Assignment2

1) **Module code**

```
`timescale 1ns / 1ps

module fibonacci_generator (
    input   wire            reset,      // External reset from switch
    input   wire            CLK100MHZ   // On Board oscillator clock
);

reg     [15:0]      fib_number;
reg     [15:0]      previous;
reg     [4:0]       write_address;
reg                 write_en;

wire    [15:0]      ila_read_value;
wire     [4:0]      ila_read_address;

wire                blk_mem_rsta_busy;
wire                blk_mem_rstb_busy;

wire                clk_wiz_locked;
wire                clk_ph1;
wire                clk_ph2;

wire                fib_gen_en;

// Enable signal to generator to ensure:
//      1) clock wizard output clocks stable
//      2) block memory ports are accessible
assign fib_gen_en = clk_wiz_locked & ~blk_mem_rsta_busy & ~blk_mem_rstb_busy;

// Read <write_address-1> from PortB on clock N ( fibonacci number stored from
PortA in <write_address-1> at clock N-1)
```

```
assign ila_read_address = write_address - 1;

// Pipeline with two stages run by two clocks 180 degress out of phase
// Stage 1: Respond to clock phase 1
// Function: Fibonacci number generation and address increment
always @(posedge clk_ph1 or posedge reset)
begin
    if(reset)
    begin
       fib_number <= 0;
       previous <= 1;
       write_address <= 6'd31;
       write_en <= 1;
    end
    else if(fib_gen_en)
    begin
        fib_number <= (fib_number+previous >= fib_number) ? fib_number+previous :
1;    // Overflow check ( when fib_number > 65536)
        previous <= (fib_number+previous >= fib_number) ? fib_number : 0;
// Overflow check ( when fib_number > 65536)
        write_address <= write_address + 1;
        write_en <= 1;
    end
end

// Stage 2: Respond to clock phase 2 (180 degree out of phase wrt clock phase 1)
//Function: Storing generated number in stage 1 into memory
tdual_port_mem fibonacci_memory (
  .clka(clk_ph2),                  // input wire clka
  .rsta(reset),                    // input wire rsta
  .ena(1'b1),                      // input wire ena
  .wea(write_en),                  // input wire [0 : 0] wea
  .addra(write_address),           // input wire [4 : 0] addra
  .dina(fib_number),               // input wire [15 : 0] dina
  .douta(),                        // output wire [15 : 0] douta
  .clkb(clk_ph2),                  // input wire clkb
```

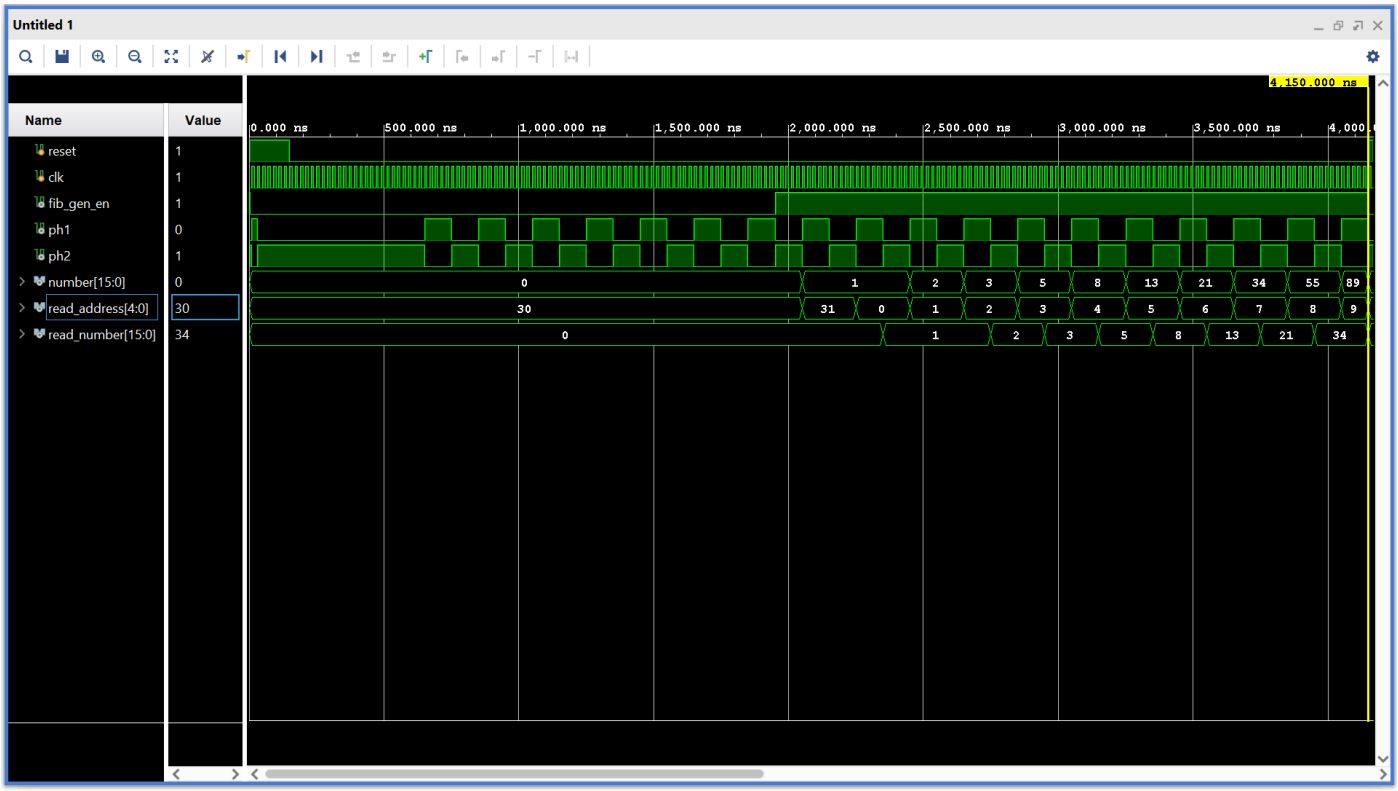
```
  .rstb(reset),                    // input wire rstb
  .enb(1'b1),                      // input wire enb
  .web(1'b0),                      // input wire [0 : 0] web
  .addrb(ila_read_address),        // input wire [4 : 0] addrb
  .dinb(),                         // input wire [15 : 0] dinb
  .doutb(ila_read_value),          // output wire [15 : 0] doutb
  .rsta_busy(blk_mem_rsta_busy),   // output wire rsta_busy
  .rstb_busy(blk_mem_rstb_busy)    // output wire rstb_busy
);

// Clock wizard:
//      Input clock 100Mhz, Output clocks 10MHz
clk_wiz_1 fibonacci_clk
   (
    // Clock out ports
    .clk_out1(clk_ph1),            // output clk_out1 - 10 MHz, 0 degree phase
shift
    .clk_out2(clk_ph2),            // output clk_out2 - 10 MHz, 180 degree phase
shift
    // Status and control signals
    .reset(reset),                 // input reset
    .locked(clk_wiz_locked),       // output locked
   // Clock in ports
    .clk_in1(CLK100MHZ)            // input clk_in1 - 100 MHz
);

// ILA:
ila_0 fibonacci_debug (
   .clk(CLK100MHZ),              // input wire clk - 100 MHz

   .probe0(fib_gen_en),          // input wire [0:0]  probe0
   .probe1(clk_ph2),             // input wire [0:0]  probe1
   .probe2(ila_read_address),    // input wire [4:0]  probe2
   .probe3(ila_read_value)       // input wire [15:0]  probe3
);
endmodule
```

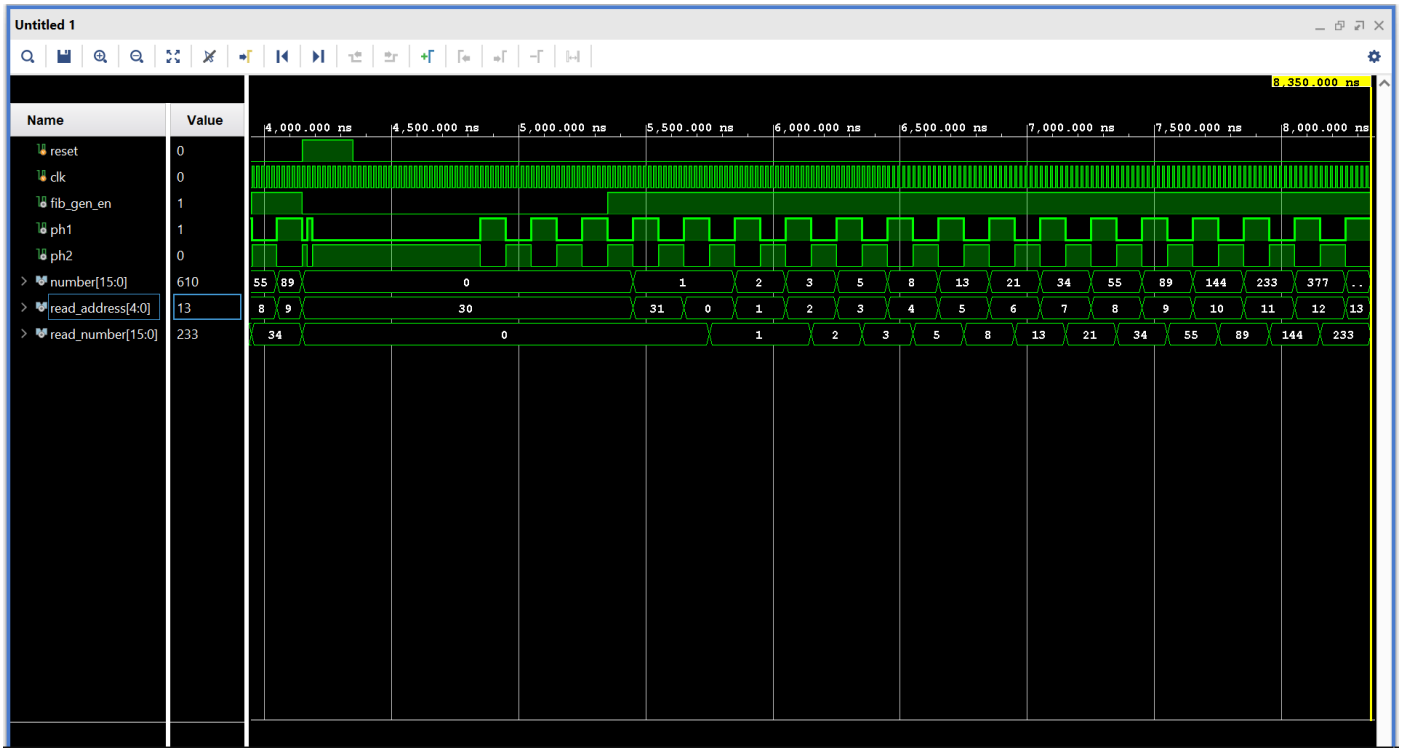
2) **Testbench code**

```
`timescale 10ns / 1ns

module fibonacci_generator_tb;

reg reset, clk;
wire    [15:0]   number;
wire    [15:0]   read_number;
wire    [4:0]    read_address;
wire             fib_gen_en;
wire             ph1, ph2;

fibonacci_generator UUT(
    .reset(reset),
    .CLK100MHZ(clk)
);

assign fib_gen_en = UUT.fib_gen_en;
assign number = UUT.fib_number;

assign read_number = UUT.ila_read_value;
assign read_address = UUT.ila_read_address;

assign ph1 = UUT.clk_ph1;
assign ph2 = UUT.clk_ph2;

initial begin
    clk = 1'b0;
    forever #1 clk = ~clk;
end

initial begin
    reset = 1;
    #15 reset = 0;
    #400 reset = 1;
```

