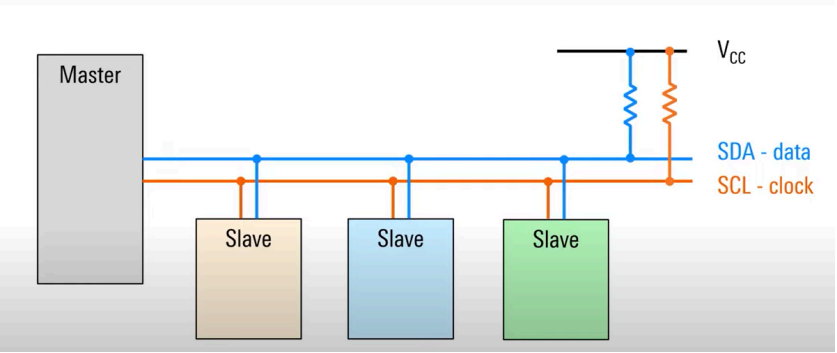
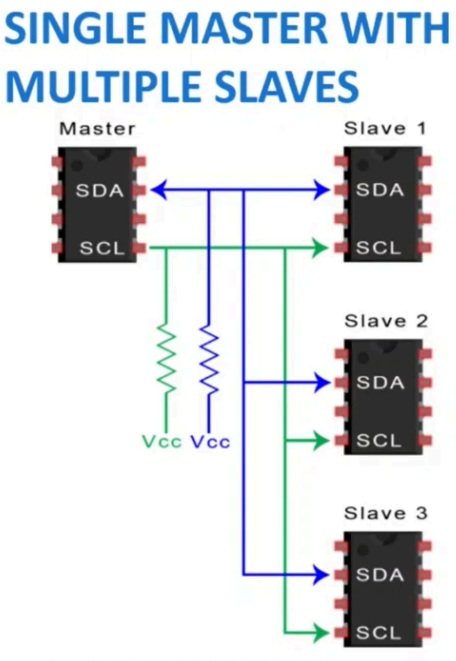
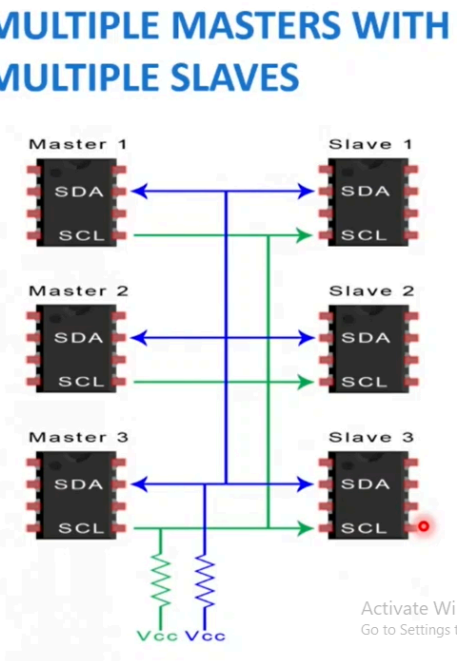
I2C Project Report

Introduction:

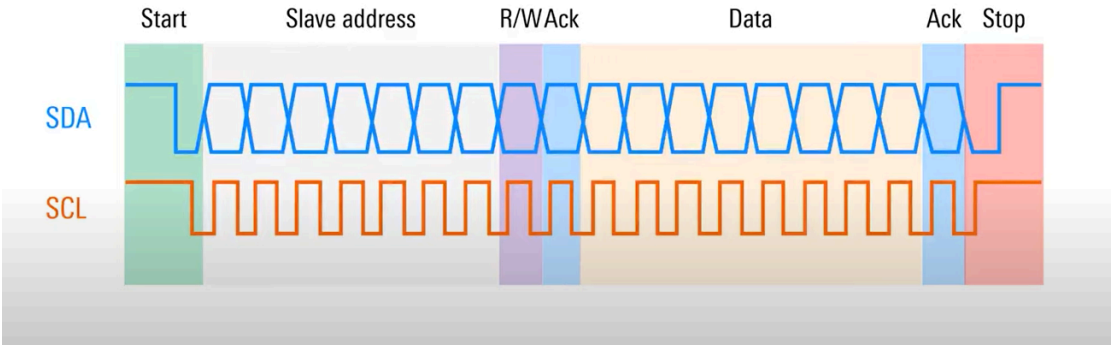
- Inter-Integrated Circuit (I2C or I2C)
- Developed in 1982 by Philips (now NXP)
- Very common, Primarily used for short-distance data communications
- Synchronous master - slave protocol
- Both master and slaves can send / receive data
- Bidirectional, half duplex
- Can run at different speeds
- Two wires: serial clock (SCL) and serial data (SDA)





