MP34DT06JTR (Datasheet)
- Use the PDM library to sample audio
  - Implemented using a DoubleBuffer
  - To enable simultaneous sampling and processing


```
PDM.setBufferSize(1024) // bytes
```

C:\Users\gn0061\AppData\Local\Arduino15\packages\arduino\hardware\mbed_nano\2.6.1\libraries\PDM

Audio sampling code

```
#include <PDM.h>
#define SAMPBUFSIZE 256
static const char channels = 1;
static const int frequency = 16000;
unsigned int sampleBuffer[SAMPBUFSIZE]; // buffer for 16-bit samples
volatile int samplesRead;  // Number of audio samples read
void setup(){
  Serial.begin(115200);
  while (!Serial);
  PDM.setBufferSize(SAMPBUFSIZE*sizeof(unsigned int));
  PDM.onReceive(onPDMdata);
  PDM.setGain(0x01);
  PDM.begin(channels, frequency);
}
void loop(){
  if (samplesRead){
    for (int i = 0; i < samplesRead; i++){
      Serial.println(sampleBuffer[i]);
    }
    samplesRead = 0;
  }
}
void onPDMdata(){
  int bytesAvailable = PDM.available();
  PDM.read(sampleBuffer, bytesAvailable);
  samplesRead = bytesAvailable / sizeof(unsigned int);
}
```

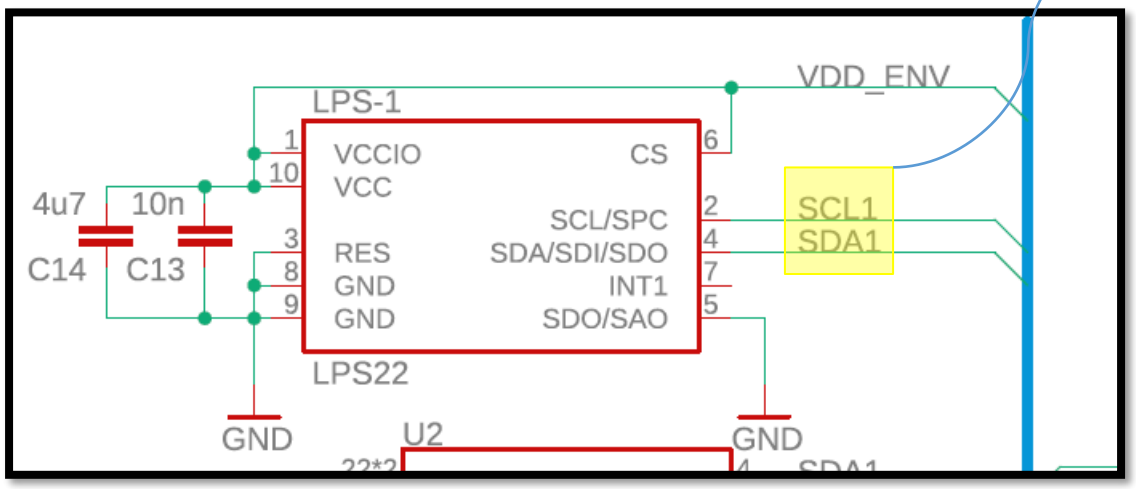
- Exercise – modify the code to:
  - Verify that data processing (the printout) is faster than sampling (the speed at which the buffer is filled), thus no data is lost
- Project ideas:
  - Save the audio data to SD card
  - Stream the audio to PC via WIFI

Interact with sensors on the I2C bus

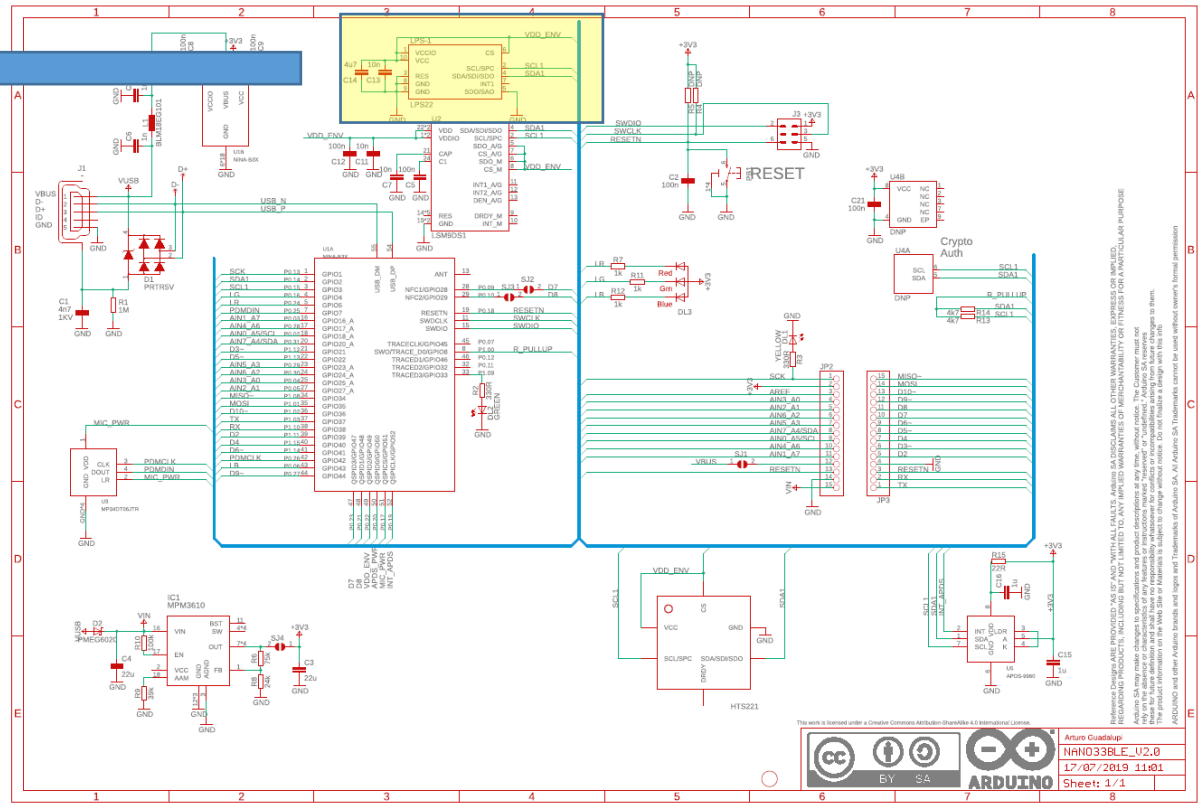
- There are ready-made libraries to read sensor data from the integrated sensors on the BLE Sense board.
- For the sake of learning, let us implement our own "device driver" to obtain sensor data.
- We will start with the barometric pressure sensor LPS22HB.

