Experiment 1 - Clock and Periodic Signal Generation

***Abstract*— This document is a student report for the 1st experiment of the Digital Logic Laboratory course (ECE 045) at University of Tehran, Department of Electrical and Computer Engineering.**

***Keywords*— Clock Generation, Ring Oscillator, LM555, Schmitt Trigger Oscillator, FGPA Design, Frequency Divider, Baud Rate Generator, LTspice, Quartus, ModelSim**

I. Clock Generation using ICs and Analog Components

A. *Ring Oscillator*

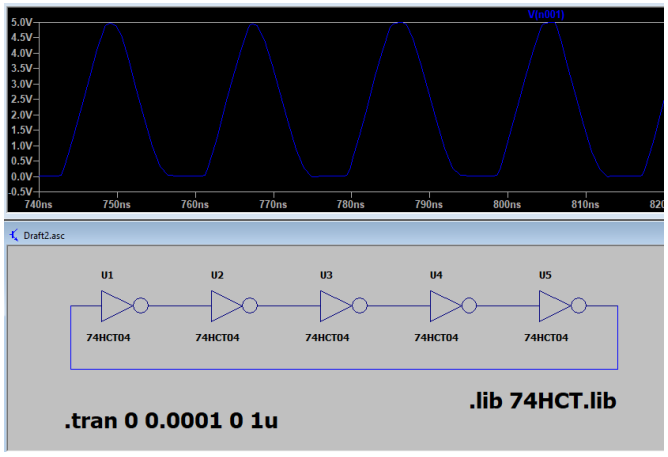

Fig. 1 Simulation waveform of a ring oscillator using 5 inverters (74HCT04)

1) *Propagation delay:* It is defined as the time from the 50% point of the input to the 50% point of the output.

Here, our input and output are connected. Therefore, the voltage waveform for all the points in the circuit are identical.

So, we will use the positive and the negative slopes of the waveform to get the propagation delay of the chain.

50% of input and output will be at +2.5v and -2.5v, referring to Fig. 1, the distance is about 8ns which is our delay.

2) The delay of a single inverter: $T = 2N*Delay_{inv}$

The period of the signal is about 18ns as shown in Fig. 1

$18ns = 2*5*Delay_{inv} \rightarrow Delay_{inv} = 1.8ns$

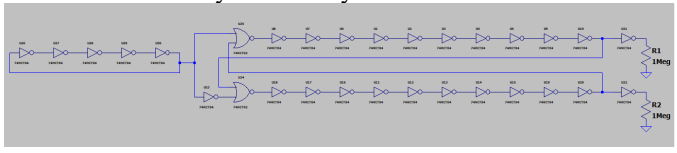

Fig. 2 Two-phase clock generator

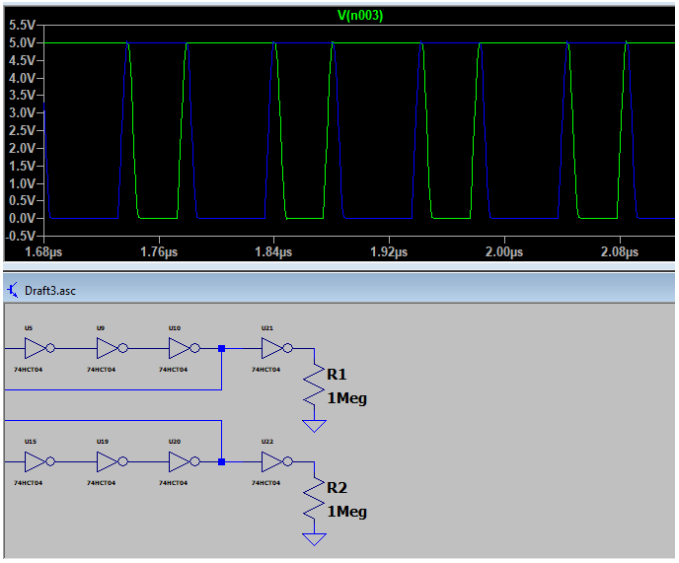

Fig. 3 Two-phase clock generator output waveform. n003 (green) is the top output (Phase1) and n005 (blue) is the bottom output (Phase2)

As shown in Fig. 3, the outputs are two non-overlapping signals which are generated from a single clock signal (the ring oscillator)

B. *LM555 Timer*

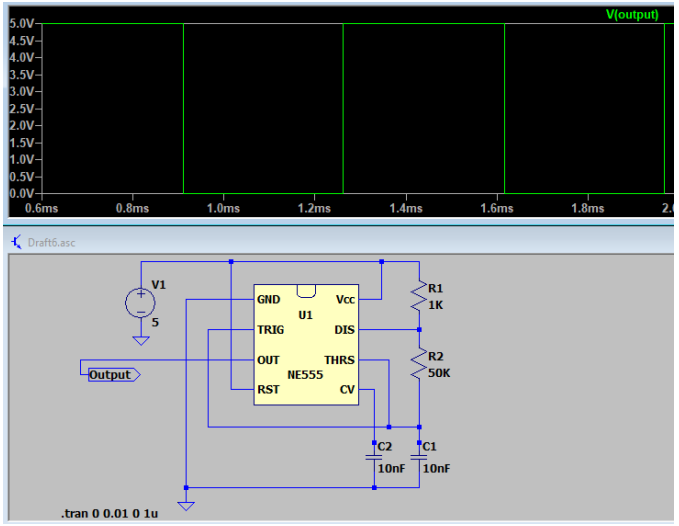

Fig. 4 LM555 in astable mode

1) From Fig. 1, we can see that the total time period of the square wave created by the LM555 IC is about 0.7ms. Therefore, the frequency (f = 1/T) is 1428.57Hz.

In each period, the signal changes at about the mid-way of the period, so the duty cycle is around 50%

We will now confirm the observations with the theoretical values:

$T = T_{charge} + T_{discharge} = 0.693 * (R_1 + 2R_2) * C$

$= 0.693 * (1 + 2*50)e3 * 10e-9 = 0.69993ms$

Duty Cycle $= (R_1+R_2) / (R_1+2R_2)$

$= (1+50) / (1 +100) = 50.5\%$

We can see that the theoretical values and our observation have the same results.

Now we will do the equations for different R2 values:

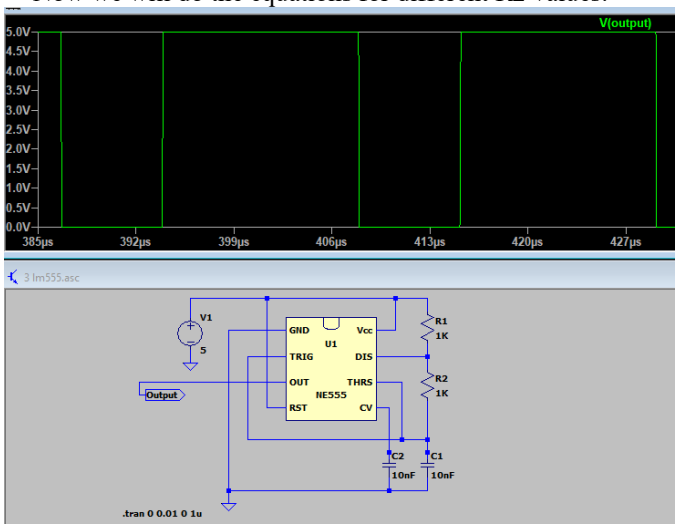

Fig. 5 R2 = 1KΩ

Observations:

T = 21us, F = 47.619KHz

Duty Cycle = 14/21 = 66.6%

Theoretical:

T = 0.693 * (1 + 2)e3 * 10e-9 = 20.79us

Duty Cycle = 2/3 = 66.6%

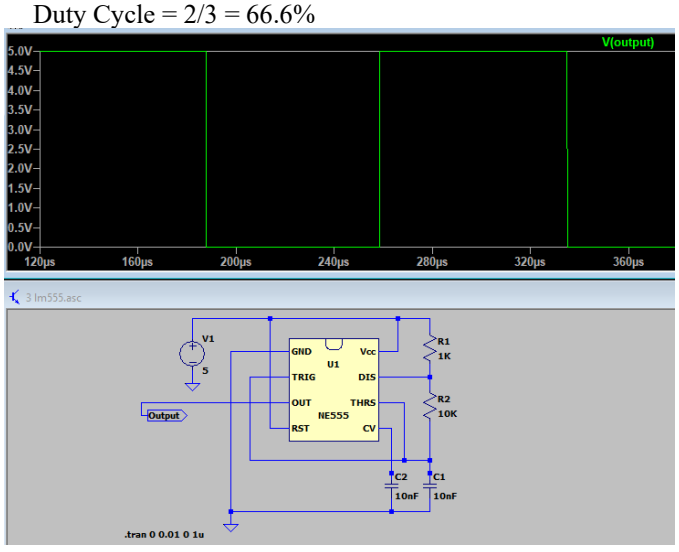

Fig. 6 R2 = 10KΩ

Observations:

T = 147us, F = 6.8KHz

Duty Cycle = 77/147 = 52.3%

Theoretical:

T = 0.693 * (1 + 20)e3 * 10e-9 = 145.5us

Duty Cycle = 11/21 = 52.3%

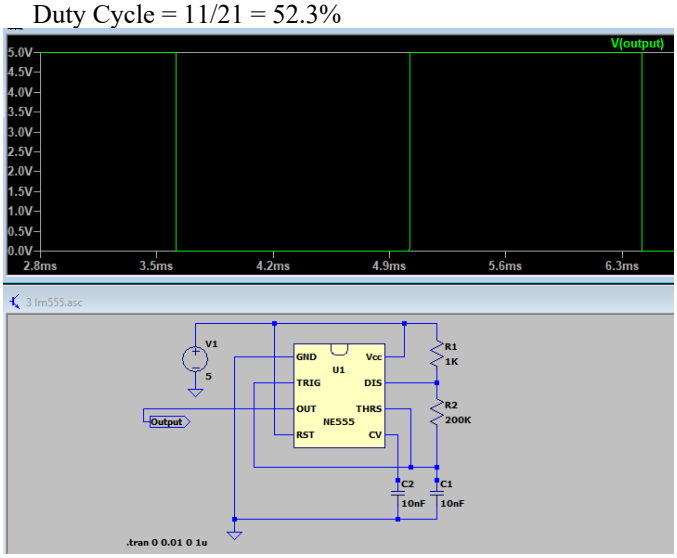

Fig. 7 R2 = 200KΩ

Observations:

T = 2.8ms, F = 375.1Hz

Duty Cycle = 1.4/2.8 = 50%

Theoretical:

T = 0.693 * (1 + 400)e3 * 10e-9 = 2.78ms

Duty Cycle = 201/401 = 50.1%

C. *Schmitt Trigger Oscillator*

f = α / RC

We will now try the circuit with different R values:

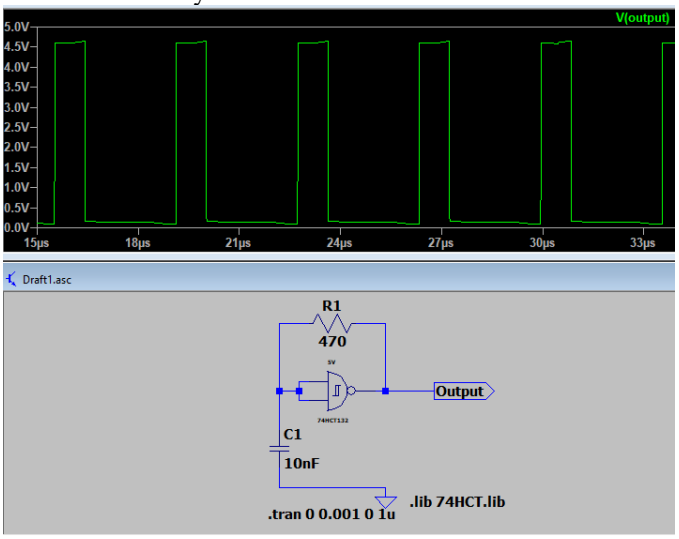

Fig. 8 R = 470Ω

T = 3.6us, F = 277.8KHz

F = α / RC, 277.8e3 = α / 470*10e-9 → α = 1.306

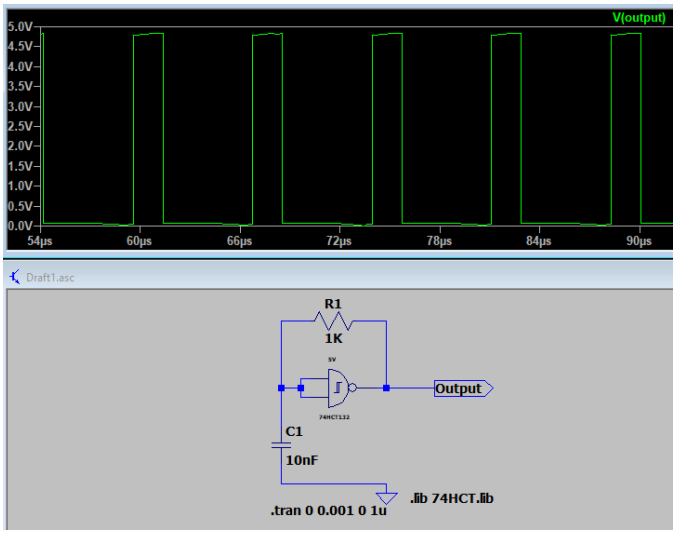

Fig. 9 R = 1KΩ

T = 7.2us, F = 138.8KHz
F = α / RC, 138.8e3 = α / e3*10e-9 → α = 1.388

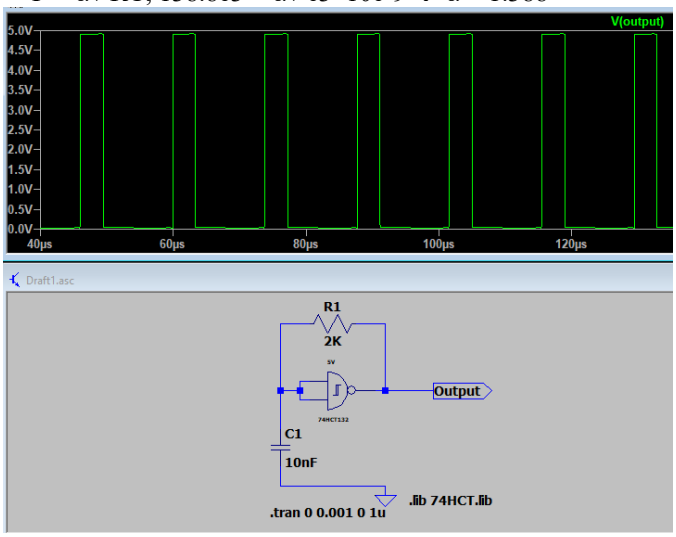

Fig. 10 R = 2KΩ

T = 14us, F = 71.4KHz
F = α / RC, 71.4e3 = α / 2e3*10e-9 → α = 1.428

So α has an average of 1.374

II. FPGA Design

A. *Ring Oscillator*

```
ring_osc.v > ...
1   `timescale 1ns/1ps
2
3   module ring_osc #(
4       parameter N = 5,
5       parameter Delay = 2
6   )(
7       output out,
8       input rst
9   );
10      wire [N:0] w;
11      assign w[0] = rst ? 1'b0 : w[N];
12
13      generate
14          genvar i;
15          for (i = 0; i < N; i = i+1) begin: Inverters
16              not #(Delay) inv(w[i+1], w[i]);
17          end
18      endgenerate
19
20      assign out = w[0];
21  endmodule
```

Fig. 11 Ring oscillator Verilog description

```
ring_osc_tb.v > ...
1   `timescale 1ns/1ps
2
3   `include "ring_osc.v"
4
5   module ring_osc_tb();
6       wire out;
7       reg rst = 1'b1;
8
9       ring_osc #(.N(5), .Delay(1.8)) osc(out, rst);
10
11      initial begin
12          #20 rst = 1'b0;
13          #200 $stop;
14      end
15  endmodule
```

Fig. 12 Ring oscillator Verilog test-bench

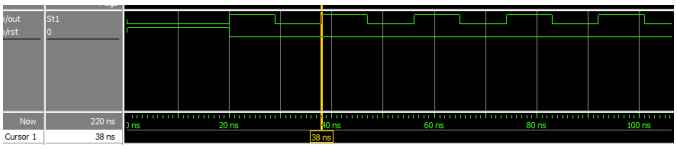

Fig. 13 Ring oscillator simulation output

As we can see, the period of the oscillator is, as expected, 18ns which will equate to 55.56MHz.

B. *Synchronous Counter as a Frequency Divider*

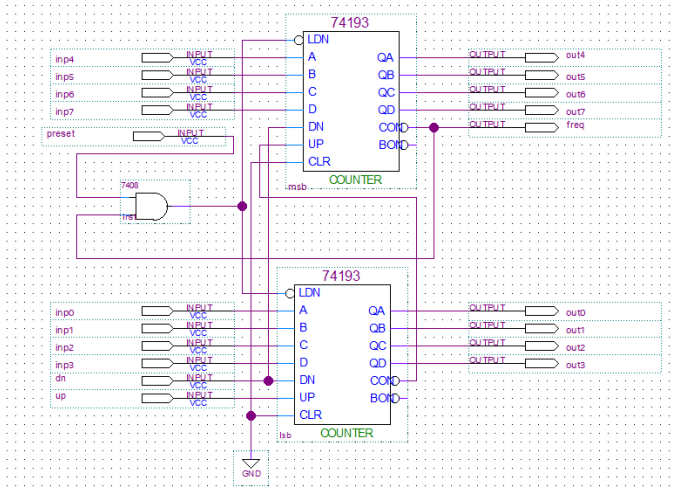

Fig. 14 Frequency divider

```
freq_divider_tb.v > ...
1   `timescale 1ns/1ps
2
3   `include "freq_divider.vo"
4
5   module freq_divider_tb();
6       reg preset, rst;
7       wire up, freq;
8       wire [7:0] i, q;
9       assign i = 8'd153;
10
11      ring_osc #(.N(5), .Delay(1.8)) osc(up, rst);
12
13      freq_divider div(.inp7(i[7]), .inp6(i[6]), .inp5(i[5]), .inp4(i[4]), .inp3(i[3]), .inp2(i[2]), .inp1(i[1]),
14                       .out7(q[7]), .out6(q[6]), .out5(q[5]), .out4(q[4]), .out3(q[3]), .out2(q[2]), .out1(q[1]),
15                       .up(up), .dn(1'b1), .preset(preset), .freq(freq));
16
17      initial begin
18          #0 {rst, preset} = 2'b10;
19          #20000 {rst, preset} = 2'b01;
20          #4000000 $stop;
21      end
22  endmodule
```

Fig. 15 Test-bench

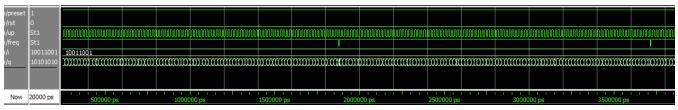

Fig. 16 Simulation results

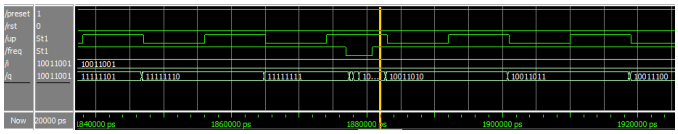

Fig. 17 When carryout becomes 0 (active-low)

The clock frequency is, as calculated before, 55.56MHz.

The counter carryout is set after 103 clocks, basically dividing the signal by 103 meaning that the frequency should be around 539.4KHz.

Observing the simulation results, the frequency is about 542.2KHz which almost matches with the theoretical value.

C. *T Flip-Flop*

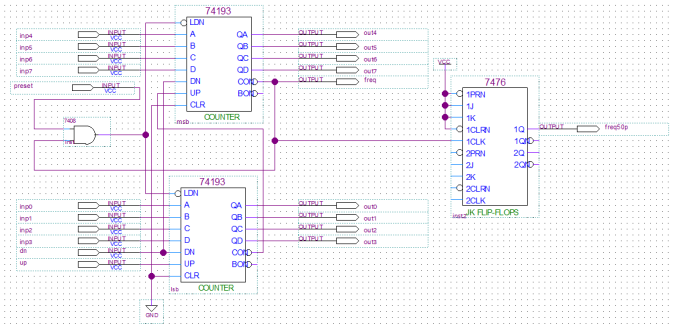

Fig. 18 Frequency divider with 50% duty cycle

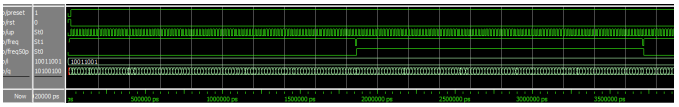

Fig. 19 Simulation results

The JK flip-flop is turned into a T flip-flop by connecting its J and K inputs.

To reach 50% duty cycle (as in Fig. 19), the TFF toggles whenever the MSB carryout is changed.

III. Baud Rate Generator for UART Serial Communication

A. *Automatic Baud Rate Calculator*

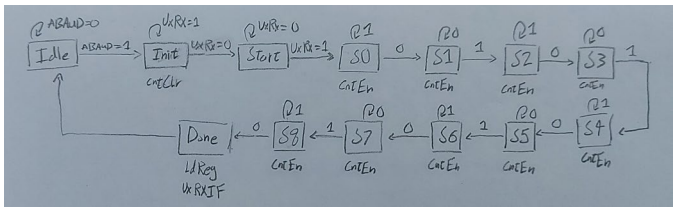

Fig. 20 Auto baud rate controller state diagram

```v
`define Idle  4'b0000
`define Init  4'b0001
`define Start 4'b0010
`define S0    4'b0011
`define S1    4'b0100
`define S2    4'b0101
`define S3    4'b0110
`define S4    4'b0111
`define S5    4'b1000
`define S6    4'b1001
`define S7    4'b1010
`define S8    4'b1011
`define Done  4'b1100

module ABRCKT_controller(
    input ABAUD, UxRX, clk,
    output reg cntClr, cntEn, UxRXIF, ldReg
);
    reg [3:0] ps, ns;

    always @(posedge clk) begin
        ps <= ns;
    end

    always @(ps, ABAUD, UxRX) begin
        case (ps)
            `Idle:  ns = ABAUD ? `Init : `Idle;
            `Init:  ns = UxRX ? `Init : `Start;
            `Start: ns = UxRX ? `S0 : `Start;
            `S0: ns = UxRX ? `S0 : `S1;
            `S1: ns = UxRX ? `S2 : `S1;
            `S2: ns = UxRX ? `S2 : `S3;
            `S3: ns = UxRX ? `S4 : `S3;
            `S4: ns = UxRX ? `S4 : `S5;
            `S5: ns = UxRX ? `S6 : `S5;
            `S6: ns = UxRX ? `S6 : `S7;
            `S7: ns = UxRX ? `S8 : `S7;
            `S8: ns = UxRX ? `Done : `S8;
            `Done: ns = `Idle;
            default: ns = `Idle;
        endcase
    end

    always @(ps) begin
        {cntClr, cntEn, ldReg, UxRXIF} = 4'd0;
        case (ps)
            `Idle:;
            `Init: cntClr = 1'b1;
            `Start:;
            `S0: cntEn = 1'b1;
            `S1: cntEn = 1'b1;
            `S2: cntEn = 1'b1;
            `S3: cntEn = 1'b1;
            `S4: cntEn = 1'b1;
            `S5: cntEn = 1'b1;
            `S6: cntEn = 1'b1;
            `S7: cntEn = 1'b1;
            `S8: cntEn = 1'b1;
            `Done: {ldReg, UxRXIF} = 2'b11;
            default:;
        endcase
    end
endmodule
```

Fig. 21 Controller Verilog description

```v
ABRCKT_datapath.v > ...
module ABRCKT_datapath(
    input cntClr, cntEn, ldReg, clk,
    output [7:0] regOut
);
    wire [7:0] bReg, cnt;
    wire co;

    register #(.N(8)) brgReg(.load(ldReg), .ldData(cnt), .out(bReg), .clr(1'b0), .clk(clk));
    counter #(.N(8)) brgCnt(.en(cntEn), .clr(cntClr), .clk(clk), .co(co), .cnt(cnt));

    assign regOut = bReg;
endmodule
```

Fig. 22 Datapath Verilog description

```v
ABRCKT.v > ...
module ABRCKT(
    input ABAUD, UxRX, clk,
    output UxRXIF,
    output [7:0] baudVal
);
    wire cntClr, cntEn, ldReg;
    wire [7:0] bReg;

    ABRCKT_datapath dp(cntClr, cntEn, ldReg, clk, bReg);
    ABRCKT_controller cu(ABAUD, UxRX, clk, cntClr, cntEn, UxRXIF, ldReg);

    assign baudVal = bReg;
endmodule
```

Fig. 23 Automatic baud rate top-level module

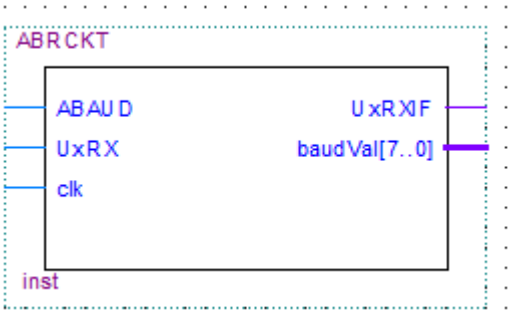

Fig. 24 Module symbol

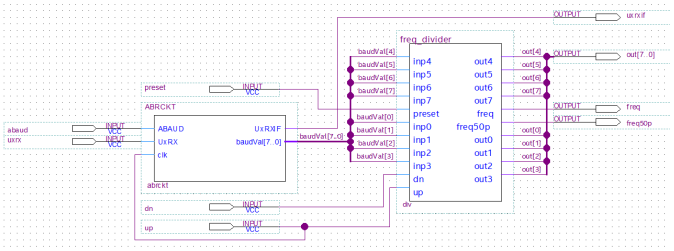

Fig. 25 BRGCKT