# Digital IC Design
## Assignment 1 – Q1

Name- Pushkal Mishra
Roll- EE20BTECH11042

To compute Y in one clock cycle, where Y is given as

$$Y = x_0 h_0 + x_1 h_1 + x_2 h_2 + \ldots\ldots + x_9 h_9$$

Verilog implementation

```
module correlation_1 (y, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, h_0,
h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9);

      output reg [11:0] y;

      input clock, reset;

      input [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;
      input [3:0] h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9;

      always @(posedge clock, reset)
      begin
            if (reset)
            begin
                  y <= 12'b0000_0000_0000;
            end
            else
            begin
                  y <= x_0 * h_0 + x_1 * h_1 + x_2 * h_2 + x_3 * h_3 + x_4 * h_4 + x_5 * h_5 +
x_6 * h_6 + x_7 * h_7 + x_8 * h_8 + x_9 * h_9;
            end
      end

endmodule
```

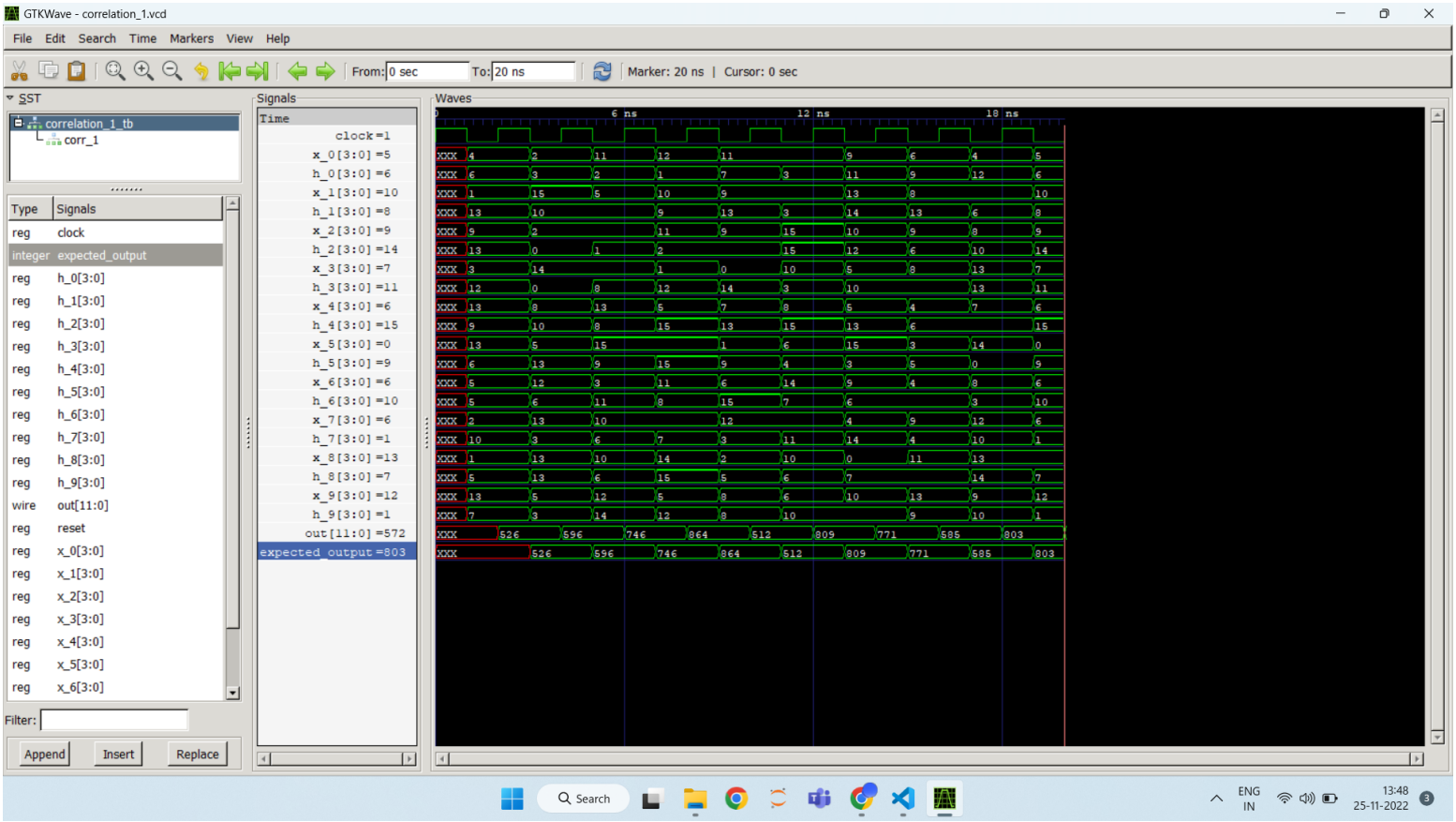
Testbench

```
`timescale 1ns/1ns
`include "correlation_1.v"

module correlation_1_tb;

    reg [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;
      reg [3:0] h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9;
    reg reset = 1'b0;
    reg clock = 1'b1;

    wire [11:0] out;
```

```
    always #1 clock = ~clock;

    correlation_1 corr_1(out, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9,
h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9);

    integer expected_output;

    initial
    begin

        $monitor($time, "x_0 = %d x_1 = %d x_2 = %d x_3 = %d x_4 = %d x_5 = %d x_6 = %d x_7 =
%d x_8 = %d x_9 = %d h_0 = %d h_1 = %d h_2 = %d h_3 = %d h_4 = %d h_5 = %d h_6 = %d h_7 = %d
h_8 = %d h_9 = %d output = %d expected = %d", x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8,
x_9, h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, out, expected_output);

        $dumpfile("correlation_1.vcd");
        $dumpvars;

        #20 $finish;

    end

    always @(negedge clock)
    begin

        x_0 <= $random;
        x_1 <= $random;
        x_2 <= $random;
        x_3 <= $random;
        x_4 <= $random;
        x_5 <= $random;
        x_6 <= $random;
        x_7 <= $random;
        x_8 <= $random;
        x_9 <= $random;

        h_0 <= $random;
        h_1 <= $random;
        h_2 <= $random;
        h_3 <= $random;
        h_4 <= $random;
        h_5 <= $random;
        h_6 <= $random;
        h_7 <= $random;
        h_8 <= $random;
        h_9 <= $random;

        expected_output = x_0 * h_0 + x_1 * h_1 + x_2 * h_2 + x_3 * h_3 + x_4 * h_4 + x_5 *
h_5 + x_6 * h_6 + x_7 * h_7 + x_8 * h_8 + x_9 * h_9;
```

```
    end

endmodule
```

