



A Multi-Source AXI4-Stream Arbiter with Priority-Based Flow Control for Heterogeneous AI Systems

Gaurav Nanaware¹, Ashwini Kulkarni¹, and
Varada Inamdar¹

Department of Electronics and Telecommunication Engineering, COEP Technological
University, Pune, India
nanawaregb24.extc@coeptech.ac.in

Abstract. While data movement latency is still a big problem for AI with heterogeneous systems. To overcome this, we are introducing a Multi-source AXI4-Stream interconnect with an input arbitration scheme Implement stage and asynchronous FIFO buffers for robust data flow by Between different processing elements. Implemented on an Intel MAX 10 FPGA and synthesized device with TSMC 100MHz 65nm technology node. the design exhibits high performance, deterministic communication. At a FPGA clock frequency of 50MHz the experimental results yield a peak Can achieve a throughput of 400 MB/s with just 4% of the logic resources. Compact silicon readiness with ASIC synthesis. footprint of 109,124 μm^2 and a setup slack of 8,408ps at 100 MHz. This Real-time depends on resource-efficient architecture as a key enabler. By ensuring that applications, especially AI linkages within remote surgeries, run efficiently. For mission-critical continuous, high precision data transmission is required.

Keywords: AXI4-Stream · Multi-source Arbiter · FPGA · ASIC · FIFO · Flow Control

1 Introduction

As modern computer chips (SoCs) get more complex with They require a common method of communication with many processors and memory units. To communicate with one another. The AXI protocol was developed by ARM in 2003. To make these connections quick and efficient[1, 20]. AXI3 travels five times as well as this is because it utilizes 5 separate channels (or lanes) To move information: three for writing and two for reading. It writes using the Write Address (AW) channel and says using the Write Data (AD) channel. Write Data (W) channel to send the data where. information, and to confirm (B) information, it arrived safely. It uses the Read Address (AR) for reading. channel to ask for data and the Read Data (R) channel to get it back. These lanes are in isolation, which means that the chip can read and write simultaneously without any traffic

jams. It uses simple "ready" and "valid" signals to insure that data is only sent along when it is ready and valid. Sends info in rapid transmission when both sides are ready groups called bursts[7]. While AXI3 worked well for video and memory, it isn't enough for modern AI systems, AXI4 was eventually introduced to handle even more data. AXI4 allows for much bigger bursts (up to 256 instead of just 16) and removes some confusing rules to make the actual hardware design simpler and faster.

In 2010, ARM released the AMBA 4 specification to provide specialized protocols for different hardware needs. AXI4 was introduced to improve speed by supporting much larger bursts of data. It is used for heavy tasks like moving high-definition video or connecting a processor to fast memory (RAM). Along with it, AXI4-Lite was released as a simple version for low-speed tasks. It saves space on the chip by removing complex features and is used for simple jobs, like setting up switches or controlling basic parts like a UART or simple sensors[16], [19]. AXI4-Stream was developed for high-speed, point-to-point data movement[13]. The main benefit of this protocol is that it moves data with almost no delay[4]. AXI4-Stream is built for just one to one connection, it does not have a built-in way to handle many different sources at the same time. This is a major problem when different sources send data at different speeds. Without a way to manage these mixed requests, a slow source could block a fast one, or important data could be lost. This generates a need for an A layer of the arbitration process that makes the decision about which source to go first. in managing buffer space to ensure stability of the system for Some critical tasks such as remote surgery. Earlier designs, such as the one by Prajwal Angadi, Sanket Venkatesh Shenvi, Sathvik Lokesh from Dayananda College of Engineering[5] was concerned with the data transfer from a single master. To two slaves with a simple round robin approach. While that It is an appropriate method for simple data exchange among chips, Does not address the issue of multiple slow sensors attempting to to transmit information simultaneously to one fast processor. Our work is different in that we use a many-to-one arbiter that applies priority rules and special buffers to prevent data being lost from lost. In this way, our design will be able to deal with the strict Requires time and safety needs for remote surgery, Much more reliable than normal data streaming procedures.

2 System Architecture and Operation

2.1 System Architecture Overview

The multi-source AXI4-Stream architecture is designed to manage data flow from multiple sources to a shared interface using priority-based control, as shown in Figure 1. The system includes four main parts: input sources, a buffering and arbitration block, an AXI master block, and an AXI slave block. Three independent sources send requests for access to the master, and The arbitration logic uses a fixed priority scheme with a dynamic override to ensure zero data loss[10]. It continuously monitors "nearly full" flags from the FIFOs; if any

buffer nears capacity, the arbiter immediately grants it bus access to prevent an overflow. Arbitration scheme usually work combination of round robin algorithm and almost signal high combination. The master block then manages the AXI4-Stream signals including TVALID, TDATA (64-bit), TID, TLAST, and C. Slave Controller TSTRB/TKEEP to send data to the slave[13]. If the slave is not ready to accept new data, it pulls TREADY low. To prevent any data loss, each buffer sends a warning signal to the arbiter whenever it is nearly full. The arbiter then quickly clears the data from these crowded buffers to make room for new incoming information. This simple method ensures the system stays stable and handles all data sources without any overflows or stops.

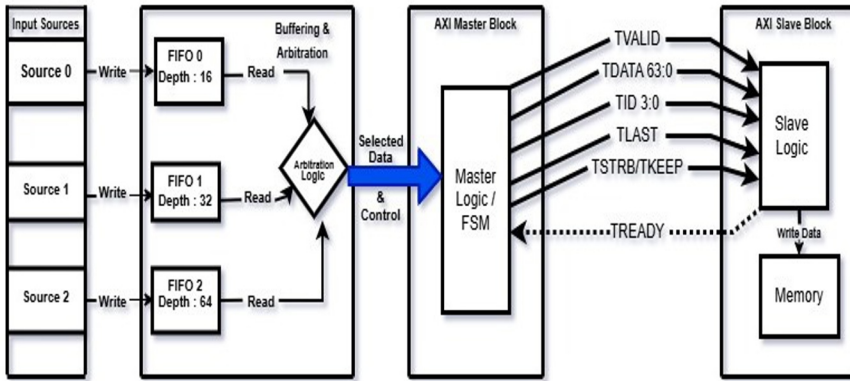


Fig. 1: AXI stream system architecture.

2.2 Master Controller Finite State Machine

The master controller finite state machine (FSM) consists of three states: *IDLE*, *TRANSMIT*, and *WAIT READY*, as shown in Fig. 2. The FSM manages the TVALID/TREADY handshake and controls AXI4-Stream data transfer.

The master controller finite state machine (FSM) consists of three states: *IDLE*, *TRANSMIT*, and *WAIT READY*, as shown in Fig. 2. The FSM manages the TVALID/TREADY handshake and controls AXI4-Stream data transfer.

Upon reset, the controller remains in the *IDLE* state until a transfer request is received. In the *TRANSMIT* state, data is being sent onto the bus if it is valid. TVALID is asserted. When the slave de-asserts TREADY, the FSM goes to the *WAIT READY* state, in which the output data and control signals are maintained until the slave becomes ready again. This mechanism ensures that data does not get corrupted and ensures reliability of the data. Transfer under back pressure. After a successful transaction, the FSM either returns to the *IDLE* state or continues with the next transfer during burst operations.

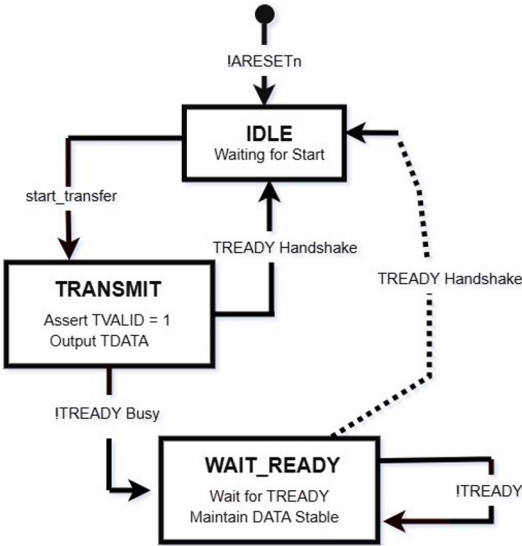


Fig. 2: Master controller finite state machine.

2.3 Slave Controller

The AXI4-Stream Slave controller architecture is used to manage From the master, a data capturing process is performed, and the data is then stored locally. The memory is as in Figure 3. Upon receiving The controller receives the data and stores it, The direct-to-integrated FIFO memory is handled by direct to memory interface [9]. If the memory is almost full or if the system is busy the slave Sends a signal to mean that it cannot accept more data. The data is then retrieved via the When an internal flag is asserted to indicate that, it is output to There is valid information available for transaction.

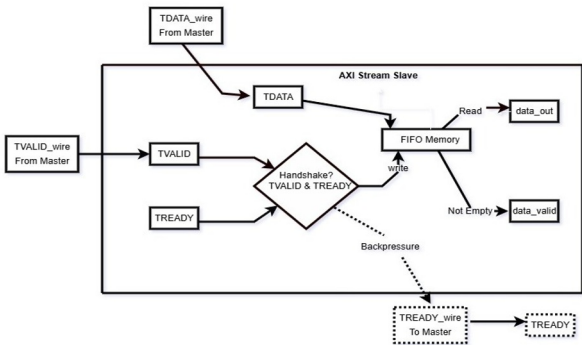


Fig. 3: AXI4-Stream slave controller architecture.

2.4 Configurable FIFO Architecture

The configurable FIFO architecture is designed to manage the data-rate mismatch. These are all situated between slow peripherals and the high-speed AXI4-Stream system, as in Fig. 4. The module is a basic component of the buffering stage and comprises of three, secondary functional blocks: memory array initializers, memory array access logic, control logic and pointer access logic status flag generation [9].

Control Logic and Pointers The write and read operations are controlled in this section. address pointers. Upon receipt of data, the write pointer will be updated to store Move data to the adjacent memory location which is next in line. Also, if the arbiter asks for data, the read pointer is moved to fetch the stored data. This dual-pointer Mechanism is such that both simultaneous and independent read/write operations can be performed, In this way continuous data flow is maintained.

Memory Array Storage This block is used to store the data physically. where the data coming in is temporarily stored before it is passed on to the master inter- face. The buffering mechanism ensures data integrity when the data is temporarily unavailable.

Status Flag Generation The module produces the status flag signals: Full, Empty and count on the fly on the FIFO. These signals provide Feed-back to arbiter - this allows for efficient flow control without overflow. and underflow conditions

FIFO Depth Calculation A transfer of one player to another in back-to-back fashion requires: The relationship between the peripheral write and read operations is used to determine the FIFO depth. The frequency and the AXI4-Stream clock frequency ($f_{\text{AXIS}} = 50 \text{ MHz}$)

$$\text{Depth} = N \times \left(1 - \frac{f_{\text{AXIS}}}{f_{\text{wr}}} \right) \quad (1)$$

If protocol overhead or idle intervals (I_{wr} , I_{rd}) are present between data operations The effective frequency ratio should be brought in the case:

$$\text{Depth} = N \times \left[1 - \left(\frac{1 + I_{\text{wr}}}{1 + I_{\text{rd}}} \right) \times \frac{f_{\text{AXIS}}}{f_{\text{wr}}} \right] \quad (2)$$

FIFOs also need to meet a requirement imposed by the arbiter – inherent latency. The maximum stall tolerance (T_{stall}). This guarantees incoming peripheral data can be received. Wait for access to the AXI4-Stream bus, while being buffered:

$$T_{\text{stall}} = \frac{\text{FIFO Depth}}{f_{\text{wr}}^{\text{eff}}} \quad (3)$$

where:

- N is the burst size
- The clock frequency of the AXI4-Stream is f_{AXIS}
- f_{wr} is the frequency of the source input (clock frequency)
- I_{wr} and I_{rd} are idle time.
- The stall tolerance is the max value of T_{stall}

The following calculated FIFO configurations for the heterogeneous interfaces are summarized below:

- **FIFO 0 (I²C @ 5 kbps):** Equation (2) takes protocol overhead into account ($f_{\text{wr}}^{\text{eff}} \approx 4.0$ kbps for 8-byte bursts). A 256 mS FIFO depth is achieved with a 128-byte depth. Substantial arbitration flexibility provided with buffer window.
- **FIFO 1 (UART @ 9600 baud):** Using a 10-bit frame structure, the 256-Approximately 266 ms stall-tolerance is offered by byte FIFO.
- **FIFO 2 (SPI @ 25 MHz):** Equation (1) indicates that a 512-byte FIFO to achieve $163.84 \mu\text{s}$ safety margin requires a depth of this microsecond. The scale buffer window is essential to avoid overflow when there are transient arbitration delays.

The results validate the choice of using heterogeneous FIFO depths and demonstrate how the suggested architecture can effectively support variety of peripheral devices Interfaces by matching buffer resources to real-time system constraints.

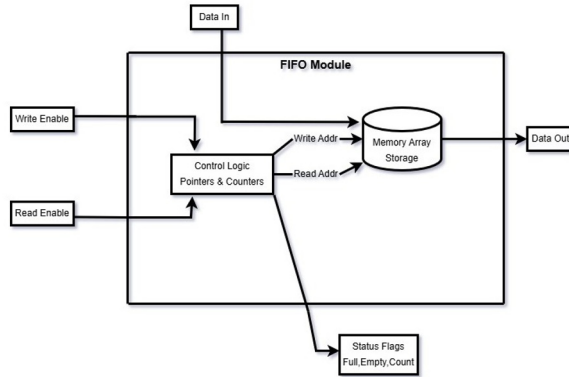


Fig. 4: Configurable FIFO architecture.

3 Results and Analysis

The system was designed in Verilog language and verified by SystemVerilog based. The Questa Advanced Simulator's testbench. Physical implementation

Test 3: Complete Transaction. The following is a waveform of a complete transaction As seen in Fig. 7, the total time required to perform the operation is from the initiation to the completion. If it is a valid data, then to the Source 0, the arbiter provides the AXI4-Stream interface signals TVALID and TREADY to start the transfer. The control signals TSTRB and TKEEP are set to 8'hFF which means that all Valid of 64 bits of data. This information is then properly sent to the data_out port, Confirming the integrity of the data in the system, from the source to the final output.

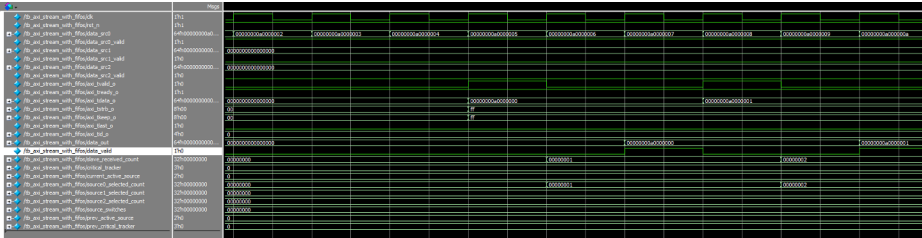


Fig. 7: Simulation waveform: complete AXI4-Stream transaction.

3.2 Synthesis Results

FPGA Implementation (Intel MAX 10). The proposed architecture was realised in an Intel MAX 10 (10M50DAF484C7G) FPGA featuring a 64-bit data path. With a clock frequency (f) of 50 MHz, the design had an experimental throughput (T) of There are 400 MB/s of throughput, which is computed as follows. relationship:

$$T = f \times \frac{W}{8}$$
(4)

where W represents the data bus width in bits. As explained in Table 1, the The design is very space efficient, only (hardly) requires. 1,848 LUTs (4%) and 1,603 flip-flops. Notably, the design consumes just 12,288 memory bits (<1%) Having no use of special block RAM (BRAM), as storage requirements are handled using distributed logic.

Table 1: Total resource utilization for FPGA implementation (Intel MAX 10).

Resource	Used / Available	Utilization
LUTs	1,848 / 49,760	4%
Flip-Flops	1,603	—
Memory Bits	12,288 / 1,677,312	<1%
BRAM	0	0

The power analysis is summarized below (Table 2), It shows a total dissipation of 105.98 mW. The power The current power usage is static at 96.18 mW with The dynamic power is still at 9.80 mW. These results confirm that the architecture is light weight, The high performance solution for deterministic data movement,

More suitable for low latency, resource constrained applications. AI-powered remote surgery connections are the kind of applications the aforementioned has enabled.

Table 2: Power analysis of the proposed design.

Parameter	Value
Total Power	105.98 mW
Static Power	96.18 mW
Dynamic Power	9.80 mW

ASIC Synthesis (TSMC 65 nm via Cadence Genus). We have synthesized the ASIC using the Cadence Genus™ tool for the TSMC 65nm CMOS technology node. The design is for a target frequency of 100 MHz. Comprised a total gate area of 109,124 μm^2 and 14,222 cells. Timing analysis identifies a very stable architecture, setting up a slack of 8,408 ps and critical path delay. 587 ps of stability for stable performance.

Table 3: Initial ASIC implementation results (TSMC 65 nm).

Metric	Preliminary Result
Total Gate Area	109,124 μm^2
Cell Count	14,222
Setup Slack (@ 100 MHz)	8,408 ps
Critical Path Delay	587 ps

4 Conclusion

This paper sets up a multi-source AXI4-Stream arbiter that acts as an effective hardware link between slow Power peripherals and powerful AI engines. By integrating an arbitration stage with asynchronous FIFOs, the architecture undertaken a validated framework. communication on different interfaces. The implementation on both FPGA and ASIC platforms, verified on both with this function, confirms Its outstanding area efficiency and silicon readiness. Based on these results, our design is ideal for For resource constrained applications that need a physical AI Link. in remote surgery, resolving clock domain mismatches and Data transfer bottlenecks in heterogeneous AI: how to mitigate them? and How to mitigate data movement in heterogeneous AI systems.

Acknowledgment

The authors would like to thank the Department of Electronics and Telecommunication Engineering, COEP Technological University, Pune, for providing the necessary facilities and technical support to carry out this work.

References

1. ARM: AMBA AXI and ACE Protocol Specification, ARM IHI 0022E (2023)
2. Makni, M., Baklouti, M., Niar, S., Abid, M.: Performance exploration of AMBA AXI4 bus protocols for wireless sensor networks. In: Proceedings of the 14th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA), pp. 1163–1169. IEEE, Hammamet (2017). <https://doi.org/10.1109/AICCSA.2017.26>
3. Silva, P.A., Paiva, P., Vetromille, M., Grellert, M.: Open-source verification IP for AXI-Stream. In: Proceedings of the 38th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI), pp. 1–5. IEEE, Manaus (2025). <https://doi.org/10.1109/SBCCI66862.2025.11218648>
4. Gao, H., Dai, H., Zeng, Y.: High-speed image processing and data transmission based on Vivado HLS and AXI4-Stream interface. In: Proceedings of the IEEE International Conference on Information and Automation (ICIA), pp. 575–579. IEEE, Wuyishan (2018). <https://doi.org/10.1109/ICInfA.2018.8812379>
5. Angadi, P., Shenvi, S.V., Lokesh, S.: Design and simulation of AXI4 stream interconnect using Verilog. In: Proceedings of the IEEE International Conference on Information Technology, Electronics and Intelligent Communication Systems (ICITE-ICS), pp. 1–4. IEEE, Bangalore (2024). <https://doi.org/10.1109/ICITEICS61368.2024.10625547>
6. Sledevič, T.: Synchronization of AXI streaming interfaces for convolution core implementation on FPGA. In: Proceedings of the 7th IEEE Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), pp. 1–4. IEEE, Liepaja (2019). <https://doi.org/10.1109/AIEEE48629.2019.8977065>
7. A.C.G., Aditya, S.S.: Constrained random verification of burst operation in AXI-4. In: Proceedings of the International Interdisciplinary Humanitarian Conference for Sustainability (IIHC), pp. 23–28. IEEE, Bengaluru (2022). <https://doi.org/10.1109/IIHC55949.2022.10060411>
8. Sangani, H., Mehta, U.: UVM based verification of read and write transactions in AXI4-Lite protocol. In: Proceedings of the IEEE Region 10 Symposium (TEN-SYMP), pp. 1–5. IEEE, Mumbai (2022). <https://doi.org/10.1109/TENSYMP54529.2022.9864552>
9. Shenoy, A.G., et al.: High speed data transfer from external interface to memory using FIFO and AXI DMA. In: Proceedings of the Global Conference on Communications and Information Technologies (GCCIT), pp. 1–7. IEEE, Bangalore (2024). <https://doi.org/10.1109/GCCIT63234.2024.10861944>
10. Noami, A., Alahdal, A., Kumar, B.P., Chandrasekhar, P., Safi, N.: High speed data transactions for memory controller based on AXI4 interface protocol SoC. In: Proceedings of the International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), pp. 1–7. IEEE, Bhilai (2021). <https://doi.org/10.1109/ICAECT49130.2021.9392539>
11. Liubavin, K.D., et al.: Design of a fully-autonomous low-power AXI4 firewall for PCI-Express NVMe SSD. In: Proceedings of the Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), pp. 166–169. IEEE, Saint Petersburg (2022). <https://doi.org/10.1109/ElConRus54750.2022.9755734>
12. Wang, B., Lu, Z.: Supporting QoS in AXI4 based communication architecture. In: Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 548–553. IEEE, Limassol (2020). <https://doi.org/10.1109/ISVLSI49217.2020.00008>

13. ARM: AMBA AXI Stream Protocol Specification, ARM IHI 0051B (2010)
14. Azarkhish, E., Rossi, D., Loi, I., Benini, L.: High performance AXI-4.0 based interconnect for extensible smart memory cubes. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1317–1322. IEEE, Grenoble (2015). <https://doi.org/10.7873/DATE.2015.0054>
15. Charyulu, M.L.N., Mounika, R., Jahangir, M.Z., Kushwah, V.S.: ASIC flow of AXI interface for ADPLL module using SCL180nm. In: Proceedings of the IEEE Madhya Pradesh Section Conference (MPCON), pp. 347–354. IEEE, Jabalpur (2025). <https://doi.org/10.1109/MPCON66082.2025.11256534>
16. Chandaluri, R., Nelakuditi, U.: Design and implementation of AXI4-Lite interface in Zynq SoC. In: Proceedings of the IEEE Delhi Section Conference (DELCON), pp. 1–4. IEEE, New Delhi (2022). <https://doi.org/10.1109/DELCON54057.2022.9753421>
17. Ma, C., Liu, Z., Ma, X.: Design and implementation of APB bridge based on AMBA 4.0. In: Proceedings of the International Conference on Consumer Electronics, Communications and Networks (CECNet), pp. 193–196. IEEE, Xianning (2011). <https://doi.org/10.1109/CECNET.2011.5768692>
18. Luna-Rodriguez, L.A., et al.: Implementation of 8-channel pulse width modulation with AXI4-Lite interface. In: Proceedings of the 19th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), pp. 1–5. IEEE, Mexico City (2022). <https://doi.org/10.1109/CCE56709.2022.9975965>
19. MS, M.S., Bhakthavatchalu, R.: High performance and low power AXI-Lite 4 master controller using System Verilog. In: Proceedings of the International Conference on Artificial Intelligence and Machine Learning Applications (AIMLA), pp. 1–6. IEEE, Namakkal (2024). <https://doi.org/10.1109/AIMLA59606.2024.10531611>
20. ARM: AMBA AXI Protocol Specification, ARM IHI 0022 (2025)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (<http://creativecommons.org/licenses/by-nc/4.0/>), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