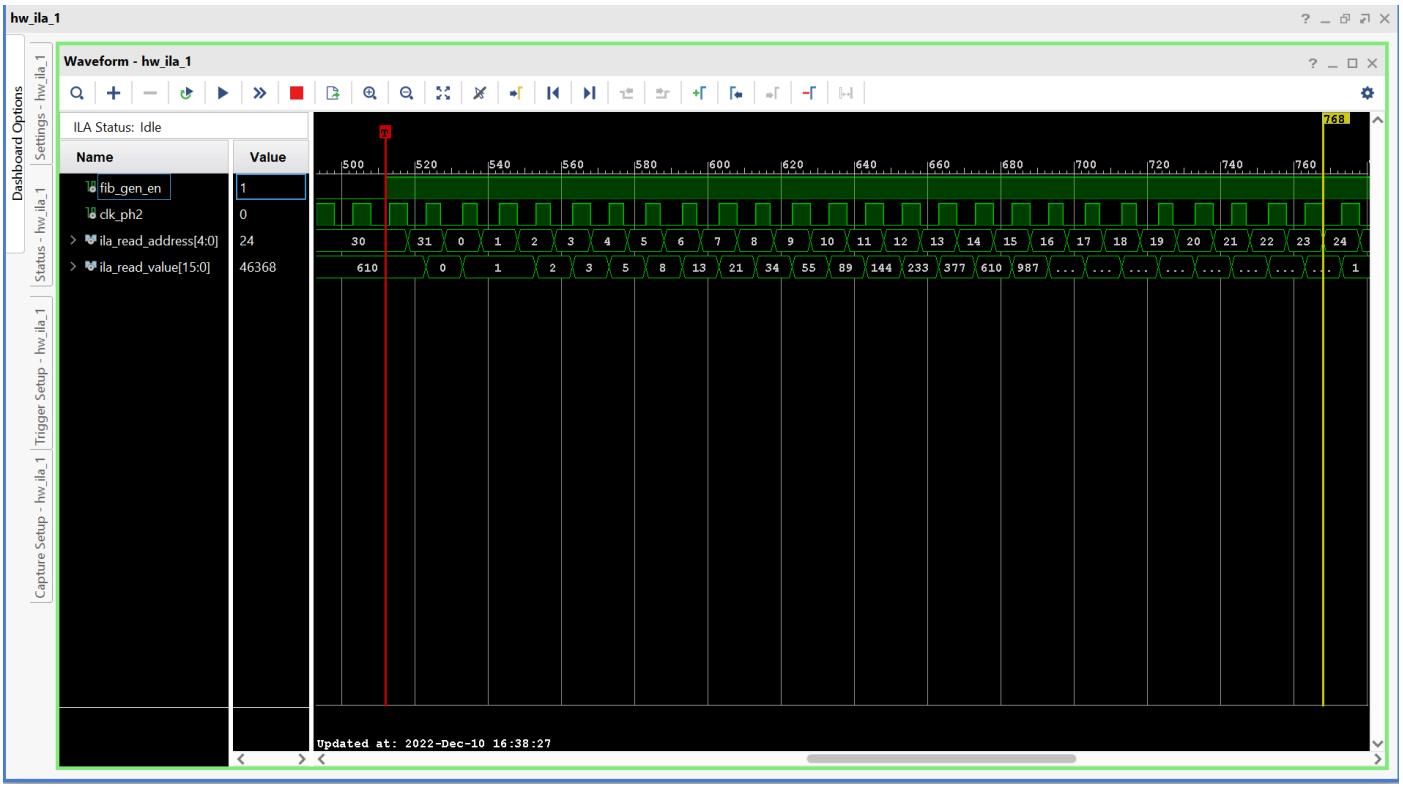
```
    #20  reset = 0;
    #400
    $finish;
end

endmodule
```

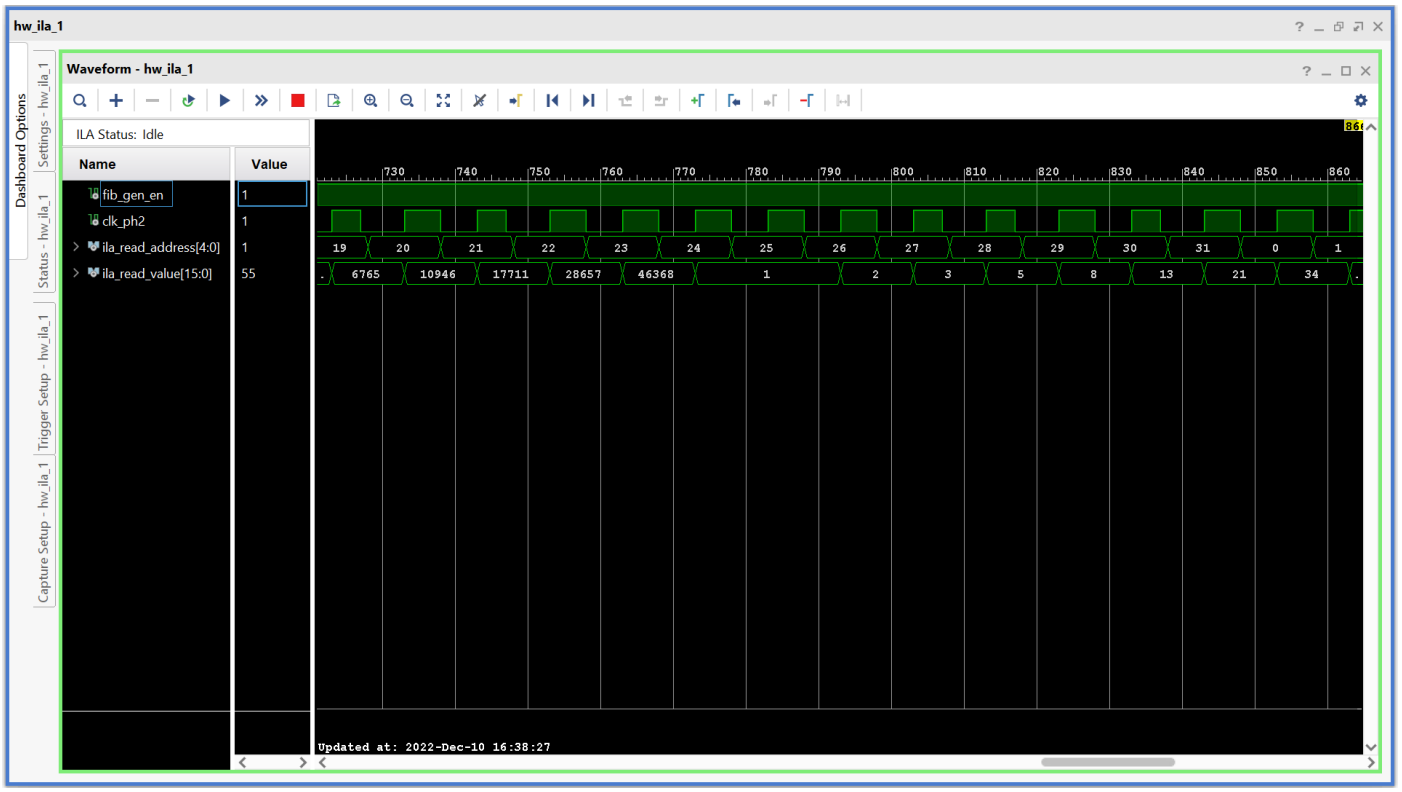
3) Timing diagram (simulation)



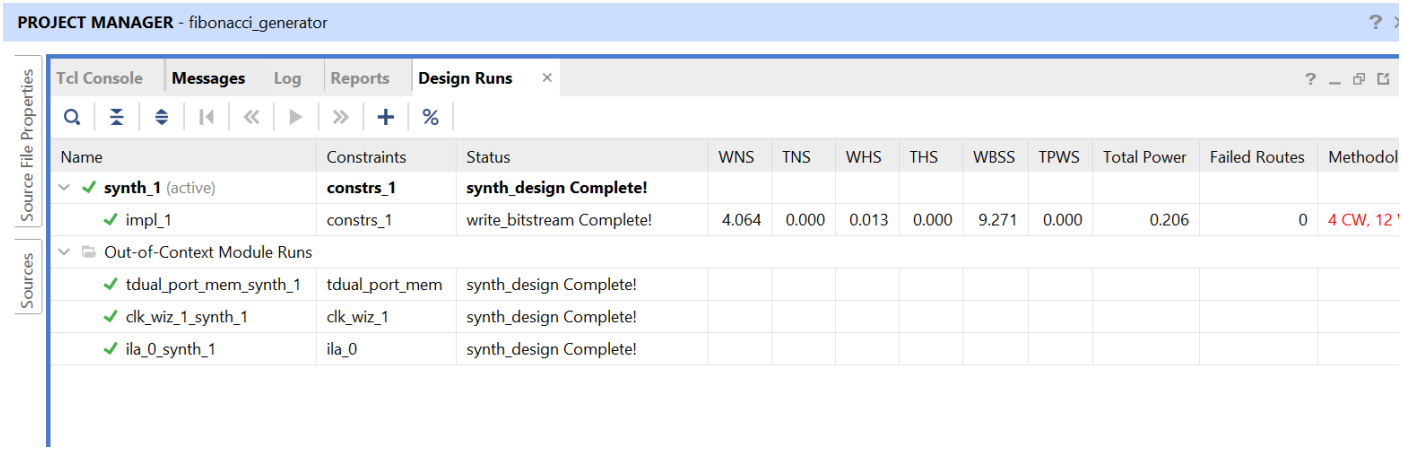


4) Timing diagram (ILA)





5) Timing report



Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodol
synth_1 (active)	constrs_1	synth_design Complete!									
impl_1	constrs_1	write_bitstream Complete!	4.064	0.000	0.013	0.000	9.271	0.000	0.206	0	4 CW, 12
Out-of-Context Module Runs											
tdual_port_mem_synth_1	tdual_port_mem	synth_design Complete!									
clk_wiz_1_synth_1	clk_wiz_1	synth_design Complete!									
ila_0_synth_1	ila_0	synth_design Complete!									

6) Constraints file

```
## This file is a general .xdc for the Nexys A7-100T

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project


## Clock signal

set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz

create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];


## Reset pin

set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { reset }]; #IO_L18P_T2_A24_15 Sch=led[0]


## Do not analyze any timing paths between clk_out2_clk_wiz_1 and sys_clk_pin ( Large path delay causes negative WHS )

## This path occurs only at ILA inputs ( violation of WHS is not critical since ILA used for signal monitoring )

set_false_path -from [get_clocks clk_out2_clk_wiz_1] -to [get_clocks sys_clk_pin];
```