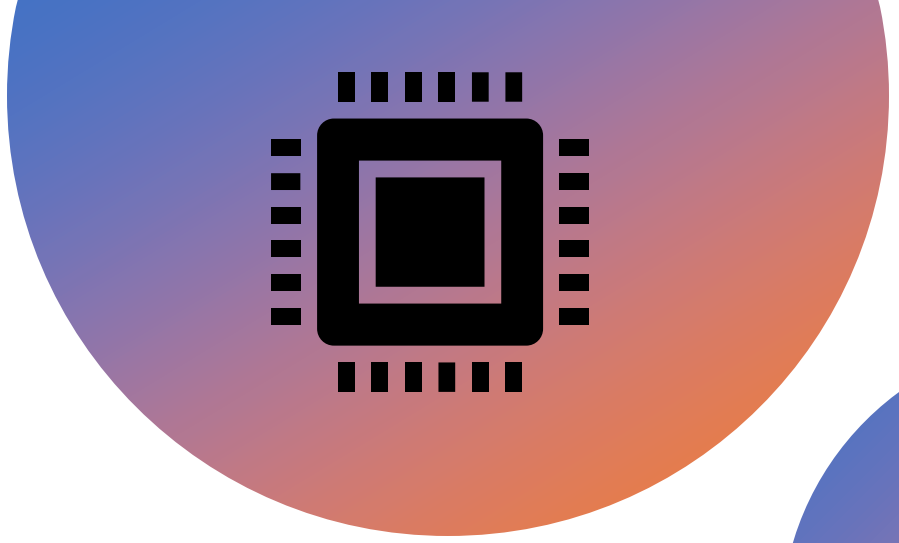
Workshop 2 – Introduction to Embedded Software

IEEE Microchips and Dip



Workshop 1 Recap

- We built a consisting of a microcontroller, an analog temperature sensor, and an I2C accelerometer
- We created an electrical schematic
- The system requirements follow:
  - The system must get temperature once a second
  - The system must get acceleration every 100 ms
  - The system must display these data for the user to view

What are we doing this workshop?

- Introduction to embedded software development
- Programming the STM32 to
  - Blink an LED
  - Say "Hello World"
  - Read the temperature sensor
  - Read the accelerometer
  - ~~Take over the world~~

