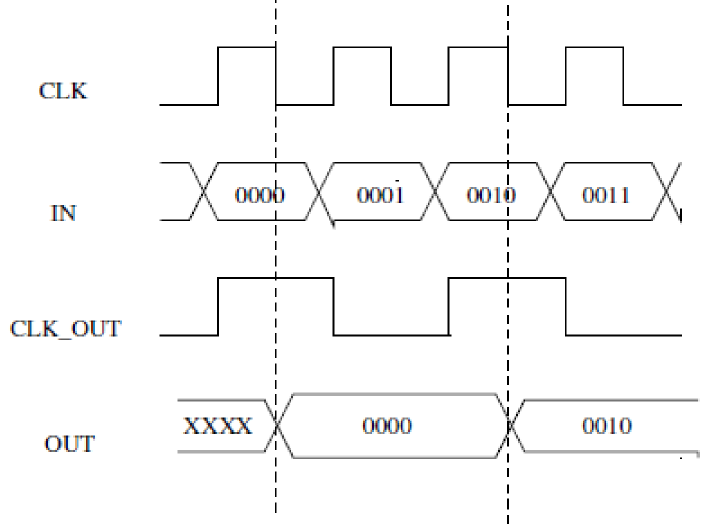
Khanh Nguyen
UIN 525000335
ECEN 449-503
Dr.Khatri

Homework 1

1. [10 points.]

   Consider an input clock signal CLK which operates at 20 MHz. Also consider a signal IN which is 4 bits wide, arriving at every positive edge of CLK. Write Verilog code whose output is the signal OUT shown below. Write a testbench to simulate your code. Test your code for 16 clock cycles, with the input IN = 0000 for the first clock cycle, 0001 for the second cycle, and so on.



Waveform for Question 1

Answer:

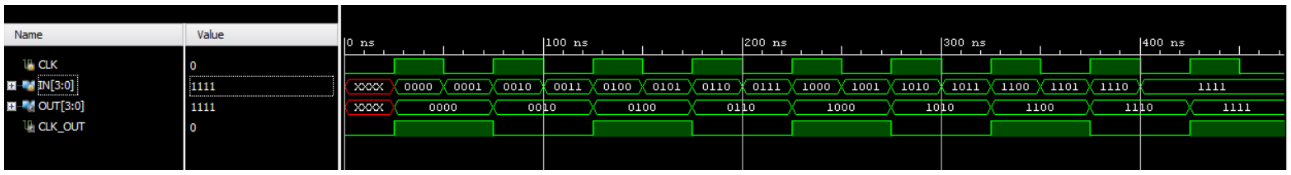
Verilog code:

```
`timescale 1ns / 1ps
`default_nettype none

module hw1_q1(CLK, IN, CLK_OUT, OUT);
        input wire CLK;
        input wire [3:0] IN;
        output reg [3:0] OUT;
        output reg [3:0] CLK_OUT;

        initial CLK_OUT = 0;
        initial OUT = 4'b0000;
        always@(posedge CLK)//CLK_OUT changes at pos edge
        begin
                CLK_OUT = ~CLK_OUT;
        end

        always@(CLK_OUT)//OUT changes at CLK_OUT
        begin
                OUT= IN;
        end

endmodule // hw1_q1
```

Test bench:

```
`timescale 1ns / 1ps
`default_nettype none
module hw1_q1_tb;
        //Inputs
        reg CLK;
        reg [3:0] IN;
        //Outputs
        wire [3:0] OUT;
        wire CLK_OUT;

        //Instantiate the Unit Under Test (UUT)
        hw1_q1 uut(
                .CLK(CLK),
                .IN(IN),
                .CLK_OUT(CLK_OUT),
                .OUT(OUT)
        );

        //generate 20MHz clock signal
        always
                #25 CLK = ~CLK; //since 2MHz is 50ns per cycle
                                                //=>#25 half cycle
        initial begin
                CLK=0;
                #25 IN=4'b0000; //inputs
                #25 IN=4'b0001;
                #25 IN=4'b0010;
                #25 IN=4'b0011;
                #25 IN=4'b0100;
                #25 IN=4'b0101;
                #25 IN=4'b0110;
                #25 IN=4'b0111;
                #25 IN=4'b1000;
                #25 IN=4'b1001;
                #25 IN=4'b1010;
                #25 IN=4'b1011;
                #25 IN=4'b1100;
                #25 IN=4'b1101;
                #25 IN=4'b1110;
                #25 IN=4'b1111;
        end

        endmodule // hw1_q1_tb
```

Result:



Result waveform

2. [10 points.]

