ARITHMETIC CIRCUITS SIMULATION USING VERILOG

NAME :ATHISH S A

ROLL NO: B21EC040

RIPPLE CARRY ADDER:

CODE:

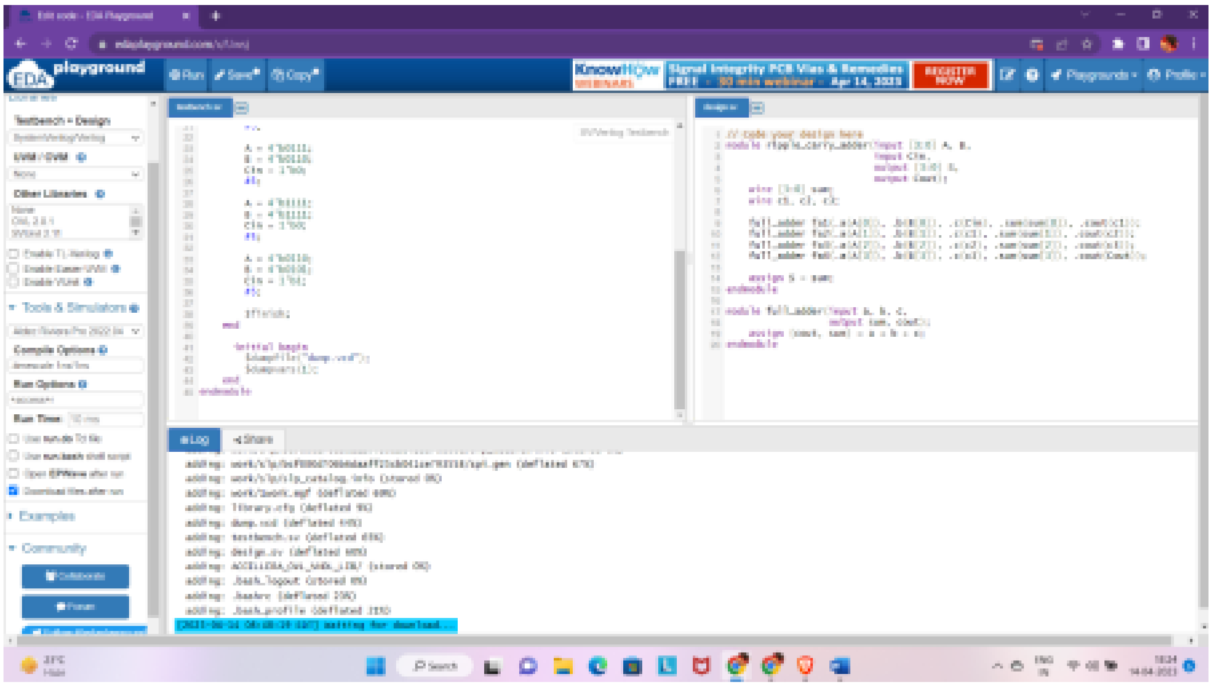
```
module ripple_carry_adder(input [3:0] A, B,
 input Cin,
 output [3:0] S,
 output Cout);
 wire [3:0] sum;
 wire c1, c2, c3;

 full_adder fa1(.a(A[0]), .b(B[0]), .c(Cin), .sum(sum[0]), .cout(c1));
full_adder fa2(.a(A[1]), .b(B[1]), .c(c1), .sum(sum[1]), .cout(c2));
full_adder fa3(.a(A[2]), .b(B[2]), .c(c2), .sum(sum[2]), .cout(c3));
 full_adder fa4(.a(A[3]), .b(B[3]), .c(c3), .sum(sum[3]), .cout(Cout));
 assign S = sum;
endmodule


module full_adder(input a, b, c,
 output sum, cout);
 assign {cout, sum} = a + b + c;
endmodule
```

TEST BENCH CODE:

```
module ripple_carry_adder_tb;
```

```
reg [3:0] A, B;
reg Cin;

wire [3:0] S;

wire Cout;

ripple_carry_adder rca(.A(A), .B(B), .Cin(Cin), .S(S),

.Cout(Cout));

initial begin

$monitor("A=%b, B=%b, Cin=%b, S=%b, Cout=%b", A, B, Cin, S, Cout);

A = 4'b0000;

B = 4'b0000;

Cin = 1'b0;

#5;


A = 4'b0001;

B = 4'b0001;

Cin = 1'b0;

#5;


A = 4'b0111;

B = 4'b0110;

Cin = 1'b0;

#5;


A = 4'b1111;

B = 4'b1111;

Cin = 1'b0;

#5;


A = 4'b0110;

B = 4'b0101;
```

```
Cin = 1'b1;
#5;


$finish;

end


initial begin

$dumpfile("dump.vcd");

$dumpvars(1);

end

endmodule
```

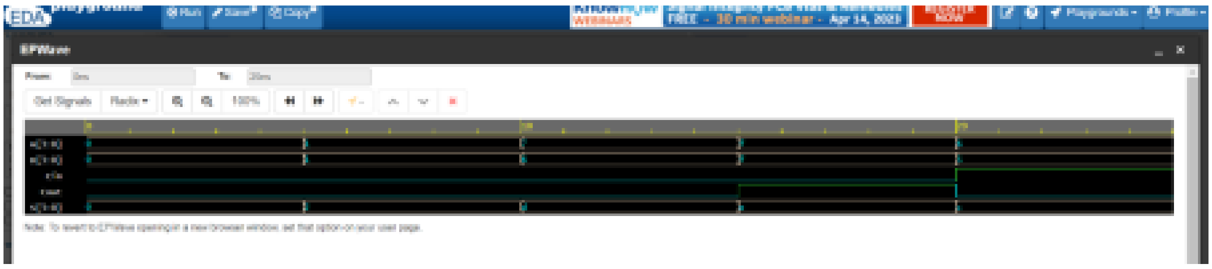


```
        #5;

        A = 4'b0111;
        B = 4'b0110;
        Cin = 1'b0;
        #5;

        A = 4'b1111;
        B = 4'b1111;
        Cin = 1'b0;
        #5;

        A = 4'b0110;
        B = 4'b0100;
        Cin = 1'b1;
        #5;

        $finish;
    end

    initial begin
        $dumpfile("dump.vcd");
        $dumpvars(1);
    end
endmodule
```

```
// Code your design here
module ripple_carry_adder(input [3:0] A, B,
                          input Cin,
                          output [3:0] S,
                          output Cout);
    wire [3:0] sum;
    wire c1, c2, c3;

    full_adder fa0(.a(A[0]), .b(B[0]), .c(Cin), .sum(sum[0]), .cout(c1));
    full_adder fa1(.a(A[1]), .b(B[1]), .c(c1), .sum(sum[1]), .cout(c2));
    full_adder fa2(.a(A[2]), .b(B[2]), .c(c2), .sum(sum[2]), .cout(c3));
    full_adder fa3(.a(A[3]), .b(B[3]), .c(c3), .sum(sum[3]), .cout(Cout));

    assign S = sum;
endmodule

module full_adder(input a, b, c,
                  output sum, cout);
    assign {cout, sum} = a + b + c;
endmodule
```

```
adding: work/_lp/[illegible]/_spl.pem (deflated [illegible])
adding: work/_lp/_lp_catalog.info (stored 0%)
adding: work/[illegible] (deflated [illegible])
adding: library.cfg (deflated [illegible])
adding: dump.vcd (deflated [illegible])
adding: testbench.sv (deflated [illegible])
adding: design.sv (deflated [illegible])
adding: [illegible] (stored 0%)
adding: .bash_logout (stored 0%)
adding: .bashrc (deflated [illegible])
adding: .bash_profile (deflated [illegible])
[2023-04-14 [illegible]] waiting for download...
```

