ROBOTC-I2C INTERFACE PACKET II

Written by : Elizabeth Mabrey, Director of Storming Robots

Before you use this document:

Scope: This tutorial will guide to program NXT as a master I2C with Arduino/Nano as the I2C slave device. Although this document will cover how to hook up basic analog and digital sensors/devices with the Nano, this is NOT meant to teach you electronic.

Before you work on this document, you must have completed Packet I.

Note: I2C protocol communication can be very extensive and complex. For example, what if a sensor module is abruptly terminated. The Arduino has a tendency holds the line LOW and will no longer responses with proper data until you manually reset the controller.

Or

You should perform good amount of error checking in order to make your new sensors module robust to use as well.

Table of Contents

Preface

THIS DOCUMENT SCOPE

This document Is not meant to be circuitry learning document!

This is meant to cover the most basics in order to prepare yourself for working with 3rd parties devices. You will use Mindstorms platform as your main controller interfacing with a prototype board on a solderless breadboard. You create create a few sensors on the breadboard and later incorporated with the mindstorms platform. You will be using RobotC as your development environment.

IMPORTANT SAFETY RULES …- MUST READ FIRST…

- **Never** wire up or hook up electronics while it is receiving power. Unplug Arduino from the computer and make sure any attached battery power is turned off or unplugged. Otherwise, it may damage the board and/or any sensors connected to it.
- **Never** put the Arduino down on a metal surface when it receives power.
- **Never** leave it open in a place where static electricity easily occurred.
- **Never** short circuit the connection!
- **Aways** disconnect the Arduino from the computer while it is connected to the NXT.

Safety for yourself and your electronic components

1. Watch out for conductors around you,
    - You are the conductors, any metallic stuff are conductors. Your fingers touch any open circuits will short the connection, that simple.
    - Must remove away from your circuit all possible conductors which are foreign to your system.
    - Especially watch out for small pins, small snails, screws, small metallic clips off, etc.
    - Have a specific container for all your loose pins, etc. to keep them away from your bot.
2. Use anti-static bag for storage generously
3. MUST keep any foreign object from the bot, even when in storage.
4. Remove power source when not in use, even batteries. That means, even when button is off, you should remove the physical connection to any power source. In the case of power pack, I am not saying removing the wire to the power pack, but just completely remove at least one battery.
5. Make sure the power supply is disconnected when wiring your circuits, especially the DC Motor or Servo Controllers.
6. Keep your robot in an insulted container…
7. Keep out moisture.. including your drink… or even a sneeze …
8. The battery should be positioned so it will not rub against sharp edges. A damaged, leaking battery is a safety hazard.

WHEN IT COMES TO WIRING:

Do's:

1. Use shortest distance and consistent color, such as Black for ground, Red for power, etc.
2. Modularized and Systematic testing and design. Yes, even for your wiring.
    - I cannot emphasize enough how important it is to practice the same principle like software development in wring. The same principal such as modularization and clear and clean organization not only also goes true with the circuit wiring, but supremely important.
    - Do some load/stress test for a completed sub-module, before you put in another sub-module.
    - Modularize to allow removal and assembling easy, to allow isolation of problematic connection.
    - Modularize you assembling, test each as it goes. Just like software development.
    - Do continuity test if you do any soldering yourself.
3. Must have clear pin maps diagram.
4. Must leave an easy path way to remove the power source, i.e. the battery pack. I mean something you can do within a couple of seconds max.. I mean it seriously… not like the old days with the protected circuits.

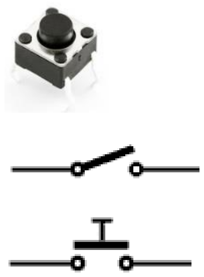
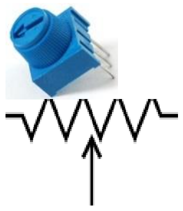
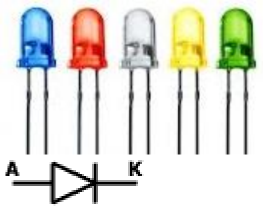
Don'ts :

1. ABSOLUTELY NO spaghetti wiring.
2. Do not leave loose batteries seating in the battery pack.
3. Do not attempt to remove and put in new wire while there is power supply to it including connection to your computer, or full set of batteries in the power pack.
4. Do NOT stack conductors, including wires which may have non-insulated area, in-between layers of boards.
5. Avoid stacking up the tight pre-bent wires against each other.
6. Avoid running wires along 'pinch points'. Sharp metal pieces and gears can damage the wires and their insulation. When possible, run wires through metal tubing and wire-tie them to structural components.
7. Do not throw your electronic device together with many other accessories such as wires, and others
8. Do not just throw your sensor in your pocket without an anti-static bag. (I know this is pretty obvious…. But unfortunately, just simply saw that too many times!)

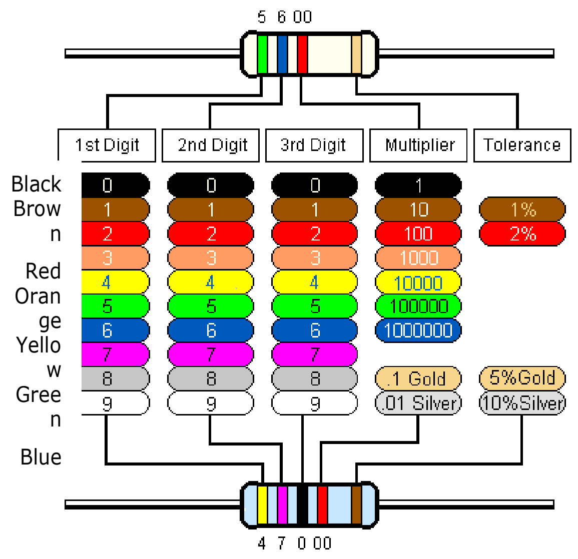
CH1 - SOME BASICS ON THE ARDUINO SIDE

CH1 -1) BASICS PARTS USED IN THIS PACKET

To interface with Arduino, lets start with Arduino side.