Developing on an Embedded Device vs a Desktop

No built-in screen to debug programs

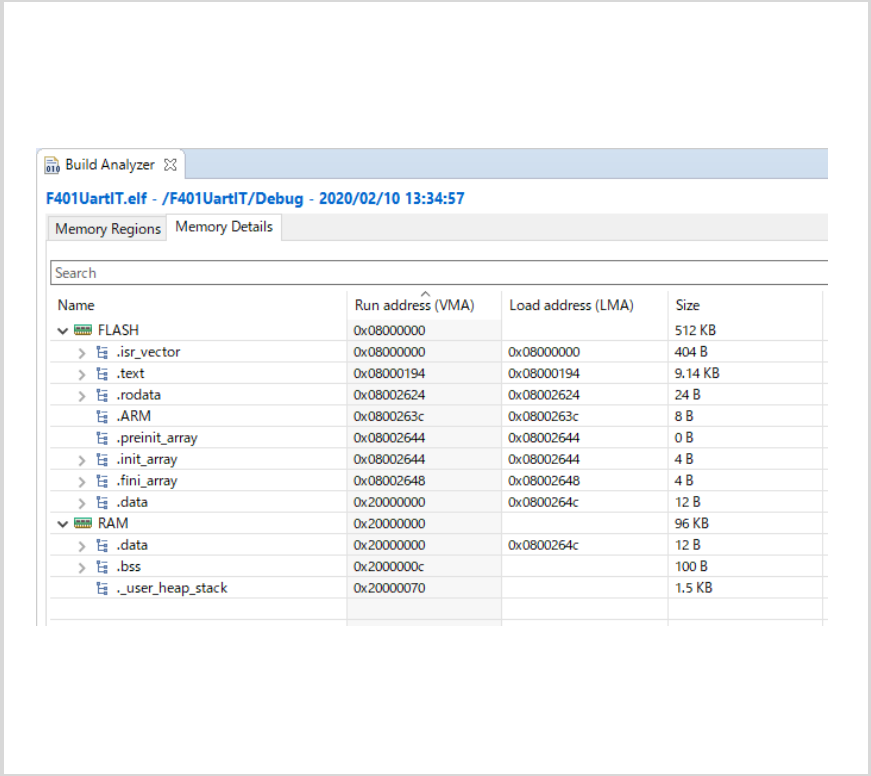
No file system

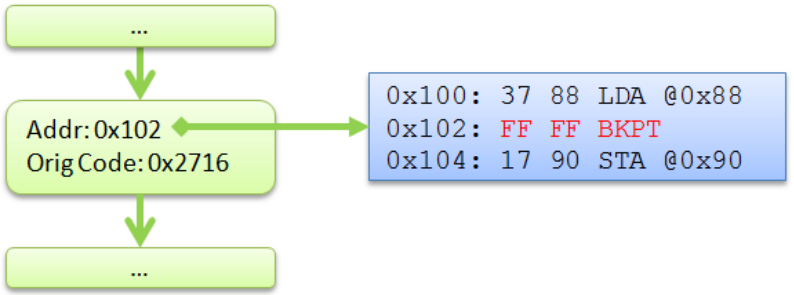
Limited memory

No floating point processor

Embedded Development Tools

- Breakpoints
- Memory viewer
- Serial to USB converters
- LEDs





C/C++

- Most embedded software is written in C
- C is supported by most compilers and
- Many embedded libraries and frameworks exist in only C
- STM32 allows for both programming language
- For many programs, C is sufficient

Why Don't We Use Arduino Framework?

Although convenient to use, it does not scale well for large programs

Targeted towards hobbist

Poorly managed dependencies

Too much abstraction, difficult to optimize and understand what is happening

Over-simplistic development environment, lacks many modern day debug tools

Without the Arduino Framework, do I need to code everything from scratch?

No, STM32 provides some code that abstracts the low-level register setting

STM32 Hardware Abstraction Layer (HAL)

The HAL abstracts most of the register writing

It provide basic functions to get the peripheral to work

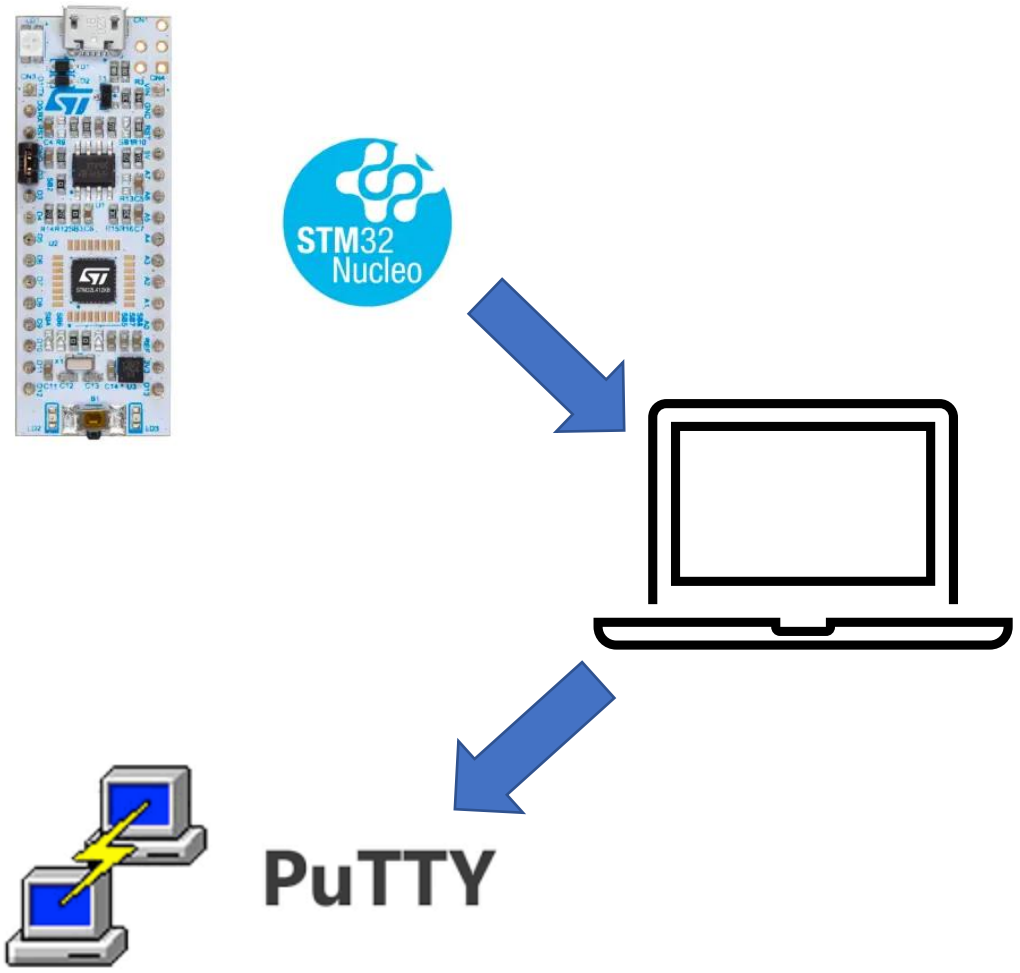
All functions are well documented by STM

STM32 Hello world – Blinky LED

1. Create New Project
2. Find the LED pinout
3. HAL function for turning on the LED

   ```
   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
   ```

4. Compile and upload to the microcontroller!



STM32 Hello world – UART

- UART (universal asynchronous receiver transmitter)
- AKA Serial communication
- Commonly used to communicate between the microcontroller and a computer using a UART to USB converter
- The STM32 Nucleo board has the UART to USB converter built in
- We will be using it to send a message to Putty or any serial terminal on your computer

Programming UART

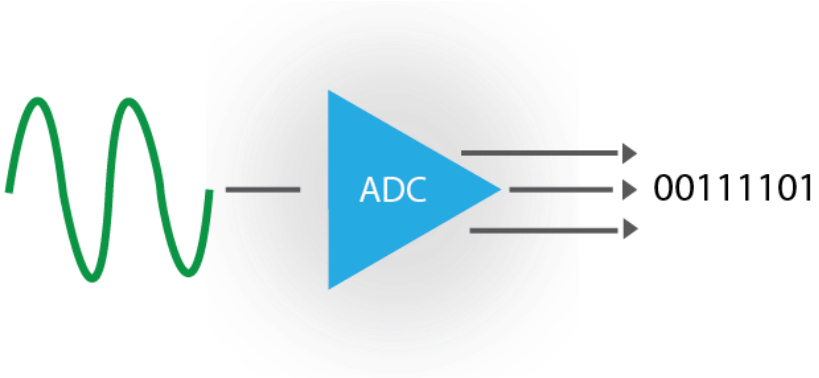
1. Configure UART peripheral on the STM32
2. HAL function for UART transmission
3. Read UART message in Putty

Reading the temperature sensor

The temperature sensor outputs an analog signal (voltage)

To read it we must use the STM32's Analog to Digital Converter (ADC)

The readings from the ADC can then be converted into a temperature unit that we understand



What does the ADC do

- Analog to Digital Converter
- Converts analog voltage to a digital signal in binary
- Allows the microcontroller to read voltage
- Many simple sensors use this method of communication
- The accuracy of the data will depend on the microcontroller's ADC resolution
- STM32L432 has 12-bit resolution

Programming ADC

1. Configure the ADC settings in the IDE
2. Read ADC conversion using the HAL
   - HAL_ADC_Start(&hadc1);
   - HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
   - HAL_ADC_GetValue(&hadc1);