EP WAVE:



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

BINARY RESTORING ADDITION:

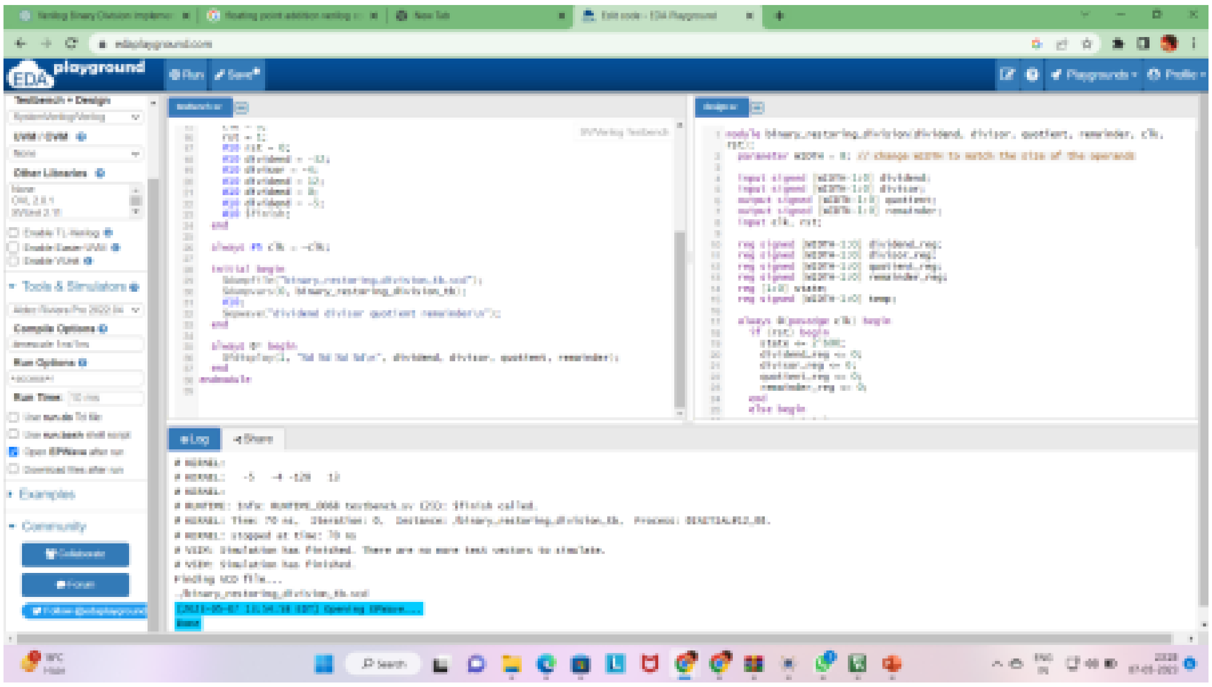
CODE:

```
module binary_restoring_division(dividend, divisor, quotient, remainder, clk,

rst);  parameter WIDTH = 8; // change WIDTH to match the size of the operands


 input signed [WIDTH-1:0] dividend;

 input signed [WIDTH-1:0] divisor;

 output signed [WIDTH-1:0] quotient;

 output signed [WIDTH-1:0] remainder;

 input clk, rst;


 reg signed [WIDTH-1:0] dividend_reg;

 reg signed [WIDTH-1:0] divisor_reg;

 reg signed [WIDTH-1:0] quotient_reg;

 reg signed [WIDTH-1:0] remainder_reg;

 reg [1:0] state;

 reg signed [WIDTH-1:0] temp;


 always @(posedge clk) begin

 if (rst) begin

 state <= 2'b00;

 dividend_reg <= 0;

 divisor_reg <= 0;

 quotient_reg <= 0;

 remainder_reg <= 0;

 end

 else begin

 case (state)

 2'b00: begin // initialize

 dividend_reg <= dividend;

 divisor_reg <= divisor;
 state <= 2'b01;
 end
 2'b01: begin // check sign
 if (dividend_reg[WIDTH-1] == divisor_reg[WIDTH-1])
state <= 2'b10;
 else
 state <= 2'b11;
 end
 2'b10: begin // subtract divisor
 temp <= dividend_reg - divisor_reg;
 if (temp[WIDTH-1] == dividend_reg[WIDTH-1])
quotient_reg[WIDTH-1] <= 1;
 divisor_reg <= divisor_reg >> 1;
 if (temp[WIDTH-1] == dividend_reg[WIDTH-1])
dividend_reg <= temp;
 state <= 2'b01;
 end
 2'b11: begin // add divisor
 temp <= dividend_reg + divisor_reg;
 if (temp[WIDTH-1] != dividend_reg[WIDTH-1])
quotient_reg[WIDTH-1] <= 1;
 divisor_reg <= divisor_reg >> 1;
 if (temp[WIDTH-1] != dividend_reg[WIDTH-1])
dividend_reg <= temp;
 state <= 2'b01;
 end
 endcase
 end
 end
 assign quotient = quotient_reg;
 assign remainder = dividend_reg;

endmodule
```

TEST BENCH CODE:

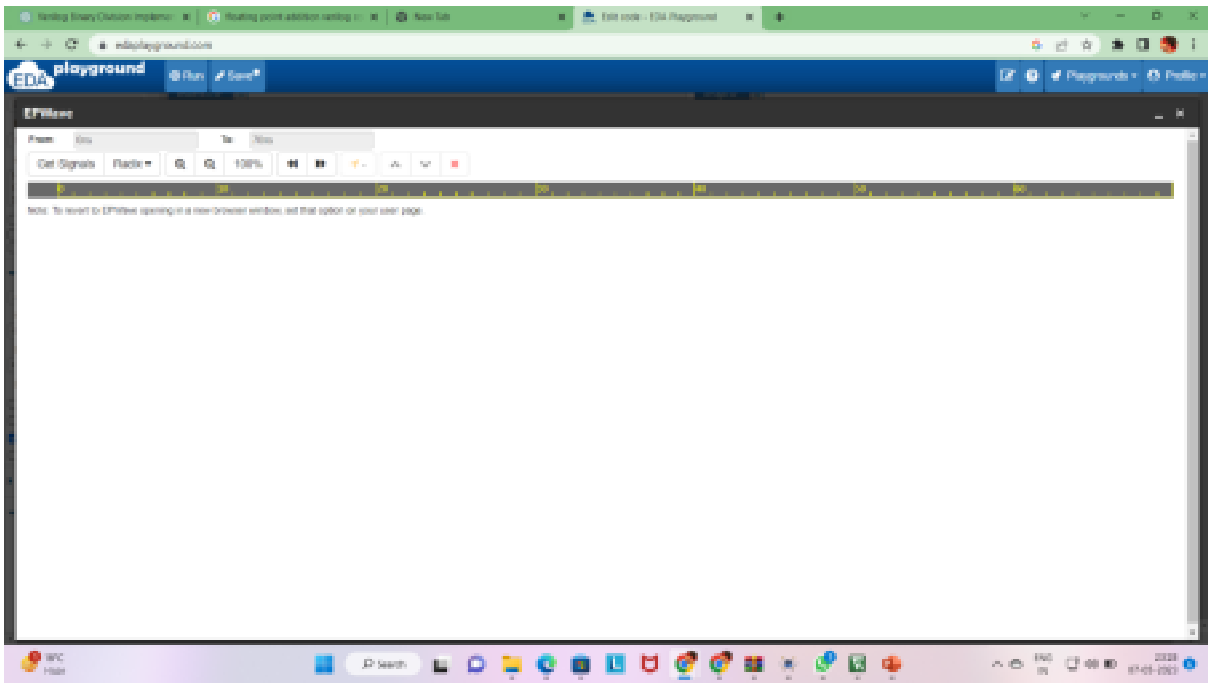
```
module binary_restoring_division_tb;

 parameter WIDTH = 8; // change WIDTH to match the size of the operands


 reg signed [WIDTH-1:0] dividend;

 reg signed [WIDTH-1:0] divisor;

 wire signed [WIDTH-1:0] quotient;

 wire signed [WIDTH-1:0] remainder;

 reg clk, rst;


 binary_restoring_division dut(.dividend(dividend), .divisor(divisor), .quotient(quotient),
.remainder(remainder), .clk(clk), .rst(rst));


 initial begin

 dividend = 16;

 divisor = 3;

 clk = 0;

 rst = 1;

 #10 rst = 0;

 #10 dividend = -32;

 #10 divisor = -4;

 #10 dividend = 12;

 #10 dividend = 0;

 #10 dividend = -5;

 #10 $finish;

 end
```

```
always #5 clk = ~clk;
initial begin

$dumpfile("binary_restoring_division_tb.vcd");

$dumpvars(0, binary_restoring_division_tb);

#10;

$epwave("dividend divisor quotient remainder\n");

end


always @* begin

$fdisplay(1, "%d %d %d %d\n", dividend, divisor, quotient, remainder);

end

endmodule
```



```
# KERNEL:
# KERNEL:    -5    -4 -128    12
# KERNEL:
# RUNTIME: Info: RUNTIME_0068 testbench.sv (20): $finish called.
# KERNEL: Time: 70 ns,  Iteration: 0,  Instance: /binary_restoring_division_tb,  Process: [illegible].
# KERNEL: stopped at time: 70 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./binary_restoring_division_tb.vcd
[2023-05-07 13:54:38 EDT] Opening EPWave...
Done
```

EPWAVE:



NON RESTORING DIVISON:

CODE:

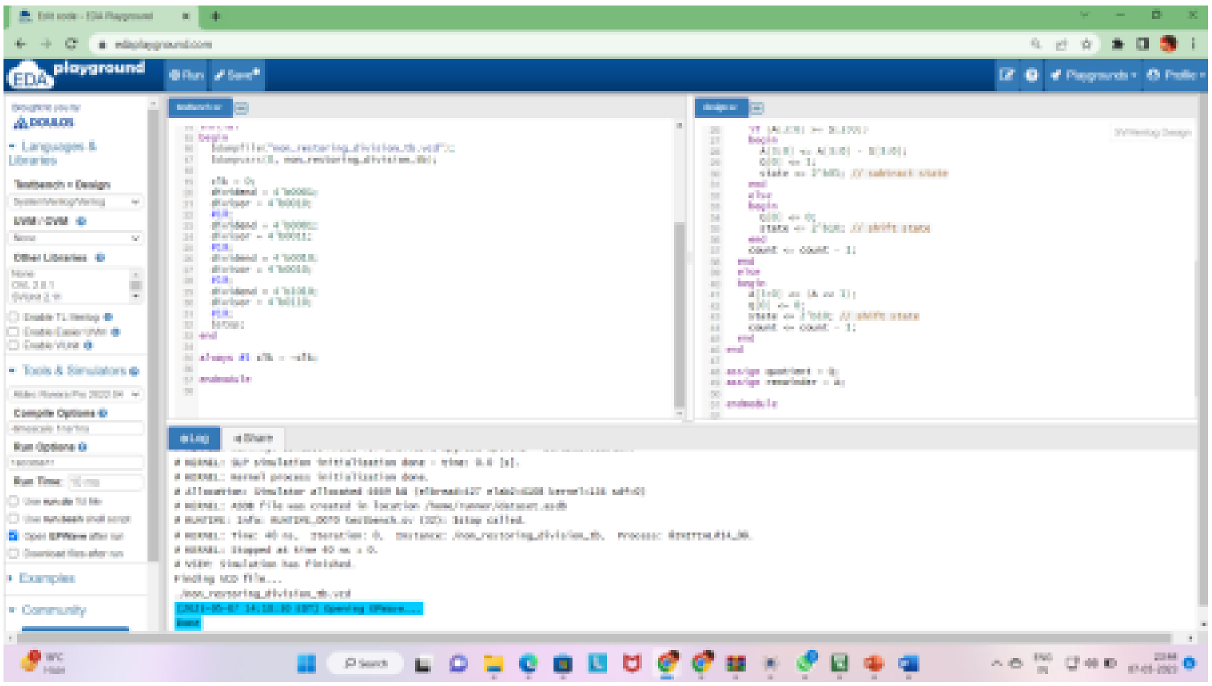
```
module non_restoring_division(clk, dividend, divisor, quotient, remainder,
state);


input clk;

input [3:0] dividend;

input [3:0] divisor;

output reg [3:0] quotient;

output reg [3:0] remainder;

output reg [1:0] state;


reg [3:0] A, S;
reg [3:0] Q;

reg [1:0] count;
```

```
always @(posedge clk)
begin
 if (count == 0)
 begin
 A <= dividend;
 S <= divisor;
 Q <= 0;
 count <= 3;
 state <= 2'b10; // shift state
end
 else if (count == 3)
 begin
 if (A[3:0] >= S[3:0])
 begin
 A[3:0] <= A[3:0] - S[3:0];
Q[0] <= 1;
 state <= 2'b01; // subtract state
end
 else
 begin
 Q[0] <= 0;
 state <= 2'b10; // shift state
end
 count <= count - 1;
 end
 else
 begin
 A[3:0] <= (A << 1);
 Q[0] <= 0;
 state <= 2'b10; // shift state
count <= count - 1;
 end
end

assign quotient = Q;
assign remainder = A;
endmodule
```

<u>TEST BENCH CODE:</u>

```
module non_restoring_division_tb();

reg clk;
reg [3:0] dividend;
reg [3:0] divisor;

wire [3:0] quotient;
wire [3:0] remainder;
wire [1:0] state;
non_restoring_division dut(clk, dividend, divisor, quotient, remainder, state);
```

```
initial
begin
 $dumpfile("non_restoring_division_tb.vcd");
 $dumpvars(0, non_restoring_division_tb);

 clk = 0;
 dividend = 4'b0001;
 divisor = 4'b0010;
 #10;
 dividend = 4'b0001;
 divisor = 4'b0011;
 #10;
 dividend = 4'b0010;
 divisor = 4'b0010;
 #10;
 dividend = 4'b1010;
 divisor = 4'b0110;
 #10;
 $stop;
end

always #5 clk = ~clk;

endmodule
```

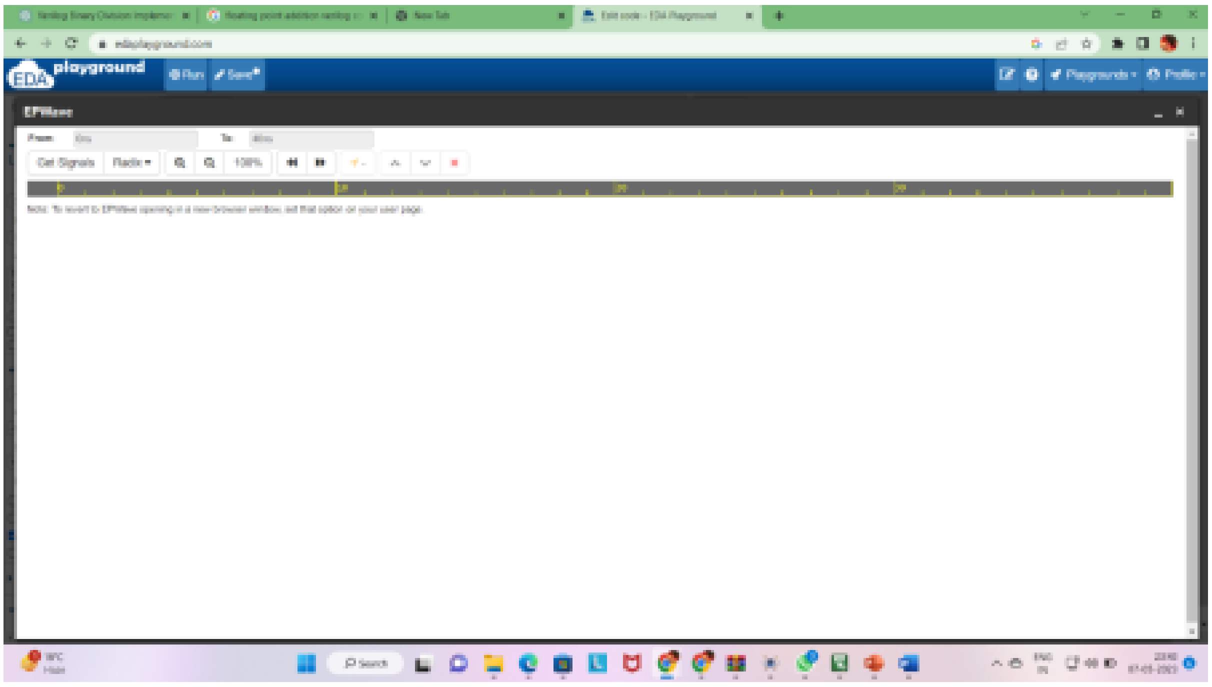


```
begin
  $dumpfile("non_restoring_division_tb.vcd");
  $dumpvars(0, non_restoring_division_tb);

  clk = 0;
  dividend = 4'b0000;
  divisor = 4'b0010;
  #10;
  dividend = 4'b0001;
  divisor = 4'b0011;
  #10;
  dividend = 4'b0010;
  divisor = 4'b0010;
  #10;
  dividend = 4'b1010;
  divisor = 4'b0110;
  #10;
  $stop;
end

always #1 clk = ~clk;

endmodule
```

```
      if (A[3:0] >= B[3:0])
      begin
        A[3:0] <= A[3:0] - B[3:0];
        Q[0] <= 1;
        state <= 2'b01; // subtract state
      end
      else
      begin
        Q[0] <= 0;
        state <= 2'b10; // shift state
      end
      count <= count - 1;
    end
    else
    begin
      A[3:0] <= {A << 1};
      Q[0] <= 0;
      state <= 2'b10; // shift state
      count <= count - 1;
    end
end

assign quotient = Q;
assign remainder = A;

endmodule
```

```
# KERNEL: SLP simulation initialization done - time: 0.0 [s].
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 4669 kB (elbread=427 elab2=4108 kernel=134 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# RUNTIME: Info: RUNTIME_0070 testbench.sv (32): $stop called.
# KERNEL: Time: 40 ns,  Iteration: 0,  Instance: /non_restoring_division_tb,  Process: @INITIAL#14_0@.
# KERNEL: Stopped at time 40 ns + 0.
# VSIM: Simulation has finished.
Finding VCD file...
./non_restoring_division_tb.vcd
[2023-05-07 14:18:30 EDT] Opening EPWave...
Done
```

EPWAVE:



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

CARRY SELECT ADDER:

CODE:

module cfa_adder(a, b, cin, sum, cout);

input [3:0] a, b;

input cin;
output [3:0] sum;

output cout;

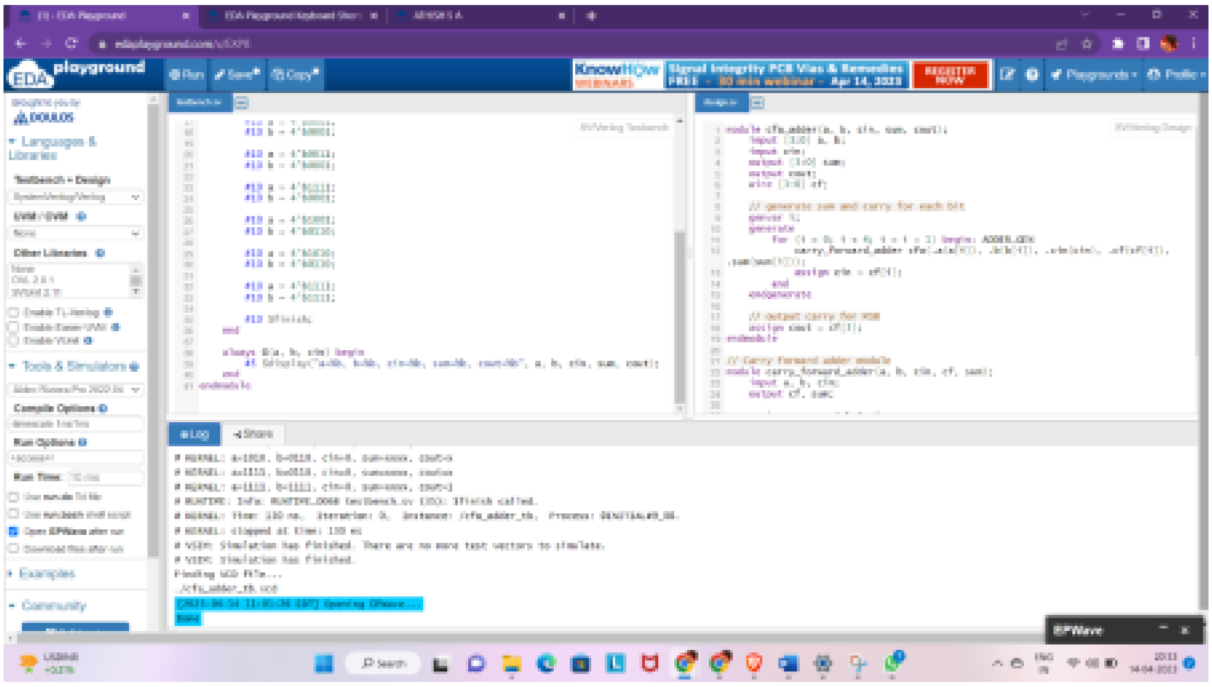
```
 wire [3:0] cf;

 // generate sum and carry for each bit
 genvar i;
 generate
 for (i = 0; i < 4; i = i + 1) begin: ADDER_GEN
 carry_forward_adder cfa(.a(a[i]), .b(b[i]), .cin(cin), .cf(cf[i]), .sum(sum[i]));
assign cin = cf[i];
 end
 endgenerate

 // output carry for MSB
 assign cout = cf[3];
endmodule

// Carry forward adder module
module carry_forward_adder(a, b, cin, cf, sum);
 input a, b, cin;
 output cf, sum;

 assign sum = a ^ b ^ cin;
 assign cf = (a & b) | (a & cin) | (b & cin);
endmodule
```