Arduino Nano V3			
6 220 Ohm resistor		3 push buttons 	
1-4 photo-resistor		1 10K ohm potentiometer	
	Parallex Ping))	Various Color and clear LED  Long end = '+' Short end = '-'	

CH1 -2) About Resistors

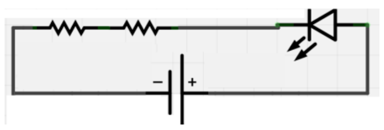


Mnemonic :

Big **B**ad **R**obots **O**n **Y**our **G**earbox **B**lunders **P**ass **G**reat **W**all

Resistor - Ohm's Law

In Series:



$R_{total} = R_1 + R_2 + \dots + R_n$

$V = I * R_{total}$

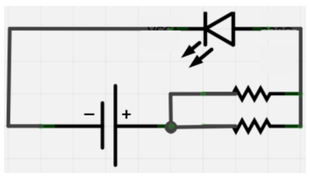
Current remains the same at each Resistor even when each resister measures different Ω.

The amount of current (think about amount of water) going through each resistor (think about various sizes or same size of pipes) will remain the same.

$I_{R1} == I_{R2} == \dots == I_{Rn}$ even if all R_n measure different Ω.

In Parallel:

$$R_{total} = \frac{1}{\frac{1}{R1} + \frac{1}{R2} + \dots + \frac{1}{Rn}} \ \Omega$$



Current will be different at various Resistor if these resistors measure different Ω.

The amount of current going through each resistor may be the different.

$I_{R1} \,!= I_{R2} \,!= \dots != I_{Rn}$ if all R_n measure different ohms.

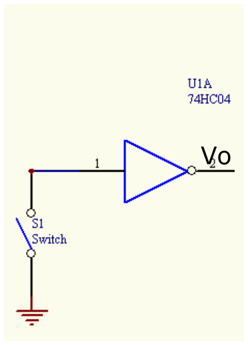
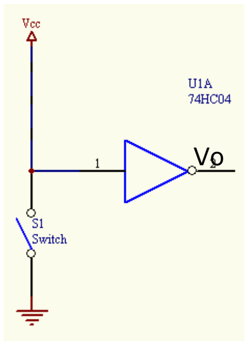
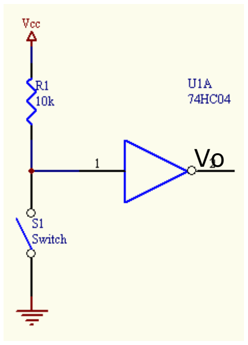
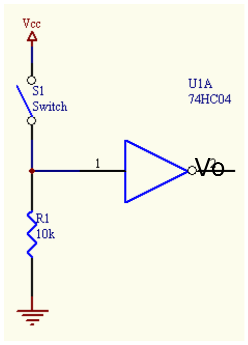
CH1 -3) PULL-UP VS. PULL-DOWN RESISTOR

The Basic use of a pull-up or pull-down resistor is to prevent a 'floating' input to a digital circuit.

A 'Pull-up' ----------- connected from the input pin to +Vdd
A 'Pull-down' ---------connected from the input pin to Gnd or 0V.

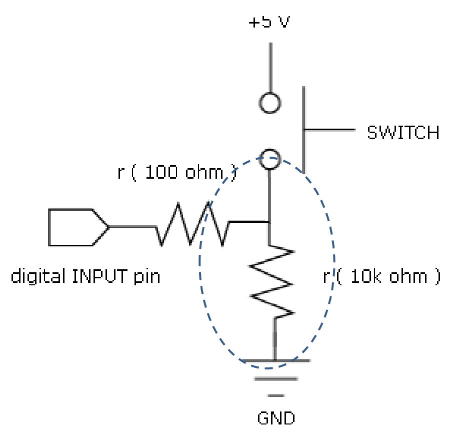
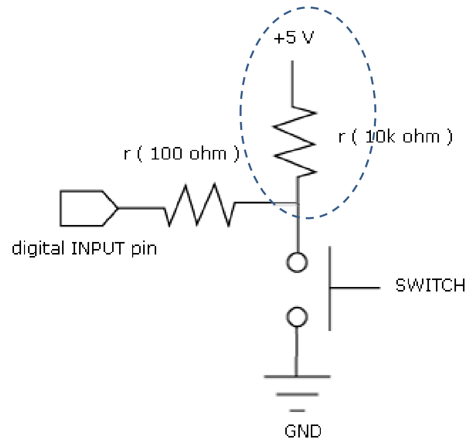
Both have the same key function: to remove noise.

How: To create a default value for a circuit to ensure that the wire is at a defined logic level even if no active devices are connected to it. The difference is that one pulls the line high, the other pulls it low. See the following diagrams:

Not good. Allow floating	Really BAD!	Pull Up Resistor	Pull Down Resistor
		 Vcc R1 10k U1A 74HC04 Vo 1 2 S1 Switch	 Vcc S1 Switch U1A 74HC04 Vo 1 2 R1 10k
When switch S1 is open (off), then input pin 1 is susceptible to a wide array of electrical problems.	When Switch S1 is closed, it shorts the circuit.	Let say there is a LED on pin1. S1 opens: LED is on S1 closes: LED is off	Let say there is a LED on pin1. S1 opens: LED is off S1 closes: LED is on

How Pull up vs down affects the flow of current

Electricity behaves a bit like water, i.e. flow to where it encounters the least resistance. Electricity takes the path of least resistance and moves between 5V and input pin.

Pull-down resistor		Pull-up resistor
 +5 V SWITCH r (100 ohm) digital INPUT pin r (10k ohm) GND		 +5 V r (100 ohm) r (10k ohm) digital INPUT pin SWITCH GND
e.g. switch == a push button		e.g. switch == a push button
Button not pushed , digitalRead(Vout) == LOW Button pushed , digitalRead(Vout) == HIGH		Button not pushed , digitalRead(Vout) == HIGH Button pushed , digitalRead(Vout) == LOW

CH1 -4) MISC.

Vin vs. Vout

- Vin == the power source
- Vout == voltage output, usually the feedback value from sensors

About LED:

- Like resistor, LED can be used to limit amount and direction of current.
- Resistor and LED may be interchanged (but polarity of LED is important).

About Diode:

— Restrict current to flow one direction only.
— Mainly for creating your own power source
— Convert AC to DC and vice versa

About Capacitor:

— Energy storage device to smooth out voltages.
— Capacitance rating : How much energy it can hold for a given voltage – μF (micro Farads)
— Voltage rating : max voltage the capacity should be exposed to.
— Never use anything over 20V. 30V+ can be dangerous to human.
— Polarity is important … it will blow up if it is put in a wrong direction.

Total capacitance in a parallel circuit : $C_T=C_1+C_2...+C_n$

Total capacitance in a series circuit: $C_T= \frac{1}{\frac{1}{C1}+\frac{1}{C2}+\ldots+\frac{1}{Cn}}$.

About inductor:

Device temporarily store energy as a form of magnetic field
Coil of wire – creating magnetic field

To smooth out voltage

About Potentiometer

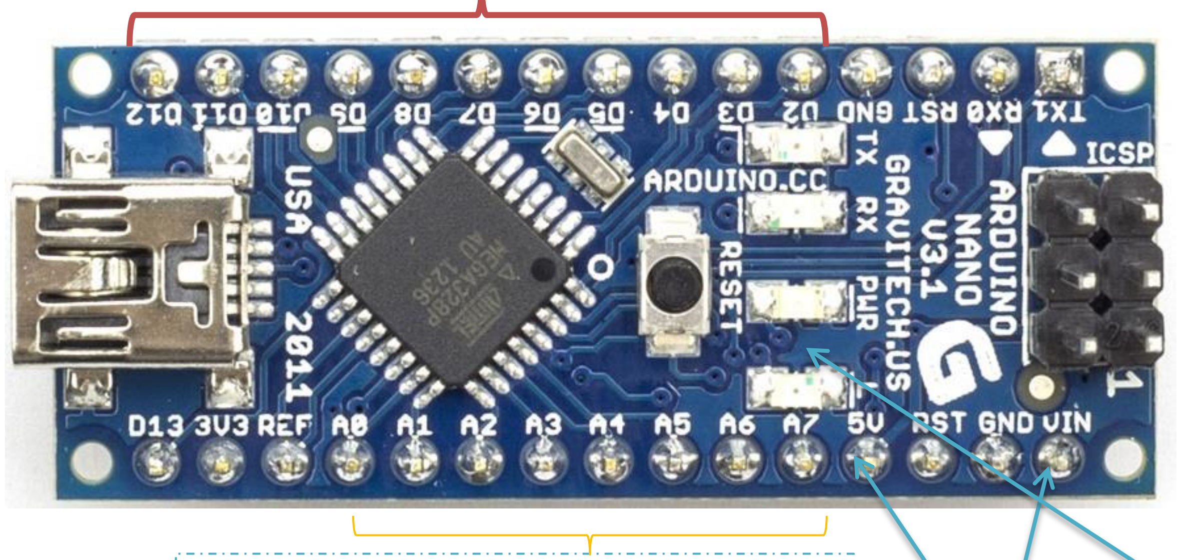
Potentiometer, an electrical device that measures potential difference between two points in a circuit by comparison with a standard battery of known potential difference.

CH2 -CONNECTING SENSORS WITH ARDUINO

CH2 -1) ARDUINO NANO BOARD PIN MAP:

D2 to D13: Digital pins. (Don't use D0 and D1 (used for Serial communication).

- With pinMode(), digitalRead(), digitalWrite(), analogWrite()


D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
ICSP
ARDUINO.CC
USA 2011
RESET
TX RX PWR
GRAVITECH.US
ARDUINO NANO V3.1
D13 3V3 REF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN

Analog Pin A0 to A7 : 10-bits resolution.

- With pinMode(), digitalRead(), digitalWrite()
- A4 & A5 == SDA & SCL respectively (but may be different in various versions)

5V – power line

GND - ground

Internal LED

CH2 -2) Learn From Samples (Sketch-C IDE)

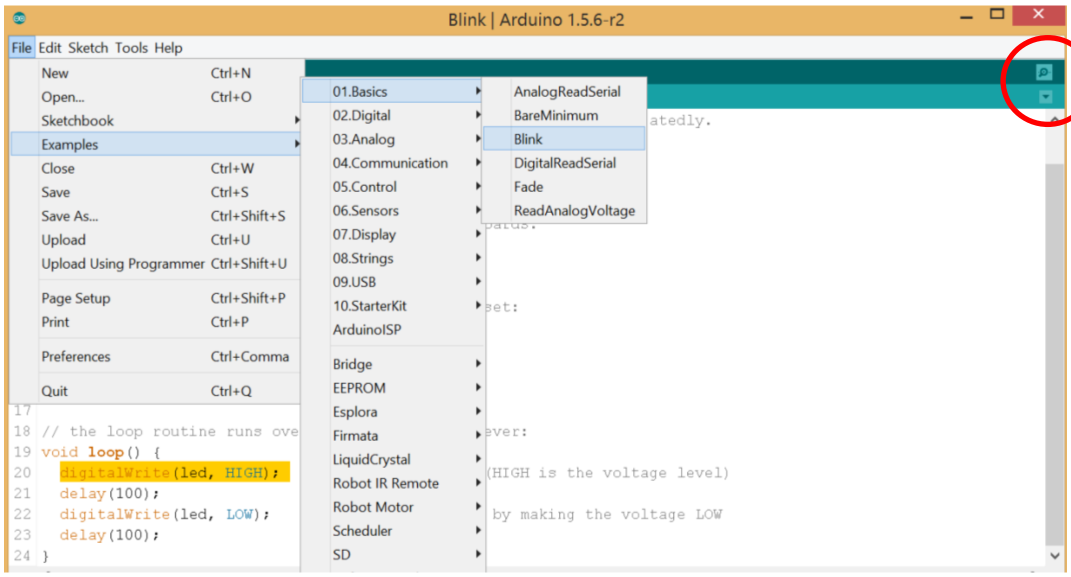
Best way to learn from Samples is :

- Compile and download to ensure it is working
- Read the code and understand how to use the APIs
- Rewrite your own by switching the pins connection.

Exp1 - Blink on embedded LED on Pin 13

Test the "Blink" Sample program from Sketch

Note : Pin 13 has an LED connected on most Arduino boards.

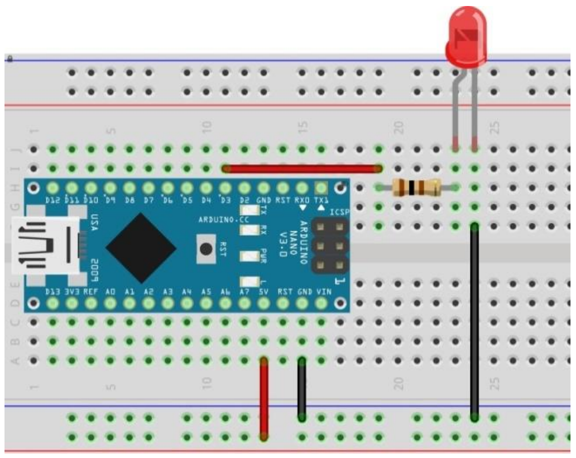


```
// the loop routine runs ove
void loop() {
  digitalWrite(led, HIGH);
  delay(100);
  digitalWrite(led, LOW);
  delay(100);
}
```

Serial monitor

Exp2 - Blink an external LED

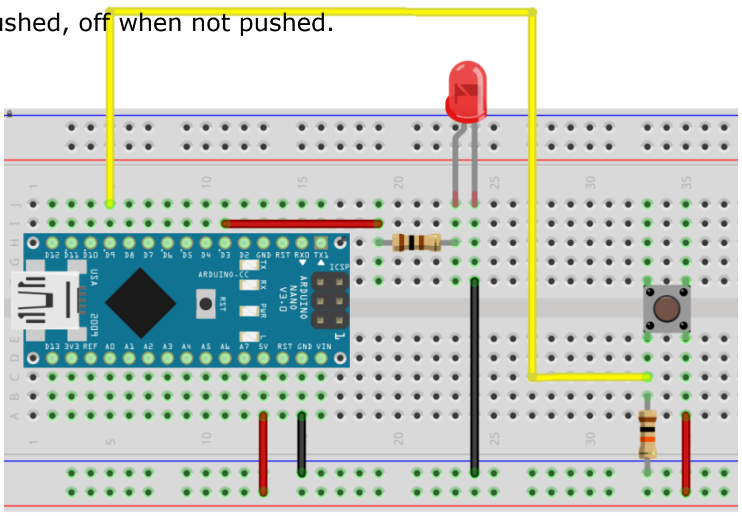
Connect an external LED to digital pin 9, and write a program to blink it.



Exp3 - Blink an external LED with a push button

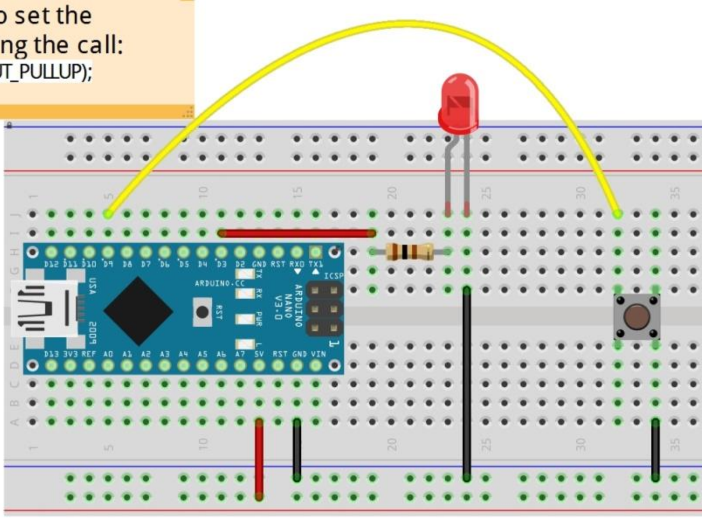
Connect and write a program to turn on the LED when pushed, off when not pushed.

Note: you have just done a **<u>pull-down resistor</u>** circuit.



Note: the following is to utilize the internal **<u>pull-up resistor</u>** circuit, or you can do it yourself.

make sure to set the pinmode using the call:
pinMode(9, INPUT_PULLUP);



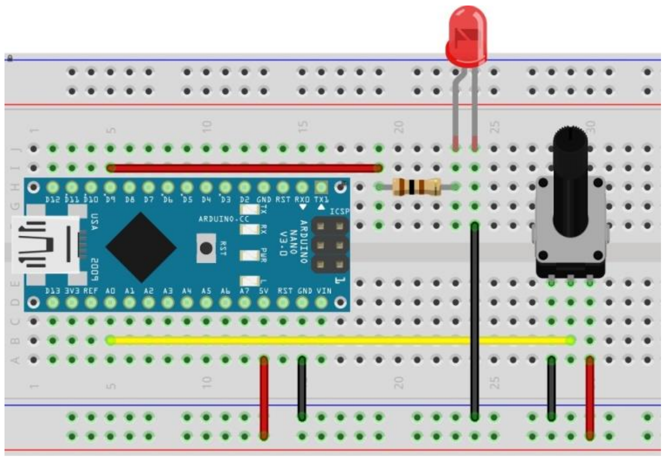
<u>Review:</u>

Pull up == the input line needs pull up… i.e. high by default …..
i.e input line with the resister connects to power.

Pull down == the input line needs pull down … i.e. low by default .. …..
i.e input line with the resister connects to Ground.

Exp 4 - Turn on/off LED with potentiometer

Refer to Examples → Analog → AnalogInput



Exp 5 : Dim/Brighten LED with potentiometer

Now, program your code to brighten/dim your LED instead of on or off.

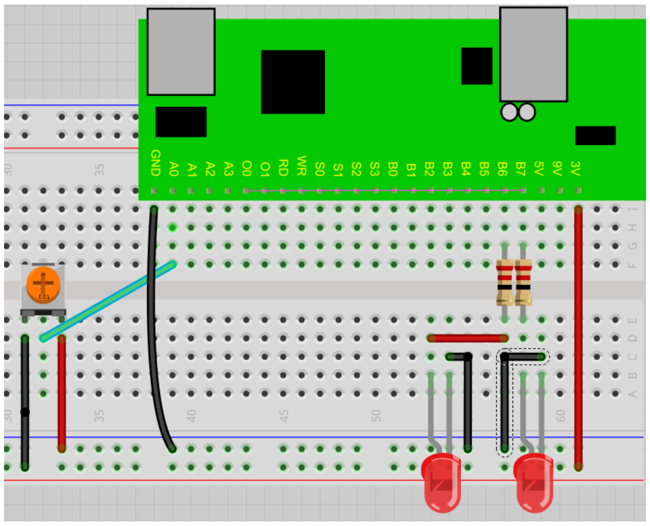
Refer to Examples → Analog → Fading

Exp 6 :Bonus Challenge - Light up 2 LEDs (connected in Parallel)

Note that the LEDs are connected in parallel circuit. To each of the LED itself, the circuit to it is not in Parallel. However the circuit system as a whole is connected in Parallel. This diagram hooks to another controller board. You need to create your own circuit using NANO instead.