Output in GTKWave




Here, I directly computed the correlation in one statement and passed it on to the output variable. Clearly the output matches the expected output.

# Digital IC Design
# Assignment 1 – Q2

Name- Pushkal Mishra
Roll- EE20BTECH11042

To compute Y keeping in mind that no two different arithmetic operations can take place in the same clock edge, where Y is given as

$$Y = x_0 h_0 + x_1 h_1 + x_2 h_2 + \ldots\ldots + x_9 h_9$$

Verilog implementation

```
module correlation_2 (y, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, h_0,
h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9);

      output reg [11:0] y;

      input clock, reset;

      input [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;
      input [3:0] h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9;

      reg [7:0] mul_0, mul_1, mul_2, mul_3, mul_4, mul_5, mul_6, mul_7, mul_8, mul_9;

      always @(posedge clock)
      begin

            mul_0 <= x_0 * h_0;
            mul_1 <= x_1 * h_1;
            mul_2 <= x_2 * h_2;
            mul_3 <= x_3 * h_3;
            mul_4 <= x_4 * h_4;
            mul_5 <= x_5 * h_5;
            mul_6 <= x_6 * h_6;
            mul_7 <= x_7 * h_7;
            mul_8 <= x_8 * h_8;
            mul_9 <= x_9 * h_9;

      end

      always @(negedge clock, reset)
      begin

            if(reset)
            begin
                  y <= 12'b0000_0000_0000;
            end

            else
```

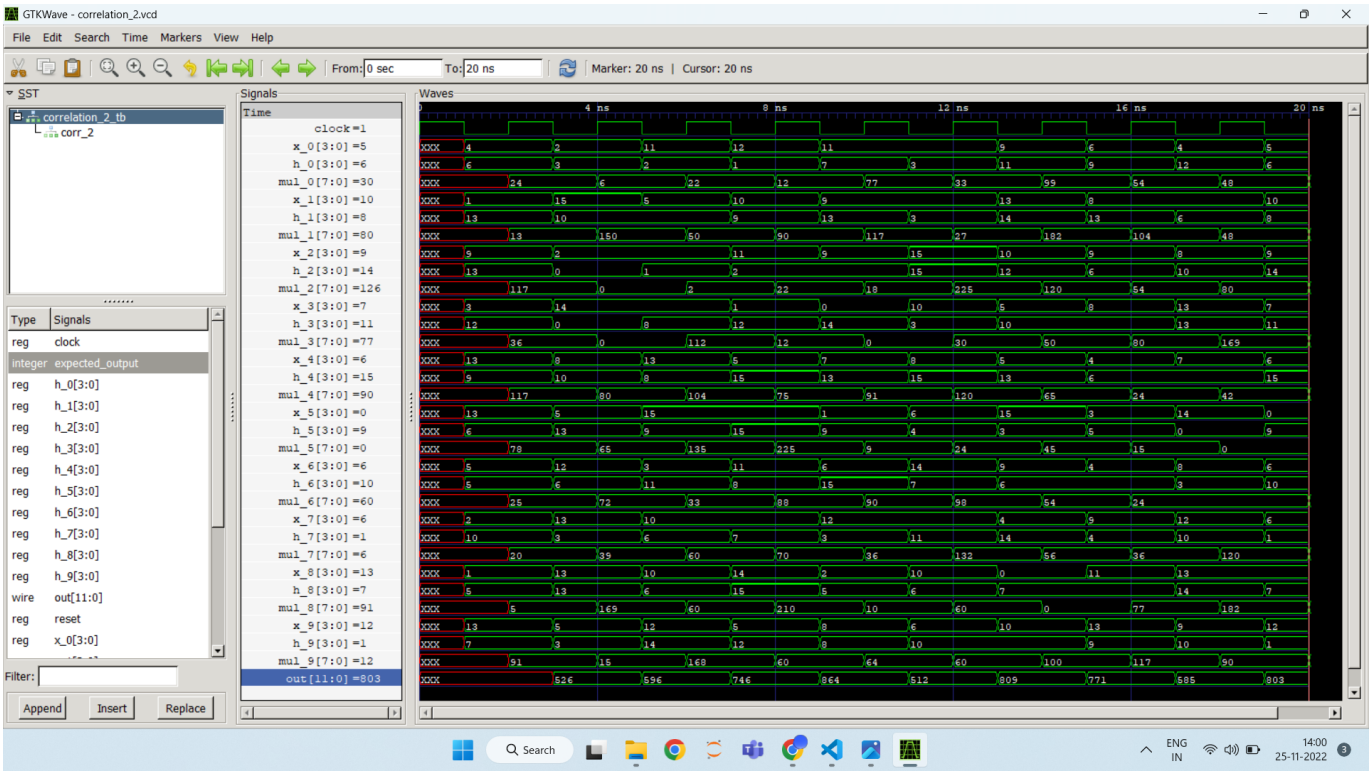
```
            begin
                  y <= mul_0 + mul_1 + mul_2 + mul_3 + mul_4 + mul_5 + mul_6 + mul_7 + mul_8 +
mul_9;
            end

      end

endmodule
```

