Structural Multiplier

- Verilog Code:

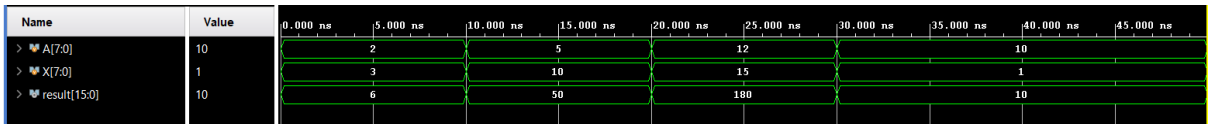
```
`timescale 1ns / 1ps
module MULTS(
    input [7:0] A, X,
    output [15:0] result
    );

    wire [15:0] PP [7:0];
    wire [15:0] sum [6:0];
    wire cout[6:0];
    genvar i;
    generate
        for (i=0; i<8; i=i+1) begin
            assign PP[i] = (X[i] * A)<<i;
        end
    endgenerate

    CLA_16bit sum1 (PP[0], PP[1], 0, cout[0], sum[0]);
    CLA_16bit sum2 (PP[2], PP[3], 0, cout[1], sum[1]);
    CLA_16bit  sum3 (PP[4], PP[5], 0, cout[2], sum[2]);
    CLA_16bit  sum4 (PP[6], PP[7], 0, cout[3], sum[3]);
    CLA_16bit  sum5 (sum[0], sum[1], 0, cout[4], sum[4]);
    CLA_16bit  sum6 (sum[2], sum[3], 0, cout[5], sum[5]);
    CLA_16bit  sum7 (sum[4], sum[5], 0, cout[6], sum[6]);
    assign result = sum[6];

endmodule
```

- Testbench Code:

```
`timescale 1ns / 1ps

module tb_MULTS;

    reg [7:0] A, X;
    wire [15:0] result;

    MULTS uut (
        .A(A),
        .X(X),
        .result(result)
    );
```

```
    initial begin
        // Test case 1
        A = 8'b0010;  // 2
        X = 8'b0011;  // 3
        #10;

        A = 8'b0101;  // 5
        X = 8'b1010;  // 10
        #10;

        A = 8'b1100;  // 12
        X = 8'b1111;  // 15
        #10;

        A = 8'b1010;  // 10
        X = 8'b0001;  // 1
        #10;
        $stop;
    end

endmodule
```

- Simulation Result:



Behavioral Multiplier

- Verilog Code:

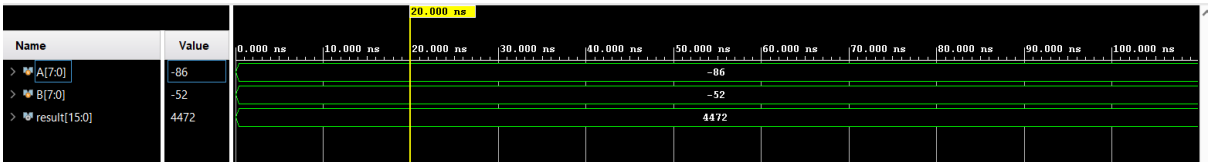
```
module MULTB(
input signed [7:0] A,B,
output reg signed  [15:0] result
);
always @(*) begin
result = A * B;
endA
endmodule
```

- Testbench Code:

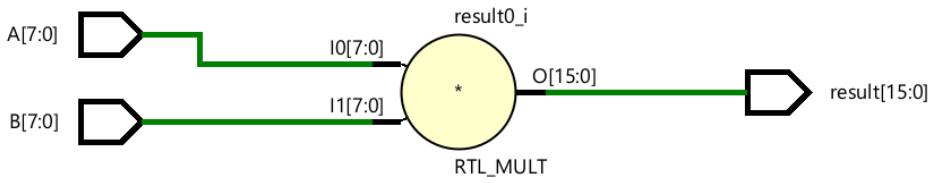
```
`timescale 1ns/1ps

module tb_MULTB;
```

```
  reg signed [7:0] A, B;
  wire signed [15:0] result;
  MULTB uut (
    .A(A),
    .B(B),
    .result(result)
  );

  initial begin
    A = 8'b10101010;
    B = 8'b11001100;
    $finish;
  end
endmodule
```