Accelerometer

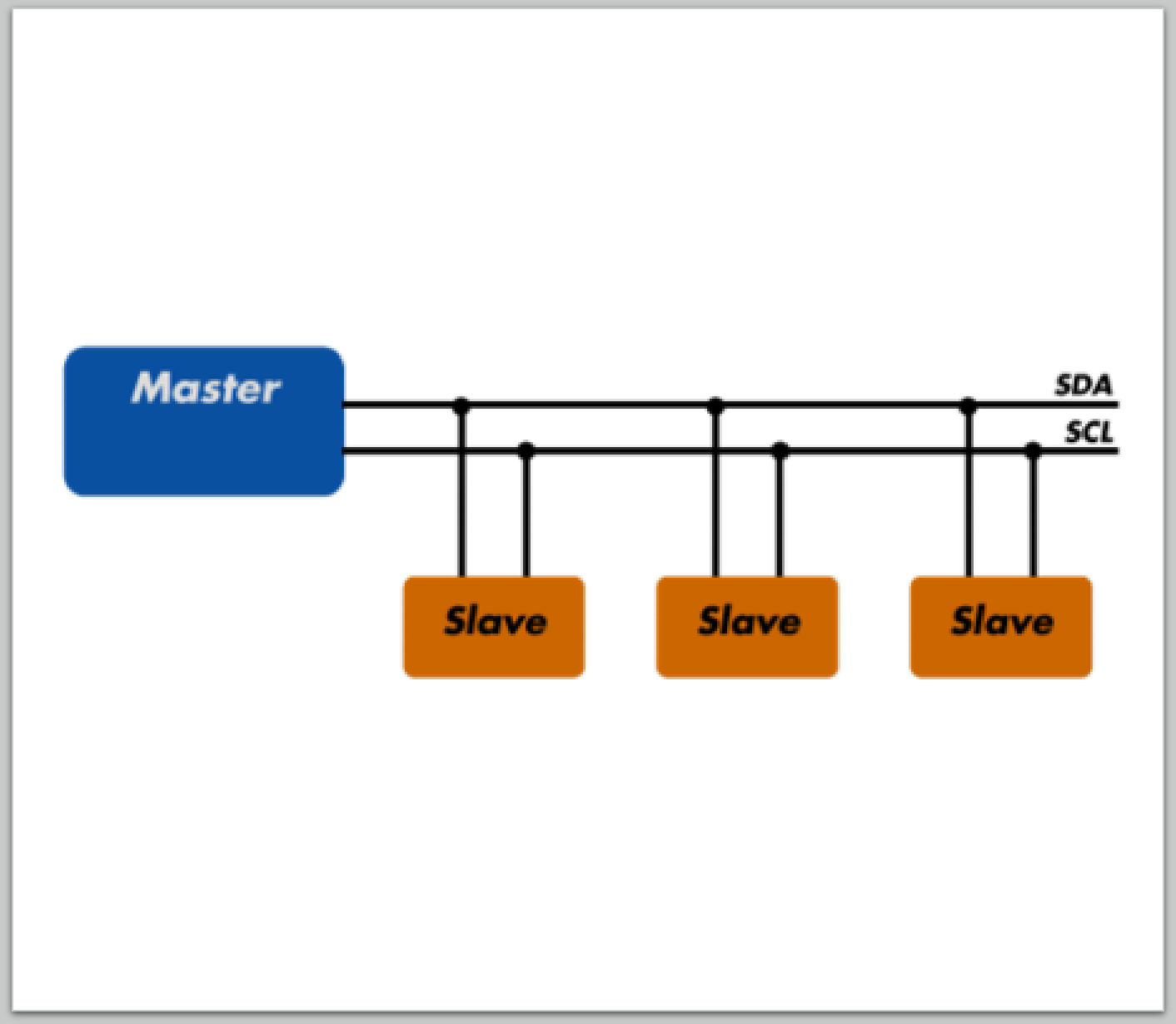
The accelerometer uses I2C to communicate

It can output acceleration in the X,Y, and Z axis

More complex sensor as we can write and read to the device

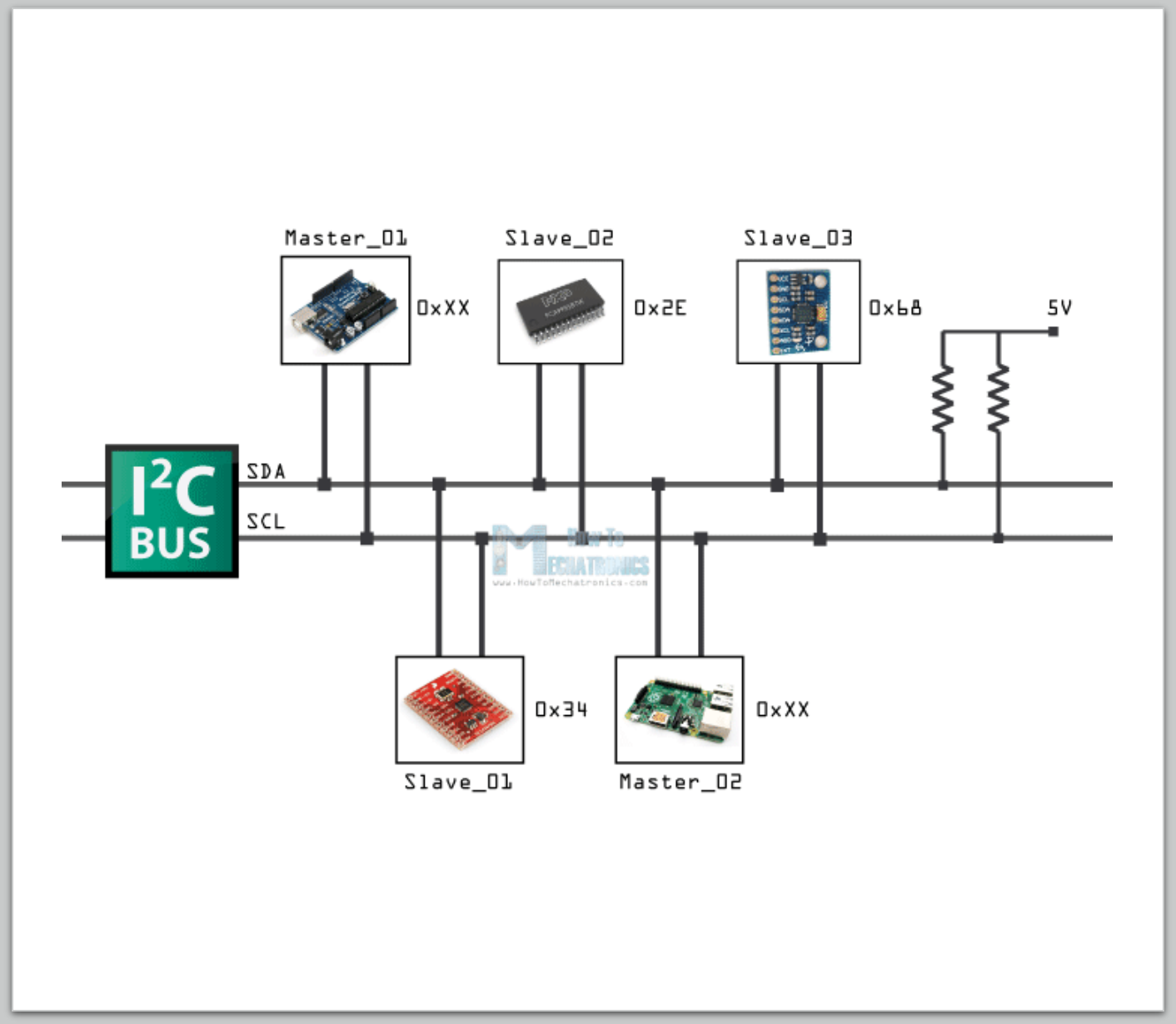
What is I2C

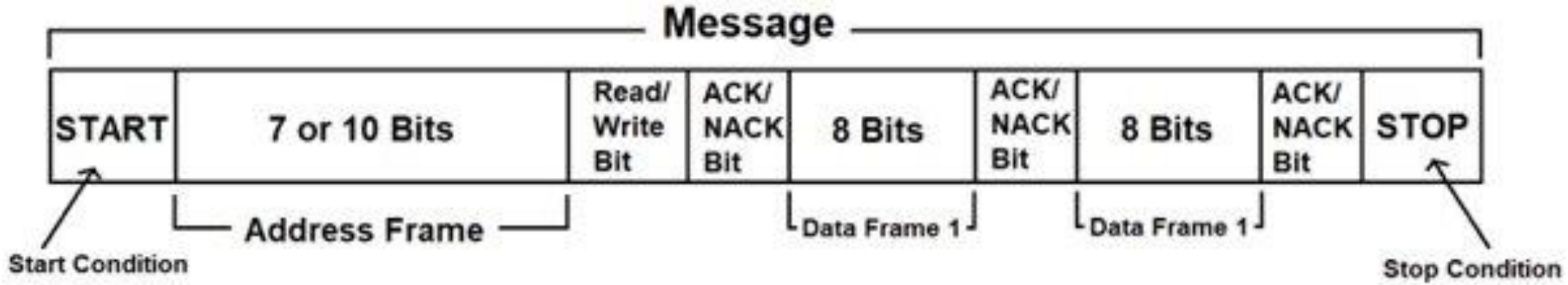
- I2C is a protocol used by many sensors and devices to talk to a microcontroller
- I2C can enable one microcontroller to talk to multiple devices
- Master-Slave(s) relation between the microcontroller and sensors
- Only requires 2 wires for communications



I2C- Hardware

- 2 wires are required for I2C communications
- SCL (Serial Clock) is used to synchronize communications
- SDA (Serial Data) is used to transmit data between the devices
- Only one device can talk at a time (half-duplex)
- Each device has an address, this is physically assigned, cannot be changed by software





I2C - Software

- MCU will need to send the device address that it wants to talk to
- MCU needs to specify if it wants to read or write
- Slave device will acknowledge it received the message
- MCU can transmit any data
- Slave will acknowledge after every byte

I2C – Sensor's Datasheet

2.4.2 I²C - inter-IC control interface

Subject to general operating conditions for Vdd and Top.

