Danmarks Tekniske Universitet

Exercise 2 Asynchronous FIFO

34349: FPGA design for Communication Systems
Spring 2025

Group

Dimitrios Vlachos - s243192
Theodoros Pontzouktzidis - s250239

Contents

1 Task 1

While the chain of flip-flops can lower the chance of metastability, it cannot eliminate it. When a multi-bit pointer is synchronized using a simple flip-flop chain, each bit of the pointer is synchronized independently. Due to the asynchronous nature of the clocks (the 2 clocks may have different frequencies), some bits of the pointer may be sampled by the destination clock before they change, while others may be sampled after they change. This can result in a temporary corrupted value of the pointer.

Adding more flip-flops to the chain reduces the probability of metastability but does not solve it. Therefore, adding more flip-flops is not a complete solution.

2 Task 2

2.1 Why the Given Circuit Works

In Gray code, only one bit changes at a time when the pointer increments or decrements for example:

$$\begin{array}{ll} \text{Binary} & 0111 \rightarrow 1000 \quad \text{(all bits changed)} \\ \text{Gray Code} & 0100 \rightarrow 1100 \quad \text{(only one bit changed)} \end{array}$$

Even if the synchronization is delayed, the synchronized value will only be off by one bit. Below is a table showing 4-bit binary addresses and their corresponding Gray code values:

Address (Binary)	Gray Code
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Table 1: 4-bit Binary to Gray Code Conversion

The circuit converts the binary pointers to Gray code before synchronization. Then the pointers are synchronized using a chain of flip-flops.

2.2 Logical Equations for Binary-to-Gray and Gray-to-Binary Converters

As we can see from Table 1. the Gray code is derived from the binary code by performing a bitwise XOR operation between each bit and the next higher bit. For an n-bit binary number, the Gray code can be calculated as follows:

$$\text{Gray}[i] = \text{Binary}[i] \oplus \text{Binary}[i+1]$$

- Starting from $i = 1$
- Here, Binary$[i]$ is the i-th bit of the binary number, and Binary$[i+1]$ is the next higher bit.
- The most significant bit (MSB) of the Gray code is the same as the MSB of the binary number.

Now, for the conversion of Gray code to binary code we can perform a bitwise XOR operation between each bit of the Gray code and the higher bit that was converted starting from the MSB. For an n-bit Gray code, the binary code can be calculated as follows:

$$\text{Binary}[i] = \text{Gray}[i] \oplus \text{Binary}[i+1]$$

- Starting from $i = n - 1$
- Here, Binary$[i+1]$ is the next higher bit of the binary number (last bit that was converted).
- The most significant bit (MSB) of the binary number is the same as the MSB of the Gray code.

2.3 Addressing Synchronization Delay

The pointers must pass through a chain of flip-flops to resolve metastability, which introduces some delay. This delay does not affect the overall operation of the FIFO. So the FIFO will function properly.

