

Design Automation Homework - 6

Iman Tabrizian (9331032)

May 27, 2017

1 QUESTION 3

- AXI defines the following independent transaction channels:
 - read address
 - read data
 - write address
 - write data
 - write response

An address channel carries control information that describes the nature of the data to be transferred. The data is transferred between master and slave using either:

- A write data channel to transfer data from the master to the slave. In a write transaction, the slave uses the write response channel to signal the completion of the transfer to the master.
- A read data channel to transfer data from the slave to the master.

Below figures depict how data transfer works in AXI protocol between master and slave:

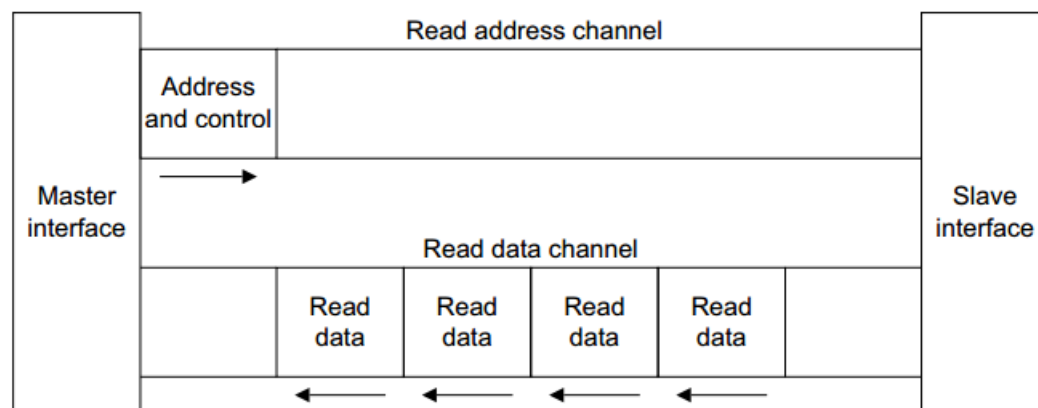


Figure A1-1 Channel architecture of reads

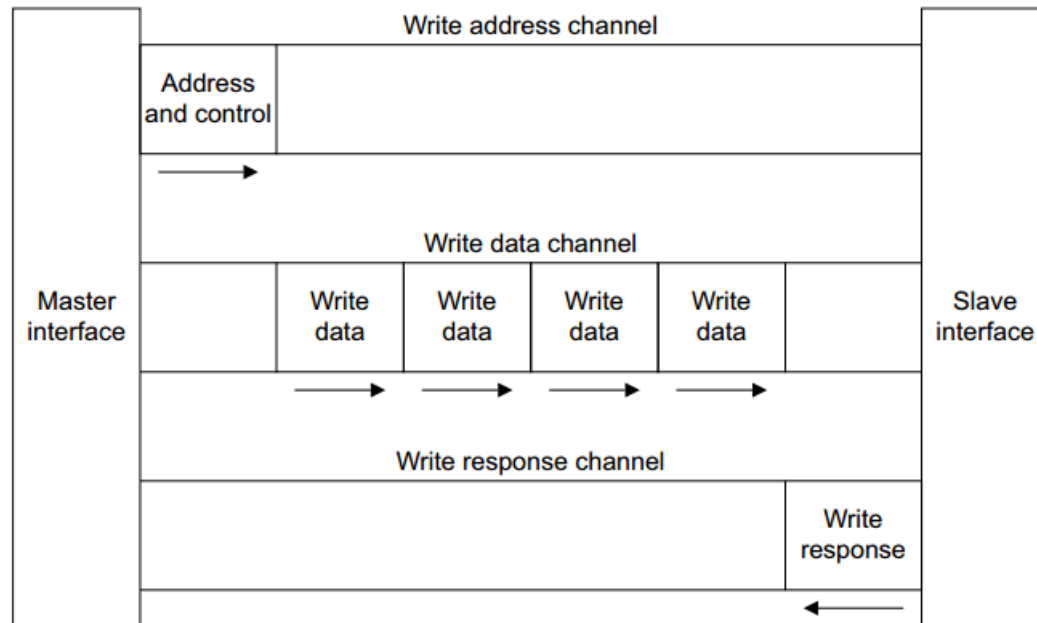


Figure A1-2 Channel architecture of writes

- – **Stream:** The AXI4-Stream protocol is used for applications that typically focus on a data-centric and data-flow paradigm where the concept of an address is not present or not required. Each AXI4-Stream acts as a single unidirectional channel for a handshake data flow.
- In this question we tried to break up to two modules: First module is stage which does what this circuit is supposed to do in one stage and a second module called cryptor which creates an instance from the previous section module and only alters inputs.

Here is the source code to stage module:

```
library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity stage is
  port(
    a: in std_logic_vector(7 downto 0);
    b: in std_logic_vector(7 downto 0);
    key: in std_logic_vector(7 downto 0);
    outputa: out std_logic_vector(7 downto 0);
    outputb: out std_logic_vector(7 downto 0)
  );
end entity;

architecture rtl of stage is

  signal le, re: std_logic_vector(7 downto 0);
  signal l1, r1: std_logic_vector(7 downto 0);
  signal l1_shifted, r1_shifted: std_logic_vector(7 downto 0);
  signal result_a: std_logic_vector(7 downto 0);
  signal result_b: std_logic_vector(7 downto 0);
```

```

begin
    le <= std_logic_vector(to_unsigned(to_integer(unsigned(a)) + to_integer(unsigned(key)), 7));
    re <= std_logic_vector(to_unsigned(to_integer(unsigned(b)) + to_integer(unsigned(key)), 7));

    l1 <= le xor re;
    l1_shifted <= l1(6 downto 0) & '0';
    result_a <= std_logic_vector(to_unsigned(to_integer(unsigned(l1_shifted)) + to_integer(unsigned(key)), 7));

    r1 <= result_a xor re;
    r1_shifted <= r1(6 downto 0) & '0';
    result_b <= std_logic_vector(to_unsigned(to_integer(unsigned(r1_shifted)) + to_integer(unsigned(key)), 7));

    outputa <= result_a;
    outputb <= result_b;

```

end rtl;

Here is the source code to cryptor module:

```

library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity cryptor is
    port(
        a: in std_logic_vector(7 downto 0);
        b: in std_logic_vector(7 downto 0);
        key: in std_logic_vector(7 downto 0);
        rst: in std_logic;
        clk: in std_logic;
        outputa: out std_logic_vector(7 downto 0);
        outputb: out std_logic_vector(7 downto 0)
    );
end entity;

architecture rtl of cryptor is

    component stage is
        port(
            a: in std_logic_vector(7 downto 0);
            b: in std_logic_vector(7 downto 0);
            key: in std_logic_vector(7 downto 0);
            outputa: out std_logic_vector(7 downto 0);
            outputb: out std_logic_vector(7 downto 0)
        );
    end component;

    type STATE is (S1, S2, S3, S4, S5, S6, S7, S8, FINISH);
    signal current_state: STATE := S1;

    signal a_current: std_logic_vector(7 downto 0);
    signal b_current: std_logic_vector(7 downto 0);

```

```

signal crypted_a: std_logic_vector(7 downto 0);
signal crypted_b: std_logic_vector(7 downto 0);

begin

crypto: stage port map(a_current, b_current, key, crypted_a, crypted_b);

process(clk)
begin
    if(rising_edge(clk)) then
        if(rst = '1') then
            current_state <= S1;
        else
            case current_state is
                when S1 =>
                    a_current <= a;
                    b_current <= b;
                    current_state <= S2;

                when S2 =>
                    a_current <= crypted_a;
                    b_current <= crypted_b;
                    current_state <= S3;

                when S3 =>
                    a_current <= crypted_a;
                    b_current <= crypted_b;
                    current_state <= S3;

                when S4 =>
                    a_current <= crypted_a;
                    b_current <= crypted_b;
                    current_state <= S3;

                when S5 =>
                    a_current <= crypted_a;
                    b_current <= crypted_b;
                    current_state <= S3;

                when S6 =>
                    a_current <= crypted_a;
                    b_current <= crypted_b;
                    current_state <= S3;

                when S7 =>
                    a_current <= crypted_a;
                    b_current <= crypted_b;
                    current_state <= S3;

                when S8 =>
                    a_current <= crypted_a;

```

```

        b_current <= crypted_b;
        current_state <= FINISH;

        when FINISH =>
            outputa <= a_current;
            outputb <= b_current;
            current_state <= FINISH;

        end case;
    end if;
end if;
end process;
end rtl;

```

- Here is the source code to elevator:

```

library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity elevator is
    port(
        floor_1_key_up: in std_logic;
        floor_1_key_down: in std_logic;
        floor_2_key_up: in std_logic;
        floor_2_key_down: in std_logic;
        floor_3_key_up: in std_logic;
        floor_3_key_down: in std_logic;
        floor_1: in std_logic;
        floor_2: in std_logic;
        floor_3: in std_logic;
        no_stop: in std_logic;
        clk: in std_logic;
        floor: out integer
    );
end entity;

architecture rtl of elevator is

    constant S1: integer := 1;
    constant S2: integer := 2;
    constant S3: integer := 3;
    signal current_state: integer := S1;

begin

    process(clk)
    begin
        if(rising_edge(clk)) then
            floor <= current_state;