Table 7. I²C slave timing values

Symbol	Parameter	I²C standard mode(1)		I²C fast mode(1)		Unit
		Min	Max	Min	Max	
$f_{(SCL)}$	SCL clock frequency	0	100	0	400	kHz
$t_{w(SCLL)}$	SCL clock low time	4.7		1.3		µs
$t_{w(SCLH)}$	SCL clock high time	4.0		0.6		
$t_{su(SDA)}$	SDA setup time	250		100		ns
$t_{h(SDA)}$	SDA data hold time	0.01	3.45	0.01	0.9	µs
$t_{h(ST)}$	START condition hold time	4		0.6		µs
$t_{su(SR)}$	Repeated START condition setup time	4.7		0.6		
$t_{su(SP)}$	STOP condition setup time	4		0.6		
$t_{w(SP:SR)}$	Bus free time between STOP and START condition	4.7		1.3		

WHO AM I 🤔

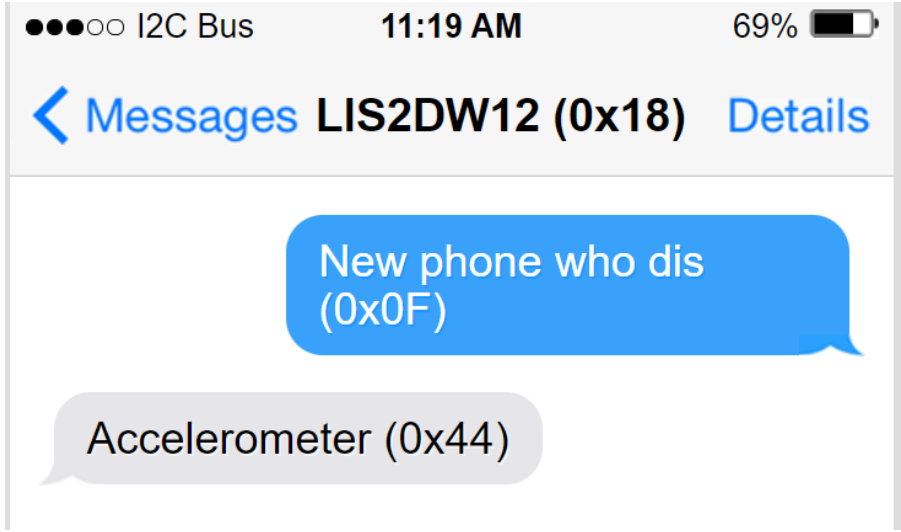
8.3 WHO_AM_I (0Fh)

Who_AM_I register (r). This register is a read-only register. Its value is fixed at 44h.

Table 26. **WHO_AM_I register default values**

0	1	0	0	0	1	0	0

We will be reading this register to make sure our I2C and accelerometer is working.



Programming I2C

1. Configure the I2C settings in the IDE
2. Write and Read from the sensor using I2C
   - HAL_I2C_Master_Transmit(handle, address, data, data length, timeout delay);
   - HAL_I2C_Master_Receive(handle, address, data, data length, timeout delay);

Reading Acceleration

8.12 OUT_X_L (28h)

X-axis LSB output register (r).

Table 47. OUT_X_L register

X_L7	X_L6	X_L5	X_L4	X_L3(1)	X_L2(1)	0	0

1. *If Low-Power Mode 1 is enabled, this bit is set to 0.*

The 8 least significant bits of linear acceleration sensor X-axis output. Together with the OUT_X_H (29h) register, it forms the output value expressed as a 16-bit word in 2's complement.

8.13 OUT_X_H (29h)

X-axis MSB output register (r).

Table 48. OUT_X_H register

X_H7	X_H6	X_H5	X_H4	X_H3	X_H2	X_H1	X_H0

The 8 most significant bits of linear acceleration sensor X-axis output. Together with the OUT_X_L (28h) register, it forms the output value expressed as a 16-bit word in 2's complement.

Challenge – Tap Detection

1

Configure the correct config registers

2

Read the register containing tap events

3

Determine if tap occurred