3 Task 3 Asynchronous FIFO

3.1 Asynchronous FIFO implementation (VHDL code)

3.1.1 Top level module

```
entity async_fifo is
 generic (
    f_DATA_WIDTH : natural := 8;
    f_ADDRESS_WIDTH : natural := 5
  );

 port (
    reset : in std_logic;
    wclk : in std_logic;
    rclk : in std_logic;

    -- OCCUPANCY
    fifo_occu_in   : out std_logic_vector(f_ADDRESS_WIDTH-1 downto 0);
    fifo_occu_out  : out std_logic_vector(f_ADDRESS_WIDTH-1 downto 0);

    -- WRITE
    write_enable   : in std_logic;
    write_data_in  : in std_logic_vector(f_DATA_WIDTH-1 downto 0);

    -- READ
    read_enable    : in std_logic;
    read_data_out  : out std_logic_vector(f_DATA_WIDTH-1 downto 0);

    full           : out std_logic;
    empty          : out std_logic
 );
end async_fifo;

architecture ar of async_fifo is
    signal wptr_sync      : std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
    signal rptr_sync      : std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
    signal wptr           : std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
    signal rptr           : std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
    signal waddr          : std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0); -- address to go in mem
    signal raddr          : std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0); -- address to go in mem
    signal w_en           : std_logic;
    signal r_en           : std_logic;
    signal f              : std_logic ;
    signal e              : std_logic ;
    signal mem_dbg       : dbg_mem := (others => (others => '0'));
```

```
component fifo_control is
    generic (
         f_DATA_WIDTH : natural := 8;
         f_ADDRESS_WIDTH : natural := 5;
         is_write_control : boolean := true
     );
    port( clk          : in  std_logic;
          reset        : in  std_logic;
          enable       : in  std_logic;
          sync_pointer : in  std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
          pointer      : out std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
          fifo_occu    : out std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
          full_empty   : buffer std_logic;
          addr_mem     : out std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
          w_r_en       : out std_logic);
end component;

component mem_control is
    port ( wclk          : in  std_logic;
           rclk          : in  std_logic;
           reset         : in  std_logic;
           write_en      : in  std_logic;
           read_en       : in  std_logic;
           raddr         : in  std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
           waddr         : in  std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
           data_in       : in  std_logic_vector((f_DATA_WIDTH - 1) downto 0);
           data_out      : out std_logic_vector((f_DATA_WIDTH - 1) downto 0));
end component;


component fifo_sync is
    port( clk      : in std_logic;
          reset    : in std_logic;
          ptr      : in std_logic_vector(f_ADDRESS_WIDTH - 1 downto 0);
          sync_ptr : buffer std_logic_vector(f_ADDRESS_WIDTH - 1 downto 0));
end component;


begin
process(rclk)
begin
if(reset = '1') then
    empty <= '0';
elsif (rising_edge(rclk)) then
    empty <= e;
end if;

end process;

process(wclk)
begin
    if(reset = '1') then
    full <= '0';
elsif (rising_edge(wclk)) then
    full <= f;
end if;
end process;

mem_ctrl : mem_control
port map (  wclk     => wclk,
            rclk     => rclk,
            reset    => reset,
            write_en => w_en,
            read_en  => r_en,
            raddr    => raddr,
            waddr    => waddr,
            data_in  => write_data_in,
            data_out => read_data_out
            );
```

```
    mem_ctrl : mem_control
    port map (  wclk     => wclk,
                rclk     => rclk,
                reset    => reset,
                write_en => w_en,
                read_en  => r_en,
                raddr    => raddr,
                waddr    => waddr,
                data_in  => write_data_in,
                data_out => read_data_out
                );

    -- map write control ports
    write_control : fifo_control
    generic map (
     f_DATA_WIDTH => f_DATA_WIDTH,
     f_ADDRESS_WIDTH => f_ADDRESS_WIDTH,
     is_write_control => true
    )
    port map( clk => wclk,
              reset => reset,
              enable => write_enable,
              sync_pointer => rptr_sync,
              pointer => wptr,
              fifo_occu => fifo_occu_in,
              full_empty => f,
              addr_mem => waddr,
              w_r_en => w_en);

    -- map read control ports
    read_control : fifo_control
    generic map (
     f_DATA_WIDTH => f_DATA_WIDTH,
     f_ADDRESS_WIDTH => f_ADDRESS_WIDTH,
     is_write_control => false
    )
    port map( clk => rclk,
              reset => reset,
              enable => read_enable,
              sync_pointer => wptr_sync,
              pointer => rptr,
              fifo_occu => fifo_occu_out,
              full_empty => e,
              addr_mem => raddr,
              w_r_en => r_en);

    read_sync : fifo_sync
    port map( clk => rclk,
              reset => reset,
              ptr  => wptr,
              sync_ptr => wptr_sync);

    write_sync : fifo_sync
    port map( clk => wclk,
              reset => reset,
              ptr  => rptr,
              sync_ptr => rptr_sync);
end ar;
```

