Embedded Systems Fundamentals

ENGD2103

Dr M A Oliver

michael.oliver@dmu.ac.uk

Lecture 3: I2C and Accelerometer

Contents

This lecture will include:-

- An overview of I2C:
  - What is the I2C Bus and what is it used for?
  - Bus characteristics
  - I2C Bus Protocol
  - Data Format
- Arduino Serial
- Accelerometer MPU6050 / GY-521 Module
- Wire library
- Example code

What is I2C?

- The name stands for “Inter-Integrated Circuit Bus”
- A Small Area Network connecting ICs and other electronic systems
- Originally intended for operation on one single board / PCB
- Synchronous Serial Signal
- Two wires carry information between a number of devices
  - One wire for the DATA
  - One wire for the CLOCK
- Originally designed by Philips (now NXP) which held the patent until 2006
- Atmel (later Microchip) used the name Two-Wire-Interface (**TWI**) to avoid licensing issues
- Licensing fees are no longer required to implement the I2C protocol.
  - The name **TWI** remained, hence we still use `#include <Wire.h>`
- Today, a variety of devices are available with I2C Interfaces
  - Microcontroller, EEPROM, Real-Timer, interface chips, LCD driver, A/D converter

What I2C Is Used For

- Data transfer between ICs and systems at relatively low rates
  - "Classic" I2C is rated to 100 Kbit/second
  - "Fast Mode" devices support up to 400 Kbit/second
  - "Fast mode Plus (Fm+)" is defined for up to 1 Mbit/second
  - "High Speed Mode" is defined for operation up to 3.4 Mbit/second
  - "Ultra Fast mode" uni-directional and up to 5 Mbit/second
    https://www.nxp.com/docs/en/user-guide/UM10204.pdf

- Reduces Board Space and Cost By:
  - Allowing use of ICs with fewer pins and smaller packages
  - Greatly reducing interconnect complexity
  - Allowing digitally controlled components to be located close to their point of use

I2C Bus Characteristics

- Includes electrical and timing specifications, and an associated bus protocol
- Two wire serial data & control bus implemented with the serial data (SDA) and clock (SCL) lines
  - For reliable operation, a third line is required: Common ground **(GND)**
- Unique start and stop condition
- Slave selection protocol uses a 7-Bit slave address
  - The bus specification allows an extension to 10 bits
- Bi-directional data transfer
- Acknowledgement after each transferred byte
- No fixed length of transfer

I2C Bus Characteristics

- True multi-master capability
  - Clock synchronization
  - Arbitration procedure
- Transmission speeds up to 100Khz (classic I2C)
- Max. line capacitance of 400pF, approximately 4 metres (12 feet)
- Allows series resistor for IC protection
- Compatible with different IC technologies

