Custom Module for AXI-Stream to Avalon Memory Mapped interface

The custom module designed here receives AXI-Stream data and generates Avalon Memory Mapped Interface signals in order to communicate with the EMIF IP to write and read data to the external memory device. The custom module acts as host and generates necessary signals to communicate with the agent, which is the EMIF IP. The pattern of Avalon Memory Mapped control signals generated by the custom module is with reference to the Traffic generator example design made by the Intel Quartus Prime software. The module generates sets of write burst instructions followed by read burst instructions where writing and reading is done from the same address.

Avalon Memory Mapped Interface

Avalon Memory Mapped Interface signals

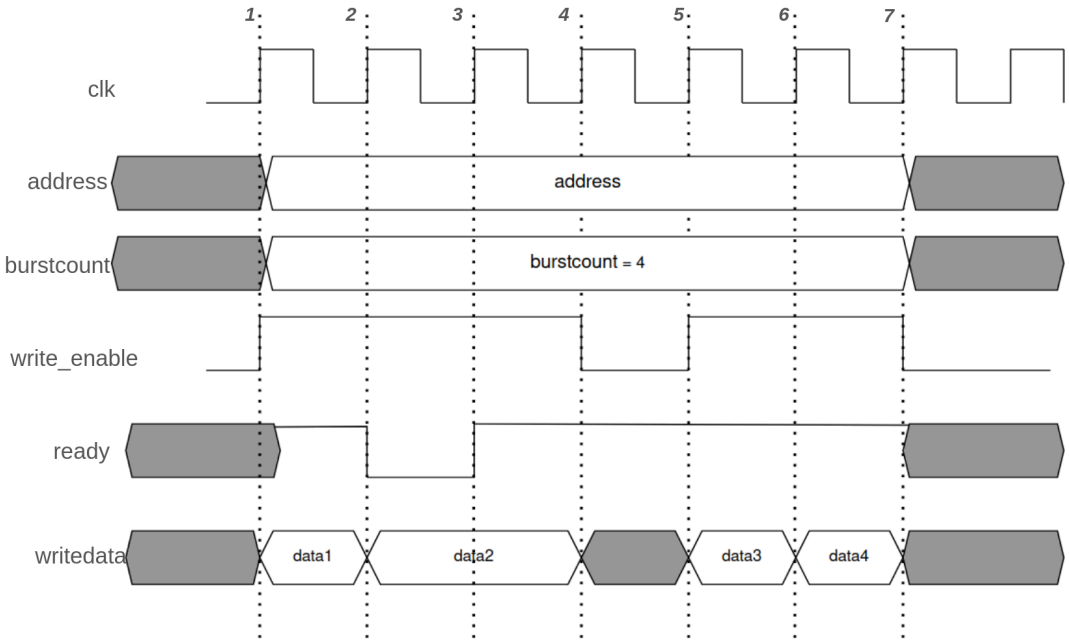
- **write_enable** : signal from host to agent, which is asserted in order to initiate write transfer.
- **read_enable** : signal from host to agent, which is asserted in order to initiate read transfer.
- **write_data** : data from host to agent which is to be written into the external memory.
- **read_data** : data from agent to host which is read from the external memory.
- **ready** : signal from agent to host that indicates the agent is ready to accept transfers.
- **readdatavalid** : signal from agent to host that indicates whether the data in the readdata line is valid or not.
- **address** : signal from host to agent, gives the start address for the burst transfer.
- **burstcount** : signal from host to agent, gives the number of transfers in each burst.
- **byteenable** : signal from host to agent, specifies which all bytes in the write data should be stored into the memory.

Avalon Memory Mapped Burst Transfers

During Avalon memory mapped burst transfers multiple transfers are carried out as a unit. When the agent asserts the **ready** signal, a burst transfer can be initiated by enabling the corresponding enable signal (**read_enable** or **write_enable**). When this happens the agent captures the **burstcount** and **address** signals at that instant and starts the burst transfer.

1. Write burst transfer
    - When a burst count of <*n*> is presented at the beginning of the burst, the agent must accept <*n*> successive units of writedata to complete the burst.

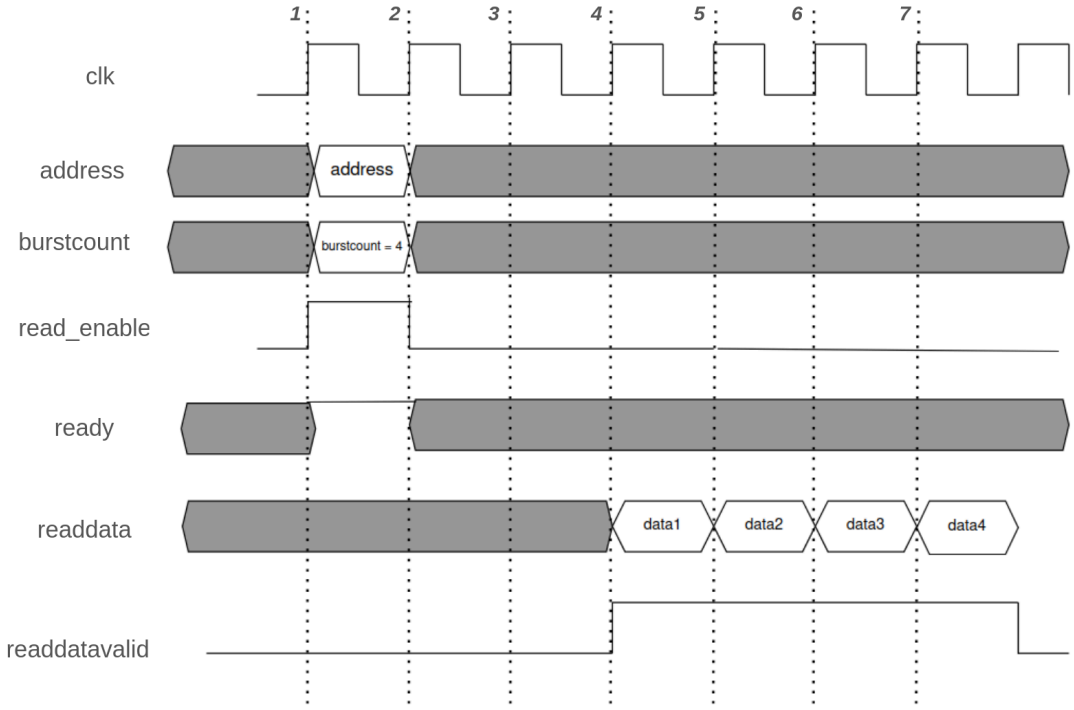
- The agent must only capture writedata when **write enable** is asserted. During the burst, the host can deassert write enable indicating that writedata is invalid. Deasserting write does not terminate the burst transfer. The agent waits to capture writedata and starts receiving write data once write enable is re-asserted.
- The agent delays a transfer by deasserting **ready** signal, forcing writedata, write, burstcount, and byteenable to be held constant.
- Each bit in the **byteenable** signal, enables each byte in the writedata.
- Throughout the duration of write burst transfer, the **burst count** and **address** should be held constant.



The above given waveform demonstrates a write burst transfer with burst count 4. This means that the agent should accept 4 write transfers.

- At instant ***1***, the **write_enable** is asserted by the host. As the **ready** signal is high here, the write transfer is initiated by the agent and the **address** and **burstcount** at this instant is captured by the agent. For write burst transfer, the **burstcount** and **address** is held constant throughout the burst. *data1* is captured by the agent.
- At instant ***2***, the **ready** signal goes low. This implies that the agent is not ready to accept the write transfer. So the host has to keep the write data constant till the agent is ready to accept data. Here write data is kept as *data2* itself in ***3*** instant also and as **ready** is high here it gets captured by the agent.
- At instant ***4***, the **write_enable** goes low, indicating that the write data here is not valid. Therefore, it is not captured by the agent.
- At instants ***5*** and ***6,*** *data3* and *data4* are captured respectively by the agent. As the burst count is 4, the write burst gets completed in instant ***6.***

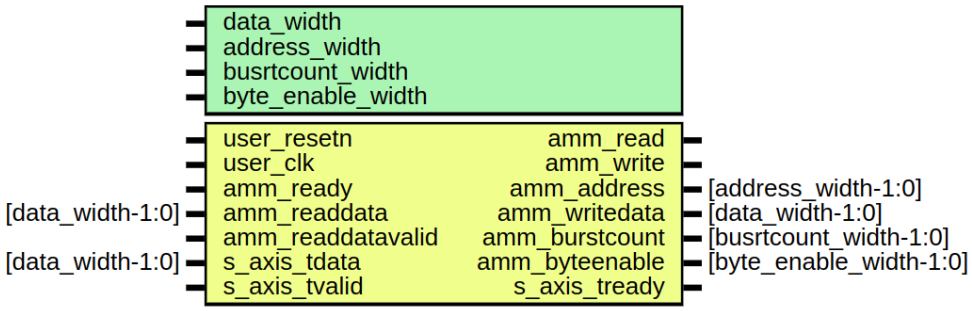
2. Read burst transfer
    - Read bursts are similar to pipelined read transfers with variable latency. i.e there is variable latency between capturing of **read_enable** by the agent and arrival of valid data in the **readdata** line.
    - **readdatavalid** indicates when the agent is presenting valid readdata.
    - Read burst can be initiated by asserting **read_enable,** whenever the **ready** signal is high. Later **read_enable** can be deasserted.
    - The **burst count** and **address** at the first clock cycle with read enable and ready is captured by the agent.



The above given waveform demonstrates a read burst transfer with burst count 4. This means that the agent should accomplish 4 read transfers.

- At instant ***1***, the **read_enable** is asserted by the host. As the **ready** signal is high here, the agent captures the read transfer along with values of **address** and **burstcount**. **read_enable**, **burstcount** and **address** can be changed after this instant.
- Data from the external memory starts coming out from instant ***4***. The latency between can be observed here (from instant ***1*** to ***4***). The arrival of valid data on the read data line is confirmed by the **readdatavalid** signal.
- As the burst count is 4, 4 data points are read from the memory and after this **readdatavalid** goes low.

Ports and parameters



Parameters

Parameter	Value	Description
data_width	128	Data width for AXI-Stream and Avalon Memory Mapped Interface.
address_width	27	Width of Address line
burstcount_width	7	Width of burstcount line
byte_enable_width	data_width/8 = 16	Width for byte enable line

Port Description

Port name	Width	Direction	Description
user_resetn	1	input	Reset signal from EMIF IP to user logic
user_clk	1	input	Clock signal from EMIF IP to user logic
AXI-stream slave interface Ports			
s_axis_tvalid	1	input	VALID signal for AXI-Stream

s_axis_tready	1	output	READY signal for AXI-Stream
s_axis_tdata	128	input	AXI-Stream data
Avalon Memory Mapped Interface Ports			
amm_ready	1	input	Indicates EMIF is ready for accepting a transfer.
amm_readdatavalid	1	input	Indicates whether data in read line from EMIF IP is valid or not.
amm_read	1	output	Read enable signal given to EMIF IP
amm_write	1	output	Write enable signal given to EMIF IP
amm_readdata	128	input	Read data from EMIF IP
amm_writedata	128	output	Write data to EMIF IP
amm_address	27	output	Start address for burst transfers.
amm_burstcount	7	output	Number of transfers per burst. (Here 28)
amm_byteenable	16	output	Byte enable for write data. (Here 16'hffff)

Logic Description

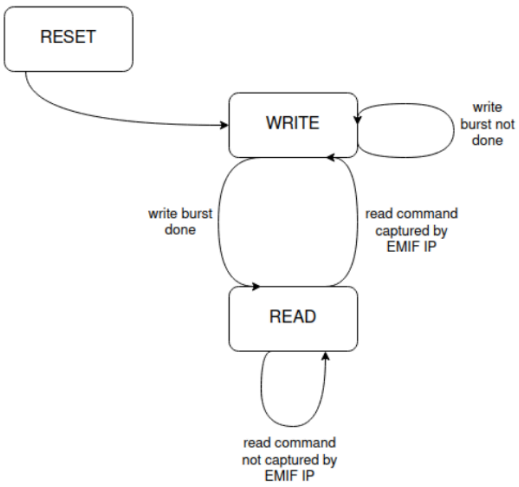
The custom module is controlled by FSM which has three states : RESET, WRITE and READ.

State Machine

- **RESET** : Reset state in the FSM. The FSM stays in this state if the **user_resetn** is low. Once the user_resetn goes high, the FSM transitions to WRITE state.

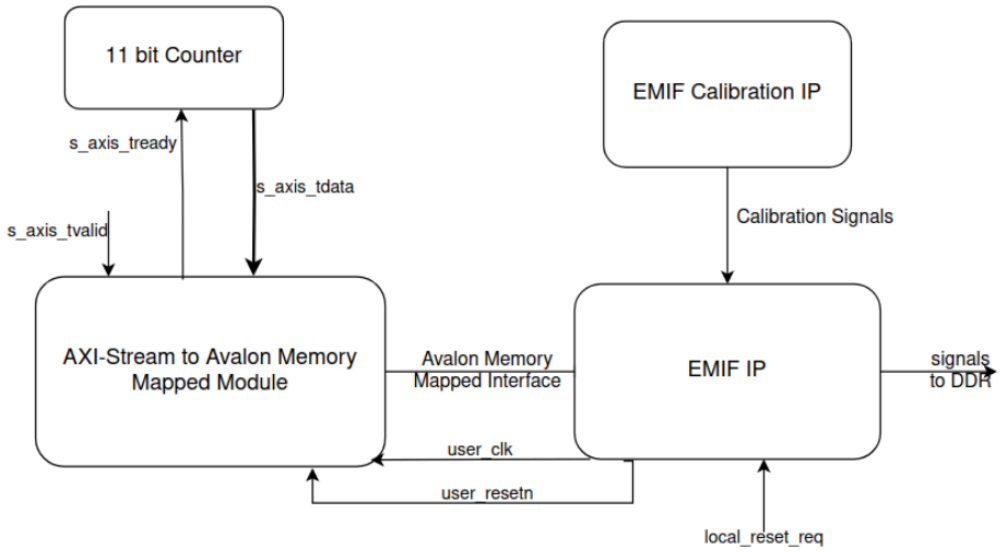
- **WRITE** : In this state, write burst happens. **amm_write** is asserted if **s_axis_tvalid** signal is high. The FSM stays in this state until the write burst is over. Once the write burst is over, it transitions to READ state.

- **READ** : In this state, read burst happens. **amm_read** is asserted. The address for read burst will be the same as that of the previously occurred write burst. It stays in the READ state until the EMIF IP captures the read command(i.e **amm_ready** and **amm_read** being high). Once the read instruction is captured, FSM transitions to WRITE state, with update of address.
  Next address = initial address + 16 * burstcount



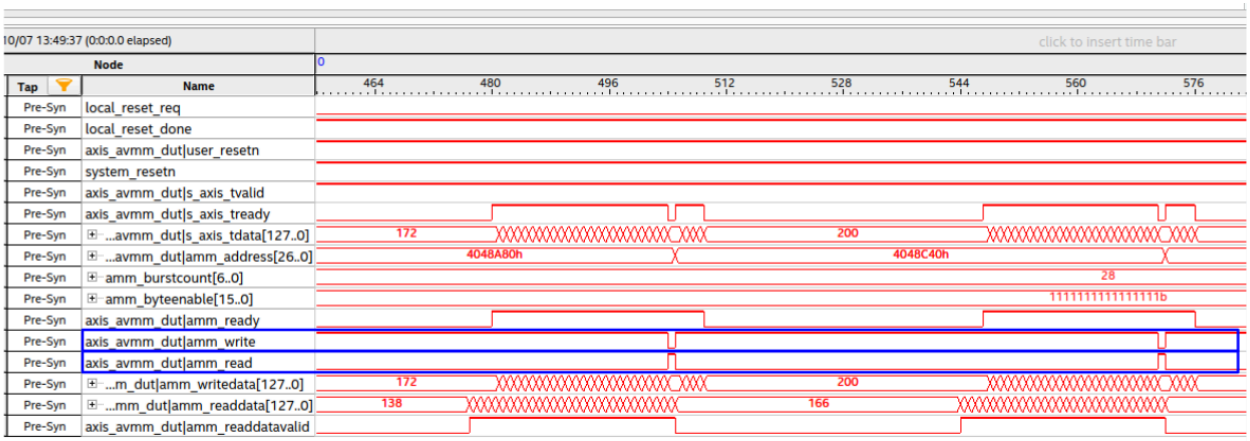
s_axis_tdata is received by the module if both **s_axis_tvalid** and **s_axis_tready** are high.
s_axis_tready is asserted only if the module is in WRITE state and **amm_ready** is high.

Overall schematic



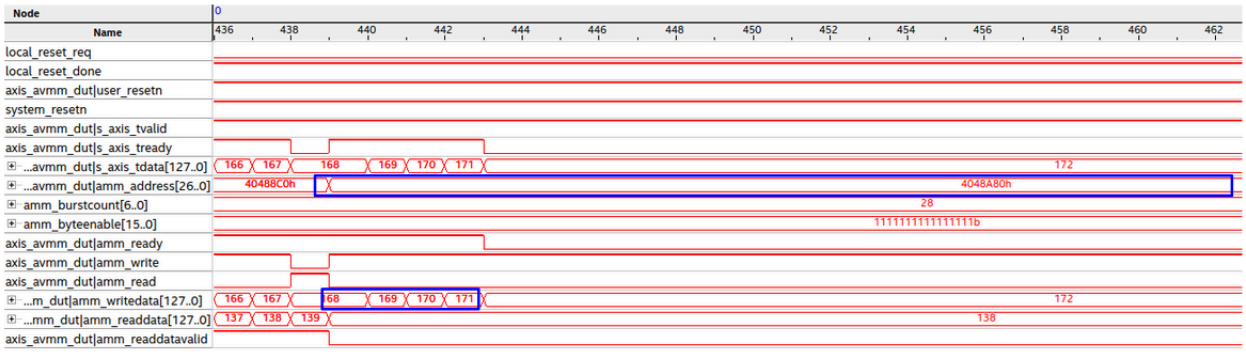
The AXI-Stream data is generated using a 11 bit counter. The clock(**user_clk**) and the reset(user_resetn) signals are generated by the EMIF IP. The signals to DDR are the signals contained in “mem” conduit of the EMIF IP. The **s_axis_tvalid** (which is set as 1) and **local_reset_req** can be modified using In-system Sources & Probes IP.

Onboard testing results

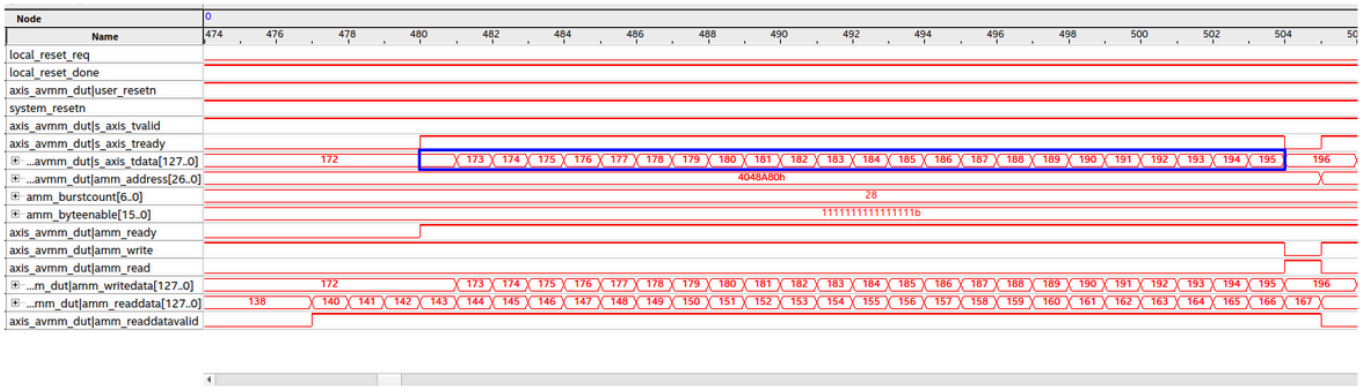


Here the **amm_write** is high when the module is in WRITE state and once the burst count number of write transfers is over module moves to READ state. It can be observed that after a certain duration for which **amm_write** is high, it becomes low and **amm_read** becomes high. As the **amm_ready** signal is high, the read transfer command is captured by the EMIF IP. Once the read transfer is captured by the EMIF IP, valid read data starts coming through **amm_readdata** with some latency.

Write burst

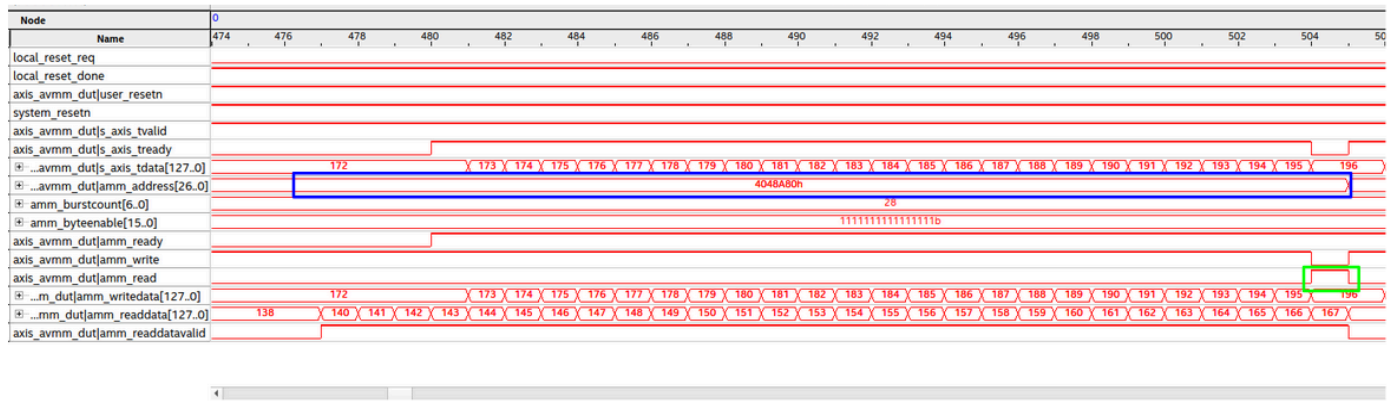


The marked portion shows the start of a write burst transfer. The start address (**amm_address**) is set as 4048A80h. The values which are being written start from “168”. After “171” the **amm_ready** signal goes low. So in order to write “172” to the memory the module should wait until the **amm_ready** signal becomes high again. During this period the **amm_writedata** is held constant.

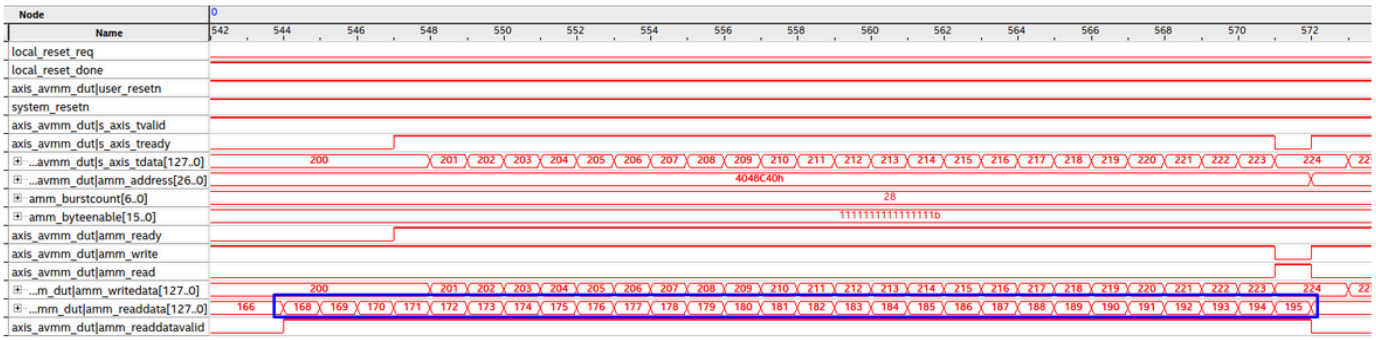


Once **amm_ready** is high again, the EMIF IP starts receiving data. Here the marked portion shows the remaining portion of the write burst. As the burstcount is set as 28, 28 write transfers takes place.
Values written to address starting at 4048A80h : 168, 169,... ,195.

Read burst



Here the portion marked in green shows that **amm_read** is made high. As **amm_ready** was high during this instant, the EMIF IP captures the read command. Here the address is 4048A80h and burstcount is 28. So the EMIF IP should start a read burst from the address 4048A80h, which has 28 read transfers in it.



After certain latency the read data starts coming through **amm_readdata** (shown in the marked region). The arrival of valid data can be confirmed from the **amm_readdatavalid** signal.

Project Codes: Github link