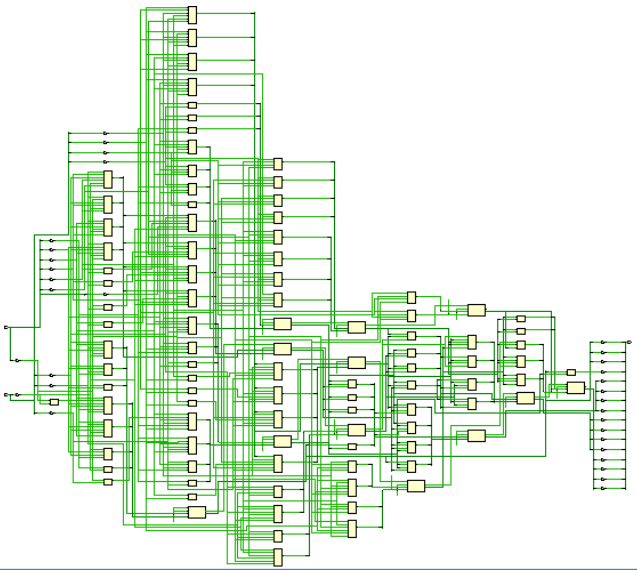
- Simulation Result



- RTL Schematic



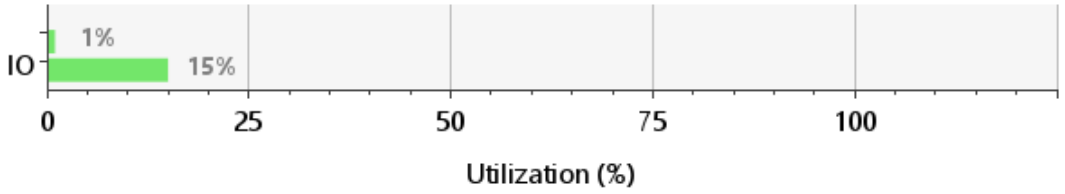
- Technology Schematic



Utilization and Timing Reports

61 LUTs are used and maximum combinational delay is 12.889 ns.

Resource	Utilization	Available	Utilization %
LUT	61	63400	0.10
IO	32	210	15.24



A[4]	result[13]	12.889	SLOW	3.123	FAST
A[2]	result[13]	12.869	SLOW	3.280	FAST
B[1]	result[13]	12.844	SLOW	3.281	FAST
A[4]	result[14]	12.796	SLOW	3.090	FAST
A[2]	result[14]	12.777	SLOW	3.251	FAST
A[4]	result[15]	12.771	SLOW	3.108	FAST
B[1]	result[14]	12.752	SLOW	3.248	FAST
A[2]	result[15]	12.751	SLOW	3.224	FAST
B[2]	result[13]	12.726	SLOW	3.253	FAST
B[1]	result[15]	12.726	SLOW	3.266	FAST
A[6]	result[13]	12.690	SLOW	3.055	FAST
A[3]	result[13]	12.658	SLOW	3.343	FAST
B[2]	result[14]	12.634	SLOW	3.220	FAST
A[4]	result[12]	12.628	SLOW	3.023	FAST
B[2]	result[15]	12.608	SLOW	3.239	FAST
A[2]	result[12]	12.608	SLOW	3.184	FAST
A[6]	result[14]	12.597	SLOW	3.119	FAST
B[1]	result[12]	12.583	SLOW	3.181	FAST
A[6]	result[15]	12.572	SLOW	3.137	FAST
A[3]	result[14]	12.565	SLOW	3.398	FAST
A[3]	result[15]	12.540	SLOW	3.416	FAST
A[4]	result[11]	12.509	SLOW	3.011	FAST
A[2]	result[11]	12.489	SLOW	3.100	FAST
B[2]	result[12]	12.465	SLOW	3.154	FAST
B[1]	result[11]	12.464	SLOW	3.086	FAST
A[4]	result[10]	12.456	SLOW	3.022	FAST
A[2]	result[10]	12.436	SLOW	3.147	FAST

A[6]	result[12]	12.429	SLOW	3.052	FAST
B[1]	result[10]	12.411	SLOW	3.133	FAST
A[1]	result[13]	12.400	SLOW	3.285	FAST
A[3]	result[12]	12.397	SLOW	3.331	FAST
B[3]	result[13]	12.389	SLOW	3.084	FAST
B[4]	result[13]	12.368	SLOW	3.154	FAST
A[3]	result[11]	12.356	SLOW	3.231	FAST
B[2]	result[11]	12.346	SLOW	3.059	FAST
A[5]	result[13]	12.336	SLOW	3.162	FAST
A[2]	result[9]	12.332	SLOW	2.989	FAST
A[6]	result[11]	12.310	SLOW	2.943	FAST
A[1]	result[14]	12.307	SLOW	3.340	FAST
B[3]	result[14]	12.297	SLOW	3.102	FAST
B[2]	result[10]	12.293	SLOW	3.106	FAST
A[3]	result[10]	12.292	SLOW	3.304	FAST
A[1]	result[15]	12.282	SLOW	3.358	FAST
B[4]	result[14]	12.276	SLOW	3.131	FAST
B[3]	result[15]	12.271	SLOW	3.120	FAST
B[5]	result[13]	12.262	SLOW	2.946	FAST
A[6]	result[10]	12.257	SLOW	3.008	FAST
B[4]	result[15]	12.250	SLOW	3.149	FAST
A[5]	result[14]	12.244	SLOW	3.214	FAST
A[3]	result[9]	12.230	SLOW	3.220	FAST
A[5]	result[15]	12.218	SLOW	3.232	FAST
A[4]	result[9]	12.186	SLOW	2.963	FAST
B[0]	result[13]	12.182	SLOW	3.209	FAST
B[5]	result[14]	12.170	SLOW	2.963	FAST