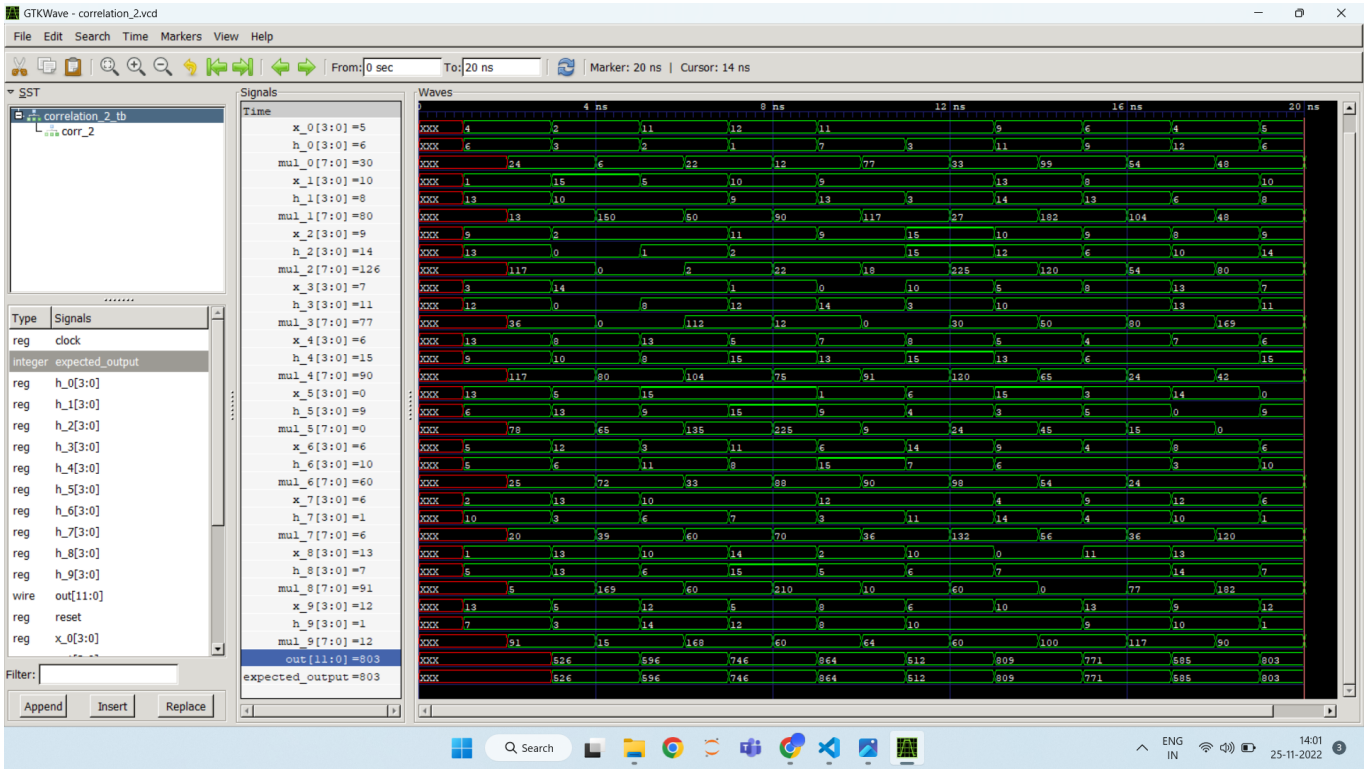
Testbench

```
`timescale 1ns/1ns
`include "correlation_2.v"

module correlation_2_tb;

    reg [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;
      reg [3:0] h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9;
    reg reset = 1'b0;
    reg clock = 1'b1;

    wire [11:0] out;

    always #1 clock = ~clock;

    correlation_2 corr_2(out, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9,
h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9);

    integer expected_output;

    initial
    begin

        $monitor($time, "x_0 = %d x_1 = %d x_2 = %d x_3 = %d x_4 = %d x_5 = %d x_6 = %d x_7 =
%d x_8 = %d x_9 = %d h_0 = %d h_1 = %d h_2 = %d h_3 = %d h_4 = %d h_5 = %d h_6 = %d h_7 = %d
h_8 = %d h_9 = %d output = %d expected = %d", x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8,
x_9, h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, out, expected_output);

        $dumpfile("correlation_2.vcd");
        $dumpvars;

        #20 $finish;

    end

    always @(negedge clock)
    begin

        x_0 <= $random;
        x_1 <= $random;
```

```
        x_2 <= $random;
        x_3 <= $random;
        x_4 <= $random;
        x_5 <= $random;
        x_6 <= $random;
        x_7 <= $random;
        x_8 <= $random;
        x_9 <= $random;

        h_0 <= $random;
        h_1 <= $random;
        h_2 <= $random;
        h_3 <= $random;
        h_4 <= $random;
        h_5 <= $random;
        h_6 <= $random;
        h_7 <= $random;
        h_8 <= $random;
        h_9 <= $random;

        expected_output = x_0 * h_0 + x_1 * h_1 + x_2 * h_2 + x_3 * h_3 + x_4 * h_4 + x_5 *
h_5 + x_6 * h_6 + x_7 * h_7 + x_8 * h_8 + x_9 * h_9;

    end

endmodule
```

Output in GTKWave






All the products are first computed in the positive edge of the clock cycle and all additions are computed in the negative edge of the clock cycle. This whole operation takes only one clock cycle to complete. Clearly from the above image, the output and expected outputs match.

