Serial Communication via I2C/I²C and SPI

I2C: Inter-Integrated circuit

SPI: Serial Peripheral Interface

Acknowledgements: The slides of the following section is created based on the slides from Eugene Ho as well as various materials from the web

Serial Communication via I2C/I²C

I2C: **I**nter-**I**ntegrated **C**ircuit

Acknowledgements: The slides of the following section is created based on the slides from Eugene Ho as well as various materials from the web

Introduction

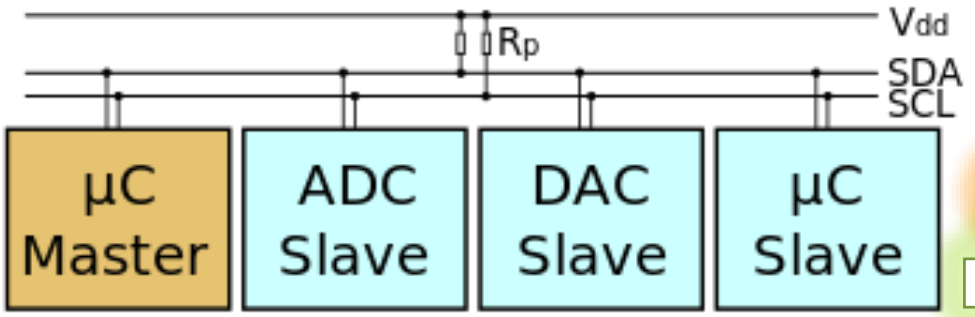
- I2C and SPI
  - Serial communication protocols
    - Bit-wise communication
  - Meant for short distances “inside the box”
  - Low complexity
  - Low cost
  - Low speed (a few Mbps at the fastest)

What is I2C or I²C?

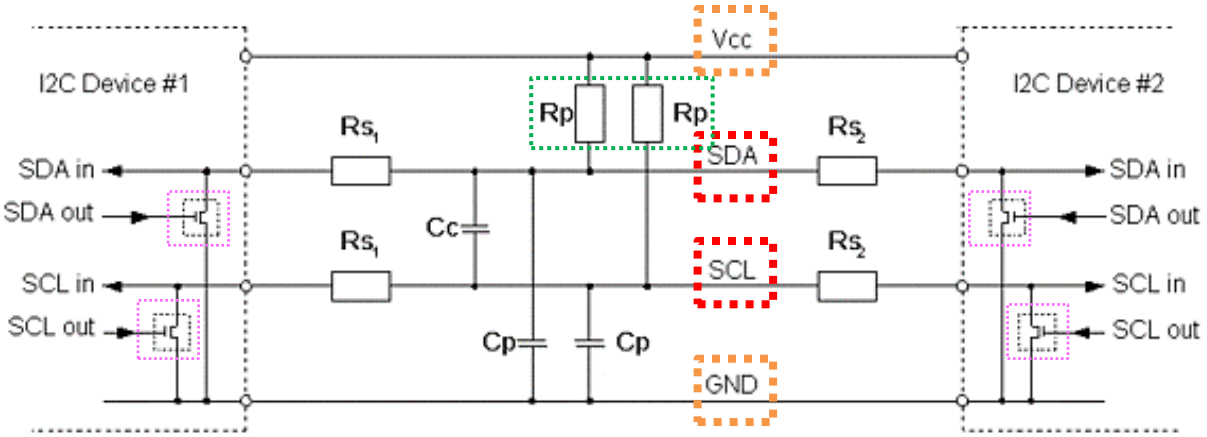
- Shorthand for an "**I**nter-**I**ntegrated **C**ircuit" bus
- Developed by Philips Semiconductor for TV sets in the 1980's
- Multi-master, multi-slave, single-ended, half-duplex, synchronous, serial computer bus
- I2C devices include
  - TYPE I: low-speed storage and sensors (i.e., 低速處理週邊裝置)
    - EEPROMs, thermal sensors, and real-time clocks
  - TYPE II: signal processing devices (i.e., 高速處理但低傳輸需求週邊裝置)
    - e.g. RF tuners, video decoders/encoders, and audio processors.