TEST BENCH CODE:

```
module cfa_adder_tb;
 reg [3:0] a, b;
 reg cin;
 wire [3:0] sum;
 wire cout;
```

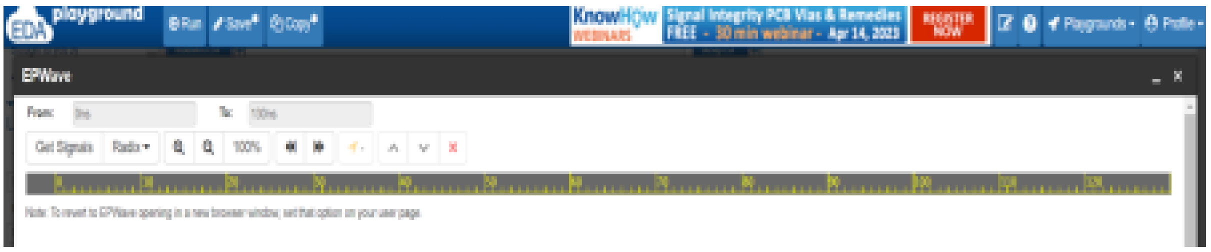
```
cfa_adder adder(.a(a), .b(b), .cin(cin), .sum(sum), .cout(cout));

initial begin
$dumpfile("cfa_adder_tb.vcd");
$dumpvars(0, cfa_adder_tb);

a = 4'b0000;
b = 4'b0000;
cin = 0;

#10 a = 4'b0001;
#10 b = 4'b0001;

#10 a = 4'b0011;
#10 b = 4'b0001;

#10 a = 4'b1111;
#10 b = 4'b0001;

#10 a = 4'b1001;
#10 b = 4'b0110;

#10 a = 4'b1010;
#10 b = 4'b0110;

#10 a = 4'b1111;
#10 b = 4'b1111;

#10 $finish;
```

```
end

always @(a, b, cin) begin
#5 $display("a=%b, b=%b, cin=%b, sum=%b, cout=%b", a, b, cin, sum, cout);
end
endmodule
```



```
# KERNEL: a=[illegible], b=[illegible], cin=0, sum=xxxx, cout=x
# KERNEL: a=1111, b=[illegible], cin=[illegible], sum=xxxx, cout=x
# KERNEL: a=1111, b=1111, cin=0, sum=xxxx, cout=1
# RUNTIME: Info: RUNTIME_0068 testbench.sv (33): $finish called.
# KERNEL: Time: 130 ns,  Iteration: 0,  Instance: /cfa_adder_tb,  Process: @INITIAL#5_0@.
# KERNEL: stopped at time: 130 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./cfa_adder_tb.vcd
[2023-04-14 11:01:28 EDT] Opening EPWave...
Done
```

EPWAVE:


EPWave
From: 0ns
To: 130ns
Get Signals
Radix
100%
Note: To revert to EPWave opening in a new browser window, set that option on your user page.

CARRY SAVE ADDER:

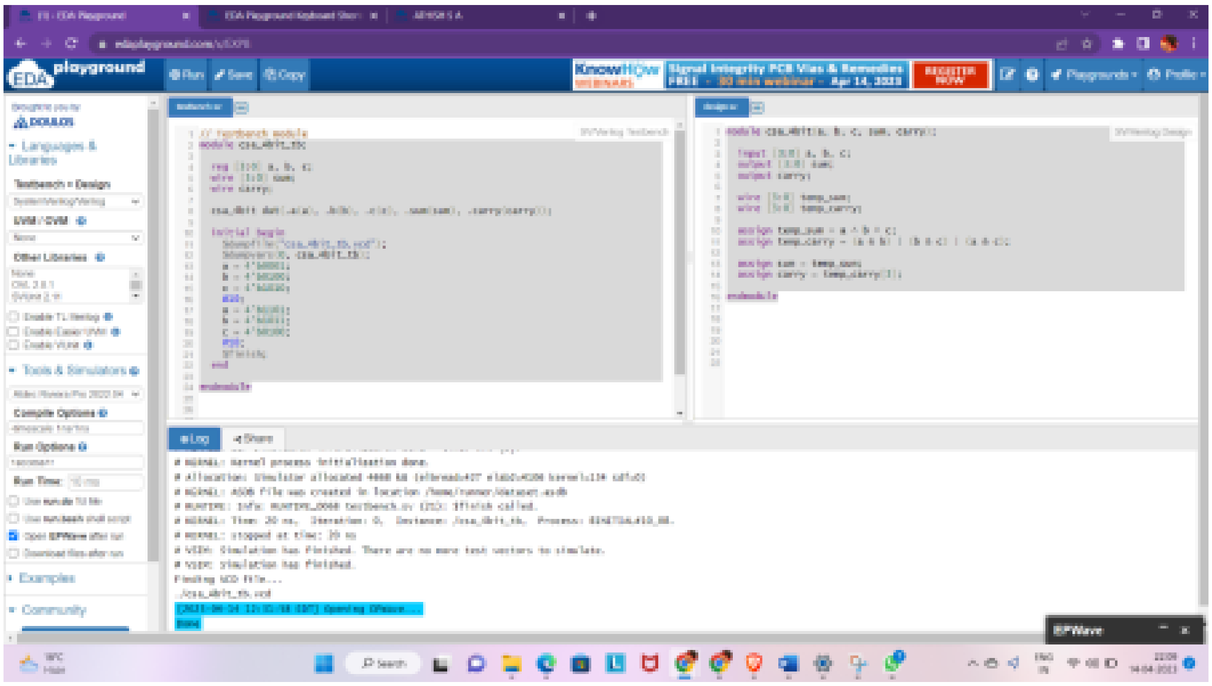
CODE:

```
module csa_4bit(a, b, c, sum, carry);
 input [3:0] a, b, c;

 output [3:0] sum;

 output carry;
 wire [3:0] temp_sum;
 wire [3:0] temp_carry;

 assign temp_sum = a ^ b ^ c;
 assign temp_carry = (a & b) | (b & c) | (a & c);

 assign sum = temp_sum;
 assign carry = temp_carry[3];

endmodule
```

<u>TEST BENCH CODE:</u>

```
module csa_4bit_tb;

 reg [3:0] a, b, c;
 wire [3:0] sum;
 wire carry;

 csa_4bit dut(.a(a), .b(b), .c(c), .sum(sum), .carry(carry));

 initial begin
 $dumpfile("csa_4bit_tb.vcd");
 $dumpvars(0, csa_4bit_tb);
 a = 4'b0001;
 b = 4'b0100;
```

```
c = 4'b1010;
#10;
a = 4'b1101;
b = 4'b1011;
c = 4'b0100;
#10;
$finish;
end


endmodule
```