```

```

case current_state is
  when S1 =>
    if(floor_1 = '1') then
      current_state <= S1;
    elsif(floor_2 = '1') then
      current_state <= S2;
    elsif(floor_3 = '1') then
      if(no_stop = '1') then
        current_state <= S3;
      else
        if(floor_2_key_up = '1') then
          current_state <= S2;
        else
          current_state <= S3;
        end if;
      end if;
    end if;
  end if;

  when S2 =>
    if(floor_2 = '1') then
      current_state <= S2;
    elsif(floor_1 = '1') then
      current_state <= S1;
    elsif(floor_3 = '1') then
      current_state <= S3;
    end if;

  when S3 =>
    if(floor_3 = '1') then
      current_state <= S3;
    elsif(floor_2 = '1') then
      current_state <= S2;
    elsif(floor_1 = '1') then
      if(no_stop = '1') then
        current_state <= S1;
      else
        if(floor_2_key_down = '1') then
          current_state <= S2;
        else
          current_state <= S1;
        end if;
      end if;
    end if;
  end if;
when others =>

end case;
end if;
end process;
end rtl;

```

And below is the testbench:

```
library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity elevator_tb is
end entity;

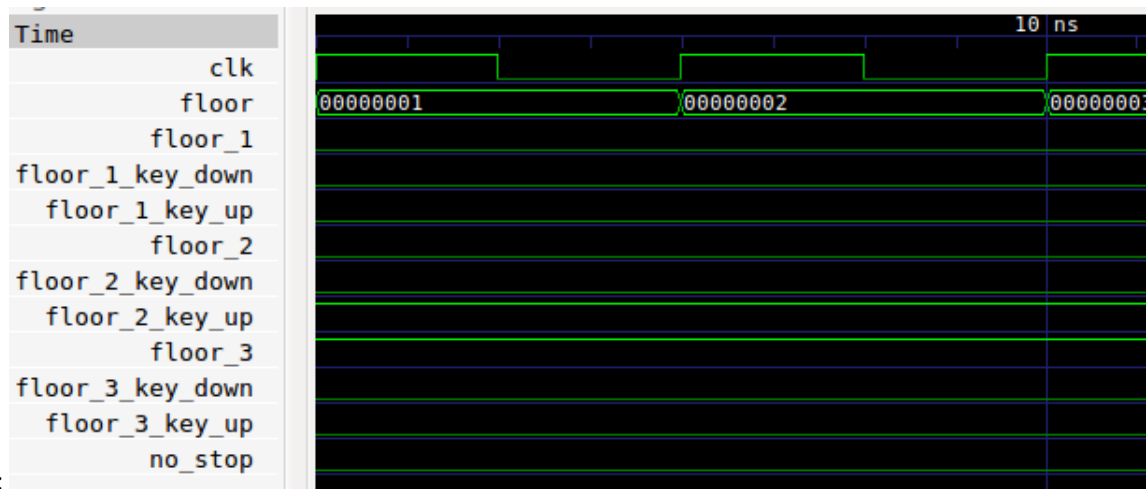
architecture tb of elevator_tb is
    component elevator is
        port(
            floor_1_key_up: in std_logic;
            floor_1_key_down: in std_logic;
            floor_2_key_up: in std_logic;
            floor_2_key_down: in std_logic;
            floor_3_key_up: in std_logic;
            floor_3_key_down: in std_logic;
            floor_1: in std_logic;
            floor_2: in std_logic;
            floor_3: in std_logic;
            no_stop: in std_logic;
            clk: in std_logic;
            floor: out integer
        );
    end component;

    signal floor_1_key_up: std_logic := '0';
    signal floor_1_key_down: std_logic := '0';
    signal floor_2_key_up: std_logic := '0';
    signal floor_2_key_down: std_logic := '0';
    signal floor_3_key_up: std_logic := '0';
    signal floor_3_key_down: std_logic := '0';
    signal floor_1: std_logic := '0';
    signal floor_2: std_logic := '0';
    signal floor_3: std_logic := '0';
    signal no_stop: std_logic := '0';
    signal clk: std_logic := '0';
    signal floor: integer := 1;

begin
    mapping: elevator port map(floor_1_key_up, floor_1_key_down,
                                floor_2_key_up, floor_2_key_down, floor_3_key_up,
                                floor_3_key_down, floor_1, floor_2, floor_3, no_stop, clk,
                                floor);

    floor_3 <= '1';
    floor_2_key_up <= '1';
    clk <= not clk after 2 ns;

end tb;
```



And here is the simulation result: