CSE211s:Introduction to Embedded Systems

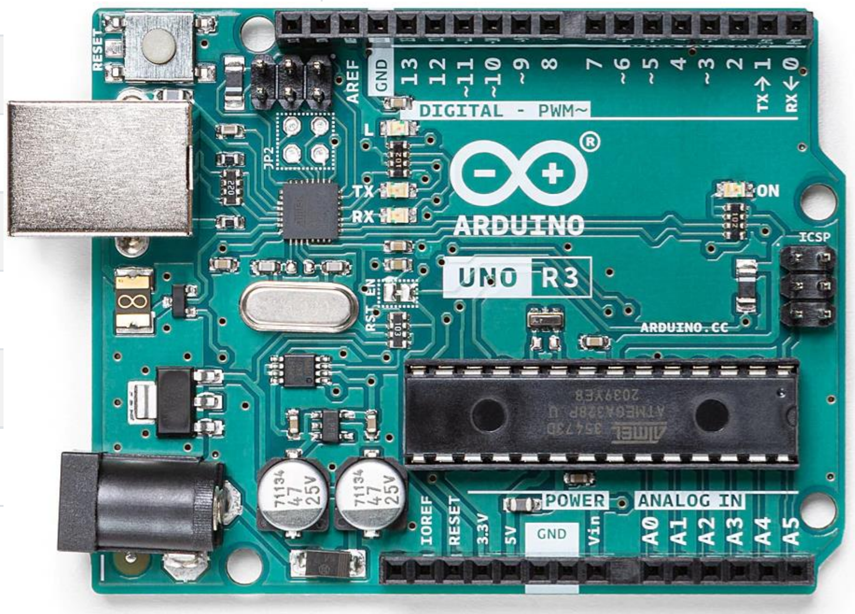
Lect. #10: Revision

Ahmed M. Zaki

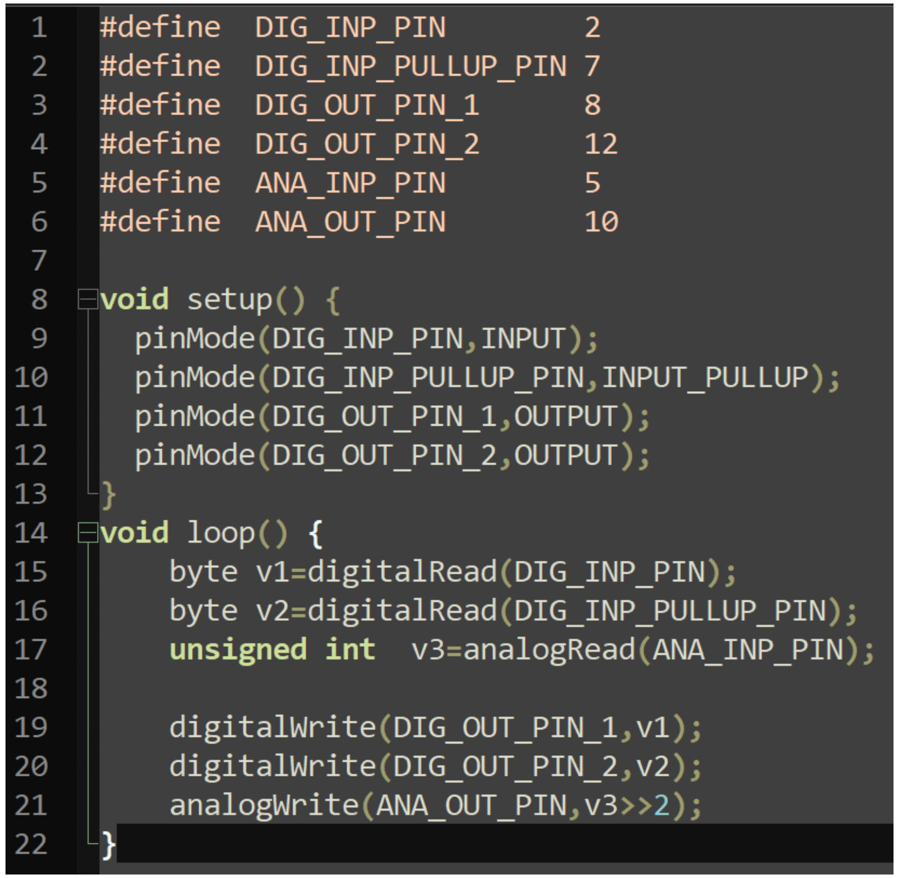
Arduino UNO

MICROCONTROLLER	ATmega328P
OPERATING VOLTAGE	5V
INPUT VOLTAGE (RECOMMENDED)	7-12V
INPUT VOLTAGE (LIMIT)	6-20V
DIGITAL I/O PINS	14 (of which 6 provide PWM output)
PWM DIGITAL I/O PINS	6
ANALOG INPUT PINS	6
DC CURRENT PER I/O PIN	20 mA
DC CURRENT FOR 3.3V PIN	50 mA

FLASH MEMORY	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
CLOCK SPEED	16 MHz
LED_BUILTIN	13
LENGTH	68.6 mm
WIDTH	53.4 mm
WEIGHT	25 g



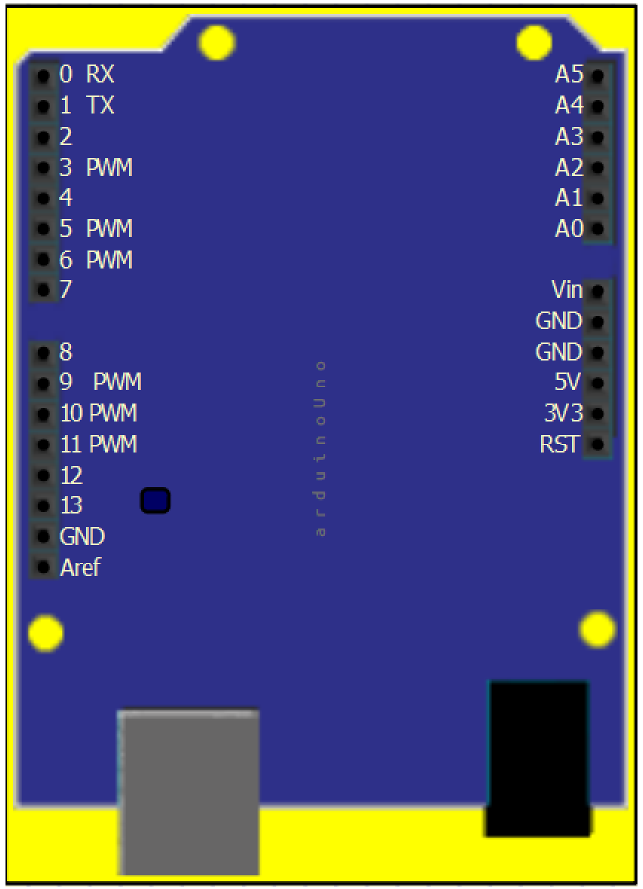
IO Configuration



```
#define  DIG_INP_PIN        2
#define  DIG_INP_PULLUP_PIN 7
#define  DIG_OUT_PIN_1      8
#define  DIG_OUT_PIN_2      12
#define  ANA_INP_PIN        5
#define  ANA_OUT_PIN        10

void setup() {
  pinMode(DIG_INP_PIN,INPUT);
  pinMode(DIG_INP_PULLUP_PIN,INPUT_PULLUP);
  pinMode(DIG_OUT_PIN_1,OUTPUT);
  pinMode(DIG_OUT_PIN_2,OUTPUT);
}
void loop() {
    byte v1=digitalRead(DIG_INP_PIN);
    byte v2=digitalRead(DIG_INP_PULLUP_PIN);
    unsigned int  v3=analogRead(ANA_INP_PIN);

    digitalWrite(DIG_OUT_PIN_1,v1);
    digitalWrite(DIG_OUT_PIN_2,v2);
    analogWrite(ANA_OUT_PIN,v3>>2);
}
```

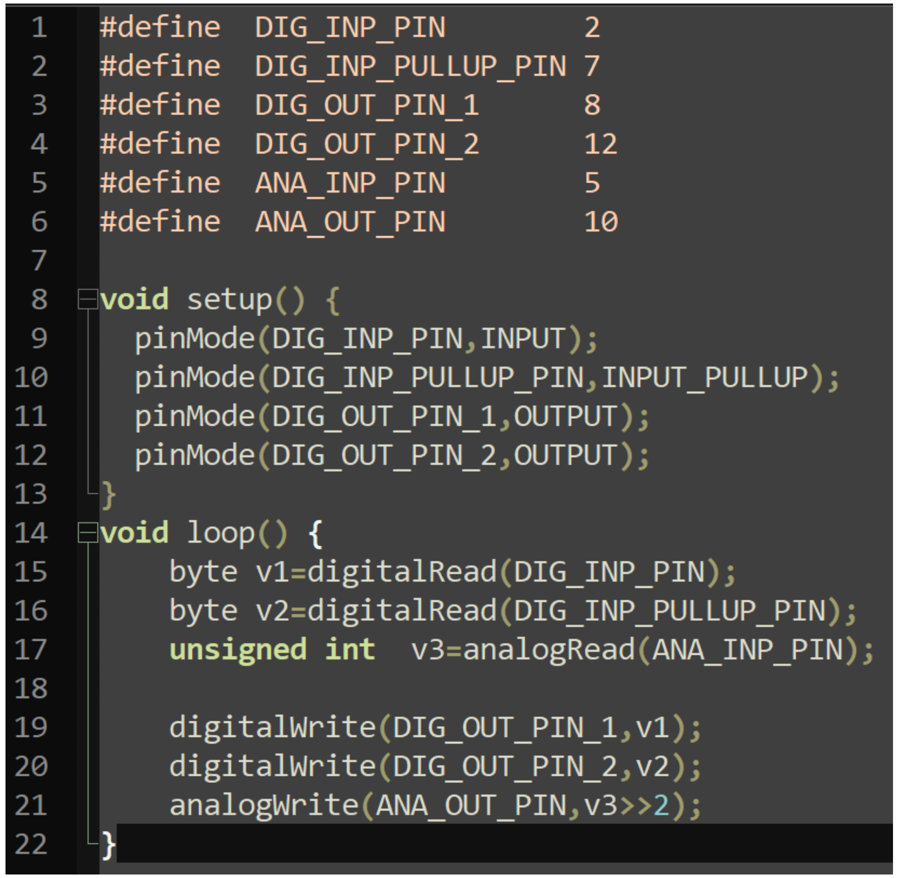
[0-13]
14 Digital I/O

[3,5,6,9,10,11]
6 of them
PWM

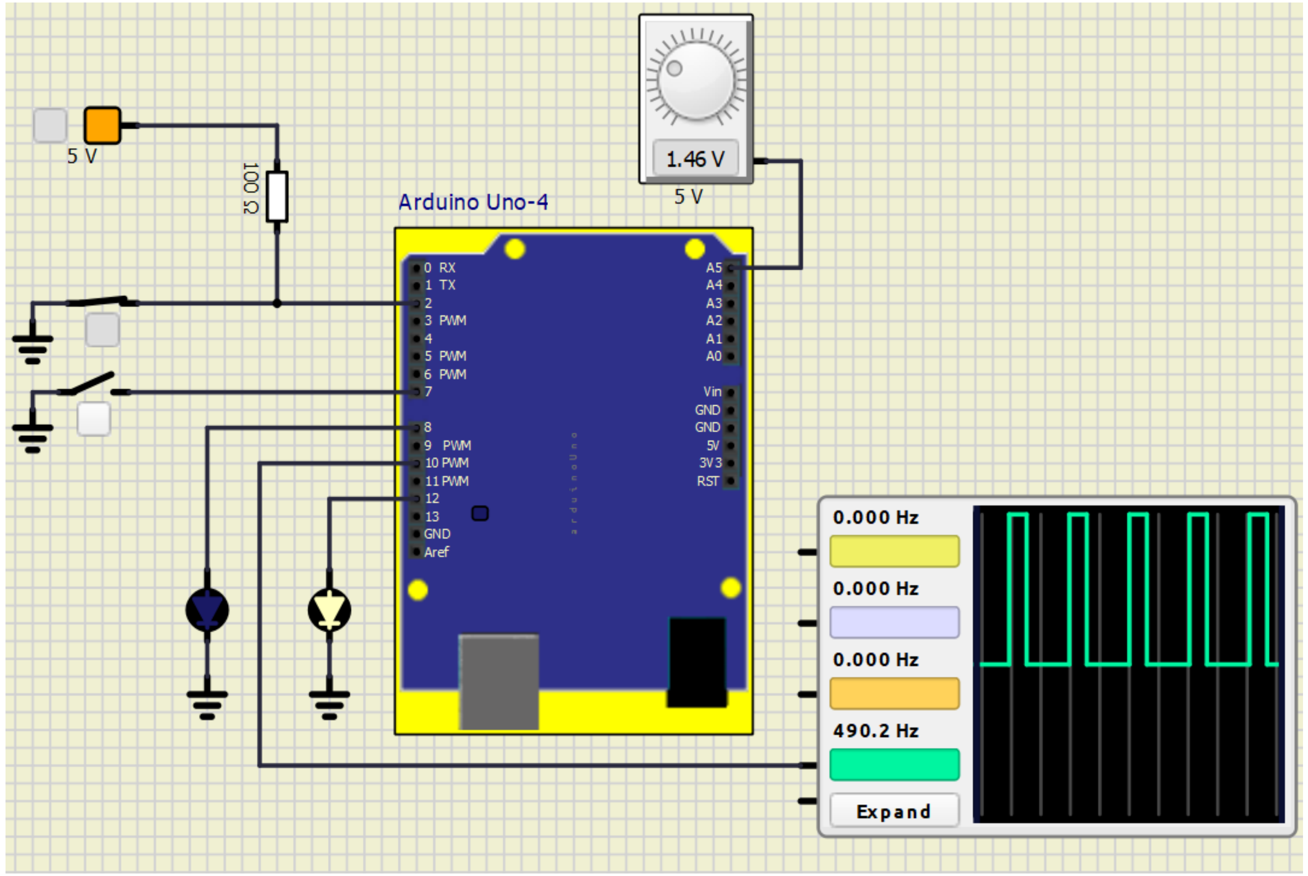


6 Analog Inputs

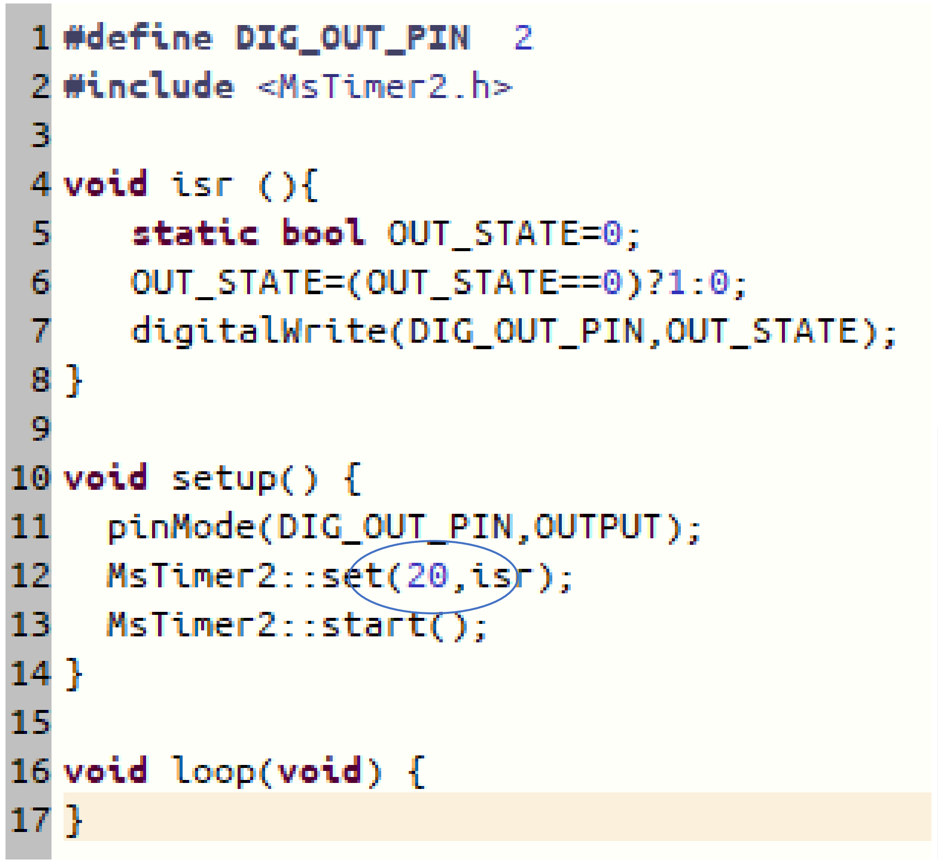
IO Configuration



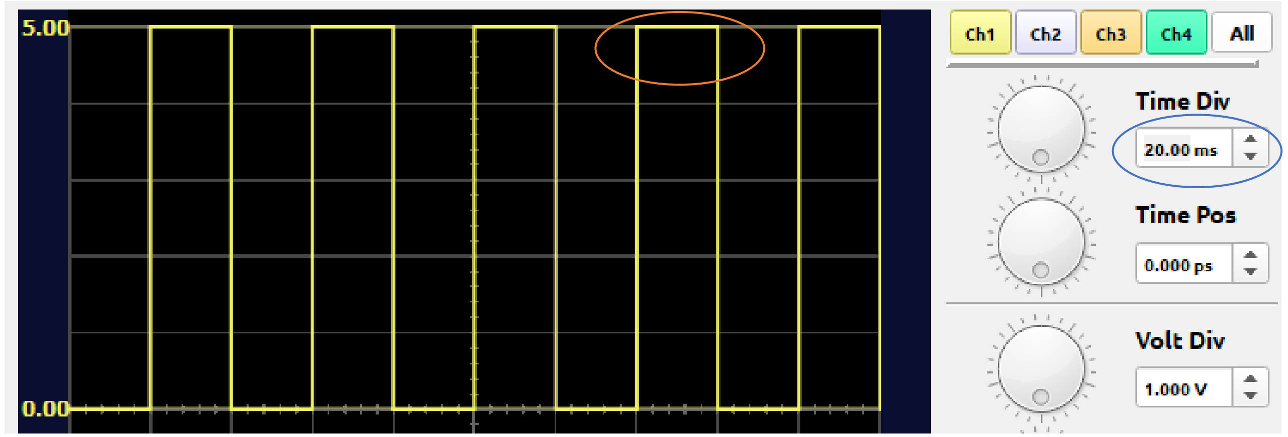
```
#define  DIG_INP_PIN        2
#define  DIG_INP_PULLUP_PIN 7
#define  DIG_OUT_PIN_1      8
#define  DIG_OUT_PIN_2      12
#define  ANA_INP_PIN        5
#define  ANA_OUT_PIN        10

void setup() {
  pinMode(DIG_INP_PIN,INPUT);
  pinMode(DIG_INP_PULLUP_PIN,INPUT_PULLUP);
  pinMode(DIG_OUT_PIN_1,OUTPUT);
  pinMode(DIG_OUT_PIN_2,OUTPUT);
}
void loop() {
    byte v1=digitalRead(DIG_INP_PIN);
