Tips for efficient AXI coding in VHDL

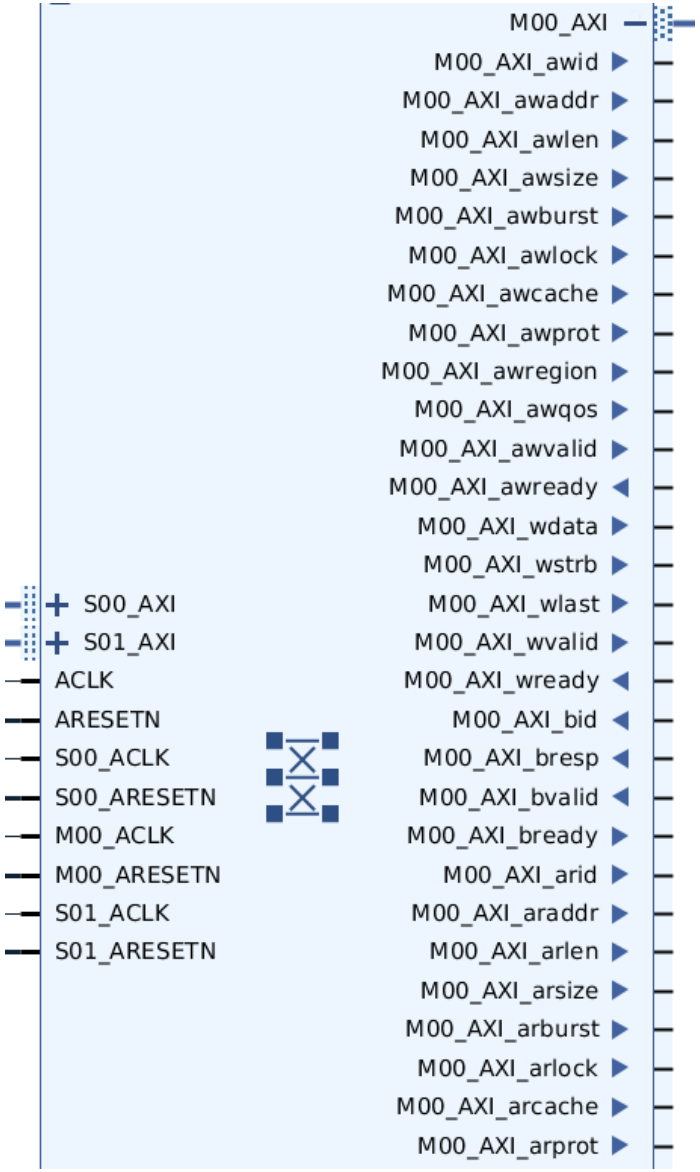
Adrian Byszuk

SY-EPC-CCE

24 May 2023

AXI4 in block design

- AXI4 is a complex bus with multiple wires going in both directions
- Pulling all these wires is time-consuming, especially when connecting to crossbar at top level
- Vivado Block Design helps with this by grouping them into interfaces



AXI4 in SystemVerilog

- SystemVerilog introduced a concept of „interfaces” and „modports”
- “interface” is essentially a structure
- „modport” defines a direction of each interface signal

```
modport Master (
  output aw_id, aw_addr, aw_len, aw_size, aw_burst, aw_lock, aw_cache,
  output w_data, w_strb, w_last, w_user, w_valid, input w_ready,
  input b_id, b_resp, b_user, b_valid, output b_ready,
  output ar_id, ar_addr, ar_len, ar_size, ar_burst, ar_lock, ar_cache,
  input r_id, r_data, r_resp, r_last, r_user, r_valid, output r_ready
);
```

```
interface AXI_BUS #(
  parameter int unsigned AXI_ADDR_WIDTH = 0,
  parameter int unsigned AXI_DATA_WIDTH = 0,
  parameter int unsigned AXI_ID_WIDTH   = 0,
  parameter int unsigned AXI_USER_WIDTH = 0
);

  localparam int unsigned AXI_STRB_WIDTH = AXI_DATA_WIDTH / 8;

  typedef logic [AXI_ID_WIDTH-1:0]   id_t;
  typedef logic [AXI_ADDR_WIDTH-1:0] addr_t;
  typedef logic [AXI_DATA_WIDTH-1:0] data_t;
  typedef logic [AXI_STRB_WIDTH-1:0] strb_t;
  typedef logic [AXI_USER_WIDTH-1:0] user_t;

  id_t             aw_id;
  addr_t           aw_addr;
  axi_pkg::len_t   aw_len;
  axi_pkg::size_t  aw_size;
  axi_pkg::burst_t aw_burst;
  logic            aw_lock;
  axi_pkg::cache_t aw_cache;
  axi_pkg::prot_t  aw_prot;
  axi_pkg::qos_t   aw_qos;
  axi_pkg::region_t aw_region;
  axi_pkg::atop_t  aw_atop;
  user_t           aw_user;
  logic            aw_valid;
  logic            aw_ready;
```

AXI4 in VHDL-2019

- VHDL 2019 introduces „mode views”, which are equivalent to SV interfaces
- VHDL 2019 support limited to Aldec Riviera-PRO and Vivado 2023.1

```
entity ZYNQ is
  port (
    Clk            : in ;
    nReset         : in ;
    AxiMaster1     : view Axi4LiteMasterView ;
    AxiMaster2     : view Axi4LiteMasterView of Axi4LiteType ;
    AxiResponder1  : view Axi4LiteResponderView ;
    AxiResponder2  : view Axi4LiteResponderView ;
```

```
type Axi4LiteWriteAddressType is record
  AWAddr      : std_logic_vector ;
  AWProt      : std_logic_vector ;
  AWValid     : std_logic ;
  AWReady     : std_logic ;
end record Axi4LiteWriteAddressType ;
```

```
view Axi4LiteWriteAddressMasterView of Axi4LiteWriteAddressType is
  AWAddr      : out ;
  AWProt      : out ;
  AWValid     : out ;
  AWReady     : in ;
end view Axi4LiteWriteAddressMasterView ;
```

```
view Axi4LiteWriteAddressResponderView is Axi4LiteWriteAddressMasterView'converse ;

view Axi4LiteResponderView is Axi4LiteMasterView'converse ;
```

AXI4 in VHDL-2008

- Similar technique can be used in VHDL-2008
- Introduced by Jim Lewis, author of OSVVM framework
- Use „inout” for port direction
- Synthesizable in Vivado!

```
--! AXI4 Address Write channel
type axi4_aw_t is record
  -- AXI4 Lite
  addr  : std_logic_vector;
  prot  : std_logic_vector(2 downto 0);
  valid : std_logic;
  ready : std_logic;
  -- AXI4 Full
  -- AXI recommended 3:0 for master, 7:0 at slave
  id : std_logic_vector;
  -- BurstLength = AxLen+1.  AXI4: 7:0,  AXI3: 3:0
  len : std_logic_vector(7 downto 0);
  -- #Bytes in transfer = 2**AxSize
  size : std_logic_vector(2 downto 0);
  -- AxBurst Binary Encoded (Fixed, Incr, Wrap, NotDefined)
  burst : std_logic_vector(1 downto 0);
  lock  : std_logic;
  -- AxCache One-hot (Write-Allocate, Read-Allocate, Modifia
  cache  : std_logic_vector(3 downto 0);
  qos    : std_logic_vector(3 downto 0);
  region : std_logic_vector(3 downto 0);
  user   : std_logic_vector; -- user config
end record axi4_aw_t;
```

```
--! AXI4 Bus with 5 channels (AW, W, B, AR, R)
type axi4_t is record
  aw : axi4_aw_t; --! Address Write channels
  w  : axi4_w_t; --! Data Write channels
  b  : axi4_b_t; --! Write Response channels
  ar : axi4_ar_t; --! Address Read channels
  r  : axi4_r_t; --! Data Read channels
end record axi4_t;
```

```
--! Array of AXI4 bus interfaces
type axi4_array_t is array (natural range <>) of axi4_t;
```

```
--! @brief VHDL wrapper for ZipCPU AXI crossbar
entity axixbar_wrapper is
  port (
    --! @name Fundamental clocks and reset
    axi_clk_i     : in std_ulogic; --! AXI bus clock
    axi_aresetn_i : in std_ulogic; --! AXI bus reset (active l

    --! @name Master and slave interfaces
    s_axi4_io : inout axi4_array_t; --! AXI4 slave interfaces
    m_axi4_io : inout axi4_array_t --! AXI4 master interfaces
  );
end axixbar_wrapper;
```

AXI4 in VHDL-2008 – signals

- Unconstrained record types allow parametrizing width at signal declaration
- Field-by-field mapping still possible
- Multiple signal drivers problem – doesn't really exist in synthesis. For simulation one needs to assign default '**Z**' driver. Any other process will override '**Z**' drivers

```
--! AXI4-Lite master bus array
signal axi4l_masters : axi4l_array_t(c_axi_masters-1 downto 0)(
  aw(addr(C_ADDR_WIDTH-1 downto 0)),
  w(data(31 downto 0),
    strb(3 downto 0)),
  ar(addr(C_ADDR_WIDTH-1 downto 0)),
  r(data(31 downto 0))
);
```

```
--! Top AXI4-Lite crossbar
x_axilxbar_w : entity work.axilxbar_wrapper
generic map(
  slave_addr_g => c_axi_address_map,
  slave_mask_g => c_axi_mask_map
)
port map(
  axi_clk_i     => axi_clk,
  axi_aresetn_i => axi_aresetn,
  -- Master connections (INTERCON is a slave)
  s_axi4l_io => axi4l_masters,
  -- Slave connections (INTERCON is a master)
  m_axi4l_io => axi4l_slaves
);
```

```
--! Fast AXI-controlled counter
x_cntr : entity work.axi_loadable_counter    no actual for generic "u
port map (
  axi_aresetn_i => axi_aresetn,
  axi_clk_i     => axi_clk,

  axi_awvalid_i => axi4l_slaves(C_AXI_S_CNTR_ID).aw.valid,
  axi_awready_o => axi4l_slaves(C_AXI_S_CNTR_ID).aw.ready,
  axi_awaddr_i  => axi4l_slaves(C_AXI_S_CNTR_ID).aw.addr(3 downto 0),
  axi_awprot_i  => axi4l_slaves(C_AXI_S_CNTR_ID).aw.prot,
  axi_wvalid_i  => axi4l_slaves(C_AXI_S_CNTR_ID).w.valid,
  axi_wready_o  => axi4l_slaves(C_AXI_S_CNTR_ID).w.ready,
  axi_wdata_i   => axi4l_slaves(C_AXI_S_CNTR_ID).w.data,
  axi_wstrb_i   => axi4l_slaves(C_AXI_S_CNTR_ID).w.strb,
```

Resources

- PULP platform SystemVerilog code - https://github.com/pulp-platform/axi/blob/master/src/axi_intf.sv
- Blog post by Jim Lewis - https://osvvm.org/archives/1668
- OSVVM implementation (VHDL-2008) - https://github.com/OSVVM/AXI4/tree/main/common/src
- EPC-CCE implementation (VHDL-2008) - https://gitlab.cern.ch/cce/fgc4/-/blob/master/common/vhd/axi4_interface_pkg.vhd


CERN