HEIG-VD HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Prof. Yann Thoma

Formal verification
Verification of an AXI module
CERN training - July 2025

Context

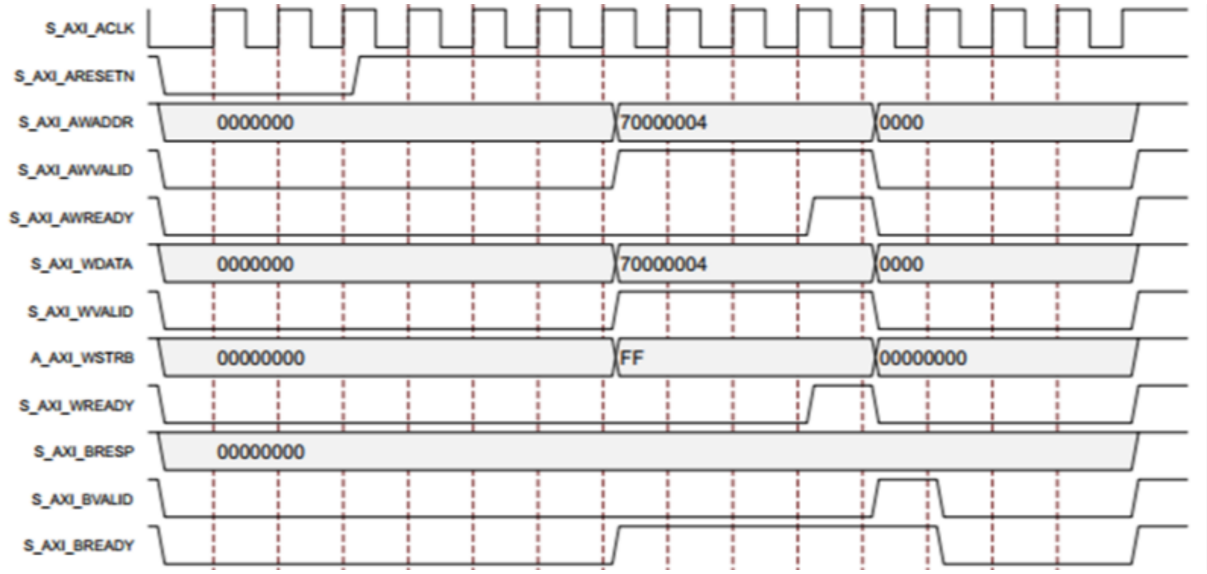
We are interested in verifying a module accessible through an AXI-lite interface.
Its entity is the following :

```
entity axi4lite_slave is
    generic (
        -- Width of S_AXI data bus
        AXI_DATA_WIDTH  : integer    := 32;  -- 32 or 64 bits
        -- Width of S_AXI address bus
        AXI_ADDR_WIDTH  : integer    := 12
    );
    port (
        clk_i           : in  std_logic;
        reset_i         : in  std_logic;
        -- AXI4-Lite
        axi_awaddr_i    : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);

        axi_awvalid_i   : in  std_logic;
        axi_awready_o   : out std_logic;
        axi_wdata_i     : in  std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        axi_wstrb_i     : in std_logic_vector((AXI_DATA_WIDTH/8)-1 downto 0);
        axi_wvalid_i    : in  std_logic;
        axi_wready_o    : out std_logic;
        axi_bresp_o     : out std_logic_vector(1 downto 0);
        axi_bvalid_o    : out std_logic;
        axi_bready_i    : in  std_logic;
        axi_araddr_i    : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);
        axi_arvalid_i   : in  std_logic;
        axi_arready_o   : out std_logic;
        axi_rdata_o     : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        axi_rresp_o     : out std_logic_vector(1 downto 0);
        axi_rvalid_o    : out std_logic;
        axi_rready_i    : in  std_logic;
        -- User input-output
        input_reg_A_i   : in  std_logic_vector(31 downto 0);
        input_reg_B_i   : in  std_logic_vector(31 downto 0);

        output_reg_A_o  : out std_logic_vector(31 downto 0);
        output_reg_B_o  : out std_logic_vector(31 downto 0);
        output_reg_C_o  : out std_logic_vector(31 downto 0)
    );
end entity axi4lite_slave;
```

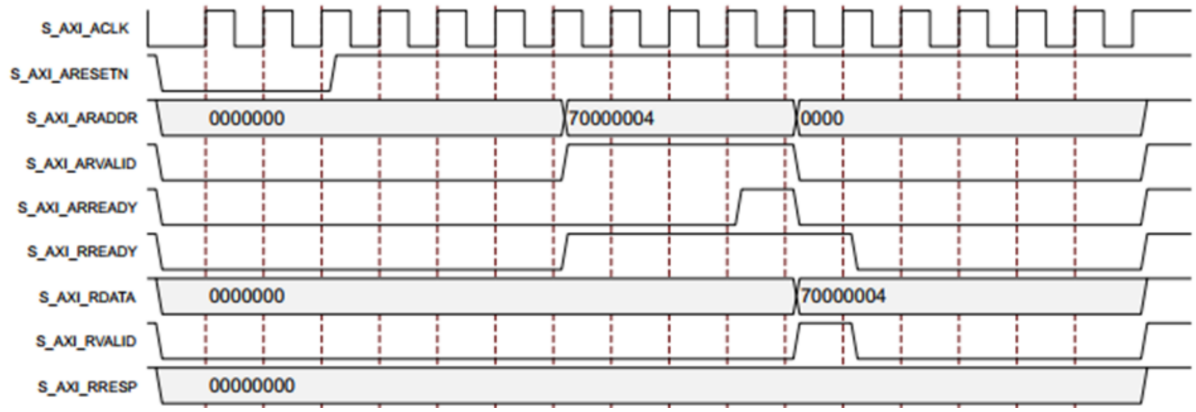
The module itself embeds different functionalities, corresponding to the following mapping :

Address	Direction	Description
0x0	Rd	Constant value : x"BADB100D"
0x1	Rd/Wr	Simple register
0x2	Rd	Reads the value present on the `input_reg_A` the previous cycle
0x4	Rd	Reads the value present on the `input_reg_B` the previous cycle
0x5	Wr	Sets the `output_reg_A` value
0x8	Wr	Sets the `output_reg_B` value
0x9	Wr	Sets the `output_reg_C` value
0xA	Wr/Rd	Sets or read the counter value
0xB	Wr	Writing at this address increments the counter

For the inputs reading (0x2 and 0x4), the previous cycle means the cycle before the last cycle of the read access.
A write transaction corresponds to the following waveform [1] :



A read transaction corresponds to the following waveform :



Your job is to test this module thanks to formal verification. As a suggestion, start with the raw AXI protocol, and then add assertions checking the behavior of the module itself. If the protocol assertions are checked it means the behavior assertions can directly some of the interesting signals (ready and valid for instance).

1. The waveform have been taken from https ://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081.