گزارش کارِ هفتمِ آزمایشگاهِ معماری کامپیوتر

تهیه و تنظیم: مبین خیری

شماره دانشجویی: 994421017

استاد راهنما: دکتر حاجی‌زاده

چکیده:

در این جلسه ابتدا اقداماتِ انجام‌شده در جلسه‌ی گذشته به طور کلی مرور و سپس ادامه‌ی برنامه‌نویسیِ قسمتِ مربوط به شیفت‌رجیستر عمومی (طراحی‌شده به روشِ سری) پی گرفته شد.

بعد از پایانِ این بخش، توجه دانشجویان بر طراحیِ نرم‌افزاریِ قطعه‌ی مالتی‌پلکسر معطوف گردید.

در آغازِ جلسه و در اولین قدم، دانشجویان در گروه‌های مختلف کد زیر را در محیط نرم‌افزار شبیه‌سازی کردند:

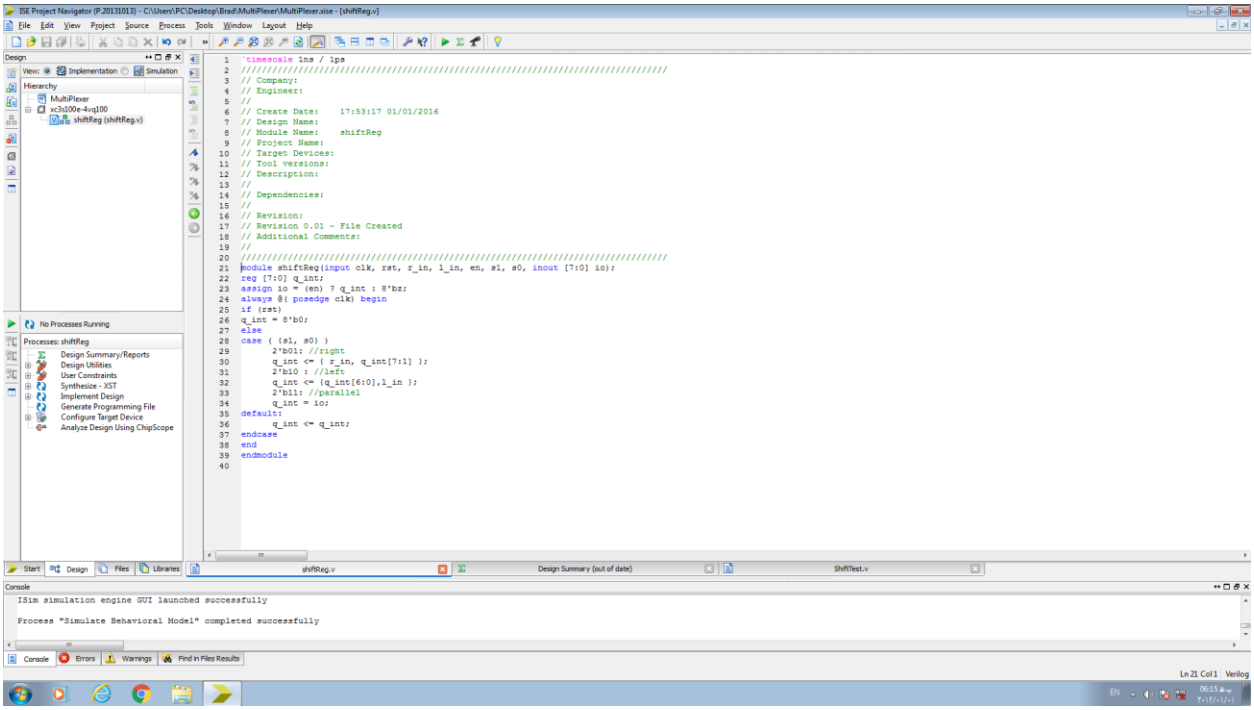
Sequential Circuit Description

Functional Registers

Universal Shift Register

```
module shift_reg (input clk, rst, r_in, l_in, en, s1, s0,
                  inout [7:0] io);
   reg [7:0] q_int;
   assign io = (en) ? q_int : 8'bz;
   always @( posedge clk ) begin
      if( rst )
         #5 q_int = 8'b0;
      else
         case ( {s1,s0} )
            2'b01 : // Shift right
            q_int <= { r_in, q_int[7:1] };
            2'b10 : // Shift left
            q_int <= { q_int[6:0], l_in };
            2'b11 : // Parallel load
            q_int = io;
            default : // Do nothing
            q_int <= q_int;
         endcase
      end

endmodule
```

نتیجه‌ی پیاده‌سازیِ این کد در محیطِ نرم‌افزارِ ISE در تصویرِ زیر آورده شده:



همچنین برای ارزیابیِ عملکردِ این قطعه لازم است که ماژولِ تستِ زیر را نوشته و آن را به کمک شبیه‌ساز اجرا کنیم.

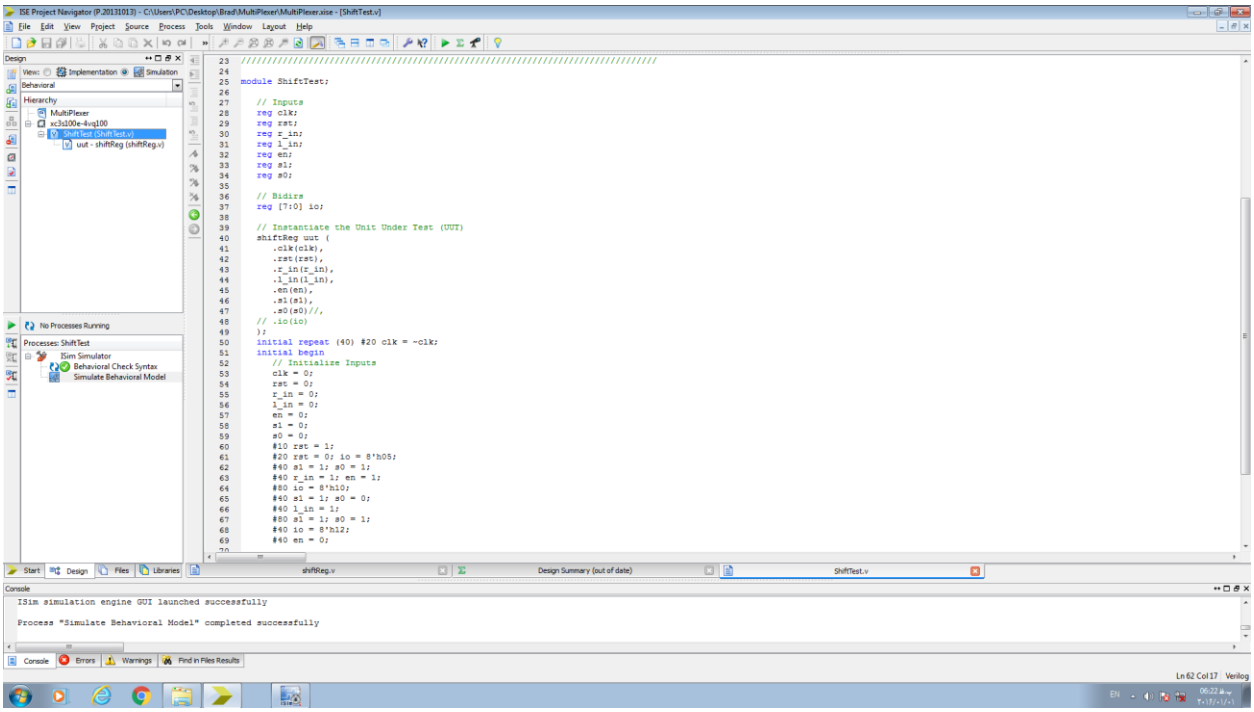
Sequential Circuit Description

Functional Registers

Universal Shift Register

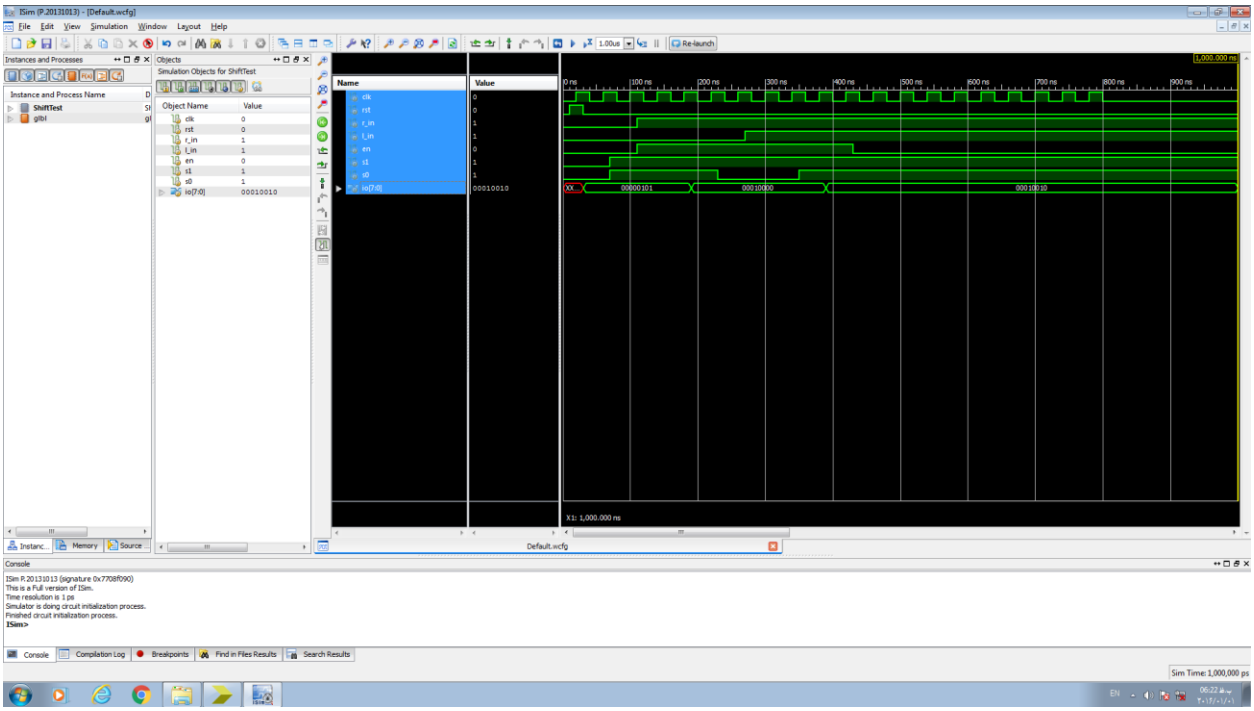
```
module tb_shift;
    // Inputs
    reg clk;
    reg rst;
    reg r_in;
    reg l_in;
    reg en;
    reg s1;
    reg s0;
    // Bidirs
    wire [7:0] io;
    // Instantiate the Unit Under Test (UUT)
    shift_reg uut (
        .clk(clk),
        .rst(rst),
        .r_in(r_in),
        .l_in(l_in),
        .en(en),
        .s1(s1),
        .s0(s0),
        .io(io) );
    initial repeat (40) #20 clk = ~clk;
    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 0;
        r_in = 0;
        l_in = 0;
        en = 0;
        s1 = 0;
        s0 = 0;
        #10 rst = 1;
        #20 rst = 0; io = 5;
        #40 s1 = 0; s0 = 1;
        #40 r_in = 1; en = 1;
        #80 io = 10;
        #40 s1 = 1; s0 = 0;
        #40 l_in = 1;
        #80 s1 = 1; s0 = 1;
        #40 io = 12;
        #40 en = 0;
    end
endmodule
```

شکلِ زیر حاصل نوشتنِ این کد در محیطِ برنامه را نشان می‌دهد:



```
//////////////////////////////////////////////////////////////////////////////////

module ShiftTest;

    // Inputs
    reg clk;
    reg rst;
    reg r_in;
    reg l_in;
    reg en;
    reg s1;
    reg s0;

    // Bidirs
    reg [7:0] io;

    // Instantiate the Unit Under Test (UUT)
    shiftReg uut (
        .clk(clk),
        .rst(rst),
        .r_in(r_in),
        .l_in(l_in),
        .en(en),
        .s1(s1),
        .s0(s0)//,
    // .io(io)
    );
    initial repeat (40) #20 clk = ~clk;
    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 0;
        r_in = 0;
        l_in = 0;
        en = 0;
        s1 = 0;
        s0 = 0;
        #10 rst = 1;
        #20 rst = 0; io = 8'h05;
        #40 s1 = 1; s0 = 1;
        #40 r_in = 1; en = 1;
        #80 io = 8'h10;
        #40 s1 = 1; s0 = 0;
        #40 l_in = 1;
        #80 s1 = 1; s0 = 1;
        #40 io = 8'h12;
        #40 en = 0;
```

در نهایت با اجرای برنامه‌ی Simulator، به نتیجه‌ی زیر دست خواهیم یافت:



در قسمتِ بعد، به کمکِ تصاویر و جدول‌هایی که در ادامه آمده‌اند، ابتدا با ساختارِ کلیِ مالتی‌پلکسرها آشنا شده و سپس نمونه‌ای از آن‌ها را طراحی خواهیم کرد.

مالتی‌پلکسر چیست؟

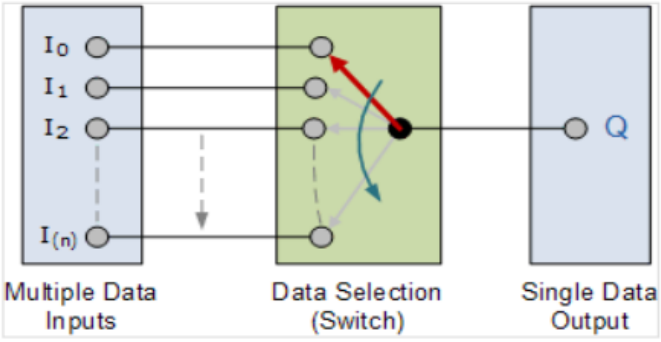
«مالتی‌پلکسر» (Multiplexer) نوعی مدار منطقی ترکیبی است که به منظور تخصیص یکی از چندین خط ورودی به تنها یک خط خروجی مشترک طراحی شده است. اینکه کدام ورودی در خروجی قرار بگیرد، توسط یک منطق کنترلی مشخص می‌شود.

روشی که در آن چند سیگنال آنالوگ یا دیجیتال را از طریق تنها یک خط انتقال مشترک و در زمان‌ها یا سرعت‌های مختلف ارسال می‌کنند Multiplexing نامیده می‌شود؛ و وسیله‌ای که این کار را انجام می‌دهد مالتی‌پلکسر (Multiplexer)نام دارد.

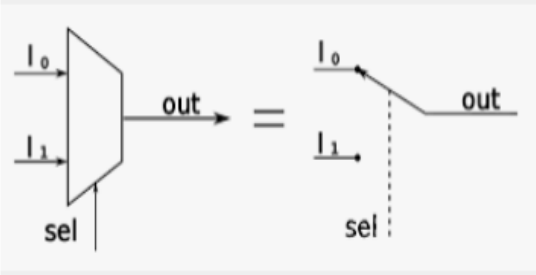
در واقع مالتی‌پلکسر و یا به اختصارMUX ، یک مدار منطقی ترکیبی است که به گونه‌ای طراحی شده تا یکی از خطوط ورودی را به یک خط خروجی مشترک سوئیچ کند. انتخاب سیگنال ورودی با استفاده از یک سیگنال کنترلی صورت می‌گیرد .هر مالتی‌پلکسر را می‌توان به صورت یک سوئیچ چرخان چندموقعیته و سریع فرض کرد، که در هر لحظه یکی از ورودی‌ها (کانال‌ها) را به خروجی وصل می‌کند.

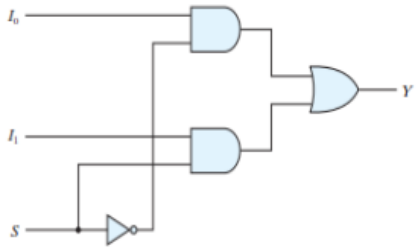
برای سوئیچ داده‌های باینری و دیجیتالی مالتی‌پلکسرها را با استفاده از گیت‌های منطقی سریع و به صورت یک مدار دیجیتالی طراحی می‌کنند. اما مالتی‌پلکسرهای آنالوگی نیز وجود دارند که با استفاده از ترانزیستورها، ماسفت‌ها (MOSFET) یا رله‌ها ساخته می‌شوند و یکی از ولتاژها یا جریان‌های ورودی را به خروجی سوئیچ می‌کنند.

Multiplexer



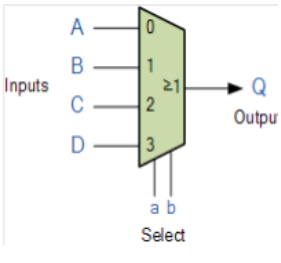
Continued

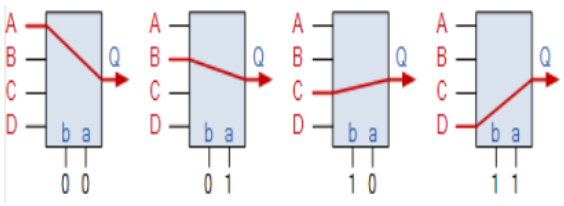


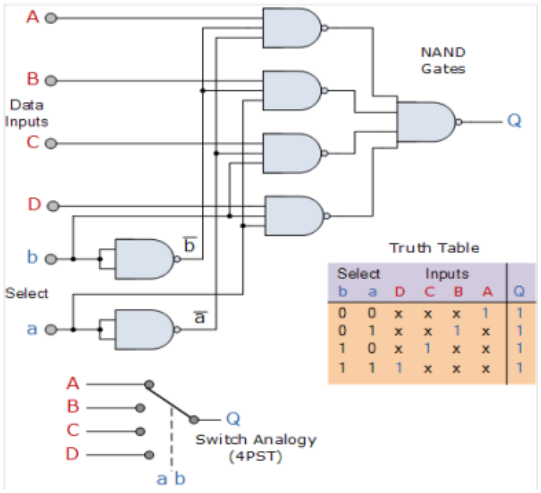


s	out
0	I0
1	I1

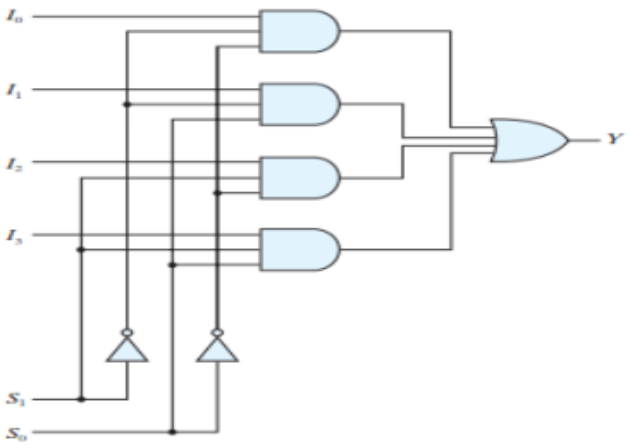
Continued







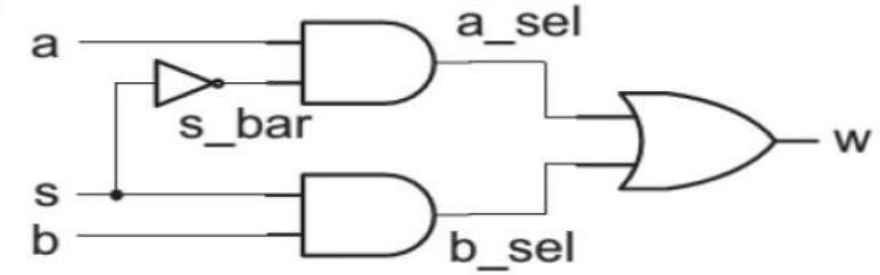
Continued



S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

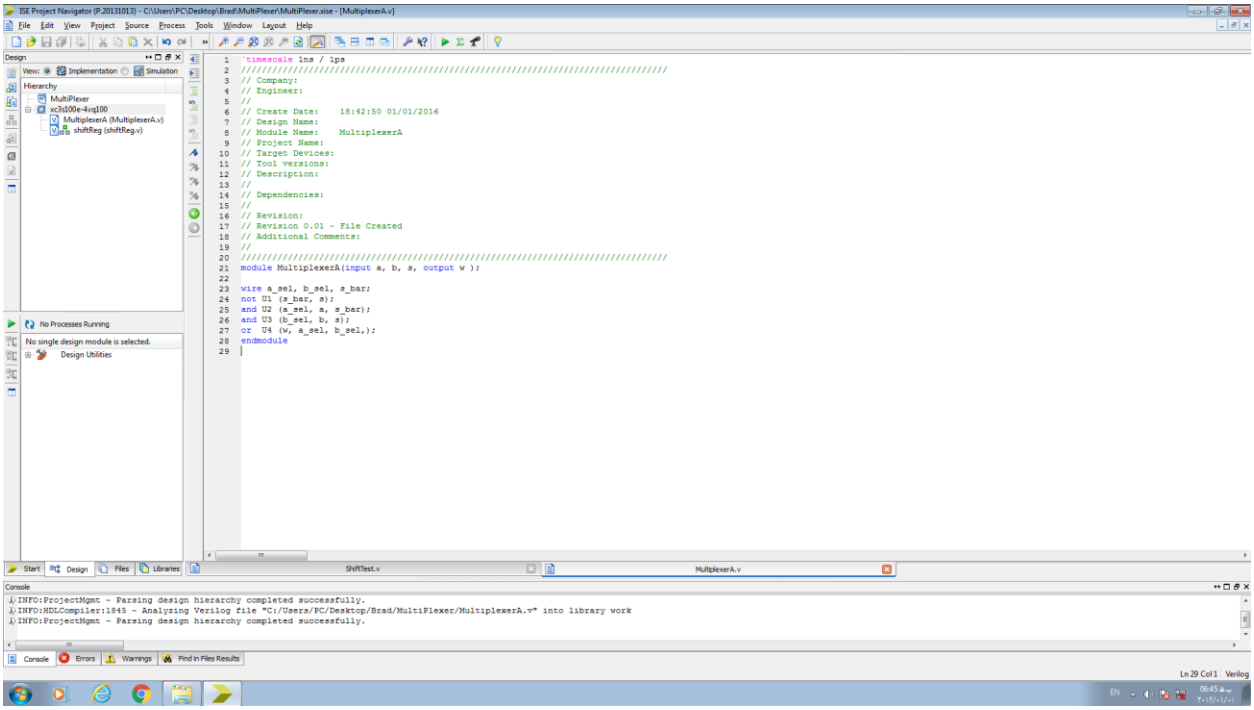
Register Transfer Level Design with Verilog

A Multiplexer Using Basic Gates

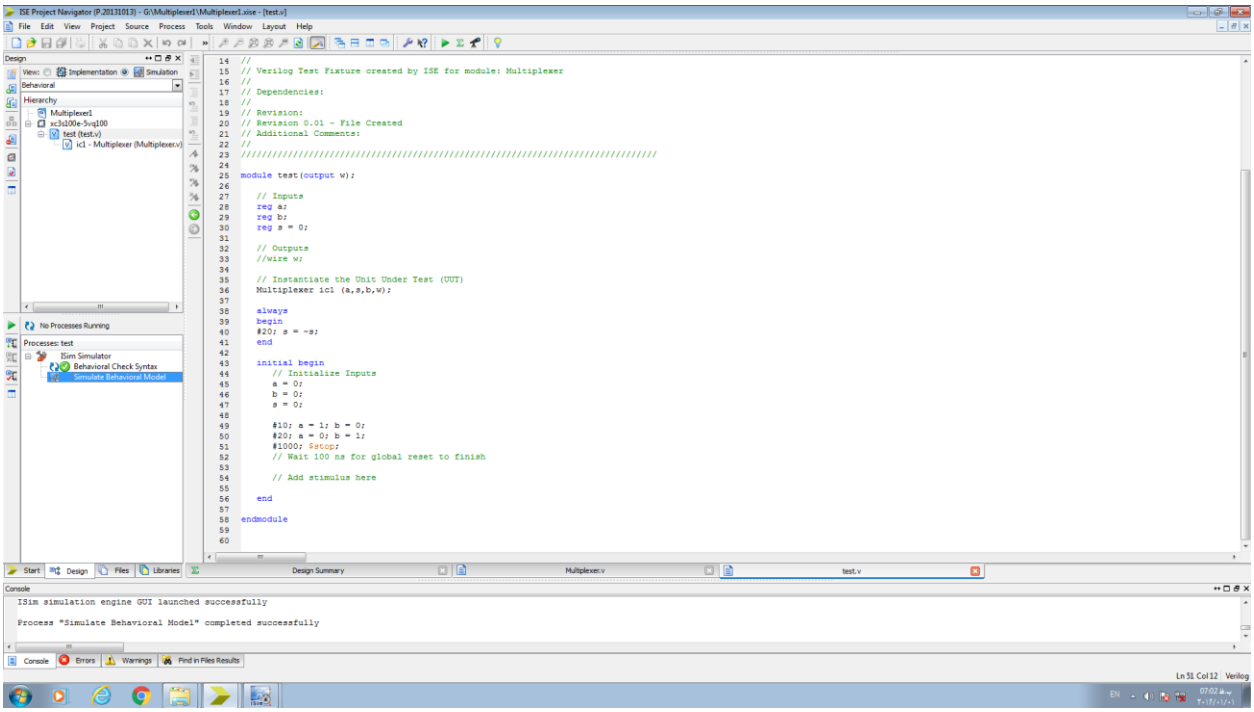


برای طراحیِ یک مالتی‌پلکسر به کمکِ گیت‌های اولیه، لازم است که مطابقِ دستورالعملِ شکلِ زیر اقدام کنیم:

حاصل نوشتنِ این برنامه در تصویر زیر آورده شده:

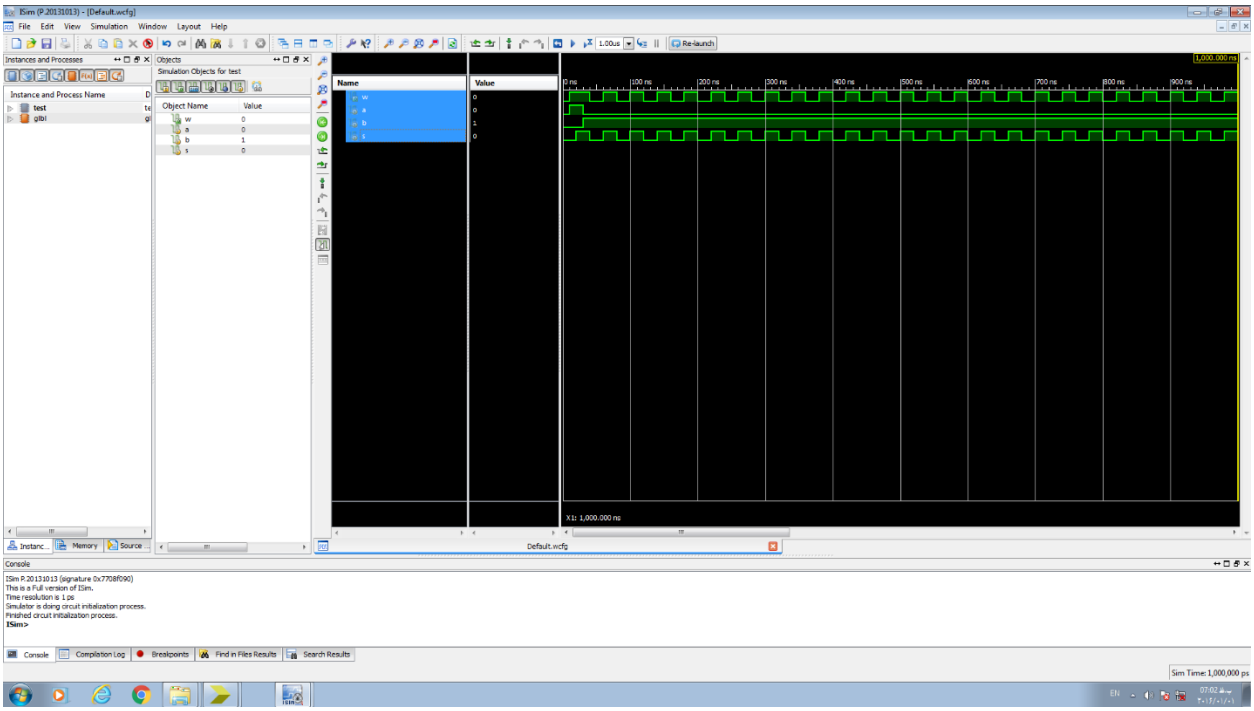


برای ارزیابیِ عملکردِ کدِ بالا، تنها کافیست دستورالعمل‌های ماژولِ تستِ زیر را به برنامه اضافه کنیم:



```
//
// Verilog Test Fixture created by ISE for module: Multiplexer
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module test(output w);

   // Inputs
   reg a;
   reg b;
   reg s = 0;

   // Outputs
   //wire w;

   // Instantiate the Unit Under Test (UUT)
   Multiplexer ic1 (a,s,b,w);

   always
   begin
   #20; s = ~s;
   end

   initial begin
      // Initialize Inputs
      a = 0;
      b = 0;
      s = 0;

      #10; a = 1; b = 0;
      #20; a = 0; b = 1;
      #1000; $stop;
      // Wait 100 ns for global reset to finish

      // Add stimulus here

   end

endmodule
```

با اجرایِ برنامه‌ی شبیه‌ساز، نتیجه‌ی نهایی بدون نقص و مشابهِ تصویر زیر خواهد بود:



پایان.