May 11, 2022

The University of Texas at Austin

Advanced Embedded MCU Systems Final Project

AXI Bus Analyzer

Jatin Khare and Abhijjith Venkkateshraj

Motivation

- How does the cdma transfer work?

```
void transfer(unsigned int *cdma_virtual_address, int length){
     dma_set(cdma_virtual_address, SA, OCM_1);         // Write source address
     dma_set(cdma_virtual_address, DA, BRAM);          // Write destination address
     dma_set(cdma_virtual_address, CDMACR, 0x1000);   // Enable interrupts
     dma_set(cdma_virtual_address, BTT, length* 4);    // *initiate the transfer*
}
```

What if something bad happens after the control has been transferred to the CDMA, and the parent (user) is waiting for the child (transfer) to complete?

Given a bit stream,

- The burst size is unknown.
- The bus width is unknown.
- The test case (`ocm1[i] == ocm2[i]`) fails?
- The user wants to know what happened to the data for a particular address that was being transferred.

ChipScope™ Pro tool inserts logic analyzer, system analyzer, and virtual I/O low-profile software cores directly into your design, allowing you to view any internal signal or node, including embedded hard or soft processors.

Introducing AXI Sniffer

AXI Sniffer

- Debugging IP that shows you what happens inside the FPGA data flow.
- Once the `void transfer()` gives the control to the PL, the PS must not intervene until the transfer is over.
- *Idea*: make the PL capable of storing its own signal information.
- Probes the AXI signals used for data transfer.
- Runs the transaction and presents the post simulation results.

What (why) is AXI?

- **Why are we even using AXI?**

What (why) is AXI?

- **Why are we even using AXI?**

  One of the most popular Advanced Microcontroller Bus Architecture (AMBA) interface interconnects.

  - Independent read and write channels
  - Multiple outstanding addresses
  - No strict timing relationship between address and data operations
  - Support for unaligned data transfers
  - Out-of-order transaction completion
  - Burst transactions based on start address

Understanding AXI Jargons

Transfer Vs. Transaction

- A transfer is a one-time flow of ‘bus-width’ bits in the bus.
- For each transfer, the address is automatically incremented within one transaction.

- A transaction is made up of multiple transfers.
- An address is given on each transaction which is followed by number of transfers on its incremented addresses.

Understanding AXI Jargons

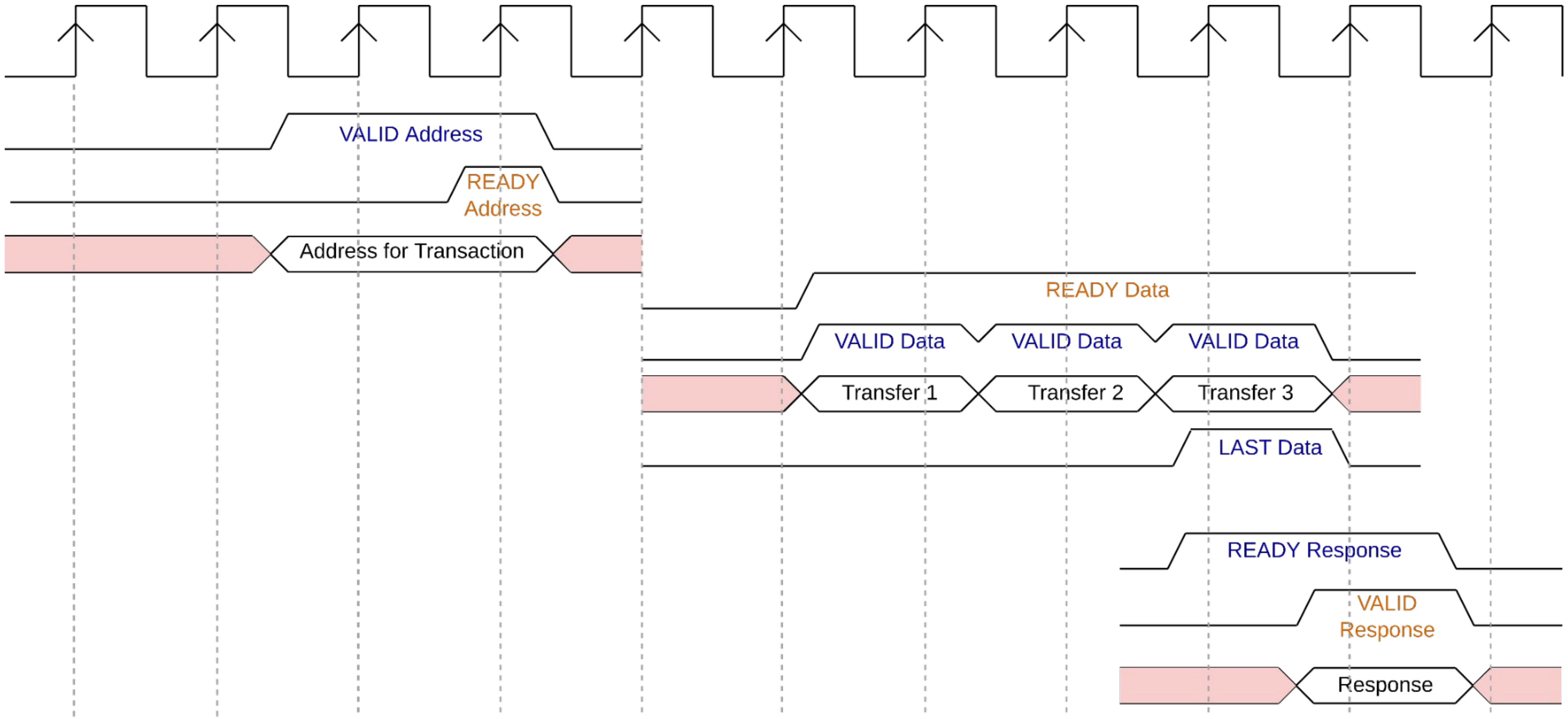
AXI Burst Length

- The number of transfers in a transaction.
- Equivalent to Vivado burst size.

AXI Burst Size

- The number of bits in one transfer.
- Equivalent to Vivado bus width.

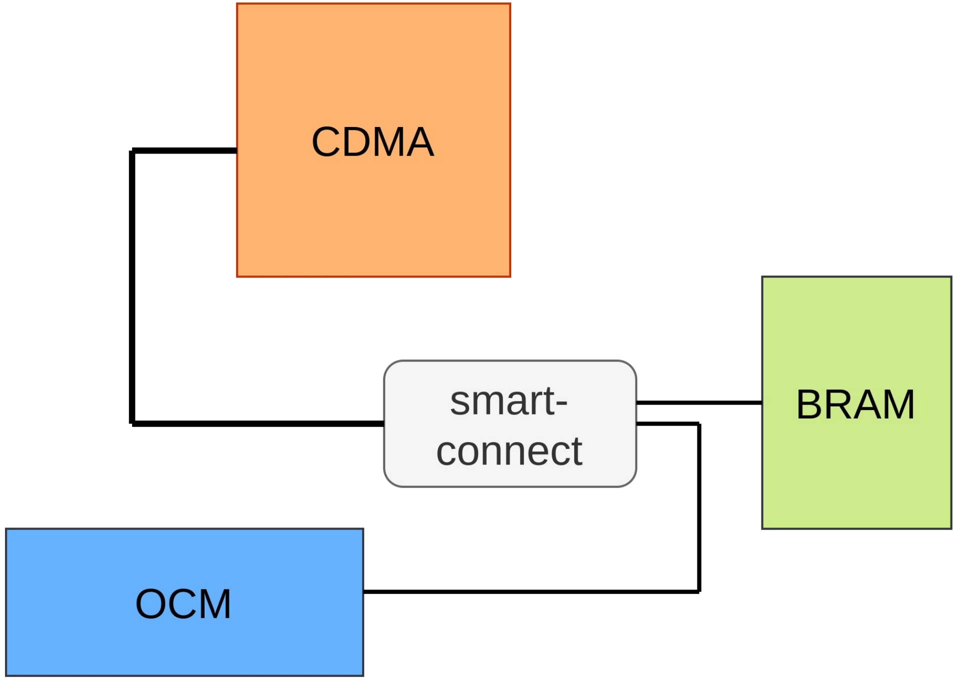
Write Transactions: AXI Timing Diagram



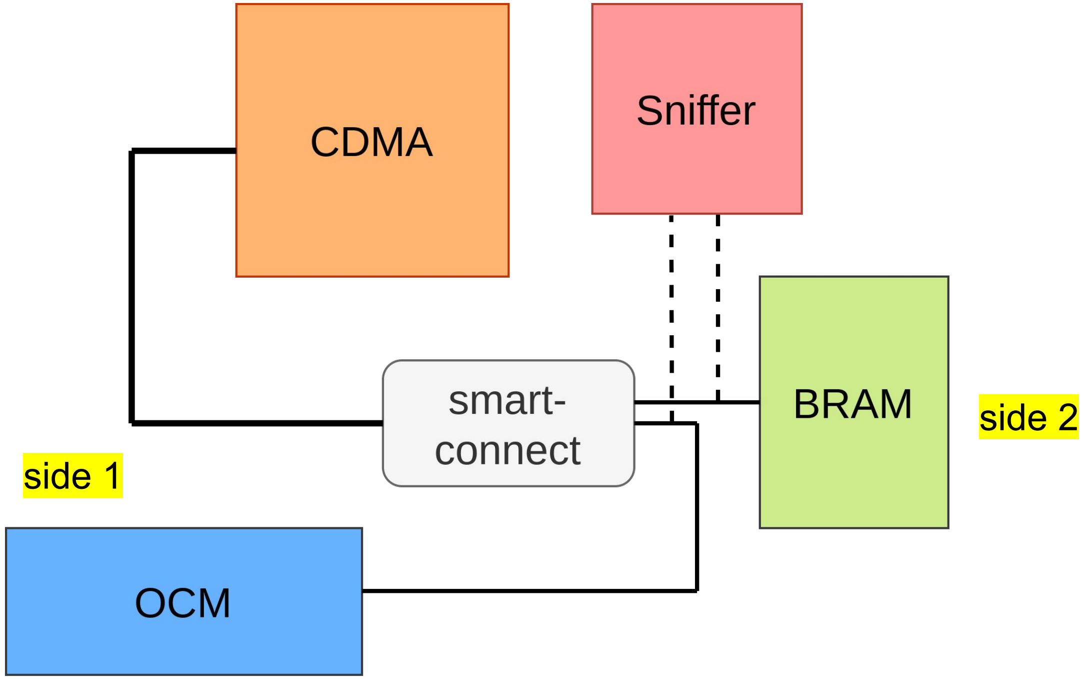
Capabilities of AXI Sniffer

- Supports data widths upto 1024 bits within the same Sniffer IP configuration.
- Easy scalable to probe any new signals of interest with minimal overhead.
- Provides design specific burst information.
- Shows the (address specific & total) Transfer and Transaction counts.
- Shows the Read and Write data in the bus, corresponding to the address of interest.
- Supports unaligned address requests.
- Later, one can also visualize the transfer via customized AXI timing diagram using generated *.json* file from the PS code.

Our Implementation

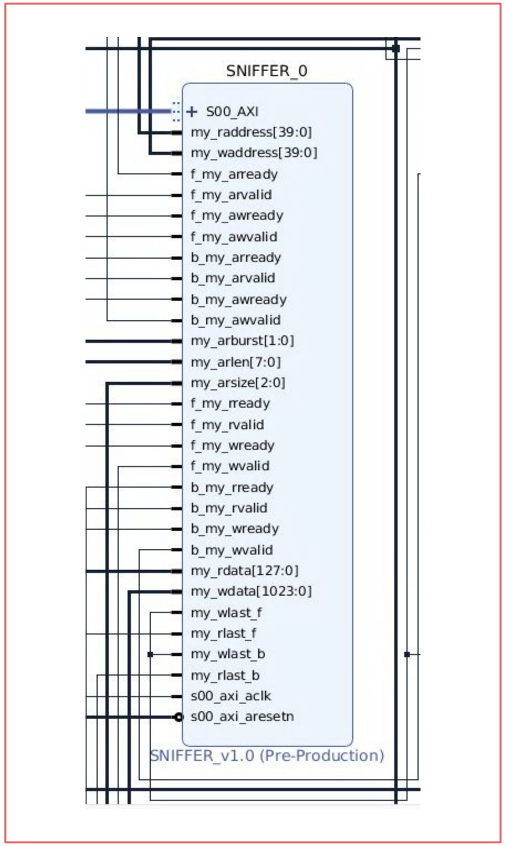


Our Implementation



Our Implementation

- Probing multiple AXI Signals
- rlast
- wlast
- ready, rvalid
- wready, wvalid
- arready, arvalid
- awready, awvalid
- arburst, arlen, arsize



Our Implementation

Burst info register: bit-field summary

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
b				e				a				d				Length								0	Size			0	0	Burst Type	

AXI Burst Length

- The number of transfers in a transaction.
- Equivalent to Vivado burst size.

Vivado burst size = AXI Length + 1

AXI Burst Size

- The number of bits in one transfer.
- Equivalent to Vivado bus width.

Vivado bus width = $2^{\text{AXI Size}}$ Bytes

Our Implementation

```
case(slv_reg4)
    5'h0: chunk_write <= MY_WDATA[1023:992];
    5'h1: chunk_write <= MY_WDATA[991:960];
    5'h2: chunk_write <= MY_WDATA[959:928];
    5'h3: chunk_write <= MY_WDATA[927:896];
     .
     .
     .
     .
   5'h1C: chunk_write <= MY_WDATA[127:96];
    5'h1D: chunk_write <= MY_WDATA[95:64];
    5'h1E: chunk_write <= MY_WDATA[63:32];
    5'h1F: chunk_write <= MY_WDATA[31:0];
   default: chunk_write <= 32'hDEFADEFA;
  endcase
```

```
5'h00   : reg_data_out <= burst_info_reg;
 5'h01   : reg_data_out <= slv_reg1;
 5'h02   : reg_data_out <= slv_reg2;
 5'h03   : reg_data_out <= slv_reg3;
 5'h04   : reg_data_out <= slv_reg4;
 5'h05   : reg_data_out <= f_read_access_count_ar;
 5'h06   : reg_data_out <= f_write_access_count_aw;
 5'h07   : reg_data_out <= f_read_access_count_r;
 5'h08   : reg_data_out <= f_write_access_count_w;
 5'h09   : reg_data_out <= b_read_access_count_ar;
 5'h0A   : reg_data_out <= b_write_access_count_aw;
 5'h0B   : reg_data_out <= b_read_access_count_r;
 5'h0C   : reg_data_out <= b_write_access_count_w;
 5'h0D   : reg_data_out <= 32'hFEED0000;
 5'h0E   : reg_data_out <= chunk_read;
 5'h0F   : reg_data_out <= chunk_write;
```

Our Implementation

- User gives the address to probe

$$\text{Transfer Count Number} = \frac{\textit{given address} - \text{Base Address}}{\text{Word Size (in Bytes) X Data Width}}$$

$$\text{Part Select Number} = \frac{(\textit{given address} - \text{Base Address})\ \%\ (\text{Word Size (in Bytes) X Data Width})}{\text{Word Size (in Bytes)}}$$

Demo

Demo parameters

OCM - CDMA Bus Width	CDMA - BRAM Bus Width	CDMA Burst Size	CDMA S & F
128	128	2	Disabled

Our Implementation

Analysis

S. No.	OCM - CDMA Bus Width	CDMA - BRAM Bus Width	CDMA Burst Size	CDMA S & F	Transactions	Transfers	Total Time
1.	128	32 (= 128 ÷ 4)	32	Enabled	32, 32	256, 1024	1301
2.	128	128 (=128 x 1)	32	Enabled	8, 8	256, 256	484
3.	128	512 (=128 x 4)	32	Enabled	2, 2	256, 64	471
4.	128	1024 (=128 x 8)	32	Enabled	1, 1	256, 32	468

*transferring 1024 words

Enable CDMA Store and Forward – Select this to enable an internal buffer in the AXI CDMA. Having an internal buffer in CDMA helps improve the performance of the AXI4 data bus.

Our Implementation

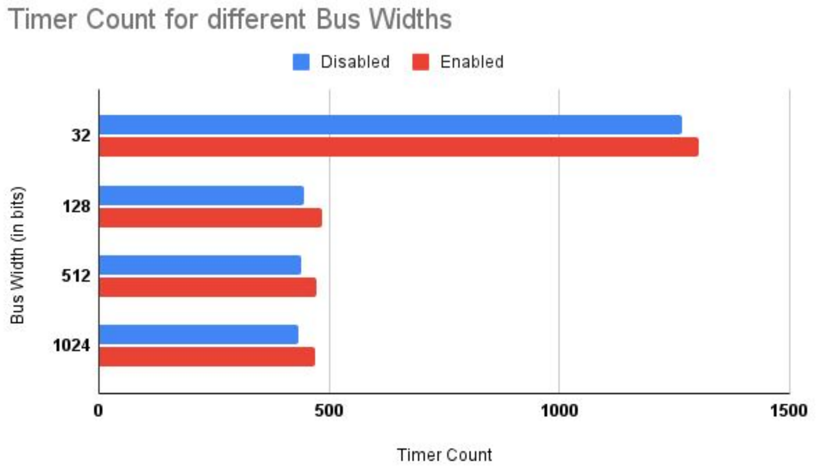
Analysis

S. No.	OCM - CDMA Bus Width	CDMA - BRAM Bus Width	CDMA Burst Size	CDMA S & F	Transactions	Transfers	Total Time
1.	128	32 (= 128 ÷ 4)	32	Disabled	32, 32	256, 1024	1266
2.	128	128 (=128 x 1)	32	Disabled	8, 8	256, 256	445
3.	128	512 (=128 x 4)	32	Disabled	2, 2	256, 64	439
4.	128	1024 (=128 x 8)	32	Disabled	1, 1	256, 32	433

*transferring 1024 words

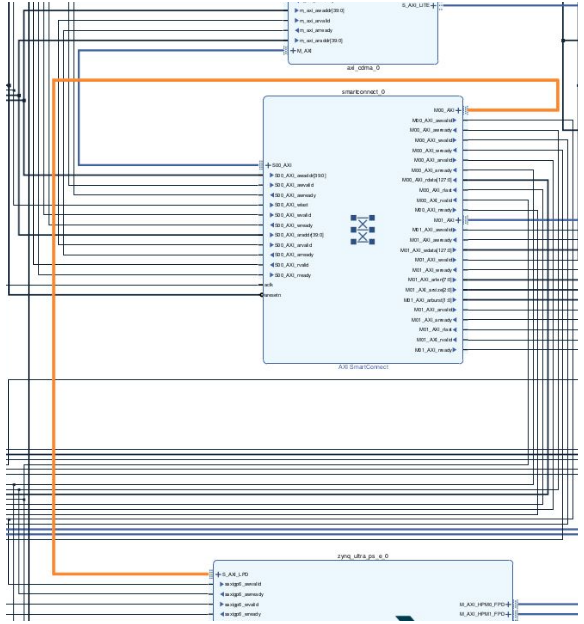
Things we learned

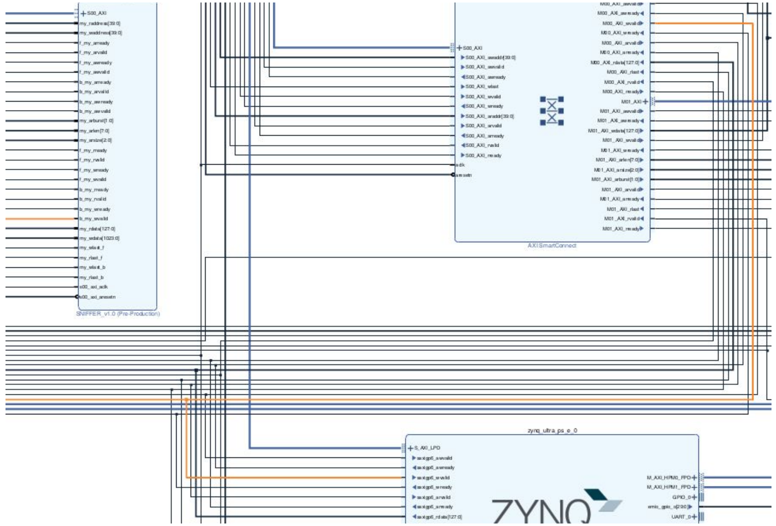
- Buffer is present even when the user disables the store and forward option in the CDMA.
- Nearly equal latency delay.



Things we learned

- Connection of wires





Things we learned

- The transfer and transaction counts do not differ from LPD in any of the ACP, HP, and FPD Ports.

- Synthesis with ACP, HP, and FPD Ports:

  If the synthesis frequency is different from the default 100 MHz, and if the port is changed from LPD to ACP, HP, or FPD – one needs to reset the frequency back to 100 MHz before changing the ports, and then set it back to your desired frequency.

Future Work

- Custom BRAM interface can be designed and interfaced with the Sniffer IP to store all the data flowing through the current burst of an AXI transaction.
- Performance analysis of various AXI interfaces via stress tests (AXI-Lite, HP and ACE) can be achieved by adding timer related signals in the Sniffer IP to capture the transfer info.

- PL performance metrics : No. of transactions, Throughput, Average Latency
- PS performance metrics : CPU Utilization, IPC, L1 Data cache access and miss rate, CPU stall cycles per instruction.

Reference : https://docs.xilinx.com/v/u/2019.1-English/ug1145-sdk-system-performance

Thank You!