LAB 5: Design Half Subtractor, Full Subtractor Using Half Subtractors

Question 1: Write a Verilog code to design a Half subtractor and test it using the Waveform

design.sv

```
module HalfSubstractor(a,b,difference,borrow);
  input a,b;
  output difference,borrow;
  xor(difference,a,b);
  assign borrow=(~a&b);
endmodule
```

testbench.sv

SV/Verilog Testbench

```
module TestModule;
  // Inputs
  reg a;
  reg b;

  // Outputs
  wire difference;
  wire borrow;

  HalfSubstractor uut (.a(a),.b(b), .difference(difference),
.borrow(borrow));
  initial begin
    // Initialize Inputs
    a = 0;
    b = 0;
    // Wait 100 ns for global reset to finish
    #100;
    a = 1;
    b = 0;
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end
endmodule
```

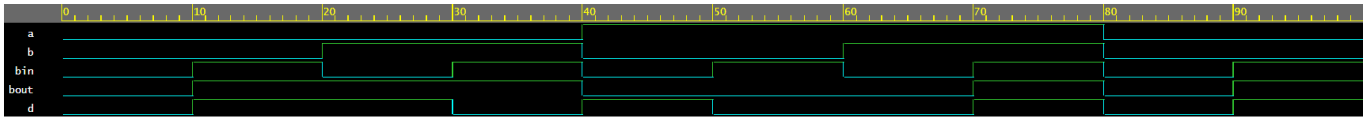
Question 2: Write a Verilog code to design a Full subtractor using Half subtractors and test it using the Waveform.

testbench.sv

```
module top;
  reg a, b, bin;
  wire d, bout;

  full_subtractor instantiation(.A(a), .B(b), .Bin(bin), .D(d), .Bout(bout));

  initial
    begin
      $dumpfile("xyz.vcd");
      $dumpvars;
      a=0;
      b=0;
      bin=0;
      #100 $finish;
    end
  always #40 a=~a;
  always #20 b=~b;
  always #10 bin=~bin;
  always @(a or b or bin)
    $monitor("At TIME(in ns)=%t, A=%d B=%d Bin=%d D = %d Bout = %d", $time, a, b, bin, d, bout);
  endmodule
```

design.sv

```
module or_gate(a0, b0, c0);
  input a0, b0;
  output c0;
  assign c0 = a0 | b0;
endmodule

module xor_gate(a1, b1, c1);
  input a1, b1;
  output c1;
  assign c1 = a1 ^ b1;
endmodule

module and_gate(a2, b2, c2);
  input a2, b2;
  output c2;
  assign c2 = a2 & b2;
endmodule

module not_gate(a3, b3);
  input a3;
  output b3;
  assign b3 = ~ a3;
endmodule

module half_subtractor(a4, b4, c4, d4);
  input a4, b4;
  output c4, d4;
  wire x;
  xor_gate u1(a4, b4, c4);
  and_gate u2(x, b4, d4);
  not_gate u3(a4, x);
endmodule

module full_subtractor(A, B, Bin, D, Bout);
  input A, B, Bin;
  output D, Bout;
  wire p, q, r;
  half_subtractor u4(A, B, p, q);
  half_subtractor u5(p, Bin, D, r);
  or_gate u6(q, r, Bout);
endmodule
```



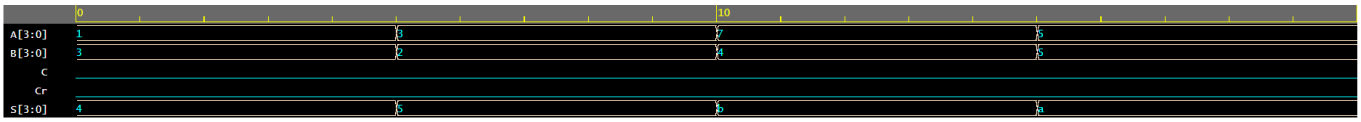
Question 3: Write the Verilog code (either structural or behavioural) for a 4-bit ripple carry adder. The block diagram of a 4-bit ripple carry adder has been given in Figure 1 for your reference.

design.sv

```
module FA (a,b,c,s,cr);
  input a,b,c;
  output s,cr;
  wire s1,c1,c2;

  xor(s1, a, b);
  xor(s, s1, c);
  and(c1, a, b);
  and(c2, s1, c);
  or(cr, c1, c2);
endmodule

module RCA(x,y,z,sum,cout);
  input [3:0]x,y;
  input z;
  output [3:0]sum;
  output cout;
  wire w1,w2,w3;

  FA f0(x[0], y[0], z, sum[0], w1);
  FA f1(x[1], y[1], w1, sum[1], w2);
  FA f2(x[2], y[2], w2, sum[2], w3);
  FA f3(x[3], y[3], w3, sum[3], cout);
endmodule
```

testbench.sv

```
module tb_RCA;
  reg [3:0]A,B;
  reg C;
  wire [3:0]S;
  wire Cr;

  RCA rca1(.x(A), .y(B), .z(C), .sum(S), .cout(Cr));
  initial
  begin
    A = 1; B = 3; C = 0; #5;
    A = 3; B = 2; C = 0; #5;
    A = 7; B = 4; C = 0; #5;
    A = 5; B = 5; C = 0; #5;
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end
endmodule
```



Question 4: Write the Verilog code to construct a 4-bit adder-subtractor using. The logic diagram of a 4-bit adder-subtractor using a ripple carry adder has been given in Figure 2 for your reference.

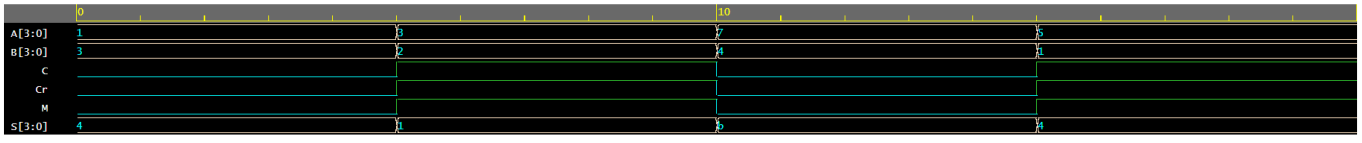
design.sv

```
module FA (a,b,c,s,cr);
  input a,b,c;
  output s,cr;
  wire s1,c1,c2;

  xor(s1, a, b);
  xor(s, s1, c);
  and(c1, a, b);
  and(c2, s1, c);
  or(cr, c1, c2);
endmodule

module FBAS(x,y,c0,m,sum,cout);
  input [3:0]x,y;
  input c0,m;
  output [3:0]sum;
  output cout;

  FA f0(x[0], y[0]^m, c0, sum[0], c1);
  FA f1(x[1], y[1]^m, c1, sum[1], c2);
  FA f2(x[2], y[2]^m, c2, sum[2], c3);
  FA f3(x[3], y[3]^m, c3, sum[3], cout);
endmodule
```

testbench.sv

SV/Verilog Testbench

```
module tb_FBAS;
  reg [3:0]A,B;
  reg C,M;
  wire [3:0]S;
  wire Cr;

  FBAS fbas1(.x(A), .y(B), .c0(C), .m(M), .sum(S), .cout(Cr));
  initial
  begin
    A = 1; B = 3; C = 0; M = 0; #5; // Adder
    A = 3; B = 2; C = 1; M = 1; #5; // Subtractor
    A = 7; B = 4; C = 0; M = 0; #5; // Adder
    A = 5; B = 1; C = 1; M = 1; #5; // Subtractor
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end
endmodule
```



Question 5: Write the Verilog code for a 4-bit carry look-ahead adder (CLA). The block diagram of the same has been given in Figure 3.

design.sv

SV/Verilog Design

```
module CLA_Adder(a,b,cin,sum,cout);
  input[3:0] a,b;
  input cin;
  output [3:0] sum;
  output cout;
  wire p0,p1,p2,p3,g0,g1,g2,g3,c1,c2,c3,c4;

  assign p0=(a[0]^b[0]),
    p1=(a[1]^b[1]),
    p2=(a[2]^b[2]),
    p3=(a[3]^b[3]);

  assign g0=(a[0]&b[0]),
    g1=(a[1] & b[1]),
    g2=(a[2] & b[2]),
    g3=(a[3] & b[3]);

  assign c0=cin,
    c1=g0|(p0 & cin),
    c2=g1|(p1 & g0)|(p1 & p0 & cin),
    c3=g2|(p2 & g1)|(p2 & p1 & g0)|(p1 & p1 & p0 & cin),
    c4=g3|(p3 & g2)|(p3 & p2 & g1)|(p3 & p2 & p1 & g0)|(p3 & p2 & p1 & p0 & cin);

  assign sum[0]=p0^c0,
    sum[1]=p1^c1,
    sum[2]=p2^c2,
    sum[3]=p3^c3;
  assign cout=c4;
endmodule
```

testbench.sv

```
module TestModule;
  // Inputs
  reg [3:0] a;
  reg [3:0] b;
  reg cin;

  // Outputs
  wire [3:0] sum;
  wire cout;

  // Instantiate the Unit Under Test
  CLA_Adder uut (
    .a(a),
    .b(b),
    .cin(cin),
    .sum(sum),
    .cout(cout)
  );
  initial begin
    // Initialize Inputs
    a = 0;
    b = 0;
    cin = 0;
    // Wait 100 ns for global reset to finish
    #100;
    a = 5;
    b = 6;
    cin = 1;
    // Wait 100 ns for global reset to finish
    #100;
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
  end
endmodule
```