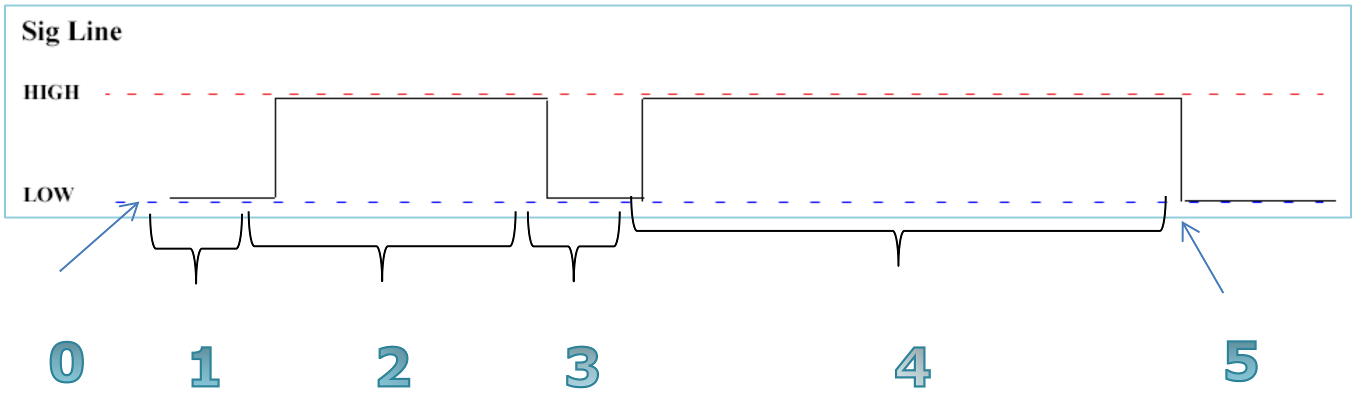
Program: Turn on led on B7 when the analog value of the potentiometer > 2^8

Turn on led on B6 when the analog value of the potentiometer > 2^7



CH2 -3) Connect to the Parallex Ping - the Ping Pulse

Since this sensor module involves a ping (a very commonly used sensor), we should briefly get an idea how the Ping signal works. You should review the Ping code from Arduino and fill in below… …



0 •pinMode to OUTPUT

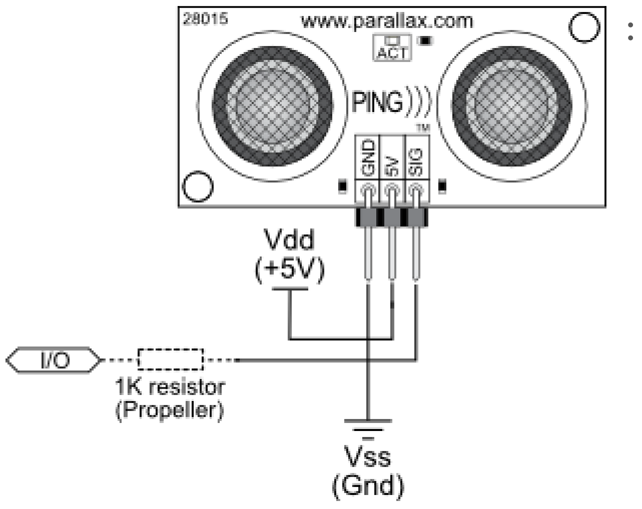
1 • We make signal line go LOW for __________ microseconds. (delayMicrosecond())

2 • Set Signal Line HIGH for ________ microseconds

3 • Here we set signal line LOW. This activates the PING sensor and it sends a sonic pulse

4 •PING sensor will set the signal line HIGH for the amount of time it took for the 'ping' to make its round trip. This is the duration what wemeasure and convert to distance

5 • Immediately followed by pulseIN() or micros() function



Hint:

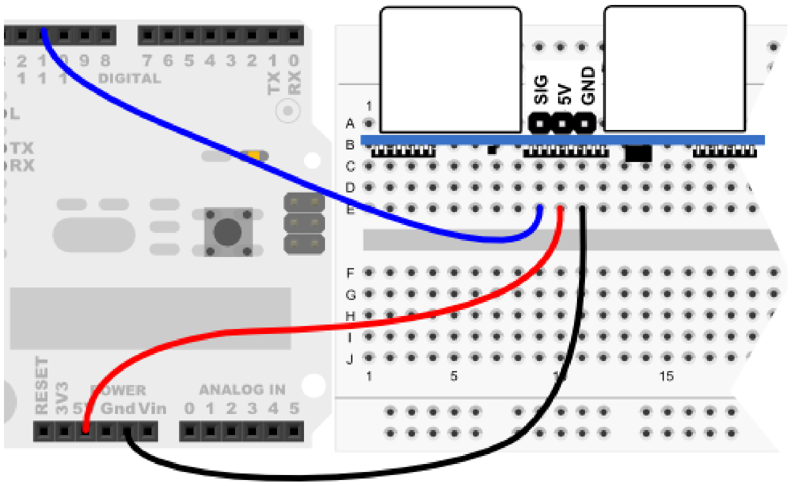
1. We make signal go LOW for 2 microseconds, then set signal high for 5 microseconds
2. Do not forget to change the pinMode for the SIG line to INPUT after step 3. That means we must make sure we set the SIG line to OUTPUT prior to step 1.

CONNECT TO ONE PARALLEX PING))

Write a program to interface with the Ping. Use the "ping" example code from Arduino.

Refer to : **Arduino → Examples → Sensors → Ping**

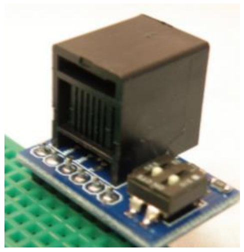
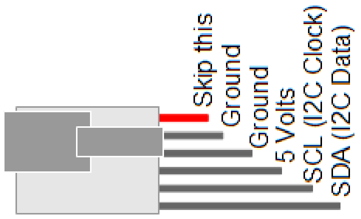
Connecting Parallex Ping

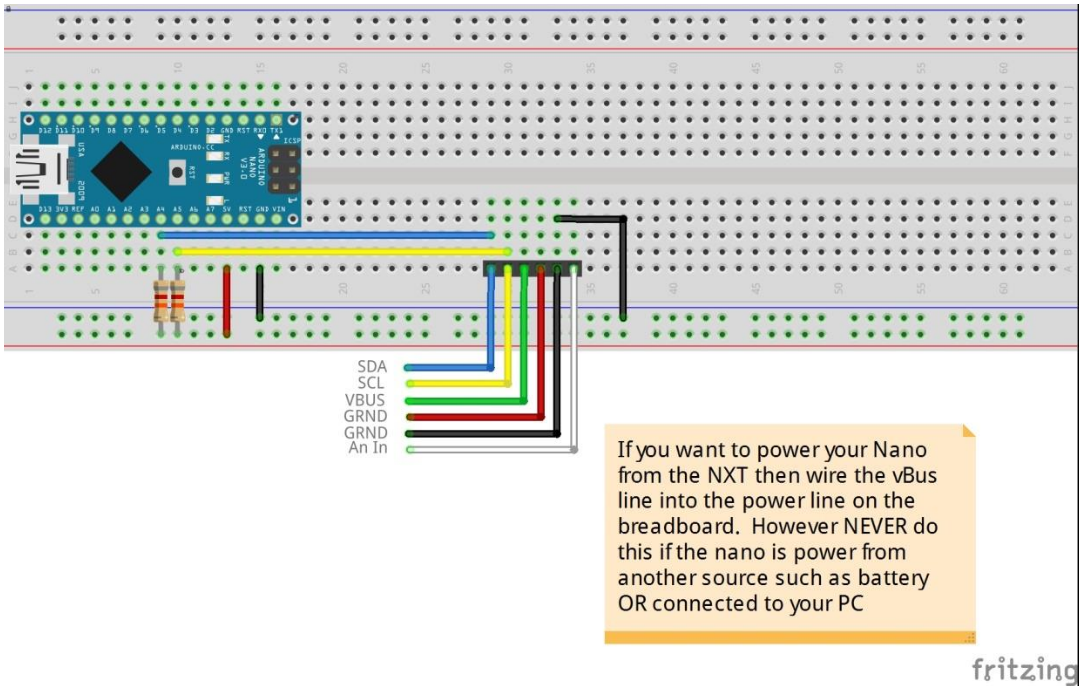


- Use the Ping sample code from the Arduino/Sketch as your Ping code template.

CH3 TALK I2C BETWEEN NXT AND ARDUINO/NANO

CH3 -1) SAMPLE CONNECTIONS DIAGRAM

Optional : a NXT breadboard adaptor from Mindsensors or Dexter Industries. OR simply strip one end of a NXT wire. A few 82K and/or 43K resistors. Typically resistor values of 82k are used. If the device you are communicating with is operating at 3.3V (rather than 4.7V) you can use 43k resistors. Resistor values are important to ensuring you can communicate at a fast speed with the NXT.	via NXT breadboard adaptor from Dexter Industries 
Or simply strip the NXT wire  Skip this Ground Ground 5 Volts SCL (I2C Clock) SDA (I2C Data)	via NXT breadboard adaptor from Mindsensors. + two 83K resistor

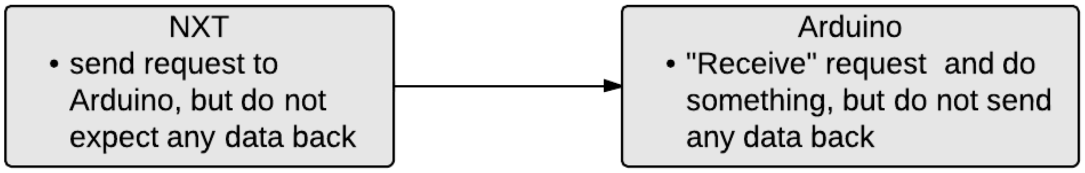


If you want to power your Nano from the NXT then wire the vBus line into the power line on the breadboard. However NEVER do this if the nano is power from another source such as battery OR connected to your PC

CH3 -2) SIMPLIED HANDSHAKING

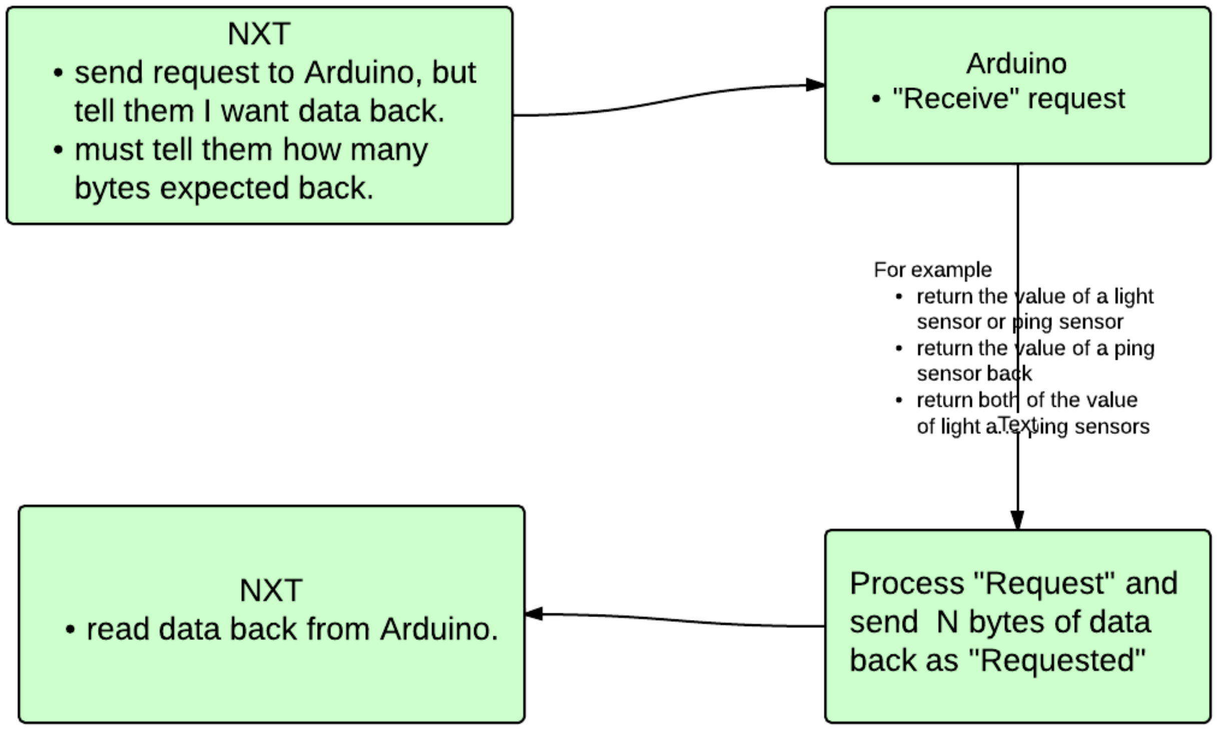
Note: This is an over-simplified flow in order to make it easier to understand as introduction.

