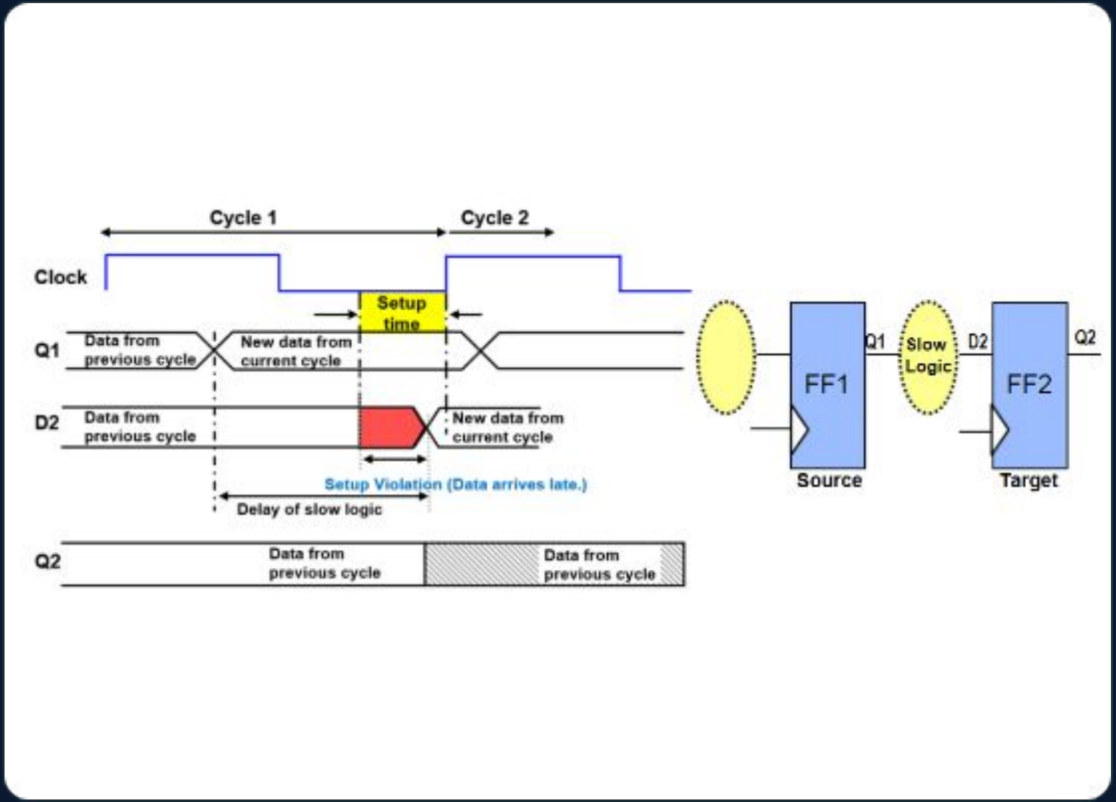
16-bit Parallel Communication Link Between FPGAs

A Robust Asynchronous CDC Bridge

RAQEEB | SAKSHAM | AAKARSH | SOHAIL

The Core Problem: Metastability

- When data crosses asynchronous clock domains (like `clk_A` to `clk_B`), the receiving flip-flop can violate its setup/hold times.
- This can cause the flop's output to enter a "metastable" state—an unknown voltage level that is neither 0 nor 1.
- This state eventually resolves, but it takes an unknown amount of time, leading to unpredictable behavior, data corruption, and system failure.
- Our project connects two FPGAs, each with its *own* clock. This is a classic Clock Domain Crossing (CDC) problem.



Project Goal & Our Solution

Project Goal

To design a **lossless** 16-bit parallel link that *guarantees* safe data transfer between two asynchronous clock domains.
The system must be robust, reliable, and prevent any data corruption from metastability.

Our Solution

A **4-Phase `VALID/READY` Handshake Protocol**.