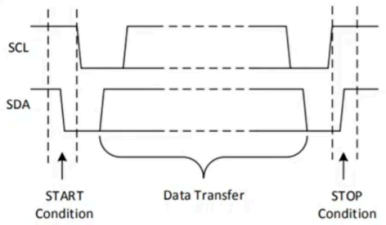


Protocol timing sequence:



How I2C Communication Practically Works?

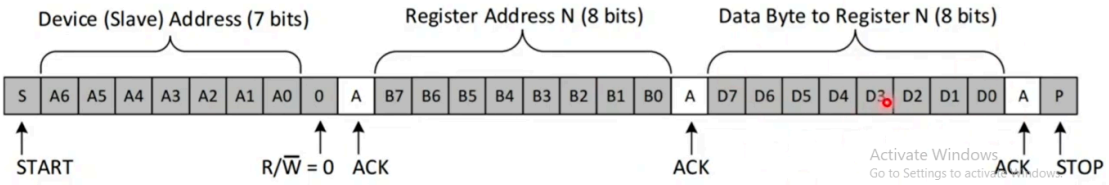
i)Sending Data to a Slave Device



Master Controls SDA Line

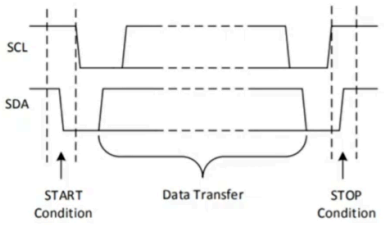
Slave Controls SDA Line

Write to One Register in a Device



How I2C Communication Practically Works?

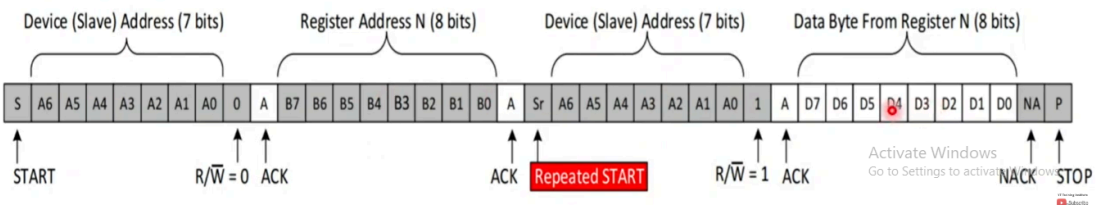
i) Reading Data from a Slave Device



Master Controls SDA Line

Slave Controls SDA Line

Read From One Register in a Device



Start:

> In the idle state SDA and SCL are both high

> The start condition occurs when a node

- First pulls SDA low

- Then pulls SCL low SDA

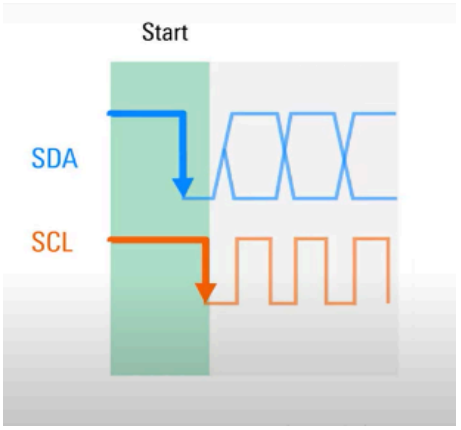
> This "claims the bus"