# Digital IC Design
# Assignment 1 – Q3

Name- Pushkal Mishra
Roll- EE20BTECH11042

To compute Y keeping in mind that we have only two multipliers and one adder, where Y is given as

$$Y = x_0 h_0 + x_1 h_1 + x_2 h_2 + \ldots\ldots + x_9 h_9$$

Verilog implementation

```
module correlation_3 ( y, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, h_0,
h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9);

  output reg [11:0] y;

  input clock, reset;

  input [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;
  input [3:0] h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9;

  reg [2:0] counter;

  reg [3:0] mul_even_x, mul_even_h, mul_odd_x, mul_odd_h;
  reg [7:0] mul_even, mul_odd;

  reg [11:0] memory_sum;

  initial
  begin

    counter <= 3'b000;
    mul_even_x <= 4'b0000;
    mul_even_h <= 4'b0000;
    mul_odd_x <= 4'b0000;
    mul_odd_h <= 4'b0000;
    mul_even <= 8'b0000_0000;
    mul_odd <= 8'b0000_0000;
    memory_sum <= 12'b0000_0000_0000;
    y <= 12'b0000_0000_0000;

  end

  always @(posedge clock)
  begin

    if (counter < 6)
    begin
```

```
    counter <= counter + 1;
  end

  else
  begin
    counter <= 3'b111;
  end

end

always @(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)
begin

  counter <= 3'b000;
  y <= 12'b0000_0000_0000;

end

always @(counter)
begin

  case (counter)

    3'b000:
    begin

      mul_even_x <= x_0;
      mul_even_h <= h_0;
      mul_odd_x  <= x_1;
      mul_odd_h  <= h_1;

    end

    3'b001:
    begin

      mul_even_x <= x_2;
      mul_even_h <= h_2;
      mul_odd_x  <= x_3;
      mul_odd_h  <= h_3;

    end

    3'b010:
    begin

      mul_even_x <= x_4;
      mul_even_h <= h_4;
      mul_odd_x  <= x_5;
      mul_odd_h  <= h_5;
```

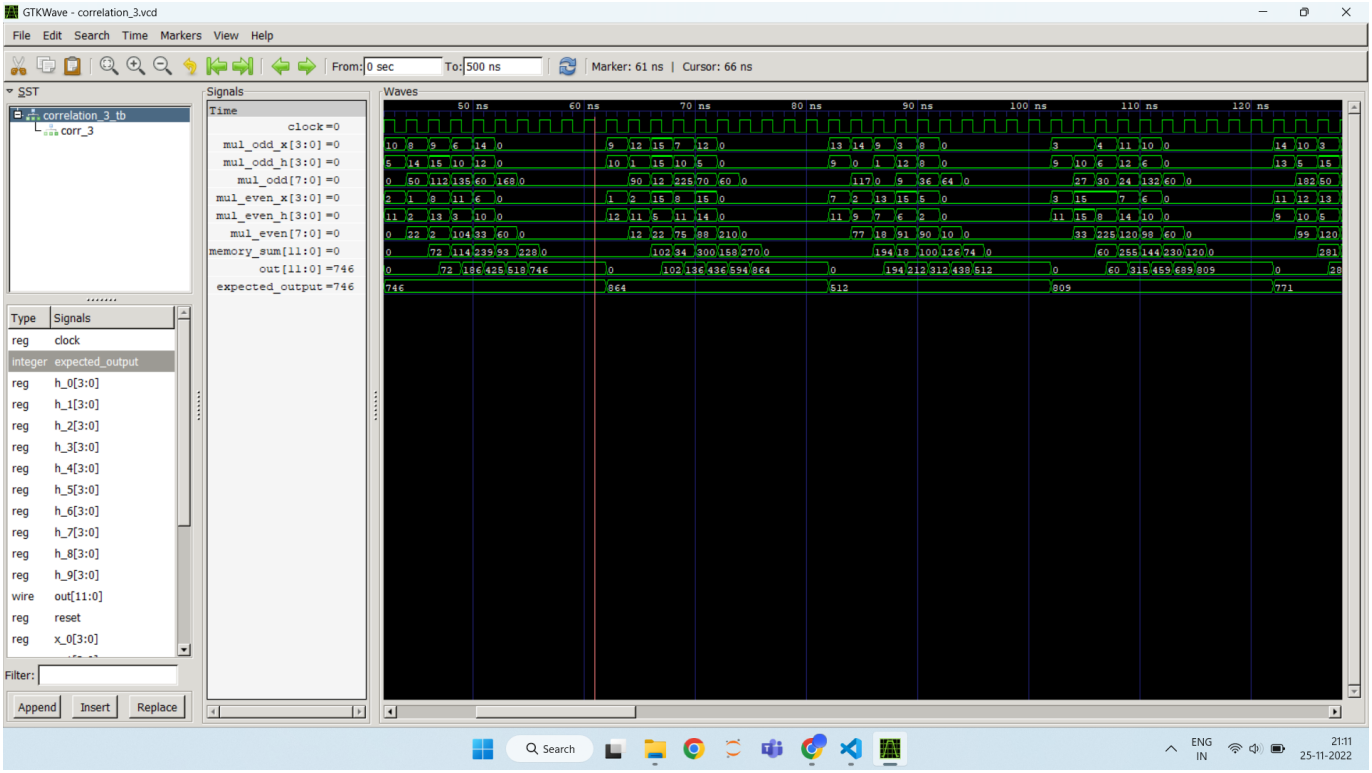
```
      end

      3'b011:
      begin

        mul_even_x <= x_6;
        mul_even_h <= h_6;
        mul_odd_x  <= x_7;
        mul_odd_h  <= h_7;

      end

      3'b100:
      begin

        mul_even_x <= x_8;
        mul_even_h <= h_8;
        mul_odd_x  <= x_9;
        mul_odd_h  <= h_9;

      end

      default:
      begin

        mul_even_x <= 4'b0000;
        mul_even_h <= 4'b0000;
        mul_odd_x  <= 4'b0000;
        mul_odd_h  <= 4'b0000;

      end

    endcase

    mul_even <= mul_even_x * mul_even_h;
    mul_odd <= mul_odd_x * mul_odd_h;
    memory_sum <= mul_even + mul_odd;

  end

  always @(negedge clock)
  begin

        y <= (y + memory_sum);

  end

endmodule
```