This provides "backpressure." The Transmitter (master) *asks* to send data (`VALID`). The Receiver (slave) *agrees* when it's ready (`READY`). This is the core optimization for **reliability**.

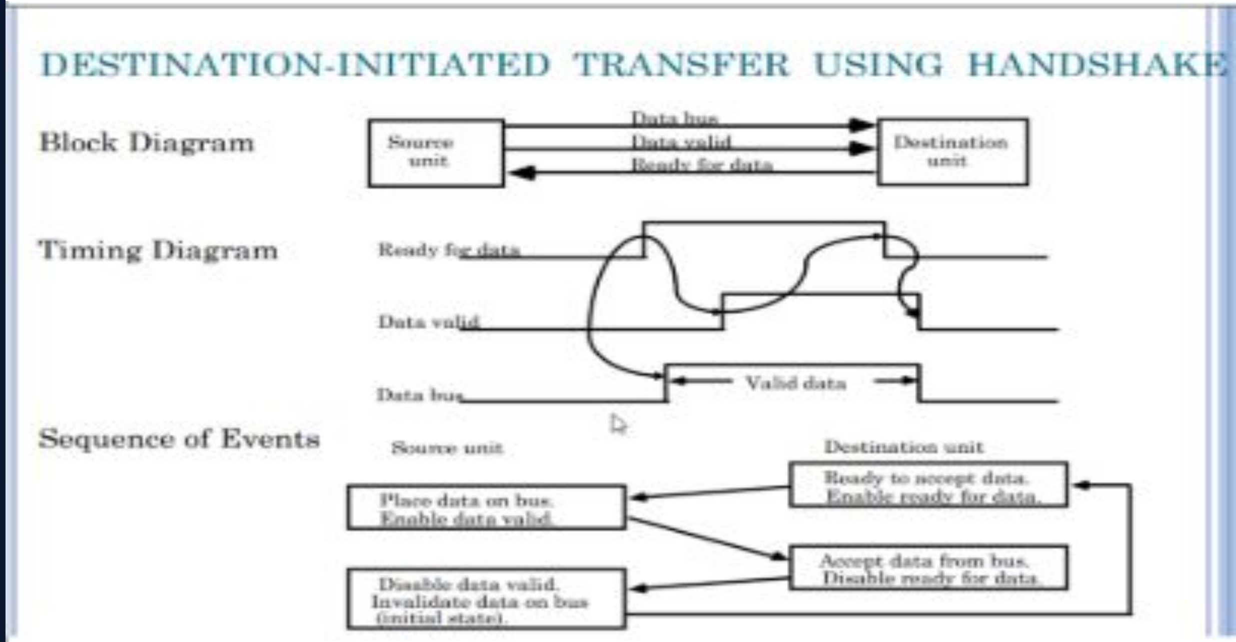
System Architecture

The system is split into two independent domains, isolated from each other:

- **Transmitter (TX) Domain**
  Operates on `clk_tx`. It contains a `FIFO` to buffer outgoing data and an FSM to manage the handshake.

- **Receiver (RX) Domain**
  Operates on `clk_rx`. It contains a `FIFO` to buffer incoming data and an FSM to acknowledge it.

The domains are only linked by the asynchronous data bus and the two 1-bit handshake signals.