A[2]	result[8]	12.160	SLOW	3.021	FAST
B[5]	result[15]	12.144	SLOW	2.982	FAST
A[1]	result[12]	12.139	SLOW	3.273	FAST
B[3]	result[12]	12.128	SLOW	3.035	FAST
A[0]	result[13]	12.125	SLOW	3.253	FAST
B[4]	result[12]	12.107	SLOW	3.065	FAST
B[0]	result[14]	12.089	SLOW	3.177	FAST
A[5]	result[12]	12.075	SLOW	3.147	FAST
B[0]	result[15]	12.064	SLOW	3.195	FAST
A[3]	result[8]	12.058	SLOW	3.190	FAST
A[0]	result[14]	12.033	SLOW	3.249	FAST
A[1]	result[11]	12.020	SLOW	3.173	FAST
A[4]	result[8]	12.014	SLOW	2.882	FAST
B[3]	result[11]	12.009	SLOW	2.961	FAST
A[0]	result[15]	12.007	SLOW	3.267	FAST
B[5]	result[12]	12.001	SLOW	2.886	FAST
B[4]	result[11]	11.988	SLOW	3.041	FAST
A[1]	result[10]	11.967	SLOW	3.272	FAST
B[1]	result[9]	11.963	SLOW	3.110	FAST
B[3]	result[10]	11.956	SLOW	3.034	FAST
A[5]	result[11]	11.956	SLOW	3.050	FAST
B[4]	result[10]	11.935	SLOW	3.064	FAST
B[0]	result[12]	11.921	SLOW	3.110	FAST
A[5]	result[10]	11.903	SLOW	3.099	FAST
B[5]	result[11]	11.882	SLOW	2.788	FAST
A[0]	result[12]	11.864	SLOW	3.182	FAST
A[2]	result[7]	11.837	SLOW	3.086	FAST

B[5]	result[10]	11.829	SLOW	3.139	FAST
B[0]	result[11]	11.802	SLOW	3.015	FAST
B[2]	result[9]	11.799	SLOW	3.078	FAST
B[1]	result[8]	11.792	SLOW	3.056	FAST
A[6]	result[9]	11.763	SLOW	2.984	FAST
B[0]	result[10]	11.749	SLOW	3.062	FAST
A[0]	result[11]	11.745	SLOW	3.087	FAST
A[3]	result[7]	11.736	SLOW	3.100	FAST
A[0]	result[10]	11.692	SLOW	3.134	FAST
A[4]	result[7]	11.692	SLOW	2.792	FAST
B[0]	result[9]	11.659	SLOW	3.039	FAST
A[7]	result[13]	11.597	SLOW	3.149	FAST
A[1]	result[9]	11.557	SLOW	3.188	FAST
B[2]	result[8]	11.534	SLOW	3.025	FAST
A[7]	result[14]	11.505	SLOW	3.117	FAST
B[0]	result[8]	11.487	SLOW	2.985	FAST
A[7]	result[15]	11.479	SLOW	3.135	FAST
B[1]	result[7]	11.469	SLOW	3.086	FAST
B[3]	result[9]	11.462	SLOW	2.975	FAST
B[4]	result[9]	11.441	SLOW	3.004	FAST
A[5]	result[9]	11.409	SLOW	3.036	FAST
A[1]	result[8]	11.385	SLOW	3.132	FAST
A[0]	result[9]	11.378	SLOW	3.111	FAST
A[3]	result[6]	11.343	SLOW	3.245	FAST
A[7]	result[12]	11.336	SLOW	3.050	FAST
B[5]	result[9]	11.335	SLOW	3.075	FAST
A[2]	result[6]	11.226	SLOW	3.085	FAST

A[7]	result[11]	11.217	SLOW	2.955	FAST
B[2]	result[7]	11.211	SLOW	3.061	FAST
A[0]	result[8]	11.206	SLOW	3.057	FAST
B[0]	result[7]	11.165	SLOW	2.960	FAST
A[7]	result[10]	11.164	SLOW	3.002	FAST
B[3]	result[8]	11.137	SLOW	2.894	FAST
A[6]	result[8]	11.123	SLOW	2.931	FAST
B[7]	result[13]	11.114	SLOW	2.941	FAST
A[5]	result[8]	11.078	SLOW	2.979	FAST
A[1]	result[7]	11.062	SLOW	3.041	FAST
B[7]	result[14]	11.021	SLOW	2.979	FAST
B[7]	result[15]	10.996	SLOW	2.850	FAST
A[0]	result[7]	10.883	SLOW	3.009	FAST
B[7]	result[12]	10.852	SLOW	2.884	FAST
B[6]	result[13]	10.843	SLOW	3.005	FAST
B[4]	result[8]	10.820	SLOW	2.923	FAST
B[3]	result[7]	10.814	SLOW	2.840	FAST
B[5]	result[8]	10.810	SLOW	2.999	FAST
A[1]	result[6]	10.770	SLOW	3.087	FAST
A[5]	result[7]	10.755	SLOW	2.987	FAST
B[6]	result[14]	10.750	SLOW	3.021	FAST
B[0]	result[6]	10.736	SLOW	3.040	FAST
B[7]	result[11]	10.734	SLOW	2.895	FAST
B[6]	result[15]	10.725	SLOW	2.867	FAST
B[7]	result[10]	10.681	SLOW	2.964	FAST
A[7]	result[9]	10.670	SLOW	2.892	FAST
A[3]	result[5]	10.630	SLOW	3.170	FAST

B[1]	result[6]	10.628	SLOW	3.080	FAST
B[2]	result[6]	10.600	SLOW	3.083	FAST
B[6]	result[12]	10.581	SLOW	2.948	FAST
A[4]	result[6]	10.527	SLOW	3.030	FAST
A[2]	result[5]	10.514	SLOW	3.058	FAST
B[4]	result[7]	10.498	SLOW	2.833	FAST
A[0]	result[6]	10.490	SLOW	3.069	FAST
B[5]	result[7]	10.488	SLOW	2.908	FAST
B[6]	result[11]	10.463	SLOW	2.929	FAST
B[6]	result[10]	10.410	SLOW	2.971	FAST
A[3]	result[4]	10.382	SLOW	3.154	FAST
A[6]	result[7]	10.328	SLOW	2.863	FAST
A[2]	result[4]	10.201	SLOW	3.013	FAST
B[7]	result[9]	10.186	SLOW	2.859	FAST
B[4]	result[6]	10.106	SLOW	2.955	FAST
A[1]	result[5]	10.058	SLOW	3.186	FAST
B[3]	result[6]	10.051	SLOW	2.864	FAST
A[7]	result[8]	10.030	SLOW	2.879	FAST
B[0]	result[5]	10.023	SLOW	2.988	FAST
B[6]	result[9]	10.001	SLOW	2.804	FAST
B[1]	result[5]	9.916	SLOW	3.053	FAST
A[5]	result[6]	9.889	SLOW	3.002	FAST
B[2]	result[5]	9.888	SLOW	3.031	FAST
B[6]	result[8]	9.829	SLOW	2.791	FAST
A[0]	result[5]	9.778	SLOW	2.988	FAST
B[0]	result[4]	9.775	SLOW	2.971	FAST
A[1]	result[4]	9.709	SLOW	3.178	FAST

B[5]	result[6]	9.682	SLOW	2.974	FAST
B[2]	result[4]	9.575	SLOW	3.015	FAST
A[4]	result[5]	9.543	SLOW	2.955	FAST
A[0]	result[4]	9.530	SLOW	2.980	FAST
B[1]	result[4]	9.519	SLOW	3.008	FAST
B[6]	result[7]	9.506	SLOW	2.747	FAST
A[3]	result[3]	9.476	SLOW	3.235	FAST
B[7]	result[8]	9.475	SLOW	2.847	FAST
B[4]	result[5]	9.394	SLOW	2.929	FAST
B[3]	result[5]	9.339	SLOW	2.837	FAST
A[7]	result[7]	9.236	SLOW	3.101	FAST
A[5]	result[5]	9.177	SLOW	3.142	FAST
A[1]	result[3]	8.976	SLOW	2.992	FAST
B[5]	result[5]	8.970	SLOW	3.226	FAST
B[0]	result[3]	8.869	SLOW	2.765	FAST
A[0]	result[3]	8.817	SLOW	2.769	FAST
B[1]	result[3]	8.767	SLOW	2.780	FAST
B[7]	result[7]	8.687	SLOW	2.854	FAST
A[4]	result[4]	8.649	SLOW	2.938	FAST
B[4]	result[4]	8.568	SLOW	2.921	FAST
A[6]	result[6]	8.562	SLOW	2.911	FAST
B[3]	result[4]	8.480	SLOW	2.790	FAST
B[6]	result[6]	8.342	SLOW	2.812	FAST
B[2]	result[3]	8.331	SLOW	2.801	FAST
A[1]	result[2]	8.212	SLOW	2.746	FAST
A[2]	result[3]	8.174	SLOW	2.785	FAST
B[1]	result[2]	8.003	SLOW	2.531	FAST

A[1]	result[2]	8.212	SLOW	2.746	FAST
A[2]	result[3]	8.174	SLOW	2.785	FAST
B[1]	result[2]	8.003	SLOW	2.531	FAST
B[3]	result[3]	7.907	SLOW	2.666	FAST
A[0]	result[2]	7.893	SLOW	2.521	FAST
A[1]	result[1]	7.846	SLOW	2.718	FAST
A[0]	result[1]	7.719	SLOW	2.559	FAST
B[1]	result[1]	7.638	SLOW	2.612	FAST
A[0]	result[0]	7.543	SLOW	2.554	FAST
B[2]	result[2]	7.523	SLOW	2.552	FAST
B[0]	result[2]	7.506	SLOW	2.493	FAST
A[2]	result[2]	7.366	SLOW	2.534	FAST
B[0]	result[1]	7.333	SLOW	2.438	FAST
B[0]	result[0]	7.156	SLOW	2.434	FAST

Structural Multiplier

- Verilog Code:

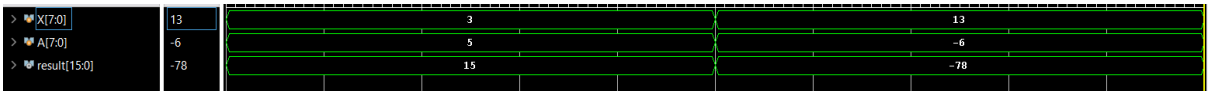
```
module MULTS_signed(
input [7:0] A,X,
output [15:0] result
);
wire [7:0] PP [7:0];
wire [15:0] PP_shifted [8:0];
wire [15:0] sum [7:0];
wire cout[7:0];

genvar i;
generate
    for (i=0; i<7; i=i+1) begin
        assign PP[i][6:0] = (X[i] * A[6:0]);
        assign PP[i][7] = ~(X[i] * A[7]);
    end
    assign PP[7][6:0]=~(X[7]*A[6:0]);
    assign PP[7][7]=X[7] * A[7];
endgenerate

generate
    for (i=0; i<8; i=i+1) begin
        assign PP_shifted[i]=PP[i]<<i;
    end
endgenerate
    assign PP_shifted[8]=16'b1000_0001_0000_0000;
    CLA_16bit s0 (.x(PP_shifted[0][15:0]), .y(PP_shifted[4][15:0]),
.c0(0), .cout(cout[0]), .sum(sum[0][15:0]));
    CLA_16bit s2 (.x(PP_shifted[1][15:0]), .y(PP_shifted[5][15:0]),
.c0(0), .cout(cout[1]), .sum(sum[1][15:0]));
    CLA_16bit s3 (.x(PP_shifted[2][15:0]), .y(PP_shifted[6][15:0]),
.c0(0), .cout(cout[2]), .sum(sum[2][15:0]));
    CLA_16bit s4 (.x(PP_shifted[3][15:0]), .y(PP_shifted[7][15:0]),
.c0(0), .cout(cout[3]), .sum(sum[3][15:0]));
    CLA_16bit s5 (.x(sum[0][15:0]), .y(sum[1][15:0]), .c0(0),
.cout(cout[4]), .sum(sum[4][15:0]));
    CLA_16bit s6 (.x(sum[2][15:0]), .y(sum[3][15:0]), .c0(0),
.cout(cout[5]), .sum(sum[5][15:0]));
    CLA_16bit s7 (.x(sum[4][15:0]), .y(sum[5][15:0]), .c0(0),
.cout(cout[6]), .sum(sum[6][15:0]));
```

```
    CLA_16bit s8 (.x(PP_shifted[8][15:0]), .y(sum[6][15:0]), .c0(0),
.cout(cout[7]), .sum(sum[7][15:0]));
    assign result = sum[7];

endmodule
```

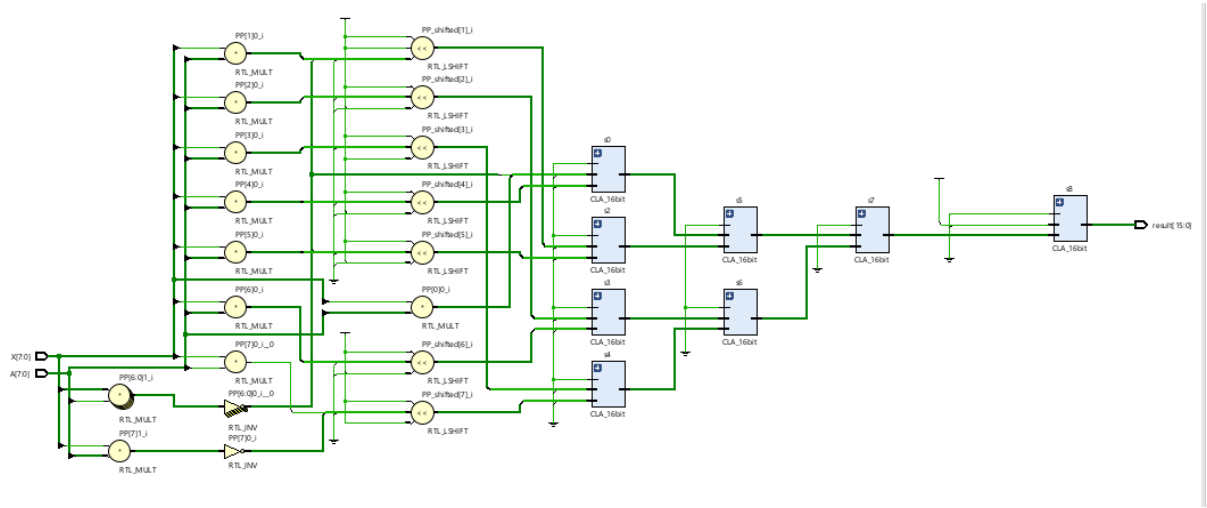
- Testbench Code:

```
`timescale 1ns/1ps

module tb_MULTS_signed;
  reg signed [7:0] A;
  reg signed [7:0] X;
  wire signed [15:0] result;
  MULTS_signed uut (
    .A(A),
    .X(X),
    .result(result)
  );

  initial begin
    // Test case 1: A = 5, X = 3
    A = 8'b00000101;
    X = 8'b00000011;
    #10;

    A = 8'b11111010;
    X = 8'b00001101;
    #10;

    $finish;
  end

endmodule
```

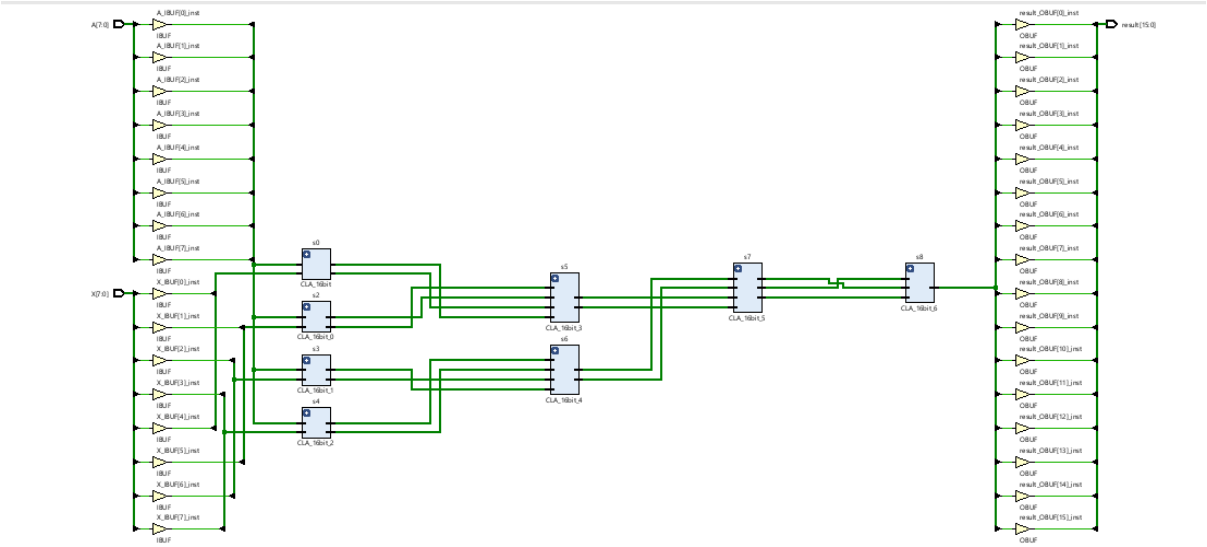
- Simulation Result



- RTL Schematic

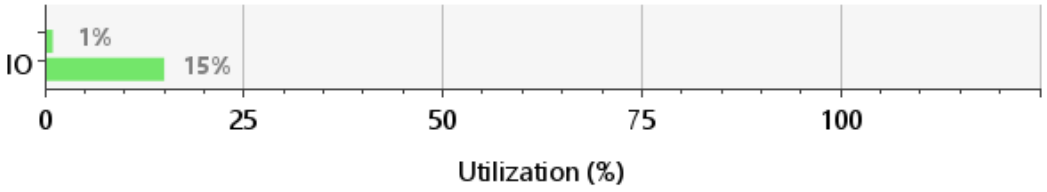


- Technology Schematic



- Utilization and Timing Reports
- 374 LUTs are used and maximum combinational delay is 23.412

Resource	Utilization	Available	Utilization %
LUT	374	63400	0.59
IO	32	210	15.24



From Port	To Port	Ma	Max Process Corner	Min Delay	Min Process Corner
X[0]	result[15]	23.412	SLOW	5.142	FAST
A[1]	result[15]	23.054	SLOW	5.267	FAST
A[3]	result[15]	22.650	SLOW	5.009	FAST
X[0]	result[14]	22.554	SLOW	4.840	FAST
X[1]	result[15]	22.343	SLOW	5.170	FAST
A[0]	result[15]	22.326	SLOW	5.590	FAST
A[1]	result[14]	22.196	SLOW	4.965	FAST
A[2]	result[15]	22.099	SLOW	5.288	FAST
A[3]	result[14]	21.791	SLOW	4.707	FAST
X[0]	result[13]	21.615	SLOW	4.580	FAST
X[2]	result[15]	21.497	SLOW	4.843	FAST
X[1]	result[14]	21.485	SLOW	4.867	FAST
A[0]	result[14]	21.468	SLOW	5.287	FAST
A[4]	result[15]	21.323	SLOW	4.939	FAST
X[4]	result[15]	21.296	SLOW	4.473	FAST
A[1]	result[13]	21.257	SLOW	4.706	FAST
A[2]	result[14]	21.241	SLOW	4.985	FAST
X[0]	result[12]	21.037	SLOW	4.325	FAST
A[3]	result[13]	20.853	SLOW	4.448	FAST
X[5]	result[15]	20.726	SLOW	4.487	FAST
A[1]	result[12]	20.679	SLOW	4.451	FAST
X[2]	result[14]	20.638	SLOW	4.541	FAST
A[5]	result[15]	20.550	SLOW	4.408	FAST
X[1]	result[13]	20.546	SLOW	4.608	FAST
A[0]	result[13]	20.529	SLOW	5.028	FAST
A[4]	result[14]	20.465	SLOW	4.674	FAST
X[3]	result[15]	20.438	SLOW	4.792	FAST

From Port	To Port	Ma	Max Process Corner	Min Delay	Min Process Corner
X[4]	result[14]	20.437	SLOW	4.207	FAST
A[2]	result[13]	20.302	SLOW	4.726	FAST
A[3]	result[12]	20.274	SLOW	4.193	FAST
X[1]	result[12]	19.968	SLOW	4.353	FAST
A[0]	result[12]	19.951	SLOW	4.773	FAST
X[5]	result[14]	19.867	SLOW	4.222	FAST
X[0]	result[11]	19.765	SLOW	3.968	FAST
A[2]	result[12]	19.724	SLOW	4.471	FAST
X[2]	result[13]	19.700	SLOW	4.282	FAST
A[5]	result[14]	19.692	SLOW	4.142	FAST
A[6]	result[15]	19.640	SLOW	4.641	FAST
X[3]	result[14]	19.580	SLOW	4.489	FAST
A[4]	result[13]	19.526	SLOW	4.500	FAST
X[4]	result[13]	19.499	SLOW	4.047	FAST
A[1]	result[11]	19.407	SLOW	4.093	FAST
X[2]	result[12]	19.121	SLOW	4.027	FAST
A[3]	result[11]	19.003	SLOW	3.854	FAST
X[0]	result[10]	18.964	SLOW	3.968	FAST
A[4]	result[12]	18.948	SLOW	4.325	FAST
X[5]	result[13]	18.929	SLOW	4.029	FAST
X[4]	result[12]	18.920	SLOW	3.792	FAST
A[6]	result[14]	18.781	SLOW	4.375	FAST
X[7]	result[15]	18.780	SLOW	4.378	FAST
A[5]	result[13]	18.753	SLOW	4.147	FAST
A[7]	result[15]	18.727	SLOW	4.508	FAST
X[1]	result[11]	18.696	SLOW	3.996	FAST
A[0]	result[11]	18.679	SLOW	4.416	FAST

From Port	To Port	M a 1	Max Process Corner	Min Delay	Min Process Corner
A[1]	result[10]	18.606	SLOW	4.134	FAST
X[6]	result[15]	18.459	SLOW	4.094	FAST
A[2]	result[11]	18.453	SLOW	4.133	FAST
X[5]	result[12]	18.350	SLOW	3.774	FAST
X[0]	result[9]	18.269	SLOW	3.778	FAST
A[3]	result[10]	18.201	SLOW	3.862	FAST
A[5]	result[12]	18.175	SLOW	3.960	FAST
X[3]	result[12]	18.063	SLOW	3.975	FAST
A[0]	result[10]	17.986	SLOW	4.416	FAST
A[1]	result[9]	17.911	SLOW	3.943	FAST
X[1]	result[10]	17.895	SLOW	3.996	FAST
A[7]	result[14]	17.868	SLOW	4.251	FAST
X[7]	result[14]	17.860	SLOW	4.121	FAST
X[2]	result[11]	17.850	SLOW	3.669	FAST
A[6]	result[13]	17.842	SLOW	4.202	FAST
A[2]	result[10]	17.651	SLOW	4.151	FAST
X[6]	result[14]	17.580	SLOW	3.828	FAST
A[4]	result[11]	17.418	SLOW	4.026	FAST
X[4]	result[11]	17.390	SLOW	3.580	FAST
A[0]	result[9]	17.366	SLOW	4.226	FAST
A[6]	result[12]	17.264	SLOW	4.024	FAST
X[2]	result[10]	17.241	SLOW	3.710	FAST
X[0]	result[8]	17.241	SLOW	3.476	FAST
A[3]	result[9]	17.118	SLOW	3.672	FAST
X[5]	result[11]	17.079	SLOW	3.562	FAST
X[1]	result[9]	16.966	SLOW	3.806	FAST
A[2]	result[9]	16.956	SLOW	3.961	FAST