I2C Bus Configuration

- 2-wire serial bus – Serial data (**SDA**) and Serial clock (**SCL**)
- I2C bus has three speeds:
  - Slow (under 100 Kbps), Fast (400 Kbps), High-speed (3.4 Mbps)
    - 5 MHz Ultra Fast-mode (UFm) for new **U**SDA and **U**SCL lines Limited to about 10 feet for moderate speeds
- No **chip select** or **arbitration** logic required
- Lines pulled high via resistors, pulled down via **open-drain/open-collector** drivers (**wired-AND**) by devices
- It supports 2^7-16=112
  - 7-bit address



I2C Typical Setup



VCC	I2C supply voltage, typically ranging from 1.2 V to 5.5 V
GND	Common ground
SDA	Serial data (I2C data line)
SCL	Serial clock (I2C clock line)
Rp	Pull-up resistance (a.k.a. I2C termination)
Rs	Serial resistance
Cp	Wire capacitance
Cc	Cross channel capacitance

Source: http://www.i2c-bus.org

I2C Mater vs. Slave nodes

- The bus has two roles for nodes:
  - Master node — node that **generates the clock (SCL) and initiates communication (SDA) with slaves**
  - Slave node — node that **receives the clock and responds** when addressed by the master
- The bus is a **multi-master bus**
  - which means any number of master nodes can be present.
  - master and slave roles may be **changed** between messages (after a STOP is sent).

Simplified I2C Protocol

- **Master** device issues a **start** condition.
  - This condition informs all the slave devices to listen on the serial data line for their respective address.
- **Master** device sends the address of the target slave device and a read/write flag.
- **Slave** device with the matching address responds with an acknowledgment signal.
- Communication proceeds between the **master** and the **slave** on the data bus.
  - Both master and slave can receive or transmit data depending on whether the communication is a read or write.
  - Transmitter sends 8 bits data to receiver, which replies a 1 bit ACK.
- When the communication is complete, master issues a **stop** condition indicating that everything is done.

Detailed I2C Protocol

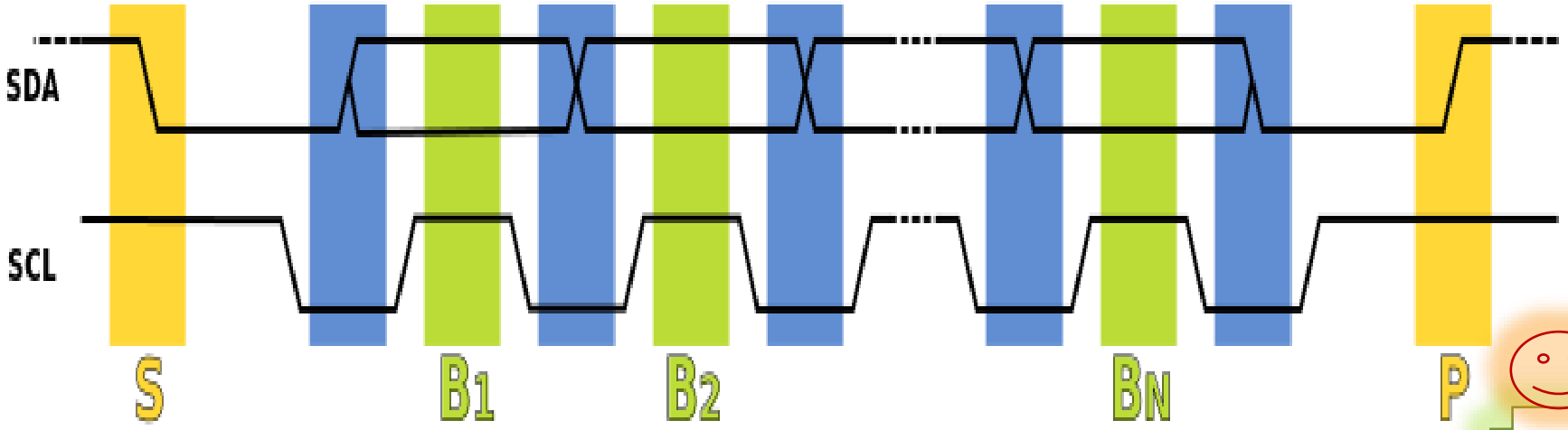
- 1. **Master** sends start condition (S) and controls the clock signal
- 2. **Master** sends a unique 7-bit slave device address
- 3. **Master** sends read/write bit (R/W) : 0 - slave receive, 1 - slave transmit
- 4. Receiver sends acknowledge bit (ACK)
- 5. Transmitter (slave or master) transmits 1 byte of data

Detailed I2C Protocol (cont.)

