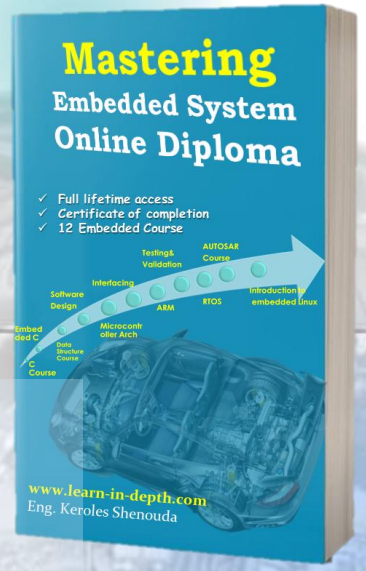
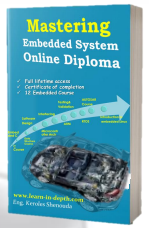
Mastering Embedded System

Online Diploma

- ✓ Full lifetime access
- ✓ Access on Android mobile and PC (Windows)
- ✓ Certificate of completion
- ✓ 12 Embedded Course



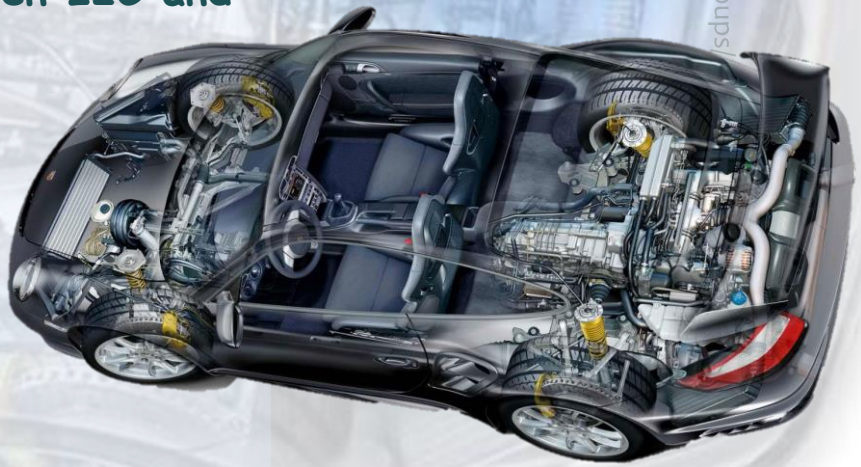


#LEARN_IN DEPTH

#Be_professional_in embedded_system

Unit 8 (MCU Interfacing) . lesson 6 **I2C Protocol**

- I2C Characteristics
- I2C Evolution
- I2C Speed Categories
- I2C Pins And open-drain Usage
- I2C Bus troubleshooting
- I^2C Bus Characteristics
- How I2C work ?
- I2C data sampling
- I2C: ACK, NACK, No ACK
- No ACK conditions
- I2C Arbitration
- I2C Clock Stretching
- Multibyte burst write/read
- I2C clock synchronization
- HOW TO SELECT Between I2C and SPI
- UART Vs I2C Vs SPI



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

(I2C) Inter - Integrated Circuit

it's time to wake up ☺

LEARN-IN-DEPTH
Be professional in
embedded system

FOLLOW US
Press here
2

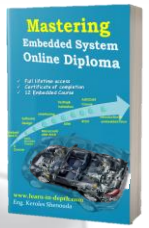
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

Eng. Keroles Shenouda

https://www.facebook.com/groups/embedded.system.KS/

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

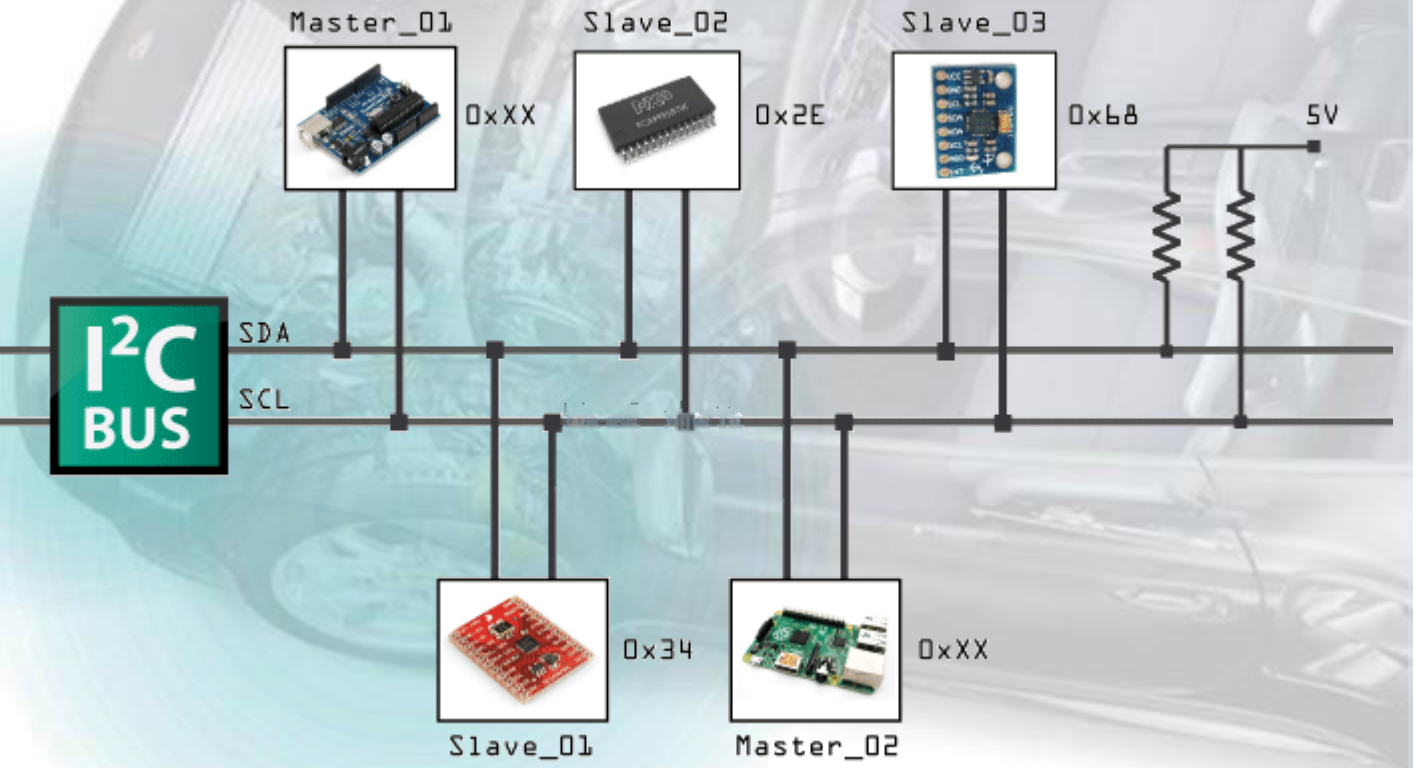


What is I²C

- The name stands for "**Inter - Integrated Circuit**"
- A Small Area Network connecting ICs and other electronic systems
- The **IIC** is also referred to as I2C (**I2C**) or **I square C** in many technical literatures.
- The **I2C bus** was originally started by **Philips**, Now It is a widely used standard adapted by many chip making companies
- **I2C** is **ideal** for communication between **low-speed devices** for a reliable communication over **a short distance.**
- Today, a variety of devices are available with I2C Interfaces
  - Microcontroller, EEPROM, Real-Timer, interface chips, LCD driver, A/D converter

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

I2C Characteristics

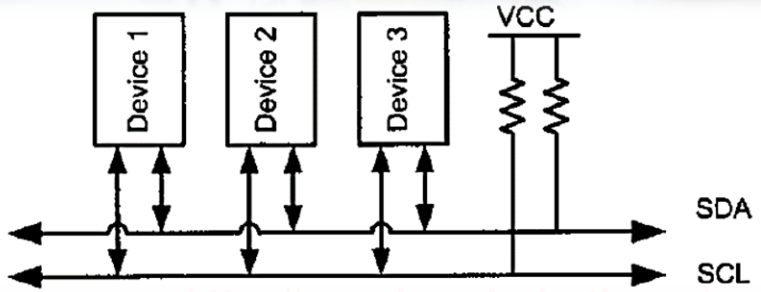


- **Multi Master Multi Slave**
- **Synchronous** Serial Signal (One wire used for the clock ☺)
- **Half duplex** (One wire use for the data)
- Also called a 2-wire bus
  - It has only clock and data, with pull -up resistors
- This **reduction** of **communication pins reduces** the **package size** and **power consumption**
- **I2C max speed is 4MHZ in ultra speed plus mode which is not exist on all MCUs**
- **Lines pulled high via resistors**
- I2C protocol is multi master capable , where SPI has no guidelines to achieve this, but depends on MCU Designer. STM SPI peripherals can be used in multi master configuration but arbitration should be handled by SW code

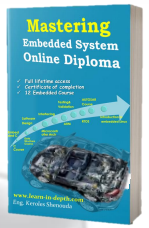
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

I2C Line Electrical Characteristics

- **I2C** devices use **only 2 bidirectional** open-drain pins for data communication. To implement I2C, only **a 4.7 kilohm pull-up resistor** for each of bus lines is needed.
- This means that if **one or more devices** pull the line to low (zero) level, the line state is zero **and the level of line will be 1 only if none of devices pull the line to low level.**
- **At any given moment the I2C bus is:**
  - (Idle) state
  - in Master transmit mode or
  - in Master receive mode.



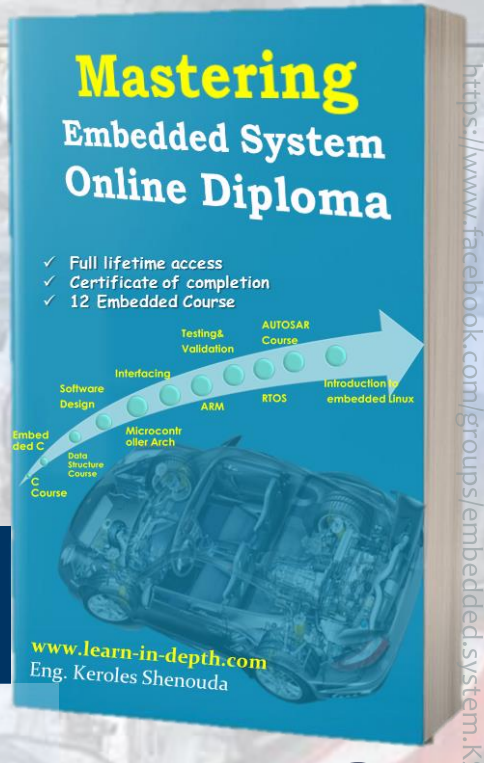
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

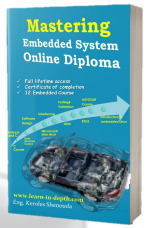
#Be_professional_in_embedded_system

I2C Speed Categories



LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



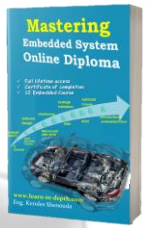
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

I2C Evolution

Speed Category	Clock Frequency	Bus Direction
Standard-Mode (SM)	Up to 100 KHz	Bidirectional
Fast-Mode (FM)	Up to 400 KHz	Bidirectional
Fast-Mode Plus (FM+)	Up to 1.0 MHz	Bidirectional
High-Speed Mode (HS-Mode)	Up to 3.4 MHz	Bidirectional
Ultra-Fast Mode (UFM)	Up to 5.0 MHz	Unidirectional

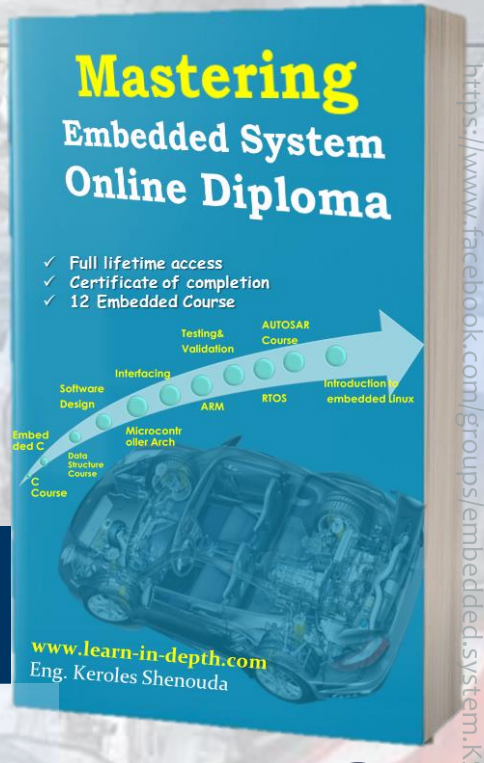
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

I2C Pins

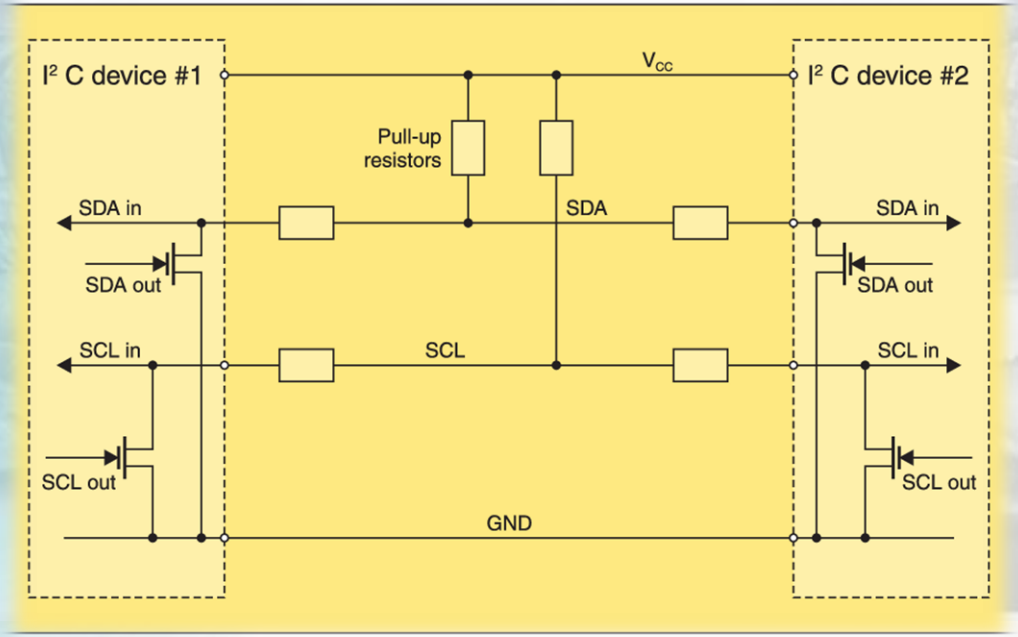


LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

I2C Pins

- The pins of the master and peripheral devices connected to the SDA and SCL lines should be internally configured as *open-drain*, also known as *open collector*



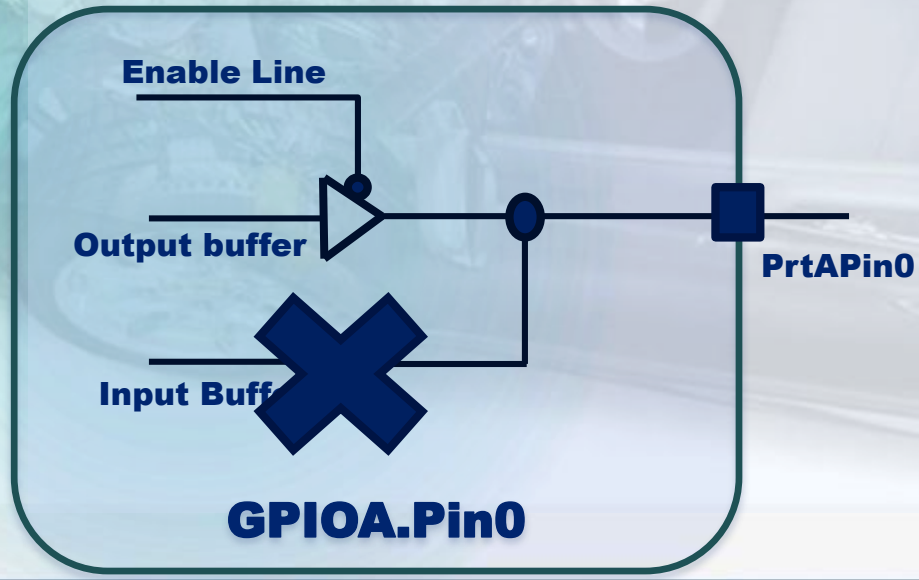
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

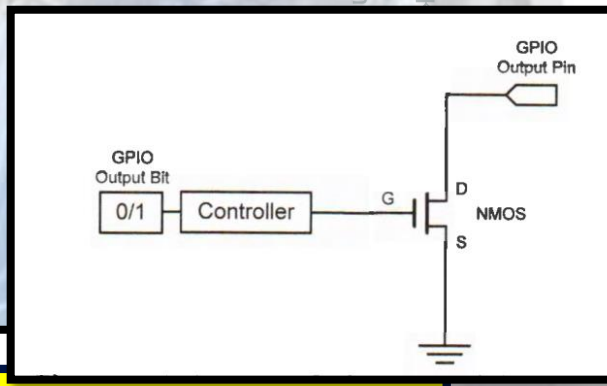
GPIO Output Modes: Open-Drain

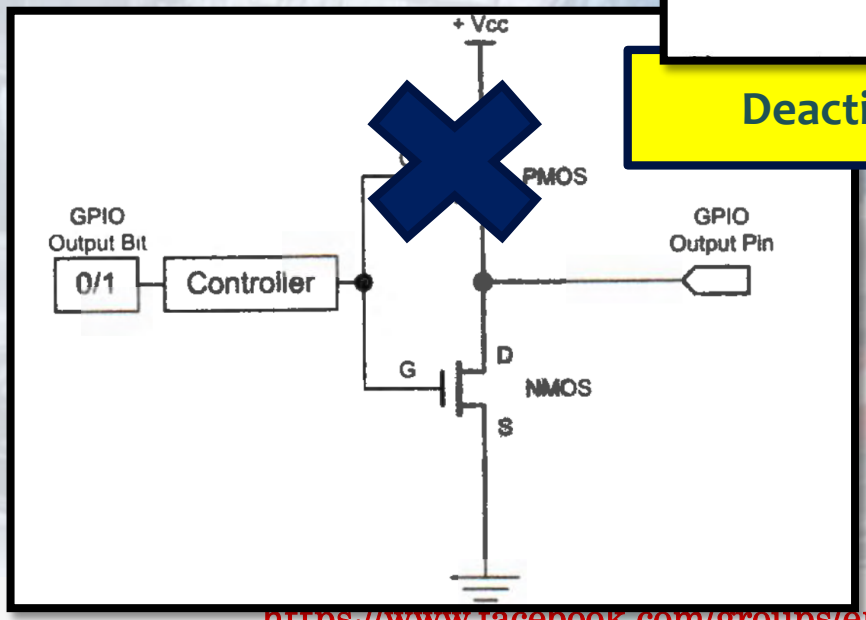
- GPIO pin in open-drain

  (also called collector) mode **can only absorb current**

- An open-drain output consists of a pair of the same type of CMOS or transistors