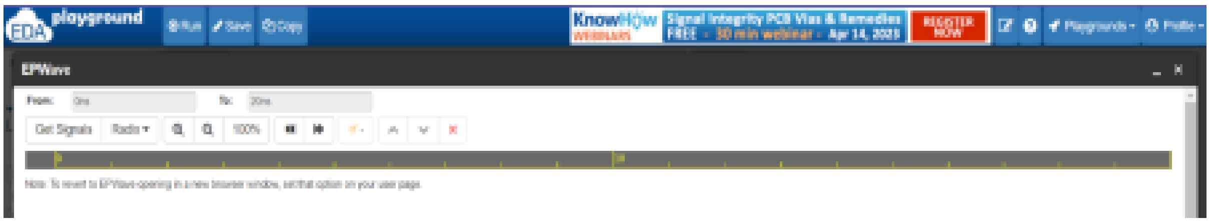


```
// testbench module
module csa_4bit_tb;

  reg [3:0] a, b, c;
  wire [3:0] sum;
  wire carry;

  csa_4bit dut(.a(a), .b(b), .c(c), .sum(sum), .carry(carry));

  initial begin
    $dumpfile("csa_4bit_tb.vcd");
    $dumpvars(0, csa_4bit_tb);
    a = 4'b[illegible];
    b = 4'b[illegible];
    c = 4'b[illegible];
    #10;
    a = 4'b[illegible];
    b = 4'b[illegible];
    c = 4'b[illegible];
    #10;
    $finish;
  end

endmodule
```

```
module csa_4bit(a, b, c, sum, carry);

  input [3:0] a, b, c;
  output [3:0] sum;
  output carry;

  wire [3:0] temp_sum;
  wire [3:0] temp_carry;

  assign temp_sum = a ^ b ^ c;
  assign temp_carry = (a & b) | (b & c) | (a & c);

  assign sum = temp_sum;
  assign carry = temp_carry[3];

endmodule
```

```
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated [illegible] kB [illegible]
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# RUNTIME: Info: RUNTIME_0068 testbench.sv ([illegible]): $finish called.
# KERNEL: Time: 20 ns,  Iteration: 0,  Instance: /csa_4bit_tb,  Process: [illegible].
# KERNEL: stopped at time: 20 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./csa_4bit_tb.vcd
[illegible] Opening EPWave...
Done
```

EP WAVE:



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

BINARY MULTIPLIER:

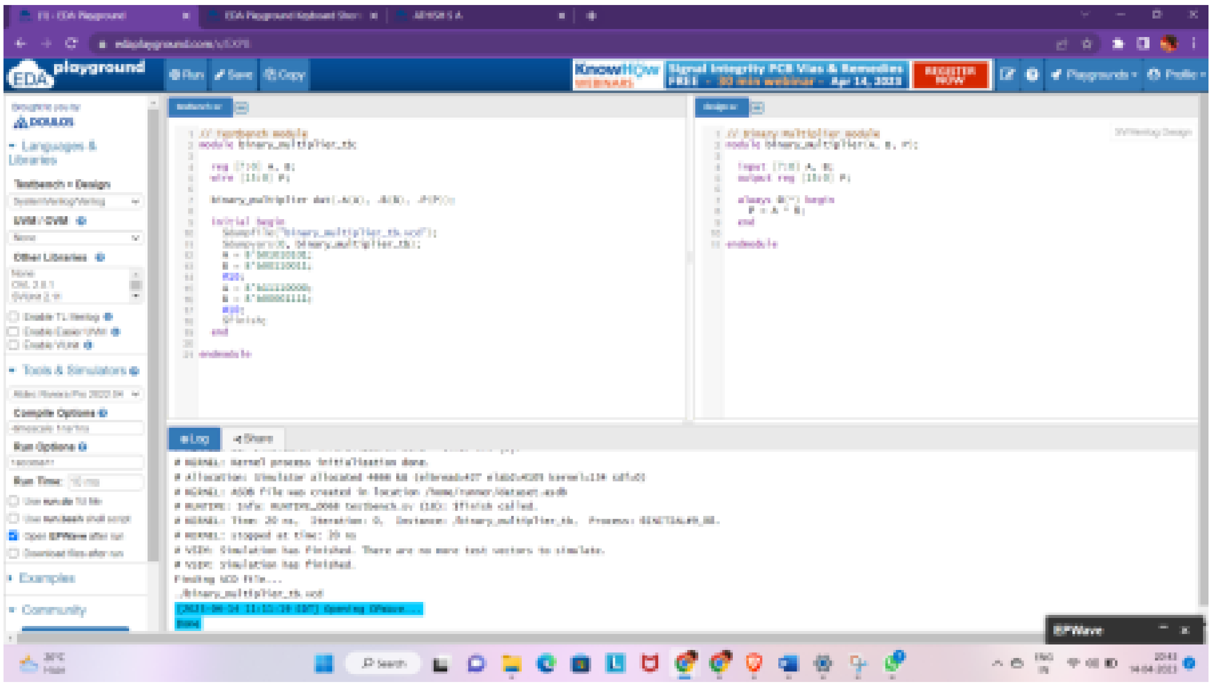
CODE:

```
module binary_multiplier(A, B, P);


 input [7:0] A, B;

 output reg [15:0] P;


 always @(*) begin

 P = A * B;

 end


endmodule
```

TEST BENCH CODE:

```
module binary_multiplier_tb;

 reg [7:0] A, B;

 wire [15:0] P;

 binary_multiplier dut(.A(A), .B(B), .P(P));

 initial begin

 $dumpfile("binary_multiplier_tb.vcd");

 $dumpvars(0, binary_multiplier_tb);

 A = 8'b01010101;

 B = 8'b00110011;

 #10;

 A = 8'b11110000;
 B = 8'b00001111;

 #10;

 $finish;

 end
```

endmodule

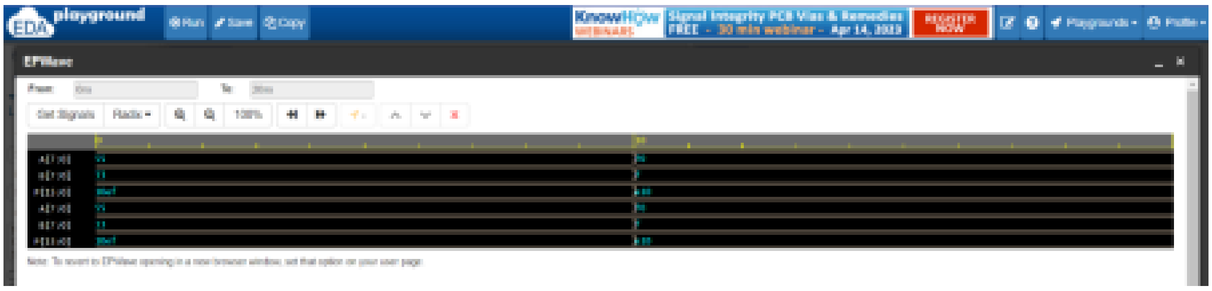


```
// testbench module
module binary_multiplier_tb;

  reg [7:0] A, B;
  wire [15:0] P;

  binary_multiplier dut(.A(A), .B(B), .P(P));

  initial begin
    $dumpfile("binary_multiplier_tb.vcd");
    $dumpvars(0, binary_multiplier_tb);
    A = 8'b01010101;
    B = 8'b00110011;
    #20;
    A = 8'b11110000;
    B = 8'b00001111;
    #10;
    $finish;
  end

endmodule
```

```
// binary multiplier module
module binary_multiplier(A, B, P);

  input [7:0] A, B;
  output reg [15:0] P;

  always @(*) begin
    P = A * B;
  end

endmodule
```

```
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 4888 kB (elbread=427 elab2=4335 kernel=134 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# RUNTIME: Info: RUNTIME_0068 testbench.sv (18): $finish called.
# KERNEL: Time: 30 ns,  Iteration: 0,  Instance: /binary_multiplier_tb,  Process: @INITIAL#9_0@.
# KERNEL: stopped at time: 30 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./binary_multiplier_tb.vcd
[2023-04-14 11:11:19 EDT] Opening EPWave...
Done
```