The Building Blocks

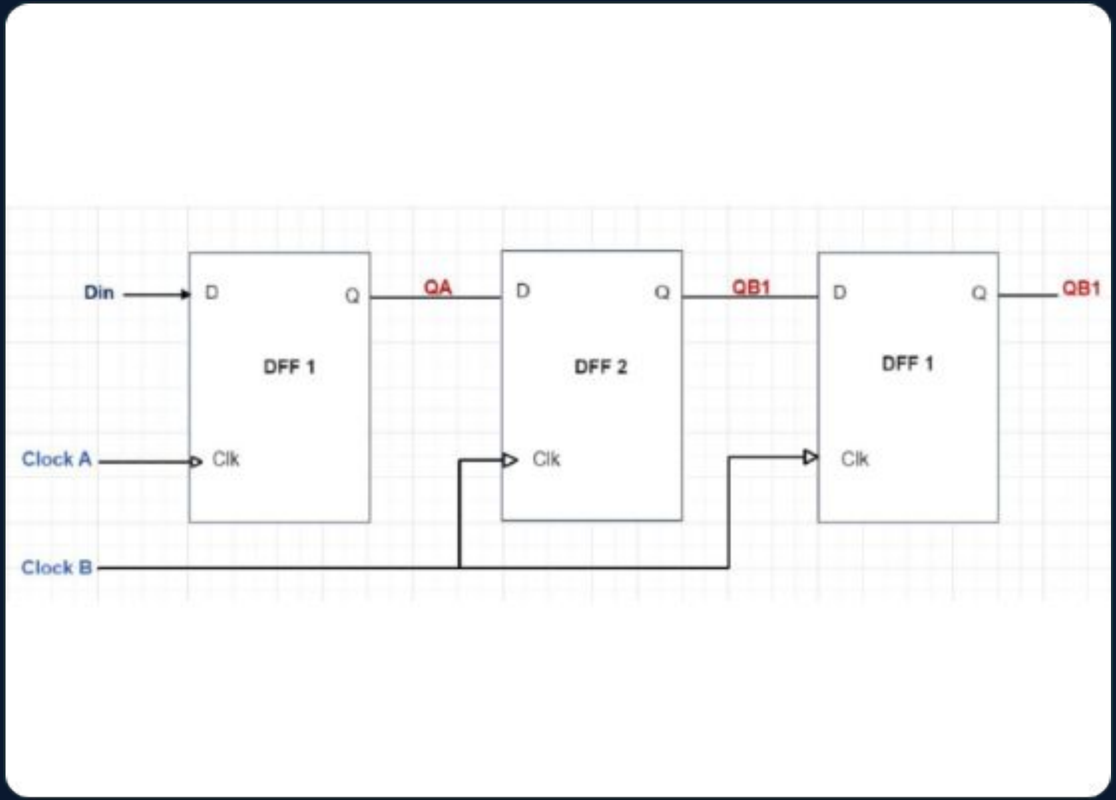
(The Core Verilog Modules)

Module 1: The 2-FF Synchronizer (`cdc_sync`)

Purpose: The Stabilizer

To safely pass a 1-bit signal (`valid`, `ready`) from one clock domain to another.

- **How it works:** A 2-Flip-Flop pipeline.
- The 1st flop (`s1_sync_reg`) samples the async signal. It is allowed to go metastable.
- The 2nd flop (`s2_sync_reg`) samples the 1st flop one cycle later. This gives the 1st flop time to resolve to a stable 0 or 1.
- This makes the Mean Time Between Failure (MTBF) astronomically high.



Module 2: The Synchronous FIFO (`FIFO`)

Purpose: The Buffer

Provides "elastic" buffering *within* each clock domain.

- **Transmitter FIFO:** Allows the user to "burst" data in. The FSM then sends it safely, one-by-one.
- **Receiver FIFO:** Stores the received data, allowing the user to read it out at their own pace.
- Uses 5-bit pointers for reliable `full` and `empty` detection.

Module 3: `transmitter` FSM (The Master)

State: IDLE

Waits for data to appear in its local FIFO (`!fifo_empty`). When data is present, it reads one word and moves to the next state.

State: WAIT_ACK

Places the data on `parallel_data_out`. Asserts `parallel_valid_out` HIGH, signaling to the receiver. Waits for `ready_sync` to go HIGH.

State: WAIT_READY_LOW

The receiver has acknowledged. De-asserts `parallel_valid_out` LOW. Waits for `ready_sync` to go LOW to complete the handshake.

Module 4: `Receiver` FSM (The Slave)

State: IDLE

Waits for `valid_sync` to go HIGH. Critically, it also checks if its own local FIFO is *not* full (`!fifo_full`).