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.



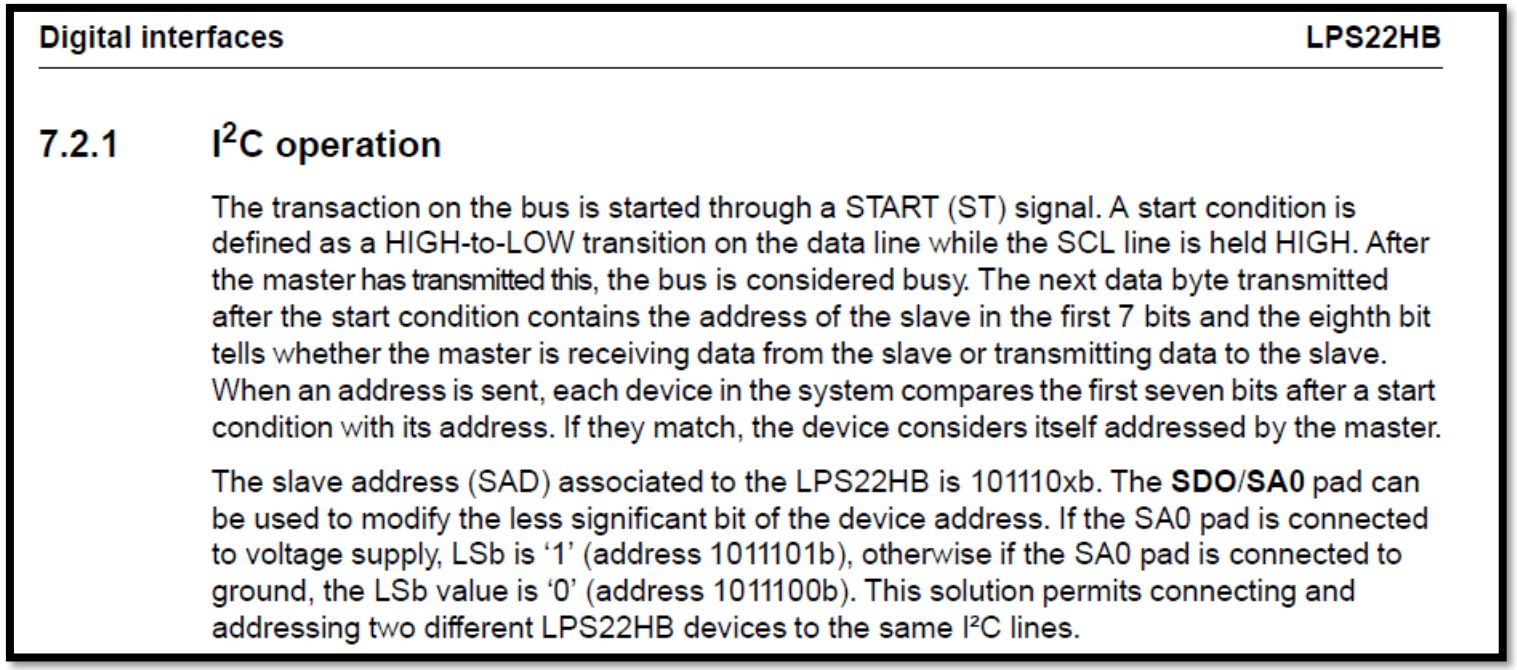
Note: SCL1 and SDA1 = Wire1 object in the Wire library; whereas SCL and SDA = Wire.



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.

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.

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^2C or SPI). Default value: 1 (0: disable; 1 enable)
I2C_DIS	Disable I^2C interface. Default value: 0 (0: I^2C enabled;1: I^2C 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.