Testbench

```
`timescale 1ns / 1ns
`include "correlation_3.v"

module correlation_3_tb ();

  reg [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;
  reg [3:0] h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9;
  reg reset = 1'b0;
  reg clock = 1'b1;

  wire [11:0] out;

  always #1 clock = ~clock;

  initial
  begin

    h_0 <= 4'b0000;
    h_1 <= 4'b0000;
    h_2 <= 4'b0000;
    h_3 <= 4'b0000;
    h_4 <= 4'b0000;
    h_5 <= 4'b0000;
    h_6 <= 4'b0000;
    h_7 <= 4'b0000;
    h_8 <= 4'b0000;
    h_9 <= 4'b0000;

    x_0 <= 4'b0000;
    x_1 <= 4'b0000;
    x_2 <= 4'b0000;
    x_3 <= 4'b0000;
    x_4 <= 4'b0000;
    x_5 <= 4'b0000;
    x_6 <= 4'b0000;
    x_7 <= 4'b0000;
    x_8 <= 4'b0000;
    x_9 <= 4'b0000;

  end

  correlation_3 corr_3 (out, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9,
h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9);

  integer expected_output;

  always @(posedge clock)
  begin
```

```
    h_0 <= $random;
    h_1 <= $random;
    h_2 <= $random;
    h_3 <= $random;
    h_4 <= $random;
    h_5 <= $random;
    h_6 <= $random;
    h_7 <= $random;
    h_8 <= $random;
    h_9 <= $random;

    x_0 <= $random;
    x_1 <= $random;
    x_2 <= $random;
    x_3 <= $random;
    x_4 <= $random;
    x_5 <= $random;
    x_6 <= $random;
    x_7 <= $random;
    x_8 <= $random;
    x_9 <= $random;

    #20;

  end

  always @(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, h_0, h_1, h_2, h_3, h_4, h_5,
h_6, h_7, h_8, h_9)
  begin

    expected_output = (x_0 * h_0) + (x_1 * h_1) + (x_2 * h_2) + (x_3 * h_3) + (x_4 * h_4) +
(x_5 * h_5) + (x_6 * h_6) + (x_7 * h_7) + (x_8 * h_8) + (x_9 * h_9);

  end

  initial begin

    $monitor($time, "x_0 = %d x_1 = %d x_2 = %d x_3 = %d x_4 = %d x_5 = %d x_6 = %d x_7 = %d
x_8 = %d x_9 = %d h_0 = %d h_1 = %d h_2 = %d h_3 = %d h_4 = %d h_5 = %d h_6 = %d h_7 = %d h_8
= %d h_9 = %d output = %d expected = %d", x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9,
h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, out, expected_output);

    $dumpfile("correlation_3.vcd");
    $dumpvars;

    #500 $finish;

  end

endmodule
```

Output in GTKWave




Here in the positive edge of the clock, the product of even x & h and odd x & h are computed separately and their sum is also computed and stored separately and in the coming negative edge the stored sum (from previous positive edge) and current sum are added. This whole operation takes 7 clock cycles to complete. Clearly from the above image, the output and expected outputs match.

# Digital IC Design
## Assignment 1 – Q4

Name- Pushkal Mishra
Roll- EE20BTECH11042

To compute Y such that there is no multiplier module available and should not replace multiplier module with shifted additions or repetitive additions, where Y is given as

$$Y = x_0 h_0 + x_1 h_1 + x_2 h_2 + \ldots\ldots + x_9 h_9$$

Here $h_0$ , $h_1$ , ..... , $h_9$ values are given.

Verilog implementation

```
module memory (product_0, product_1, product_2, product_3, product_4, product_5, product_6,
product_7, product_8, product_9, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8,
x_9);

    output reg [7:0] product_0, product_1, product_2, product_3, product_4, product_5,
product_6, product_7, product_8, product_9;

    input clock, reset;
    input [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;

    reg [7:0] memory_h0 [0:15];
    reg [7:0] memory_h1 [0:15];
    reg [7:0] memory_h2 [0:15];
    reg [7:0] memory_h3 [0:15];
    reg [7:0] memory_h4 [0:15];
    reg [7:0] memory_h5 [0:15];
    reg [7:0] memory_h6 [0:15];
    reg [7:0] memory_h7 [0:15];
    reg [7:0] memory_h8 [0:15];
    reg [7:0] memory_h9 [0:15];

    integer i;
    parameter h_0 = 1;
    parameter h_1 = 2;
    parameter h_2 = 3;
    parameter h_3 = 4;
    parameter h_4 = 5;
    parameter h_5 = 6;
    parameter h_6 = 7;
    parameter h_7 = 8;
    parameter h_8 = 9;
    parameter h_9 = 10;

    initial
    begin
```

```
        product_0 <= 8'b0000_0000;
        product_1 <= 8'b0000_0000;
        product_2 <= 8'b0000_0000;
        product_3 <= 8'b0000_0000;
        product_4 <= 8'b0000_0000;
        product_5 <= 8'b0000_0000;
        product_6 <= 8'b0000_0000;
        product_7 <= 8'b0000_0000;
        product_8 <= 8'b0000_0000;
        product_9 <= 8'b0000_0000;

        for (i = 0; i < 16; i++)
        begin

            memory_h0[i] = (h_0 * i);
            memory_h1[i] = (h_1 * i);
            memory_h2[i] = (h_2 * i);
            memory_h3[i] = (h_3 * i);
            memory_h4[i] = (h_4 * i);
            memory_h5[i] = (h_5 * i);
            memory_h6[i] = (h_6 * i);
            memory_h7[i] = (h_7 * i);
            memory_h8[i] = (h_8 * i);
            memory_h9[i] = (h_9 * i);

        end

    end

    always @(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, reset)
    begin

        if (reset)
        begin

            product_0 <= 8'b0000_0000;
            product_1 <= 8'b0000_0000;
            product_2 <= 8'b0000_0000;
            product_3 <= 8'b0000_0000;
            product_4 <= 8'b0000_0000;
            product_5 <= 8'b0000_0000;
            product_6 <= 8'b0000_0000;
            product_7 <= 8'b0000_0000;
            product_8 <= 8'b0000_0000;
            product_9 <= 8'b0000_0000;

        end

        else
```

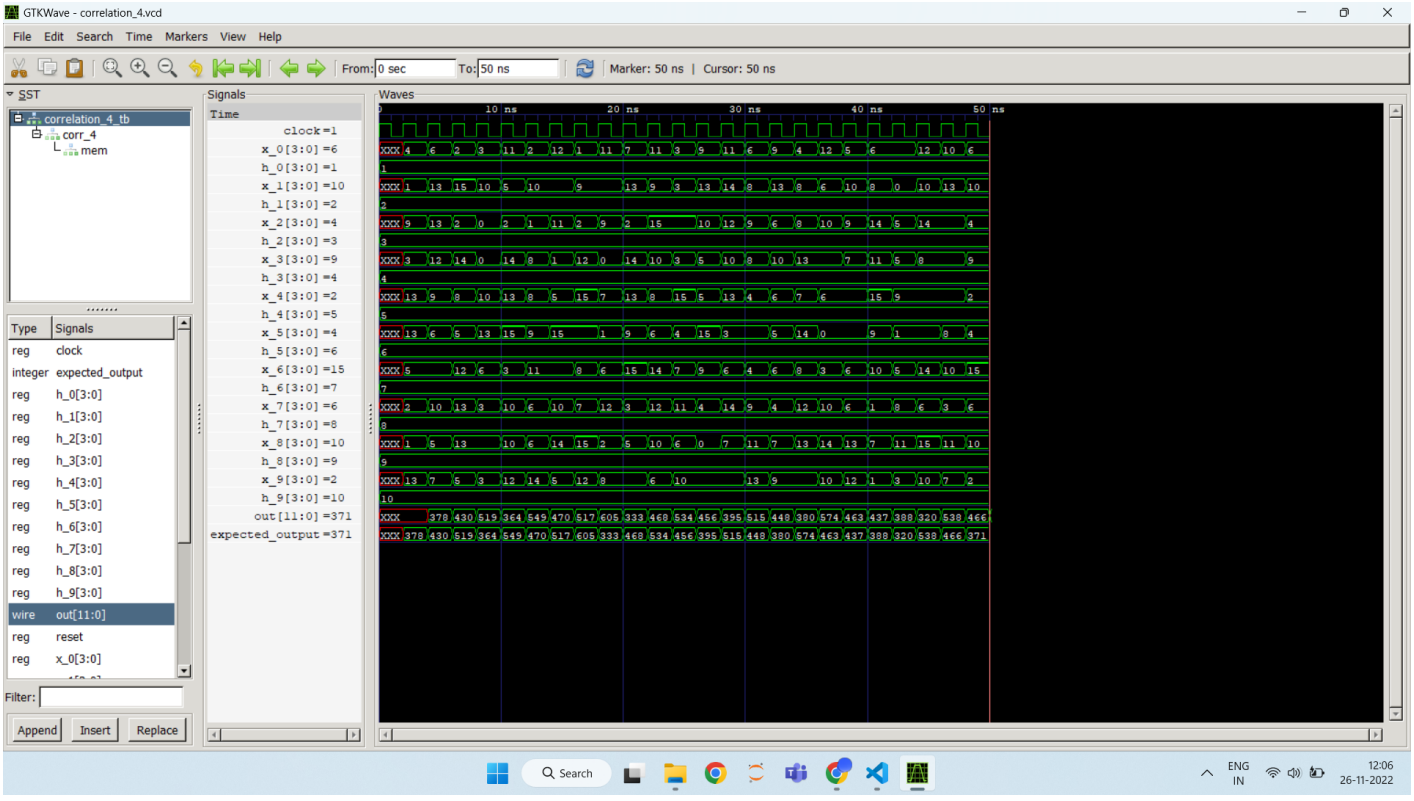
```
        begin

            product_0 <= memory_h0[x_0];
            product_1 <= memory_h1[x_1];
            product_2 <= memory_h2[x_2];
            product_3 <= memory_h3[x_3];
            product_4 <= memory_h4[x_4];
            product_5 <= memory_h5[x_5];
            product_6 <= memory_h6[x_6];
            product_7 <= memory_h7[x_7];
            product_8 <= memory_h8[x_8];
            product_9 <= memory_h9[x_9];

        end
    end

endmodule

module correlation_4 (y, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9);

    output reg [11:0] y;

      input clock, reset;
      input [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;

    wire [7:0] product_0, product_1, product_2, product_3, product_4, product_5, product_6,
product_7, product_8, product_9;

    memory mem (product_0, product_1, product_2, product_3, product_4, product_5, product_6,
product_7, product_8, product_9, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8,
x_9);

    initial
    begin
        y <= 12'b0000_0000_0000;
    end

    always @(posedge clock, reset)
    begin
        if (reset)
        begin
            y <= 12'b0000_0000_0000;
        end
        else
        begin
            y <= (product_0 + product_1 + product_2 + product_3 + product_4 + product_5 +
product_6 + product_7 + product_8 + product_9);
        end
    end
endmodule
```

Testbench

```
`timescale 1ns/1ns
`include "correlation_4.v"

module correlation_4_tb ();

    reg [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;
    reg [3:0] h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9;

    reg reset = 1'b0;
    reg clock = 1'b1;

    wire [11:0] out;

    always #1 clock <= ~clock;

    correlation_4 corr_4 (out, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8,
x_9);

    integer expected_output;

    initial
    begin

        h_0 <= 4'b0001;
        h_1 <= 4'b0010;
        h_2 <= 4'b0011;
        h_3 <= 4'b0100;
        h_4 <= 4'b0101;
        h_5 <= 4'b0110;
        h_6 <= 4'b0111;
        h_7 <= 4'b1000;
        h_8 <= 4'b1001;
        h_9 <= 4'b1010;

        $dumpfile("correlation_4.vcd");
        $dumpvars;

        #50 $finish;

    end

    always @(posedge clock)
    begin

        x_0 <= $random;
        x_1 <= $random;
        x_2 <= $random;
        x_3 <= $random;
        x_4 <= $random;
```

```
        x_5 <= $random;
        x_6 <= $random;
        x_7 <= $random;
        x_8 <= $random;
        x_9 <= $random;

        #2;

    end

    always @(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)
    begin
        expected_output = (x_0 * 32'd1) + (x_1 * 32'd2) + (x_2 * 32'd3) + (x_3 * 32'd4) + (x_4
* 32'd5) + (x_5 * 32'd6) + (x_6 * 32'd7) + (x_7 * 32'd8) + (x_8 * 32'd9) + (x_9 * 32'd10);
    end

endmodule
```

Output in GTKWave




Since we already know the values of $h_0$ , $h_1$ , ..... , $h_9$ , we can pre compute all 16 possible products (since each $x_i$ is 4 bits long) for each coefficient $h_i$ and store it. Then for each $x_i$ we can retrieve the product value from memory and sum it for all $x_i$ . This whole process will take only 1 clock cycle. Clearly from the above image, the output and expected outputs match.

# Digital IC Design
## Assignment 1 – Q5

Name- Pushkal Mishra
Roll- EE20BTECH11042

To compute Y such that there is no multiplier and no memory module available and multiplier should not be replaced with repetitive addition, where Y is given as

$$Y = x_0 h_0 + x_1 h_1 + x_2 h_2 + \ldots\ldots + x_9 h_9$$

Here $h_0$ , $h_1$ , ..... , $h_9$ values are given.

Verilog implementation

```
module retrieve (product_0, product_1, product_2, product_3, product_4, product_5, product_6, product_7, product_8, product_9, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9);

    output reg [7:0] product_0, product_1, product_2, product_3, product_4, product_5, product_6, product_7, product_8, product_9;

    input clock, reset;
    input [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;

    reg [3:0] h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9;

    initial
    begin

        h_0 <= 4'b0101;
        h_1 <= 4'b1010;
        h_2 <= 4'b0010;
        h_3 <= 4'b0110;
        h_4 <= 4'b1101;
        h_5 <= 4'b1110;
        h_6 <= 4'b0001;
        h_7 <= 4'b1001;
        h_8 <= 4'b1000;
        h_9 <= 4'b1111;

        product_0 <= 8'b0000_0000;
        product_1 <= 8'b0000_0000;
        product_2 <= 8'b0000_0000;
        product_3 <= 8'b0000_0000;
        product_4 <= 8'b0000_0000;
        product_5 <= 8'b0000_0000;
        product_6 <= 8'b0000_0000;
        product_7 <= 8'b0000_0000;
        product_8 <= 8'b0000_0000;
        product_9 <= 8'b0000_0000;
```

```
    end

    always @(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, reset)
    begin

        if (reset)
        begin

            product_0 <= 8'b0000_0000;
            product_1 <= 8'b0000_0000;
            product_2 <= 8'b0000_0000;
            product_3 <= 8'b0000_0000;
            product_4 <= 8'b0000_0000;
            product_5 <= 8'b0000_0000;
            product_6 <= 8'b0000_0000;
            product_7 <= 8'b0000_0000;
            product_8 <= 8'b0000_0000;
            product_9 <= 8'b0000_0000;

        end

        else
        begin

            product_0 <= x_0 + (x_0 << 2);
            product_1 <= (x_1 << 1) + (x_1 << 3);
            product_2 <= (x_2 << 1);
            product_3 <= (x_3 << 1) + (x_3 << 2);
            product_4 <= x_4 + (x_4 << 2) + (x_4 << 3);
            product_5 <= (x_5 << 1) + (x_5 << 2) + (x_5 << 3);
            product_6 <= x_6;
            product_7 <= x_7 + (x_7 << 3);
            product_8 <= x_8 << 3;
            product_9 <= x_9 + (x_9 << 1) + (x_9 << 2) + (x_9 << 3);

        end

    end

endmodule

module correlation_5 (y, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9);

    output reg [11:0] y;

      input clock, reset;
      input [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;

    wire [7:0] product_0, product_1, product_2, product_3, product_4, product_5, product_6,
```