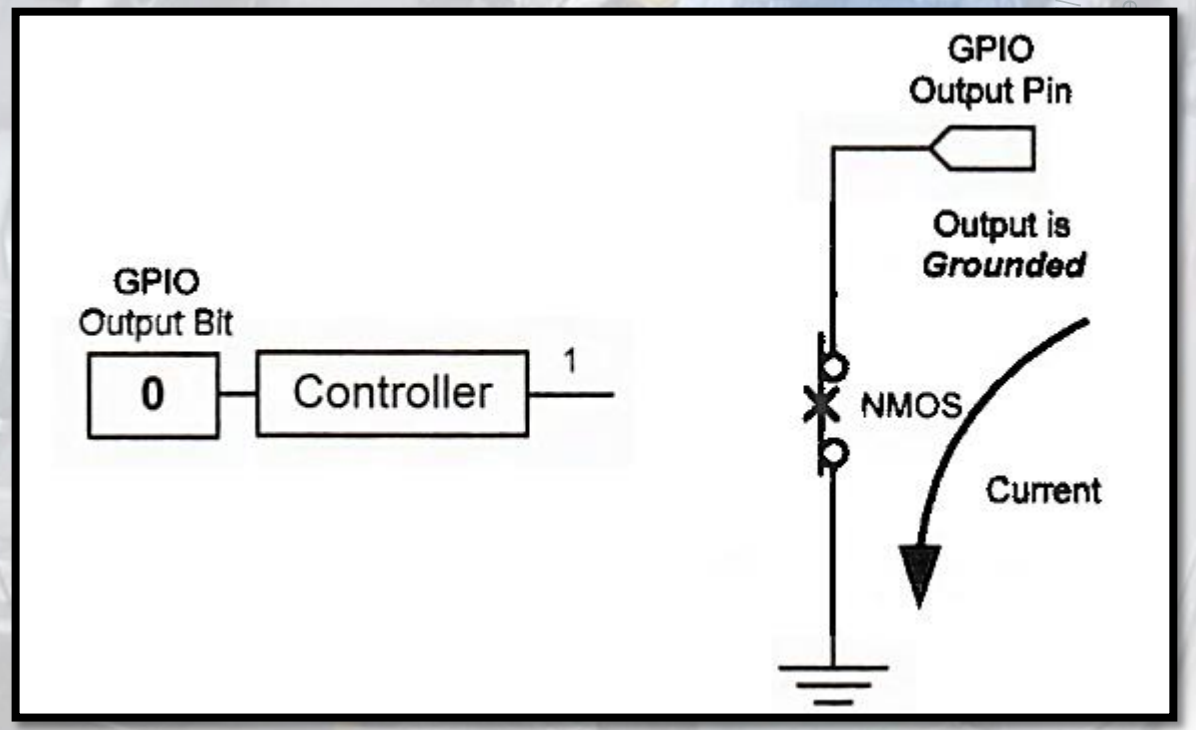




Deactivated

GPIO Output Modes: Open-Drain

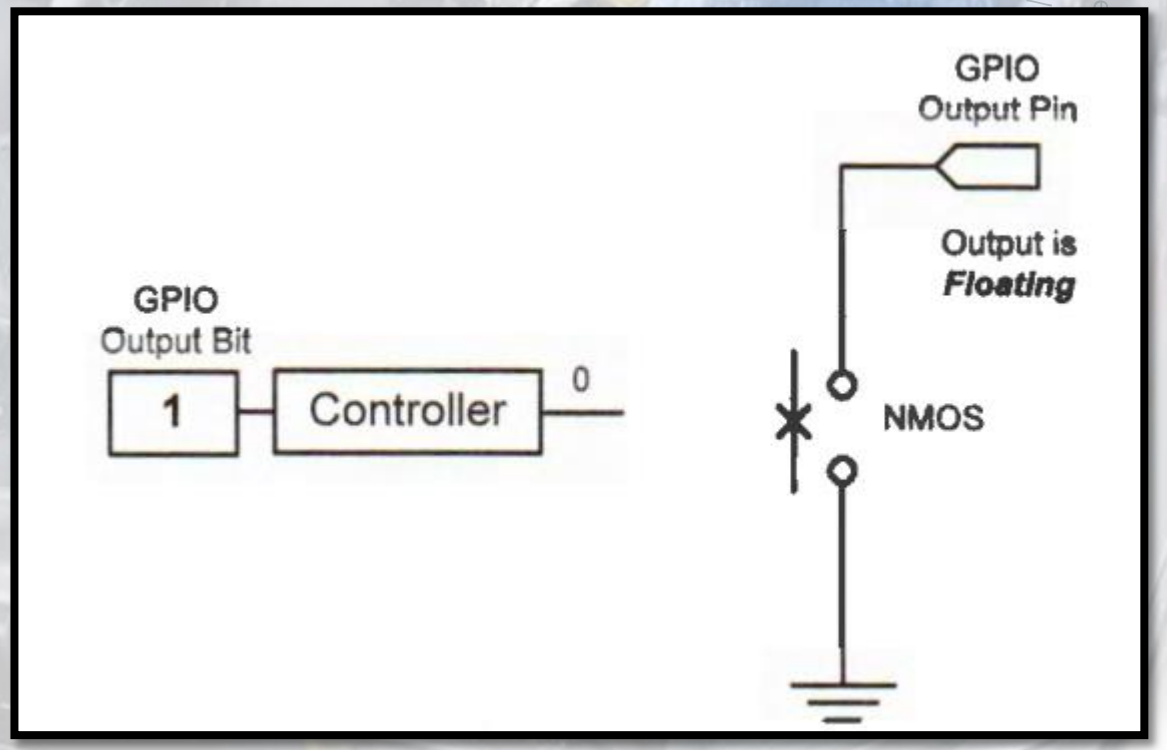
- When software outputs a logic 0,
- the **open-drain circuit** can **sink** an electric current from the external load connected to the GPIO pin.



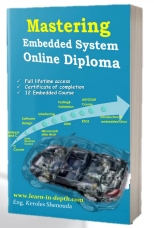
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

GPIO Output Modes: Open-Drain

- when software outputs a logic 1,
- it cannot supply any electric current to the external load because the output pin is **floating**, connected to neither the power supply nor the ground.



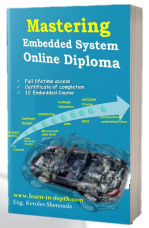
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



GPIO Output Modes: Open-Drain

- **An open-drain output** has only two states:
- **low voltage (logic 0),**
- **and high impedance** (logic 1). **It often has an external pull-up resistor**

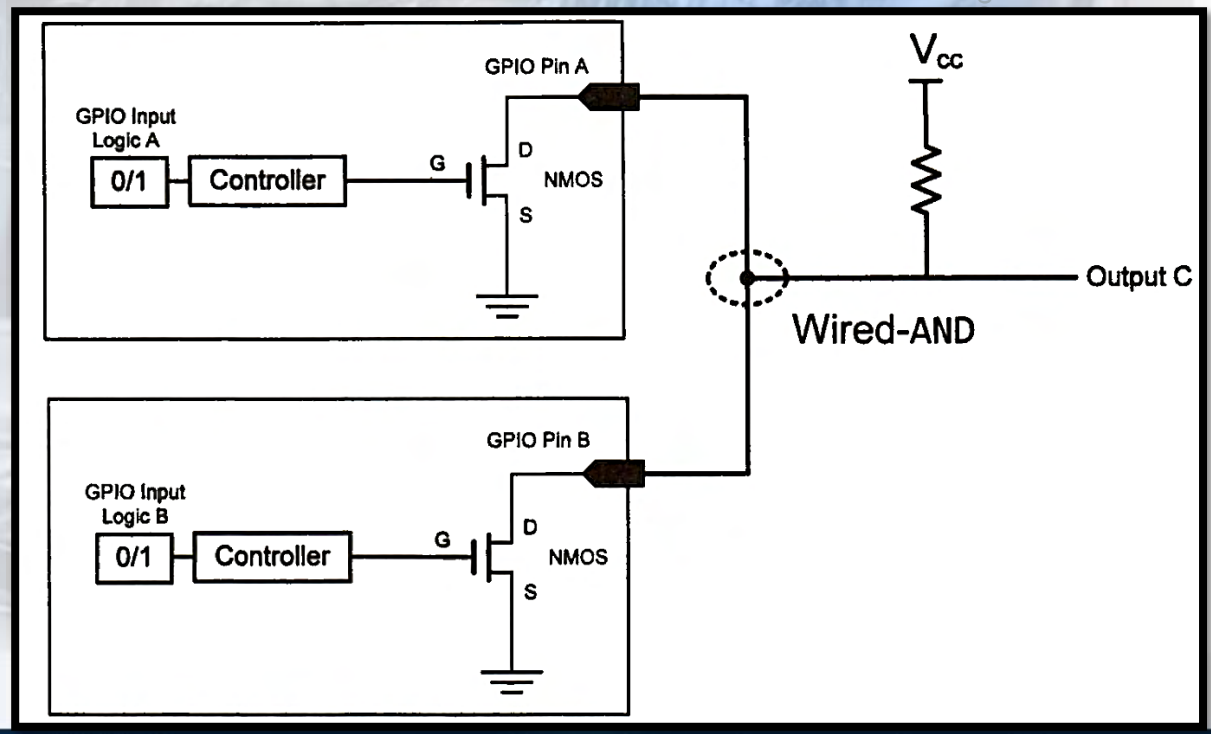
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



What is the important usage of open-drain outputs ?

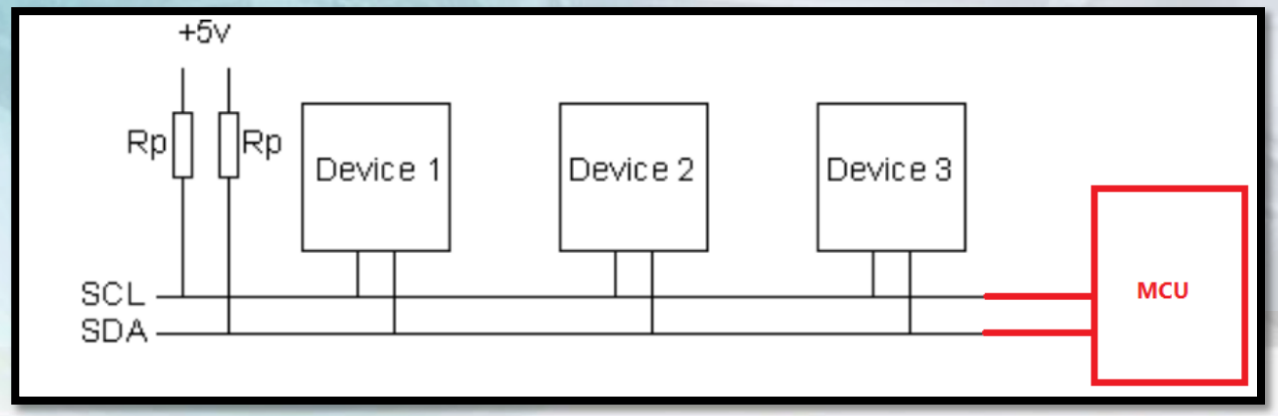
- One important usage of open-drain outputs is to connect directly **several outputs together** and **implement wired logic AND** (active high) or OR (active low) circuit in an easy way.
- If multiple open-drain output pins are connected and **are pulled up via a shared resistor**,
- any output pin can drive the output voltage low.
- The pin voltage is high if and only if all pins output a high voltage level.

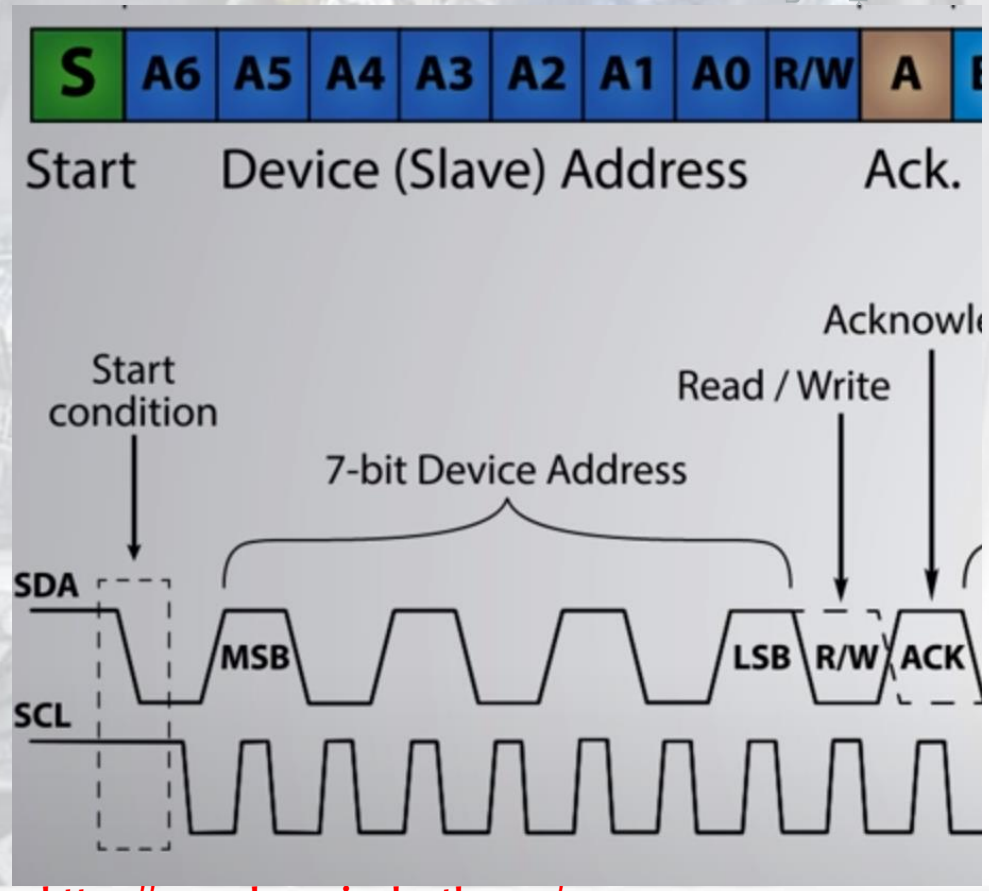
Inputs				Output
Logic A	Logic B	Circuit A	Circuit B	
0	0	Drain	Drain	0
0	1	Drain	Open	0
1	0	Open	Drain	0
1	1	Open	Open	1



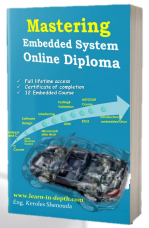
Driving I2C Bus

- *in this case both lines have pull-up resistors*
- The pull-up resistor connects to HIGH (which is the same as the supply voltage or Vcc) at one end and connects to one or more external pins of the **open-drain devices** (via the bus). **Thus, if any one of the open-drain devices is set to sink current, current flow for all of the devices sinks to ground.**





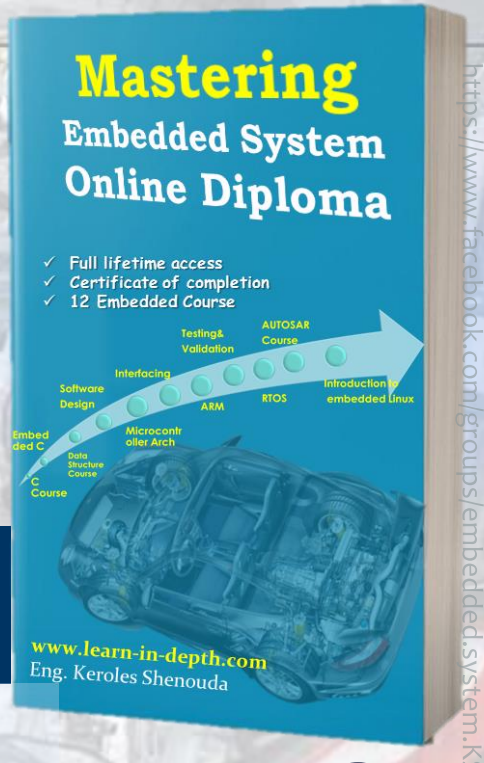
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

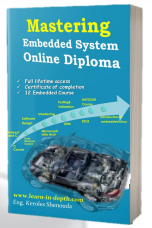
#Be_professional_in_embedded_system

I2C Bus troubleshooting



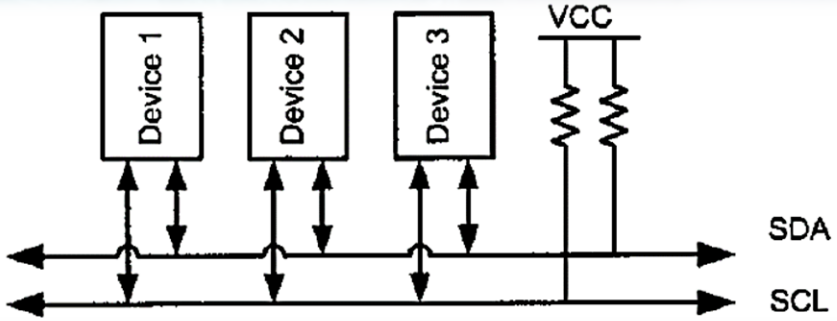
LEARN-IN-DEPTH
Be professional in embedded system

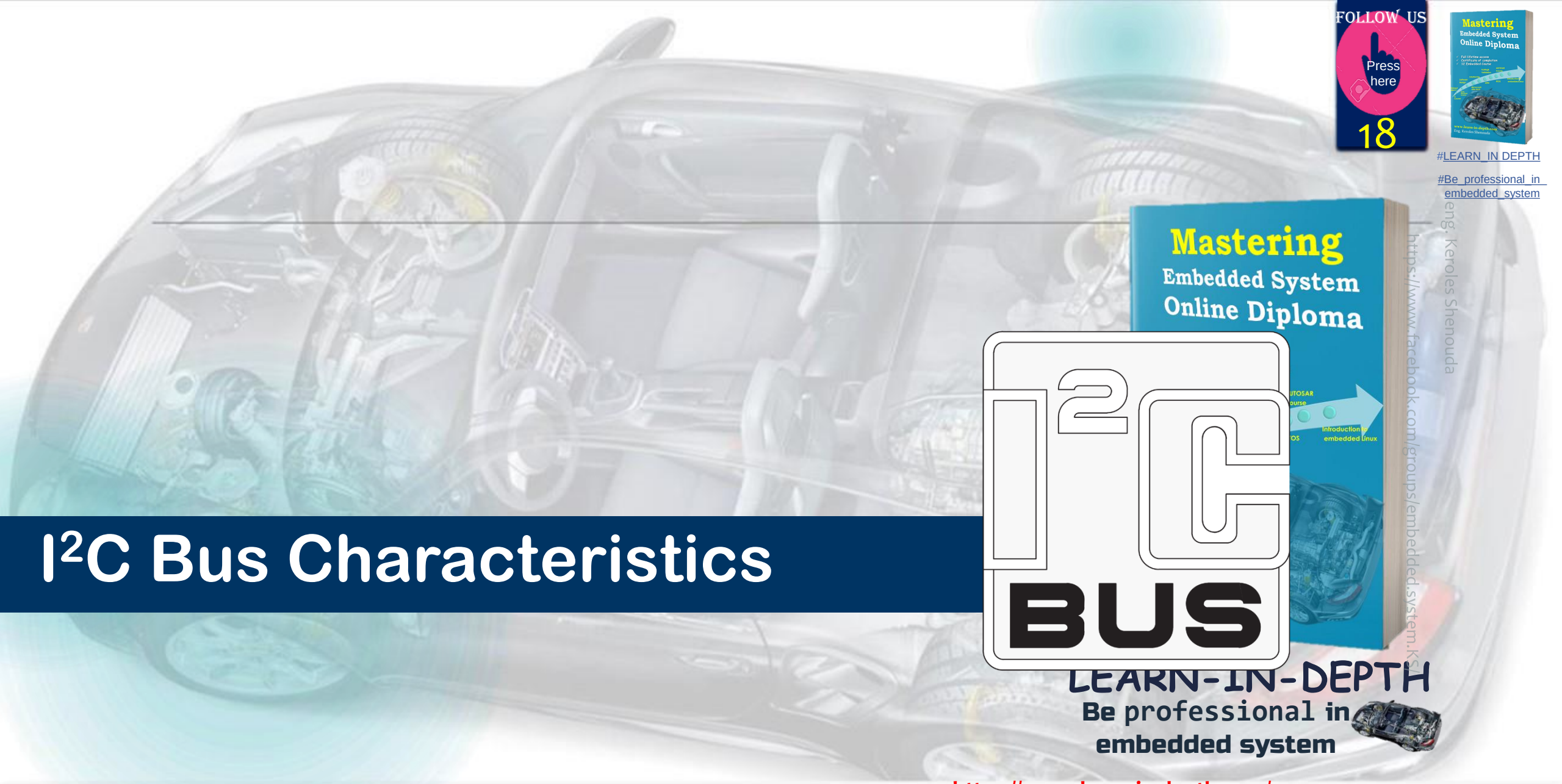
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



I2C Bus Basic troubleshooting

- After I2C initialization, the SDA and SCL lines must be Held at high 3.3V or 1.8 V depending up on IO voltage Levels of your board