One way Handshaking


NXT
• send request to Arduino, but do not expect any data back
Arduino
• "Receive" request and do something, but do not send any data back

e.g.
- blnk the LED
- turn the RGB to red

Bi-directional Handshaking


NXT
• send request to Arduino, but tell them I want data back.
• must tell them how many bytes expected back.
Arduino
• "Receive" request
For example
• return the value of a light sensor or ping sensor
• return the value of a ping sensor back
• return both of the value of light and ping sensors
Text
Process "Request" and send N bytes of data back as "Requested"
NXT
• read data back from Arduino.

CH3 -3) APIS THAT YOU NEED TO KNOW

HEADER FILE AND BASIC APIS

```
#include <Wire.h>

volatile uint8_t Request;

//…some global data. E.g. one of the sensors is ping, and one function request from
the //master is to turn blink on or off
volatile uint8_t PingValue;
volatile bool Blink=false;

void setup()
{
… Wire.begin( );
… Wire.onReceive( recEventFunction );
… Wire.onRequest( reqEventFunction);
}

Void loop()
{
   PingValue = …some function to obtain the distance
}

void recEventFunction( int nbytesRead )
{    …
    Request =  Wire.read();   …
}

void reqEventFunction()
{
    Switch (Request)
    { case ….
    }
}
```

List of APIs:

- begin()
- requestFrom()   // used by master device only. Not applicable for our sample
- beginTransmission() // used by master device only. Not applicable for our sample
- endTransmission()   // used by master device only. Not applicable for our sample
- write()
- available()
- read()
- onReceive()
- onRequest()

Consult www.**arduino**.cc/en/Reference/**Wire** for more details.

Caution:

recEventFunction() and reqEventFunction() are called based on something called system interrupt. For the sake of staying within the scope of this document. We won't go into that. However, one thing you MUST know is that these routines MUST return AS QUICKLY AS possible.

NEXT SECTION WILL SHOW YOU WITH THE CODE.

CH3 -4) SAMPLE I2C SEND MESSAGE WITHOUT REPLY NEEDED

```
I2CRequest[0] = 3;    // Message size        ①
I2CRequest[1] = 0x02; // I2C Address
I2CRequest[2] = 0x41; // 0x41 is command
I2CRequest[3] = 0x09; // turn on pings

if (!writeI2C(link,I2CRequest)){          ②
  return -1;
}
```

① I2CRequest buffer is made. Setting command mode is 0x41 and the command to send is 0x09 (turn off pings sensors) Notice that the message size is set to 3.

② Call to writeI2C(). Notice we do not request any data back. We just want to send a command.

③ receiveEvent() is called on the Arduino. There will be two bytes put into the receiveBuffer (0x41 and 0x09).

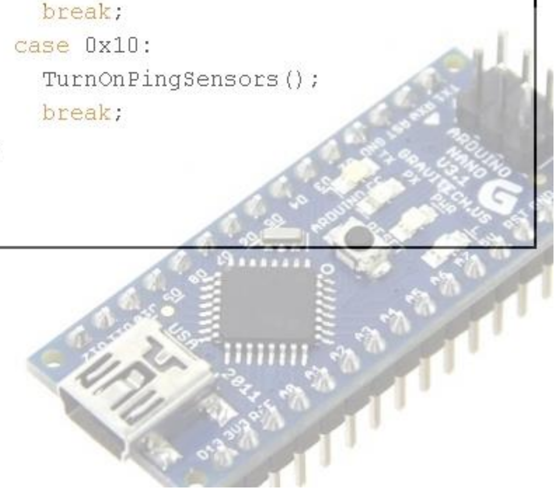
④ receiveEvent() looks at the first element of receiveBuffer. If it is the command value (0x41) it calls processCommand().

⑤ Process command looks at the second byte. If it is 0x09 it turns off the Ping sensors. To turn them back on again the NXT should make the same call except I2CRequest[3] should equal