- 6. Receiver issues an ACK bit for the byte received
- 7. Repeat 5 and 6 if more bytes need to be transmitted.
- 8.a) For write transaction (master transmitting), master issues stop condition (P) after last byte of data.
- 8.b) For read transaction (master receiving), master does not acknowledge final byte, just issues stop condition (P) to tell the slave the transmission is done

Timing diagram

- Data Transfer is initiated with a **START** bit (S) signaled by SDA being pulled low while SCL stays high.
- SDA sets the 1st data bit level while keeping SCL low (during blue bar time.)
- The data is sampled (received) when SCL rises (green) for the first bit (B1).
- This process repeats, SDA transitioning while SCL is low, and the data being read while SCL is high (B2, Bn).
- A **STOP** bit (P) is signaled when SDA is pulled high while SCL is high.



I2C Enhancement Features

- "Clock stretching" – when the slave (receiver) needs more time to process a bit, it can pull SCL low.
  - The master waits until the slave has released SCL before sending the next bit.
- "General call" broadcast – addresses every device on the bus
- 10-bit extended addressing for new designs. 7-bit addresses all exhausted

I2C Tradeoffs

- Advantages:
  - Good for communication with on-board devices that are accessed occasionally.
  - Easy to link multiple devices because of addressing scheme
  - Cost and complexity do not scale up with the number of devices

- Disadvantages:
  - The complexity of supporting software components can be higher than that of competing schemes (for example, SPI).