    byte v2=digitalRead(DIG_INP_PULLUP_PIN);
    unsigned int  v3=analogRead(ANA_INP_PIN);

    digitalWrite(DIG_OUT_PIN_1,v1);
    digitalWrite(DIG_OUT_PIN_2,v2);
    analogWrite(ANA_OUT_PIN,v3>>2);
}
```



Interrupts
(Timer)



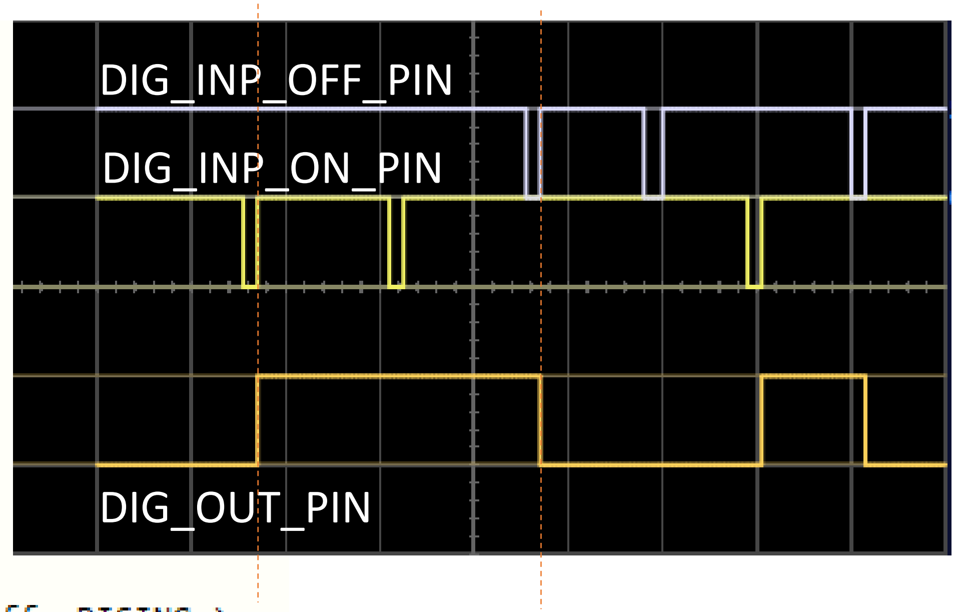
```
#define DIG_OUT_PIN   2
#include <MsTimer2.h>

void isr (){
   static bool OUT_STATE=0;
   OUT_STATE=(OUT_STATE==0)?1:0;
   digitalWrite(DIG_OUT_PIN,OUT_STATE);
}

void setup() {
  pinMode(DIG_OUT_PIN,OUTPUT);
  MsTimer2::set(20,isr);
  MsTimer2::start();
}

void loop(void) {
}
```



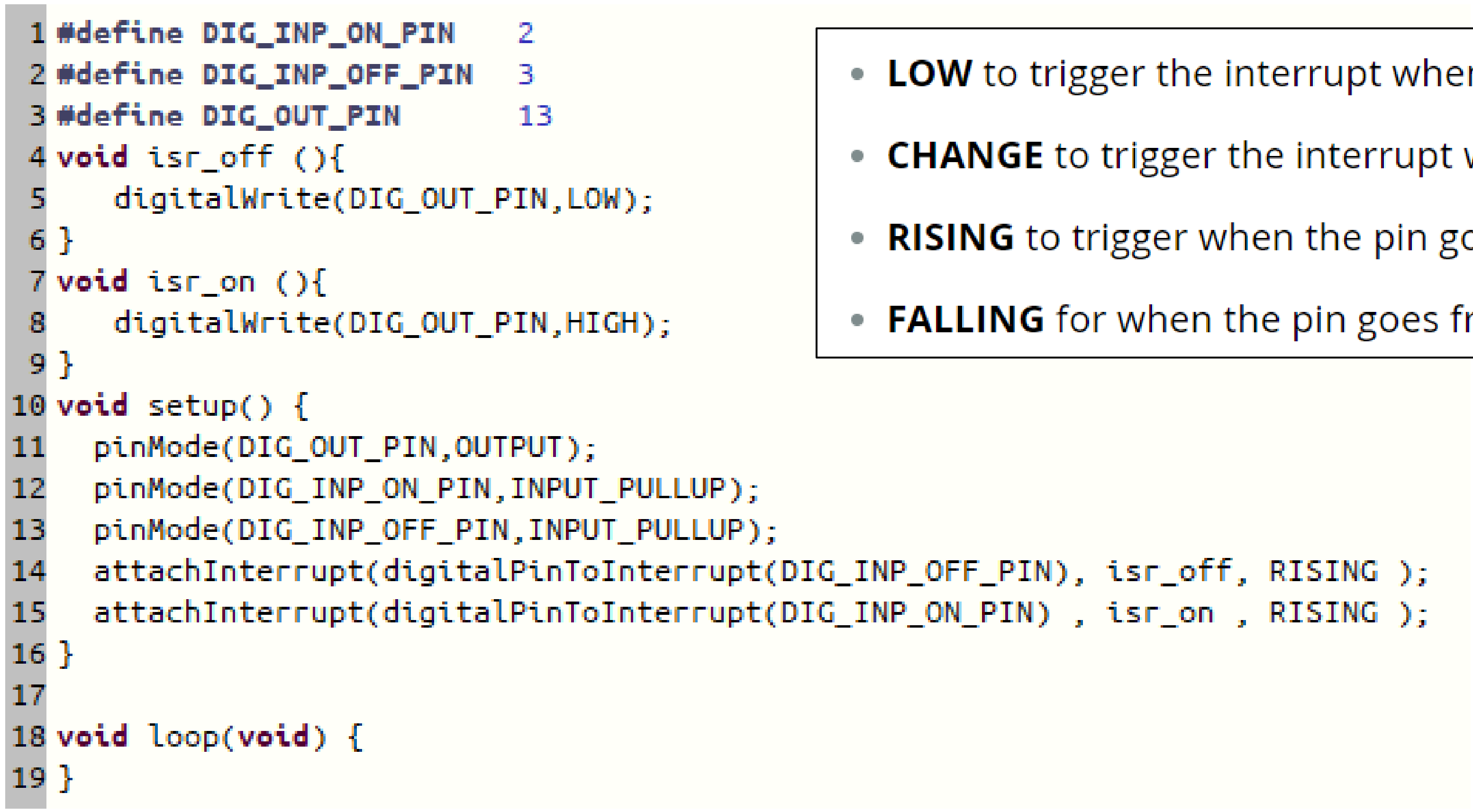
Time Div 20.00 ms
Time Pos 0.000 ps
Volt Div 1.000 V