Action: Acknowledge

If `valid_sync` is HIGH and it has space, it latches the data from `parallel_data_in` and asserts `parallel_ready_out` HIGH.

State: WAIT_VALID_LOW

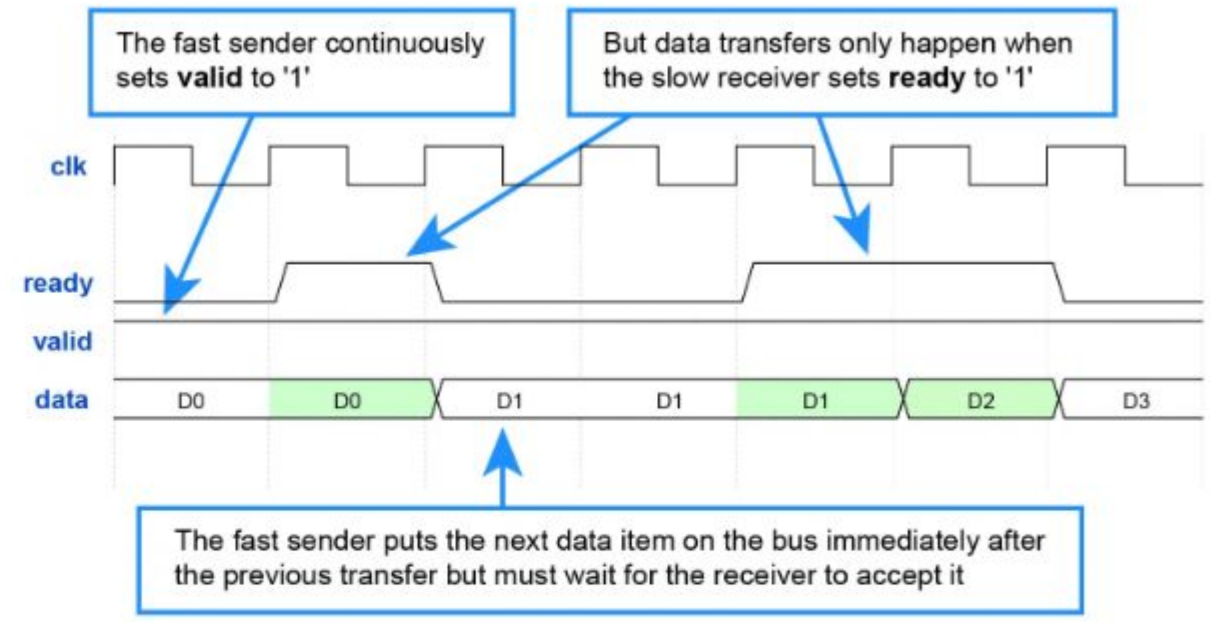
Waits for `valid_sync` to go LOW. This confirms the transmitter saw the acknowledgment. It then writes the latched data to its FIFO and de-asserts `ready`.

The 4-Phase Handshake in Action

This timing diagram shows the complete, lossless transfer of one word.

- **1. TX Asserts VALID:** Transmitter puts data on the bus and raises `valid`.
- **2. RX Asserts READY:** Receiver sees `valid`, latches the data, and raises `ready`.
- **3. TX De-asserts VALID:** Transmitter sees `ready`, knows the data was taken, and drops `valid`.
- **4. RX De-asserts READY:** Receiver sees `valid` is low, drops `ready`, and writes to its FIFO.

The cycle is now complete and ready for the next transfer.



Analysis: The Core Trade-Off

Pro: Reliability (Lossless)

Our design is **robust and guaranteed lossless**. The `READY` signal provides essential "backpressure."

Data is *never* lost if the receiver is busy or its FIFO is full. This is the correct, safe, professional solution for critical data.

Con: Latency (Slow)

The 4-phase handshake is **slow on a per-word basis**. We must pay a multi-cycle latency penalty for the *entire round-trip* acknowledgment.

This includes propagation delay and 2-3 cycles of synchronization on *both* sides of the bridge.

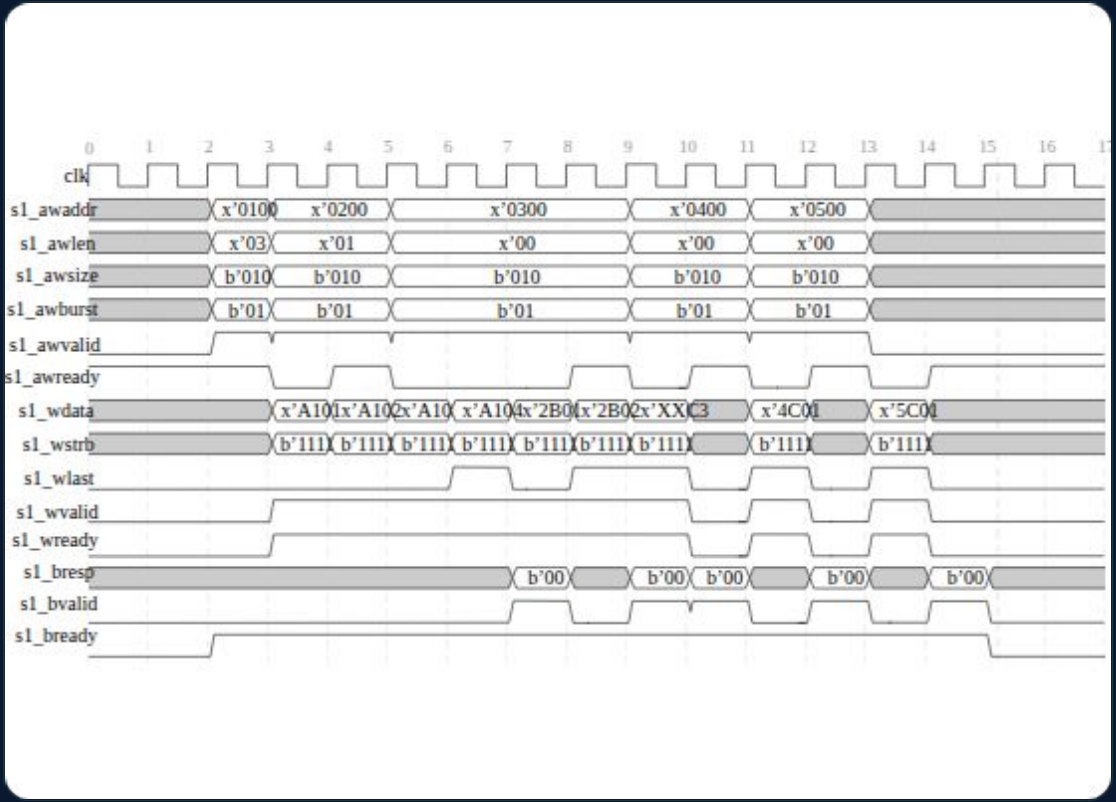
Future Optimization: Burst Transfers

How to Increase Throughput?

Modify the protocol to be burst-oriented, similar to AXI-Stream.

- `VALID` stays HIGH for an entire *burst* of data (e.g., 64 words).
- `READY` stays HIGH as long as the Receiver's FIFO has space.
- Data is transferred on **every single clock cycle** as long as both signals are HIGH.

Result: We pay the latency penalty **once per burst**, not once per word, dramatically increasing the data rate.



Conclusion

We successfully designed a **robust, lossless** 16-bit CDC bridge.
Solved the critical problem of metastability using **2-FF synchronizers**.

Guaranteed data integrity with a **4-phase `VALID/READY` handshake**.

Identified a clear path to high-throughput via **burst transfers**.

Thank You

RAQEEB | SAKSHAM | AAKARSH | SOHAIL