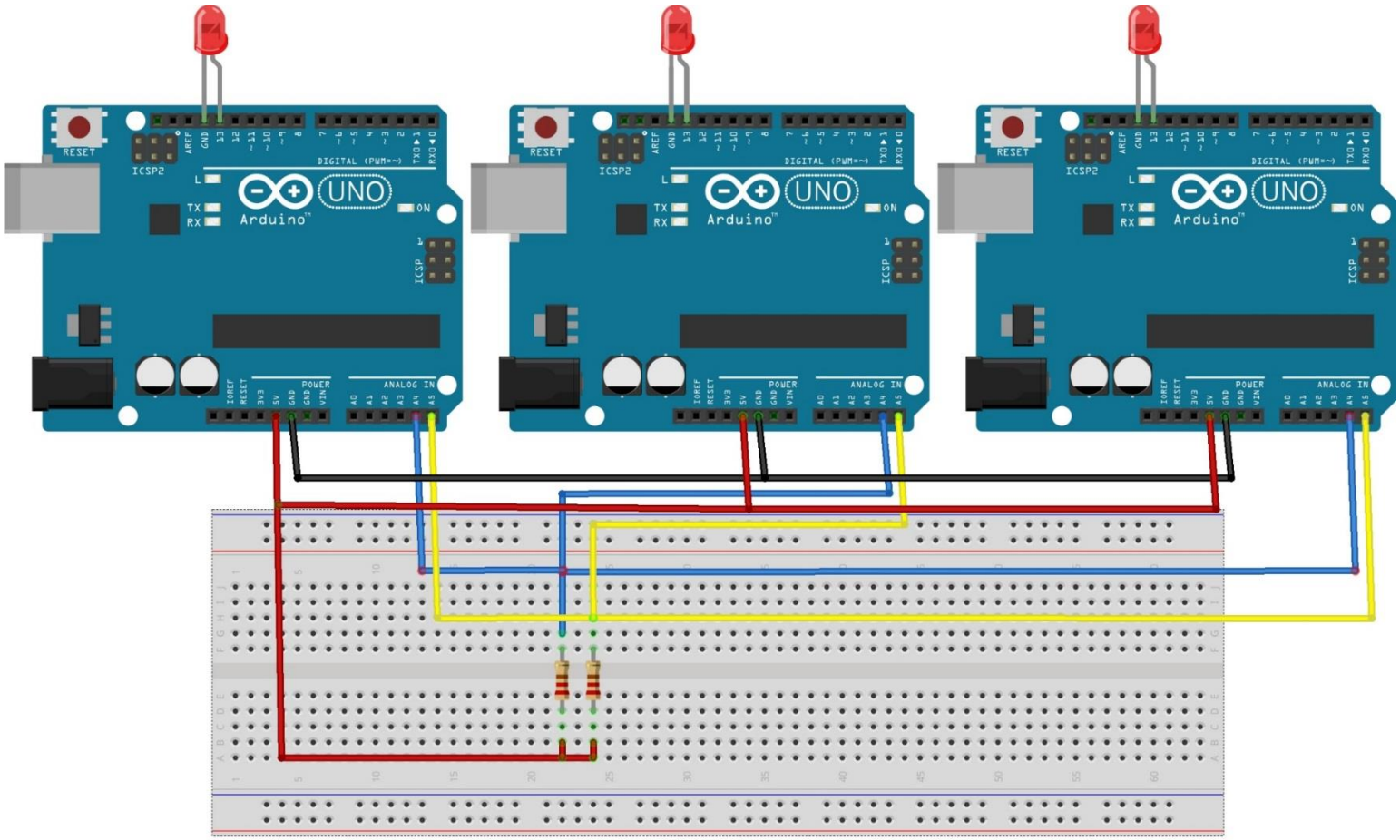
Arduino Wire library

- This library allows you to communicate with I2C / TWI（Two Wire Interface）devices.
  - <Wire.h>

- As a reference the table below shows where I2C / TWI pins are located on various Arduino boards.

Board	I2C / TWI pins
Uno, Ethernet	A4 (SDA), A5 (SCL)
Mega2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1, SCL1

Three Arduino connected using I²C



begin()

- Wire.begin()
- Wire.begin(address)
- 說明

  啟動Wire library並加入I2C bus.
- 參數

  address: the 7-bit slave address (optional);

  if not specified, join the bus as a master.

onReceive()

- Wire.onReceive(receiveEvent)
- 說明

  註冊一個自定義函數，當裝置收到來自 別的位址的訊息時自動呼叫這個自定義函數。
- 參數

  receiveEvent: 當接收到資料時會被呼叫的自定義**函數**;
  這個函數必須帶一個 int 的參數而且沒有返回值。
  例如: void receiveEvent(int numBytes)

beginTransmission() endTransmission() write()

- Wire.beginTransmission(address)
- Wire.write(value)
- Wire.endTransmission()

- 說明

  首先設定要傳送的地址 再來傳送的資料 以及 停止傳送。

- 參數

  address: the 7-bit address of the device to transmit to

  value: a value to send as a single byte

read()

- Wire.read()
- 說明

  讀取一個從別的裝置所傳送的位元組.

requestFrom()

- Wire. requestFrom()
- 說明
  - Used by the master to request bytes from a slave device. The bytes may then be retrieved with the available() and read() functions.
- **Syntax**
  - Wire.requestFrom(address, quantity)
    Wire.requestFrom(address, quantity, stop)
    - address: the 7-bit address of the device to request bytes from
    - quantity: the number of bytes to request
    - stop : boolean. true will send a stop message after the request, releasing the bus. false will continually send a restart after the request, keeping the connection active.

requestFrom()

- Wire. requestFrom() Example
- On *Address_1* device

```
Wire.requestFrom(Address_2, 6);  // loop()
While( wire.available() ){
incomingByte = wire.read();
Serial.print(incomingByte)
}
```

- On *Address_2* device

```
Wire.onRequest(requestEvent);  // setup()
Void requestEvent{wire.write( “hello\n” )}  //len=6
```

Sample Codes on Multi Slaves

```
#include <Wire.h>

const int ADDRESS_1= n ;    // n = 1 , 2 , 3
const int ledPin = 13;
char incomingByte = 0;// 收到從ADDRESS_1 的字元
const int delay_ms = 1000;  //延遲1000毫秒 等於 1秒
```

Sample Codes on Multi Slaves (cont.)

```
void setup() {
 Wire.begin(ADDRESS_1); //slave node
 Wire.onReceive( receiveEvent );

 pinMode( ledPin, OUTPUT );  // ledPin = 13
} // setup()
```

Sample Codes on Multi Slaves (cont.)

```
void receiveEvent(int howMany) {
  // receive one byte from others
  incomingByte = Wire.read();

  // turn on or off LED according to the received data
  digitalWrite( ledPin, incomingByte );
} // receiveEvent()
```

Sample Codes on Multi Slaves (cont.)

```
void loop() {

  if ( incomingByte == HIGH ) {
    delay(delay_ms);
    ADDRESS_TO(ADDRESS_2, HIGH );
  }
  else {
    delay(delay_ms);
    ADDRESS_TO(ADDRESS_2, LOW );
  }

} // loop()
```

Sample Codes on Multi Slaves (cont.)

```
void ADDRESS_TO( int address, byte value ) {
  // send data to ADDRESS_2
  Wire.beginTransmission( address );
  Wire.write( value );
  Wire.endTransmission();
} // ADDRESS_TO()
```

Useful References

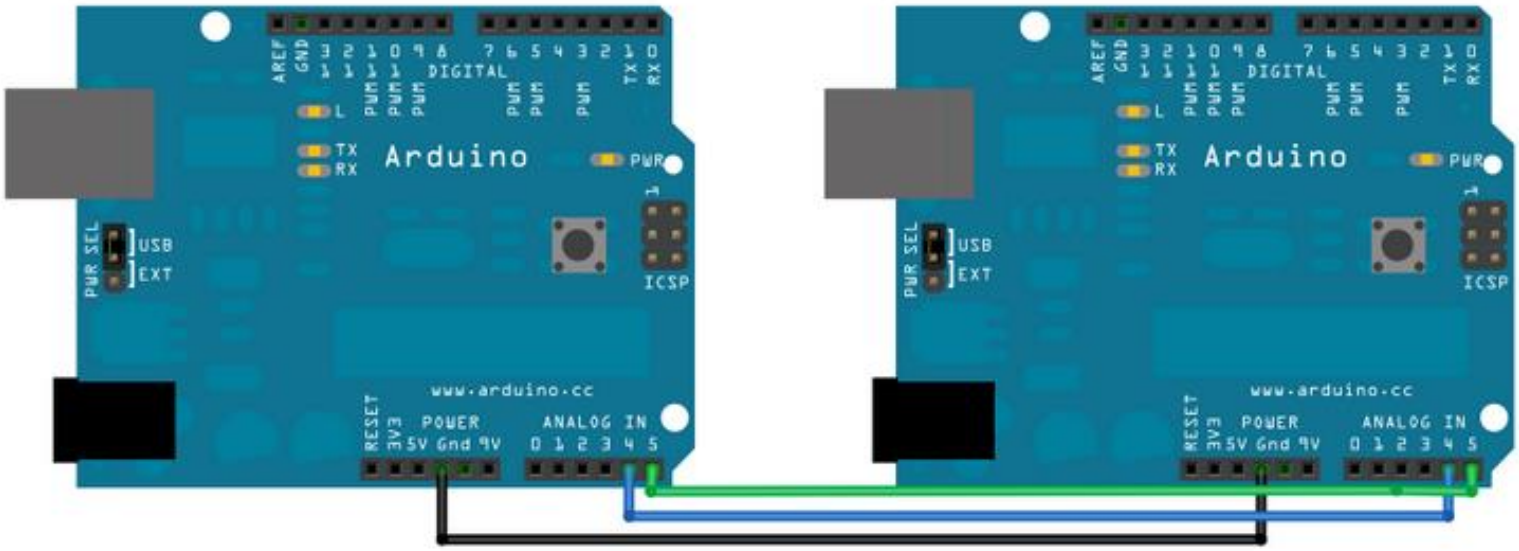
Examples

- Digital Potentiometer: Control an Analog Devices AD5171 Digital Potentiometer.
- Master Reader/Slave Writer: Program two Arduino boards to communicate with one another in a Master Reader/Slave Sender configuration via the I2C.
- Master Writer/Slave receiver:Program two Arduino boards to communicate with one another in a Master Writer/Slave Receiver configuration via the I2C.
- SFR Ranger Reader: Read an ultra-sonic range finder interfaced via the I2C.
- Add SerCom : Adding mores Serial interfaces to SAMD microcontrollers.

https://www.arduino.cc/en/Reference/Wire

Master Reader/Slave Sender

- Arduino 1, the Master, is programmed to receive 6 bytes of data every half second to a uniquely addressed Slave.
- Once that message is received, it can then be viewed in the master board's serial monitor window opened on the USB connected computer running the Arduino Software (IDE).



Code for Master Reader - Program for Arduino 1

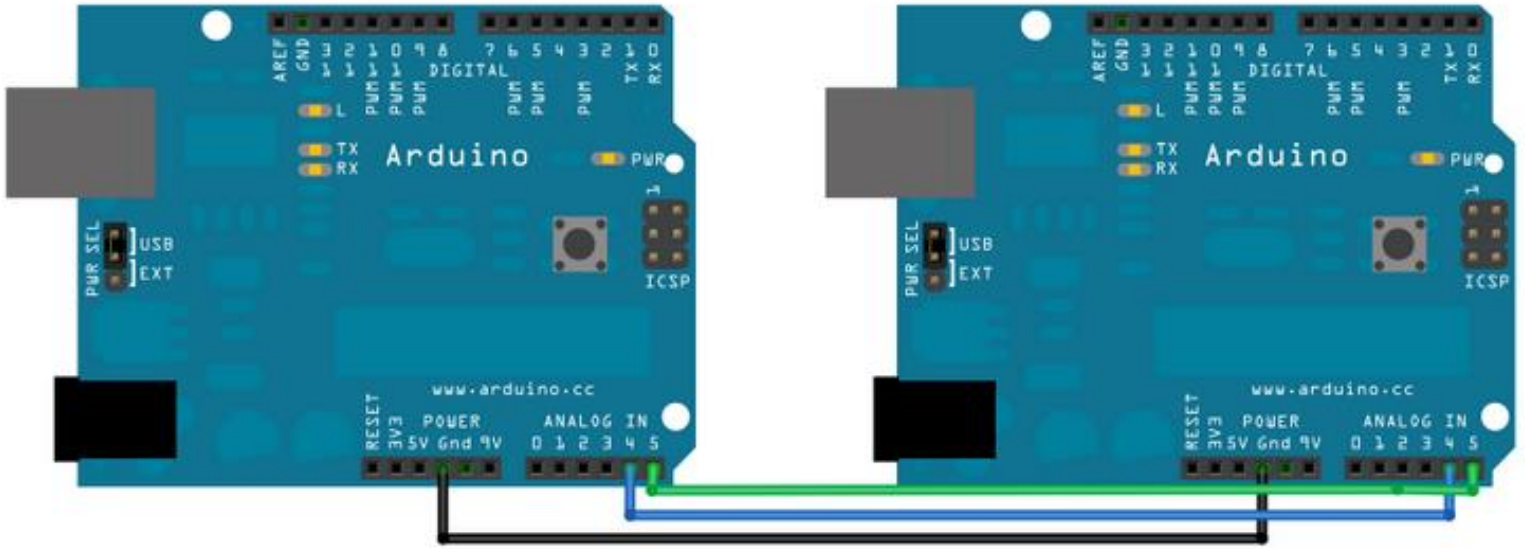
```
#include <Wire.h>

void setup() {
  Wire.begin();        // join i2c bus (address optional for master)
  Serial.begin(9600);  // start serial for output
}

void loop() {
  Wire.requestFrom(8, 6);    // request 6 bytes from slave device #8

  while (Wire.available()) { // slave may send less than requested
    char c = Wire.read(); // receive a byte as character
    Serial.print(c);         // print the character
  }

  delay(500);
}
```

Code for Slave Sender - Program for Arduino 2

```
#include <Wire.h>

void setup() {
  Wire.begin(8);                // join i2c bus with address #8
  Wire.onRequest(requestEvent); // register event
}

void loop() {
  delay(100);
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent() {
  Wire.write("hello "); // respond with message of 6 bytes
  // as expected by master
}
```

Master Writer/Slave Receiver

- Arduino 1, the Master, is programmed to send 6 bytes of data every half second to a uniquely addressed Slave.
- Once that message is received, it can then be viewed in the Slave board's serial monitor window opened on the USB connected computer running the Arduino Software (IDE).



Master Writer Code - Program for Arduino 1

```
#include <Wire.h>

void setup() {
  Wire.begin(); // join i2c bus (address optional for master)
}

byte x = 0;

void loop() {
  Wire.beginTransmission(8); // transmit to device #8
  Wire.write("x is ");        // sends five bytes
  Wire.write(x);              // sends one byte
  Wire.endTransmission();    // stop transmitting

  x++;
  delay(500);
}
```

Slave Receiver Code –Arduino 2

```
#include <Wire.h>

void setup() {
  Wire.begin(8);                // join i2c bus with address #8
  Wire.onReceive(receiveEvent); // register event
  Serial.begin(9600);           // start serial for output
}

void loop() {
  delay(100);
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany) {
  while (1 < Wire.available()) { // loop through all but the last
    char c = Wire.read(); // receive byte as a character
    Serial.print(c);         // print the character
  }
  int x = Wire.read();    // receive byte as an integer
  Serial.println(x);         // print the integer
}
```

Practice

- Basic
  - A版有個按鈕，B版有個LED，A版按下按鈕，B版LED燈亮起，放開則燈滅。
  - 兩個版子可以主動互傳簡易訊息
    - A主動傳給B：在B的serial monitor可以看到”Hello, this is board A”
    - 休息五秒
    - B主動傳給A：在A的serial monitor可以看到”Hello, this is board B”
- Advanced (一定要做)
  - 停車場+樂透機系統
  - 影片已公告

Serial Communication via SPI

SPI: Serial Peripheral Interface

Acknowledgements: The slides of the following section is created based on the slides from Eugene Ho as well as various materials from the web

What is SPI?

- Shorthand for "**S**erial **P**eripheral **I**nterface"
- Defined by **Motorola** on the MC68HCxx line of microcontrollers
- Generally faster than I2C, capable of several Mbps

- Applications:
  - Like I2C, used in EEPROM, Flash, and real time clocks
- Better suited for "data streams", i.e. ADC converters
- **Full duplex** capability, i.e. communication between a codec and digital signal processor

SPI Bus Configuration

- Synchronous serial data link operating at full duplex
- Master/slave relationship
- 2 data signals:
  - MOSI – master data output, slave data input
  - MISO – master data input, slave data output
- 2 control signals:
  - SCLK – clock
  - /SS – slave select (no addressing)

SPI vs. I2C

- For point-to-point, SPI is simple and efficient
  - Less overhead than I2C due to lack of addressing, plus SPI is full duplex.
- For multiple slaves, each slave needs separate slave select signal
  - More effort and more hardware than I2C

SPI Protocol

- 2 Parameters, Clock Polarity (CPOL) and Clock Phase (CPHA), determine the active edge of the clock

- Master and slave must agree on parameter pair values in order to communicate

CPOL	CPHA	Active edge
0	0	Rising
0	1	Falling
1	0	Falling
1	1	Rising

SPI Protocol (cont.)

- Hardware realization is usually done with a simple shift register
- SPI interface defines only the communication lines and the clock edge
- There is no specified flow control! No acknowledgement mechanism to confirm receipt of data

AVR Support for SPI

- Supported by all AVR 8-bit µC except ATTiny and some AT90s

- ATmega323 - SPI mode when SPE bit in SPCR is set:
- PB6=MISO, PB5=MOSI, PB7=SCK, PB4=/SS
- SPCR – sets bit rate, CPOL, CPHA, M/S
- SPDR – used for data transfer to and from SPI shift register

- For multiple slaves, must employ "bit-banging". Use software to control serial communication at general-purpose I/O pins.

Summary

- I2C and SPI provide good support for communication with slow peripheral devices that are accessed intermittently, mainly EEPROMs and real-time clocks
- I2C easily accommodates multiple devices on a single bus.
- SPI is faster, but gets complicated when there is more than one slave involved.

References

- I2C:
- http://www-us2.semiconductors.philips.com/acrobat/various/
- I2C_BUS_SPECIFICATION_1995.pdf
- http://www.esacademy.com/faq/i2c/index.htm
- http://www.embedded.com/story/OEG20020528S0057

- SPI:
- MC68HC11 manual
- http://www.mct.net/faq/spi.html
- http://links.epanorama.net/links/serialbus.html
- http://www.embedded.com/story/OEG20020124S0116