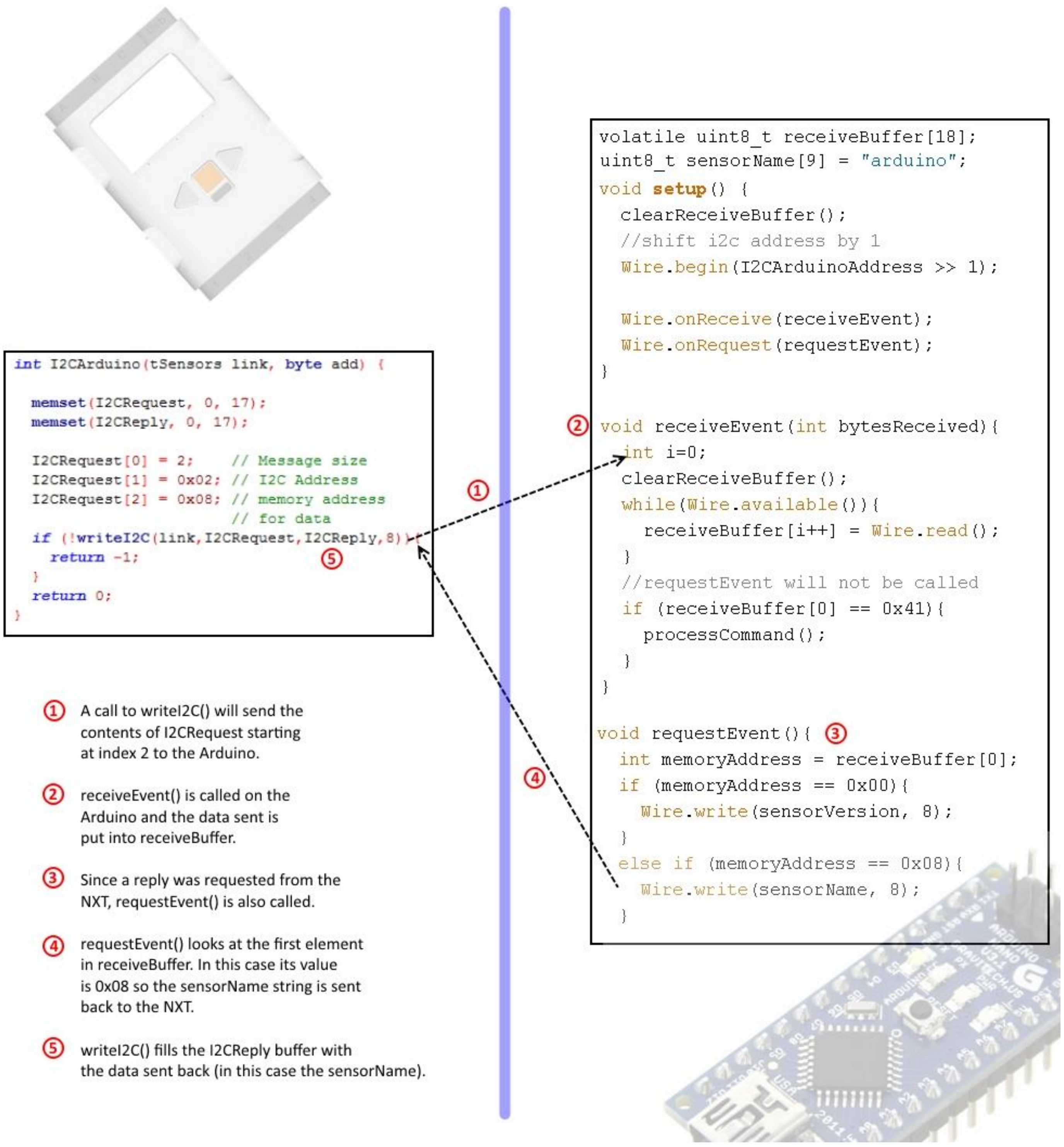
```
volatile uint8_t receiveBuffer[18];
uint8_t sensorName[9] = "arduino";
void setup() {
  clearReceiveBuffer();
  //shift i2c address by 1
  Wire.begin(I2CArduinoAddress >> 1);

  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
}

void receiveEvent(int bytesReceived){
③ int i=0;
  clearReceiveBuffer();
  while(Wire.available()){
    receiveBuffer[i++] = Wire.read();
  }
  //requestEvent will not be called
  if (receiveBuffer[0] == 0x41){
④   processCommand();
  }
}
void processCommand(){
  switch (receiveBuffer[1]){
⑤   case 0x09:
      TurnOffPingSensors();
      break;
    case 0x10:
      TurnOnPingSensors();
      break;
  }
}
```



CH3 -5) SAMPLE I2C SEND MESSAGE WITH REPLY NEEDED



```
int I2CArduino(tSensors link, byte add) {

  memset(I2CRequest, 0, 17);
  memset(I2CReply, 0, 17);

  I2CRequest[0] = 2;    // Message size
  I2CRequest[1] = 0x02; // I2C Address
  I2CRequest[2] = 0x08; // memory address
                        // for data
  if (!writeI2C(link,I2CRequest,I2CReply,8)){
    return -1;
  }
  return 0;
}
```

```
volatile uint8_t receiveBuffer[18];
uint8_t sensorName[9] = "arduino";
void setup() {
  clearReceiveBuffer();
  //shift i2c address by 1
  Wire.begin(I2CArduinoAddress >> 1);

  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
}

void receiveEvent(int bytesReceived){
  int i=0;
  clearReceiveBuffer();
  while(Wire.available()){
    receiveBuffer[i++] = Wire.read();
  }
  //requestEvent will not be called
  if (receiveBuffer[0] == 0x41){
    processCommand();
  }
}

void requestEvent(){
  int memoryAddress = receiveBuffer[0];
  if (memoryAddress == 0x00){
    Wire.write(sensorVersion, 8);
  }
  else if (memoryAddress == 0x08){
    Wire.write(sensorName, 8);
  }
```

① A call to writeI2C() will send the contents of I2CRequest starting at index 2 to the Arduino.

② receiveEvent() is called on the Arduino and the data sent is put into receiveBuffer.

③ Since a reply was requested from the NXT, requestEvent() is also called.

④ requestEvent() looks at the first element in receiveBuffer. In this case its value is 0x08 so the sensorName string is sent back to the NXT.

⑤ writeI2C() fills the I2CReply buffer with the data sent back (in this case the sensorName).

Reminder:

- Create a shared header file which contains the I2C address for the slave device, as well as all the command addresses.
- with the RobotC - Xander's i2c API – writeI2C(…) . If you choose to use the native calls such as sendI2Cmsg(…), you need to make sure you will do sufficient error checking.
- As mentioned in I2C Packet I, you may need to modify the message buffer type to ubyte msg[**17**].

Download Sample Codes - NXT as the master device, and Nano as the slave device for further examples.

CH3 -6) TO GET STARTED…

Steps:

1) Design your slave device.
   a. which sensor(s) on what pin(s)
   b. what are they for will also determine how you will mount it.
   c. write a program on the Arduino side to ensure your code works in terms of obtaining the data. e.g. obtain light sensor value and ping value in the "loop()" function.
2) Create the header file which contains the memory addresses used for the communication including:
   a. Your slave device I2C address (must be even number)
   b. The list commands addresses to meet your design
3) Modify your program by incorporating the two event functions – onReceive and onRequest.
4) Deploy proper development practice - MUST divide and conquer.. one sensor at a time. Test it systematically.
5) Program the master device to send request to obtain data.

NOTE: if you choose not to use proper development practice and just code everything without testing one at a time, but then running into issues, instructor will ask you to dissect your code to practice "divide and conquer" before they will lend more assistance.