I2C Bus Definitions

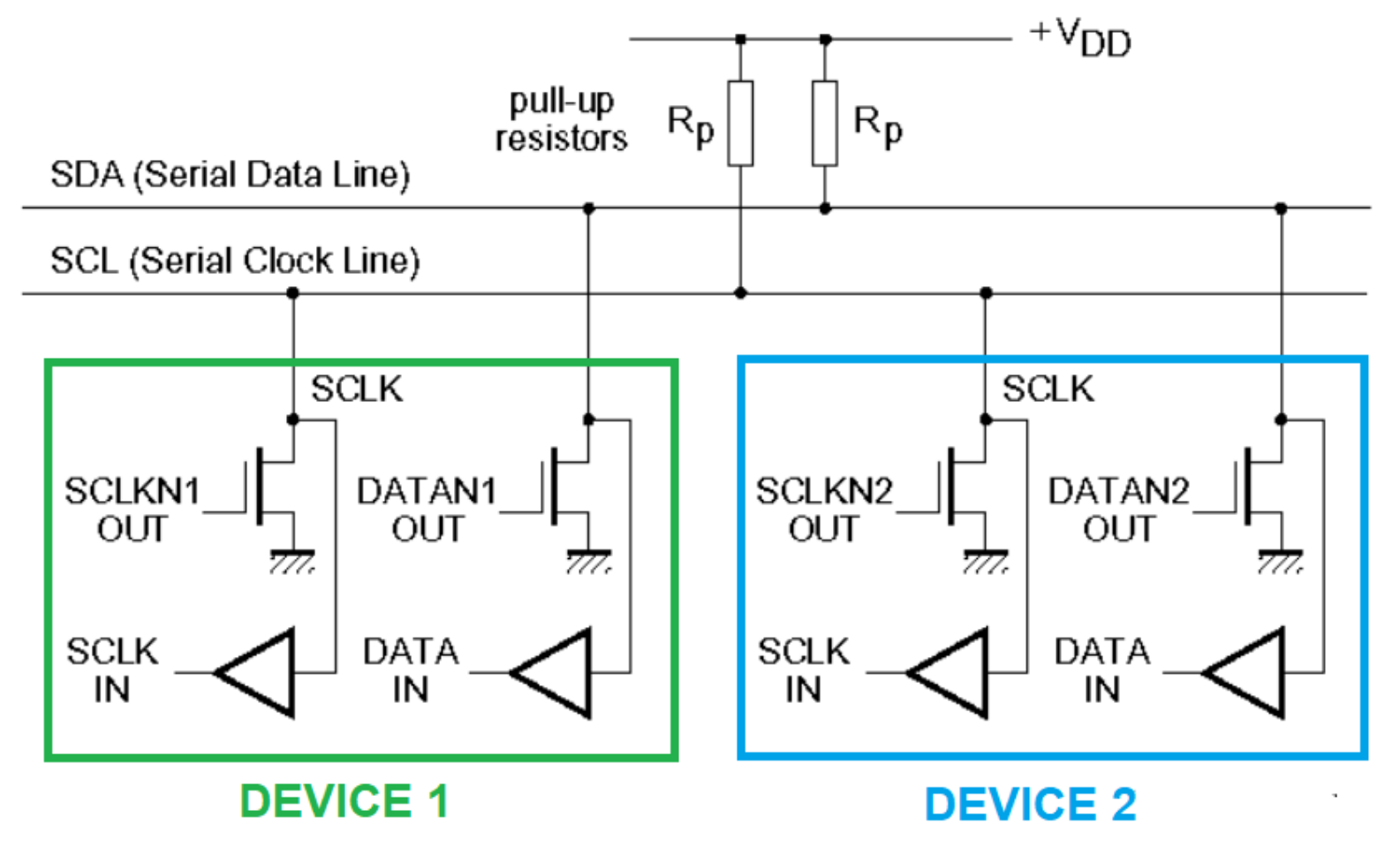
Master

- Initiates a transfer by generating **Start** and **Stop Conditions**
- Generates the clock
- Transmits the slave address
- Determines data transfer direction

Slave

- Responds only when addressed
- Timing is controlled by the clock line

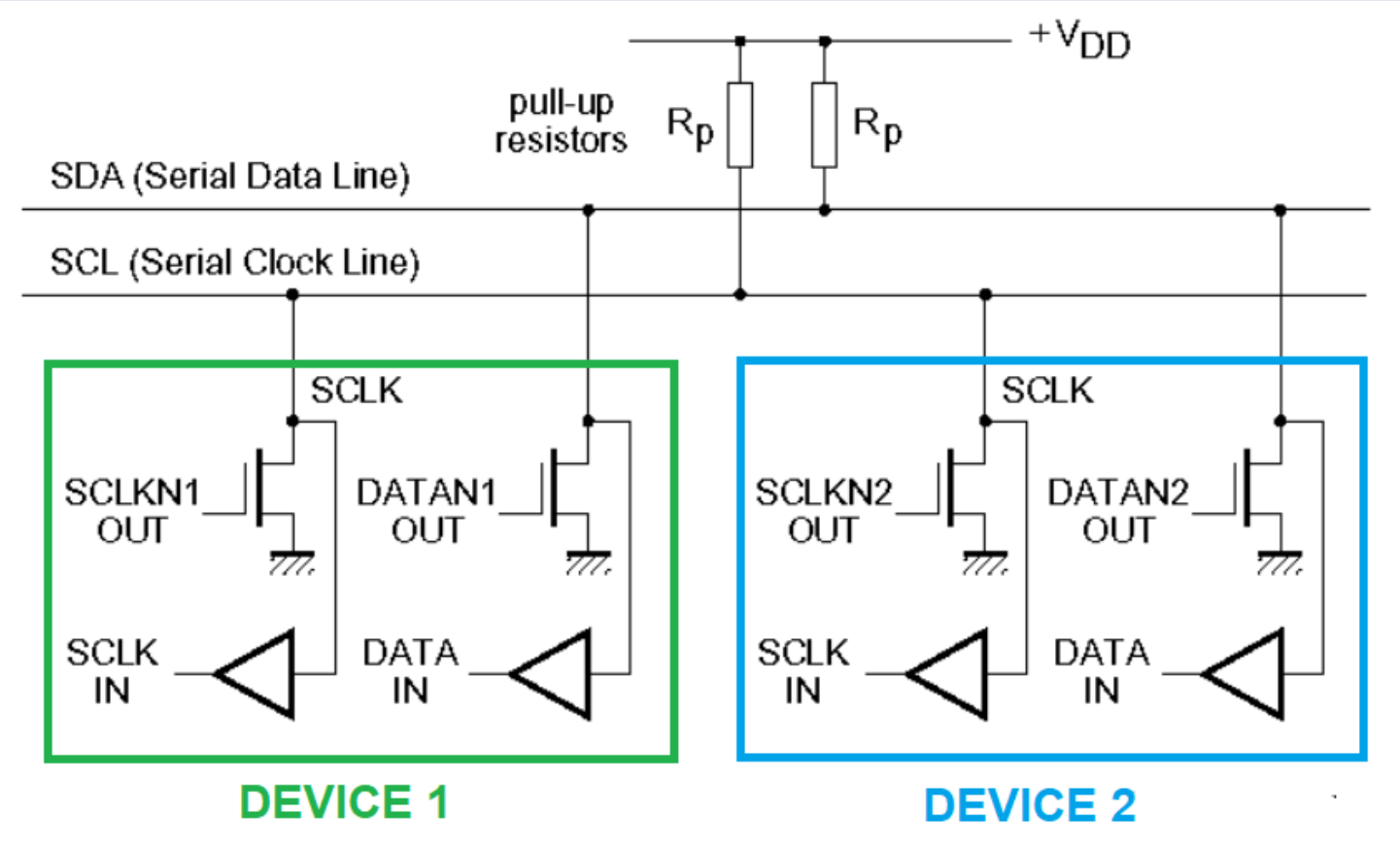
I2C Bus Configuration



I2C Hardware Details

- Devices connected to the bus must have an open drain or open collector output for serial clock and data signal
- The device must also be able to sense the logic level on these pins
- All devices have a common ground reference
- The serial clock and data lines are connected to Vcc (typically +5V) through pull up resistors
- At any given moment the I2C bus is in one of three modes:
  - Idle mode, or
  - in Master transmit mode, or
  - in Master receive mode.

I2C Electrical Aspects

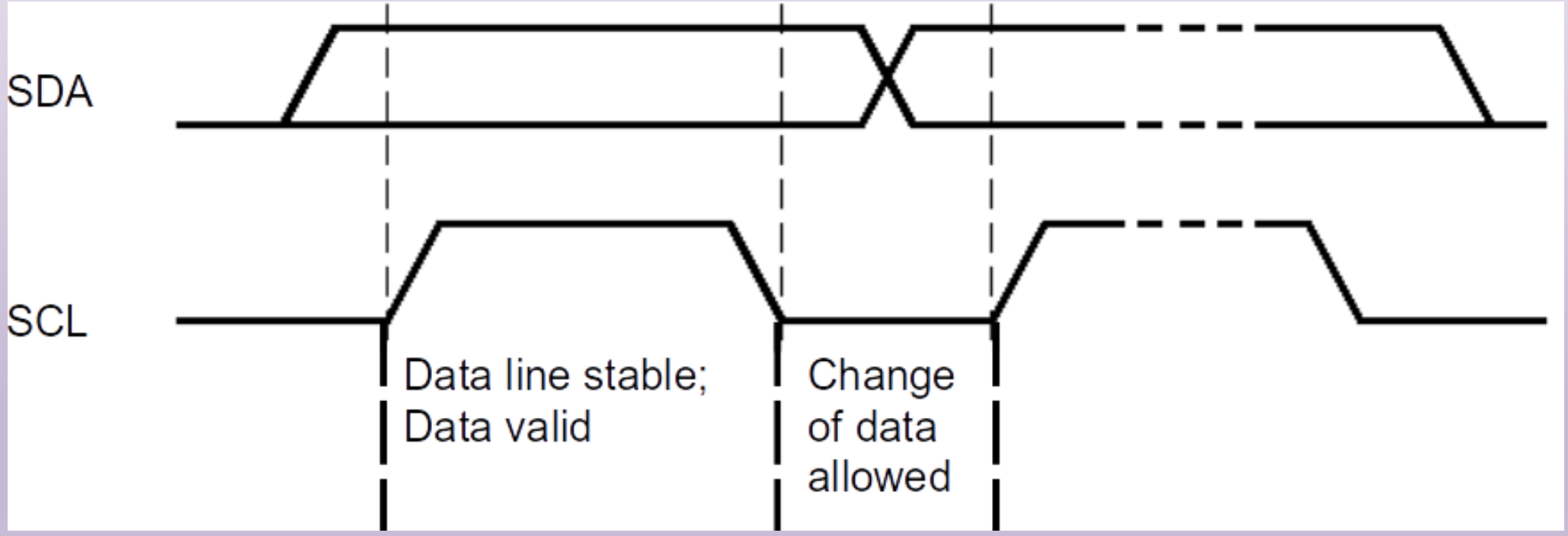


I2C devices are wire ANDed together.

If any single node writes a zero, the entire line is zero

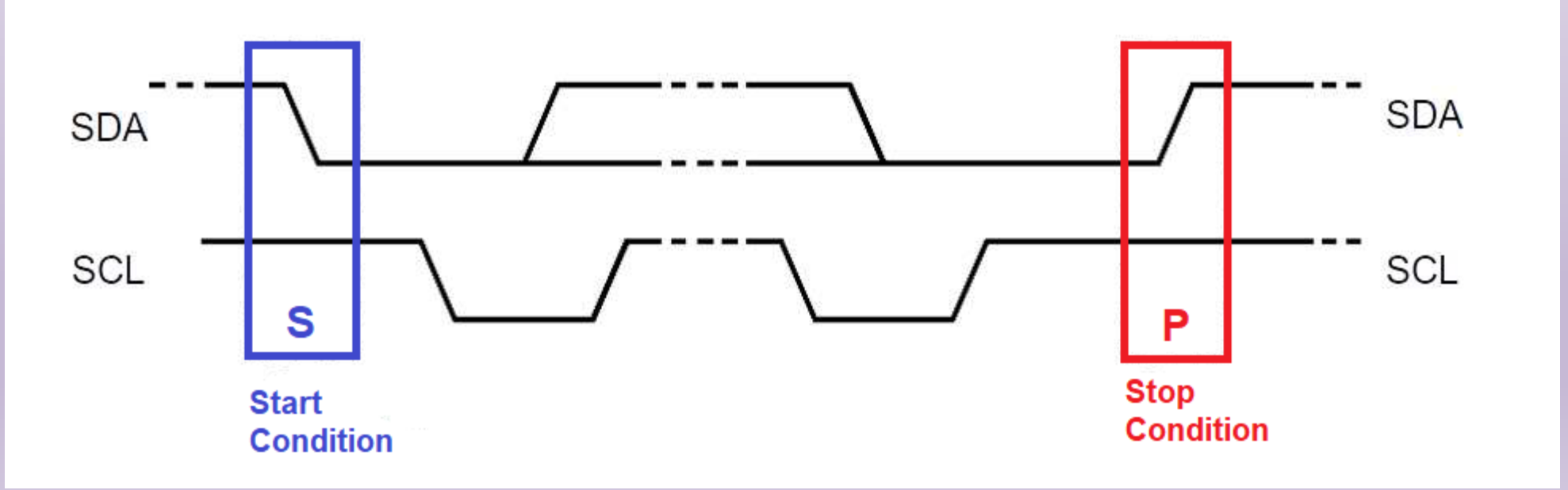
Bit Transfer on the I2C Bus

During normal data transfer, the data line only changes state when the clock is low



Start and Stop Conditions

- Stop and Start Conditions instantiated by the Master
- When the clock (**SCL**) line is **high**:-
  - A high-to-low transition of the data (SDA) line is a **Start Condition**
    The I2C bus is now considered busy.
  - A low-to-high transition of the data (SDA) line is a **Stop Condition**
  - The I2C bus is now considered idle.

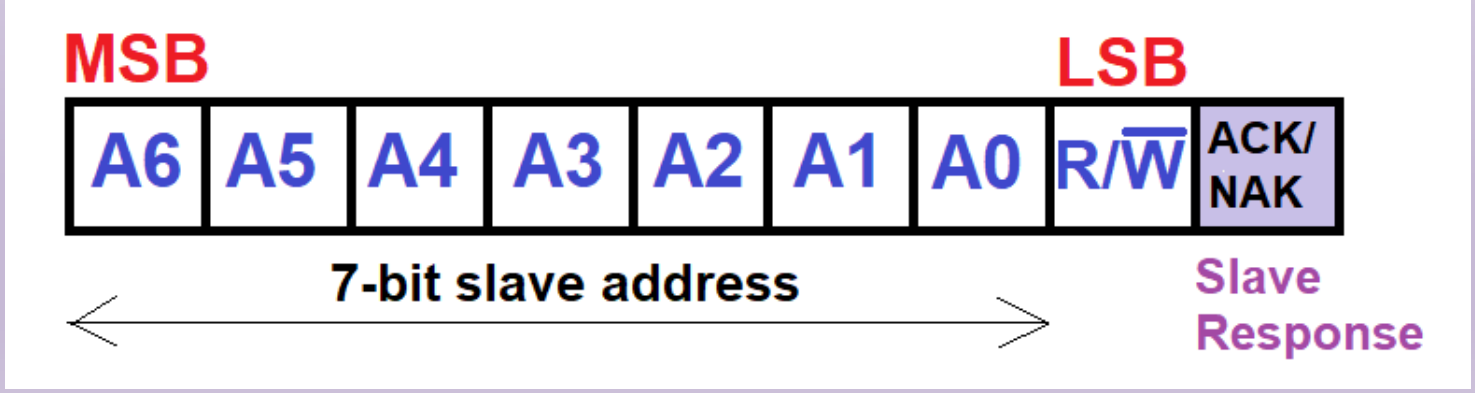


I2C Addressing

- An I2C bus is capable of supporting multiple I2C slave devices.
- Each slave device has an address (either 7 or 10 bits).
- A slave will only respond to transactions corresponding to its address and ignore other addresses.
- A slave device either has a fixed address, or an address that is partially fixed and partially selectable
  - Having partially selectable addresses can resolve address clashes.

First Byte In I2C Transaction

- This is the first byte to follow the Start Condition.
- The 7-most significant bits represent the 7-bit slave address (MSbit first)
- This is followed by a **R**ead / **W**rite bit:-
  - 1 indicates a read (from the Slave to the Master)
  - 0 indicates a write (from the Master to the Slave)
- Addressed Slave Acknowledges / Negative-Acknowledges (ACK/NAK) the transaction



I2C Bus Connections

Masters can be:-

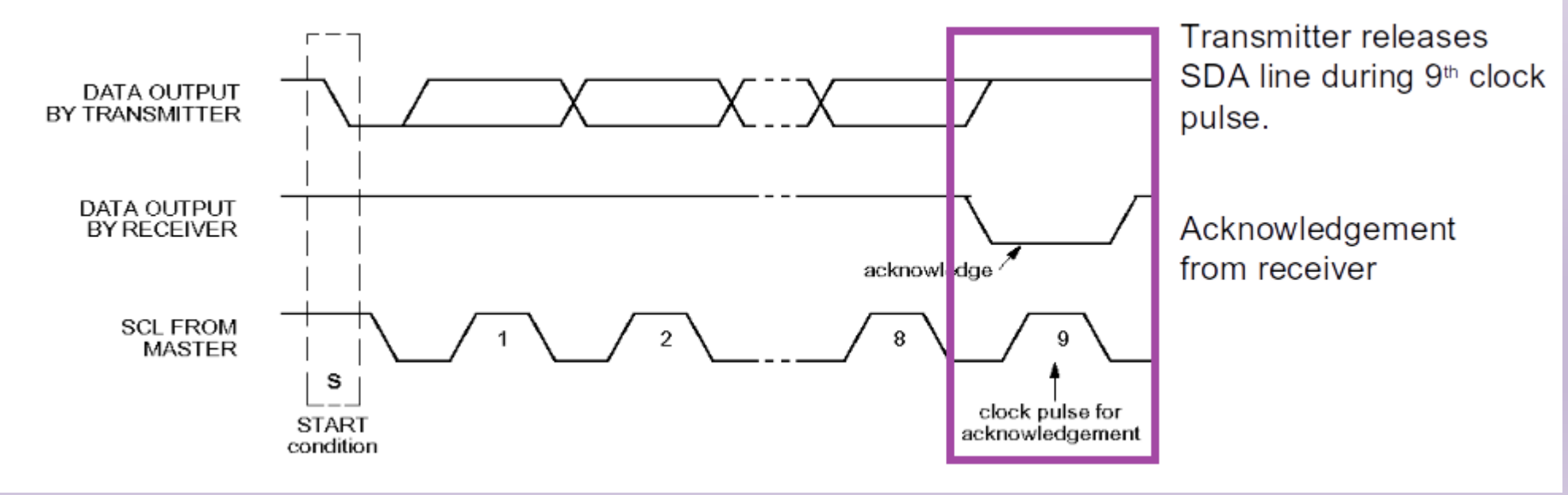
- Transmitter Only (e.g. writing to an I2C LCD display)
- Transmitter and Receiver (e.g. for interrogating an I2C sensor)

Slaves can be:-

- Receiver Only (e.g. reading information to be sent to an I2C LCD display)
- Receiver and Transmitter (e.g. for an I2C sensor, receiving requests and transmitting responses).

I2C Acknowledge

- Master/slave receivers pull data line low for one clock pulse after reception of a byte
- Master receiver leaves data line high after receipt of the last byte requested
- Slave receiver leaves data line high on the byte following the last byte it can accept



I2C Acknowledge

An Acknowledgement can be generated by a Slave in response to a Master Transmitter.

- On correct reception of a data byte
- On correct reception of an address byte.

An Acknowledgement can be generated by a Master Receiver in response to a Slave.

- On correct reception of a data byte

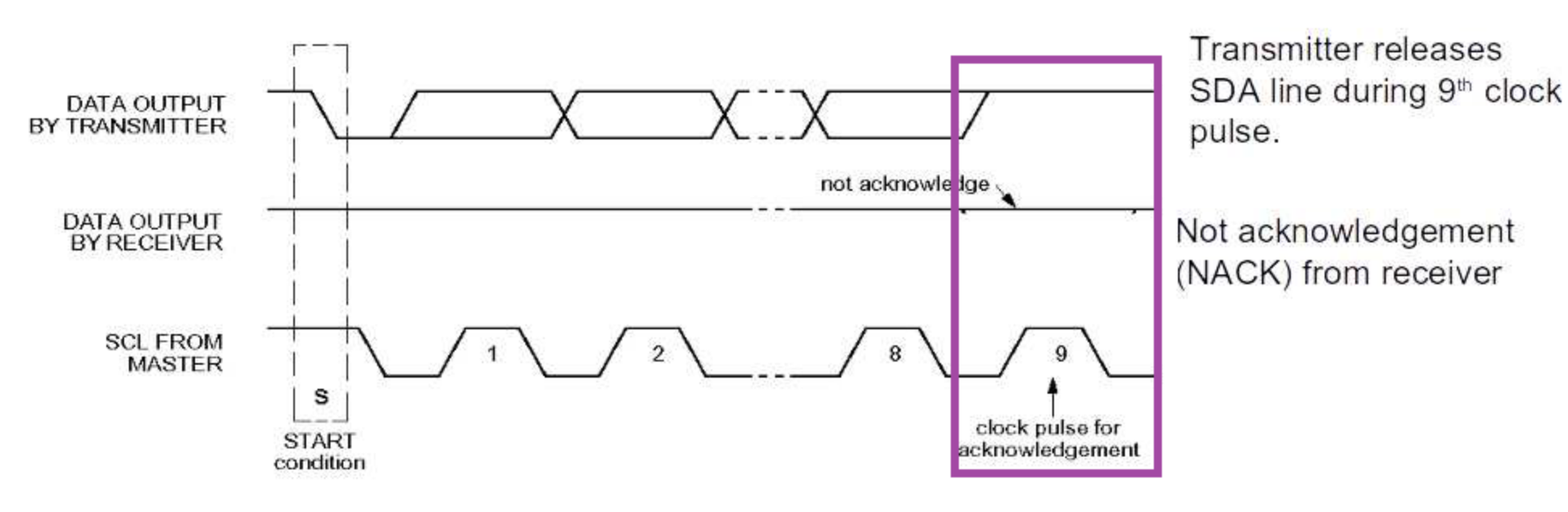
The following scenarios ***do not*** generate acknowledgements.

- From Slave to Master Receiver (as the Master receiver generates the ACK)
- From Master Transmitter to Slave (as the Slave generates the ACK).

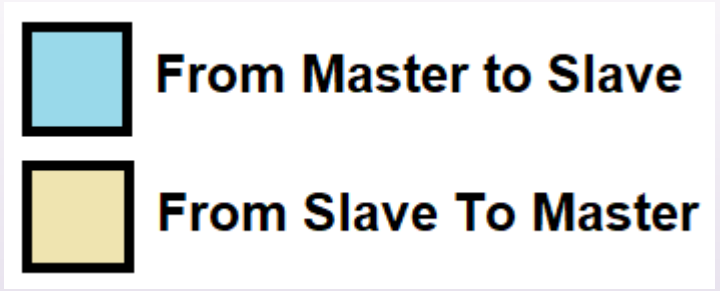
I2C Negative Acknowledge

Receiver leaves data line high for one clock pulse after reception of a byte

- Generated by Slave to Master Transmitter when address or data byte not correctly received.
- Generated by Master Receiver to Slave if the last data byte was correctly received.



Data Formats



Master Writing To Slave

Master Reading From Slave.

Master is receiver of data. Slave is transmitter of data

Arduino Serial

- Used for communication between the Arduino and a computer or other devices
- Serial port (also known as a UART)
- Uses digital pins D0 (RX) and D1 (TX)
- Uses the USB port when communicating with a computer
- Used for debugging purposes
- Plenty of information and examples at
  - https://www.arduino.cc/reference/en/language/functions/communication/serial/

- Example follows…….

Arduino Serial

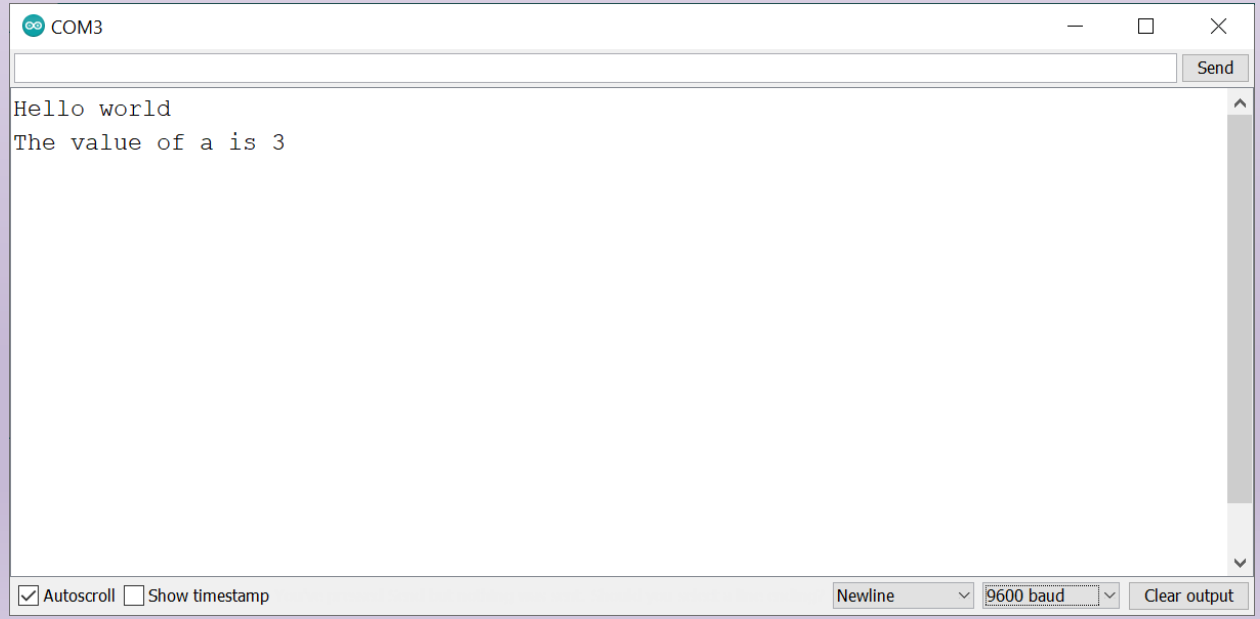
```
void setup(){
    int a = 3;

    Serial.begin(9600);

    Serial.println("Hello world");
    Serial.print("The value of a is ");
    Serial.println(a);
}
```

Setting up serial output with Baud Rate 9600 (other common value is 115200).

Free text with carriage return

Free text without carriage return

Writing the value of 'a' to the serial port (with carriage return)



Use the “Serial Monitor” functionality within the Arduino IDE to view the output. Ensure the Baud Rate of the Serial Monitor matches that specified in `Serial.begin()`.

Accelerometer MPU6050 GY-521

Features

- Accelerometer
- Gyroscope
- Thermometer
- Digital Motion Processor (DMP)
- Secondary I2C Interface.

Applications

- Image stabilisation
- Security, i.e. “no-touch”
- Gesture recognition and command
- Wearable sensors for health, fitness and fall detection

Datasheets

Lots of ideas for different applications from datasheet at

https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

Register map at

https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

For detailed documentation and code examples with DMP programming, please visit

https://invensense.tdk.com/developers/

You may need to register (free) and look into the Software Downloads section for info on the “DMP layer”.

Wire Library

- Provides native Arduino support for a subset of the full I2C known as TWI
- Master, Slave and events are supported and implemented
- Single Master I2C implementations are supported
- Implementation is blocking with multiple timing limitations
- ATmega328P has internal pull-up resistors on the SDA and SCL pins and the Wire library enables them by default, i.e. no need for external pull-up resistors
- The available hardware allows much more. The Wire library does not fully utilise the available I2C/TWI hardware behind the SDA and SCL pins
- Reference at https://www.arduino.cc/en/Reference/Wire

Sample Code

Initialization code for reading the accelerometer.

```
#include <Wire.h>

#define MPU_6050        // TODO: Add I2C address of MPU6050
#define PWR_MGMT_1      // TODO: Add address of PWR_MGMT_1 register
#define ACCEL_ZOUT_HI   // TODO: Add ACCEL_ZOUT (MSB) register

void setup()
{
    // Set-up Serial port to run at 115,200 Baud
    Serial.begin(115200);

    // Set-up the I2C for Accelerometer / Gyro
    Wire.begin();

    //Put the MPU6050 IC into the correct operating mode
    Wire.beginTransmission(MPU_6050);
    Wire.write(PWR_MGMT_1);              // Power Management 1 Register
    Wire.write(0);                       // set to zero (wakes up the MPU-6050)
    Wire.endTransmission(true);
}
```

Sample Code

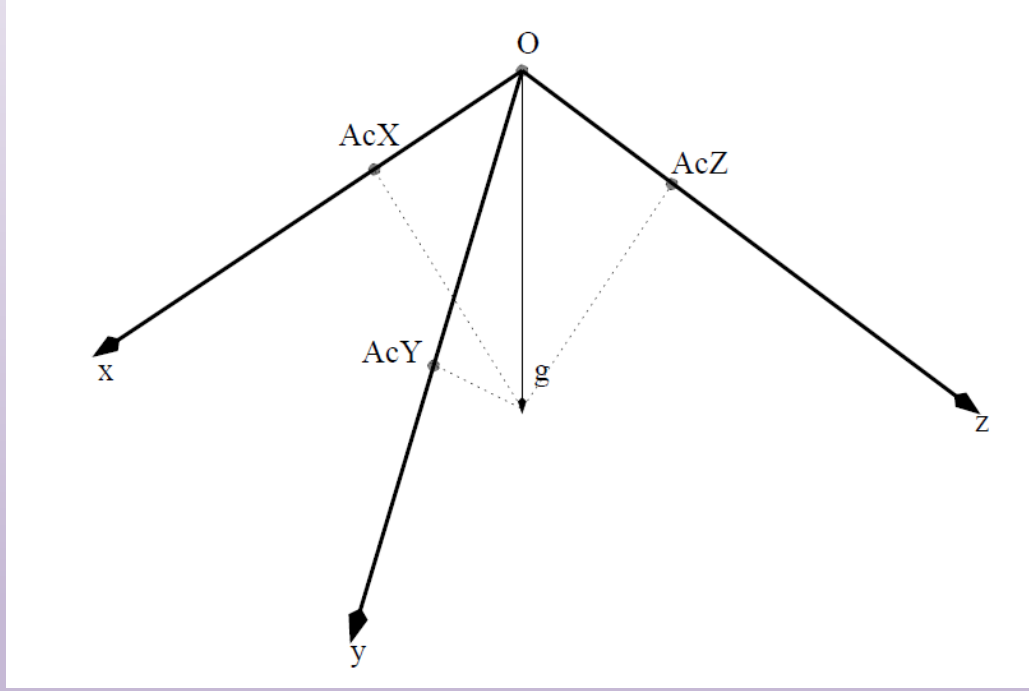
Loop code for reading the accelerometer.

```
void loop()
{
    int   AccZ;

    Wire.beginTransmission(MPU_6050);
    Wire.write(ACCEL_ZOUT_HI);
    Wire.endTransmission(false);

    // Attempt to read 2 consecutive bytes from the MPU6050.
    // From above, these will be the MSB of the Z-component of
    // acceleration followed by its LSB.
    Wire.requestFrom(MPU_6050, 2 ,true);  // request 2 registers
    if (Wire.available() >= 2)
    {
        // Two 8-bit reads to give 16-bit result
        AccZ = Wire.read() << 8;          // Read and shift MSB
        AccZ |= Wire.read();              // Append the LSB

        Serial.print("AccZ = ");
        Serial.println(AccZ);
    }

    delay(1000);      // 1 second delay between samples
}
```

Can you extend this to efficiently read x-, y- and z-accelerometer values?

Accelerometer Orientation

Use the values reported from the accelerometer to discover the orientation.



- Hint: What is the orientation when the AcZ value is ***positive*** and largest of the three components AcX, AcY and AcZ?

- Hint: What is the orientation when the AcZ value is ***negative*** and largest of the three components AcX, AcY and AcZ?

Accelerometer Orientation Coursework

This comprises Formative Assessment Part 3

- Use an Arduino, MPU6050, seven segment display, 74HC595 shift-register, push-button, resistors, wires and a breadboard
- For each of the three axes of the accelerometer determine the value of gravity for that axis
- Display the orientation on the seven segment display using the letters F, L, U, l, r, b for “Flat”, “Landscape”, “Upside-down”, “left portrait”, “right-portrait” and “base-up” correspondingly.
- Complete by the end of your lab session in Week 4
- Be ready to demo during your lab session in Week 5
- Work individually

Summary

- Overview of the I2C protocol
- Introduced the Serial library
  - For debugging
- Introduced the Wire library
  - For I2C transactions
- Sample code to drive the accelerometer (MPU6050)
- How the accelerometer outputs can be used to determine orientation.