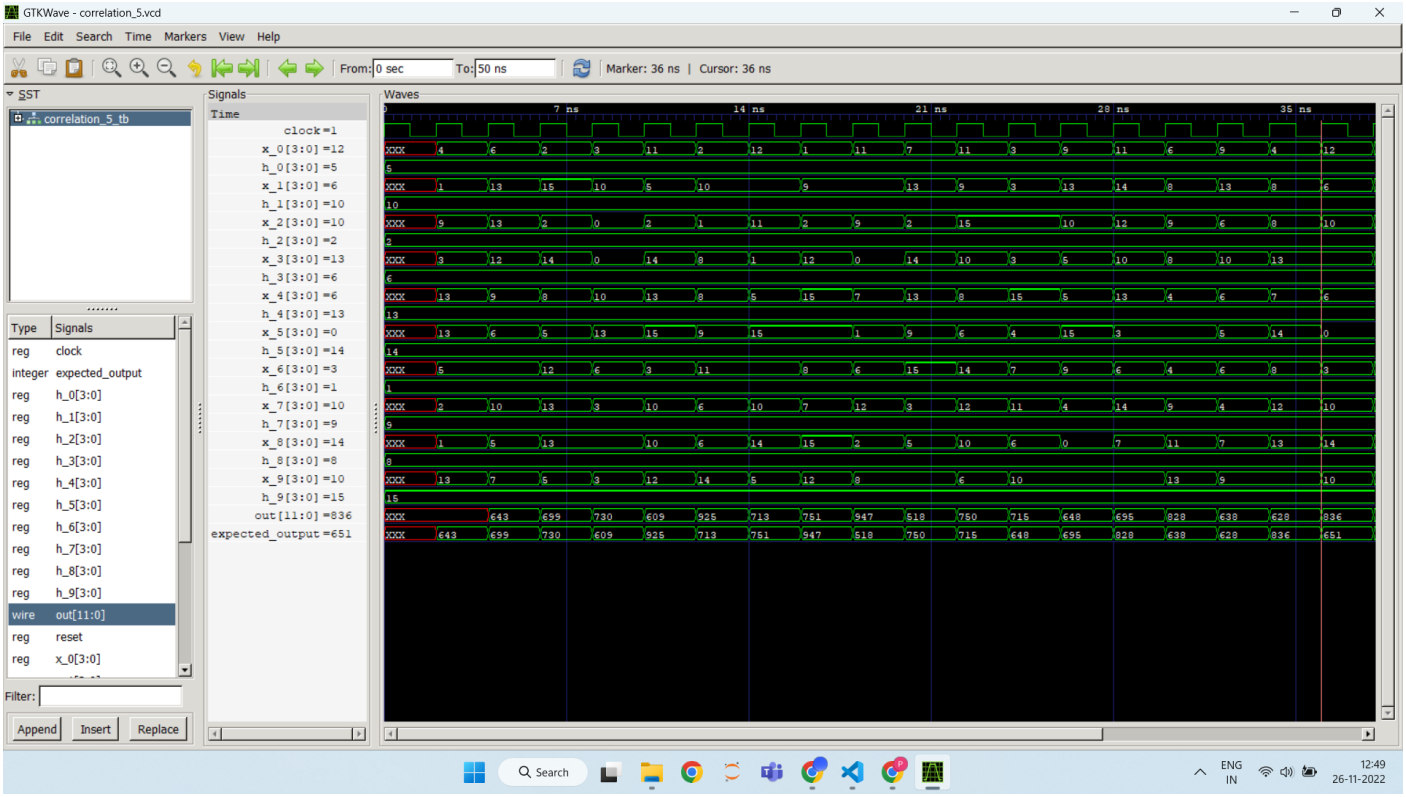
```
product_7, product_8, product_9;

    retrieve ret (product_0, product_1, product_2, product_3, product_4, product_5, product_6,
product_7, product_8, product_9, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8,
x_9);

    always @(posedge clock, reset)
    begin

        if (reset)
        begin
            y <= 12'b0000_0000_0000;
        end

        else
        begin
            y <= (product_0 + product_1 + product_2 + product_3 + product_4 + product_5 +
product_6 + product_7 + product_8 + product_9);
        end

    end

endmodule
```

Testbench

```
`timescale 1ns/1ns
`include "correlation_5.v"

module correlation_5_tb ();

    reg [3:0] x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9;
    reg [3:0] h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9;

    reg reset = 1'b0;
    reg clock = 1'b1;

    wire [11:0] out;

    always #1 clock <= ~clock;

    correlation_5 corr_5 (out, clock, reset, x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8,
x_9);

    integer expected_output;

    initial
    begin

        h_0 <= 4'b0101;
```

```
        h_1 <= 4'b1010;
        h_2 <= 4'b0010;
        h_3 <= 4'b0110;
        h_4 <= 4'b1101;
        h_5 <= 4'b1110;
        h_6 <= 4'b0001;
        h_7 <= 4'b1001;
        h_8 <= 4'b1000;
        h_9 <= 4'b1111;

        $dumpfile("correlation_5.vcd");
        $dumpvars;

        #50 $finish;

    end

    always @(posedge clock)
    begin

        x_0 <= $random;
        x_1 <= $random;
        x_2 <= $random;
        x_3 <= $random;
        x_4 <= $random;
        x_5 <= $random;
        x_6 <= $random;
        x_7 <= $random;
        x_8 <= $random;
        x_9 <= $random;

        #2;

    end

    always @(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)
    begin
        expected_output = (x_0 * h_0) + (x_1 * h_1) + (x_2 * h_2) + (x_3 * h_3) + (x_4 * h_4)
+ (x_5 * h_5) + (x_6 * h_6) + (x_7 * h_7) + (x_8 * h_8) + (x_9 * h_9);
    end

endmodule
```

Output in GTKWave




Since we already know the values of $h_0$ , $h_1$ , ..... , $h_9$ , we can pre determine the number of shifts required for each $x_i$ and add all the shifted versions to obtain the product. This whole operation only takes one clock cycle. Clearly from the above image, the output and expected outputs match.