Interrupts
(Digital Pins With Interrupts)

```
#define DIG_INP_ON_PIN    2
#define DIG_INP_OFF_PIN   3
#define DIG_OUT_PIN       13
void isr_off (){
   digitalWrite(DIG_OUT_PIN,LOW);
}
void isr_on (){
   digitalWrite(DIG_OUT_PIN,HIGH);
}
void setup() {
  pinMode(DIG_OUT_PIN,OUTPUT);
  pinMode(DIG_INP_ON_PIN,INPUT_PULLUP);
  pinMode(DIG_INP_OFF_PIN,INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(DIG_INP_OFF_PIN), isr_off, RISING );
  attachInterrupt(digitalPinToInterrupt(DIG_INP_ON_PIN) , isr_on , RISING );
}

void loop(void) {
}
```



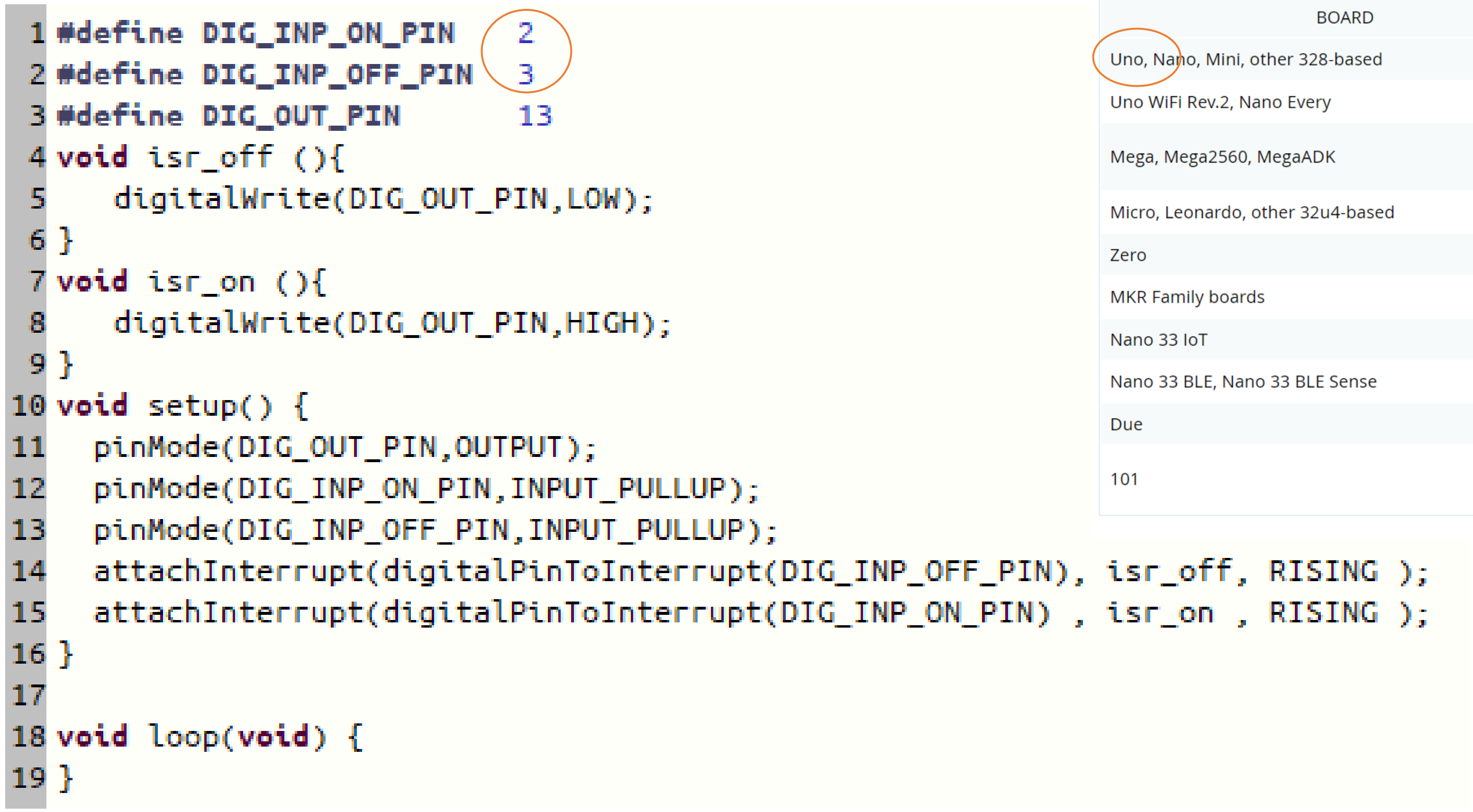
Interrupts
(Digital Pins With Interrupts) – Cont.



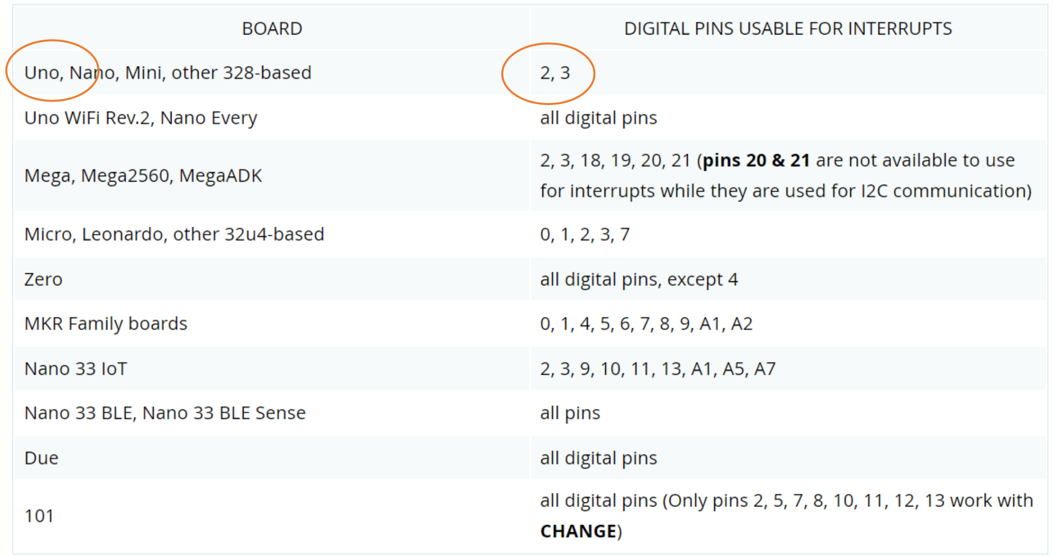
```
#define DIG_INP_ON_PIN    2
#define DIG_INP_OFF_PIN   3
#define DIG_OUT_PIN       13
void isr_off (){
   digitalWrite(DIG_OUT_PIN,LOW);
}
void isr_on (){
   digitalWrite(DIG_OUT_PIN,HIGH);
}
void setup() {
  pinMode(DIG_OUT_PIN,OUTPUT);
  pinMode(DIG_INP_ON_PIN,INPUT_PULLUP);
  pinMode(DIG_INP_OFF_PIN,INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(DIG_INP_OFF_PIN), isr_off, RISING );
  attachInterrupt(digitalPinToInterrupt(DIG_INP_ON_PIN) , isr_on , RISING );
}

void loop(void) {
}
```

- **LOW** to trigger the interrupt whenever the pin is low,
- **CHANGE** to trigger the interrupt whenever the pin changes value
- **RISING** to trigger when the pin goes from low to high,
- **FALLING** for when the pin goes from high to low.

Interrupts
(Digital Pins With Interrupts) – Cont.



```
#define DIG_INP_ON_PIN    2
#define DIG_INP_OFF_PIN   3
#define DIG_OUT_PIN       13
void isr_off (){
   digitalWrite(DIG_OUT_PIN,LOW);
}
void isr_on (){
   digitalWrite(DIG_OUT_PIN,HIGH);
}
void setup() {
  pinMode(DIG_OUT_PIN,OUTPUT);
  pinMode(DIG_INP_ON_PIN,INPUT_PULLUP);
  pinMode(DIG_INP_OFF_PIN,INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(DIG_INP_OFF_PIN), isr_off, RISING );
  attachInterrupt(digitalPinToInterrupt(DIG_INP_ON_PIN) , isr_on , RISING );
}

void loop(void) {
}
```



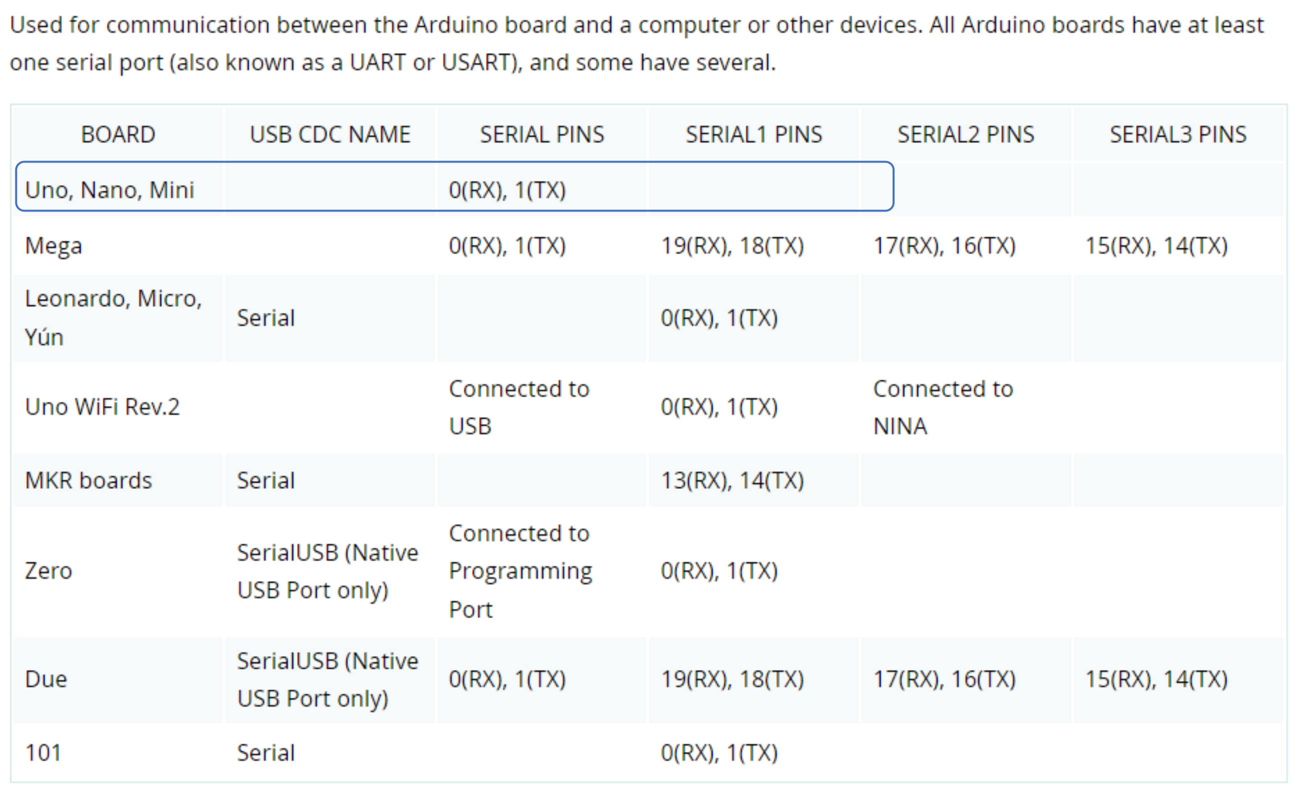
BOARD	DIGITAL PINS USABLE FOR INTERRUPTS
Uno, Nano, Mini, other 328-based	2, 3
Uno WiFi Rev.2, Nano Every	all digital pins
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21 (**pins 20 & 21** are not available to use for interrupts while they are used for I2C communication)
Micro, Leonardo, other 32u4-based	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**)

Communications

- UART: universal asynchronous receiver-transmitter
- I^{2}C: Inter-Integrated Circuit
- SPI: Serial Peripheral Interface

Communications

UART/Serial



Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART), and some have several.

BOARD	USB CDC NAME	SERIAL PINS	SERIAL1 PINS	SERIAL2 PINS	SERIAL3 PINS
Uno, Nano, Mini		0(RX), 1(TX)			
Mega		0(RX), 1(TX)	19(RX), 18(TX)	17(RX), 16(TX)	15(RX), 14(TX)
Leonardo, Micro, Yún	Serial		0(RX), 1(TX)		
Uno WiFi Rev.2		Connected to USB	0(RX), 1(TX)	Connected to NINA	
MKR boards	Serial		13(RX), 14(TX)		
Zero	SerialUSB (Native USB Port only)	Connected to Programming Port	0(RX), 1(TX)		
Due	SerialUSB (Native USB Port only)	0(RX), 1(TX)	19(RX), 18(TX)	17(RX), 16(TX)	15(RX), 14(TX)
101	Serial		0(RX), 1(TX)		

Communications

UART/Serial (cont.)

Arduino Mega example:

```
// Arduino Mega using all four of its Serial ports
// (Serial, Serial1, Serial2, Serial3),
// with different baud rates:

void setup() {
  Serial.begin(9600);
  Serial1.begin(38400);
  Serial2.begin(19200);
  Serial3.begin(4800);

  Serial.println("Hello Computer");
  Serial1.println("Hello Serial 1");
  Serial2.println("Hello Serial 2");
  Serial3.println("Hello Serial 3");
}
void loop() {}
```

- `Serial.print(78, BIN)` gives "1001110"
- `Serial.print(78, OCT)` gives "116"
- `Serial.print(78, DEC)` gives "78"
- `Serial.print(78, HEX)` gives "4E"
- `Serial.print(1.23456, 0)` gives "1"
- `Serial.print(1.23456, 2)` gives "1.23"
- `Serial.print(1.23456, 4)` gives "1.2346"

Communications

UART/Serial (cont.)

```
int incomingByte = 0; // for incoming serial data

void setup() {
  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}

void loop() {
  // send data only when you receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();

    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte, DEC);
  }
}
```

Communications

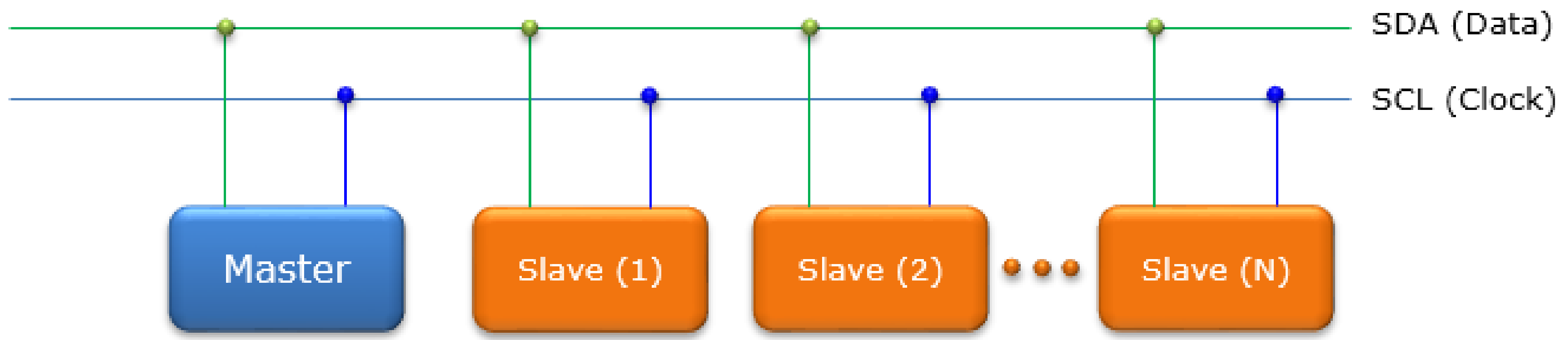
UART/Serial (cont.)

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

void loop() {
  Serial.write(45); // send a byte with the value 45

  int bytesSent = Serial.write("hello");  //send the string "hello" and return the length of the string.
}
```

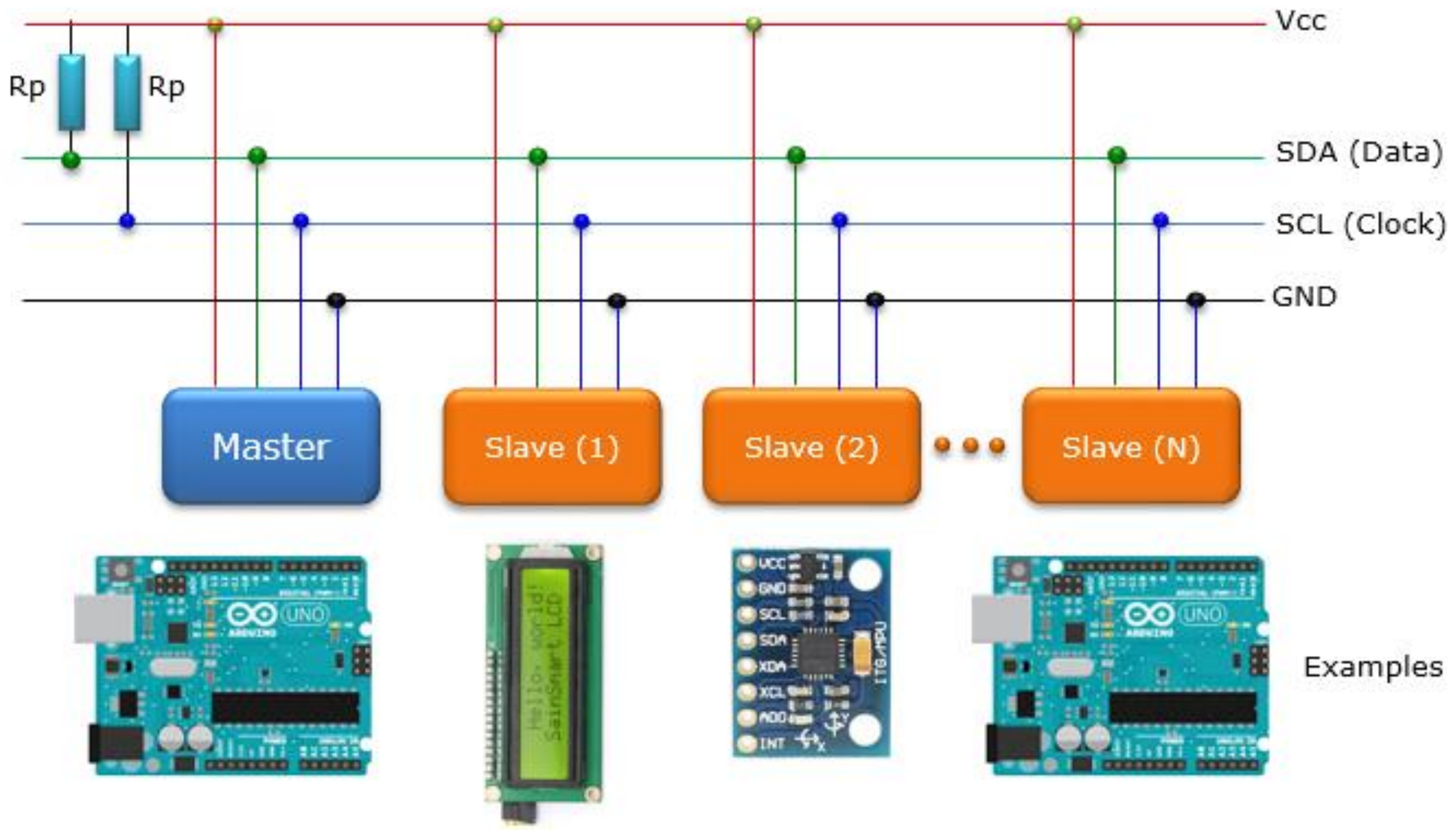
(I^2C) Inter Integrated Circuit

- I2C or I^2C came from the term Inter Integrated Circuit. You see two 'I's and one 'C'
- I2C is implemented by only Two Wires (SDA, SCL), it is also called TWI (Two Wire Interface).
- It is a kind of serial communication technology which is originally designed for exchanging data between multiple IC chips.
- This is very simple communication mechanism, and It is very smartly designed. It requires only two lines as. One of the line is used for sending Clock signal and the other line is to send/receive data.
- Also, these two lines can be shared by multiple devices. This communication works as Master-Slave mode. One Master can control / communicate multiple Slaves.

(I^2C) Inter Integrated Circuit (Cont.)



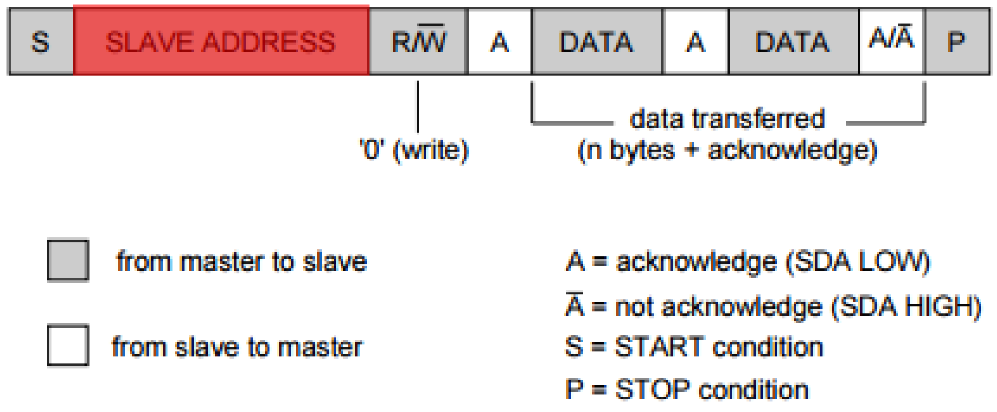
(I^2C) Inter Integrated Circuit (Cont.)



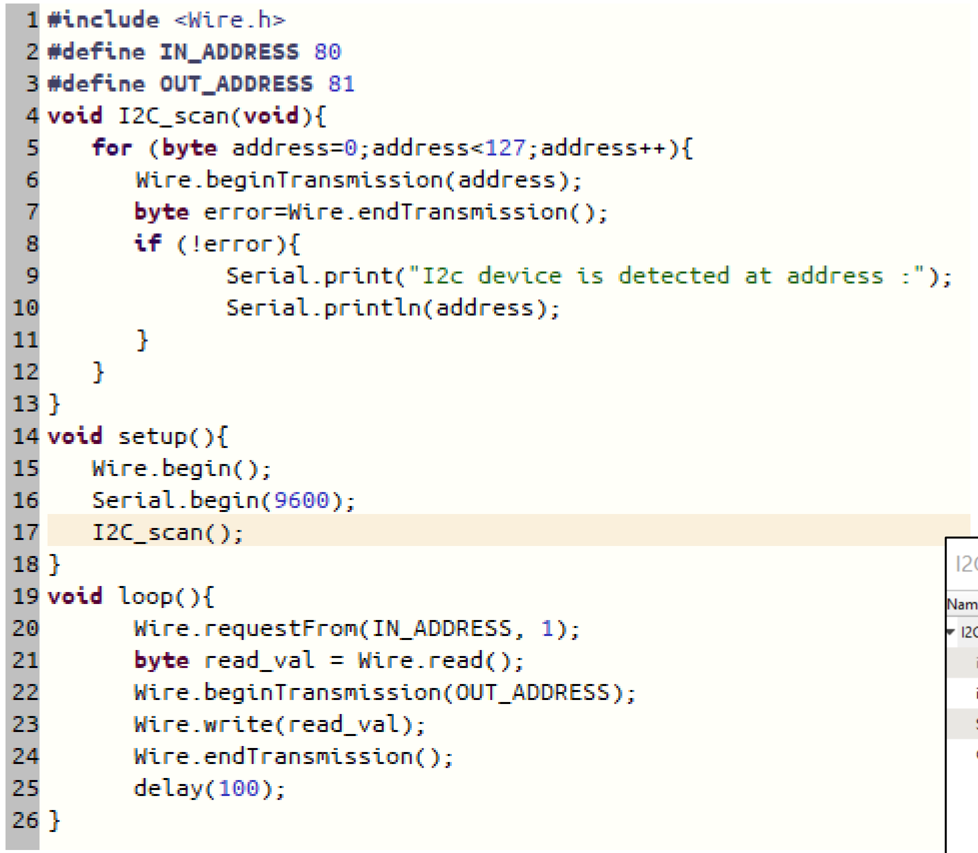
(I^2C) Inter Integrated Circuit (Cont.)

How can a Master send data to a specific Slave ?

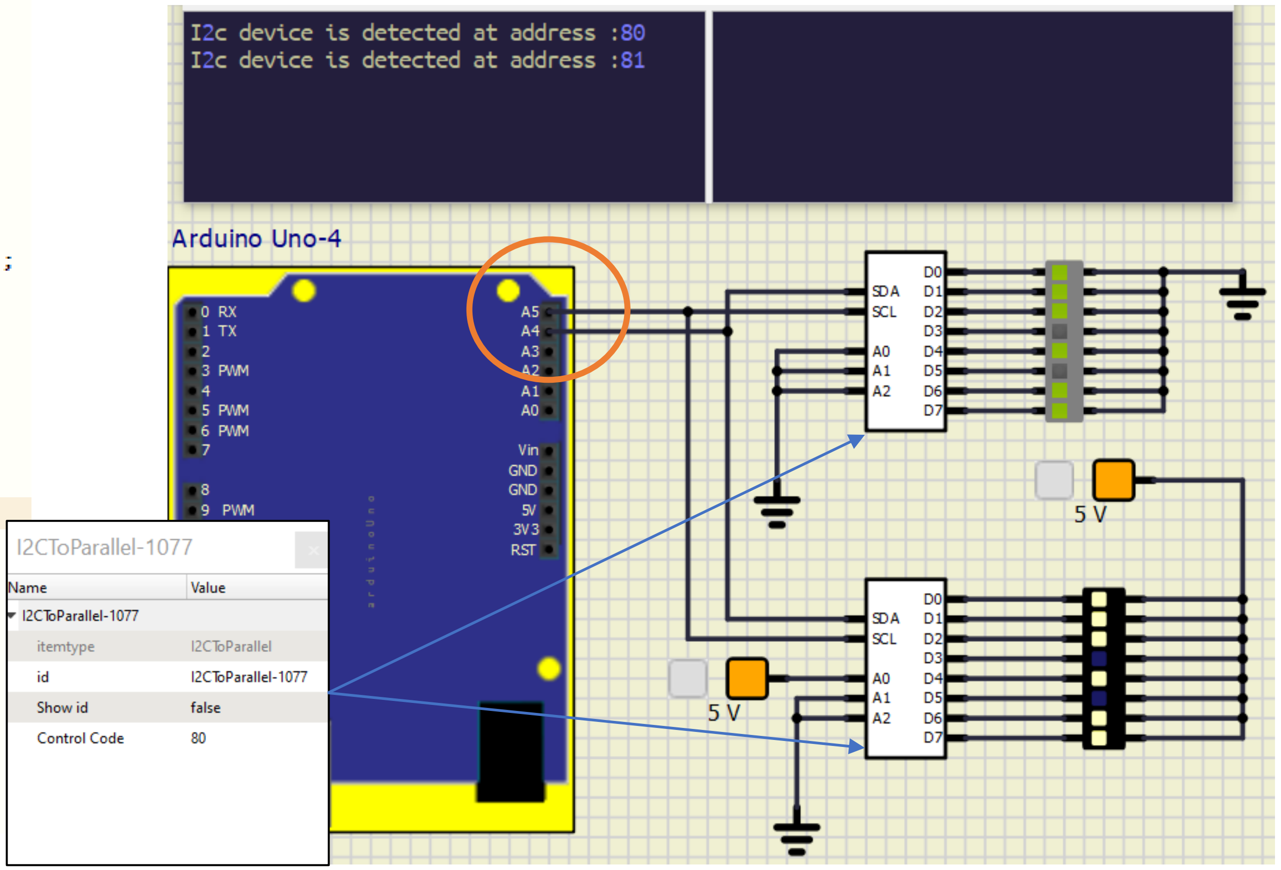
Step	Direction	Message
1	Master -> Slave	Start
2	Master -> Slave	Slave Address
3	Master <- Slave	Ack
4	Master -> Slave	Data
5	Master <- Slave	Ack
6	Master -> Slave	Stop



(I^2C) Example(Port Expander)



```
#include <Wire.h>
#define IN_ADDRESS 80
#define OUT_ADDRESS 81
void I2C_scan(void){
   for (byte address=0;address<127;address++){
      Wire.beginTransmission(address);
      byte error=Wire.endTransmission();
      if (!error){
             Serial.print("I2c device is detected at address :");
             Serial.println(address);
      }
   }
}
void setup(){
   Wire.begin();
   Serial.begin(9600);
   I2C_scan();
}
void loop(){
      Wire.requestFrom(IN_ADDRESS, 1);
      byte read_val = Wire.read();
      Wire.beginTransmission(OUT_ADDRESS);
      Wire.write(read_val);
      Wire.endTransmission();
      delay(100);
}
```

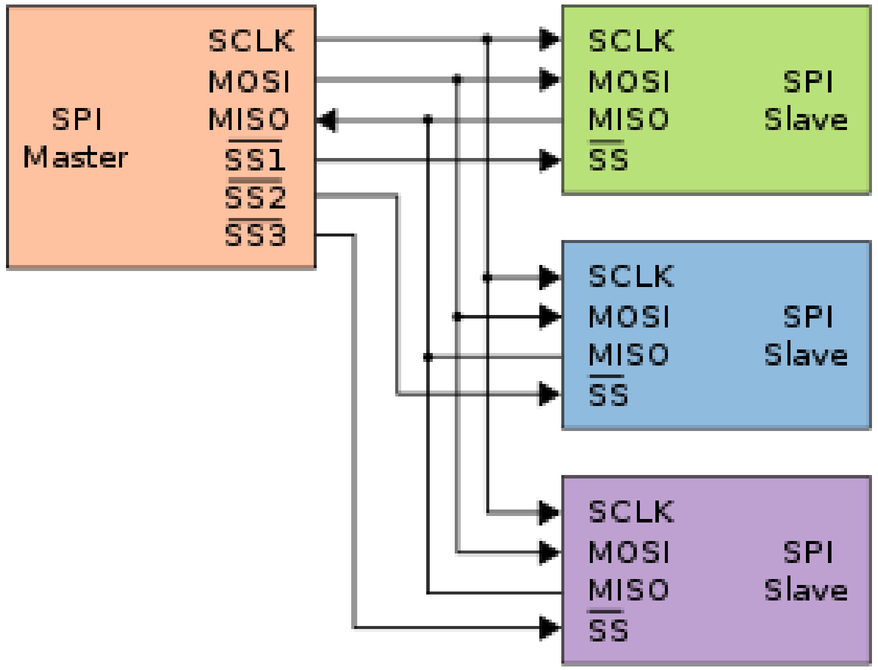


```
I2c device is detected at address :80
I2c device is detected at address :81
```

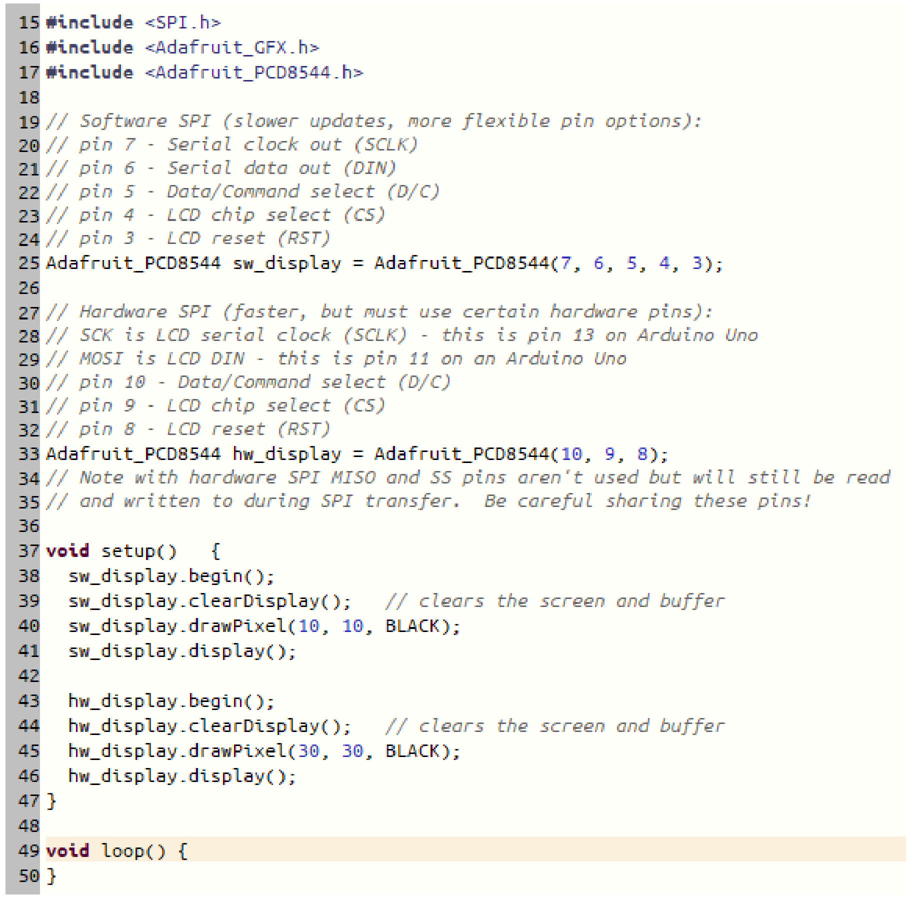
Name	Value
I2CToParallel-1077	
itemtype	I2CToParallel
id	I2CToParallel-1077
Show id	false
Control Code	80

(SPI) Serial Peripheral Interface

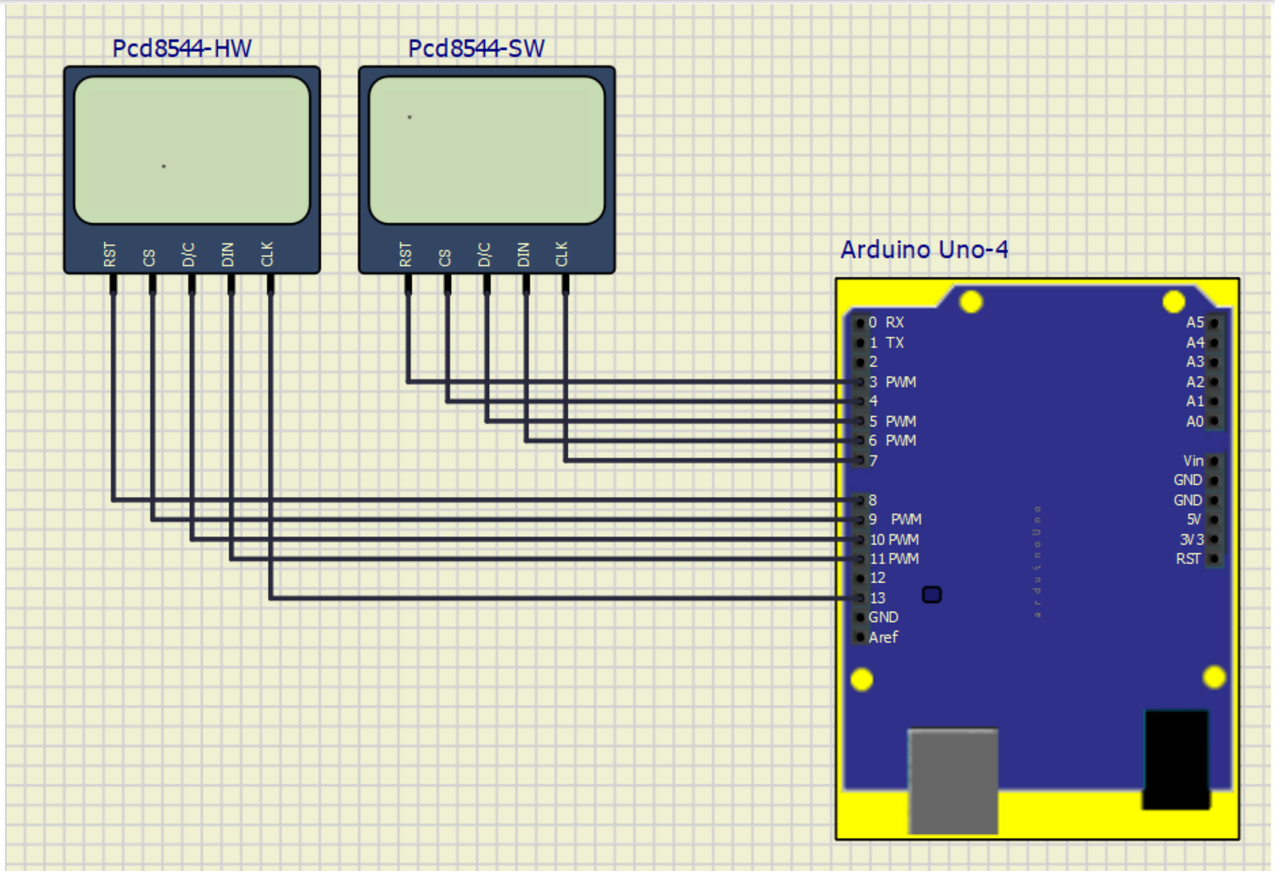
- Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.
- With an SPI connection there is always one master device (usually a microcontroller) which controls the peripheral devices. Typically, there are three lines common to all the devices:
    - MISO (Master In Slave Out) - The Slave line for sending data to the master,
    - MOSI (Master Out Slave In) - The Master line for sending data to the peripherals,
    - SCK (Serial Clock) - The clock pulses which synchronize data transmission generated by the master and one line specific for every device:
    - SS (Slave Select) - the pin on each device that the master can use to enable and disable specific devices.
- When a device's Slave Select pin is low, it communicates with the master. When it's high, it ignores the master. This allows you to have multiple SPI devices sharing the same MISO, MOSI, and CLK lines.
- Arduino UNO (SCLCK : 13, MISO : 12, MOSI0 : 11, SS : 10)



(SPI) Example (LCD SW/HW)



```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>

// Software SPI (slower updates, more flexible pin options):
// pin 7 - Serial clock out (SCLK)
// pin 6 - Serial data out (DIN)
// pin 5 - Data/Command select (D/C)
// pin 4 - LCD chip select (CS)
// pin 3 - LCD reset (RST)
Adafruit_PCD8544 sw_display = Adafruit_PCD8544(7, 6, 5, 4, 3);

// Hardware SPI (faster, but must use certain hardware pins):
// SCK is LCD serial clock (SCLK) - this is pin 13 on Arduino Uno
// MOSI is LCD DIN - this is pin 11 on an Arduino Uno
// pin 10 - Data/Command select (D/C)
// pin 9 - LCD chip select (CS)
// pin 8 - LCD reset (RST)
Adafruit_PCD8544 hw_display = Adafruit_PCD8544(10, 9, 8);
// Note with hardware SPI MISO and SS pins aren't used but will still be read
// and written to during SPI transfer.  Be careful sharing these pins!

void setup()   {
  sw_display.begin();
  sw_display.clearDisplay();   // clears the screen and buffer
  sw_display.drawPixel(10, 10, BLACK);
  sw_display.display();

  hw_display.begin();
  hw_display.clearDisplay();   // clears the screen and buffer
  hw_display.drawPixel(30, 30, BLACK);
  hw_display.display();
}

void loop() {
}
```



Difference Between I2C vs SPI

What is I2C Protocol?
I2C stands for Inter-Integrated Circuit. I2C is a simple two-wire serial protocol used to communicate between two devices or chips in an embedded system. I2C has two lines SCL and SDA, SCL is used for clock and SDA is used for data.

What is SPI Protocol?
SPI stands for Serial Peripheral interface. SPI is a four-wire serial communication protocol. SPI follows master-slave architecture. The four lines of SPI are MOSI, MISO, SCL and SS. SCL is a serial clock that is used for entire data communication. Slave Select(SS) is used to select the slave. Master out Slave In (MOSI) is the output data line from the master and Master in Slave out(MISO) is the input data line for the Master.

Advantage I2C:
● Easy device add on : It is easy to add new slave devices into the bus. Just add the new device without adding a new slave select unlike SPI.

Advantage SPI :
● Speed : SPI can support up to 10MB/s however I^{2}C in the ultra-fast mode can support only 5MB/s.