**FPGA-Based I²C Address Translator**

***1. Abstract***

The I²C Address Translator is a digital communication bridge designed to resolve address conflicts between multiple I²C slave devices that share the same physical address.  
In a standard I²C system, multiple devices with identical addresses cannot coexist on the same bus.  
This project solves that problem by introducing a translation layer implemented on an FPGA.

The translator functions as:

* An I²C Slave to the external master (such as a microcontroller or processor).
* An I²C Master to two target slave devices.

It maps unique virtual addresses (0x49, 0x4A) used by the external master to a common physical address (0x48) used by the actual I²C devices.  
This ensures transparent communication for both read and write operations.

The design is written in Verilog HDL, simulated in Xilinx Vivado, and can be implemented on any FPGA development board that supports I²C-compatible voltage levels.

***2. Introduction***

The I²C (Inter-Integrated Circuit) protocol is a widely used two-wire serial communication standard that supports multiple masters and slaves on a shared bus.  
However, a limitation arises when multiple devices with identical addresses must be connected simultaneously.

The I²C Address Translator eliminates this limitation by introducing an FPGA-based layer that allows the main master to access two identical devices with distinct virtual addresses.

**Key Applications**

* Multi-sensor networks using identical ICs (e.g., temperature or ADC sensors).
* Redundant systems needing mirrored devices.
* Industrial automation setups with shared bus architectures.

***3. System Overview***

**Functional Description**

The translator monitors I²C transactions initiated by the external master.  
It decodes the 7-bit address and determines which target to access, translating the virtual address internally before forwarding the transaction.

| **Virtual Address** | **Physical Address** | **Target** |
| --- | --- | --- |
| 0x49 | 0x48 | Target 1 |
| 0x4A | 0x48 | Target 2 |

This allows two I²C devices with identical physical addresses to operate independently on the same master bus.

***4. System Architecture***

A diagram of a diagram

AI-generated content may be incorrect.A diagram of a computer system

AI-generated content may be incorrect.

***5. Module I/O Description***

| **Signal** | **Direction** | **Description** |
| --- | --- | --- |
| clk | Input | System clock (typically 50 MHz). |
| rst | Input | Active-high synchronous reset. |
| scl\_in | Input | I²C clock from external master. |
| sda\_in | Inout | I²C data from external master. |
| scl\_out1, sda\_out1 | Output/Inout | Target device #1 interface. |
| scl\_out2, sda\_out2 | Output/Inout | Target device #2 interface. |

***6. Operation Phases***

**A. Write Operation (External → Target)**

1. Master sends START + [0x49 or 0x4A, Write=0].
2. Translator acknowledges and identifies target.
3. Translator triggers internal master FSM.
4. Master FSM sends START + [0x48, Write=0] on target bus.
5. Data forwarded from external master to selected target.
6. Transaction ends with STOP.

**B. Read Operation (External ← Target)**

1. Master sends START + [0x49 or 0x4A, Read=1].
2. Translator acknowledges and determines target.
3. Translator initiates a read on the target bus (0x48, Read=1).
4. Data received from target and relayed to external master.
5. Acknowledgment and STOP handled accordingly.

***7. FSM Descriptions***

**Slave FSM**

| **State** | **Description** |
| --- | --- |
| IDLE | Wait for START condition. |
| ADDR | Capture address and R/W bit. |
| ACK | Acknowledge valid address. |
| WRITE | Receive and store data for forwarding. |
| READ | Send read data from target to master. |
| STOP | Reset FSM on STOP condition. |

**Master FSM**

| **State** | **Description** |
| --- | --- |
| IDLE | Wait for trigger from slave FSM. |
| START | Generate START on target bus. |
| ADDR | Send 0x48 with R/W bit. |
| ACK | Wait for target acknowledgment. |
| WRITE | Transmit data to target. |
| READ | Receive data from target. |
| STOP | End communication. |

***8. SDA Line Tri-State Handling***

The I²C SDA line uses **open-drain** logic, meaning it can only be driven low or released (high-impedance).  
To avoid contention, tri-state buffers are used:

assign sda\_in = sda\_in\_en ? sda\_in\_out : 1'bz;

assign sda\_out1 = sda\_out1\_en ? sda\_out1\_out : 1'bz;

assign sda\_out2 = sda\_out2\_en ? sda\_out2\_out : 1'bz;

***9. Clock Division Logic***

A clock divider generates the standard I²C 100 kHz clock from a 50 MHz system clock:

if (clk\_div == 250) scl\_gen = ~scl\_gen;

This clock is used to control the FSM for target-side communication.

***10. Key Features***

* Dual-role operation — Acts as both I²C Slave and Master.
* Supports read/write translation between two identical slaves.
* Transparent virtual-to-physical address mapping.
* Proper open-drain and tri-state line handling.
* Modular FSM architecture for improved clarity.
* Accurate I²C timing through clock division.

***11. Design Verification and Simulation***

**Tool:** Xilinx Vivado 2023.x  
**Language:** Verilog HDL

**Verification Performed**

* START and STOP condition detection.
* Address recognition and translation verification.
* Read and Write transaction testing.
* ACK/NACK sequence validation.
* SDA line contention testing and waveform verification.

***12. Testbench Development Status***

I have successfully completed the design code and achieved RTL implementation without any errors.  
Currently, I am developing the testbench to verify and validate the design functionality.

Although a portion of the testbench is written, it is under active debugging and refinement.  
To maintain the report’s quality and accuracy, incomplete or error-prone testbench code has not been included here.

My current focus is on achieving a clear, verified, and simulation-proven testbench through continuous testing and correction.  
With additional time, I am confident of completing the testbench successfully to achieve full design verification.

***13. Future Scope***

* Extend the translator to support multi-byte data transfers.
* Integrate configurable virtual address mapping via registers.
* Add auto-detection of connected I²C devices.
* Develop a universal testbench for multi-target simulation and waveform verification.
* Optimize the design for higher-speed I²C modes (400 kHz / 1 MHz).

***14. Conclusion***

The FPGA-Based Dual-Level I²C Address Translator enables communication between a single master and multiple identical devices sharing the same physical address.  
By introducing virtual address mapping, dual FSM control, and protocol-level translation, the design overcomes a fundamental limitation of the I²C protocol.

This project demonstrates skills in Verilog RTL design, FSM-based architecture, I²C protocol understanding, and timing synchronization — essential competencies for modern FPGA-based embedded communication systems.