- 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.

```
#define CTRL_REG2 0x11

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

- 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
```

- 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
```

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();
}
```

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);
}
```

Read data

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

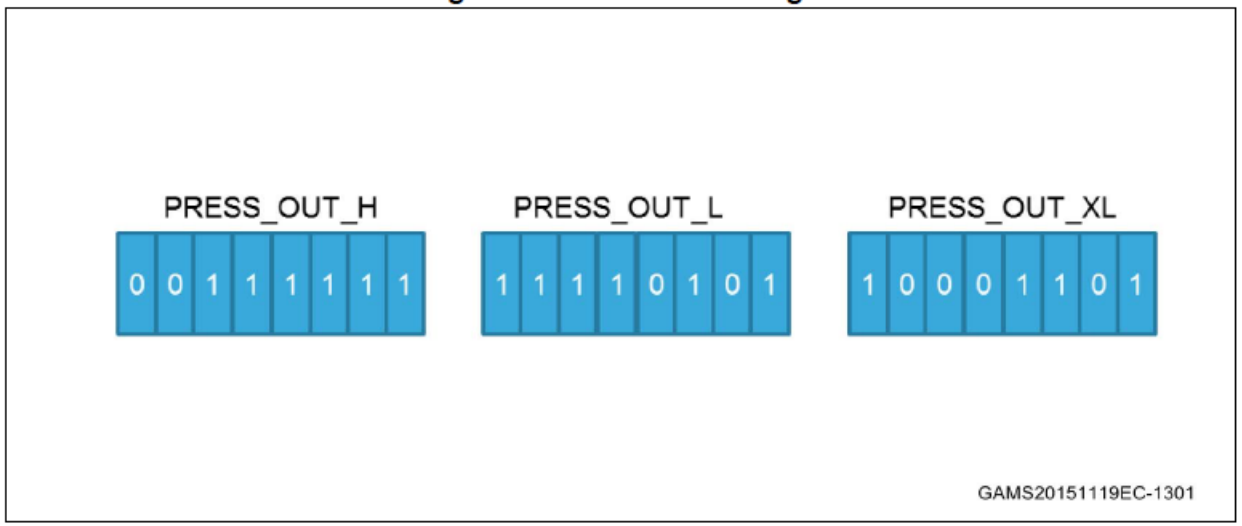
Table 14. Transfer when master is receiving (reading) one byte of data from slave

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);
}
```

```
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;
}
```

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;
```

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);
}
```

Exercise

- Write your own device driver for the integrated IMU sensor LSM9DS1

Advanced Programming Techniques

- The Nano 33 BLE is based on the nRF52840 microprocessor, which has lots of powerful features
- But most of the advanced features are not available through the standard Arduino functions
- How to learn / use these features and develop serious products?
- My experience is:
  - Read the nRF52840 Product Specification, again and again
  - Find sample code that does something close to what you need, digest it, line by line
  - For unfamiliar functions and symbols, dig into the source code directory and find the definitions

Access registers by address

4.4.1 Registers

Base address	Peripheral	Instance	Description	Configuration
0x10000000	FICR	FICR	Factory information configuration	

Table 8: Instances

Register	Offset	Description
CODEPAGESIZE	0x010	Code memory page size
CODESIZE	0x014	Code memory size
DEVICEID[0]	0x060	Device identifier
DEVICEID[1]	0x064	Device identifier
ER[0]	0x080	Encryption root, word 0
ER[1]	0x084	Encryption root, word 1

4.4.1.11 INFO.RAM

Address offset: 0x10C

RAM variant

Bit number	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ID	A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
Reset 0xFFFFFFFF	**1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1**

ID	Acce	Field	Value ID	Value	Description
A	R	RAM			RAM variant
			K16	0x10	16 kB RAM
			K32	0x20	32 kB RAM
			K64	0x40	64 kB RAM
			K128	0x80	128 kB RAM
			K256	0x100	256 kB RAM
			Unspecified	0xFFFFFFFF	Unspecified

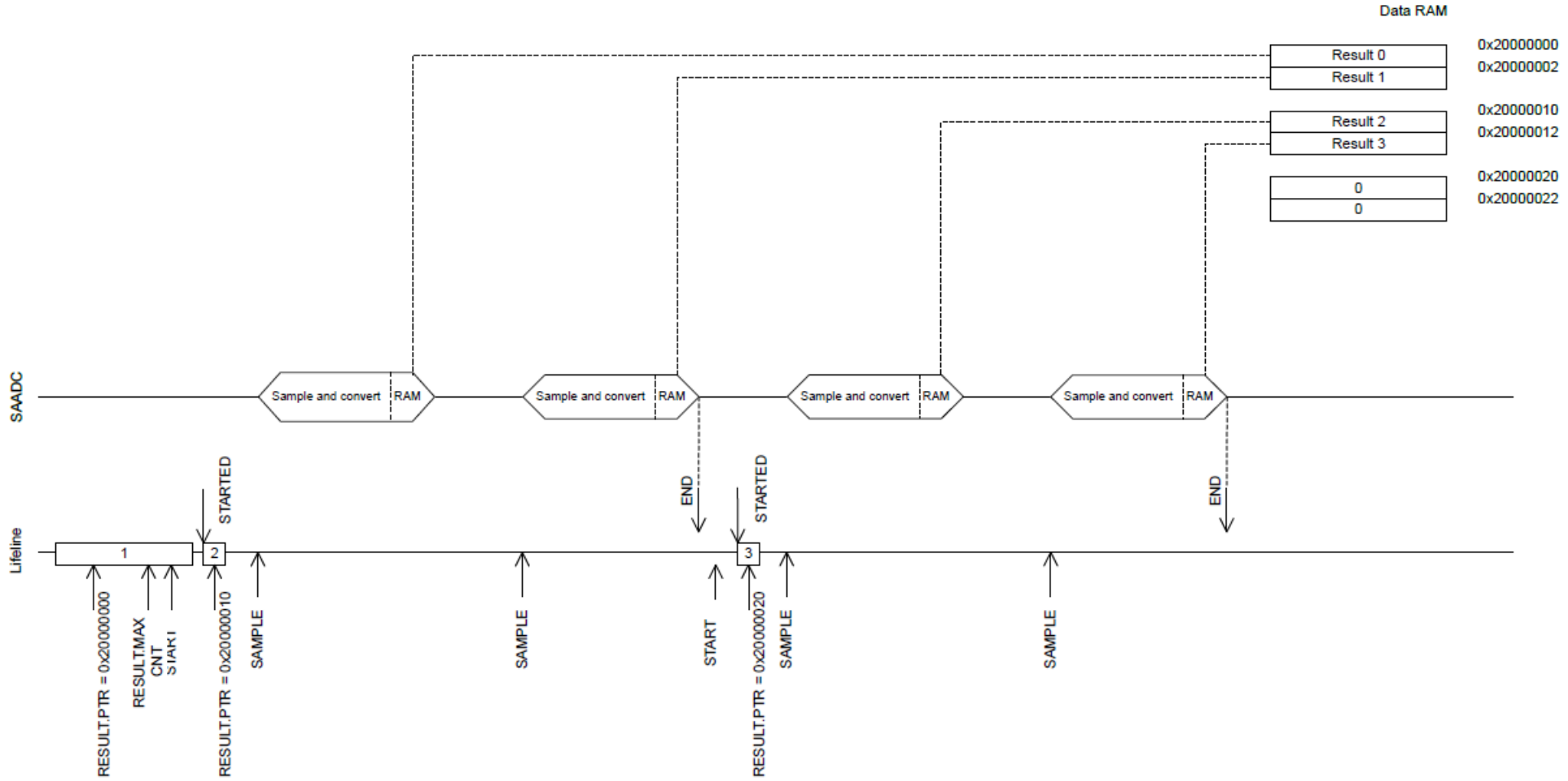
```
// nrf52840: get device information from
// 4.4 FICR - Factory information configuration registers
// Page 31 on PS ver 1.7
uint32_t *INFO_RAM_addr = (uint32_t *)(0x10000000 + 0x10C);
char msg[30];
void setup() {
  // read the RAM information
  uint32_t ram = *INFO_RAM_addr;
  switch(ram){
    case 0x10:
      strcpy(msg, "16kB RAM"); break;
    case 0x20:
      strcpy(msg, "32kB RAM"); break;
    case 0x40:
      strcpy(msg, "64kB RAM"); break;
    case 0x80:
      strcpy(msg, "128kB RAM"); break;
    case 0x100:
      strcpy(msg, "256kB RAM"); break;
    default:
      strcpy(msg, "Unspecified RAM");
  }
  Serial.begin(9600);
}
void loop() {
  Serial.println(msg);
  delay(2000);
}
```

Exercises

- Use the same approach (i.e., access registers by address) to
  - Get the total flash size on the MCU
  - Turning on and off some GPIO pin (e.g., the one connected to the LED_BUILTIN)
- These exercises force you to read the product specification

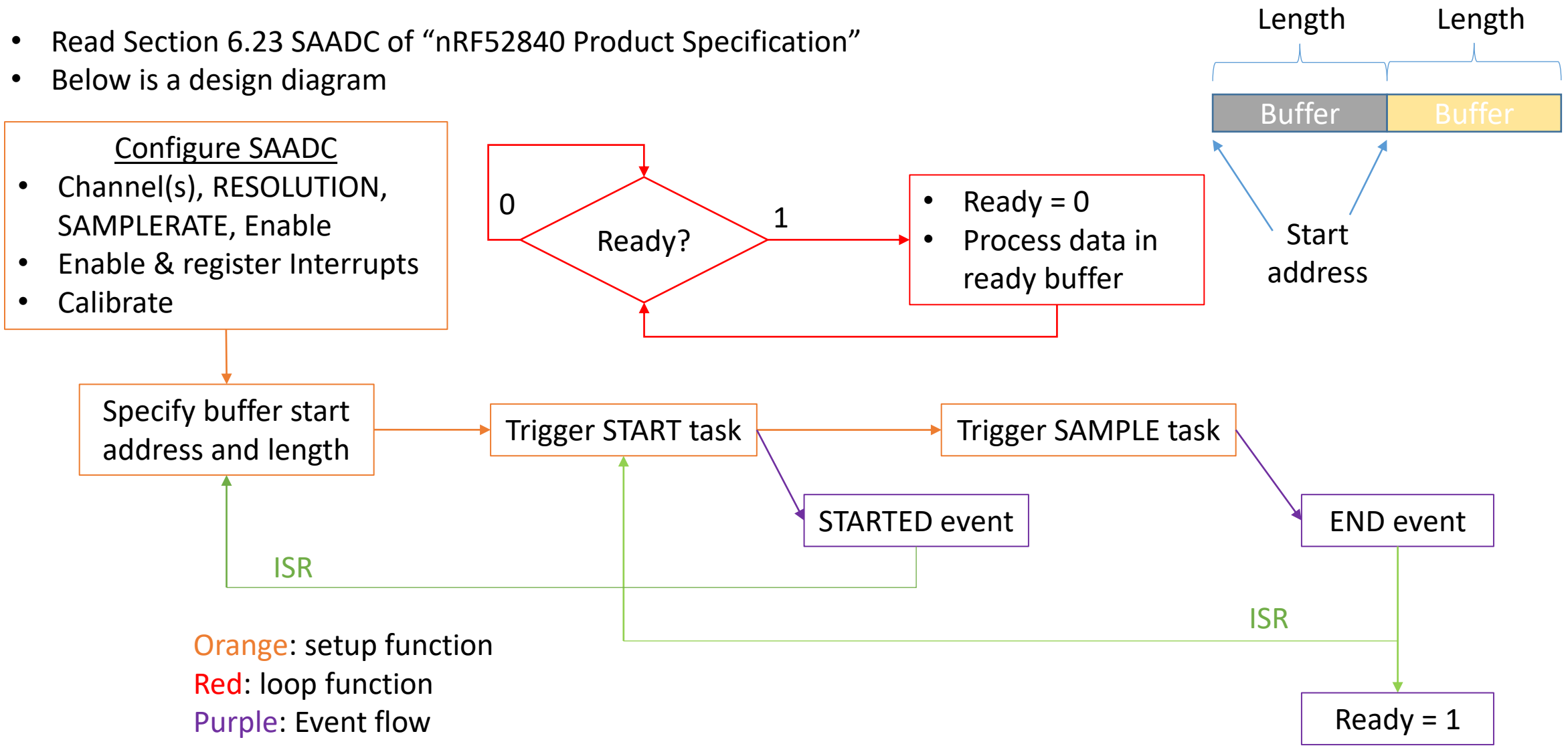
nRF52840 SAADC

Continuously read analog data at a given sampling rate.
Section 6.23 SAADC – Successive approximation analog-to-digital converter



nRF52840's SAADC continuous sampling

- Read Section 6.23 SAADC of "nRF52840 Product Specification"
- Below is a design diagram



SAADC Configuration

```
// Disable the SAADC during configuration
nrf_saadc_disable();
// Configure A2 and A3 pins in differential mode
const nrf_saadc_channel_config_t channel_config = {
    .resistor_p = NRF_SAADC_RESISTOR_DISABLED,
    .resistor_n = NRF_SAADC_RESISTOR_DISABLED,
    .gain = NRF_SAADC_GAIN1_6,
    .reference = NRF_SAADC_REFERENCE_INTERNAL,
    .acq_time = NRF_SAADC_ACQTIME_3US,
    .mode = NRF_SAADC_MODE_DIFFERENTIAL,
    .burst = NRF_SAADC_BURST_DISABLED,
    .pin_p = NRF_SAADC_INPUT_AIN2, // Pin A2
    .pin_n = NRF_SAADC_INPUT_AIN3  // Pin A3
};
// initialize the SAADC channel by calling the hal function, declared in nrf_saadc.h
nrf_saadc_channel_init(1, &channel_config);  // use channel 1
nrf_saadc_resolution_set(NRF_SAADC_RESOLUTION_12BIT);  // Configure the resolution
nrf_saadc_oversample_set(NRF_SAADC_OVERSAMPLE_DISABLED);  // Disable oversampling
// Enable Continuous Mode & set to 16MHz / 1000 (CC) = 16 kHz
NRF_SAADC->SAMPLERATE = (SAADC_SAMPLERATE_MODE_Timers << SAADC_SAMPLERATE_MODE_Pos)
                        | ((uint32_t)1000 << SAADC_SAMPLERATE_CC_Pos);
// Configure RESULT Buffer and MAXCNT
NRF_SAADC->RESULT.PTR = (uint32_t)SAADC_RESULT_BUFFER;
NRF_SAADC->RESULT.MAXCNT = SAADC_RESULT_BUFFER_SIZE;
// Set the END mask to an interrupt: when the result buffer is filled, trigger an interrupt
nrf_saadc_int_enable(NRF_SAADC_INT_END);
// Enable the STARTED event interrupt, to trigger an interrupt each time the STARTED event happens
nrf_saadc_int_enable(NRF_SAADC_INT_STARTED);
// Register the interrupts in NVIC, these functions are declared in core_cm4.h
NVIC_SetPriority(SAADC_IRQn, 3UL);
NVIC_EnableIRQ(SAADC_IRQn);
nrf_saadc_enable();  // Enable the SAADC
```

Global variables

- Used both the HAL (Hardware Abstraction Layer) functions and the MDK (Microprocessor Development Kit) functions by nRF
- Header file locations:
  - C:\Users\gn0061\AppData\Local\Arduino15\packages\arduino\hardware\mbed_nano\2.6.1\cores\arduino\mbed\targets\TARGET_NORDIC\TARGET_NRF5x\TARGET_SDK_15_0\modules\nrfx\mdk
  - C:\Users\gn0061\AppData\Local\Arduino15\packages\arduino\hardware\mbed_nano\2.6.1\cores\arduino\mbed\targets\TARGET_NORDIC\TARGET_NRF5x\TARGET_SDK_15_0\modules\nrfx\hal

Notes on the SAADC Configuration

“.acq_time = NRF_SAADC_ACQTIME_3US,” should be set according to the source resistance

The acquisition time indicates how long the capacitor is connected, see TACQ field in CH[n].CONFIG register. The required acquisition time depends on the source resistance (R_{source}). For high source resistance the acquisition time should be increased:

TACQ [µs]	Maximum source resistance [kΩ]
3	10
5	40
10	100
15	200
20	400
40	800

Table 95: Acquisition time

When using VDDHDIV5 as input, the acquisition time needs to be 10 µs or higher.

6.23.2 Reference voltage and gain settings

Each SAADC channel can have individual reference and gain settings.

This is configured in registers CH[n].CONFIG (n=0..7) on page 394. Available configuration options are:

- VDD/4 or internal 0.6 V reference
- Gain ranging from 1/6 to 4

```
.gain = NRF_SAADC_GAIN1_6,
.reference = NRF_SAADC_REFERENCE_INTERNAL,
```

The gain setting can be used to control the effective input range of the SAADC:

```
Input range = (±0.6 V or ±VDD/4)/gain
```

For example, selecting VDD as reference, single-ended input (grounded negative input), and a gain of 1/4 will result in the following input range:

```
Input range = (VDD/4)/(1/4) = VDD
```

With internal reference, single-ended input (grounded negative input) and a gain of 1/6, the input range will be:

```
Input range = (0.6 V)/(1/6) = 3.6 V
```

Inputs AIN0 through AIN7 cannot exceed VDD or be lower than VSS.

6.23.3 Digital output

The digital output value from the SAADC is calculated using a formula.

$$RESULT = (V(P) - V(N)) * (GAIN/REFERENCE) * 2^{(RESOLUTION - m)}$$

where

V(P)

is the voltage at input P

V(N)

is the voltage at input N

GAIN

is the selected gain

REFERENCE

is the selected reference voltage

RESOLUTION

is output resolution in bits, as configured in register RESOLUTION on page 395

m

is 0 for single-ended channels

is 1 for differential channels

Results are sign extended to 16 bits and stored as little-endian byte order in RAM.

6.23.8 Calibration

The SAADC has a temperature dependent offset.

Therefore, it is recommended to calibrate the SAADC at least once before use, and to re-run calibration every time the ambient temperature has changed by more than 10 °C.

Offset calibration is started by triggering the CALIBRATEOFFSET task, and the CALIBRATEDONE event is generated when calibration is done.

Calibration code:

```
// Calibrate the SAADC by finding its offset
NRF_SAADC->TASKS_CALIBRATEOFFSET = 1;
while (NRF_SAADC->EVENTS_CALIBRATEDONE == 0);
NRF_SAADC->EVENTS_CALIBRATEDONE = 0;
while (NRF_SAADC->STATUS == (SAADC_STATUS_STATUS_Busy << SAADC_STATUS_STATUS_Pos));
Serial.println(F("Finished SAADC Configuration"));
```


```
// Trigger the START task
nrf_saadc_task_trigger(NRF_SAADC_TASK_START);
delay(5);  // Allow some time for the START task to trigger
// Trigger the SAMPLE task
nrf_saadc_task_trigger(NRF_SAADC_TASK_SAMPLE);
```

This completes the void setup() function!

The Global Variables – set up the buffer array and indicator variables

```
const uint16_t NUM_SAADC_RESULT_BUFFERS = 2;
const uint16_t SAADC_RESULT_BUFFER_SIZE = 8192;
// Contains the (16-bit) results for the SAADC - volatile buffer
volatile nrf_saadc_value_t SAADC_RESULT_BUFFER[NUM_SAADC_RESULT_BUFFERS * SAADC_RESULT_BUFFER_SIZE];
volatile uint8_t READY_INDEX = 0;  // the segment index that is filled and ready to move
volatile uint32_t BUFFER_INDEX = 0;  // Buffer index (For the SAADC Buffers)
```

The Interrupt Service Routine (ISR)

```
extern "C" {
  void SAADC_IRQHandler_v() {
    // Check to see if the ADC has filled up the result buffer
    if (nrf_saadc_event_check(NRF_SAADC_EVENT_END)){
      nrf_saadc_event_clear(NRF_SAADC_EVENT_END); // Clear the END event
      READY_INDEX = BUFFER_INDEX == 0 ? NUM_SAADC_RESULT_BUFFERS - 1 : BUFFER_INDEX - 1;
      nrf_saadc_task_trigger(NRF_SAADC_TASK_START);  // Trigger the START task
    } else if (nrf_saadc_event_check(NRF_SAADC_EVENT_STARTED)){
      nrf_saadc_event_clear(NRF_SAADC_EVENT_STARTED);  // Clear the STARTED event
      // Move the RESULT.PTR to the next segment of the ping-pong buffer
      BUFFER_INDEX = BUFFER_INDEX >= NUM_SAADC_RESULT_BUFFERS - 1 ? 0 : BUFFER_INDEX + 1;
      NRF_SAADC->RESULT.PTR = (uint32_t)(SAADC_RESULT_BUFFER + (BUFFER_INDEX * SAADC_RESULT_BUFFER_SIZE));
    }
  }
}
```

The loop

- Currently does nothing
- Just flashes the LED to verify that things work

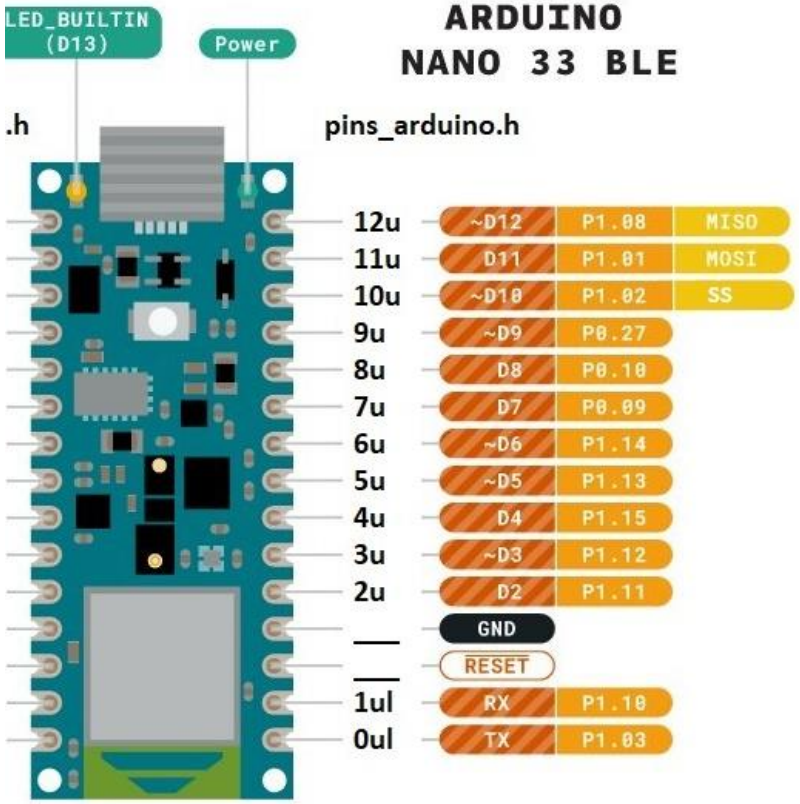
```
void loop() {
  if(READY_INDEX){
    digitalWrite(13, HIGH);
  } else {
    digitalWrite(13, LOW);
  }
}
```

The LED should blink at a round 1 Hz frequency.

- **Project ideas:**
  - Sample audio at 16 KHz (or higher) and stream data continuously to PC or Cellphone via Bluetooth or WIFI
    - Kind of a remote microphone
  - Take a short audio clip (e.g., 1 sec) and perform inference based on some TensorFlow Lite model
    - Wake words recognition
  - Record and send short audio clips to PC for training data collection for the TF Lite model
  - Bioelectric sensing -> smart health
  - Vibration sensing -> machine diagnosis
  - ADC -> DSP -> DAC tasks

6.9.2 Registers

Base address	Peripheral	Instance	Description	Configuration
0x50000000	GPIO	GPIO	General purpose input and output	Deprecated
0x50000000	GPIO	P0	General purpose input and output, port 0	P0.00 to P0.31 implemented
0x50000300	GPIO	P1	General purpose input and output, port	P1.00 to P1.15 implemented



If we want to set pin D2 in sense mode, and sense for high level:

```
NRF_P1->PIN_CNF[11] &= ~((uint32_t)GPIO_PIN_CNF_SENSE_Msk);
NRF_P1->PIN_CNF[11] |= ((uint32_t)GPIO_PIN_CNF_SENSE_High << GPIO_PIN_CNF_SENSE_Pos);
```

nrf52840.h

```
typedef struct {                                    /*!< GPIO Structure                                                */
  __I  uint32_t  RESERVED0[321];
  __IO uint32_t  OUT;                               /*!< Write GPIO port                                               */
  __IO uint32_t  OUTSET;                            /*!< Set individual bits in GPIO port                              */
  __IO uint32_t  OUTCLR;                            /*!< Clear individual bits in GPIO port                            */
  __I  uint32_t  IN;                                /*!< Read GPIO port                                                */
  __IO uint32_t  DIR;                               /*!< Direction of GPIO pins                                        */
  __IO uint32_t  DIRSET;                            /*!< DIR set register                                              */
  __IO uint32_t  DIRCLR;                            /*!< DIR clear register                                            */
  __IO uint32_t  LATCH;                             /*!< Latch register indicating what GPIO pins that have met the criteria
                                                         set in the PIN_CNF[n].SENSE registers                         */
  __IO uint32_t  DETECTMODE;                        /*!< Select between default DETECT signal behaviour and LDETECT mode */
  __I  uint32_t  RESERVED1[118];
  __IO uint32_t  PIN_CNF[32];                       /*!< Description collection[n]: Configuration of GPIO pins         */
} NRF_GPIO_Type;
```

DETECTMODE

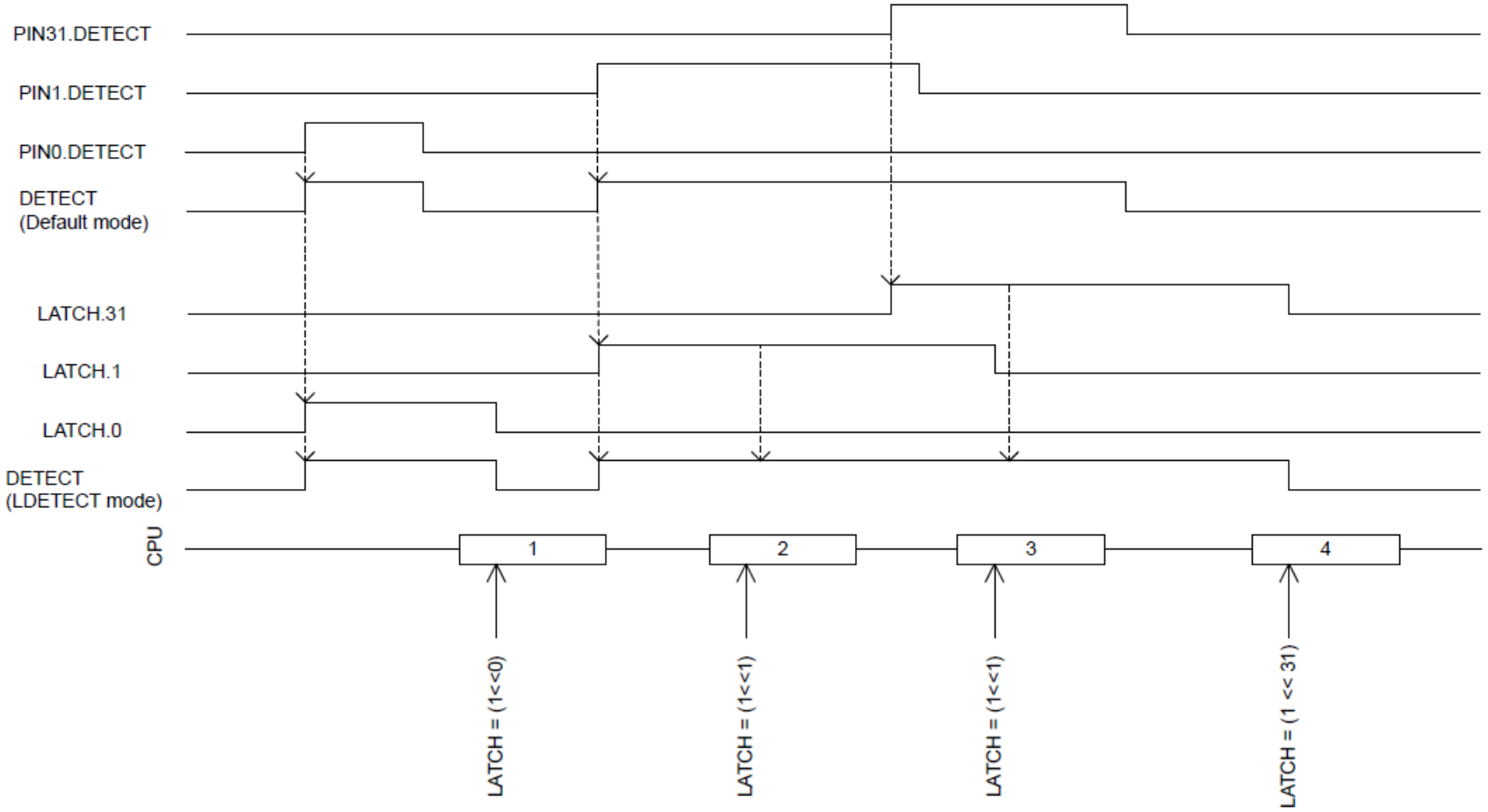


Figure 46: DETECT signal behavior

Follow the directions in the PS

Setting the system to System OFF while DETECT is high will cause a wakeup from System OFF reset.

This feature is always enabled even if the peripheral itself appears to be IDLE, meaning no clocks or other power intensive infrastructure have to be requested to keep this feature enabled. This feature can therefore be used to wake up the CPU from a WFI or WFE type sleep in System ON when all peripherals and the CPU are idle, meaning the lowest power consumption in System ON mode.

In order to prevent spurious interrupts from the PORT event while configuring the sources, the following must be performed:

1. Disable interrupts on the PORT event (through INTENCLR.PORT).
2. Configure the sources (PIN_CNF[n].SENSE).
3. Clear any potential event that could have occurred during configuration (write '0' to EVENTS_PORT).
4. Enable interrupts (through INTENSET.PORT).

Read:

Section 5.3.1 Power – Power supply

Section 6.9 GPIO

Section 6.10 GPIOTE

For implementation, refer to nrf52840.h and nrf52840_bitfield.h in the directory:

C:\Users\gn0061\AppData\Local\Arduino15\packages\arduino\hardware\mbed_nano\2.6.1\cores\arduino\mbed\targets\TARGET_NORDIC\TARGET_NRF5x\TARGET_SDK_15_0\modules\nrfx\mdk

```
 // System is turned off after a few seconds. Connect Pin D2 to V3.3 to wake up.
const uint8_t P1_pin = 11;  // P1.11 is D2
void setup() {
  // Startup
  for(uint8_t i=0; i<5; i++){
    digitalWrite(13, HIGH); delay(500); digitalWrite(13, LOW); delay(500);
  }
  // Start Preparing for the wake-up mechanism
  // Disable port interrupt
  NRF_GPIOTE->INTENCLR |= ((uint32_t)GPIOTE_INTENCLR_PORT_Clear << GPIOTE_INTENCLR_PORT_Pos);
  // Configure wake-up mechanism
  NRF_P1->DETECTMODE = 0;  // default detect mode
  NRF_P1->LATCH &= ~((uint32_t)GPIO_LATCH_PIN11_Msk);  // Clear latch status of PIN11
  // Set direction to input mode
  NRF_P1->PIN_CNF[P1_pin] &= ~((uint32_t)GPIO_PIN_CNF_DIR_Msk);
  NRF_P1->PIN_CNF[P1_pin] |= ((uint32_t)GPIO_PIN_CNF_DIR_Input << GPIO_PIN_CNF_DIR_Pos);
  // Connect input buffer
  NRF_P1->PIN_CNF[P1_pin] &= ~((uint32_t)GPIO_PIN_CNF_INPUT_Msk);
  NRF_P1->PIN_CNF[P1_pin] |= ((uint32_t)GPIO_PIN_CNF_INPUT_Connect << GPIO_PIN_CNF_INPUT_Pos);
  // Set pulldown
  NRF_P1->PIN_CNF[P1_pin] &= ~((uint32_t)GPIO_PIN_CNF_PULL_Msk);
  NRF_P1->PIN_CNF[P1_pin] |= ((uint32_t)GPIO_PIN_CNF_PULL_Pulldown << GPIO_PIN_CNF_PULL_Pos);
  // Generate Sense signal when pin is HIGH
  NRF_P1->PIN_CNF[P1_pin] &= ~((uint32_t)GPIO_PIN_CNF_SENSE_Msk);
  NRF_P1->PIN_CNF[P1_pin] |= ((uint32_t)GPIO_PIN_CNF_SENSE_High << GPIO_PIN_CNF_SENSE_Pos);
  // Clear any port event that might have occurred during the configuration
  NRF_GPIOTE->EVENTS_PORT = 0;
  // Enable port interrupt
  NRF_GPIOTE->INTENSET |= ((uint32_t)GPIOTE_INTENSET_PORT_Set << GPIOTE_INTENSET_PORT_Pos);
  // End preparing for the wake-up mechanism
  // Now put device into System Off mode
  NRF_POWER->SYSTEMOFF = 1;
}
void loop() {}
```

Wake up from SYSTEMOFF power-saving mode.