- Node is now the master

- Prevents any other nodes from taking control of the bus

- Reduces the risk of contention

> Master that has seized the bus also starts the clock



STOP

> Stop condition indicates the end of data bytes

— First, SCL returns (and remains) high

- Then, SDA returns (and remains) high

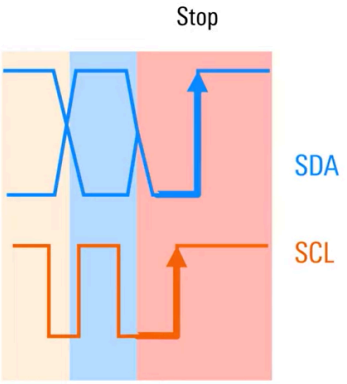
> Recall that for data bytes, SDA only transitions when clock is low

— SDA transition when SCL high = stop condition SDA

> Bus becomes idle SCL

— No clock signal

- Any node can now use the start condition to claim the bus and begin a new communication



Slave address

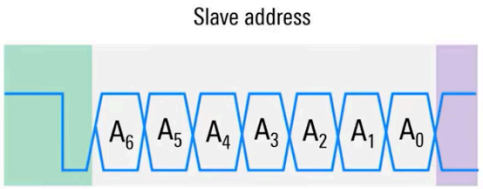
> Each I2C node on a bus must have Slave address a unique, fixed address

— Normally 7 bits long, MSB first

— 10 bit addresses also supported, but these are uncommon

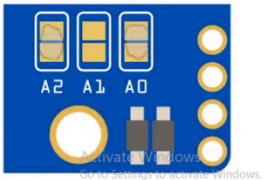
> Address may be hard coded

> Address may be (partially) configurable via external address lines or jumpers



Address: 0101$A_2A_1A_0$

Address: 0101101

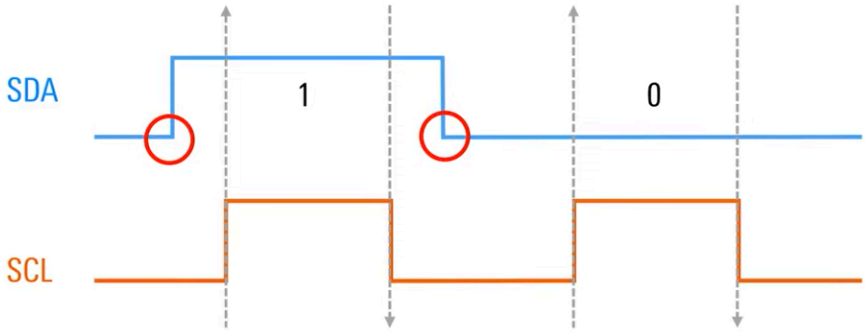


Aside: timing relationship between SDA and SCL

> SDA does not change between clock rising edge and clock falling edge

> During data transmission, SDA only transitions while SCL is low

- An SDA transition when SCL is high, indicates a start or stop condition



Read / write bit

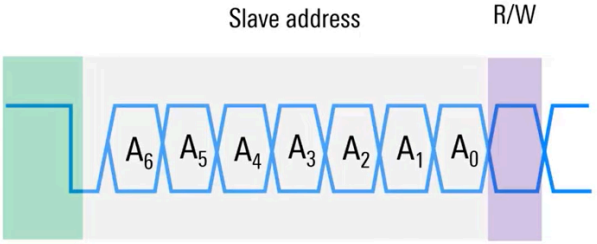
> Read /write bit follows the Slave address

> Set by master to indicate desired operation

0 ->master wants to write data to slave

- 1 —> master wants to read data from slave

> Often interpreted and/or decoded as part of the address byte



Ack(knowledge) bit

> Sent by the receiver of a byte of data

~ 0 -> acknowledgement (ACK)

~ 1 -> negative acknowledgement (NACK)

> Recall that I2C is idle high

- Lack of response = NACK

> Used after slave address and each data byte

> ACK after data byte(s) confirms receipt of data

> ACK after slave address confirms that

~ A slave with that address is on the bus

— The slave is ready to read /write data (depending on RIW bit)

Data byte(s)

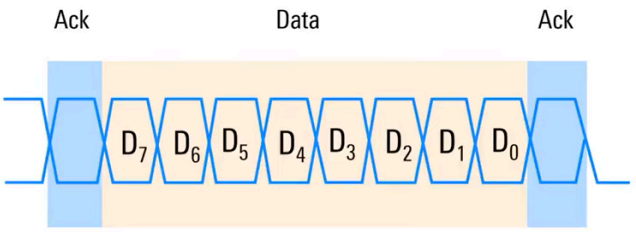
> Data byte contains the information being transferred between master and slave

— Memory or register contents, addresses, etc.

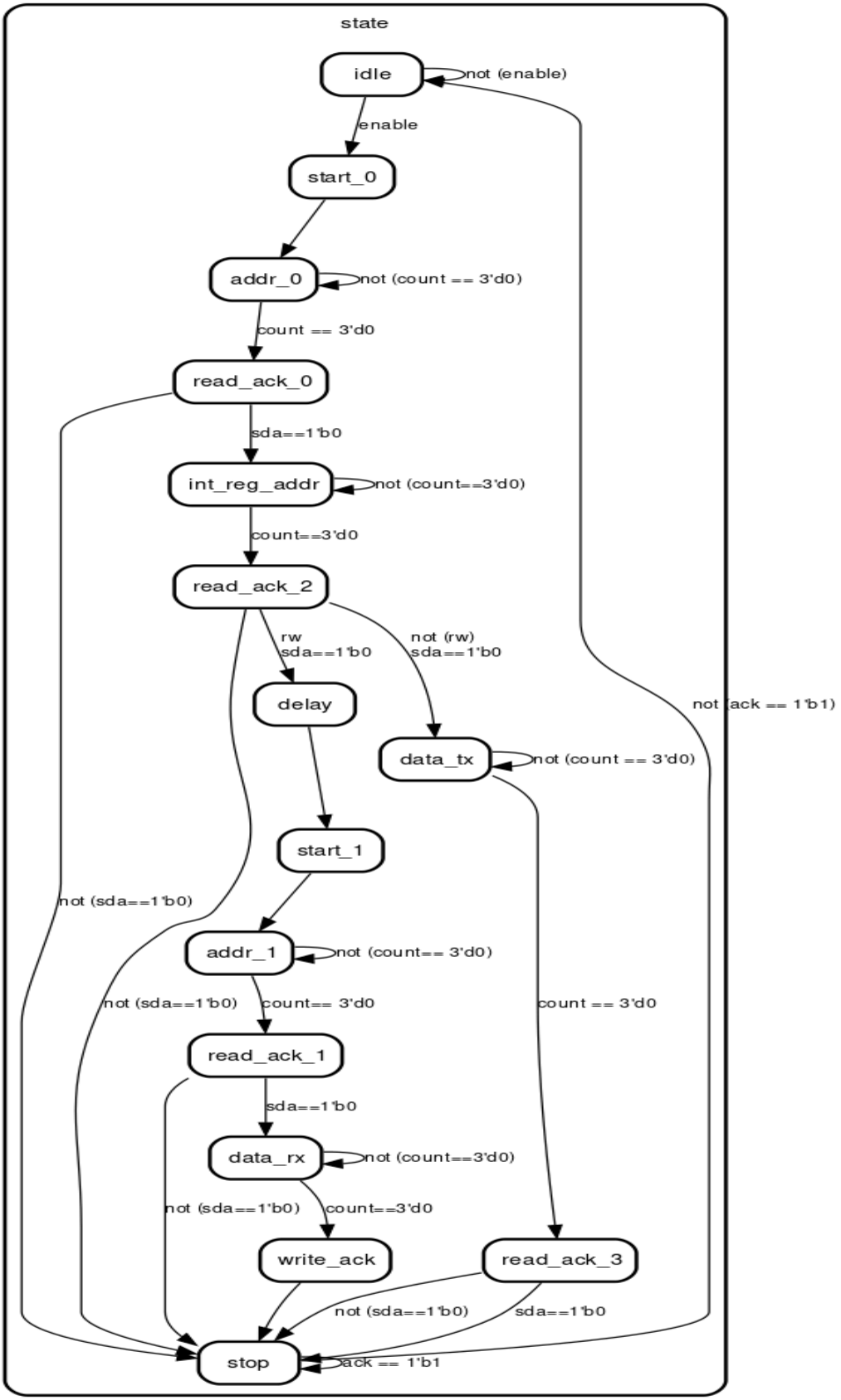
> Always 8 bit long. MSB first