   Write a C program that reads a character string from an input file and arranges the words in that string lexicographically (as shown below) and writes the result to an output file. The input file name is IN.txt and the output file name is OUT.txt. Both these files contain exactly one line. For example if the input file contains
   This is an example string.
   then output file should contain
   an example is string. This

   Test your code on the following strings

   (a) Hello hola how are you

   (b) The big cat ate the apple

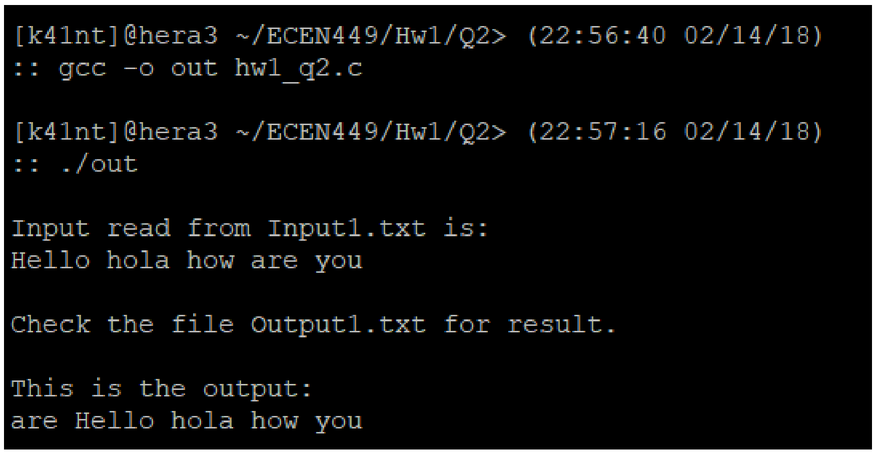
Answer:

C code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void sort(char arr[][100], int n)
{
   int i, j, min_idx;
   // One by one move boundary of unsorted subarray
   char minStr[100];
   for (i = 0; i < n-1; i++)
   {
      // Find the minimum element in unsorted array
      int min_idx = i;
      strcpy(minStr, arr[i]);
      for (j = i+1; j < n; j++)
      {
         // If min is greater than arr[j]
         if (strcasecmp(minStr, arr[j]) > 0)
         {
            // Make arr[j] as minStr and update min_idx
            strcpy(minStr, arr[j]);
            min_idx = j;
         }
      }
      // Swap the found minimum element with the first element
      if (min_idx != i)
      {
          char temp[100];
          strcpy(temp, arr[i]); //swap item[pos] and item[i]
          strcpy(arr[i], arr[min_idx]);
          strcpy(arr[min_idx], temp);
       }
   }
}



int main(){
char words[100][100]; // to save words
int i=0;
int j=0;

printf("\nInput read from Input2.txt is:\n");
// Reads input file
FILE *fptr;
fptr = fopen("Input2.txt", "r");
while(!feof(fptr)){
           fscanf(fptr,"%s",&words[i]);// read from file
           printf("%s ",words[i]); // print to screen
           i++;
}
fclose(fptr);// close file
printf("\n\n");
sort(words,i); // sort the array
printf("Check the file Output2.txt for result. \n\n");
printf("This is the output: \n");

//Writes output on screen and to file
FILE *output;
output = fopen("Output2.txt", "w");
for(j=0 ; j<i; j++){
           printf("%s ",words[j]);// print to screen
           fprintf(output,"%s ",&words[j]);// print into file
}
fclose(output);// close file
printf("\n");
return 0;
}
```

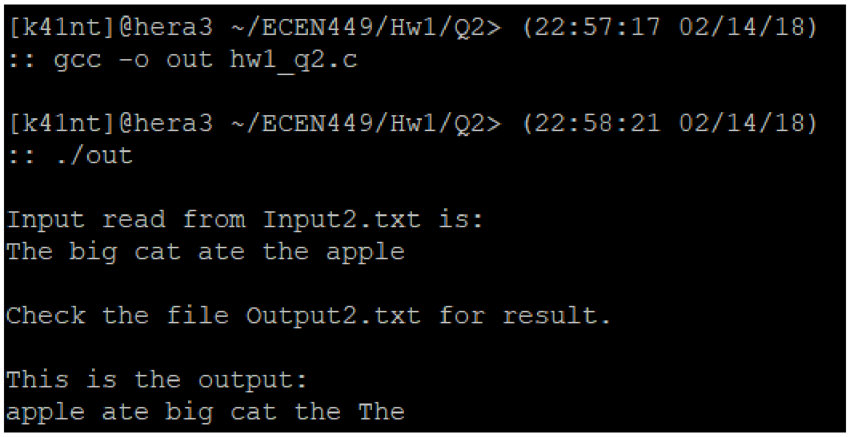
Results:

Sentence 1:



```
[k41nt]@hera3 ~/ECEN449/Hw1/Q2> (22:56:40 02/14/18)
:: gcc -o out hw1_q2.c

[k41nt]@hera3 ~/ECEN449/Hw1/Q2> (22:57:16 02/14/18)
:: ./out

Input read from Input1.txt is:
Hello hola how are you

Check the file Output1.txt for result.

This is the output:
are Hello hola how you
```

Sentence 2:



```
[k41nt]@hera3 ~/ECEN449/Hw1/Q2> (22:57:17 02/14/18)
:: gcc -o out hw1_q2.c

[k41nt]@hera3 ~/ECEN449/Hw1/Q2> (22:58:21 02/14/18)
:: ./out

Input read from Input2.txt is:
The big cat ate the apple

Check the file Output2.txt for result.

This is the output:
apple ate big cat the The
```

3. [10 points.]

Design an 8-function ALU that accepts 4-bit inputs *a* and *b*, a 3-bit input signal *select*, and produces a 5-bit output *out*. The ALU implements the following functions based on 3-bit input signal *select*.

- -

select signal function

- -

3'b000 out = a
3'b001 out = a+b
3'b010 out = a-b
3'b011 out = a/b
3'b100 out = a%b (remainder)
3'b101 out = a << 1
3'b110 out = (a≥b) (magnitude comparison)
3'b111 out = (a>b) (magnitude comparison)

You must simulate all these eight functions using a testbench. For each of these functions, test the design for 9 values in all (*a* = 3, 2, 1 and *b* = 1, 2, 3.)

Answer:

Verilog code:

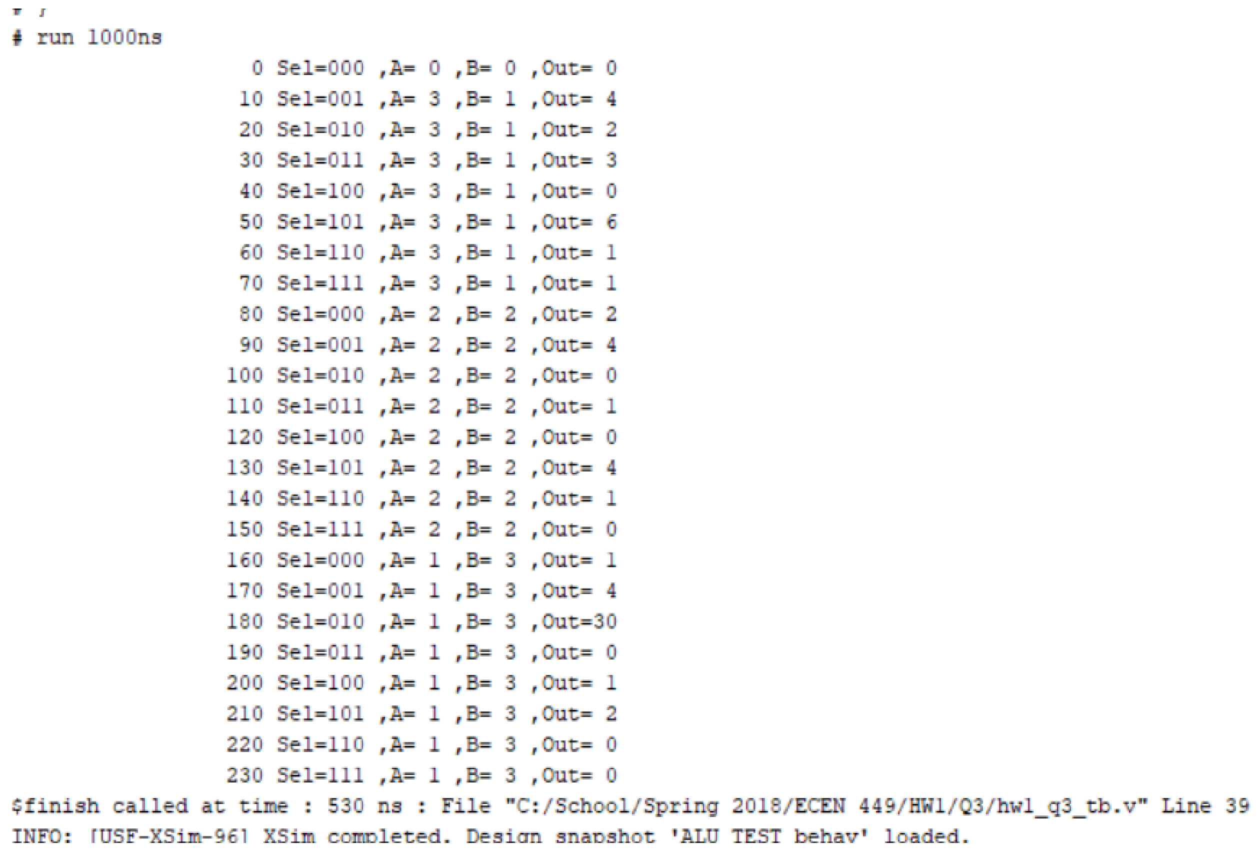
```
`timescale 1ns / 1ps
`default_nettype none

module ALU(OUT, A, B, SEL);
input [3:0] A;
input [3:0] B;
output [4:0] OUT;
input [2:0] SEL; //SELECT signal
reg [4:0] OUT;

// Arithmetic operations based on SELECT signal
always@(A or B or SEL)
  begin
    case(SEL)
      3'b000:
        OUT = A;
      3'b001:
        OUT = A+B;
      3'b010:
        OUT = A-B;
      3'b011:
        OUT = A/B;
      3'b100:
        OUT = A%B;
      3'b101:
         OUT = A<<1;
      3'b110:
         OUT = A>=B;
      3'b111:
         OUT = (A>B);
    default:
      OUT = 5'b11111;
    endcase
  end
endmodule
```

Test bench:

```
`timescale 1ns / 1ps
`default_nettype none

module ALU_TEST;
wire [4:0] out;
reg [3:0] a;
reg [3:0] b;
reg [2:0] select;

ALU ALU_test(out,a,b,select);

  initial
    begin
      $monitor($time, " Sel=%b ,A=%d ,B=%d ,Out=%d", select, a, b, out);
      select = 3'b000; a=4'b0000; b=4'b0000;
      #10 select = 3'b001; a=4'b0011; b=4'b0001;
      #10 select = 3'b010; a=4'b0011; b=4'b0001;
      #10 select = 3'b011; a=4'b0011; b=4'b0001;
      #10 select = 3'b100; a=4'b0011; b=4'b0001;
      #10 select = 3'b101; a=4'b0011; b=4'b0001;
      #10 select = 3'b110; a=4'b0011; b=4'b0001;
      #10 select = 3'b111; a=4'b0011; b=4'b0001;
      #10 select = 3'b000; a=4'b0010; b=4'b0010;
      #10 select = 3'b001; a=4'b0010; b=4'b0010;
      #10 select = 3'b010; a=4'b0010; b=4'b0010;
      #10 select = 3'b011; a=4'b0010; b=4'b0010;
      #10 select = 3'b100; a=4'b0010; b=4'b0010;
      #10 select = 3'b101; a=4'b0010; b=4'b0010;
```

```
            #10 select = 3'b110; a=4'b0010; b=4'b0010;
            #10 select = 3'b111; a=4'b0010; b=4'b0010;
            #10 select = 3'b000; a=4'b0001; b=4'b0011;
            #10 select = 3'b001; a=4'b0001; b=4'b0011;
            #10 select = 3'b010; a=4'b0001; b=4'b0011;
            #10 select = 3'b011; a=4'b0001; b=4'b0011;
            #10 select = 3'b100; a=4'b0001; b=4'b0011;
            #10 select = 3'b101; a=4'b0001; b=4'b0011;
            #10 select = 3'b110; a=4'b0001; b=4'b0011;
            #10 select = 3'b111; a=4'b0001; b=4'b0011;
        #300 $finish;
        end

endmodule
```

Result:



```
# run 1000ns
                     0 Sel=000 ,A= 0 ,B= 0 ,Out= 0
                    10 Sel=001 ,A= 3 ,B= 1 ,Out= 4
                    20 Sel=010 ,A= 3 ,B= 1 ,Out= 2
                    30 Sel=011 ,A= 3 ,B= 1 ,Out= 3
                    40 Sel=100 ,A= 3 ,B= 1 ,Out= 0
                    50 Sel=101 ,A= 3 ,B= 1 ,Out= 6
                    60 Sel=110 ,A= 3 ,B= 1 ,Out= 1
                    70 Sel=111 ,A= 3 ,B= 1 ,Out= 1
                    80 Sel=000 ,A= 2 ,B= 2 ,Out= 2
                    90 Sel=001 ,A= 2 ,B= 2 ,Out= 4
                   100 Sel=010 ,A= 2 ,B= 2 ,Out= 0
                   110 Sel=011 ,A= 2 ,B= 2 ,Out= 1
                   120 Sel=100 ,A= 2 ,B= 2 ,Out= 0
                   130 Sel=101 ,A= 2 ,B= 2 ,Out= 4
                   140 Sel=110 ,A= 2 ,B= 2 ,Out= 1
                   150 Sel=111 ,A= 2 ,B= 2 ,Out= 0
                   160 Sel=000 ,A= 1 ,B= 3 ,Out= 1
                   170 Sel=001 ,A= 1 ,B= 3 ,Out= 4
                   180 Sel=010 ,A= 1 ,B= 3 ,Out=30
                   190 Sel=011 ,A= 1 ,B= 3 ,Out= 0
                   200 Sel=100 ,A= 1 ,B= 3 ,Out= 1
                   210 Sel=101 ,A= 1 ,B= 3 ,Out= 2
                   220 Sel=110 ,A= 1 ,B= 3 ,Out= 0
                   230 Sel=111 ,A= 1 ,B= 3 ,Out= 0
$finish called at time : 530 ns : File "C:/School/Spring 2018/ECEN 449/HW1/Q3/hw1_q3_tb.v" Line 39
INFO: [USF-XSim-96] XSim completed. Design snapshot 'ALU_TEST_behav' loaded.
```

4. [10 points.]

   Consider a 12 bit serial data string DAT, which arrives at 10 Mbps. Write Verilog code to count the number of occurrences of "000" in DAT. Simulate the Verilog code using a testbench, using the following values of DAT:

   (a) 000100111000

   (b) 110001101101

   (c) 100100100000

Answer:

Verilog code:

```
module occur_counter(in, reset,clk, out, count);
        input in;
        input clk;
        input reset;
        output reg out;
        output reg [2:0] count;
        reg [1:0] stt; //state

        always @(posedge clk or negedge reset)
                if (reset==1)
                        begin
                                stt=2'b00;
                                out=0;
                                count=0;
                        end
                else
                        begin
                                case (stt)

                                        2'b00: //first state
                                        if (in==0)
                                                begin stt=2'b01;//move to next state
                                                        out=0;
                                                end
                                        else
                                                begin stt=2'b00; //otherwise stay
                                                        out=0;
                                                end

                                        2'b01: //second state
                                        if (in==0)
                                                begin stt=2'b10; //move to next state
                                                        out=0;
                                                end
                                        else
                                                begin stt=2'b00; //otherwise stay
                                                        out=0;
                                                end

                                        2'b10: // third state
                                        if (in==0)
                                                begin stt=2'b10; // third 0 caught, go back to third state
                                                        out=1; // out=1, we increment count
                                                end
                                        else
                                                begin stt=2'b00; // go back to first state
                                                        out=0;
                                                end

                                        default: stt=2'b00;

