I2C Address Translator

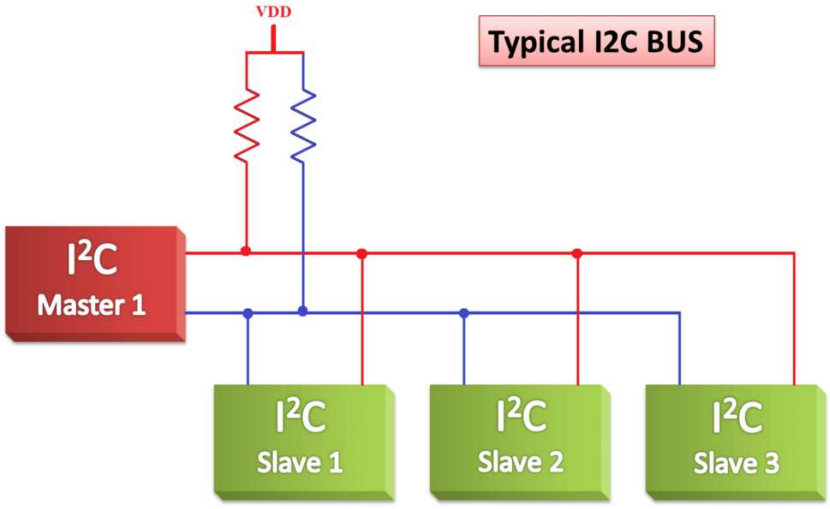
Project overview:

- I built the I2C protocol and address translator using Verilog on EDA Playground.
- It is a Synchronous two-wired communication protocol used for short distance communication between integrated circuits.

Features:

- Two-wired communication (SDA, SCL)
- Data lines are connected pulled-up resistor
- Support single-master and multi-slave architecture.
- Half-duplex communication
- Slave addressed by 7-bit unique address
- Operation frequency: 400 kHz
- Master-slave communication
    - Master generates clock and initiates communication
    - Slave responds when addressed

Architecture overview:



1. Components
    - Master 1
        - The master initiates communication on the I²C bus.
        - It Generates the clock signal (SCL) and controls START/STOP conditions.

- Slave1, Slave2, Slave3
  - Slaves respond to commands from the master.
  - Each slave has a unique address.
- Pull-up Resistors
  - Both SDA and SCL lines are connected to VDD through resistors.
  - I2c uses open drain configuration.
- I2C Address Translator
  - It allows a master to communicate with multiple slaves that might have conflicting or overlapping addresses**.** It modifies the address to avoid conflicts.
  - The translator sits between the master and multiple slave devices.
  - Continuously check for conflict on sda line, if two slave try to send data on sda line at a same time then generate conflict signal.
  - Then changes the slave address of one of them.

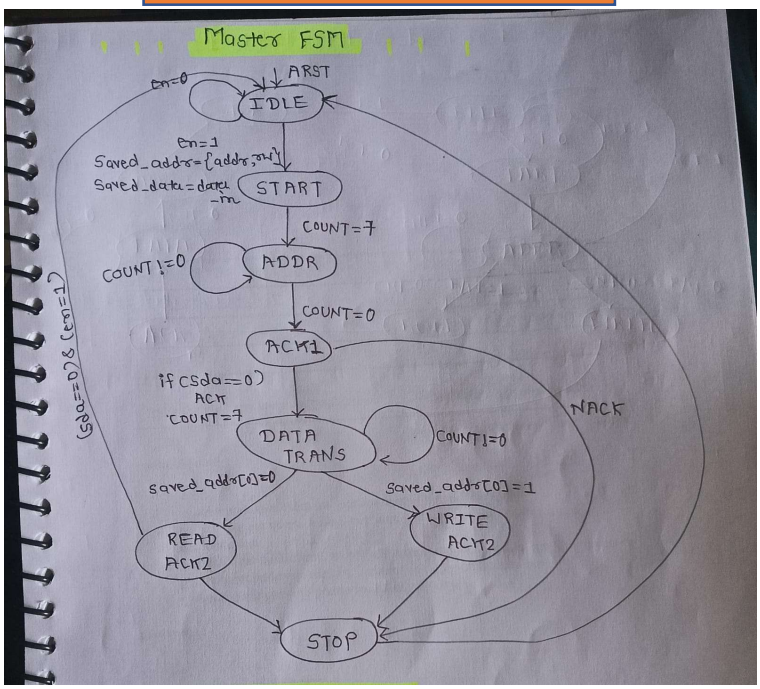
2. Bus Lines

- **SDA** (Serial Data Line)
  - Data can be driven by master as well as slave.
  - That's why it is bidirectional
- **SCL** (Serial Clock line)
  - Controlled by the master to synchronize data transfers

3. Working

1. Master initiates communication by sending a START bit on SDA while SCL is high
   - Sda = 0 means start
   - Sda = 1 means stop
2. Then master send 7-bit unique slave address to which it wants to communicate.
   - It transfers one bit at a time synchronized with SCL.
3. Master send read/write bit followed by slave address.
   - Sda = 1 means read
   - Sda = 0 means write
4. Slave acknowledges (ACK) if the address matches.
   - Sda = 0 means ACK
   - Sda = 1 means NACK
5. Master send data serially in write mode or master read data serially on SDA in read mode synchronized with SCL.
6. Master terminates the communication by sending stop bit.

I2C Master FSM



1. **IDLE**
   - Initial state after reset.
   - SDA and SCL are released so high due to pulled-up.
   - If en=1, master begins communication.
   - It saves the slave address and data to be transmitted.
   - en= 1 → next state → START

2. **START**
   - Master generates the Start bit: SDA goes low while SCL is high.
   - Initialize count=7 for address transmission.
   - Next state → ADDR

3. **ADDR**
   - Send 7-bit slave address and R/W bit on SDA
   - Each clock cycle, count decrements.
   - When count=0 , all address bits have been sent.
   - Next state → ACK1

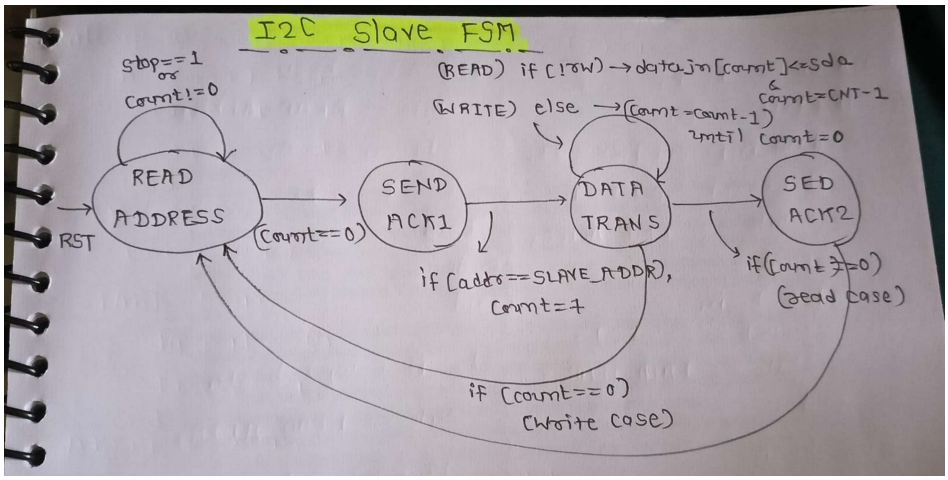
4. **ACK1**
   - Slave pulls SDA low means ACK (slave address matches)
   - if SDA=1 means NACK
   - if ACK received:
   - load count=7 for 8-bit data transfer
   - next state → DATA_TRANS

5. **DATA_TRANS**
    - If saved_addr[0] means R/W bit :
    - If R/W is 0 → write operation → send 8-bit data from master to slave
    - If R/W is 1 → read operation → receive 8-bit data from slave
    - Next state → WRITE_ACK2 (if write mode)
    - Next state → READ_ACK2 (if read mode)

6. **READ_ACK2 / WRITE_ACK2**
    - **WRITE_ACK2**: After sending data, master waits for ACK from slave.
    - **READ_ACK2**: After reading data, master sends ACK/NACK.
    - next state → STOP

7. **STOP**
    - Master generates Stop bit: SDA goes high while SCL is high.
    - next state → IDLE

I2C Slave FSM



1. **READ ADDRESS**
    - This is the initial/reset state.
    - The slave receives on SDA to capture **the** 7-bit address + R/W bit sent by the master.
    - A counter=7 then decrement shifts in each bit.
    - When all bits are received count=0:
    - If received address = slave's own address (slave_address), then → SEND ACK1
    - Next state → back to READ ADDRESS

2. **SEND ACK**
    - If the address matches, the slave pulls SDA low to send ACK to the master.
    - If R/W = 0 → Master wants to write→ next state DATA TRANS (to receive data).
    - If R/W = 1 → Master wants to read→ Slave will go to DATA TRANS (to send data).

3. **DATA TRANS**
    - If master writes then slave shift in 8bit data
    - Count decrement on each bit sent
    - If master read then slave puts data on SDA
    - Count decrement on each bit sent
    - When count=0 means all data bit sent then, next state → SEND ACK2

4. **SEND ACK2**
    - If WRITE mode**:** Slave ACKs the received byte.
    - If READ mode**:** Master sends ACK/NACK.
    - If stop bit occur then next state → READ ADDRESS

How address translator is implemented?

- I2C address translator is a module that placed between master and slave.
- Continuously check for conflict on sda line, if two slaves try to send data on sda line at a same time then generate conflict signal.
- Now it pauses the data transmission between master and slave, release the sda and scl lines.
- To generate conflict signal it checks sda_en signal of every slave available in system, if at any time two slave's sda_en becomes high then generates conflict.
- Once conflict generated it changes address of one of them because for smooth operation slave address should be unique.

Design Challenges faced

- Debugging the I2C master and slave design codes was challenging due to the complexity of the FSM and the bidirectional nature of the SDA line.
- Resolved the issue by declaring the SDA line as a tri-state signal, which enabled proper bidirectional communication.
- I have an idea of i2c address translator how it detects conflict on sda line and then changes slave address one of them.
- Although I gave my best effort, I was not able to fully implement the I²C address translator in this project.

References:

- https://in.images.search.yahoo.com/yhs/search;_ylt=AwrKBz5DlsJoORYEvQrnHgx.;_ylu=Y29sbwMEcG9zAzEEdnRpZAMEc2VjA3BpdnM-?p=i2c+protocol&type=type80160-3051786277&param1=3249001214&hsimp=yhs-002&hspart=sz&ei=UTF-8&fr=yhs-sz-002#id=12&iurl=https%3A%2F%2Febics.net%2Fwp-content%2Fuploads%2F2023%2F06%2Fimage-36-1024x683.png&action=click