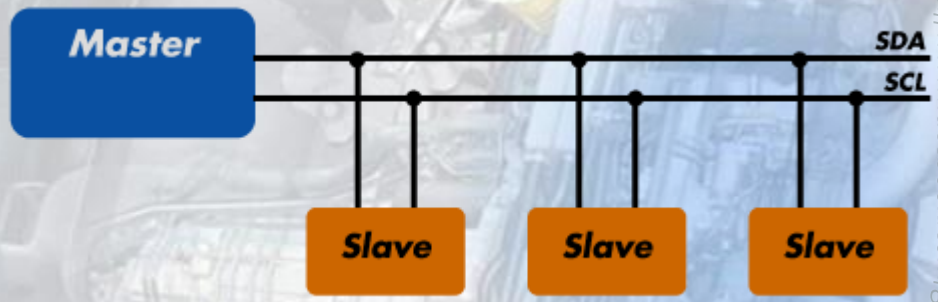


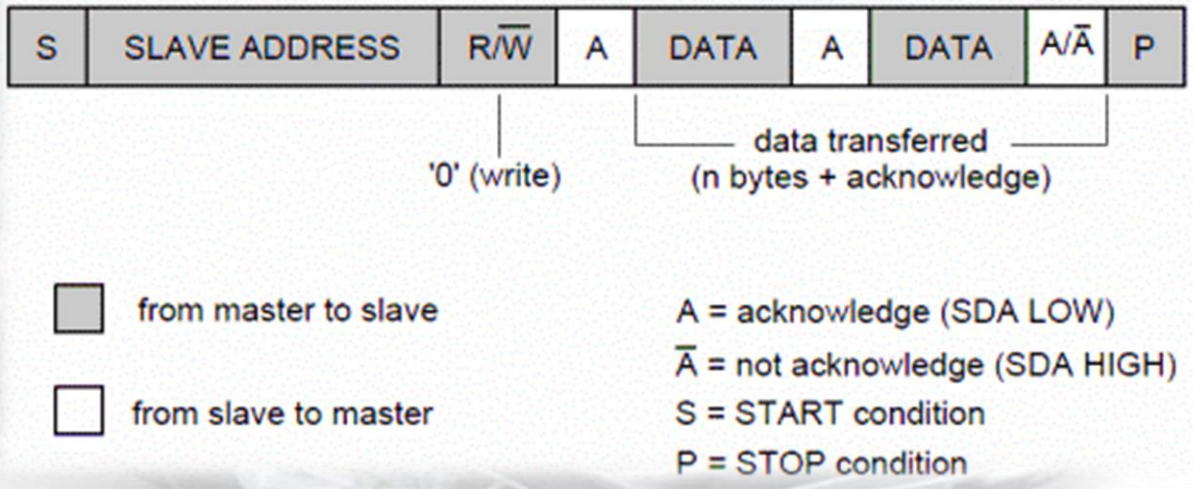
I²C Bus Characteristics

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

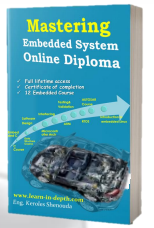
I2C Bus Characteristics

- Two wire serial data & control bus implemented with the serial data (SDA) and clock (SCL) lines
- 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 terminology Concepts:
  - **Master** This is the device that generates clock, starts communication, sends I2C commands and stops communication
  - **Slave** This is the device that listens to the bus and is addressed by the master
  - **Arbitration** A process to determine which of the masters on the bus can use it when more masters need to use the bus



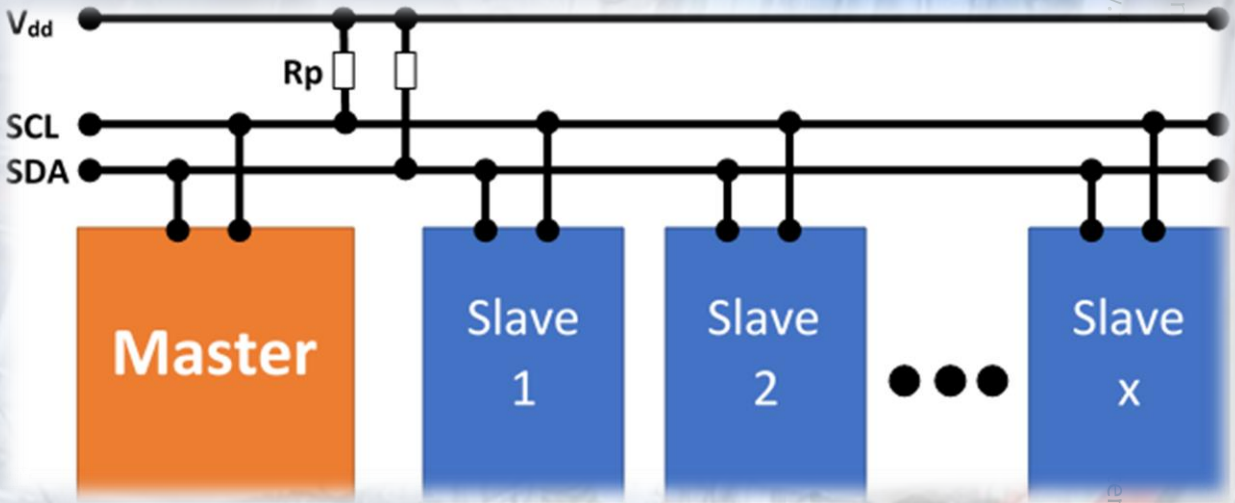


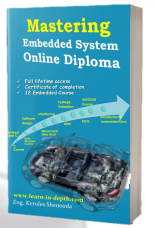
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



I2C Bus Definitions

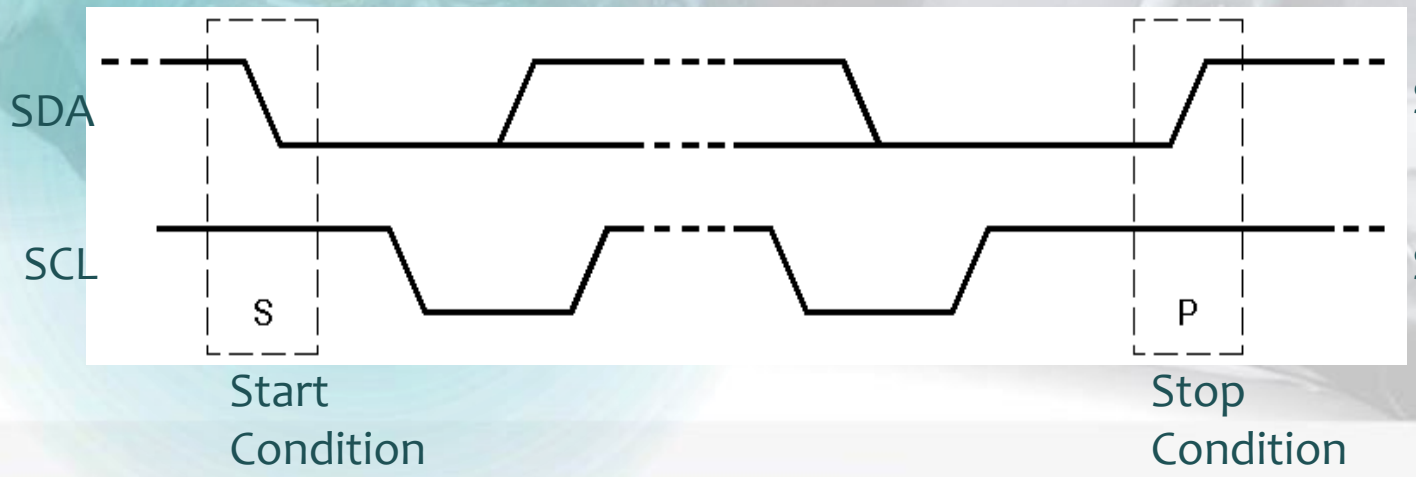
- **Master:**
  - Initiates a transfer by generating start and stop conditions
  - Generates the clock
  - Transmits the slave address
  - Determines data transfer direction
- **Slave:**
  - Responds only when addressed
  - Timing is controlled by the clock line





Start and Stop Conditions

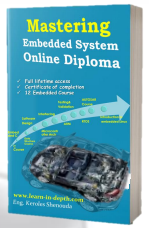
- each transmission is initiated by a START condition and is terminated by a STOP condition.
- the START and STOP conditions are generated by the master.
- START and STOP conditions are generated by keeping the level of the SCL line high and then changing the level of the SDA line
- The START condition is generated by a high-to-low change in the SDA line when SCL is high.
- The STOP condition is generated by a low-to-high (ideal case) change in the SDA line when SCL is high



SDA Start: when SCL:HIGH , Falling edge of SDA occurs

SCL Stop: when SCL: HIGH , Rising edge of SDA occurs

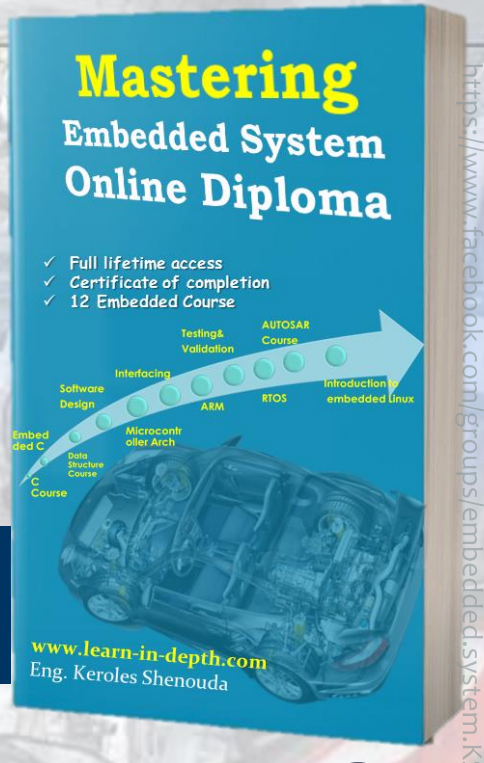
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

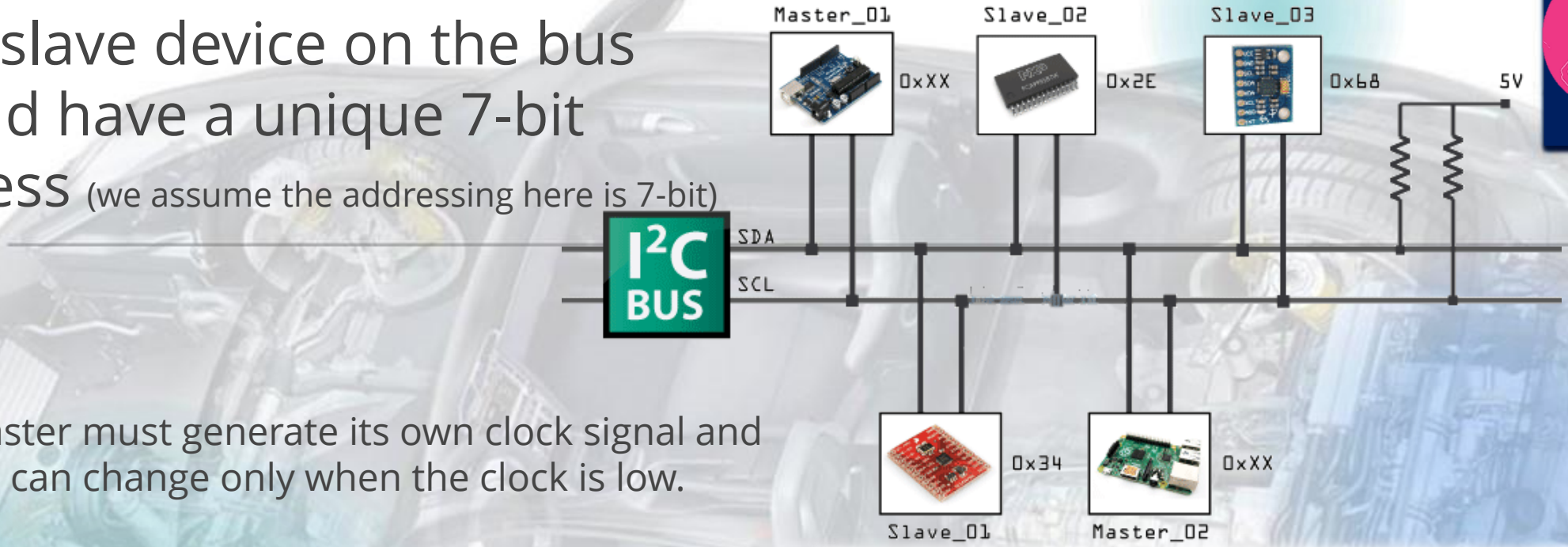
How I2C work ?



LEARN-IN-DEPTH
Be professional in embedded system

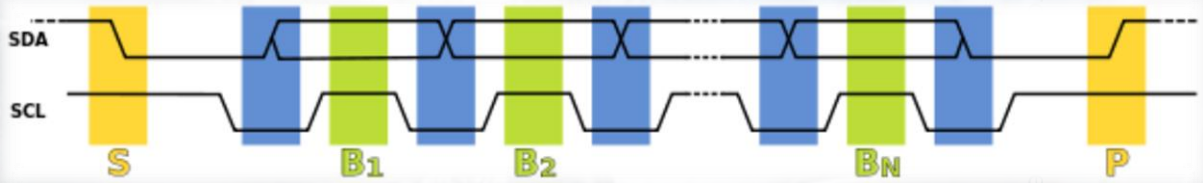
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Each slave device on the bus should have a unique 7-bit address (we assume the addressing here is 7-bit)



Each master must generate its own clock signal and the data can change only when the clock is low.

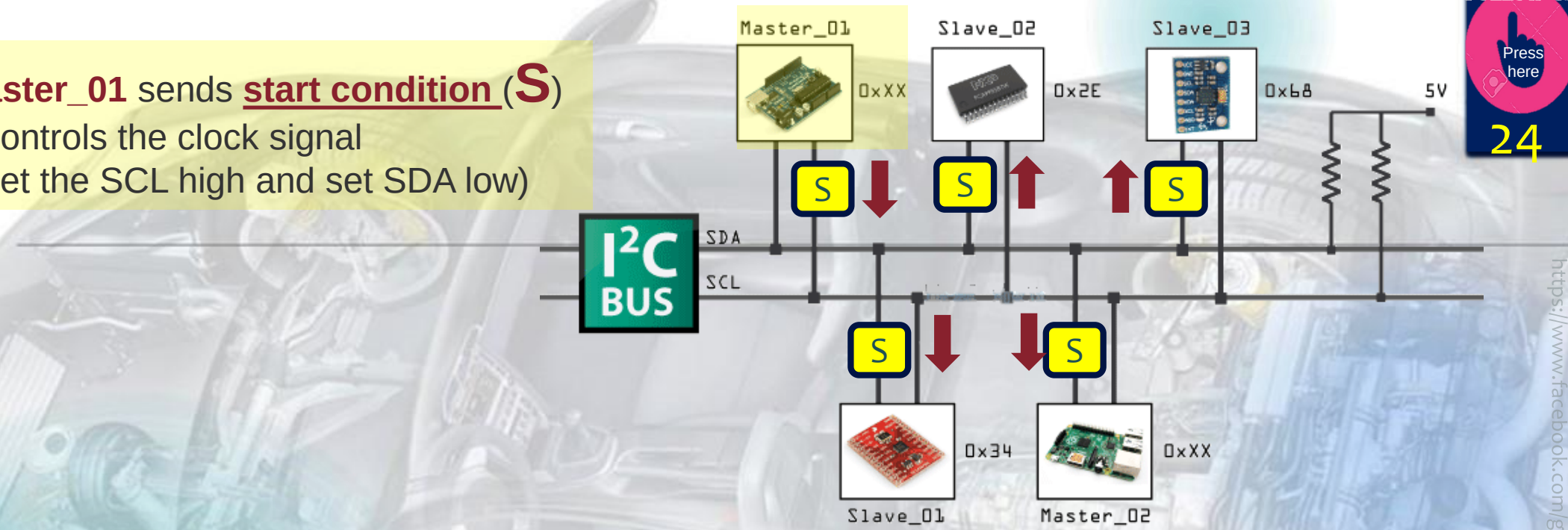
the clock is in the high state until the first master pulls it low.

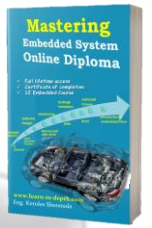


The 7-bit address lets the master address a maximum of 128 slaves on the bus.

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

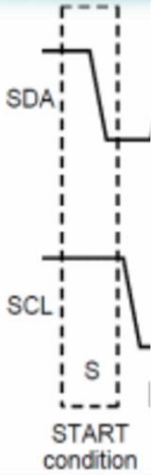
1. Master_01 sends **start condition** (**S**)
and controls the clock signal
(by set the SCL high and set SDA low)





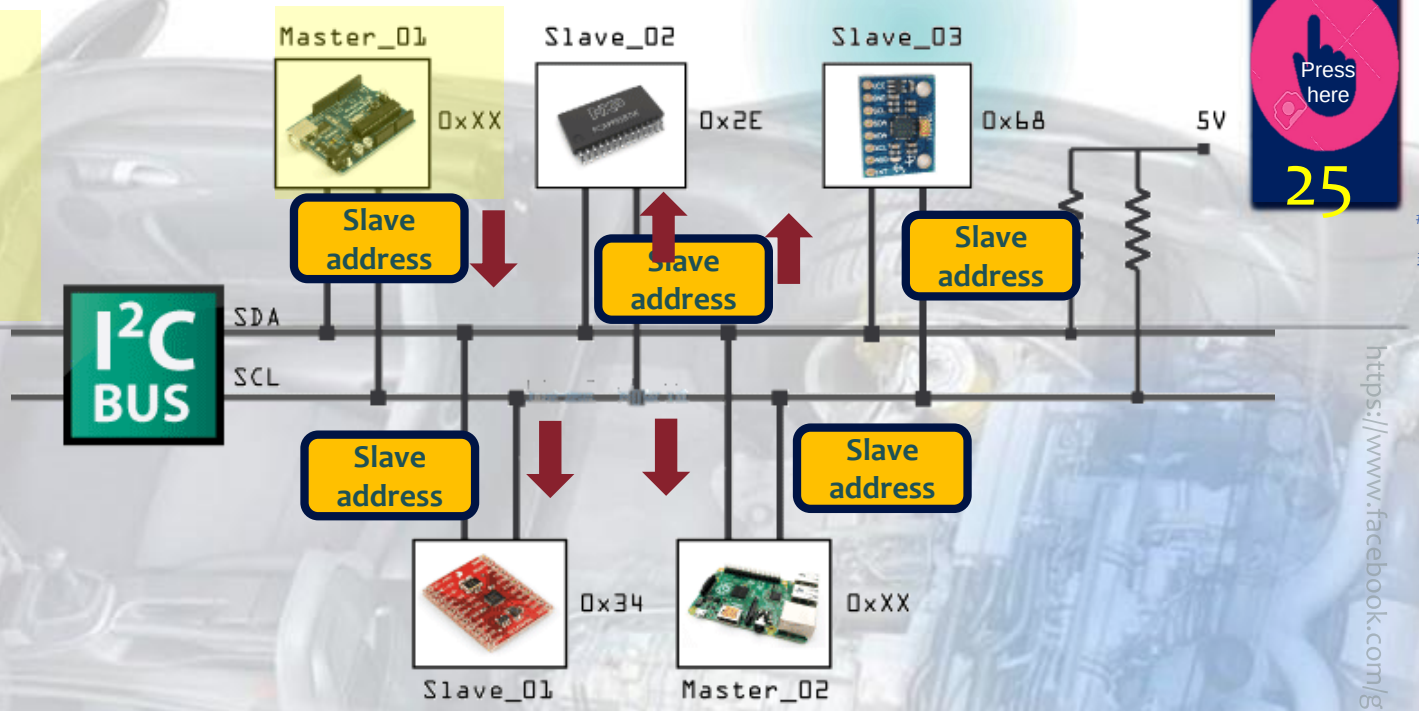
#LEARN_IN DEPTH
#Be_professional_in_embedded_system

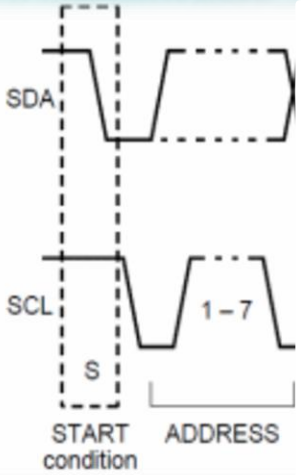
S



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

2. Master_01 send a unique 7 bit slave device address (in this example we assume the I2C address is 7 bit)
For example slave address = 0x34 (slave_01)



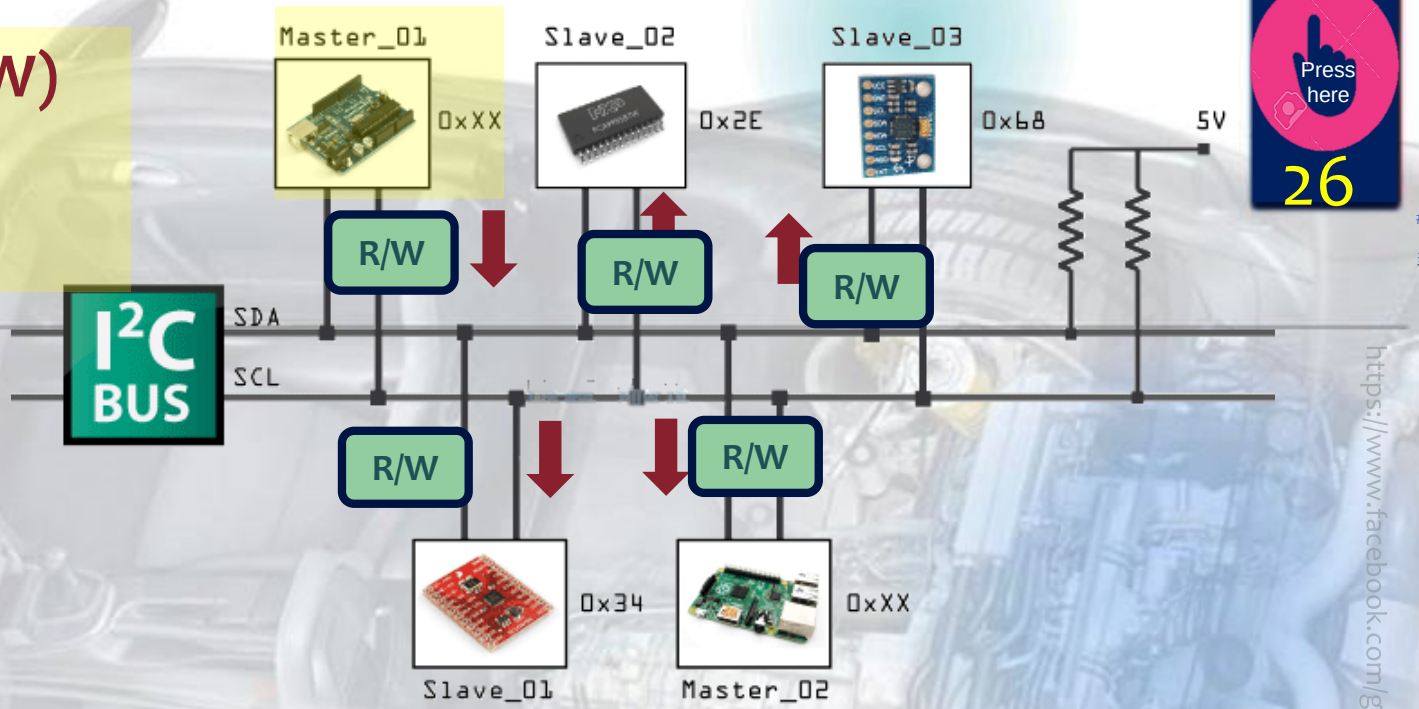


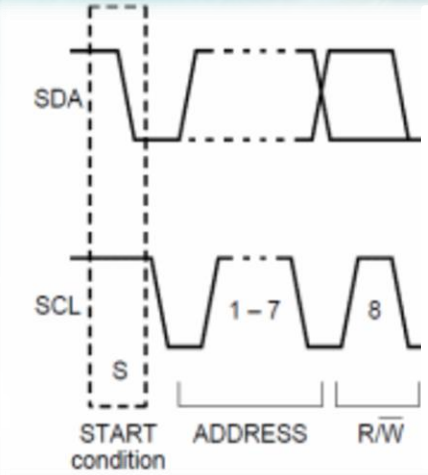
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

3. Master_01 sends read/write bit (R/W)

- **0** means **W** (**slave receive**)
- **1** means **R** (**slave transmit**)

We assume in this example that the master will send the 'W' to the Slave

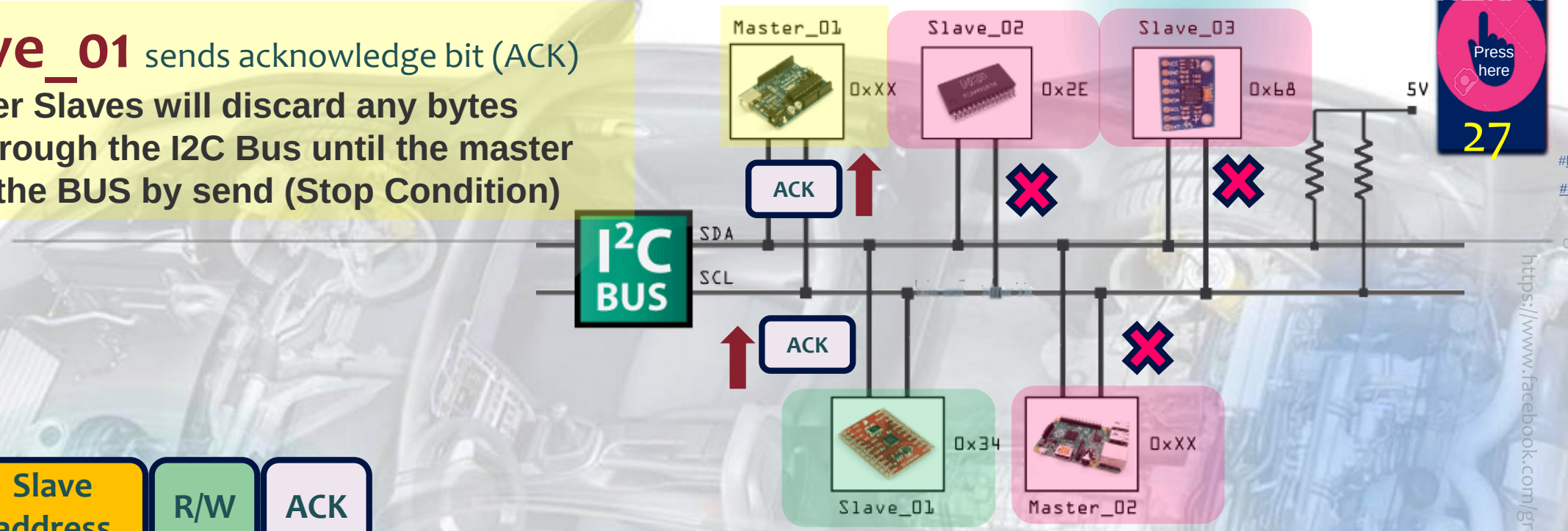


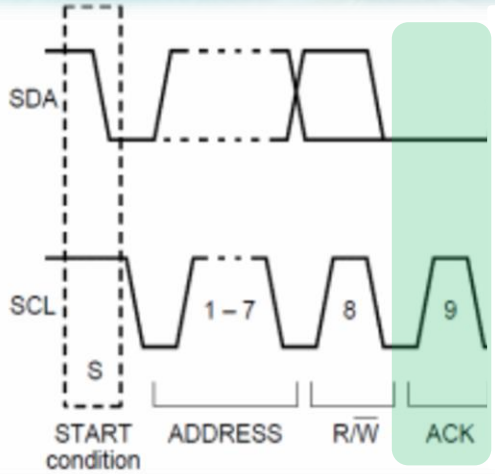


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

4. Slave_01 sends acknowledge bit (ACK)

The other Slaves will discard any bytes come through the I2C Bus until the master release the BUS by send (Stop Condition)



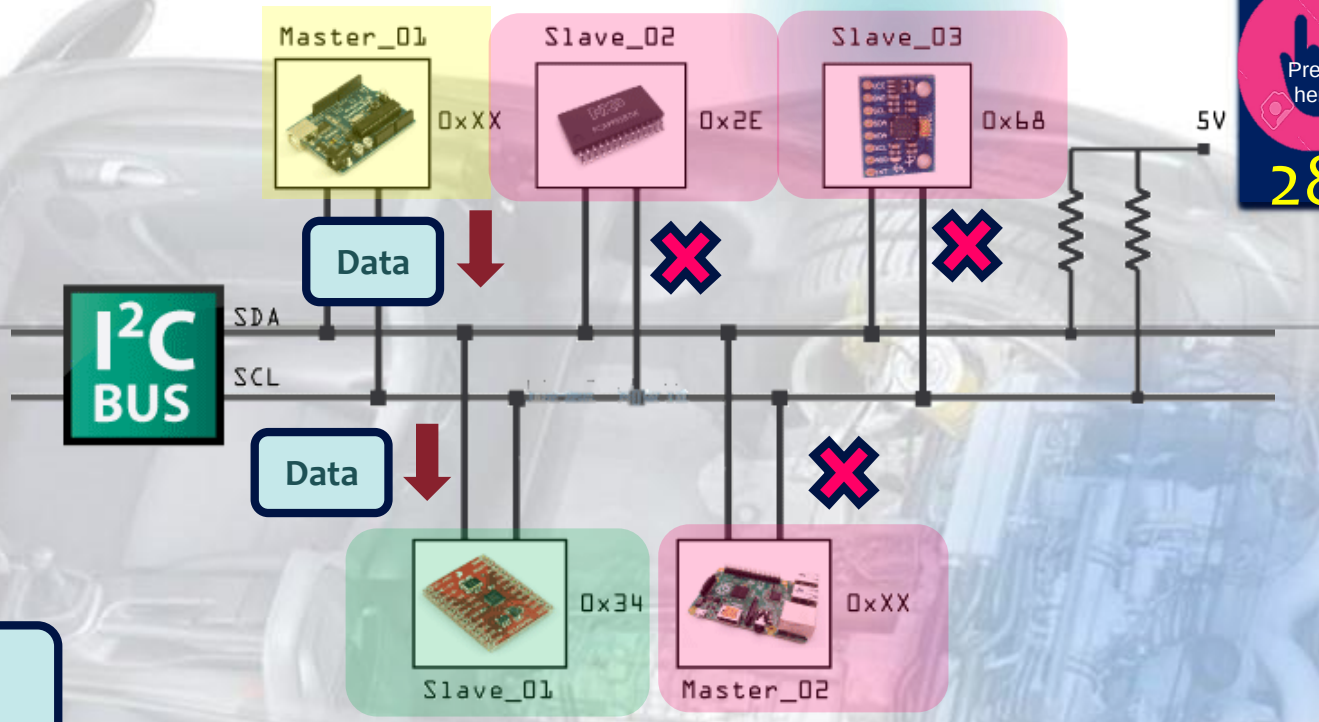


Slave devices that need some time to process received byte or are not ready yet to send the next byte, can pull the clock low to signal to the master that it should wait

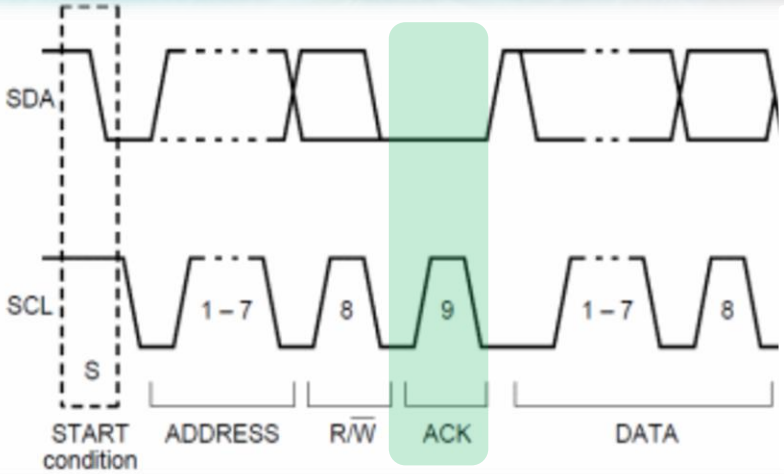
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

5. Transmitter (slave or master) transmits 1 byte of data

We assume in this example that the master will send the DATA to the Slave

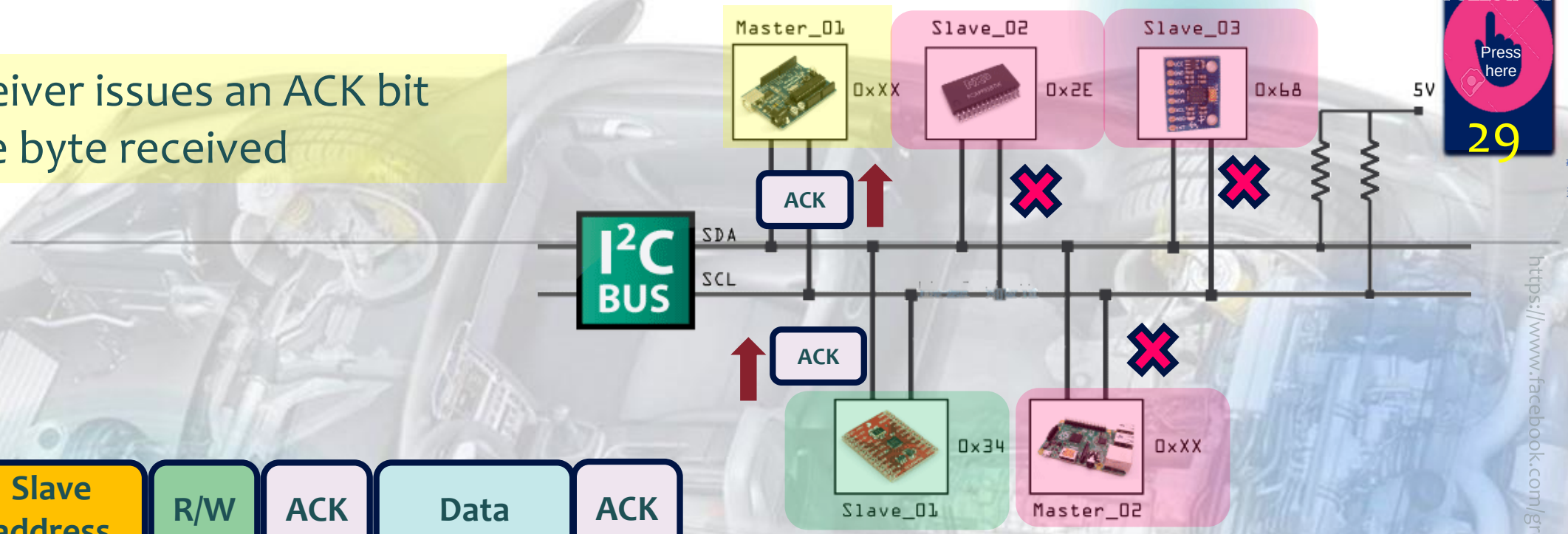


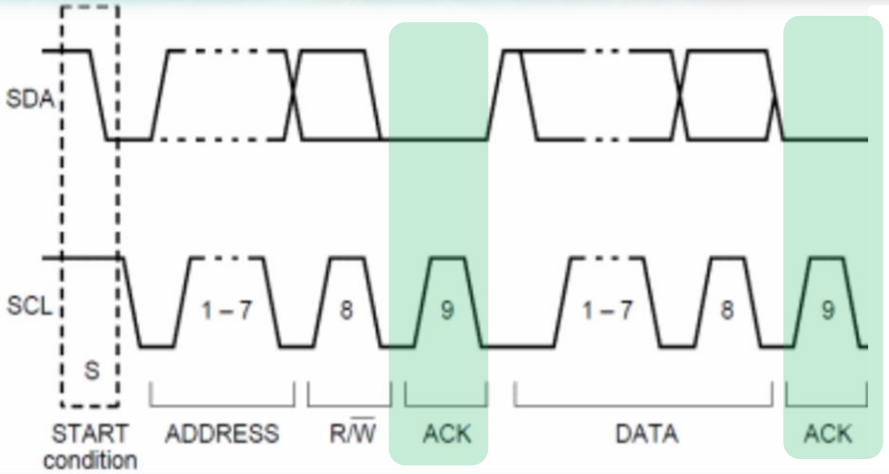
S	Slave address	R/W	ACK	Data



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

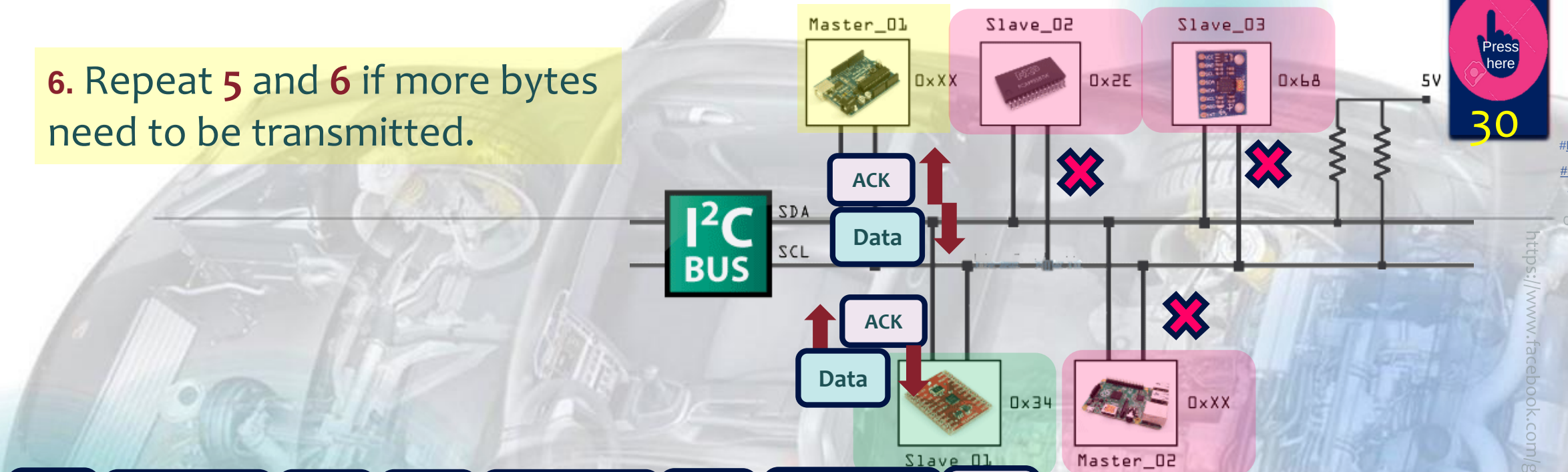
6. Receiver issues an ACK bit for the byte received

