AMBA AXI Bus Technical Report

Advanced extensible Interface (AXI) [1] is a specification designed to interconnect components introduced by ARM [2] in 2003 as part of the AMBA3 specification. In 2010 ARM introduced the AMBA4 specification including the AXI4, AXI4 Lite and AXI4 Stream. The need for high speed, high performance system on chip (SoC) for a new age of digital communication requires an advanced bus infrastructure to prevent bottlenecking. [3] AXI is designed to meet communication requirements for a wide range of components such as memory, processing system, programming logic units, HDMI and other peripherals allowing high data transfer rate, low latency and backwards compatibility with AHB and APM protocols in previous AMBA3 specification.

AXI offers many additional features compared to AHB, [4] including support for outstanding, out of order and atomic operations, [5] burst transactions where only start address is provided, separate channels to allow direct memory access (DMA) and low power mode. There are 2 main reasons why AXI is significantly faster than AHB, [6] independent channels allows read and write transactions to happen simultaneously. However this can only happen when a slave is able to process both 1 write and 1 read transaction in a single clock cycle, this usually possible as the master can sent a read transaction to 1 slave and a write transaction to a different slave, if any of the slave cannot handle both read and write in the same cycle. In addition, outstanding transactions (OT) can occur when the master clock is not in sync with the slave clock, the AXI protocol allows the system to continue to issue addresses even though a response is not received.

Although both the older AMBA2 AHB protocol and the AMBA4 AXI protocol are bus masters. They are different in many ways, AXI was designed to increase efficiency, remove the need of using complex bridges when running high frequency operations and to reduce latency using an exclusive multi-channel read/write bus rather than a shared single channel bus. [5] AHB uses a fix bidirectional link pipeline for all transfers rather than independent channels where they are unidirectional (except for return signals). AXI also utilizes 'burst based' data transfer which allows multiple transactions and simultaneous read and write rather than AHB which uses a sperate address for each data item allowing only one transaction at any given point in time. AXI also has additional features such as security support.

[7] A downside of the AXI protocol is that it uses on average 50 percent more power. In terms of bus latencies, AHB has an edge over AXI as AXI starts at 64 bytes transactions whist AHB starts at 16 bytes. Moreover, one of the differences between AHB and AXI is that there is no 'split or retry response' in AXI. [8]In AHB at any give point in time only one master can be communicating with a slave, once a master has initiated a transaction it has to be completed before the next transaction. Split or retry response allows the bus to be released if the slave is busy. On the other hand, AXI can initiate multiple concurrent transactions in different channels. If a slave is taking some time to response, the system can deal with other transactions. [9] One of the reasons why AXI has a higher maximum operating frequency than AHB is that AXI supports pipeline register to be inserted in the channels whilst AHB does not support this. This can have a huge effect especially when SoC designs are big as data lines must travel for a long distance. On the other hand, AXI utilizes 5 parallel channels which requires a significant more connections than AHB which may cause layout issues when designing an SoC.

There are 5 channels in AXI, 2 channels for reading (Figure 1) and 3 for writing (Figure 2). From the diagram you can see that read and writes can have different latencies as they are transferred in different channels, the data is also transferred as bursts. A easy way to understand this is through a simple timing diagram of reading 16 bytes from a memory (Figure 3) using incrementing burst with 4 transfers and 4 bytes per transfer. Port `arburst` tells the memory the type of burst, in this case 0b01 represents incremental, `araddr` is the starting address of the memory where data is read, `arlen` specifies the number of transfers+1, `arsize` specifies the size of the burst using powers of 2 where 2^2 is 4 bytes, `arready` is provided from the memory to tell the master when it is ready, `arvalid` is provided from the master to tell the memory all the other `ar` signals are still valid, `rdata` transfers the data, `rresp` tells the master if the read failed or succeed, `rlast` indicates if it is the last burst of data, `rready`is from the master to indicate that it is ready to receive data, `rvalid` is from the memory to indicate when data packets are being sent from `rdata`. The red line shows the clock edges in which the transfers take place.

AXI write burst is quite similar to read burst. Another example (Figure 4) again with incrementing burst but with 3 transfers and 4 bytes per transfer. All the `aw` signals has the same function as the `ar` signal in a read burst. Port `wstrb` specifies strobed writes where 0b1111 indicates the memory writes all the bits, `bresp` is the response from the memory telling master if the write is successful. The three red lines shows the clock edges in which the transfer takes place.

To conclude, AHB may be more suitable in [9] systems which requires less silicon area such as small SoC, IoT devices or devices which does not require high performance demands and low operating frequency. AXI is suitable in large scale SoC with multiple clocks, high operating frequency, low latency and high bandwidth.

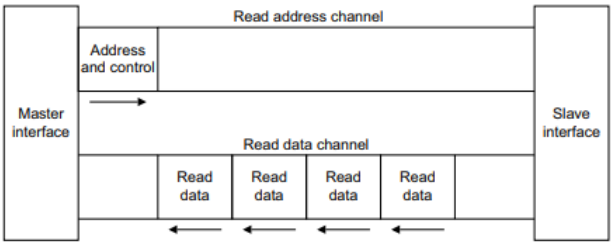

Figure 1 - AXI Read Channels [3]

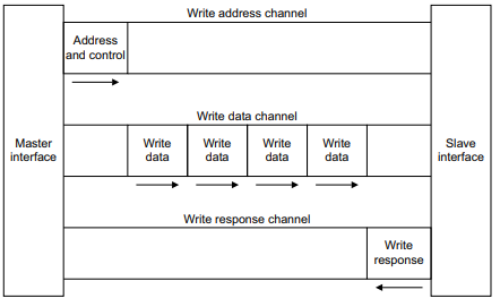

Figure 2 - AXI Write Channels [3]

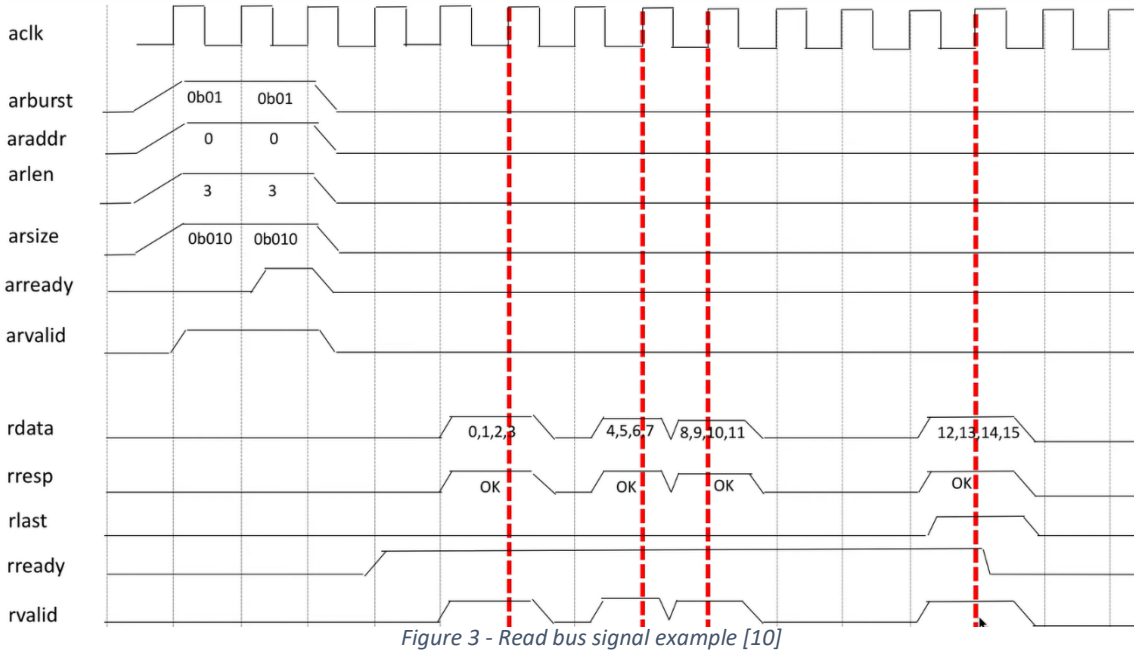

Figure 3 - Read bus signal example [10]

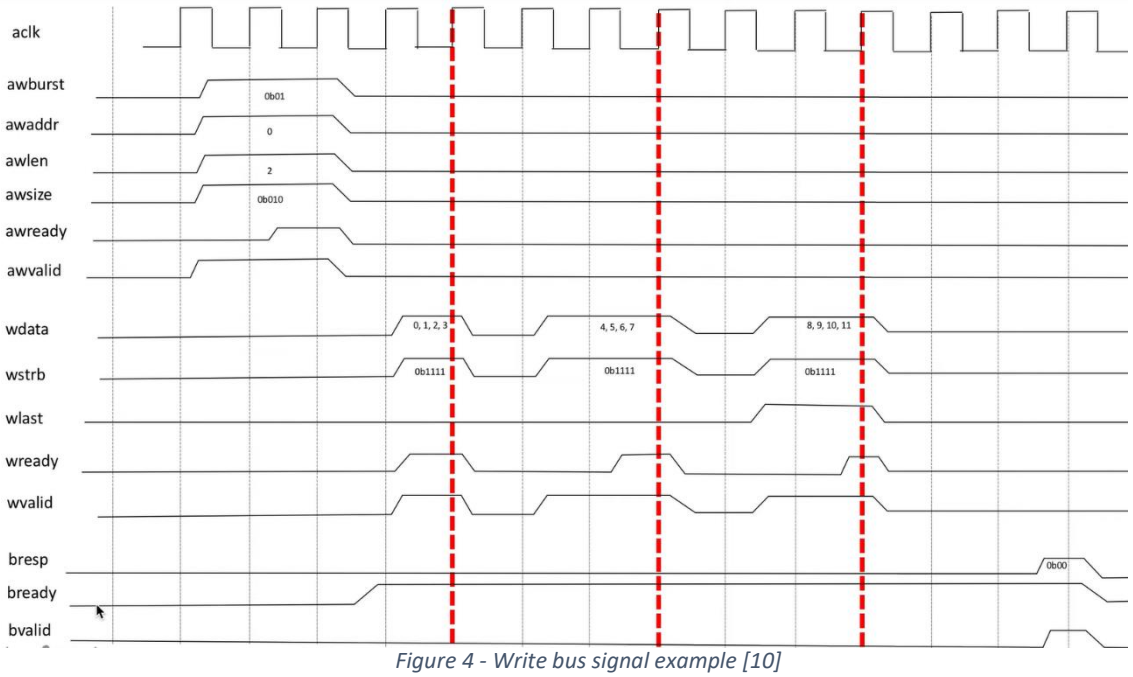

Figure 4 - Write bus signal example [10]

References:
[1] anysilicon, “Understanding AXI Protocol - A quick introduction,” anysilicon, 01 May 2018. [Online]. Available: https://anysilicon.com/understanding-axi-protocol-quick-introduction/. [Accessed 01 04 2020].
[2] ARM Developer, “AMBA - Arm Developer,” ARM, [Online]. Available: https://developer.arm.com/architectures/system-architectures/amba. [Accessed 2020 04 01].
[3] ARM, “AMBA AXI AND ACE PROTOCOL SPECIFICATION,” [Online]. Available: http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf. [Accessed 01 04 2020].
[4] B. Wade, “Introduction to AXI Protocol: Understanding the AXI interface,” 24 10 2016. [Online]. Available: https://community.arm.com/developer/ip-products/system/b/soc-design-blog/posts/introduction-to-axi-protocol-understanding-the-axi-interface. [Accessed 01 04 2020].
[5] D. M. Mick Posner, “Designing Using the AMBA (TM) 3 AXI (TM) Protocol -- Easing the Design Challenges and Putting the Verification Task on a Fast Track to Success,” Design&Reuse, [Online]. Available: https://www.design-reuse.com/articles/10299/designing-using-the-amba-tm-3-axi-tm-protocol-easing-the-design-challenges-and-putting-the-verification-task-on-a-fast-track-to-success.html. [Accessed 2020 04 01].
[6] A. Mittal, “AXI Vs AHB OR AHB Vs AXI Difference between AXI and AHB,” [Online]. Available: http://www.vlsiip.com/amba/axi_vs_ahb.html. [Accessed 01 04 2020].
[7] Difference Between, “Difference Between AHB and AXI,” DB, [Online]. Available: http://www.differencebetween.net/technology/difference-between-ahb-and-axi/. [Accessed 2020 04 01].
[8] J. Zhao, “Why there is no split or retry responce in AXI ?,” ARM community, 2016. [Online]. Available: https://community.arm.com/developer/ip-products/system/f/soc-design-forum/7024/why-there-is-no-split-or-retry-responce-in-axi. [Accessed 01 04 2020].
[9] A. Mittal, “AXI Vs AHB OR AHB Vs AXI Difference between AXI and AHB,” [Online]. Available: http://www.vlsiip.com/amba/axi_vs_ahb.html. [Accessed 01 04 2020].
[10] D. Huff, “What is AXI,” Dillon Huff, 24 04 2019. [Online]. Available: https://www.youtube.com/watch?v=1zw1HBsjDH8. [Accessed 01 04 2020].