3.1.2 Memory Controller

```
entity mem_control is
generic (
    f_DATA_WIDTH    : natural := 8;
    f_ADDRESS_WIDTH : natural := 5
);

port(
    wclk        : in  std_logic;
    rclk        : in  std_logic;
    reset       : in  std_logic;
    write_en    : in  std_logic;
    read_en     : in  std_logic;
    raddr       : in  std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
    waddr       : in  std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
    data_in     : in  std_logic_vector((f_DATA_WIDTH - 1) downto 0);
    data_out    : out std_logic_vector((f_DATA_WIDTH - 1) downto 0)

);

end mem_control;

architecture mem of mem_control is
    type   t_Memory is array (0 to 2**(f_ADDRESS_WIDTH-1)-1)
                    of std_logic_vector((f_DATA_WIDTH-1) downto 0);

    signal Mem : t_Memory;
    file output_file : text open write_mode is "output.txt";
    signal read_ff : std_logic;

begin

    process(rclk,reset)
    begin
    if(reset = '1') then
       read_ff <= '0';
    elsif (rising_edge(rclk)) then
       read_ff <= read_en;
    end if;
    end process;

    data_out <= Mem(to_integer(unsigned(raddr(f_ADDRESS_WIDTH-2 downto 0))))
             when read_ff = '1' else (others => 'X');

    mem_write : process(wclk)
    variable line_var : line;
    begin
       if(rising_edge(wclk)) then
          if(write_en = '1') then
             Mem(to_integer(unsigned(waddr(f_ADDRESS_WIDTH-2 downto 0)))) <= data_in;
          end if;
       end if;

    end process;


end mem;
```

3.1.3 Synchronization circuit

```
entity fifo_sync is
 generic (
   f_DATA_WIDTH    : natural := 8;
   f_ADDRESS_WIDTH : natural := 5
 );

 port (
   clk         : in std_logic;
   reset       : in std_logic;
   ptr         : in std_logic_vector(f_ADDRESS_WIDTH-1 downto 0);
   sync_ptr    : buffer std_logic_vector(f_ADDRESS_WIDTH-1 downto 0) := (others => '0')
 );
end fifo_sync;

architecture ar of fifo_sync is
   signal binary_to_gray : std_logic_vector(f_ADDRESS_WIDTH-1 downto 0) := (others => '0');
   signal ff_1           : std_logic_vector(f_ADDRESS_WIDTH-1 downto 0);
   signal ff_2           : std_logic_vector(f_ADDRESS_WIDTH-1 downto 0);
begin

   gen_b2g : for i in 0 to f_ADDRESS_WIDTH-2 generate
      binary_to_gray(i) <= ptr(i) xor ptr(i+1);
   end generate;

   sync_ptr(f_ADDRESS_WIDTH-1) <= ff_2(f_ADDRESS_WIDTH-1);
   gen_g2b : for i in f_ADDRESS_WIDTH-2 downto 0 generate
      sync_ptr(i) <= sync_ptr(i+1) xor ff_2(i);
   end generate;

   process(clk,reset)
   begin
      if(reset = '1') then
         ff_1 <= (others => '0');
      elsif (rising_edge(clk)) then
         ff_1 <= binary_to_gray;
      end if;
   end process;

   process(clk,reset)
   begin
      if(reset = '1') then
         ff_2 <= (others => '0');
      elsif (rising_edge(clk)) then
         ff_2 <= ff_1;
      end if;
   end process;


end ar;
```