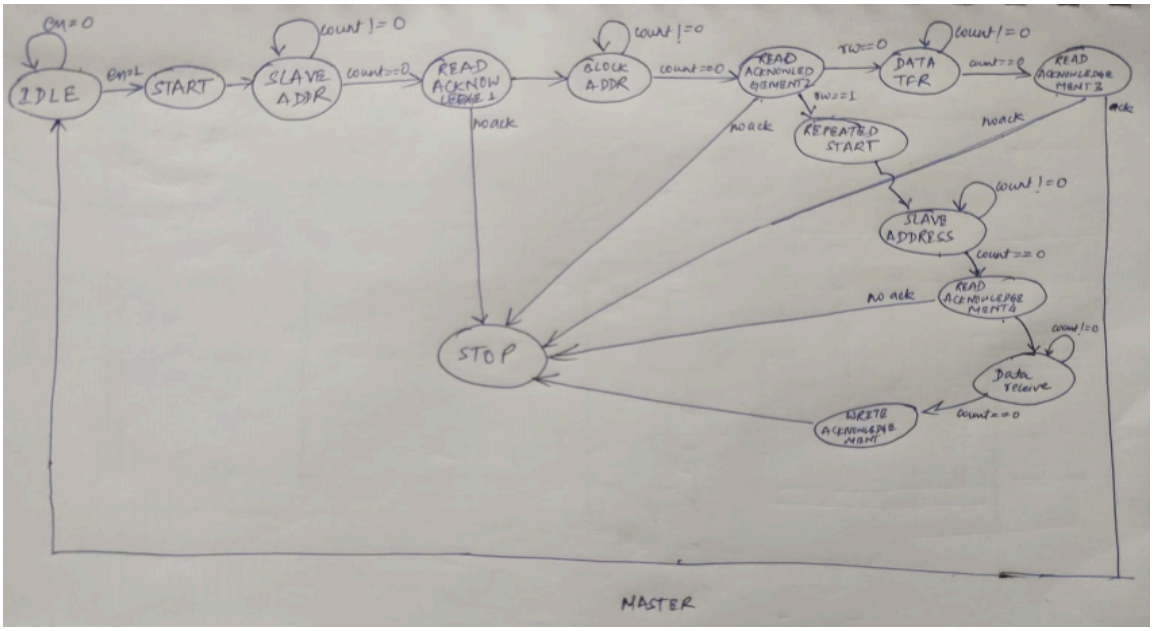
> Always followed by an acknowledgement bit

— Set to zero by the receiver if data has been received properly

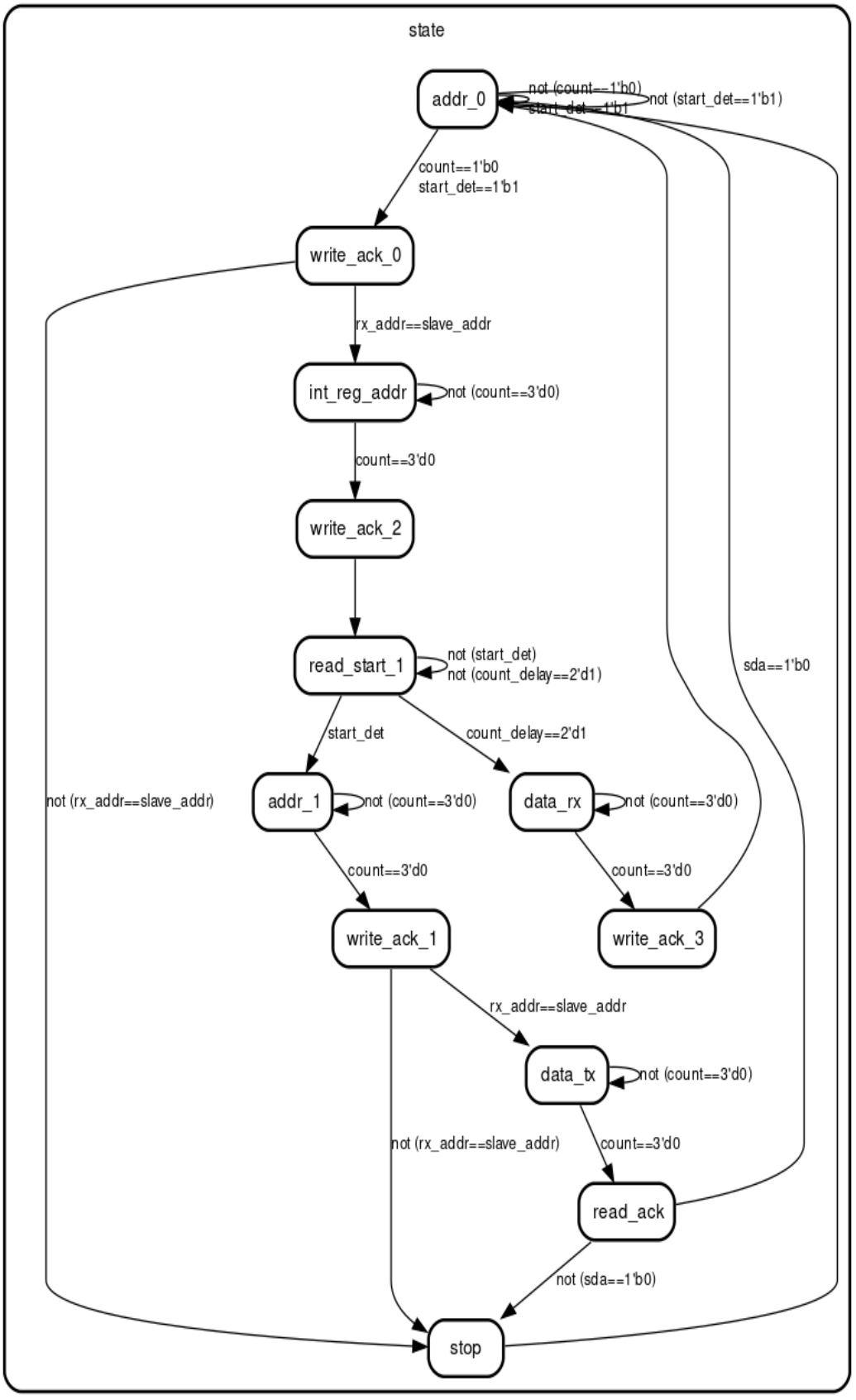


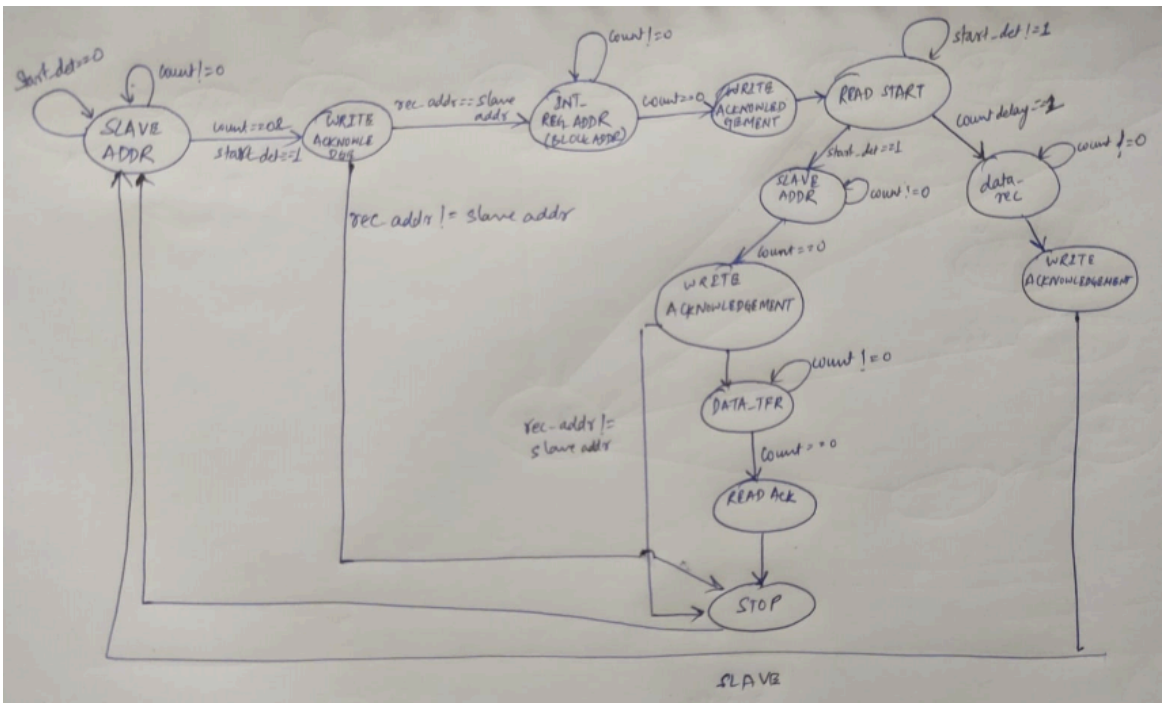
MASTER FSM


state
idle
not (enable)
enable
start_0
addr_0
not (count == 3'd0)
count == 3'd0
read_ack_0
sda==1'b0
int_reg_addr
not (count==3'd0)
count==3'd0
read_ack_2
rw
sda==1'b0
not (rw)
sda==1'b0
delay
data_tx
not (count == 3'd0)
not (ack == 1'b1)
start_1
not (sda==1'b0)
addr_1
not (count== 3'd0)
not (sda==1'b0)
count== 3'd0
count == 3'd0
read_ack_1
sda==1'b0
data_rx
not (count==3'd0)
not (sda==1'b0)
count==3'd0
write_ack
read_ack_3
not (sda==1'b0)
sda==1'b0
stop
ack == 1'b1


en=0
IDLE
en=1
START
count != 0
SLAVE ADDR
count==0
READ ACKNOWLEDGE 1
noack
count != 0
BLOCK ADDR
count==0
READ ACKNOWLEDGEMENT 2
noack
rw==1
REPEATED START
count != 0
SLAVE ADDRESS
count==0
READ ACKNOWLEDGEMENT 3
No ack
count != 0
Data Relcive
count==0
WRITE ACKNOWLEDGEMENT
rw==0
DATA TFR
count != 0
count==0
READ ACKNOWLEDGEMENT 4
noack
ack
STOP
MASTER

SLAVE FSM


state
addr_0
not (count==1'b0)
not (start_det==1'b1)
start_det==1'b1
count==1'b0
start_det==1'b1
write_ack_0
rx_addr==slave_addr
int_reg_addr
not (count==3'd0)
count==3'd0
write_ack_2
read_start_1
not (start_det)
not (count_delay==2'd1)
sda==1'b0
start_det
count_delay==2'd1
not (rx_addr==slave_addr)
addr_1
not (count==3'd0)
data_rx
not (count==3'd0)
count==3'd0
count==3'd0
write_ack_1
write_ack_3
rx_addr==slave_addr
data_tx
not (count==3'd0)
not (rx_addr==slave_addr)
count==3'd0
read_ack
not (sda==1'b0)
stop



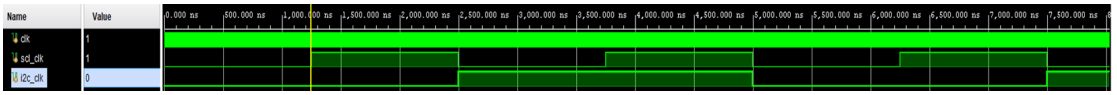
clocks used

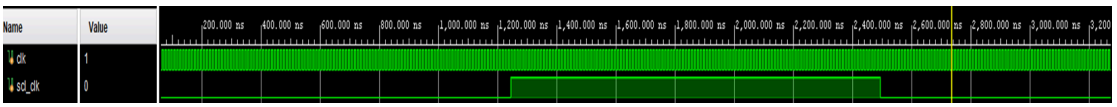
CLK=1GHz

SCL Frequency=400KHz

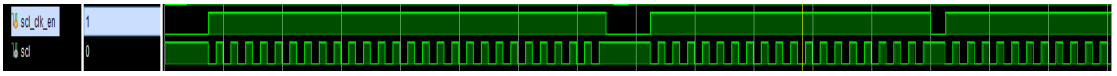
I2C Frequency=200KHz

The SCL clock is a derived clock at which slave functions, while I2C clock is a derived clock at which master functions.

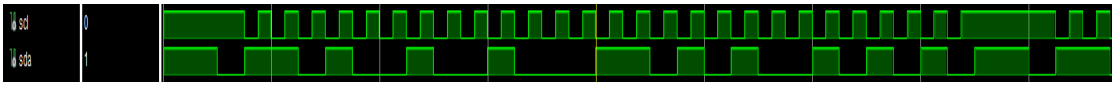




SCL clock gets enabled only when scl_enable==1

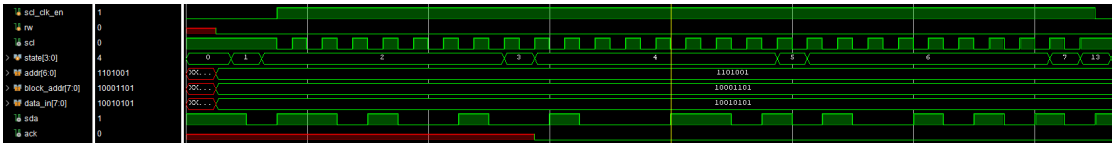


START condition: SCL==1 && SDA transitioning from 1 to 0

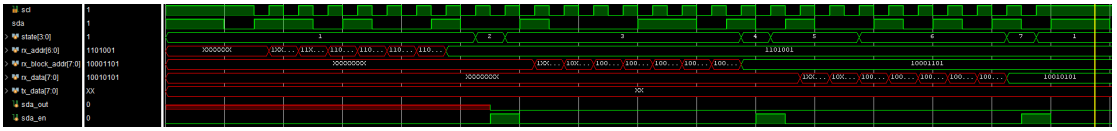


SLAVE ADDRESS, BLOCK ADDRESS, and Data are written over SDA from MASTER to SLAVE with acknowledgement bit = 0 if acknowledgement is received.

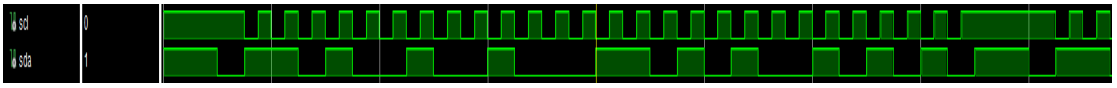
RW==0 => WRITE Operation



SLAVE ADDRESS, BLOCK ADDRESS, was read by the slave, and SLAVE sent an acknowledgement to MASTER. DATA written over SLAVE in the specified ADDRESS.

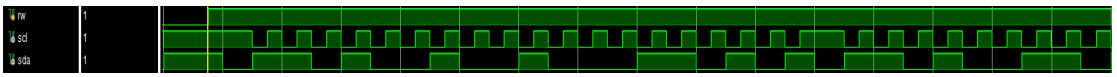


STOP condition: SCL==1 && SDA transitioning from 0 to 1



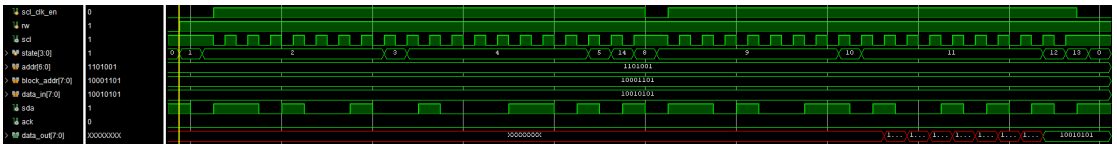
READ==1

START condition: SCL==1 && SDA transitioning from 1 to 0

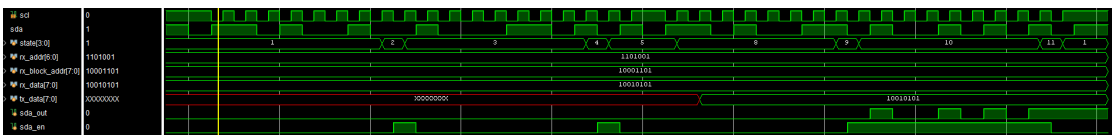


MASTER performs READ operation

READ data from specified SLAVE ADDRESS and BLOCK ADDRESS



SLAVE sends data to MASTER



STOP condition