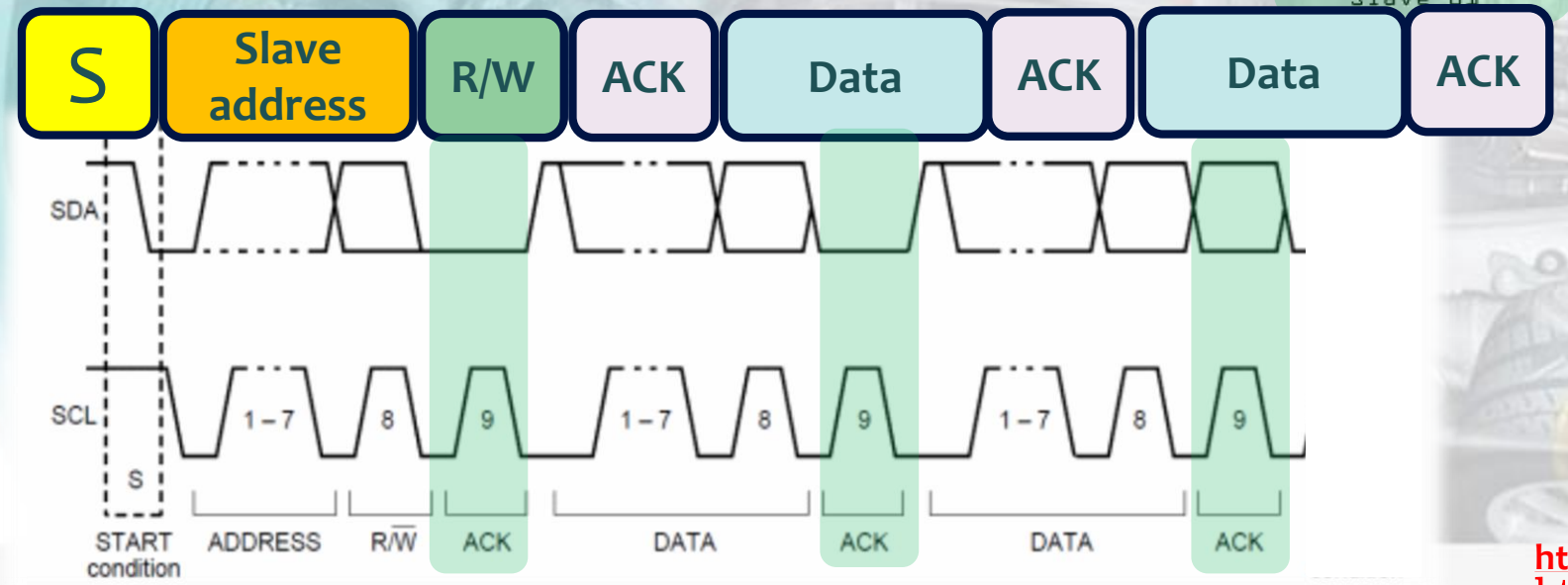




https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

6. Repeat 5 and 6 if more bytes need to be transmitted.

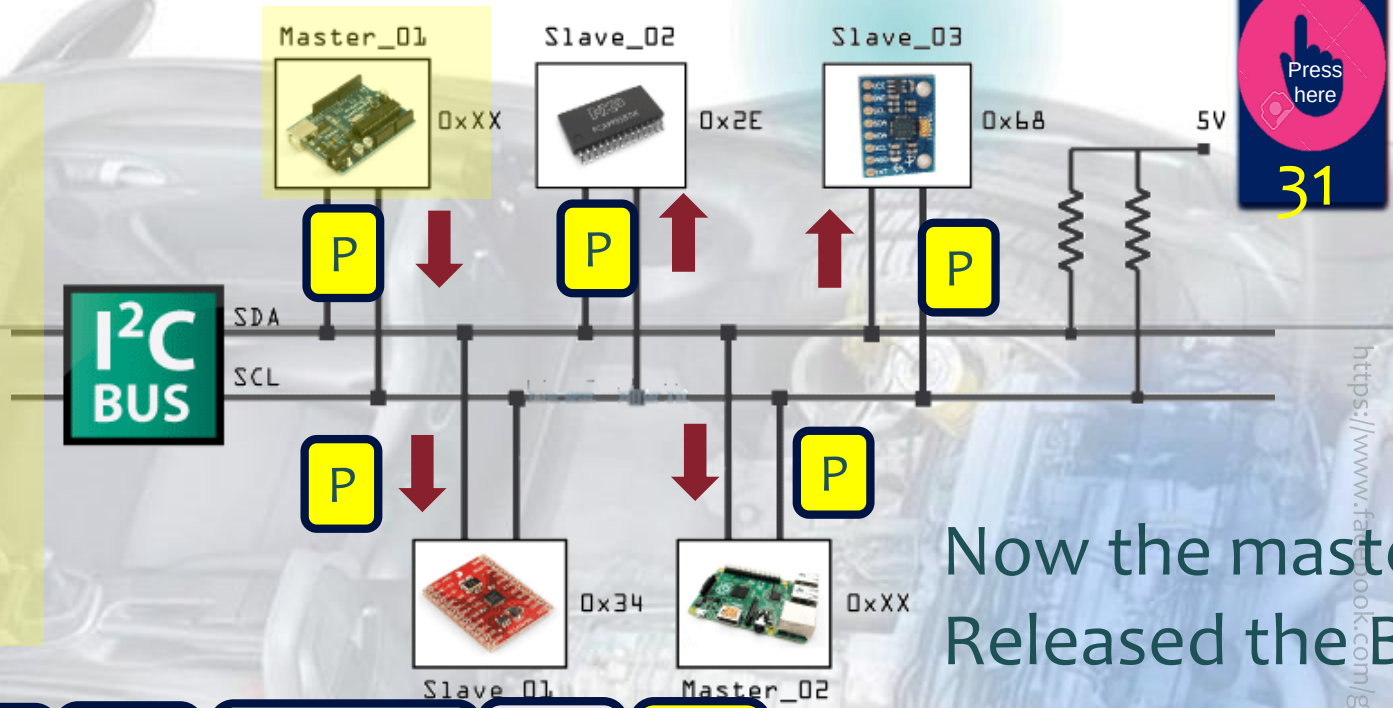




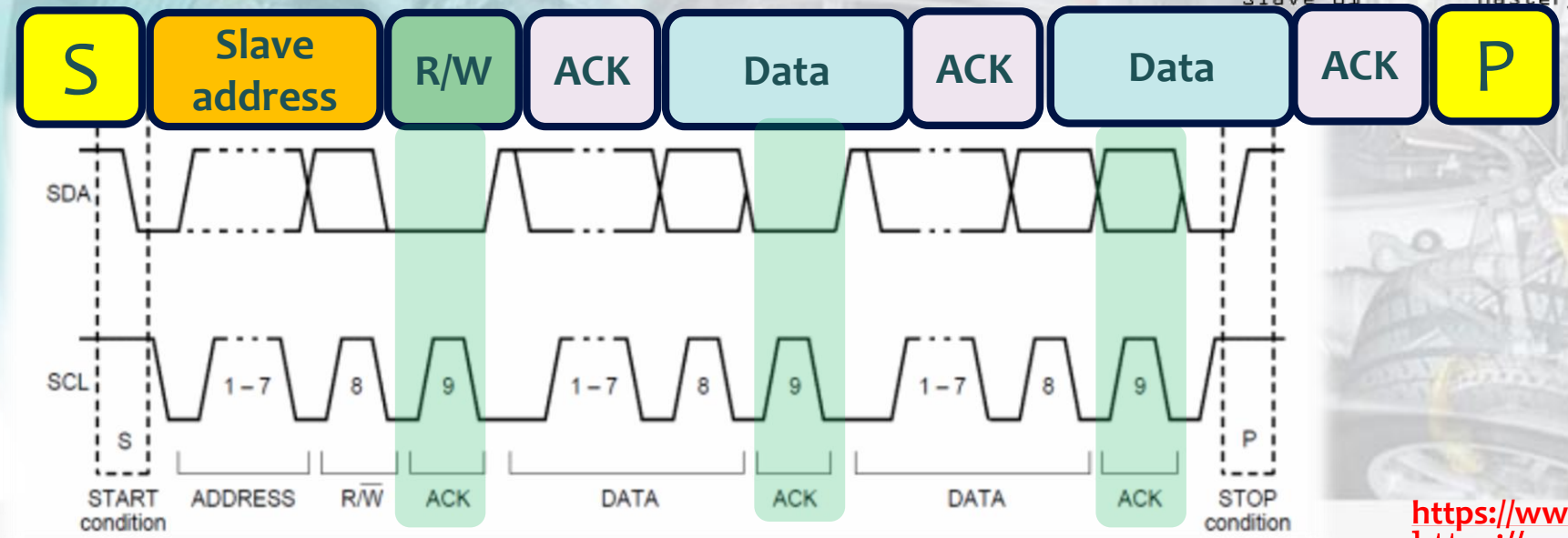
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

7. **If master transmitting**, master issues stop condition (P) after last byte of data

else master receiving, last byte is not acknowledged by master then the master issues stop condition (P)



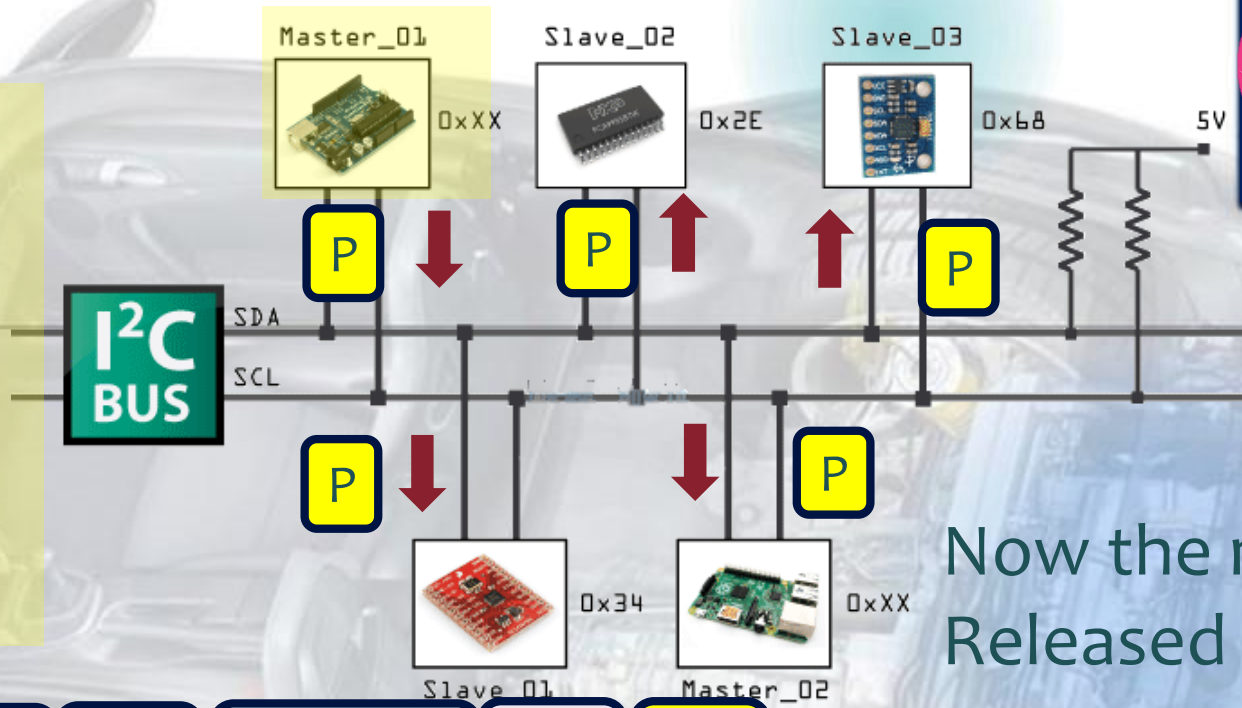
Now the master_01 Released the BUS



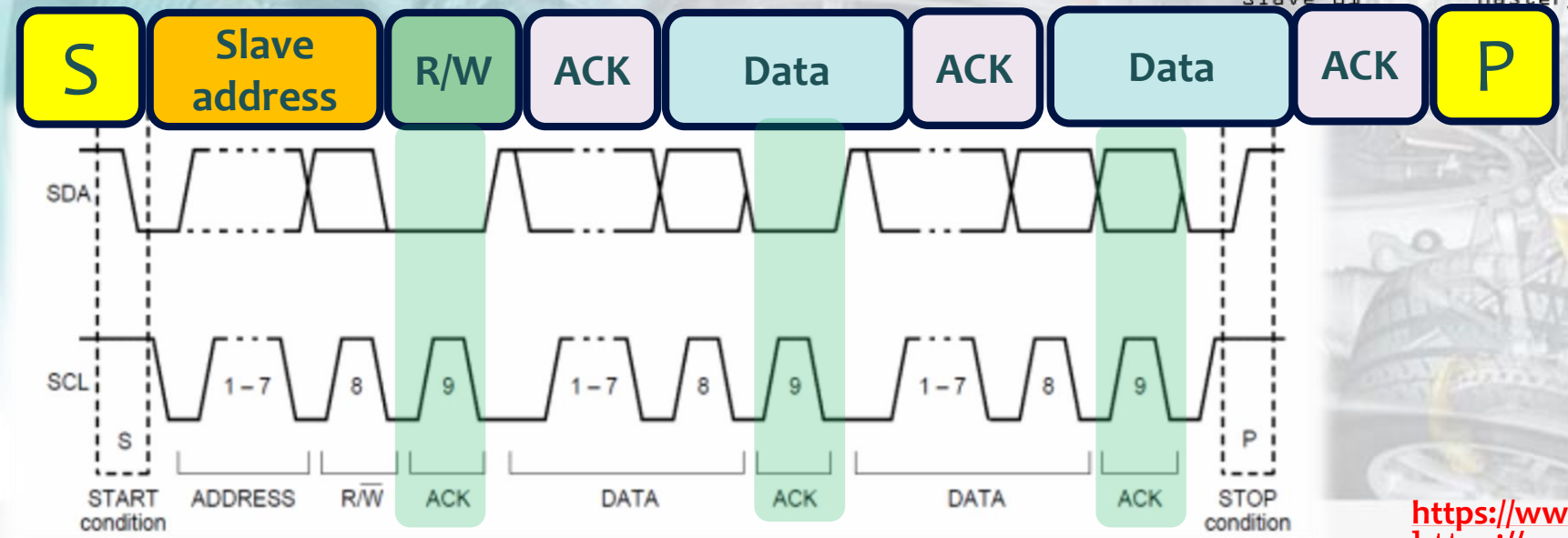
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

7. **If master transmitting**, master issues stop condition (P) after last byte of data

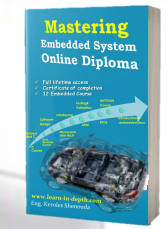
else master receiving, last byte is not acknowledged by master then the master issues stop condition (P)



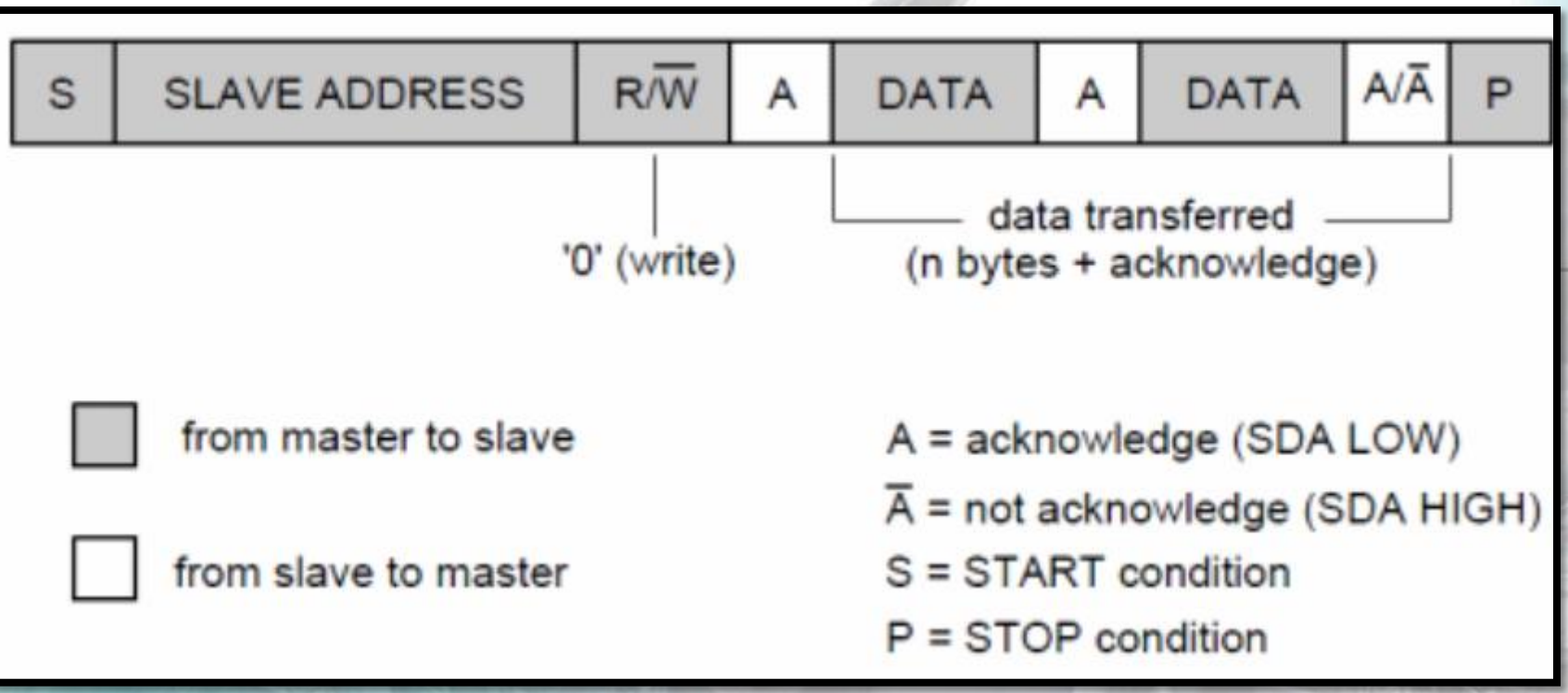
Now the master_01 Released the BUS



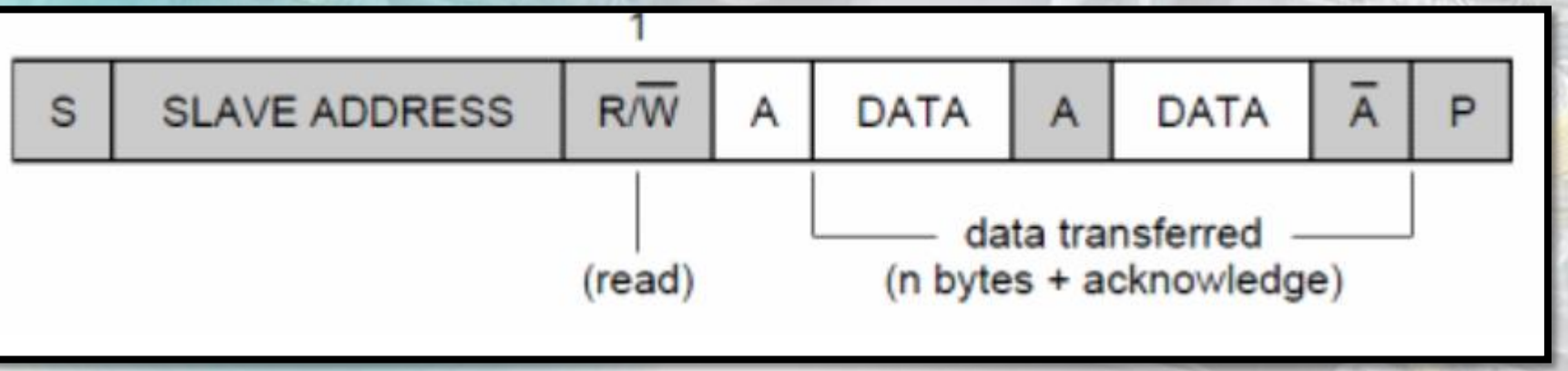
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

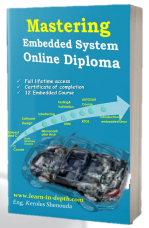


Master Write To Slave



Master Read from Slave

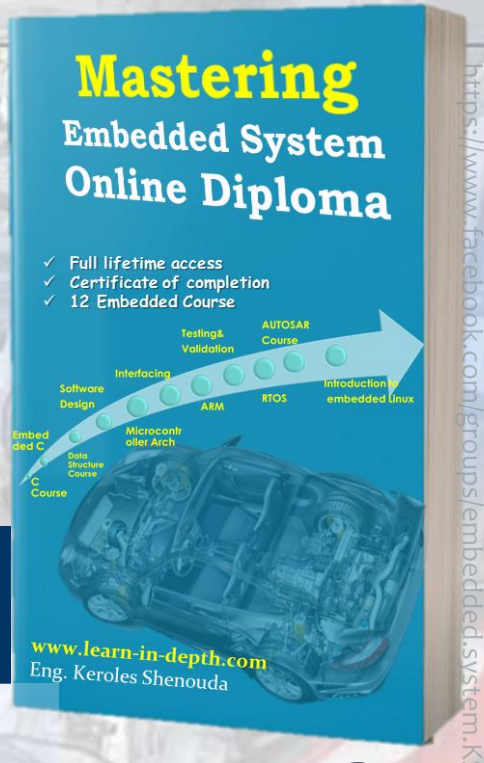




#LEARN_IN DEPTH

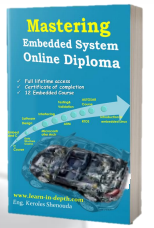
#Be_professional_in_embedded_system

I2C data sampling



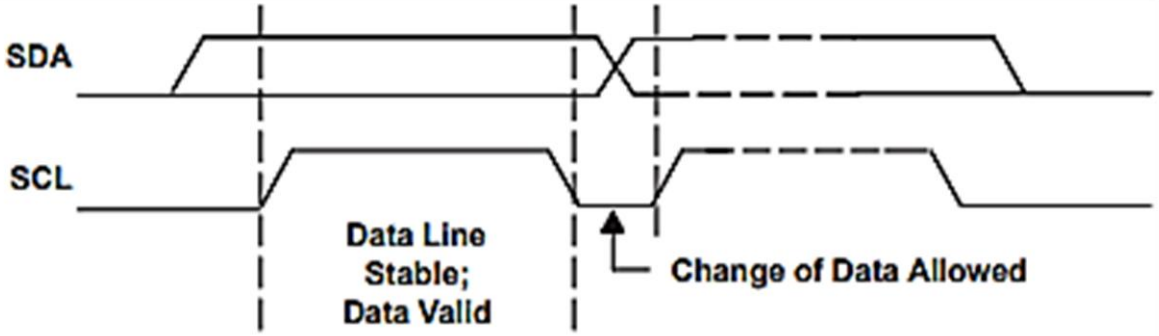
LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