                                endcase // stt
                        end

        always @(posedge out)
                begin
                        count=count+1;
                end

endmodule // occur_counter
```

Test bench:
000100111000

```
// case 000100111000
module occur_counter_tb1;
        reg reset, clk;
        reg in;
        wire [2:0]count;
        wire out;

        occur_counter utt1(
                .in(in),
                .reset(reset),
                .clk(clk),
                .out(out),
                .count(count)
```

```
		);

	initial begin
		clk=1;
		reset=1;
		in=1;
	end // initial

	always
		#50 clk=~clk; //10Mbps=10 Mhz= 1e-7s=> half clock =50e-9s (50 units delay)

	initial begin
		#100 reset=1'b0;
		//start the DAT 000100111000
		in=1'b0;
		#100 in=1'b0;
		#100 in=1'b0;
		#100 in=1'b1;
		#100 in=1'b0;
		#100 in=1'b0;
		#100 in=1'b1;
		#100 in=1'b1;
		#100 in=1'b1;
		#100 in=1'b0;
		#100 in=1'b0;
		#100 in=1'b0;
		#100 $finish;

	end // initial

endmodule // occur_counter_tb
```

100100100000

```
// case 100100100000
module occur_counter_tb2;
	reg reset, clk;
	reg in;
	wire [2:0]count;
	wire out;
	occur_counter utt2(
		.in(in),
		.reset(reset),
		.clk(clk),
		.out(out),
		.count(count)
		);

	initial begin
		clk=1;
		reset=1;
		in=1;
	end // initial

	always
		#50 clk=~clk;

	initial begin
		#100 reset=1'b0;
		//start the DAT 100100100000
		#100 in=1'b1;
		#100 in=1'b0;
		#100 in=1'b0;
		#100 in=1'b1;
		#100 in=1'b0;
		#100 in=1'b0;
		#100 in=1'b1;
		#100 in=1'b0;
		#100 in=1'b0;
		#100 in=1'b0;
		#100 in=1'b0;
		#100 in=1'b0;
		$finish;
	end // initial

endmodule // occur_counter_tb
```

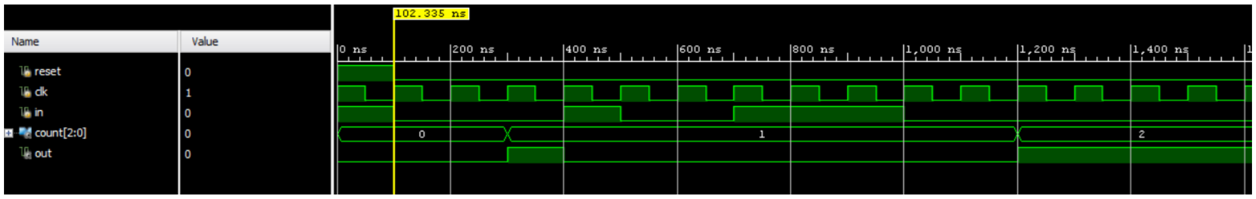
110001101101

```
// case 110001101101
module occur_counter_tb3;
	reg reset, clk;
	reg in;
	wire [2:0]count;
	wire out;

	occur_counter utt3(
		.in(in),
		.reset(reset),
		.clk(clk),
		.out(out),
		.count(count)
		);

	initial begin
		clk=1;
		reset=1;
		in=1;
	end // initial

	always
		#50 clk=~clk;

	initial begin
		#100 reset=1'b0;
		//start the DAT 110001101101
		#100 in=1'b1;
		#100 in=1'b1;
		#100 in=1'b0;
		#100 in=1'b0;
		#100 in=1'b0;
		#100 in=1'b1;
		#100 in=1'b1;
		#100 in=1'b0;
		#100 in=1'b1;
		#100 in=1'b1;
		#100 in=1'b0;
		#100 in=1'b1;
```

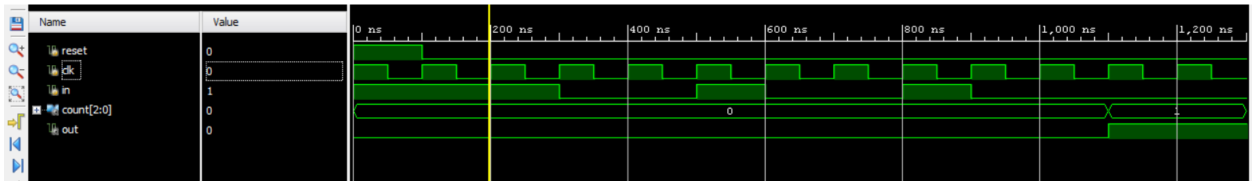
```
                    #100 $finish;
            end // initial

endmodule // occur_counter_tb
```

Result:

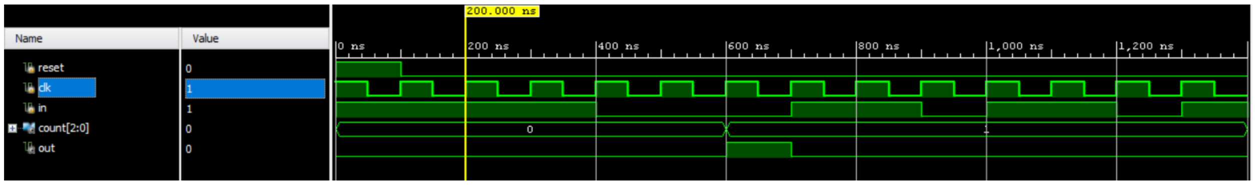
000100111000



100100100000



110001101101



5. [10 points.]

Design a behavioral Verilog module for a 4-bit synchronous counter whose count sequence is 0, 1, 2, 3, 4......... 13, 14, 15, 14, 13, 12, ..., 1, 0, 1, 2, ... The output variables are *a*, *b*, *c* and *d* (*a* being the most significant bit) with *clk* as the clock variable.

Answer:

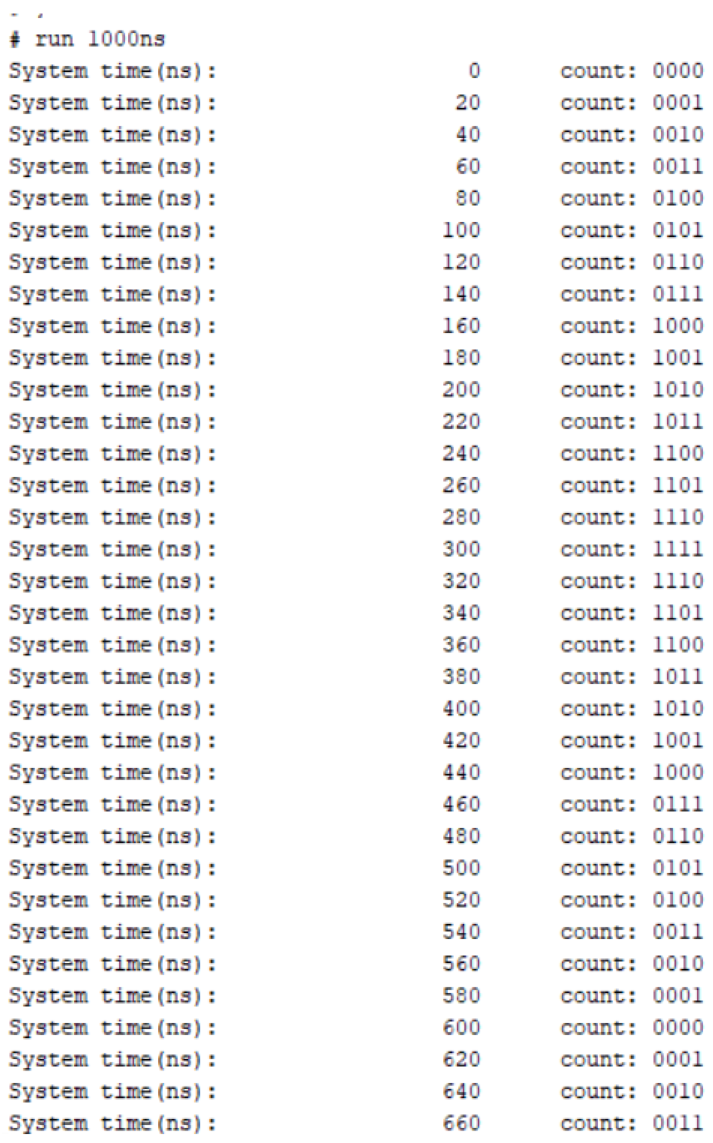
Verilog code:

