Chapter 22
I2C: Inter-Integrated Circuit

Dr. Yifeng Zhu
Electrical and Computer Engineering, U of Maine
Additions and Edits by Prof Mark Lawford
Computing and Software, McMaster University

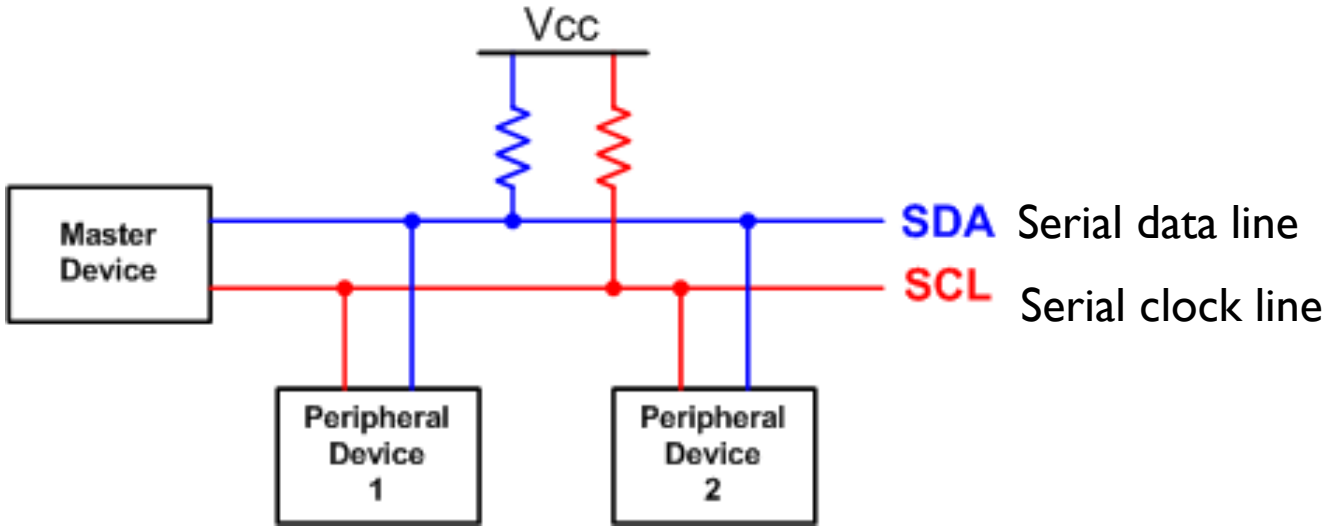
Fall 2017

Inter-Integrated Circuit (I2C)

- Designed for low-cost, medium data rate applications by Philips in the early 1980's
  - Original purpose: connect a CPU to peripheral chips in a TV-set
  - Today: a de-facto standard for 2-wire communications
  - Since October 10, 2006, no licensing fees are required to implement the I²C protocol. However, fees are still required to obtain I²C slave addresses allocated by NXP (acquired Philips).

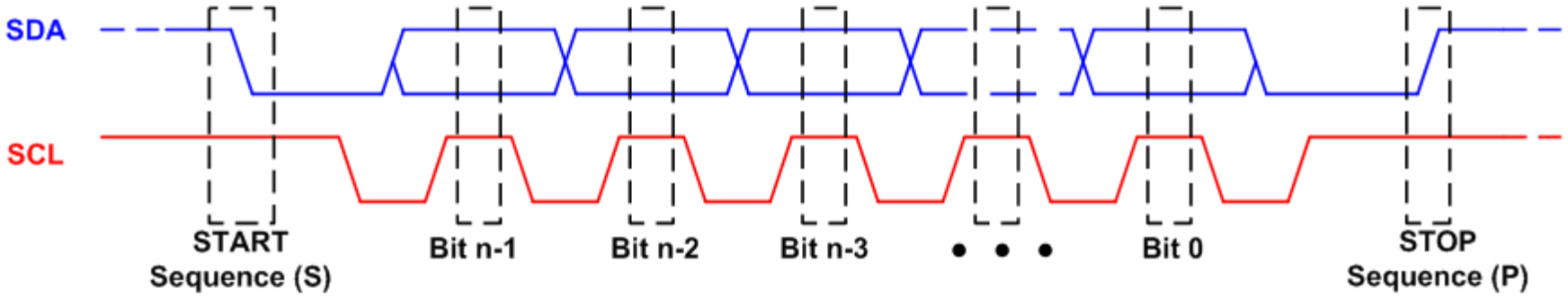
- Characteristics
  - Serial, byte-oriented
  - Multi-master, multi-slave
  - Two bidirectional open-drain lines, plus ground
    - Serial Data Line (SDA)
    - Serial Clock Line (SCL)
    - SDA and SCL need to pull up with resistors

Inter-Integrated Circuit (I2C)



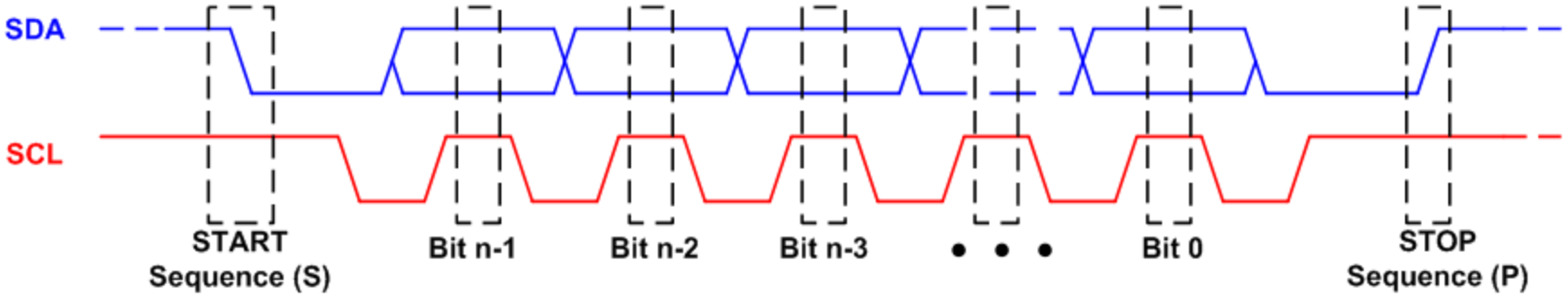
- Up to 100 kbit/s in the standard mode, up to 400 kbit/s in the fast mode, and up to 3.4 Mbit/s in the high-speed mode.
- SDA and SCL have to be open-drain
  - Connected to positive if the output is 1
  - In high impedance state if the output is 0
- Each Device has an unique address (7, 10 or 16 bits). Address 0 used for broadcast
- STM32L's internal pull-up is too weak (internal 100KΩ)
- External pull-up (4.7 kΩ for low speed, 3 kΩ for standard mode, and 1 kΩ for fast mode).

Timing Diagram



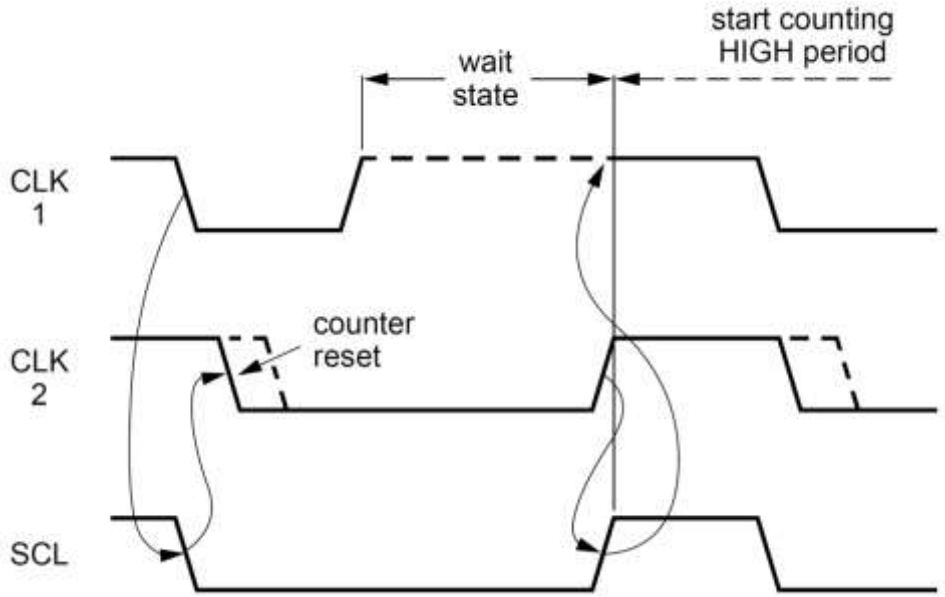
- A **START** condition is a high-to-low transition on SDA when SCL is high.
- A **STOP** condition is a low to high transition on SDA when SCL is high.
- The address and the data bytes are sent most significant bit first.
- Master generates the clock signal and sends it to the slave during data transfer

Multiple Masters



- "Wired-AND" bus: A sender can pull the lines to low, even if other senders are trying to drive the lines to high
- In single master systems, arbitration is not needed.
- Arbitration for multiple masters:
  - During data transfer, the master constantly checks whether the SDA voltage level matches what it has sent.
  - When two masters generate a START setting concurrently, the first master which detects SDA low while it has actually intended to set SDA high will lose the arbitration and let the other master complete the data transfer.

Clock Synchronization



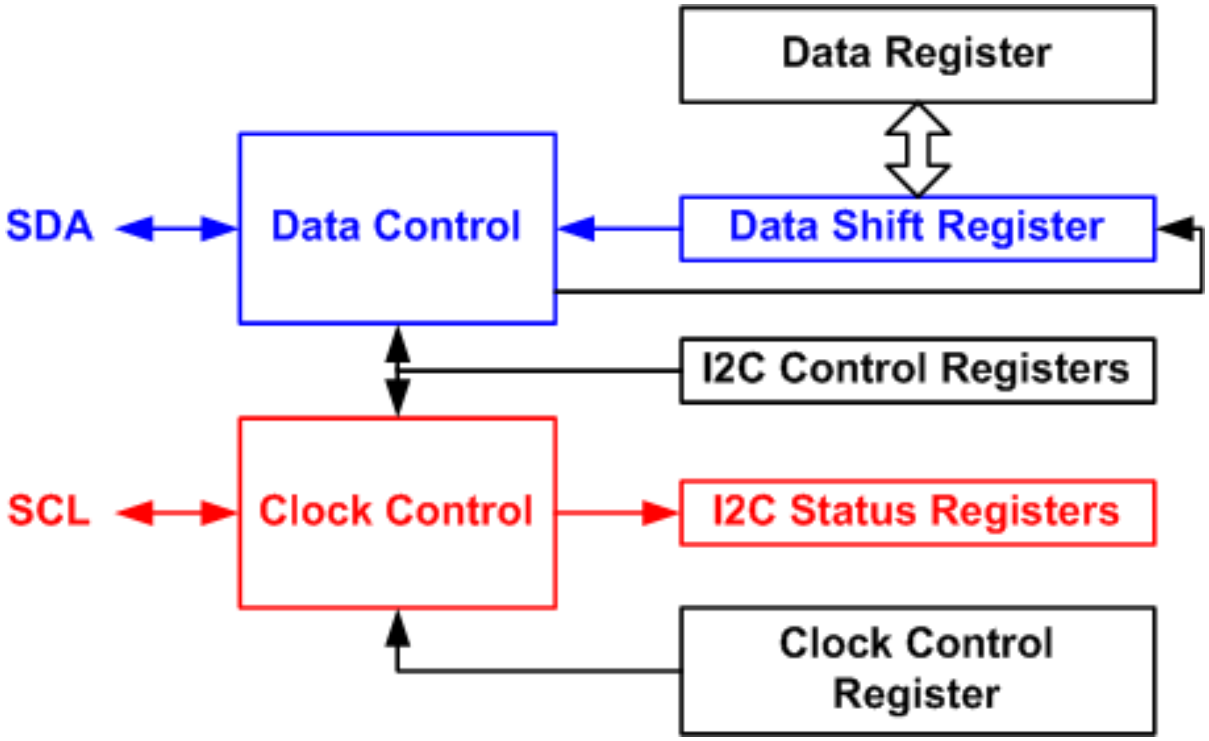
Source: I2C Specifications

- Clock synchronization is needed when there are multiple masters.
- **Wired-AND** connection for clock synchronization
  - Each master has a counter. Counter resets if SCL goes LOW. When the counter counts down to zero, the master releases SCL and thus SCL goes high.
  - SCL remains LOW if any master pulls it LOW.
  - When all masters concerned have counted off their LOW period, the clock line is released and goes HIGH.
  - After going high, all masters start counting their HIGH periods. The first master to complete its HIGH period pulls the SCL line LOW again.

Working Modes

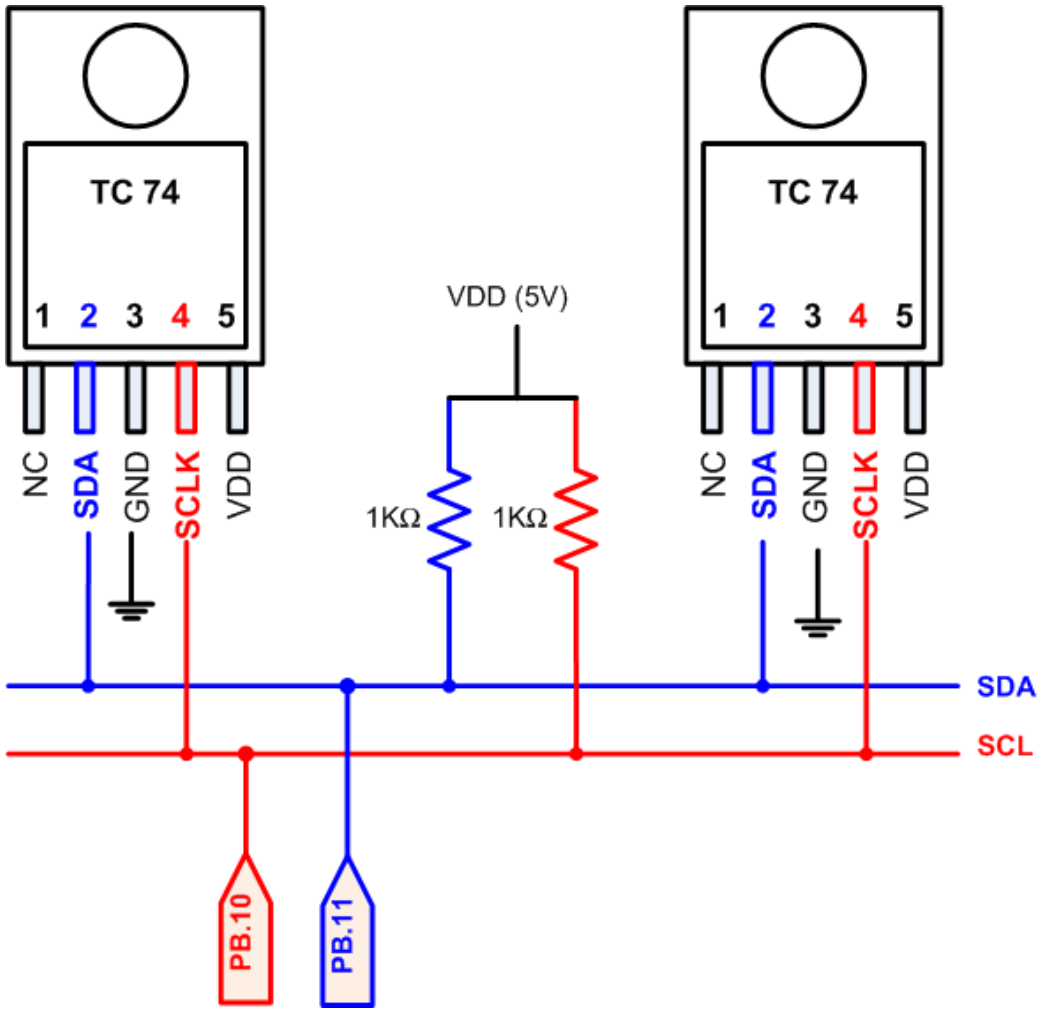
- Master-sender□
  - Master issues START and ADDRESS, and then transmits data to the addressed slave device
- Master-receiver□
  - Master issues START and ADDRESS, and receives data from the addressed slave device
- Slave-sender□
  - Master issues START and the ADDRESS of the slave, and then the slave sends data to the master
- Slave-receiver□
  - Master issues START and the ADDRESS of the slave, and then the slave receives data from the master.

STM32L I2C Module

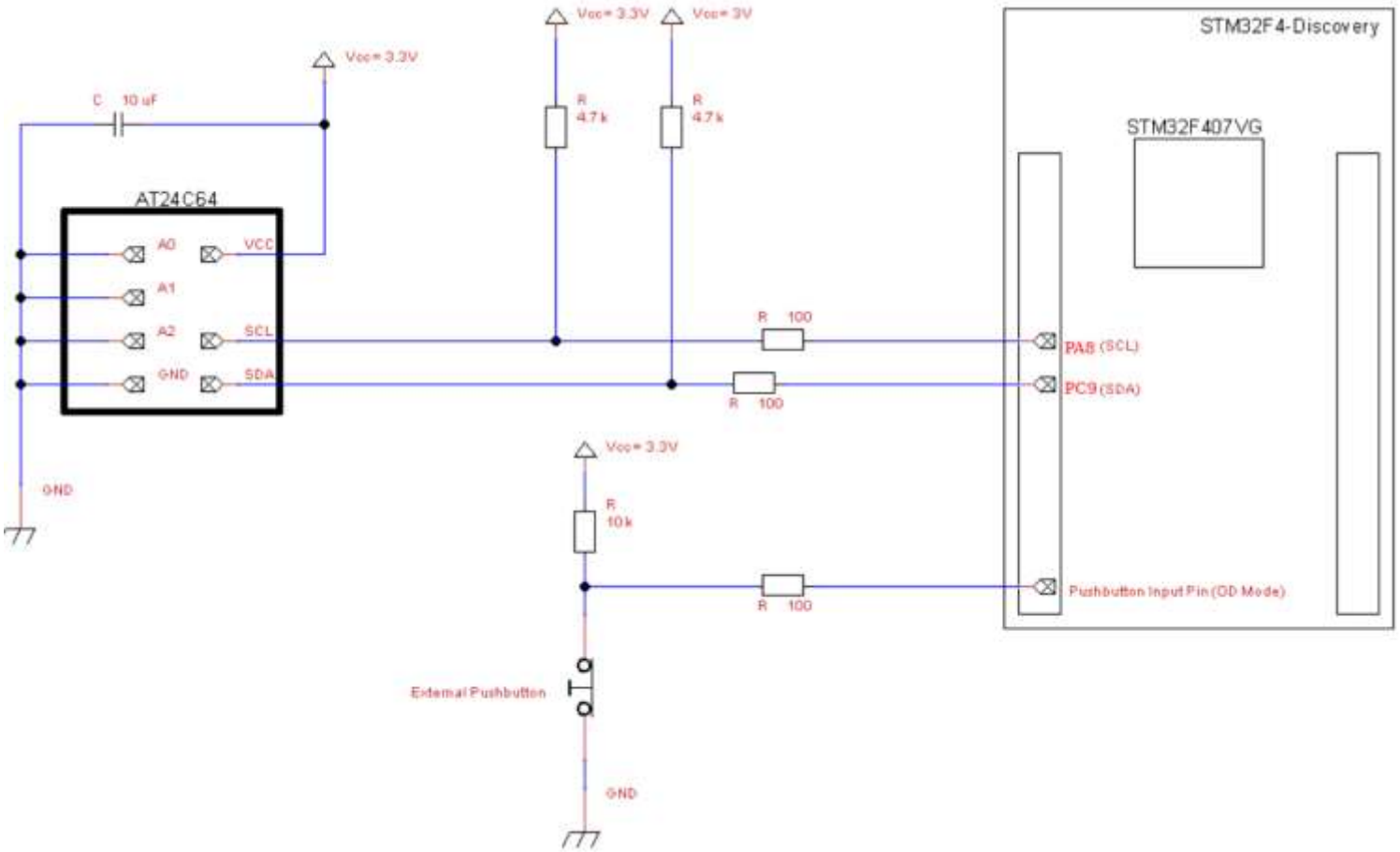


- During sending, the I²C hardware automatically sets the *TxE* flag in the status register if an acknowledge pulse is received from the slave.
- During receiving, the I²C hardware then automatically sets the *RxNE* flag in the status register if a byte has been successfully received.

Interfacing Serial Digital Thermal Sensor



Interfacing a Serial (I2C) EEPROM

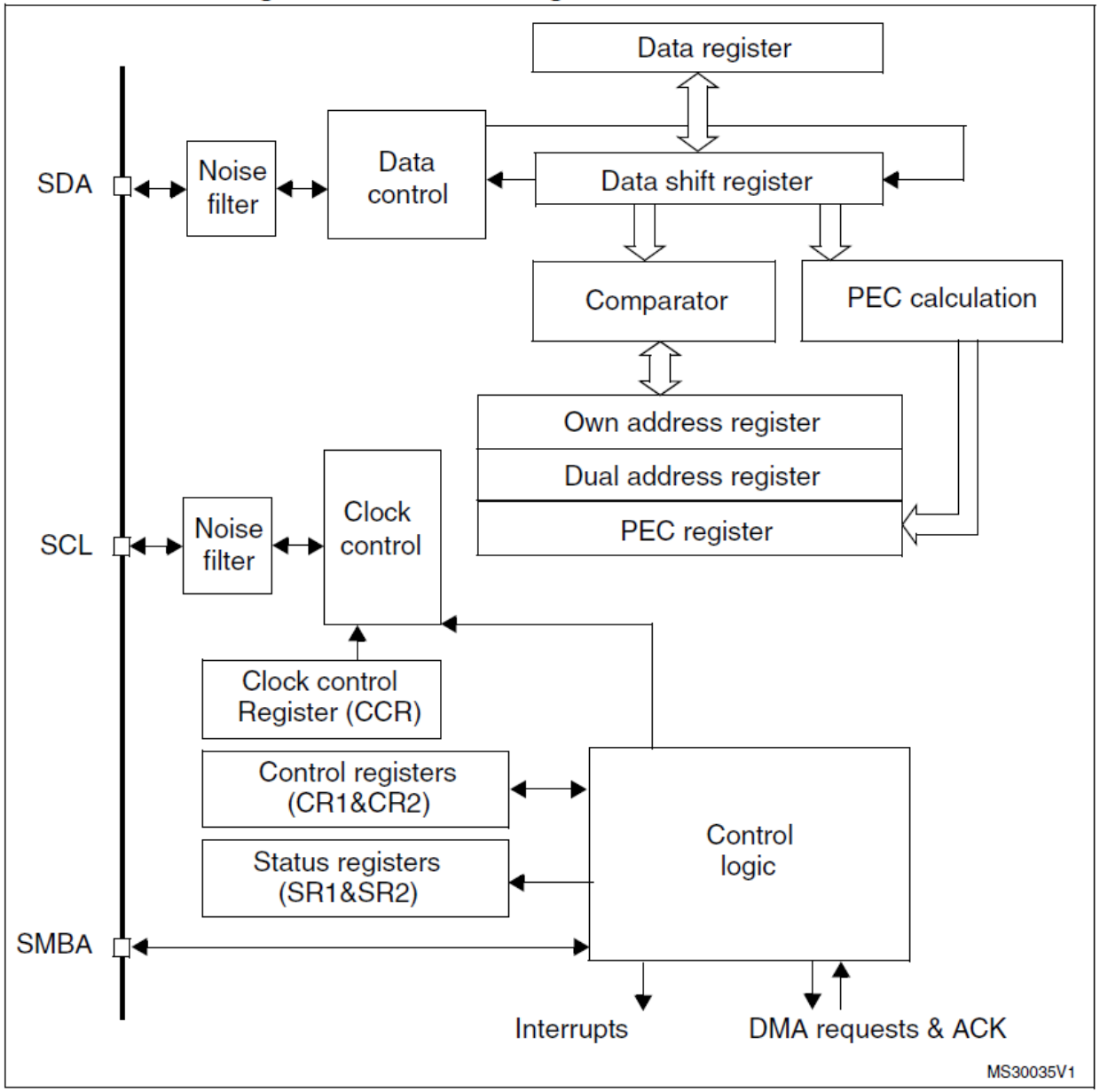


Why do we use PA8 and PC9?

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11	I2C1/2/3	SPI1/2/3/4/5/6	SPI2/3/SAI1	SPI3/USART1/2/3	USART6/UART4/5/7/8	CAN1/2/TIM12/13/14/LCD	OTG2_HS/OTG1_FS	ETH	FMC/SDIO/OTG2_FS	DCMI	LCD	SYS
	PC8	-	-	TIM3_CH3	TIM8_CH3	-	-	-	-	USART6_CK	-	-	-	SDIO_D0	DCMI_D2	-	EVENTOUT
	PC9	MCO2	-	TIM3_CH4	TIM8_CH4	I2C3_SDA	I2S_CKIN	-	-	-	-	-	-	SDIO_D1	DCMI_D3	-	EVENTOUT
	PA8	MCO1	TIM1_CH1	-	-	I2C3_SCL	-	-	USART1_CK	-	-	OTG_FS_SOF	-	-	-	LCD_R6	EVENTOUT

- **Table 12. STM32F427xx and STM32F429xx alternate function mapping**
- **STM32F427xx/STM32F429xx Datasheet (DocID024030)**

Figure 240. I²C block diagram for STM32F42x/43x



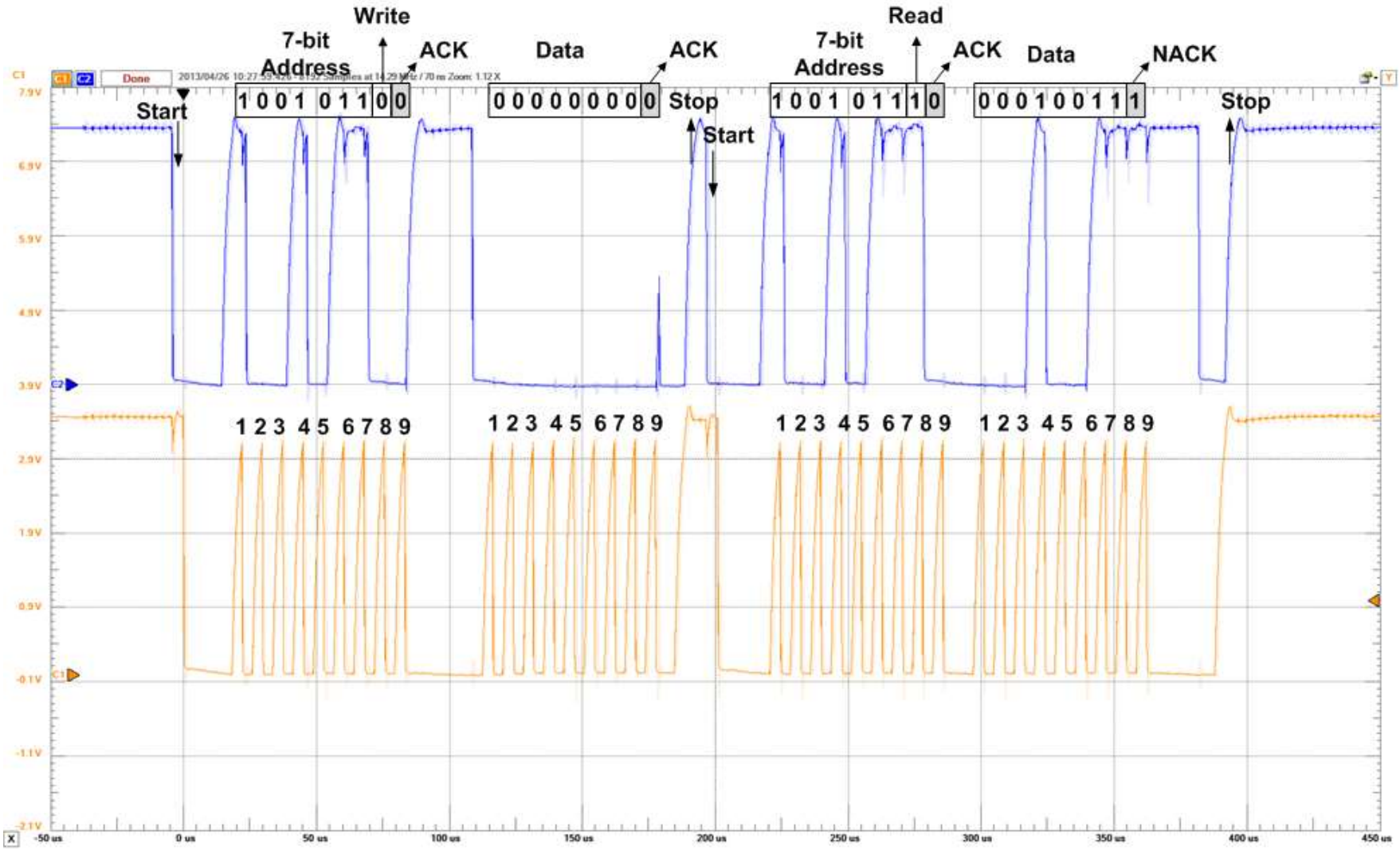
1. SMBA is an optional signal in SMBus mode. This signal is not applicable if SMBus is disabled.

I2C Peripheral

- From Ref Manual
- Can be Master or Slave (2 different addresses)
- Built-in //2serial
- Has noise filters
- Has 2 ISRs
  - Successful address/data Comm
  - Error condition

RM0090 Reference manual STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced ARM®-based 32-bit MCUs

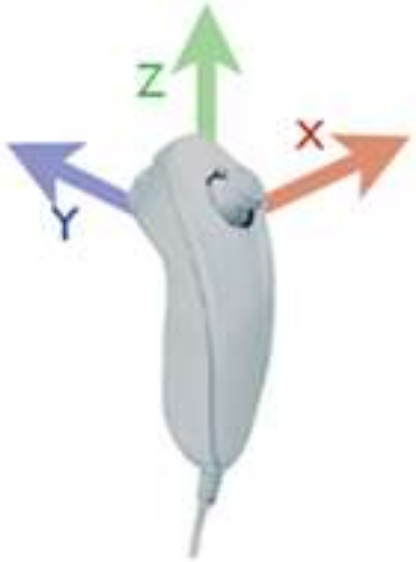
I2C Data

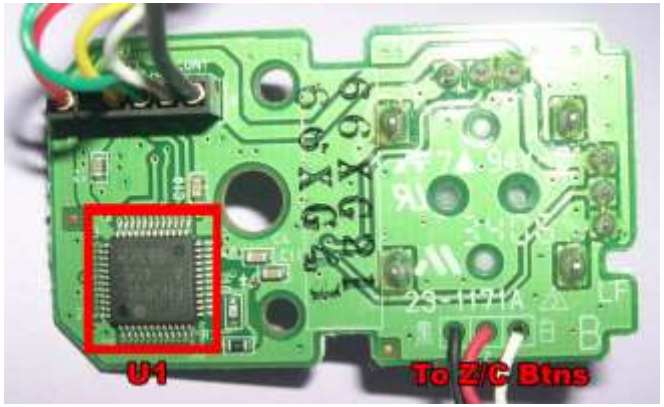

Write
7-bit Address
ACK
Data
ACK
Read
7-bit Address
ACK
Data
NACK
Start
1 0 0 1 0 1 1 0 0
0 0 0 0 0 0 0 0 0
Stop
Start
1 0 0 1 0 1 1 1 0
0 0 0 1 0 0 1 1 1
Stop
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
7.9V
6.9V
5.9V
4.9V
3.9V
2.9V
1.9V
0.9V
-0.1V
-1.1V
-2.1V
-50 us
0 us
50 us
100 us
150 us
200 us
250 us
300 us
350 us
400 us
450 us

I2C Initialization

```
void I2C_Initialization(void){
  // Enable the clock of I2C
  RCC->APB1ENR  |= RCC_APB1ENR_I2C2EN;              // I2C 2 clock enable

  // Reset the I2C (set and reset by software)
  RCC->APB1RSTR |= RCC_APB1RSTR_I2C2RST;
  RCC->APB1RSTR &= ~RCC_APB1RSTR_I2C2RST;

  // I2C Control register.  ACK is set and cleared by software, and cleared by hardware when PE=0.
  I2C2->CR1 |= I2C_CR1_ACK;             // Acknowledge after a byte is received
  I2C2->CR1 &= ~I2C_CR1_SMBUS;          // SMBus Mode: 0 = I2C mode;  1 = SMBus mode
  I2C2->CR2 &= ~I2C_CR2_FREQ;           // Clear Peripheral CLK frequency FREQ[5:0]
  I2C2->CR2 |= I2C_CR2_FREQ_1;          // Set Peripheral clock frequency as 2 MHz
  I2C2->CR2 |= I2C_CR2_ITEVTEN;         // Event Interrupt Enable
  I2C2->CR1 &= ~I2C_CR1_PE;             // Disable I2C to configure TRISE
  I2C2->TRISE &= ~I2C_TRISE_TRISE;     // Clear Maximum Rise Time
  I2C2->TRISE |= 17;                    // Set Maximum Rise Time for standard mode

  // I2C own address registers
  // Before the STM32 sends its start sequence it sits listening  to the I2C lines waiting for its address
  // This is helpful if STM32 is used as slave
  I2C2->OAR1 %= ~(0x01);

  I2C2->OAR1 = 0x00;   // Set up the address byte to select the slave device
  I2C2->OAR2 = 0x00;   // Set up the I2C own address2

  // I2C Clock control register
  I2C2->CCR &= ~I2C_CCR_FS;   // 0 = Standard Mode ((up to 100 kHz));
                              // 1 = Fast Mode (up to 400 kHz)
  I2C2->CCR &= ~I2C_CCR_CCR;
  I2C2->CCR |= 0x08;
  I2C2->CR1 |= I2C_CR1_PE;   // Enable I2C2
}
```

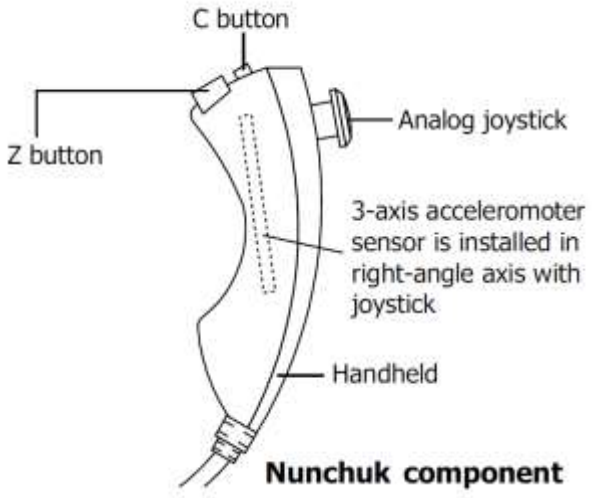
Wii Nunchuck

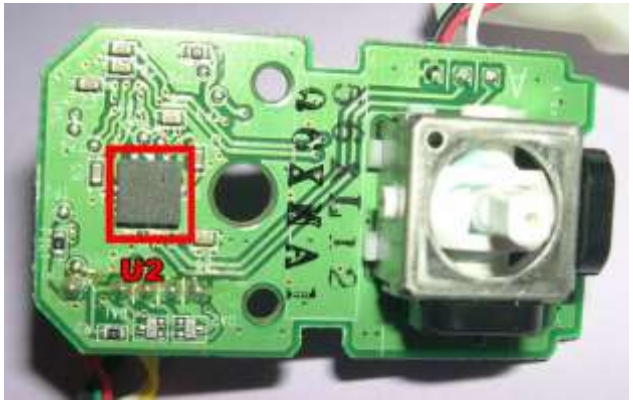




- 3 Axis accelerometer.
- Can capture X, Y and Z motion
- STMicroelectronics **LIS3L02AL**
  - 3.0 V to 3.6 V supply voltage
  - ~2 mW power consumption
  - ±2 g full-scale
  - 3 acceleration channels plus multiplexed analog output
  - Ratiometric output voltage
  - Power-down mode
  - Embedded self-test
  - 10000 g high shock survivability

Wii Nunchuck





Offset	Data							
0x00	Joystick X							
0x01	Joystick Y							
0x02	Accelerometer X (bit 9 to bit 2)							
0x03	Accelerometer Y (bit 9 to bit 2)							
0x04	Accelerometer Z (bit 9 to bit 2)							
0x05	Accel. Z (bit 1)	Accel. Z (bit 0)	Accel. Y (bit 1)	Accel. Y (bit 0)	Accel. X (bit 1)	Accel. X (bit 0)	C button	Z button

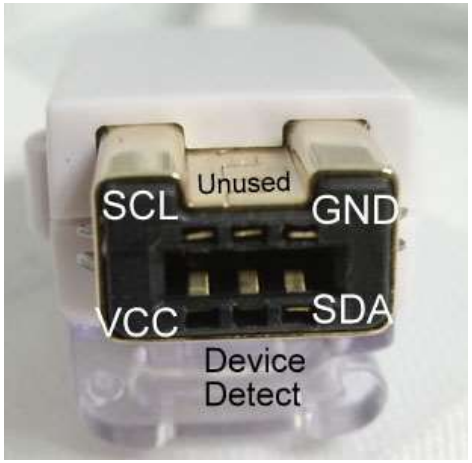
- Two push buttons (labeled as C and Z)
- An 8-bit 2-axis analog joystick (X, Y) and
- A 10-bit 3-axis accelerometer sensor (X, Y, and Z)
- Communication is encrypted.
- One possible decoding is

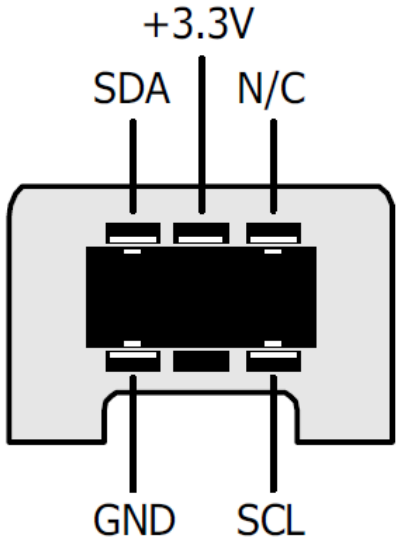
  Data = (Received data XOR 0x17) + 0x17

Joystick X	0x80 = Center, 0x00 = Full left, 0xFF = Full right
Joystick Y	0x80 = Center, 0x00 = Full up, 0xFF = Full down
Acceleration	0 – 1023.
Button	0 = pressed, 1 = released

- I2C Standard Mode (100Kbps)

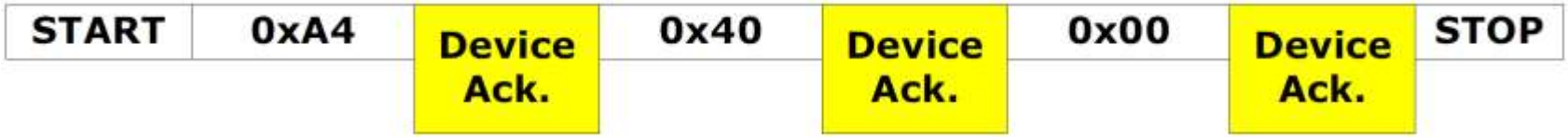
Interfacing Wii Nunchuck





Nunchuck Commands

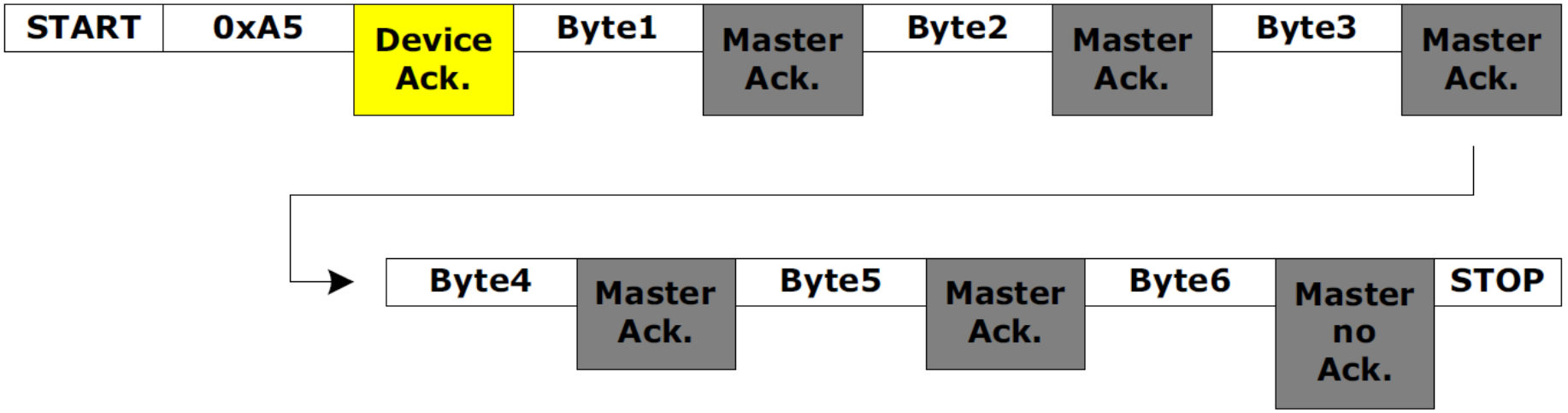
Initialization Command



Conversion Command

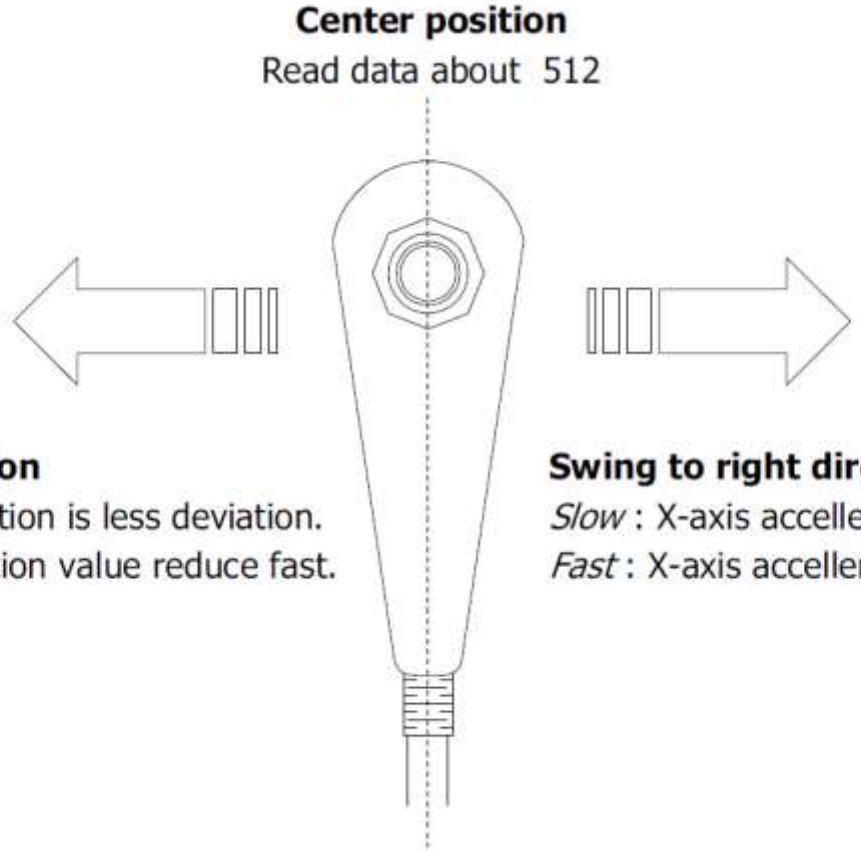
- Each Nunchuck has Two Slave Addresses
  - Slave command address: 0xA4
  - Slave data address: 0xA5

Nunchuck Read Data



Offset	Data							
0x00	Joystick X							
0x01	Joystick Y							
0x02	Accelerometer X (bit 9 to bit 2)							
0x03	Accelerometer Y (bit 9 to bit 2)							
0x04	Accelerometer Z (bit 9 to bit 2)							
0x05	Accel. Z (bit 1)	Accel. Z (bit 0)	Accel. Y (bit 1)	Accel. Y (bit 0)	Accel. X (bit 1)	Accel. X (bit 0)	C button	Z button

Left and Right

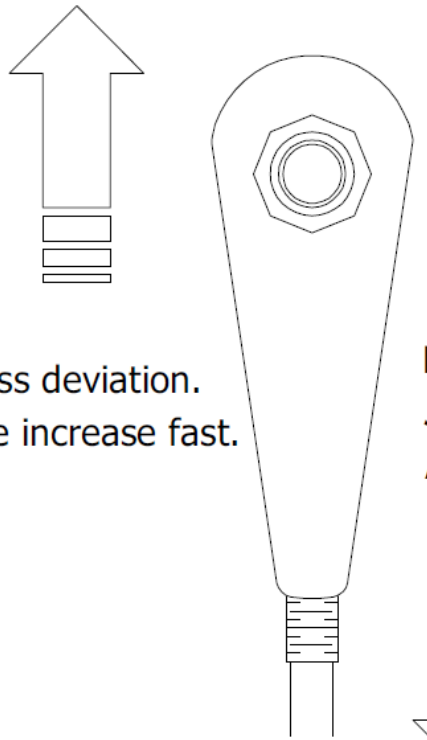


Swing to left direction
Slow : X-axis accelleration is less deviation.
Fast : X-axis accelleration value reduce fast.

Swing to right direction
Slow : X-axis accelleration is less deviation.
Fast : X-axis accelleration value increase fast.

Image from ZX-NUMCHUK (#800339)

Forward and Backward



Center position
Read data about 512

Move forward direction
Slow : Y-axis accelleration is less deviation.
Fast : Y-axis accelleration value increase fast.

Move backward direction
Slow : Y-axis accelleration is less deviation.
Fast : Y-axis accelleration value reduce fast.

Image from ZX-NUMCHUK (#800339)

Up and Down

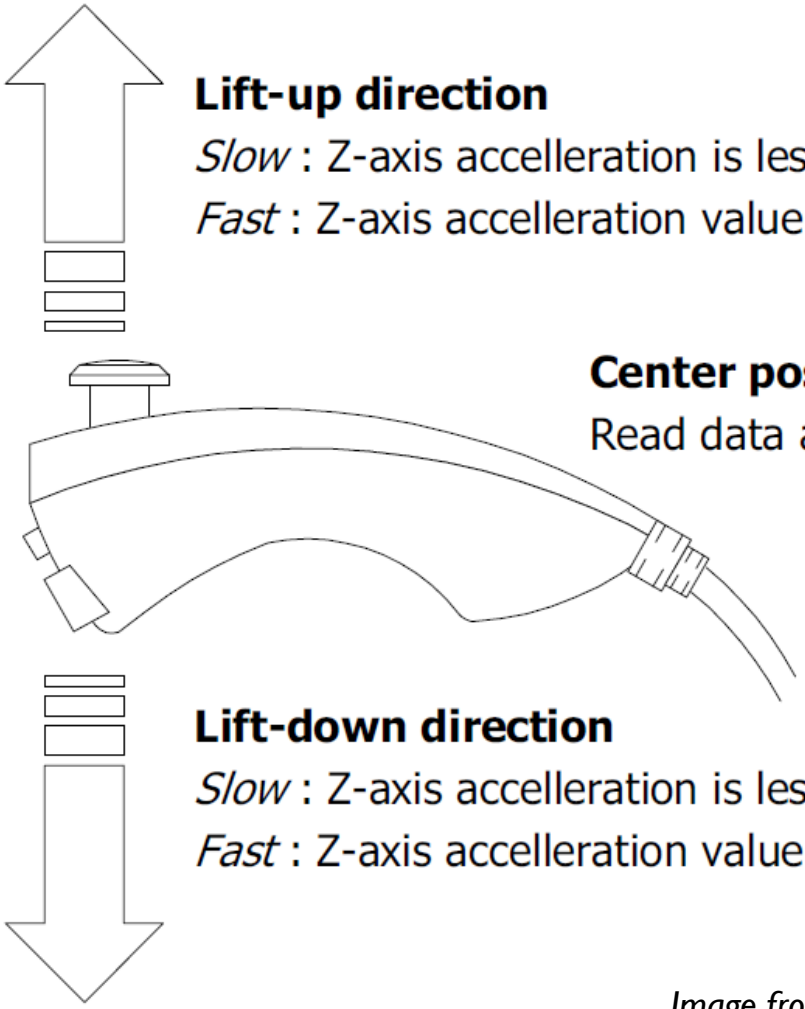


Image from ZX-NUMCHUK (#800339)

References

- Materials are adopted from:
  - ZX-NUNCHUK Wii-Nunchuk interface board
  - Chad - Wii Nunchuk/Wiichuck Adapter