3.1.4 Fifo Controller

```
entity fifo_control is
    generic (
       f_DATA_WIDTH : natural := 8;
       f_ADDRESS_WIDTH : natural := 5;
       is_write_control : boolean := true
    );
    port(clk          : in  std_logic;
         reset        : in  std_logic;
         enable       : in  std_logic;
         sync_pointer : in  std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
         pointer      : out std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
         fifo_occu    : out std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
         full_empty   : buffer std_logic;
         addr_mem     : buffer std_logic_vector((f_ADDRESS_WIDTH - 1) downto 0);
         w_r_en       : out std_logic);
  end fifo_control;

architecture arch of fifo_control is
  signal lsbs_equal : std_logic;
  begin
  -- compute occupancy
   fifo_occu <= std_logic_vector(unsigned(addr_mem) - unsigned(sync_pointer));
   lsbs_equal <= '1' when sync_pointer(f_ADDRESS_WIDTH-2 downto 0) = addr_mem(f_ADDRESS_WIDTH-2 downto 0) else '0';
   pointer <= addr_mem;
   -- process to calculate empty or full
   process(sync_pointer,lsbs_equal,addr_mem)
      begin
      full_empty <= '0';
      if(is_write_control) then
       if ( lsbs_equal = '1') then
         if(sync_pointer(f_ADDRESS_WIDTH - 1) /= addr_mem(f_ADDRESS_WIDTH - 1)) then
           full_empty <= '1';
          else
           full_empty <= '0';
          end if;
       end if;
      else
         if( lsbs_equal = '1') then
            if (sync_pointer(f_ADDRESS_WIDTH - 1) = addr_mem(f_ADDRESS_WIDTH - 1))  then
             full_empty <= '1';
              else
             full_empty <= '0';
          end if;
         end if;
      end if;
  end process;


  process (enable,full_empty)
  begin
  w_r_en <= '0';
   if enable = '1' then
            if is_write_control and full_empty = '0' then
               w_r_en <= '1';
            elsif is_write_control = false and full_empty = '0' then
               w_r_en <= '1';
            else
               w_r_en <= '0';
            end if;
         end if;
  end process;
  -- process to compute on rising clock edge
  process(reset, clk)
   begin
      if reset = '1' then
         addr_mem   <= (others => '0');
      elsif rising_edge(clk) then
         if enable = '1' then
            if is_write_control and full_empty = '0' then
               addr_mem <= std_logic_vector(unsigned(addr_mem) + 1);
            elsif is_write_control = false and full_empty = '0' then
               addr_mem <= std_logic_vector(unsigned(addr_mem) + 1);
            else
               addr_mem <= addr_mem;
            end if;
         end if;
      end if;
   end process;
```

3.1.5 Testbench

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library std;
use std.textio.all;

entity async_fifo_tb is
end async_fifo_tb;

architecture tb of async_fifo_tb is
    signal wclk        : std_logic:='0';
    signal rclk        : std_Logic:='0';
    signal reset       : std_logic:='0';
    signal data_in     : std_logic_vector(7 downto 0);
    signal data_out    : std_logic_vector(7 downto 0);
    signal occu_in     : std_logic_vector(4 downto 0);
    signal occu_out    : std_logic_vector(4 downto 0);
    signal full        : std_logic;
    signal empty       : std_logic;
    signal w_en        : std_logic;
    signal r_en        : std_logic;
    constant WCLOCK_PERIOD      : time := 100 ns;
    constant RCLOCK_PERIOD      : time := 50 ns;



begin
    wclk <= not wclk after WCLOCK_PERIOD;
    rclk <= not rclk after RCLOCK_PERIOD;
    reset <= '1', '0' after  150 ns;

    dut : entity work.async_fifo
       port map (
          reset => reset,
          wclk  => wclk,
          rclk  => rclk,
          fifo_occu_in => occu_in,
          fifo_occu_out=> occu_out,
          full => full,
          empty => empty,
          write_enable => w_en,
          write_data_in => data_in,
          read_enable   => r_en,
          read_data_out => data_out);

    stimulus:
    process
     variable i : integer := 0;
     variable line_var : line;
     file output_file : text open write_mode is "tb_out.txt";
     variable data : std_logic_vector(7 downto 0);

       procedure do_write (data : std_logic_vector(7 downto 0)) is
       begin

          w_en <= '1';
          data_in <= data;
          wait for WCLOCK_PERIOD;
          wait for WCLOCK_PERIOD;
          w_en <='0';
       end do_write;
```

```
      procedure do_nwrites(
      n : integer;
      data : std_logic_vector(7 downto 0)) is
      variable i : integer := 0;
      variable line_var : line;
      variable temp :  std_logic_vector(7 downto 0);
      begin
         temp := data;
         while( i < n ) loop

               w_en <='1';
               data_in <= temp;
               wait for WCLOCK_PERIOD;
               wait for WCLOCK_PERIOD;
               i := i + 1;

               temp := std_logic_vector(unsigned(temp) + 1);

         end loop;
         w_en <= '0';
      end do_nwrites;

      procedure do_read is
      begin
         wait until (empty = '0');
         r_en <= '1';
         wait for RCLOCK_PERIOD;
         wait for RCLOCK_PERIOD;
          write(line_var, string'("Data read: "));
          write(line_var, to_integer(unsigned(data_out)));
          writeline(output, line_var);
         r_en <= '0';
      end do_read;

      procedure do_nreads(n : integer) is
      variable i : integer := 0;
      begin
         while(i < n) loop

            r_en <= '1';
            wait for RCLOCK_PERIOD;
            wait for RCLOCK_PERIOD;
            write(line_var, string'("Data read: "));
            write(line_var, to_integer(unsigned(data_out)));
            writeline(output, line_var);
            i := i +1;
            --r_en <= '0';
         end loop;
         r_en <='0';
      end do_nreads;

   begin
      wait until reset = '0';

      data := X"00";
      do_nwrites(20,data);
      do_nreads(20);
      data := X"10";
      do_write(data);
      data := X"20";
      do_write(data);
      wait until (empty = '0');
      do_nreads(2);

      assert false report "End of simulation" severity failure;
      wait;
   end process stimulus;


end tb;
```

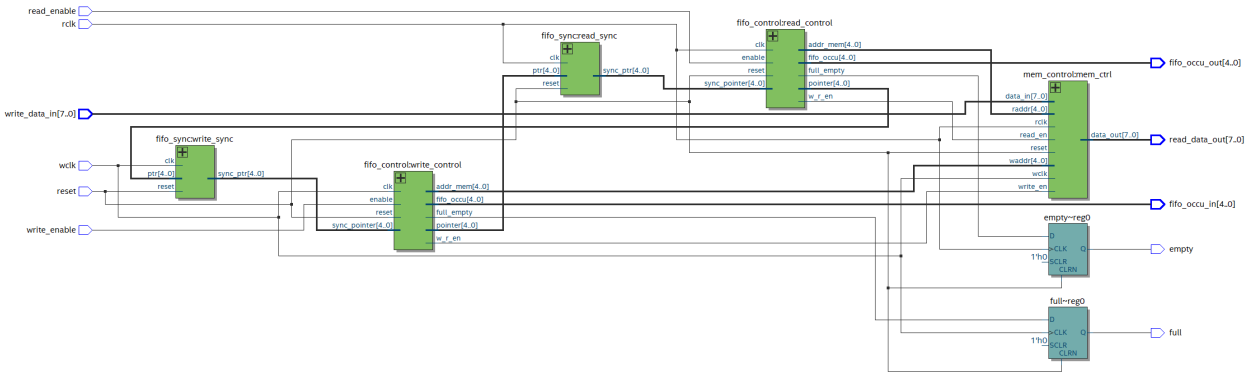
3.2 Synthesis report, Place & route report

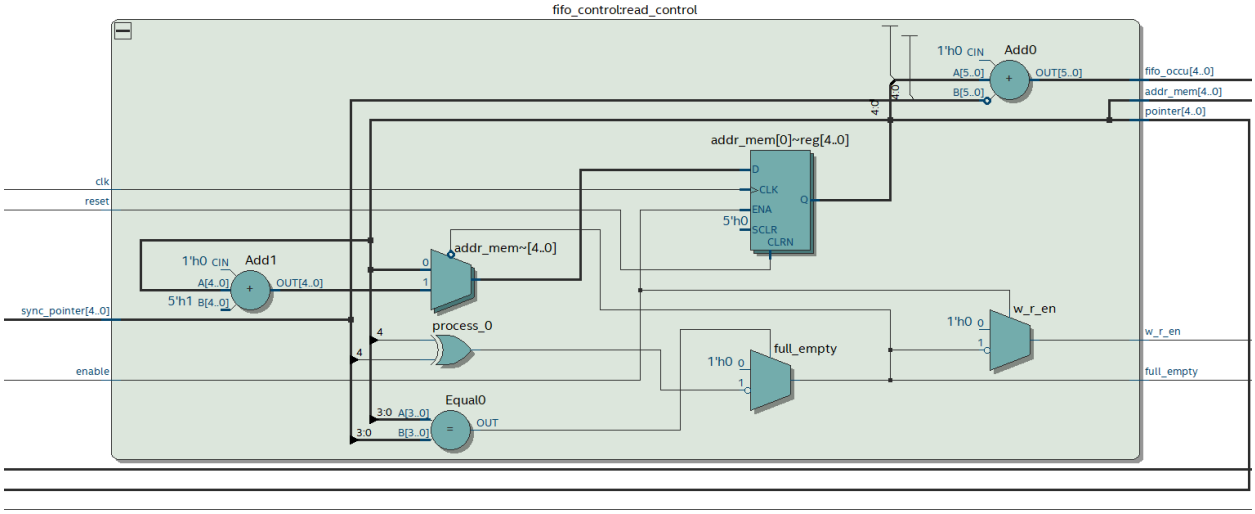
3.2.1 Synthesis

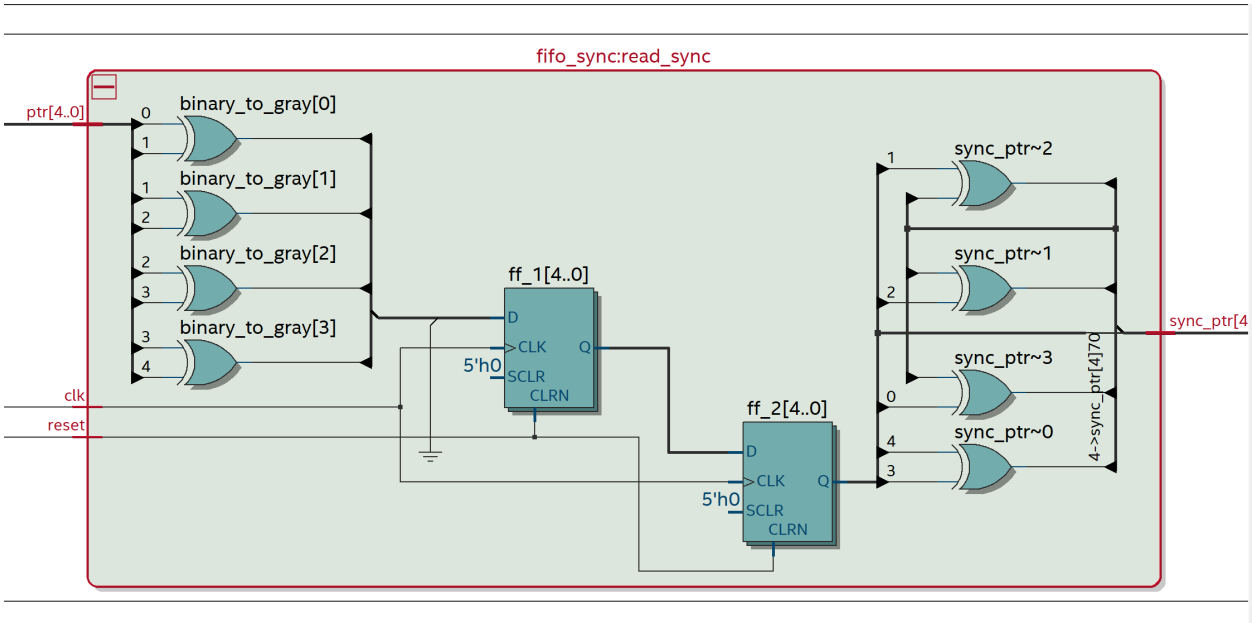
Revision Name	async_fifo
Top-level Entity Name	async_fifo
Family	Cyclone V
Logic utilization (in ALMs)	N/A
Total registers	37
Total pins	33
Total virtual pins	0
Total block memory bits	128
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Revision Name	async_fifo
Top-level Entity Name	async_fifo
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	31 / 56,480 (< 1 %)
Total registers	37
Total pins	33 / 268 (12 %)
Total virtual pins	0
Total block memory bits	128 / 7,024,640 (< 1 %)
Total RAM Blocks	1 / 686 (< 1 %)
Total DSP Blocks	0 / 156 (0 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

3.2.2 Netlists





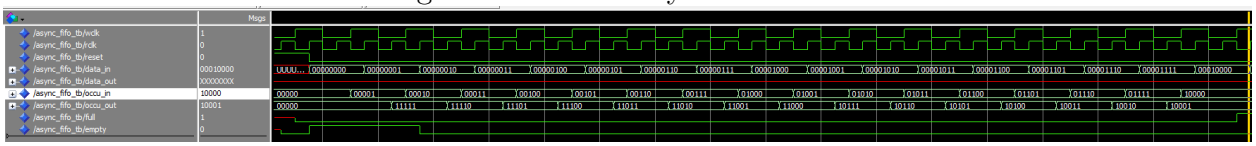


3.3 Test results

For testing we use the procedures in the testbench to perform single / multiple reads / writes. First, we perform 20 writes in a row to assert that the size of the fifo (16) is not violated. The results are as follows:

```
# Writting Addr: 0  | Value: 0
# Writting Addr: 1  | Value: 1
# Writting Addr: 2  | Value: 2
# Writting Addr: 3  | Value: 3
# Writting Addr: 4  | Value: 4
# Writting Addr: 5  | Value: 5
# Writting Addr: 6  | Value: 6
# Writting Addr: 7  | Value: 7
# Writting Addr: 8  | Value: 8
# Writting Addr: 9  | Value: 9
# Writting Addr: 10 | Value: 10
# Writting Addr: 11 | Value: 11
# Writting Addr: 12 | Value: 12
# Writting Addr: 13 | Value: 13
# Writting Addr: 14 | Value: 14
# Writting Addr: 15 | Value: 15
```

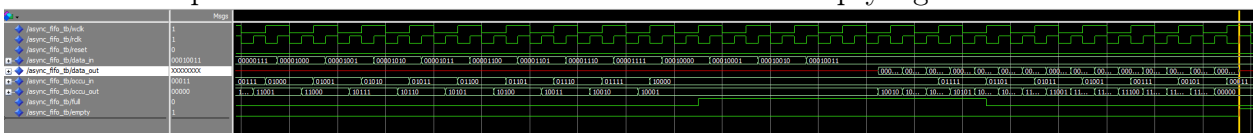
By examining the prints as well as the simulation graphs, we notice that the writes stop when the fifo is full and the full flag is raised correctly.



Afterwards, we repeated the same process but with reads:

```
# Data read: 0
# Data read: 1
# Data read: 2
# Data read: 3
# Data read: 4
# Data read: 5
# Data read: 6
# Data read: 7
# Data read: 8
# Data read: 9
# Data read: 10
# Data read: 11
# Data read: 12
# Data read: 13
# Data read: 14
# Data read: 15
```

The reads stop when we reach the end of the fifo and the empty signal is raised.



Finally, we performed 2 writes to check that the memory addresses that the data are being written to start from the beginning of the fifo since now we have an empty fifo after our reads earlier.

```
: Writting Addr: 0 | Value: 16
: Writting Addr: 1 | Value: 32
```

Readinge fifo after the writes we get the correct first value.

```
# Data read: 16
```