```
module ctrl_4bit(a,b,c,d,clk,rst,dir); //define top entity
input clk,rst,dir; //define input signals
output a,b,c,d; //output signals
reg[3:0] ctr; //internal register
reg a,b,c,d;
initial ctr=4'b0000;

always@(posedge clk or posedge rst)
   begin
      if(rst==1'b1)
         ctr<=4'b0000;
      else if (dir==1'b0) // if dir=0, it's increasing
         ctr<=ctr+4'b0001; // increment
      else if (dir==1'b1)// if dir=1, it's decreasing
         ctr<=ctr-4'b0001; // decrement



      a <= ctr[3]; //assigning counter output
      b <= ctr[2];
      c <= ctr[1];
      d <= ctr[0];
   end
endmodule
```

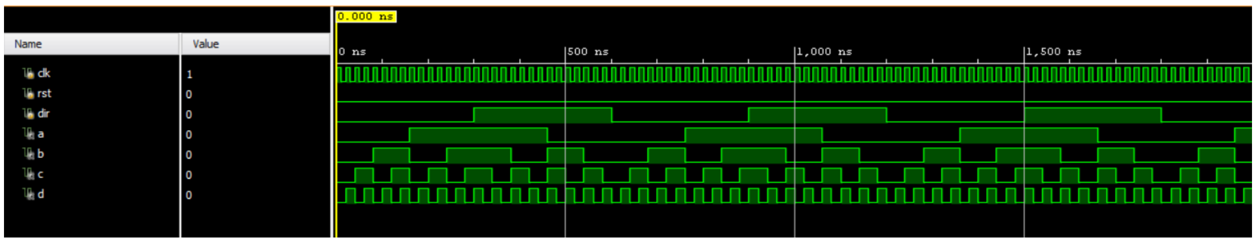
Test bench:

```
//testbench for the 4-bit synchronous counter
module testbench();
   reg clk,rst,dir;
   wire a,b,c,d;

   ctrl_4bit uut(
      .a(a),
      .b(b),
      .c(c),
      .d(d),
      .clk(clk),
      .rst(rst),
      .dir(dir)
      );

   //initializing inputs
   initial begin
      clk=1'b1;
      rst=1'b0;
      dir=1'b0;
      $monitor("System time(ns):",$time,"      count: %b%b%b%b",a,b,c,d);
      end

   //simulating inputs at various time instants
   always
      #10 clk = ~clk;



   always
      #300 dir=~dir;

endmodule
```

Result:
(next page)



```
# run 1000ns
System time(ns):                    0       count: 0000
System time(ns):                   20       count: 0001
System time(ns):                   40       count: 0010
System time(ns):                   60       count: 0011
System time(ns):                   80       count: 0100
System time(ns):                  100       count: 0101
System time(ns):                  120       count: 0110
System time(ns):                  140       count: 0111
System time(ns):                  160       count: 1000
System time(ns):                  180       count: 1001
System time(ns):                  200       count: 1010
System time(ns):                  220       count: 1011
System time(ns):                  240       count: 1100
System time(ns):                  260       count: 1101
System time(ns):                  280       count: 1110
System time(ns):                  300       count: 1111
System time(ns):                  320       count: 1110
System time(ns):                  340       count: 1101
System time(ns):                  360       count: 1100
System time(ns):                  380       count: 1011
System time(ns):                  400       count: 1010
System time(ns):                  420       count: 1001
System time(ns):                  440       count: 1000
System time(ns):                  460       count: 0111
System time(ns):                  480       count: 0110
System time(ns):                  500       count: 0101
System time(ns):                  520       count: 0100
System time(ns):                  540       count: 0011
System time(ns):                  560       count: 0010
System time(ns):                  580       count: 0001
System time(ns):                  600       count: 0000
System time(ns):                  620       count: 0001
System time(ns):                  640       count: 0010
System time(ns):                  660       count: 0011
```


0.000 ns
Name
Value
0 ns
500 ns
1,000 ns
1,500 ns
clk 1
rst 0
dir 0
a 0
b 0
c 0
d 0