FPGA Module for I²C Virtual Address Translation

Architecture Overview

- Required to design an FPGA module to handle cases where multiple slaves share the same default I²C address.
- In such cases, the master cannot determine which slave to communicate with.
- The FPGA acts as an interpreter by assigning virtual addresses to each device.

Example:

- Both devices originally have the default address 48H.
- FPGA assigns:
  - Device 1 → 49H
  - Device 2 → 48H

The master communicates with the FPGA, which exposes both addresses (48H and 49H) for easy identification.

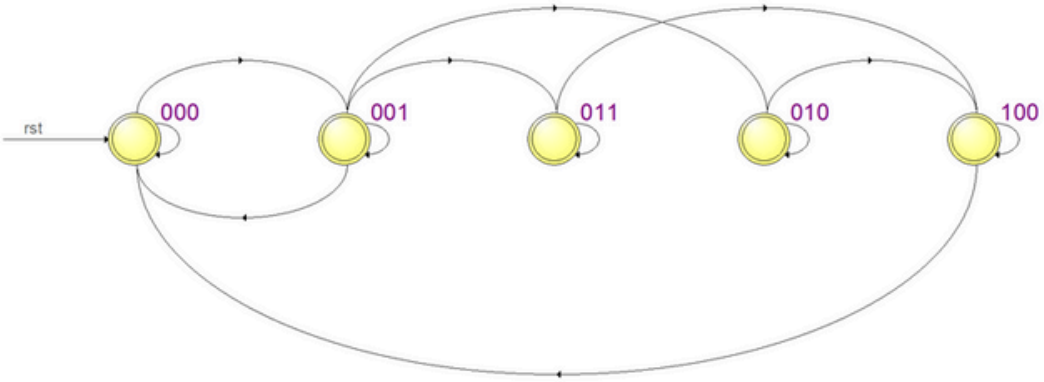
Functionality:

- FPGA behaves as a slave towards the master.
- FPGA behaves as a master towards the physical slave devices.

The design is implemented using a Finite State Machine (FSM) to properly distribute each process.

FSM Logic Explanation

The FSM consists of **5 states:**



State-0 (Idle)

- All registers and pins are reset to zero.
- When scl_in is high and sda_in transitions from 1 → 0, the FSM transitions to State-1.

State-1 (Address Translation)

- On each positive edge of scl_in, the address is fetched from sda_in and stored in an 8-bit address_shift_reg.
- Upper 7 bits represent the address, while the LSB determines read (1) or write (0).
- If the fetched address matches:
  - 49H → enable = 1 → communication is directed to Device 1.
  - 48H → enable = 0 → communication is directed to Device 2.
- If neither matches, FSM returns to Idle (State-0).
- Based on the LSB (read/write flag):
  - 0 → transition to State-2 (Write to Slave).
  - 1 → transition to State-3 (Read from Slave).

State-2 (Write to Slave)

- On each positive edge of scl_in, data bits are fetched from sda_in into data_shift_reg.
- After 8 bits are received, data is transferred to device_tx_reg.
- Then, on each scl_in positive edge, the bits are transmitted sequentially to the slave via sda_dev_out.
- Acknowledgment check:
  - If sda_dev_in == 0, continue transfer.
  - Otherwise, FSM moves to State-4 (Stop).

State-3 (Read from Slave)

- On each positive edge of scl_dev, data is fetched from sda_dev_in into device_rx_reg.
- After 8 bits are received, the data is moved to data_shift_reg.
- Then, on each scl_dev positive edge, the bits are transmitted sequentially to the master via sda_out.
- Acknowledgment check:
  - If sda_in == 0, continue reading.
  - Otherwise, FSM transitions to State-4 (Stop).

State-4 (Stop Condition Detection)

- When scl_in stays high and sda_in transitions from 0 → 1, the FSM resets back to Idle (State-0).

Method of Address Translation

- The FPGA fetches the address from the master and compares it with predefined virtual addresses:
  - virtual_address1 = 49H
  - virtual_address2 = 48H
- If the fetched address matches:
  - 49H → enable = 1 → route signals to sda_out1, scl_out1, sda_dev_out1.
  - 48H → enable = 0 → route signals to sda_out2, scl_out2, sda_dev_out2.

This ensures each slave device is accessed using its unique virtual address.

Design Challenges Faced

1. **Understanding protocol behavior**
   - While the theory is straightforward, implementing protocol behavior in hardware exposed many hidden complexities.
2. **Clock edge detection**
   - The design primarily runs on the posedge of clk, but some states required updates based on the posedge of scl_in or scl_dev, making detection tricky.
3. **Shift register management**
   - Using shift registers for transmission/reception ensured correct sequencing but added latency.
   - Found that storing data first and then transmitting improves stability.
   - Transmitting as we store would increase throughput—but requires redesigning State-2 and State-3.
4. **Testbench design**
   - A proper, well-structured testbench is essential for verifying I2C protocol functionality, since a random brute-force approach cannot reliably capture the required signal sequencing and data flow.
   - A fully descriptive testbench was created to verify the system.
   - Verification steps:
     - Provide address 49H with LSB=1 → Receive from Device 1 and forward to master.
     - Provide address 48H with LSB=0 → Transmit to Device 2.
   - Ensured correct flow and maintained a data rate of 100 kbps by using a 200 kHz base clock, generating each required posedge signal state by state.
   - Modified the testbench using Gemini to save all values in table form in a log file.

Limitations and Future Scope

- Current design supports only two devices.
- With advanced techniques (e.g., user input + dynamic programming), the FPGA could scale to support any number of slaves.
- This requires deeper exploration of resource optimization and protocol flexibility.

Open Questions / Doubts

- How can we further optimize resource utilization and timing constraints?
- What techniques or common practices are used in industry to handle such constraints?
- Can Data Structures and Algorithms (DSA) concepts be applied here to improve scalability and throughput?

Acknowledgment

Special thanks to the Vicharak recruitment team for providing me with the opportunity to contribute in this domain.