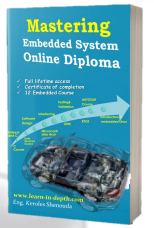


I2C data sampling

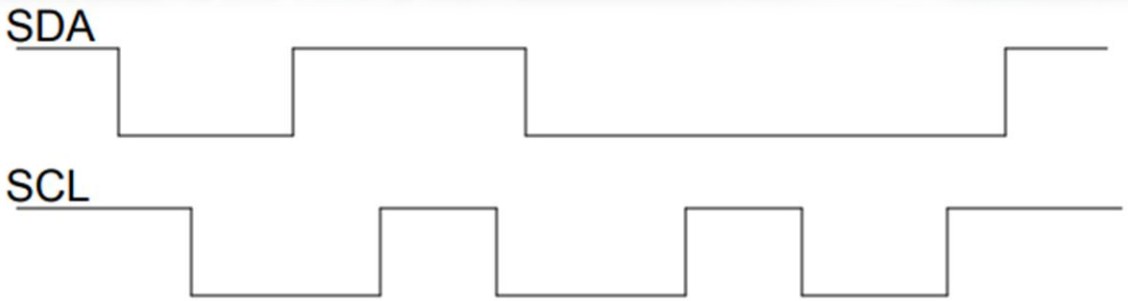
- The state of SDA(high or low) can change only when SCL is low.
- **This means SDA must be stable when SCL high.**



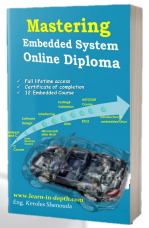
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



I2C Signals



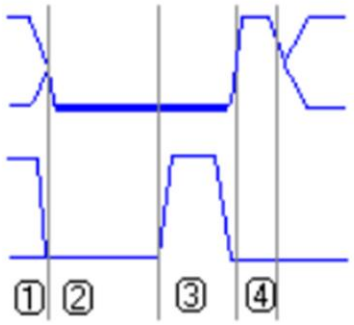
- Start = SDA + SCL
- Stop = SDA + SCL
- Data = value when SCL high; transition when SCL low
- ACK = receiver pulls SDA low; transmitter lets it float



I2C: ACK, NACK, No ACK

Get ACK from slave:

1. master pulls SCL low to complete the transmission of the bit
2. slave pulls SDA low
3. master issues a clock pulse on the SCL line
4. slave releases the SDA line when clock pulse complete

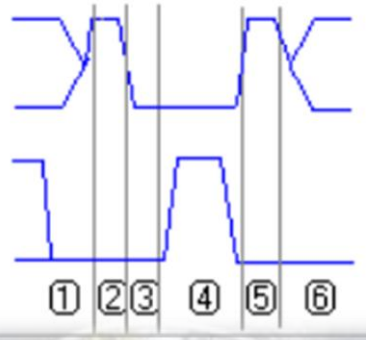


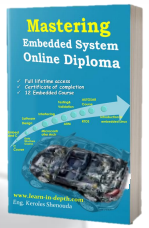
- In case of data being written to a slave, the ACK must be completed before a stop condition can be generated. The slave will be blocking the bus (SDA kept low by slave) until the master has generated a clock pulse on the SCL line.

Master Write to Slave

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

I2C: ACK, NACK, No ACK





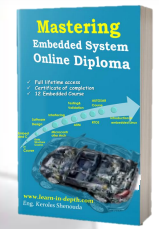
Master Send ACK to slave:

1. Slave transmits last bit
2. **Slave** will **release** the SDA line: line will go high
3. **Master** will **pull the SDA line low**
4. **Master** will put a **clock pulse** on the **SCL** line
5. **Master** will **release** the SDA line.
6. **Slave** will now **regain control** of the **SDA line**

Note: An Acknowledge of a byte received by a slave is always **necessary**, **except on the last byte received**.

Master Read from Slave

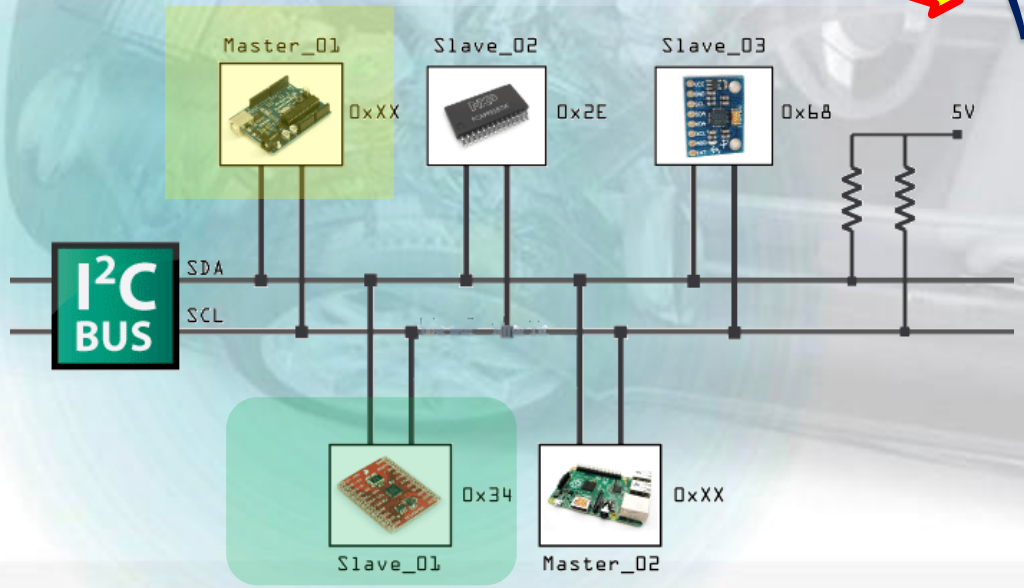
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



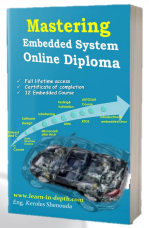
I2C: ACK, NACK, No ACK

- No acknowledge incase (Master Send data to the Slave)
- If, after transmission of the 8th bit from the master to the slave, the slave does not pull the SDA line low, then this is considered a No ACK condition

⟺ What is the reason ?

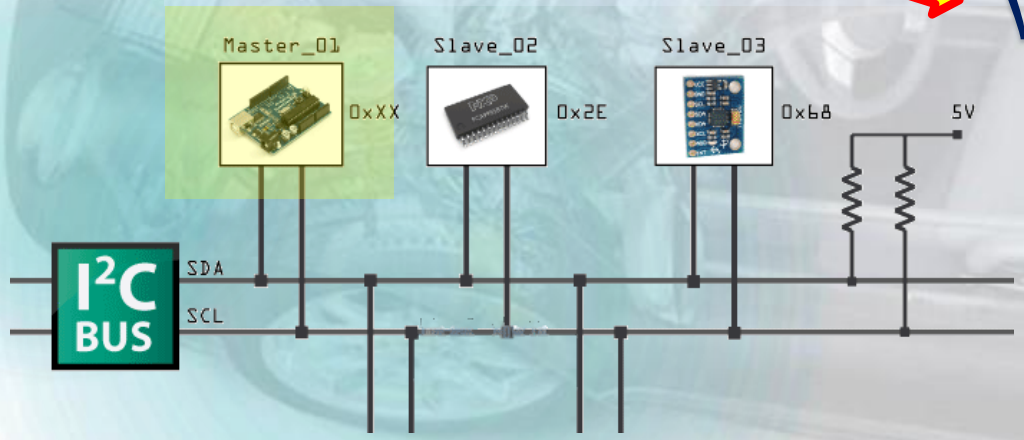


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



I2C: ACK, NACK, No ACK

- No acknowledge incase (Master Send data to the Slave)
- If, after transmission of the 8th bit from the master to the slave, the slave does not pull the SDA line low, then this is considered a No ACK condition



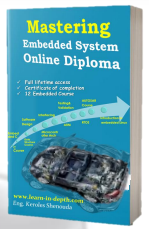
The **slave** is not longer there

What is the reason ?

Maybe:

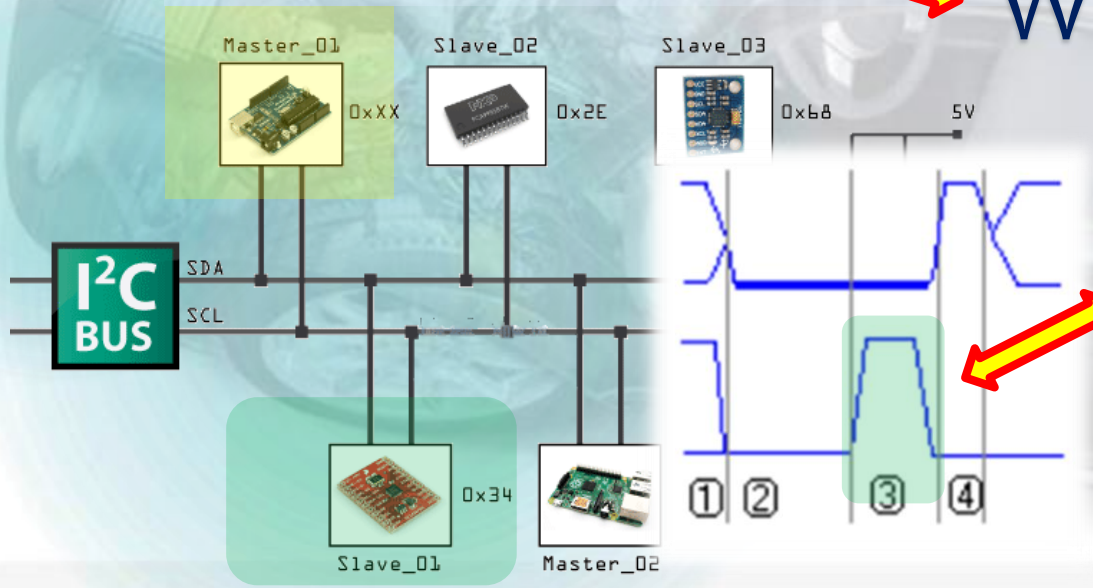
The **slave** is not longer there

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



I2C: ACK, NACK, No ACK

- No acknowledge incase (Master Send data to the Slave)
- If, after transmission of the 8th bit from the master to the slave, the slave does not pull the SDA line low, then this is considered a No ACK condition



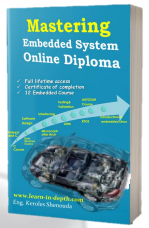
What is the reason ?

Maybe:

-The **slave** is not longer there.

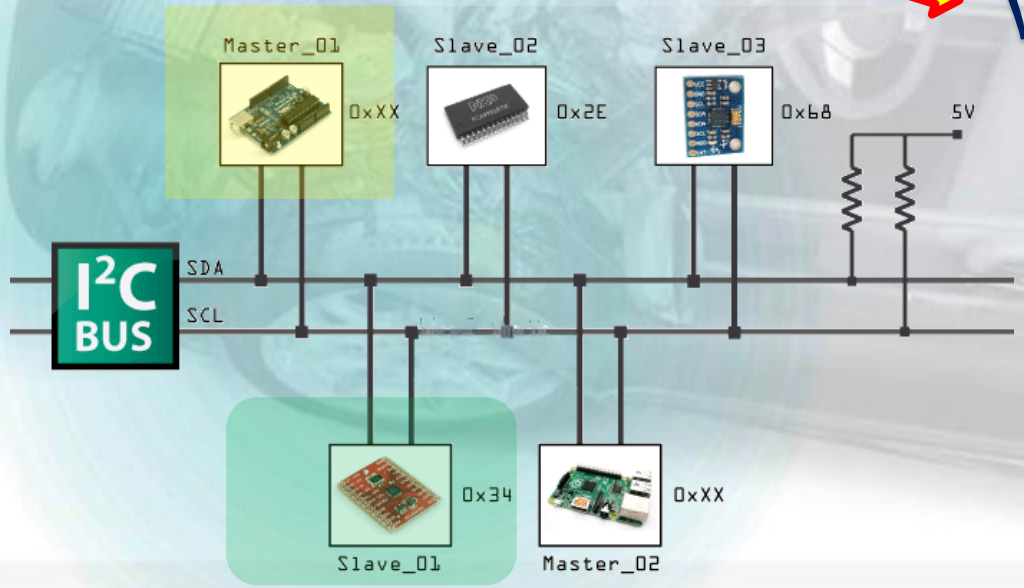
-The **slave missed** a **pulse** and got out of sync with the SCL line of the master

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



I2C: ACK, NACK, No ACK

- No acknowledge incase (Master Send data to the Slave)
- If, after transmission of the 8th bit from the master to the slave, the slave does not pull the SDA line low, then this is considered a No ACK condition



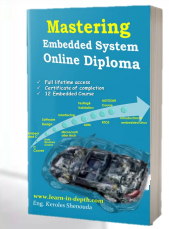
What is the reason ?

Maybe:

-The **slave** is not longer there.

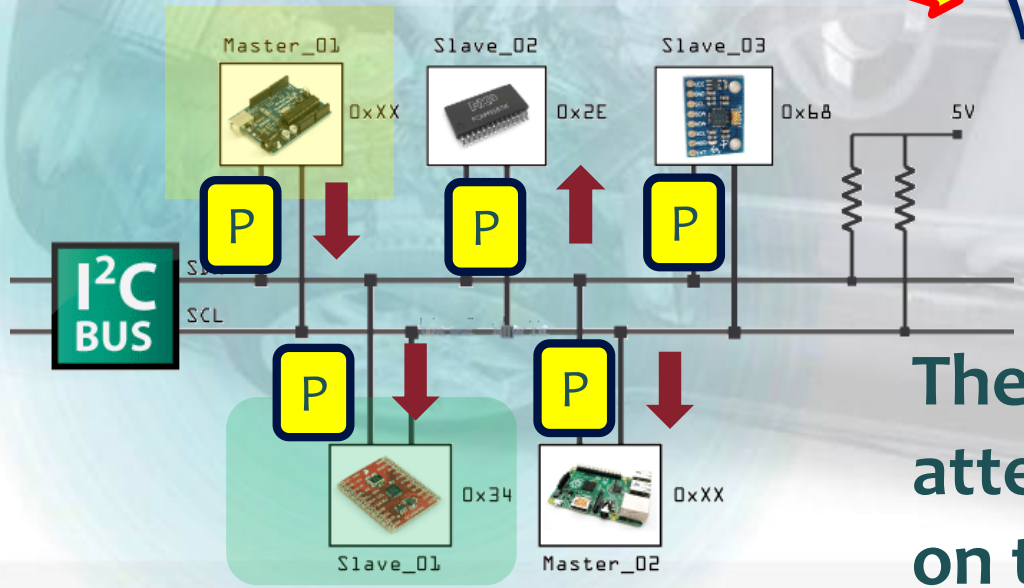
-The **slave missed** a **pulse** and got out of sync with the SCL line of the master.

-The **bus is "stuck".** One of the lines could be held low **permanently.**



I2C: ACK, NACK, No ACK

- No acknowledge incase (Master Send data to the Slave)
- If, after transmission of the 8th bit from the master to the slave, the slave does not pull the SDA line low, then this is considered a No ACK condition

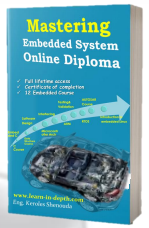


What is the reason ?

Maybe:
-The **slave** is not longer there.
-The **slave missed** a **pulse** and got out of sync with the SCL line of the master.
-The **bus is "stuck".** One of the lines could be held low **permanently.**

Then Master should abort by attempting to send a stop condition on the bus.

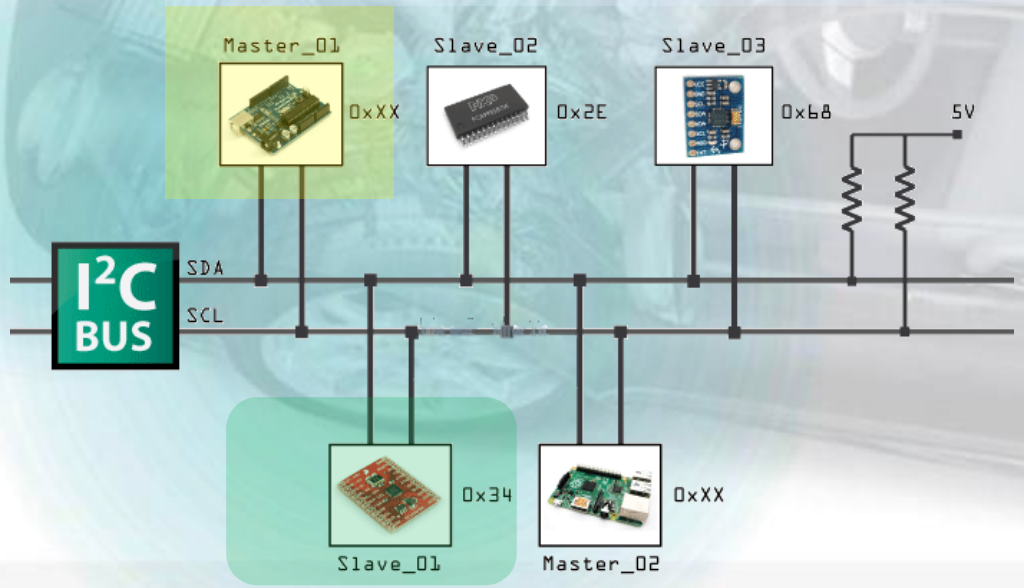
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



I2C: ACK, NACK, No ACK

- Not acknowledge (NACK) - after a master has read a byte from a slave

That happen If the master wants to stop receiving data from the slave, it will not send ACK and then it must be able to send a stop condition

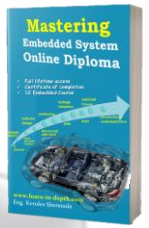


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

NACK Conclusion

When we received the NACK	Who was transmitting data	What it means
After the slave address	Master	No device with that address exists on this bus
After the command byte	Master	Invalid command
After the data transfer	Master	The receiving device will not be able to accept any more data
After the data transfer	Slave	The master does not want to transfer any additional data

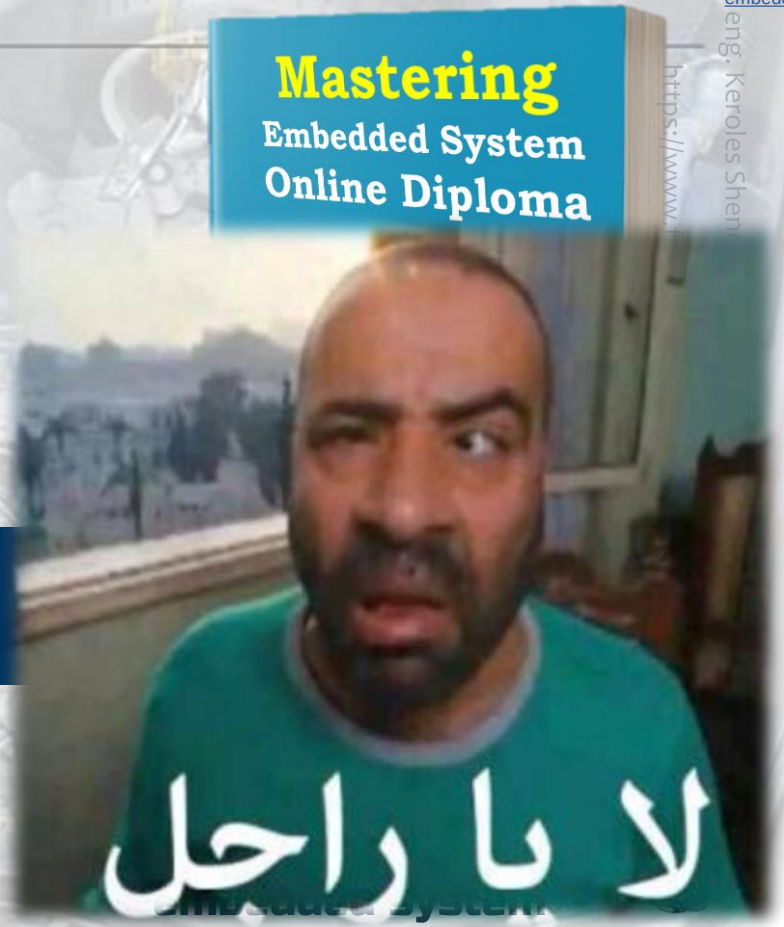
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



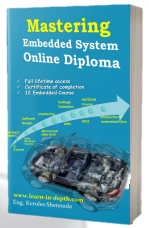
#LEARN_IN DEPTH

#Be_professional_in_embedded_system

What happen if two masters initiate start condition **at the same time** or have multiple slave devices that respond to a single address?

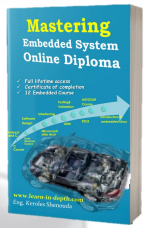


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



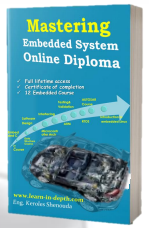
I2C Arbitration

- I2C protocol supports a **multi-master** bus system. This doesn't mean that more than one master can use the bus at the same time. Rather, each master **waits** for the current transmission to **finish** and then starts to use the bus.
- But it is possible that two or more masters initiate a transmission at about the same time (a **bus collision** might occur). In this case the **arbitration** happens.
- is done by having each device that may be trying to use the data line place their bit of data as they normally would (driving SDA low or leaving it to idle high) and then reading the voltage level on the line to see if it is what the device expects.



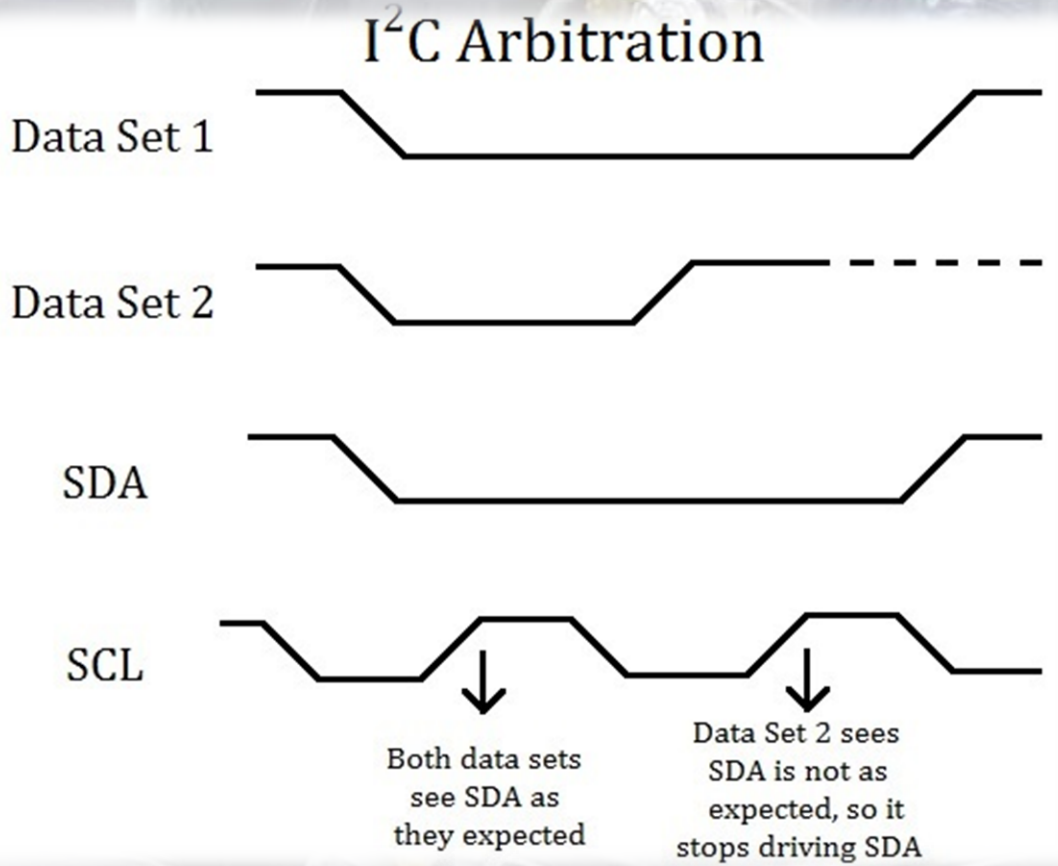
Arbitration

- The **Arbitration** is done by having **each device** that may be trying to use the data line **place their bit of data as they normally would** (driving SDA low or leaving it to idle high) **and then reading the voltage level** on the line **to see if it is what the device expects.**
- If a device sees that the line has been **driven low when it expects** it **to be idling high**, **it will conclude that another device must also be using the SDA line** and losing arbitration and **wait** for **a stop condition** before **attempting** to **transmit** its **message again**.

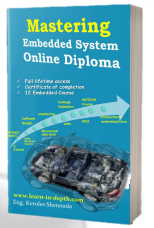


Arbitration

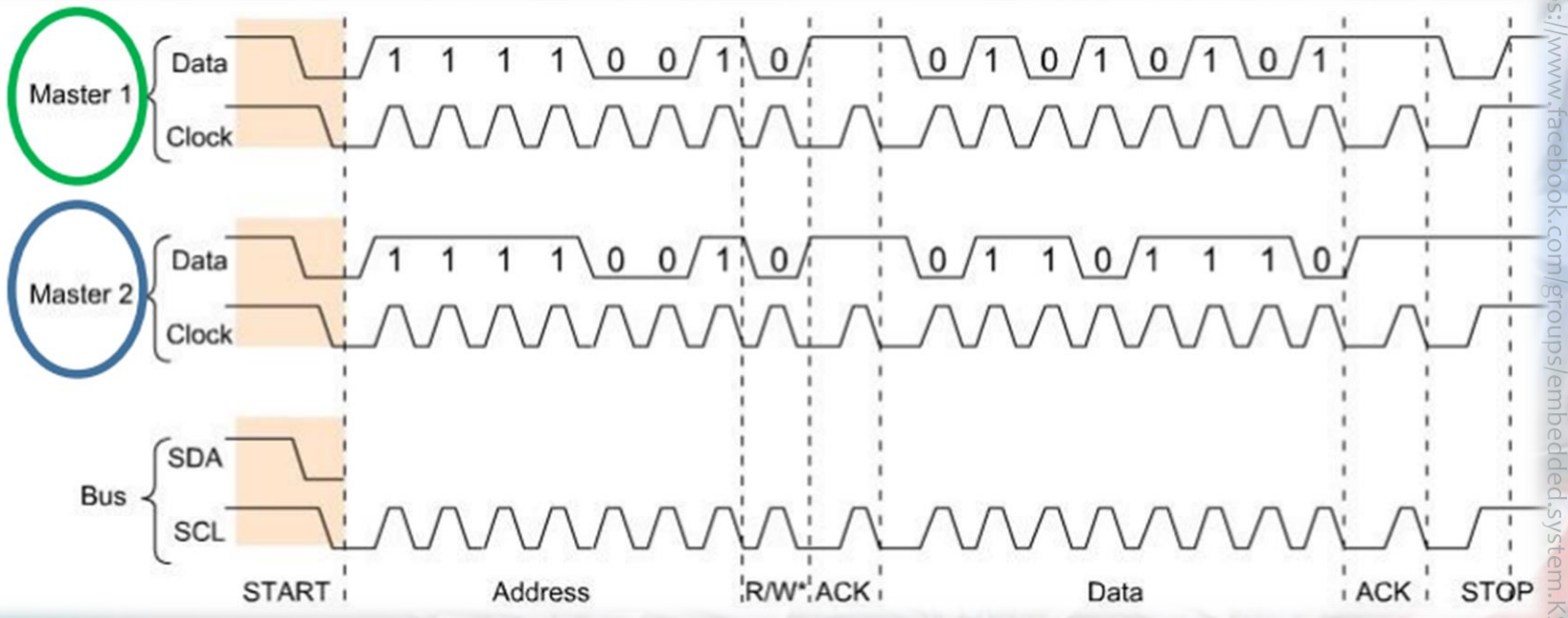
- The **second device**, **seeing** that the **voltage level** is as **it expected**, **will continue transmitting its message although** it will also continue to check the line after each bit in case there is another device that has happened to exactly match its bit values



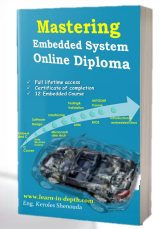
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



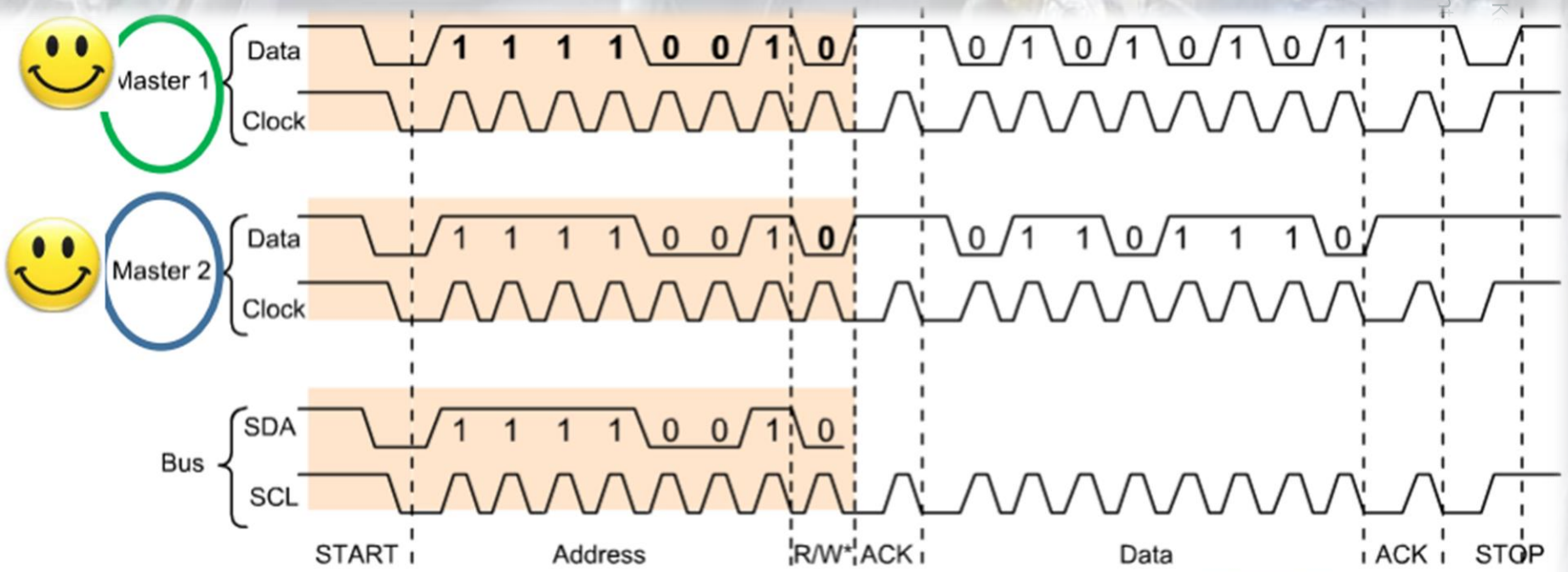
I2C Bus Arbitration(another example)



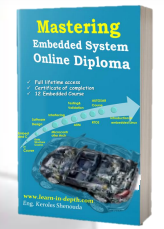
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



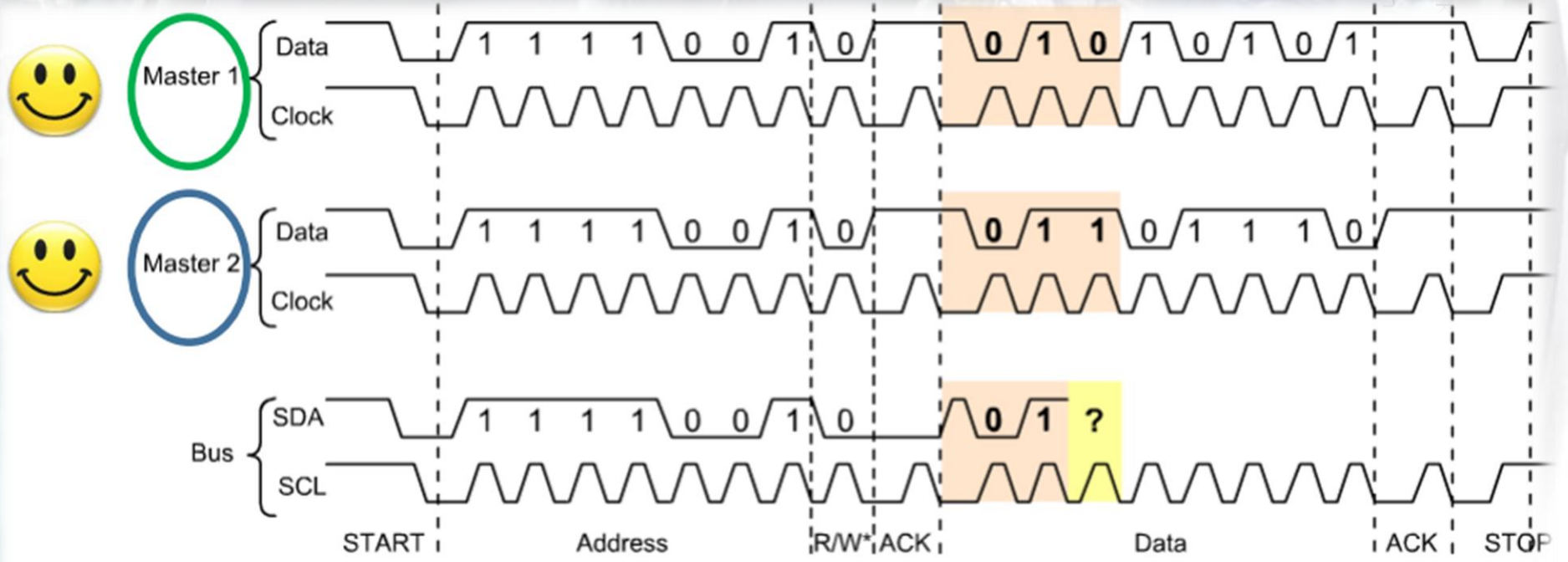
I2C Bus Arbitration(another example)



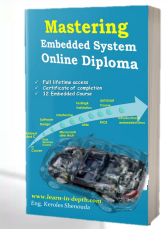
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



I2C Bus Arbitration(another example)

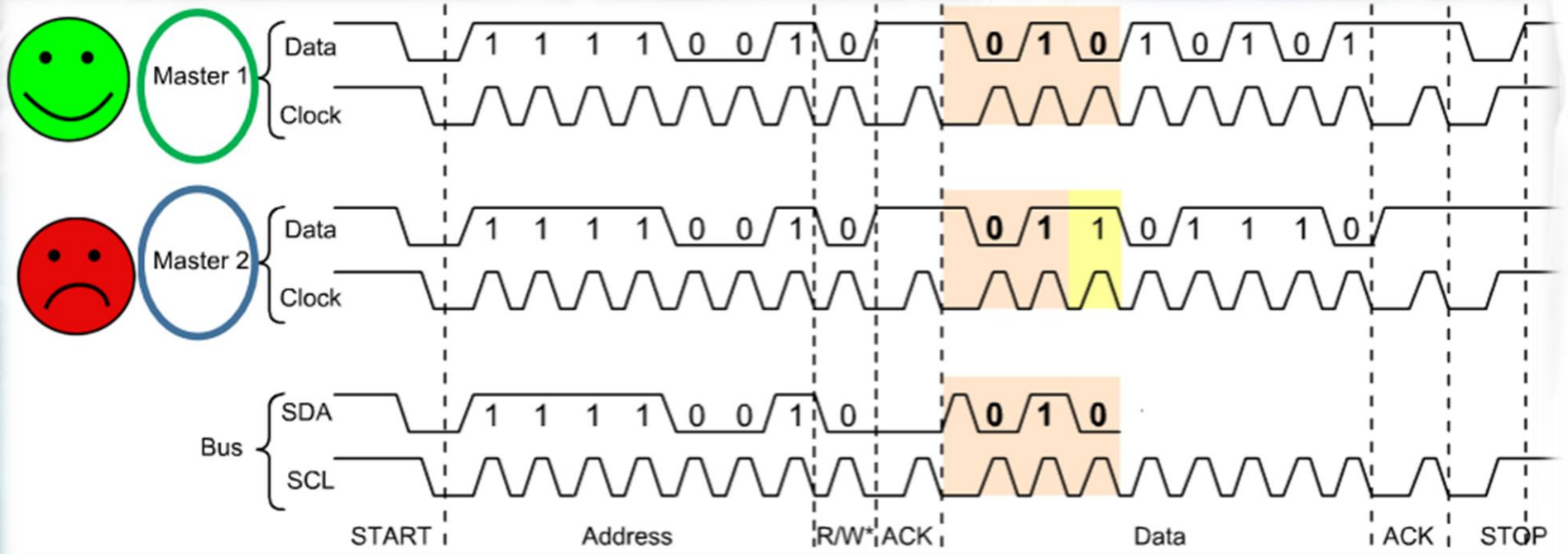


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

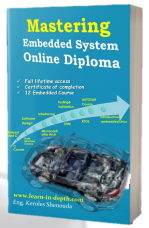


I2C Bus Arbitration(another example)

The **moment their data bits do not match any more** then **The Master 2 loses arbitration** **and It must backs Off** because when Master 2 tries to move the SDA line high the data on the bus remains low (**Master 1 already occupies it**)



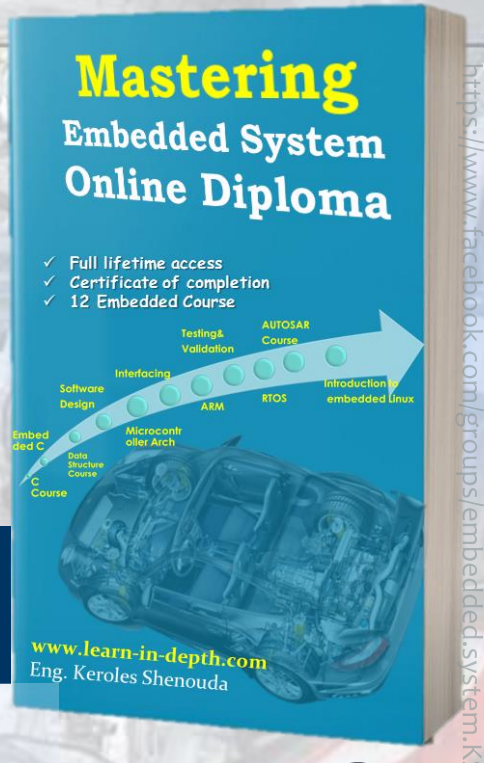
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

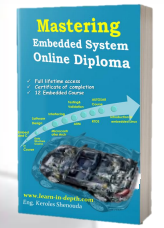
#Be_professional_in_embedded_system

I2C Clock Stretching



LEARN-IN-DEPTH
Be professional in embedded system

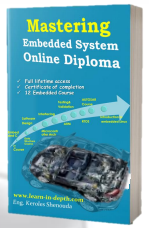
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



What is I2C clock stretching?

- In I2c, communication can be paused by the clock stretching to holding the SCL line low and it cannot continue until the SCL line released high again.

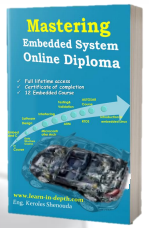
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



- In I2C, slave able to receive a byte of data on the fast rate but sometimes **slave** **takes more time in processing the received bytes** in that situation slave pull the SCL line to pause the transaction and after the processing of the received bytes, it again released the SCL line high again to resume the communication.

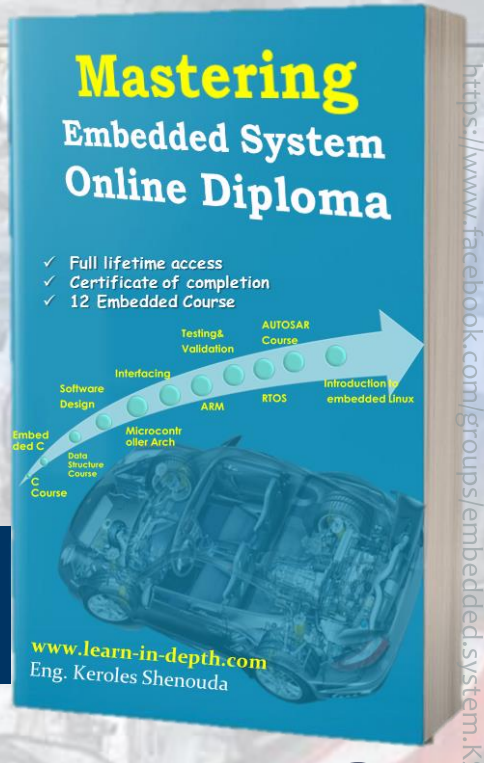
Note: *In the I2c communication protocol, most of the I2C slave devices do not use the clock stretching feature, but every master should support the clock stretching.*

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

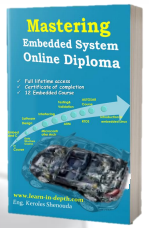
#Be_professional_in_embedded_system



Multibyte burst write/read

LEARN-IN-DEPTH
Be professional in embedded system

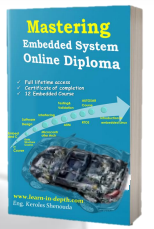
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Multibyte burst write

- Burst mode writing is an effective means of loading consecutive locations
- In burst mode, we provide the address of the first location, followed by the data for that location. From then on, consecutive bytes are written to consecutive memory locations.
  1. Generate a START condition.
  2. Transmit the slave address followed by zero (for write).
  3. Transmit the address of the first location.
  4. Transmit the data for the first location and from then on, simply provide consecutive bytes of data to be placed in consecutive memory locations.
  5. Generate a STOP condition.

Start	Slave address	Write	ACK	First location address	ACK	Data byte #1	ACK	Data byte #2	ACK	Data byte #3	ACK	Stop
S	1111000	0	A	00001111	A	00000001	A	00000010	A	00000011	A	P

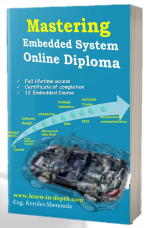


Multibyte burst read

- Burst mode reading is an effective way of bringing out the contents of consecutive locations. In burst mode, we provide the address of the first location only

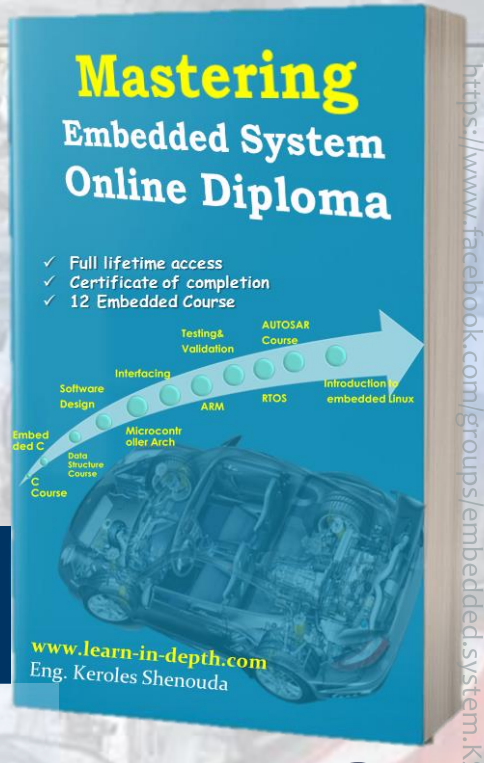
1. Generate a **START** condition
2. Transmit the **slave address followed by zero** (for address write).
3. Transmit the **address of the first location**.
4. Generate a **START (REPEATED START)** condition.
5. Transmit the **slave address followed by one** (for read).
6. **Read the data from the first location** and from then on, bring contents out from consecutive memory locations.
7. Generate a STOP condition.

Start	Slave address	Write	ACK	First location address	ACK	Start	Slave address	Read	ACK	Data byte #1	ACK	Data byte #2	ACK	Data byte #3	ACK	Stop
S	1111000	0	A	00001111	A	S	1111000	1	A	xxxxxxxx	A	xxxxxxxx	A	xxxxxxxx	A	P



#LEARN_IN DEPTH

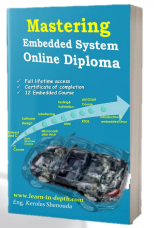
#Be_professional_in_embedded_system



I2C clock synchronization

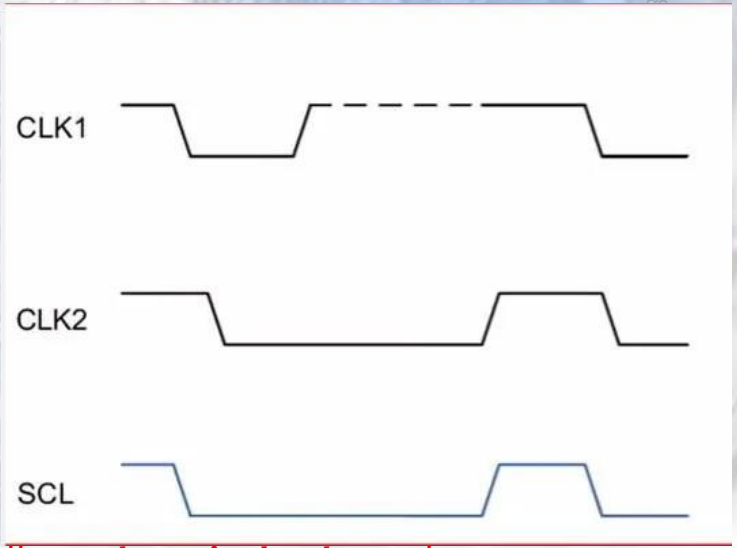
LEARN-IN-DEPTH
Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

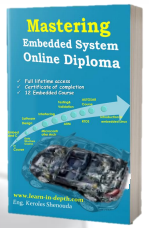


2C clock synchronization

- the clock is always generated by the master and this clock is shared by both master and slave. In the case of multi-master,
- all master generate their own SCL clock, hence it is necessary that the clock of ***all master should be synchronized***.
- In the i2C, this clock synchronization is done by **wired and logic**.
- The SCL clock would be the Anding (clk1 & clk2) of clk1 and clk2 and most interesting thing is that highest logic 1 of SCL line defines by the CLK which has lowest logic 1.

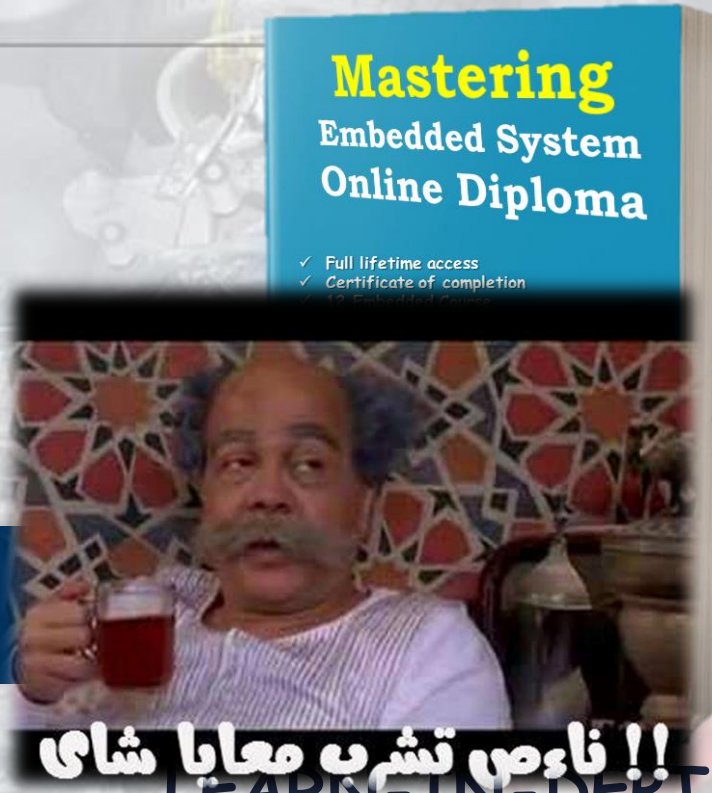


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

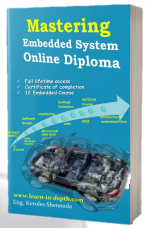


In your opinion ☺

HOW TO SELECT Between I2C and SPI



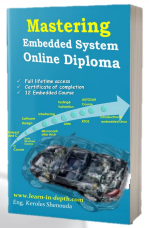
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

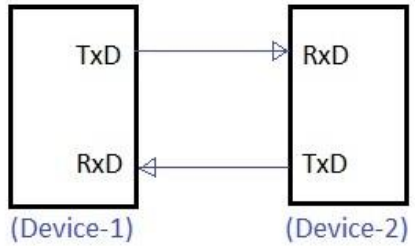
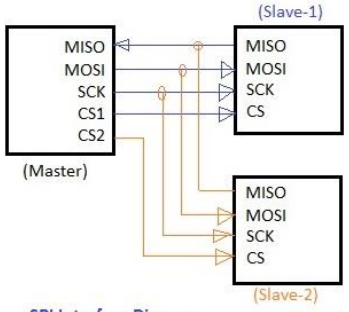
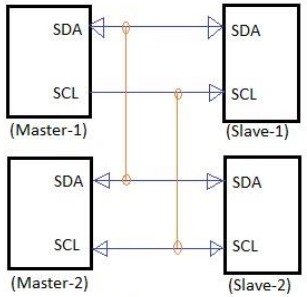


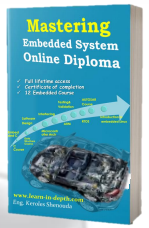
Selecting Between I2C and SPI

- the two main serial communication protocols, requires a good understanding of the advantages and limitations of I2C, SPI, and your application
- **I2C** requires only **two wires**, while **SPI** requires **three or four**
- **SPI** supports **higher speed full-duplex communication** while **I2C** is **slower**
- **I2C** supports **multiple devices on the same bus without additional select signal lines** through in-communication device addressing while **SPI** requires **additional signal lines to manage multiple devices on the same bus**
- **I2C ensures** that **data sent is received by the slave** device while **SPI** does not verify that data is received correctly
- **I2C can be locked** up by one device that **fails** to release the communication bus
- **I2C** is **cheaper** to implement than the **SPI** communication protocol
- **SPI** only supports one **master device** on the bus while **I2C** supports multiple master devices

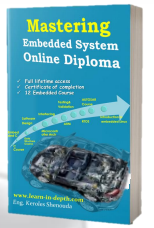
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

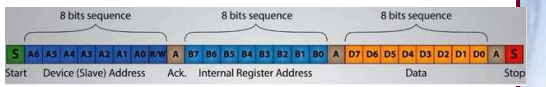


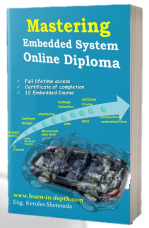
Features	UART	SPI	I2C
Full Form	Universal Asynchronous Receiver/Transmitter	Serial Peripheral Interface	Inter-Integrated Circuit
Interface Diagram	 UART Interface Diagram	 SPI Interface Diagram	 I2C Interface Diagram
Pin Designations	TxD: Transmit Data RxD: Receive Data	SCLK: Serial Clock MOSI: Master Output, Slave Input MISO: Master Input, Slave Output SS: Slave Select	SDA: Serial Data SCL: Serial Clock



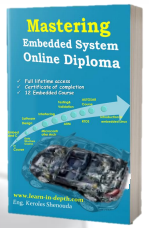
Features	UART	SPI	I2C
Data rate	As this is is asynchronous communication, data rate between two devices wanting to communicate should be set to equal value. Maximum data rate supported is about 230 Kbps to 460kbps.	Maximum data rate limit is not specified in SPI interface. Usually supports about 10 Mbps to 20 Mbps	I2C supports 100 kbps, 400 kbps, 3.4 Mbps. Some variants also supports 10 Kbps and 1 Mbps.
Distance	Lower about 50 feet	highest	Higher
Type of communication	Asynchronous	Synchronous	Synchronous



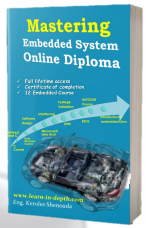
Features	UART	SPI	I2C
Number of masters	Not Application	One	One or more than One
Clock	No Common Clock signal is used. Both the devices will use there independent clocks.	There is one common serial clock signal between master and slave devices.	There is common clock signal between multiple masters and multiple slaves.
Hardware complexity	lesser	less	more
Protocol	For 8 bits of data one start bit and one stop bit is used.	Each company or manufacturers have got their own specific protocols to communicate with peripherals. Hence one needs to read datasheet to know read/write protocol for SPI communication to be established. For example we would like SPI communication between microcontroller and EPROM. Here one need to go through read/write operational diagram in the EPROM data sheet.	 It uses start and stop bits. It uses ACK bit for each 8 bits of data which indicates whether data has been received or not. Figure depicts the data communication protocol.



Features	UART	SPI	I2C
Software addressing	As this is one to one connection between two devices, addressing is not needed.	Slave select lines are used to address any particular slave connected with the master. There will be 'n' slave select lines on master device for 'n' slaves.	There will be multiple slaves and multiple masters and all masters can communicate with all the slaves. Upto 27 slave devices can be connected/addressed in the I2C interface circuit.
Advantages	• It is simple communication and most popular which is available due to UART support in almost all the devices with 9 pin connector. It is also referred as RS232 interface.	• It is simple protocol and hence so not require processing overheads. • Supports full duplex communication. • Due to separate use of CS lines, same kind of multiple chips can be used in the circuit design. • SPI uses push-pull and hence higher data rates and longer ranges are possible. • SPI uses less power compare to I2C	• Due to open collector design, limited slew rates can be achieved. • More than one masters can be used in the electronic circuit design. • Needs fewer i.e. only 2 wires for communication. • I2C addressing is simple which does not require any CS lines used in SPI and it is easy to add extra devices on the bus. • It uses open collector bus concept. Hence there is bus voltage flexibity on the interface bus. • Uses flow control.



Features	UART	SPI	I2C
Disadvantages	• They are suitable for communication between only two devices. • It supports fixed data rate agreed upon between devices initially before communication otherwise data will be garbled.	• As number of slave increases, number of CS lines increases, this results in hardware complexity as number of pins required will increase. • To add a device in SPI requires one to add extra CS line and changes in software for particular device addressing is concerned. • Master and slave relationship can not be changed as usually done in I2C interface. • No flow control available in SPI.	• Increases complexity of the circuit when number of slaves and masters increases. • I2C interface is half duplex. • Requires software stack to control the protocol and hence it needs some processing overheads on microcontroller/microprocessor .



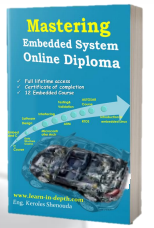
#LEARN_IN DEPTH

#Be_professional_in embedded_system

References

- https://docs.google.com/viewer?a=v&pid=sites&srcid=ZmtlLnV0bS5teXxyaWR6dWFuLXMtd2Vic2l0ZXxneDo2ODU0NzlkM2JkOTg4MjRk
- http://www.avrprojects.net/index.php/avr-projects/sensors/38-humidity-and-temperature-sensor-dht11?showall=&start=1
- http://www.cse.wustl.edu/~lu/cse467s/slides/dsp.pdf
- http://www.avr-tutorials.com/
- Microprocessor: ATmega32 (SEE3223-10) http://ridzuan.fke.utm.my/microprocessor-atmega32-see3223-10
- http://circuitdigest.com/article/what-is-the-difference-between-microprocessor-and-microcontroller
- http://cs4hs.cs.pub.ro/wiki/roboticsisfun/chapter2/ch2_7_programming_a_microcontroller
- Embedded Systems with ARM **Cortex-M Microcontrollers in** Assembly Language and C Dr. Yifeng Zhu Third edition June 2018

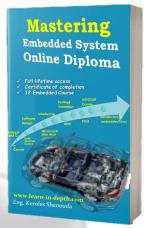
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



References

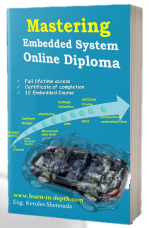
- http://techdifferences.com/difference-between-interrupt-and-polling-in-os.html
- http://www.bogotobogo.com/Embedded/hardware_interrupt_software_interrupt_latency_irq_vs_fiq.php
- Preventing Interrupt Overload Presented by Jiyong Park Seoul National University, Korea 2005. 2. 22. John Regehr, Usit Duogsaa, School of Computing, University.
- First Steps Embedded Systems Byte Craft Limited reference
- COMPUTER ORGANIZATION AND ARCHITECTURE DESIGNING FOR PERFORMANCE EIGHTH EDITION **William Stallings**
- **Getting Started with the Tiva™ TM4C123G LaunchPad Workshop**

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



References

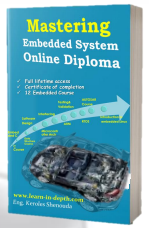
- Tiva™ TM4C123GH6PM Microcontroller DATA SHEET
- Interrupts and Exceptions COMS W6998 Spring 2010
- THE **AVR MICROCONTROLLER**. AND EMBEDDED SYSTEMS Using Assembly and C. Muhammad Ali **Mazidi**.
- http://embedded-lab.com/blog/tinkering-ti-msp430f5529/27/



References

- https://docs.google.com/viewer?a=v&pid=sites&srcid=ZmtlLnV0bS5teXxyaWR6dWFuLXMtd2Vic2l0ZXxneDo2ODU0NzlkM2JkOTg4MjRk
- http://www.avrprojects.net/index.php/avr-projects/sensors/38-humidity-and-temperature-sensor-dht11?showall=&start=1
- http://www.cse.wustl.edu/~lu/cse467s/slides/dsp.pdf
- http://www.avr-tutorials.com/
- Microprocessor: ATmega32 (SEE3223-10) http://ridzuan.fke.utm.my/microprocessor-atmega32-see3223-10
- http://circuitdigest.com/article/what-is-the-difference-between-microprocessor-and-microcontroller
- AVR Microcontroller and Embedded Systems: Using Assembly and C (Pearson Custom Electronics Technology) 1st Edition https://www.amazon.com/AVR-Microcontroller-Embedded-Systems-Electronics/dp/0138003319

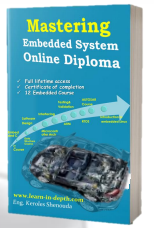
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



References

- https://www.newbiehack.com/MicrocontrollersABeginnersGuideIntroductionandInterfacinganLCD.aspx
- http://www.slideshare.net/MathivananNatarajan/asynchronous-serial-data-communication-and-standards
- https://www.slideshare.net/AnkitSingh13/uart-32550652
- **the avr** microcontroller and embedded. System using assembly and c. Muhammad Ali **Mazidi**
- Embedded Systems lectures "Engr. Rashid Farid Chishti"
- https://www.corelis.com/education/SPI_Tutorial.htm
- http://ftm.futureelectronics.com/2014/09/nxp-macronics-nor-series-quad-spi-flash-a-simpler-faster-alternative-to-standard-spi-flash-when-adding-external-memory-to-32-bit-mcu-systems/
- http://www.byteparadigm.com/products/spi-storm/spi-storm-advanced-information/
- https://stackoverflow.com/questions/17125505/what-makes-a-better-constant-in-c-a-macro-or-an-enum
- https://blog.digilentinc.com/i2c-how-does-it-work/#prettyPhoto
- https://rophoenixmakerevolution.files.wordpress.com/2015/09/spi-and-can-bus.pdf
- http://denethor.wlu.ca/cp316/lectures/Serial_Interconnect_Bus.pdf
- https://aticleworld.com/i2c-interview-questions/

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



#LEARN_IN DEPTH

#Be_professional_in_embedded_system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/