EP WAVE:



BOOTHS MULTIPLIER:

CODE:

module booths_multiplier(A, B, P);

input [7:0] A, B;
output reg [15:0] P;

reg [7:0] A_reg, B_reg;

reg [1:0] count;

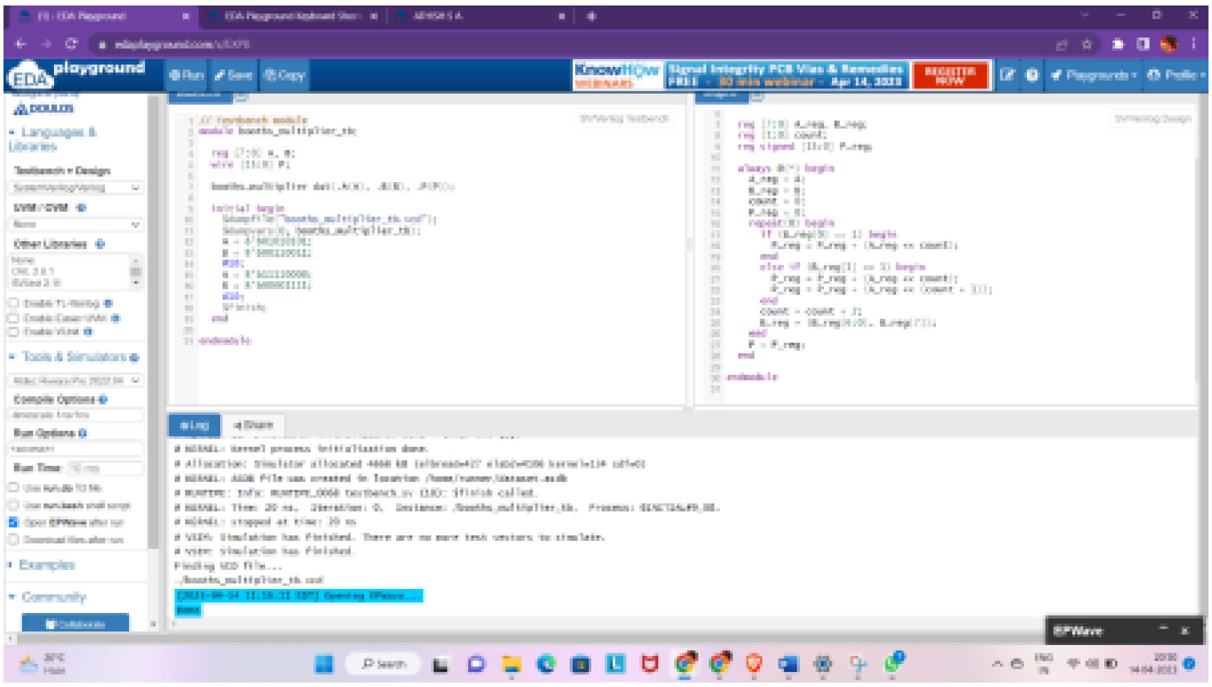
reg signed [15:0] P_reg;

```
 always @(*) begin
 A_reg = A;
 B_reg = B;
 count = 0;
 P_reg = 0;
 repeat(8) begin
 if (B_reg[0] == 1) begin
 P_reg = P_reg + (A_reg << count);
end
 else if (B_reg[1] == 1) begin
 P_reg = P_reg + (A_reg << count);  P_reg =
P_reg + (A_reg << (count + 1));  end
 count = count + 2;
 B_reg = {B_reg[6:0], B_reg[7]};
end
 P = P_reg;
 end


endmodule
```

TEST BENCH CODE:

```
module booths_multiplier_tb;


 reg [7:0] A, B;
 wire [15:0] P;


 booths_multiplier dut(.A(A), .B(B), .P(P));


 initial begin
```

```
$dumpfile("booths_multiplier_tb.vcd");

$dumpvars(0, booths_multiplier_tb);

A = 8'b01010101;

B = 8'b00110011;

#10;

A = 8'b11110000;

B = 8'b00001111;

#10;

$finish;

end


endmodule
```

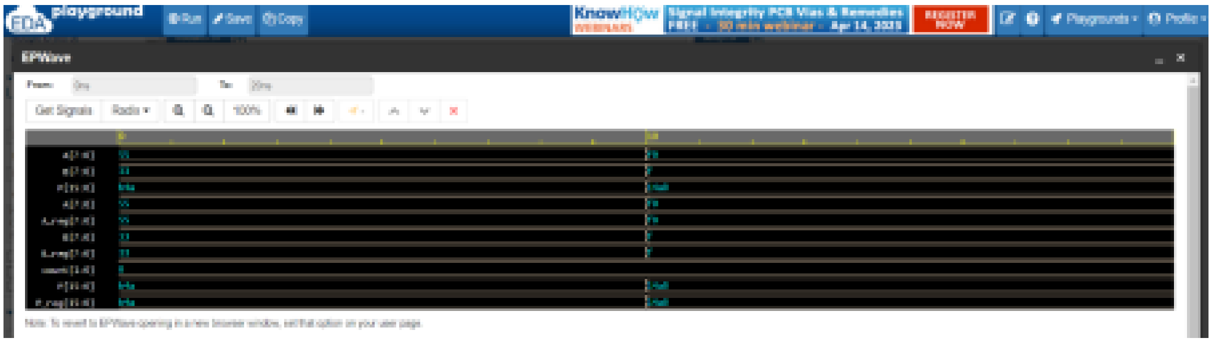


```
// testbench module
module booths_multiplier_tb;

  reg [7:0] A, B;
  wire [15:0] P;

  booths_multiplier dut(.A(A), .B(B), .P(P));

  initial begin
    $dumpfile("booths_multiplier_tb.vcd");
    $dumpvars(0, booths_multiplier_tb);
    A = 8'b01010101;
    B = 8'b00110011;
    #10;
    A = 8'b11110000;
    B = 8'b00001111;
    #10;
    $finish;
  end

endmodule
```

```
  reg [7:0] A_reg, B_reg;
  reg [3:0] count;
  reg signed [15:0] P_reg;

  always @(*) begin
    A_reg = A;
    B_reg = B;
    count = 0;
    P_reg = 0;
    repeat(8) begin
      if (B_reg[0] == 1) begin
        P_reg = P_reg + (A_reg << count);
      end
      else if (B_reg[1] == 1) begin
        P_reg = P_reg + (A_reg << count);
        P_reg = P_reg - (A_reg << (count + 1));
      end
      count = count + 1;
      B_reg = {B_reg[6:0], B_reg[7]};
    end
    P = P_reg;
  end

endmodule
```

```
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 4668 kB (elbread=427 elab2=4108 kernel=134 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# RUNTIME: Info: RUNTIME_0068 testbench.sv (18): $finish called.
# KERNEL: Time: 20 ns,  Iteration: 0,  Instance: /booths_multiplier_tb,  Process: @INITIAL#9_0@.
# KERNEL: stopped at time: 20 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./booths_multiplier_tb.vcd
[2023-04-14 11:16:11 EDT] Opening EPWave...
Done
```

EPWAVE:



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

WALLACE MULTIPLIER:

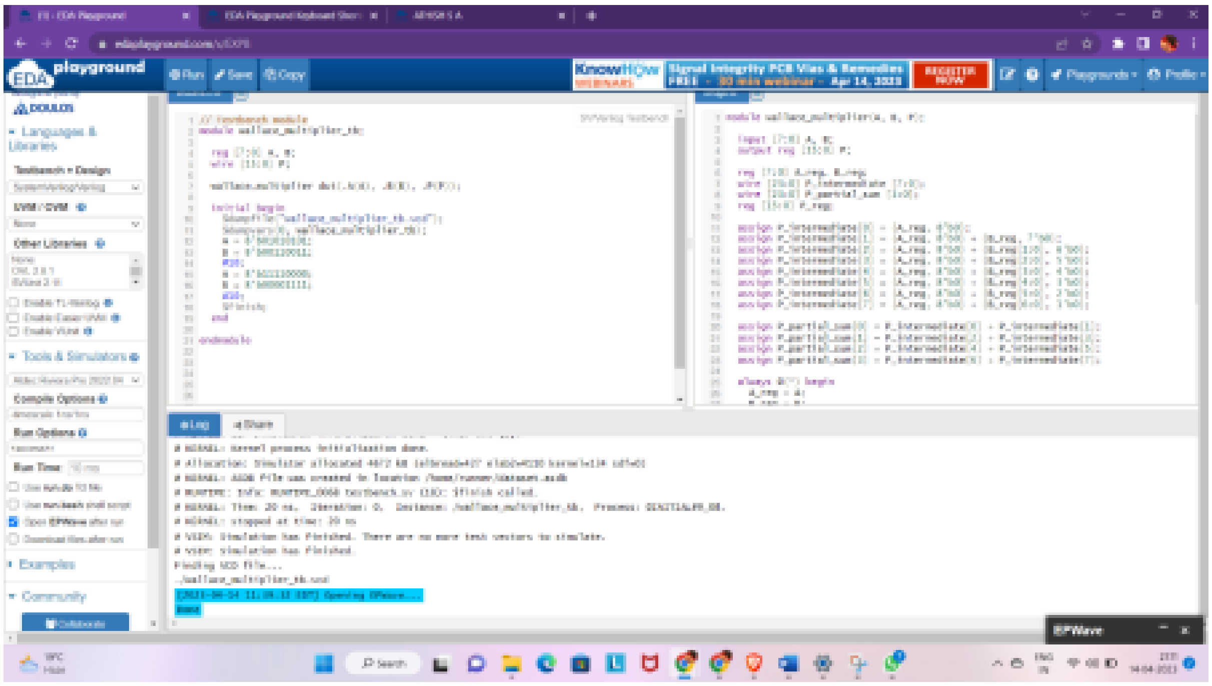
CODE:

```
module wallace_multiplier(A, B, P);


 input [7:0] A, B;

 output reg [15:0] P;


 reg [7:0] A_reg, B_reg;

 wire [23:0] P_intermediate [7:0];

 wire [23:0] P_partial_sum [3:0];

 reg [15:0] P_reg;


 assign P_intermediate[0] = {A_reg, 8'b0};

 assign P_intermediate[1] = {A_reg, 8'b0} + {B_reg, 7'b0};

 assign P_intermediate[2] = {A_reg, 8'b0} + {B_reg[1:0], 6'b0};

 assign P_intermediate[3] = {A_reg, 8'b0} + {B_reg[2:0], 5'b0};

 assign P_intermediate[4] = {A_reg, 8'b0} + {B_reg[3:0], 4'b0};
  assign P_intermediate[5] = {A_reg, 8'b0} + {B_reg[4:0], 3'b0};

assign P_intermediate[6] = {A_reg, 8'b0} + {B_reg[5:0], 2'b0};

assign P_intermediate[7] = {A_reg, 8'b0} + {B_reg[6:0], 1'b0};


  assign P_partial_sum[0] = P_intermediate[0] + P_intermediate[1];

assign P_partial_sum[1] = P_intermediate[2] + P_intermediate[3];
assign P_partial_sum[2] = P_intermediate[4] + P_intermediate[5];

assign P_partial_sum[3] = P_intermediate[6] + P_intermediate[7];


 always @(*) begin

 A_reg = A;

 B_reg = B;

 P_reg = 0;

 for (int i = 0; i < 4; i = i + 1) begin

 P_reg = P_reg + (P_partial_sum[i] << (3*i));

 end

 P = P_reg;

 end


endmodule
```

TEST BENCH CODE:

```
module wallace_multiplier_tb;


 reg [7:0] A, B;

 wire [15:0] P;


 wallace_multiplier dut(.A(A), .B(B), .P(P));


 initial begin

 $dumpfile("wallace_multiplier_tb.vcd");
 $dumpvars(0, wallace_multiplier_tb);

 A = 8'b01010101;

 B = 8'b00110011;

 #10;

 A = 8'b11110000;

 B = 8'b00001111;
```

```
#10;

$finish;

end


endmodule
```



```
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 4672 kB (elbread=427 elab2=4130 kernel=114 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# RUNTIME: Info: RUNTIME_0068 testbench.sv (18): $finish called.
# KERNEL: Time: 20 ns,  Iteration: 0,  Instance: /wallace_multiplier_tb,  Process: @INITIAL#8_0@.
# KERNEL: stopped at time: 20 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./wallace_multiplier_tb.vcd
[2023-04-14 11:09:12 EDT] Opening EPWave...
Done
```

EPWAVE:

DIVISION CIRCUIT:

CODE:

```
module division_circuit(dividend, divisor, quotient);
```

```
 input [7:0] dividend, divisor;

 output reg [7:0] quotient;


 reg [7:0] remainder;


 always @(*) begin

 quotient = 0;

 remainder = dividend;

 for (int i = 0; i < 8; i = i + 1) begin

 quotient = quotient << 1;

 remainder = remainder << 1;

 if (remainder >= divisor) begin

 quotient[i] = 1;

 remainder = remainder - divisor;

 end

 end

 end
endmodule
```

TEST BENCH CODE:

```
module division_circuit_tb;


 reg [7:0] dividend, divisor;

 wire [7:0] quotient;


 division_circuit dut(.dividend(dividend), .divisor(divisor), .quotient(quotient));


 initial begin

 $dumpfile("division_circuit_tb.vcd");
 $dumpvars(0, division_circuit_tb);

 dividend = 8'b11011010;

 divisor = 8'b00101100;

 #10;

 dividend = 8'b11110000;

 divisor = 8'b00001111;

 #10;

 $finish;

 end

endmodule
```

EPWAVE: