Lab 7: I2C – Using a Wii Nunchuck Controller

COEN-4720 Embedded Systems
Cris Ababei
Dept. of Electrical and Computer Engineering, Marquette University

1. Objective

The objective of this lab is to learn about I2C communication protocol. This will be done by using the Wii NunChuck to control a moving circle on the LCD display attached to the Nucleo board. We will also look at coding in C++ the communication and functions related to the NunChuck – this serves as a good example to compare coding in C vs. coding in C++.

2. I2C Introduction

I2C (inter-integrated circuit) is a two-wire protocol used to connect one or more "masters" with one or more "slaves". The case of a single master (LPCxxxx) communicating with two slave devices is illustrated in Fig.1. In this configuration, a single master (LPCxxxx) communicates with two slaves over the pair of signal wires SCL (serial clock line) and SDA (serial data/address). Example slave devices include temperature, humidity, and motion sensors as well as serial EEPROMs, serial RAMs, LCDs, tone generators, other microcontrollers, etc.

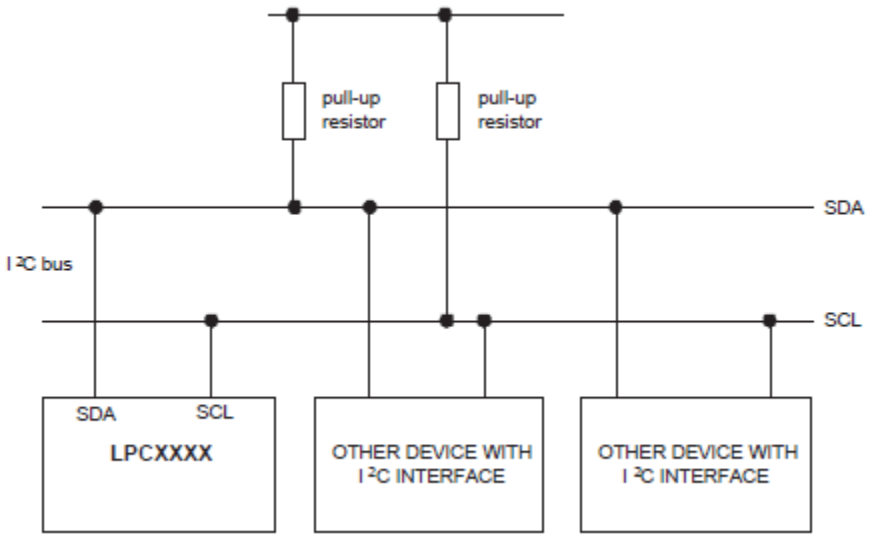

Figure 1: I2C bus configuration.

Electrically, the value of the two signal wires is "high" unless one of the connected devices pulls the signal "low". The two pull-up resistors in Fig.1 force the default value of the two bus wires to VCC (typically 3.3V). Any device on the bus may safely force either wire low (to GND) at any time because the resistors limit the current draw; however, the communication protocol constrains when this should occur. To communicate, a master drives a clock signal on SCL while driving, or allowing a slave to drive SDA. Therefore, the bit-rate of a transfer is determined by the master.

The I2C Physical Protocol:
Communication between a master and a slave consists of a sequence of transactions where the master utilizes the SCL as a clock for serial data driven by the master or a slave on SDA as shown in Fig.2.

When the master wishes to talk to a slave, it begins by issuing a start sequence on the I2C bus. A start sequence is one of two special sequences defined for the I2C bus, the other being the stop sequence. These are also referred to as Start condition (S) and Stop condition (P).

Figure 2: I2C physical protocol.

A transaction consists of a sequence of bytes. Each byte is sent as a sequence of 8 bits. The bits of each byte of data are placed on the SDA line starting with the MSB. The SCL line is then pulsed high, then low. For every 8 bits transferred, the device receiving the data sends back an acknowledge bit, so there are actually 9 SCL clock pulses to transfer each 8 bit byte of data. If the receiving device sends back a low ACK bit, then it has received the data and is ready to accept another byte. If it sends back a high (Not Acknowledge, NACK) then it indicates it cannot accept any further data and the master should terminate the transfer by sending a stop sequence.

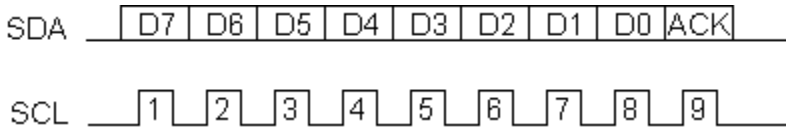

Figure 3: Illustration of sending of 1 byte of data.

The data may be sent by either the slave or the master, as the protocol dictates, and the ACK or NACK is generated by the receiver of the data. Start (S) and Stop (P) conditions are always generated by the master. A high to low transition on SDA while SCL is high defines a Start. A low to high transition on SDA while SCL is high defines a Stop.

There are three **types of transactions** on the I2C bus, all of which are initiated by the master. These are: write, read - depending on the state of the direction bit (R/W), and combined transactions. The first two of these (illustrated in Fig.4) are:

(1) Write transaction - Data transfer from a master transmitter to a slave receiver. The first byte transmitted by the master is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte, unless the slave device is unable to accept more data.

(2) Read transaction - Data transfer from a slave transmitter to a master receiver. The first byte (the slave address) is transmitted by the master. The slave then returns an acknowledge bit.

Next follows the data bytes transmitted by the slave to the master. The master returns an acknowledge bit after all received bytes other than the last byte. At the end of the last received byte, a "not acknowledge" is returned. The master device generates all of the serial clock pulses and the START and STOP conditions. A transfer is ended with a STOP condition or with a repeated START condition. Since a repeated START condition is also the beginning of the next serial transfer, the I2C-bus will not be released.

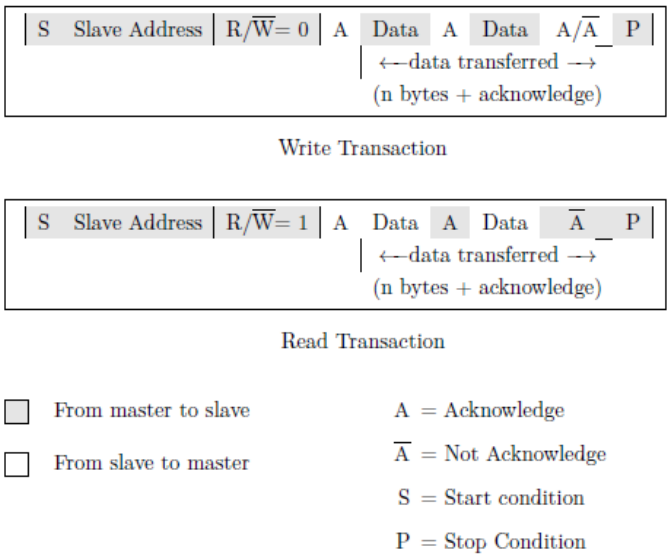

Figure 4: I2C write and read transactions.

The STM32L053R8 MCU I2C interfaces are byte oriented and have four operating modes: master transmitter mode, master receiver mode, slave transmitter mode, and slave receiver mode. Please read Chapter 28 of the MCU user manual for details on each of these operating modes. Additional tutorials and descriptions are listed at [2].

I2C Device Addressing:
All I2C addresses are either 7 bits or 10 bits. The use of 10 bit addresses is rare. Most common chips use 7 bit addresses - we can have up to 128 devices on the I2C bus. When sending out the 7 bit address, we still always send 8 bits. The extra bit is used to inform the slave if the master is writing to it or reading from it. If the bit is zero the master is writing to the slave. If the bit is 1 the master is reading from the slave. The 7 bit address is placed in the upper 7 bits of the byte and the Read/Write (R/W) bit is in the LSB.

3. Wii NunChuck

The Wii NunChuk is an input device with a joystick, two buttons, and a three-axis accelerometer as illustrated in Fig.5 and Fig.6. The three axes X, Y, and Z correspond to the data produced by the accelerometer, joystick. X is right/left, Y is forward/backwards, and Z is up/down (accelerometer only).

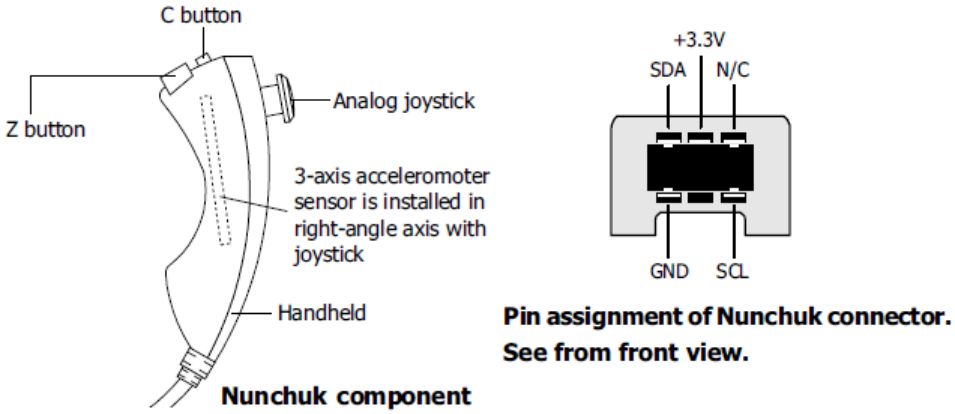

Figure 5: NunChuck basic information.

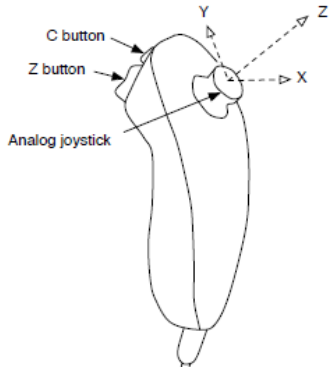

Figure 6: NunChuck X,Y,Z.

The NunChuck communicates via I2C. We will connect the NunChuck to the I2C1 bus of the STM32L053R8 microcontroller on the Nucleo board. We will initialize the Nunchuk to a known state and then regularly "poll" its state. The data are read from the NunChuck in a six-byte read transaction. These data are formatted as illustrated in Fig.7 and are read beginning with byte 0x0 (little-endian). The only complication with this format is that the 10-bit/axis accelerometer data are split.

	7	6	5	4	3	2	1	0
0x00	Joystick JX							
0x01	Joystick JY							
0x02	Accelerometer AX[9:2]							
0x03	Accelerometer AY[9:2]							
0x04	Accelerometer AZ[9:2]							
0x05	AZ[1:0]		AY[1:0]		AX[1:0]		C	Z

Figure 7: Formatting of data from NunChuck.

The NunChuck is a slave I2C bus device. It has 2 slave IDs for writing (0xA4) and reading (0xA5) data which is shown in Fig.8.

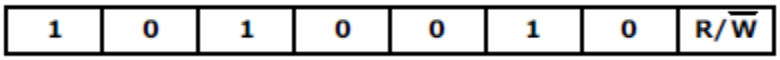

Figure 8: Slave IDs.

Communication with the NunChuk consists of two phases – an initialization phase (executed once) in which specific data are written to NunChuk and a repeated read phase in which the six data bytes are read. Each read phase consists of two transactions – a write transaction which sets the read address to zero, and a read transaction.

Initialize start NunChuk command

The initialization consists of two write transactions, each of which writes a single byte to a register internal to the I2C slave: reg[0xf0]=0x55, reg[0xfb]= 0x00. Normally this done once only.

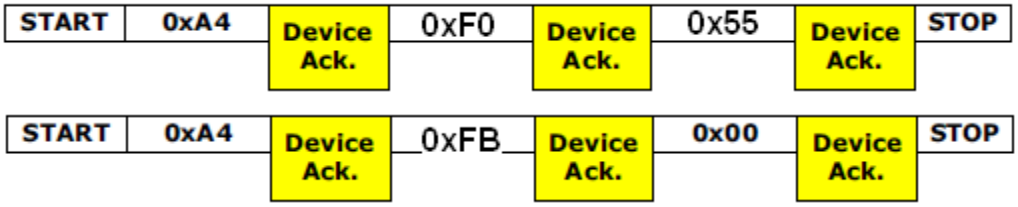

Figure 9: Phase 1 - initialization.

The read process consists of writing a 0 and then reading 6 bytes of data.

Conversion command (0x00)

Send this command to get all sensor data and store into the 6-byte register within Nunchuk controller. This must be executed before reading data from the Nunchuk.

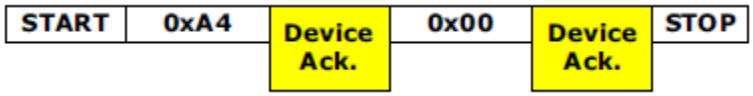

Figure 10: Phase 2 - send this first.

Data read command

Send the slave ID for reading (0xA5) and wait for the stream data 6-byte from the Nunchuk.

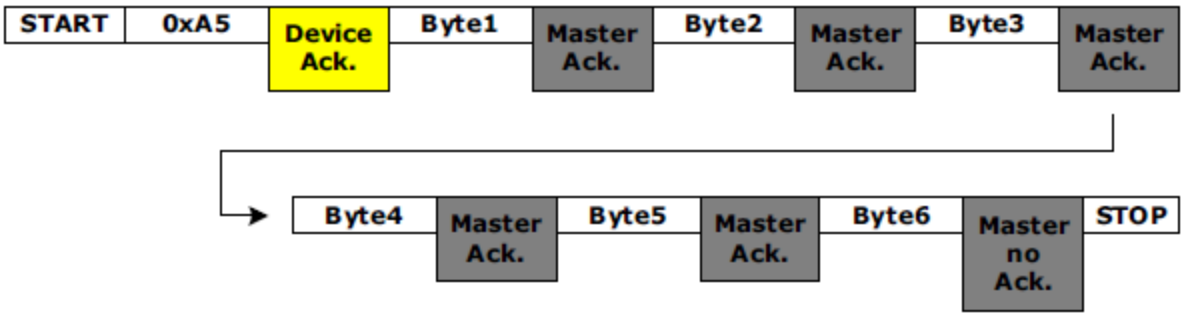

Figure 11: Phase 2 - reading 6 bytes.

The joystick data are in the range 0..255 roughly centered at 128. The dynamic range is somewhat less than the full range (approximately 30-220).
The accelerometer data are in the range 0..1023 where 0 corresponds to -2g and 1023 corresponds to +2g. The accelerometer can be used both to detect motion (acceleration), but also as a "tilt sensor" when it is not in motion because we can use the earth's gravitational field as a reference. Suppose we have measured values of gravity in three dimensions, Gx, Gy, Gz. Then:

$$G_x^2 + G_y^2 + G_z^2 = 1g^2$$

From this, it is possible to compute "pitch" (rotation around the X axis), "roll" (rotation around the Y axis) and "yaw" (rotation around the Z axis). For joystick replacement, it is sufficient to compute (after range conversion to -512..511).

$$pitch = atan\left(\frac{AX}{\sqrt{AY^2 + AZ^2}}\right)$$

$$roll = atan\left(\frac{AY}{\sqrt{AX^2 + AZ^2}}\right)$$

Remember that this is for 360 degrees and a reasonable of motion to measure is perhaps 90 degrees.

Please read more about NunChuck hacking in the cited references [3] and by searching on the Internet.

4. Description

Prerequisites for this Lab
Read Chapter 14 from the textbook and Chapter 28 from the user manual of the microcontroller..

Example 1
In this example, we create a project that uses the Wii NunChuck controller to move a small circle on the LCD display. In addition, all the data from the controller (joystick position, etc.) are printed on the display too.

While the complete project folder is included in the files for this lab, you should create your own project from scratch and then copy the necessary files and/or code from the provided project of this example, in the lba7_files.zip on the course website.
When creating the new project, inside STM32CubeMX Tool, right after you created the new project, select to use the SPI1 peripheral and configure it – similarly to how you did in the previous lab. That will be used to connect the LCD display.
Then, enable the **I2C1 peripheral** too. Set the Parameter Settings as shown in Fig.12. This I2C peripheral will be used to connect to the NunChuck controller. You should also configure its GPIO Settings to **use pins D15/PB8 for SCL and D14/PB9 as SDA**; this can be done by clicking on the pins PB8 and PB9 of the chip diagram and selecting from the pull-down menu the SCL and SDA modes for these two pins (See Fig.13). This is very important.

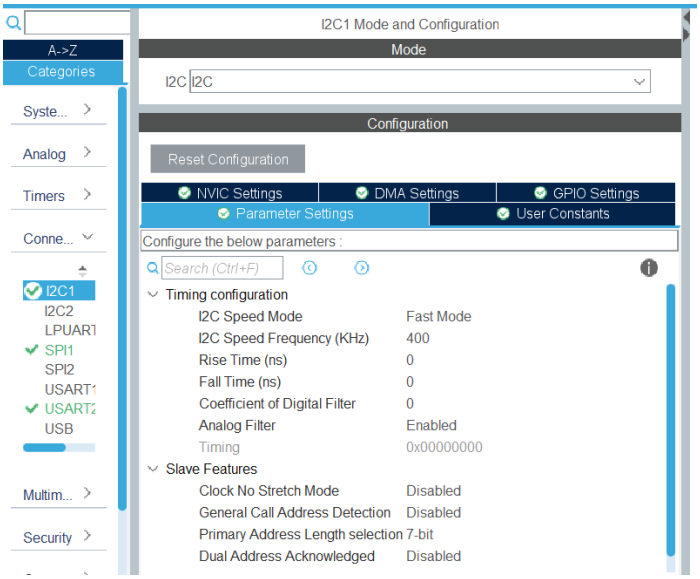

Figure 12: Configuration of I2C1 device in STM32CubeMX/CubeIDE – Parameter Settings.

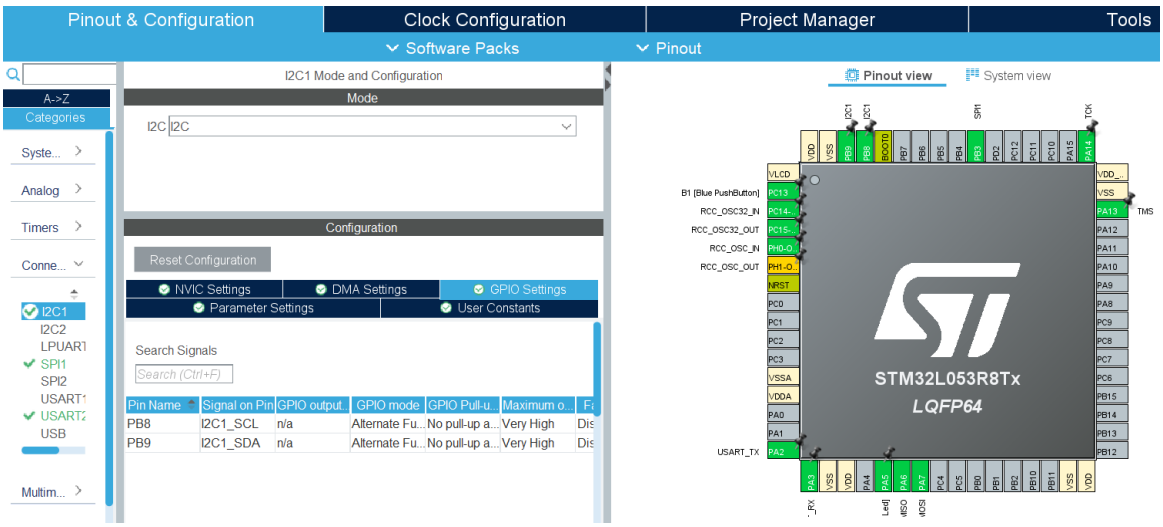

Figure 13: Configuration of I2C1 device in STM32CubeMX/CubeIDE – GPIO Settings.

Copy the LCD/ and UGUI/ folders (used first time in previous lab) into your own project inside Drivers/ folder. Also, copy from the provided main.c the code portions that you are missing inside your newly created main.c.

Create the necessary connections to the LCD display module – if you have not kept it already connected from the previous lab.

Then, also create the connections to the Wii NunChuck controller.

Wii NunChuck -> Nucleo Board
GND (-) -> GND
VCC (+) -> 3.3V
Clock (c) -> **PB8** (SCL of I2C1 peripheral of MCU)
Data (d) -> **PB9** (SDA of I2C1 peripheral of MCU)

Before building, go to Project -> Properties ->C/C++ General -> Paths and Symbols. Add to your paths Drivers/LCD and Drivers/UGUI. Also, inside the Project Explorer of the CubeIDE, expand Drivers/UGUI and right-click on ugui_sim_x11.c select Resource Configurations -> Exclude from build… Do the same for ugui_sim.c.

Now, finally, build the project and program the board. Observe operation and comment. You should see the circle move on the LCD display as you control it with the NunChuck (see Fig.14).

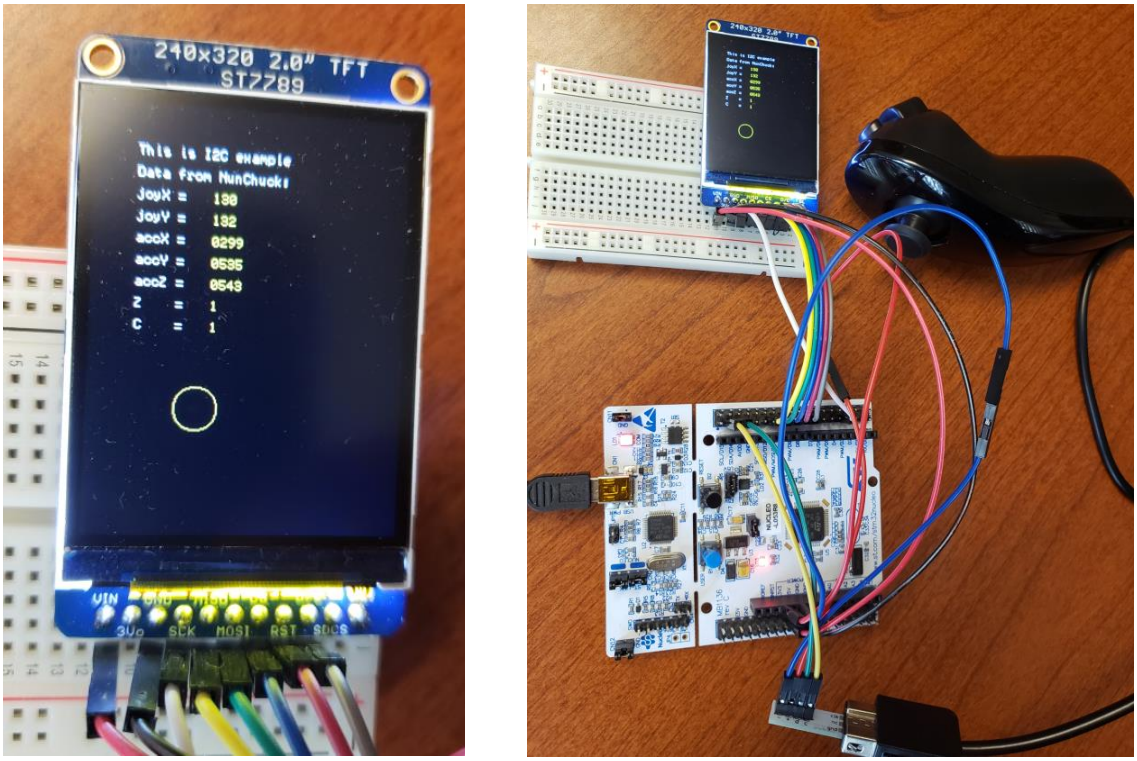

Figure 14: Demo of example 1.

Example 2: Example 1 with C++ coding of NunChuck
In this example, we replicate the project from example 1. But, everything related to the NunChuck is coded in C++. This is done so that you start thinking in terms of coding in C++ as well – which is much more elegant.

Create a new project and use the provided code in the files for this lab. Build and test the functionality, which should be exactly the same as of example 1.

5. Lab Assignment

Create a new application, similar to the example 1 from this lab, but with this functionality: Instead of using the joystick x, y data to move the circle, you must use the accelerometer data. The circle will move based on how the user pitches and rolls the NunChuck. Another change you must implement is to add a crosshair to circle so that we now move a crosshair circle rather than just a simple circle. The color of the crosshair should be red.

6. References and Credits

[1] Textbook:
Carmine Noviello, Mastering STM32 - Second Edition, 2022, Available to purchase online:
https://leanpub.com/mastering-stm32-2nd <--- Buy from
https://github.com/cnoviello/mastering-stm32-2nd <--- Code examples

[2] Pointers on I2C related information:
--Tutorial 1: http://www.best-microcontroller-projects.com/i2c-tutorial.html
--Tutotial 2: http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html
--Tutorial 3: http://embedded-lab.com/blog/?p=2583
--Wikipedia entry: http://en.wikipedia.org/wiki/I%C2%B2C

[3] Wii NunChuck related:
--Wikipedia, Wii remote: http://en.wikipedia.org/wiki/Wii_Remote
--Wiibrew, Wiimote/Extension Controllers:
http://wiibrew.org/wiki/Wiimote/Extension_Controllers
--Dangerousprototype: http://dangerousprototypes.com/docs/Wii_Nunchuck_quick_guide
--NXP (Freescale Semiconductor), Tilt sensing using linear accelerometers, 2012.
https://www.nxp.com/docs/en/application-note/AN3461.pdf
--Adafruit Wii NunChuck Breakout: https://learn.adafruit.com/adafruit-wii-nunchuck-breakout-adapter/arduino-use
--Wii NunChuck adaptor: https://www.sparkfun.com/products/9281
--Some repositories on GitHub (good starting point for code):
https://github.com/search?q=arduino%20nunchuck&type=repositories