From Port	To Port	M a 1	Max Process Corner	Min Delay	Min Process Corner
A[7]	result[13]	16.930	SLOW	4.348	FAST
X[7]	result[13]	16.922	SLOW	4.284	FAST
A[5]	result[11]	16.904	SLOW	3.748	FAST
A[1]	result[8]	16.883	SLOW	3.718	FAST
A[4]	result[10]	16.726	SLOW	4.026	FAST
X[4]	result[10]	16.699	SLOW	3.692	FAST
X[6]	result[13]	16.641	SLOW	3.866	FAST
X[2]	result[9]	16.620	SLOW	3.519	FAST
X[3]	result[11]	16.561	SLOW	3.637	FAST
A[7]	result[12]	16.351	SLOW	4.093	FAST
X[7]	result[12]	16.343	SLOW	4.090	FAST
A[0]	result[8]	16.337	SLOW	3.945	FAST
X[5]	result[10]	16.277	SLOW	3.868	FAST
X[0]	result[7]	16.154	SLOW	3.345	FAST
A[5]	result[10]	16.102	SLOW	4.036	FAST
X[6]	result[12]	16.063	SLOW	3.872	FAST
X[0]	result[6]	16.039	SLOW	3.387	FAST
X[1]	result[8]	15.937	SLOW	3.524	FAST
A[2]	result[8]	15.928	SLOW	3.728	FAST
A[3]	result[8]	15.891	SLOW	3.370	FAST
X[3]	result[10]	15.873	SLOW	3.720	FAST
A[1]	result[7]	15.796	SLOW	3.588	FAST
A[6]	result[11]	15.730	SLOW	3.686	FAST
A[1]	result[6]	15.681	SLOW	3.644	FAST
X[2]	result[8]	15.591	SLOW	3.752	FAST
A[4]	result[9]	15.533	SLOW	3.836	FAST
X[4]	result[9]	15.505	SLOW	3.608	FAST

From Port	To Port	Ma 1	Max Process Corner	Min Delay	Min Process Corner
A[6]	result[10]	15.042	SLOW	3.768	FAST
A[5]	result[9]	15.019	SLOW	3.846	FAST
X[3]	result[9]	14.868	SLOW	3.643	FAST
X[1]	result[7]	14.850	SLOW	3.393	FAST
A[2]	result[7]	14.841	SLOW	3.597	FAST
A[7]	result[11]	14.818	SLOW	3.735	FAST
A[3]	result[7]	14.804	SLOW	3.239	FAST
X[0]	result[5]	14.783	SLOW	3.417	FAST
X[7]	result[11]	14.752	SLOW	3.751	FAST
X[1]	result[6]	14.736	SLOW	3.553	FAST
A[2]	result[6]	14.726	SLOW	3.532	FAST
A[3]	result[6]	14.689	SLOW	3.894	FAST
X[2]	result[7]	14.504	SLOW	3.621	FAST
X[0]	result[4]	14.490	SLOW	3.861	FAST
A[1]	result[5]	14.425	SLOW	3.591	FAST
X[6]	result[11]	14.399	SLOW	3.675	FAST
X[2]	result[6]	14.390	SLOW	3.719	FAST
A[4]	result[8]	14.242	SLOW	3.555	FAST
X[4]	result[8]	14.214	SLOW	3.361	FAST
A[1]	result[4]	14.132	SLOW	3.923	FAST
A[7]	result[10]	14.130	SLOW	3.735	FAST
X[7]	result[10]	14.064	SLOW	3.791	FAST
X[3]	result[8]	13.896	SLOW	3.533	FAST
A[0]	result[5]	13.879	SLOW	3.567	FAST
A[6]	result[9]	13.729	SLOW	3.596	FAST
X[6]	result[10]	13.711	SLOW	3.715	FAST
A[0]	result[4]	13.587	SLOW	3.882	FAST

From Port	To Port	Ma 1	Max Process Corner	Min Delay	Min Process Corner
X[1]	result[5]	13.479	SLOW	3.416	FAST
A[2]	result[5]	13.470	SLOW	3.624	FAST
A[3]	result[5]	13.433	SLOW	3.740	FAST
A[5]	result[8]	13.375	SLOW	3.550	FAST
X[0]	result[3]	13.248	SLOW	4.015	FAST
X[1]	result[4]	13.187	SLOW	3.782	FAST
A[2]	result[4]	13.177	SLOW	3.894	FAST
X[2]	result[5]	13.134	SLOW	3.387	FAST
A[3]	result[4]	13.046	SLOW	3.928	FAST
A[1]	result[3]	12.890	SLOW	3.829	FAST
X[2]	result[4]	12.841	SLOW	3.657	FAST
X[7]	result[9]	12.841	SLOW	3.601	FAST
A[7]	result[9]	12.817	SLOW	3.545	FAST
X[3]	result[7]	12.752	SLOW	3.435	FAST
A[6]	result[8]	12.558	SLOW	3.463	FAST
X[4]	result[7]	12.538	SLOW	3.230	FAST
A[4]	result[7]	12.516	SLOW	3.425	FAST
X[3]	result[6]	12.447	SLOW	3.643	FAST
X[6]	result[9]	12.433	SLOW	3.525	FAST
X[4]	result[6]	12.423	SLOW	3.576	FAST
A[4]	result[6]	12.401	SLOW	3.510	FAST
A[0]	result[3]	12.363	SLOW	3.823	FAST
X[0]	result[2]	12.301	SLOW	3.876	FAST
X[5]	result[7]	12.150	SLOW	3.343	FAST
X[7]	result[8]	12.023	SLOW	3.491	FAST
X[5]	result[6]	12.004	SLOW	3.705	FAST
A[5]	result[7]	11.975	SLOW	3.420	FAST

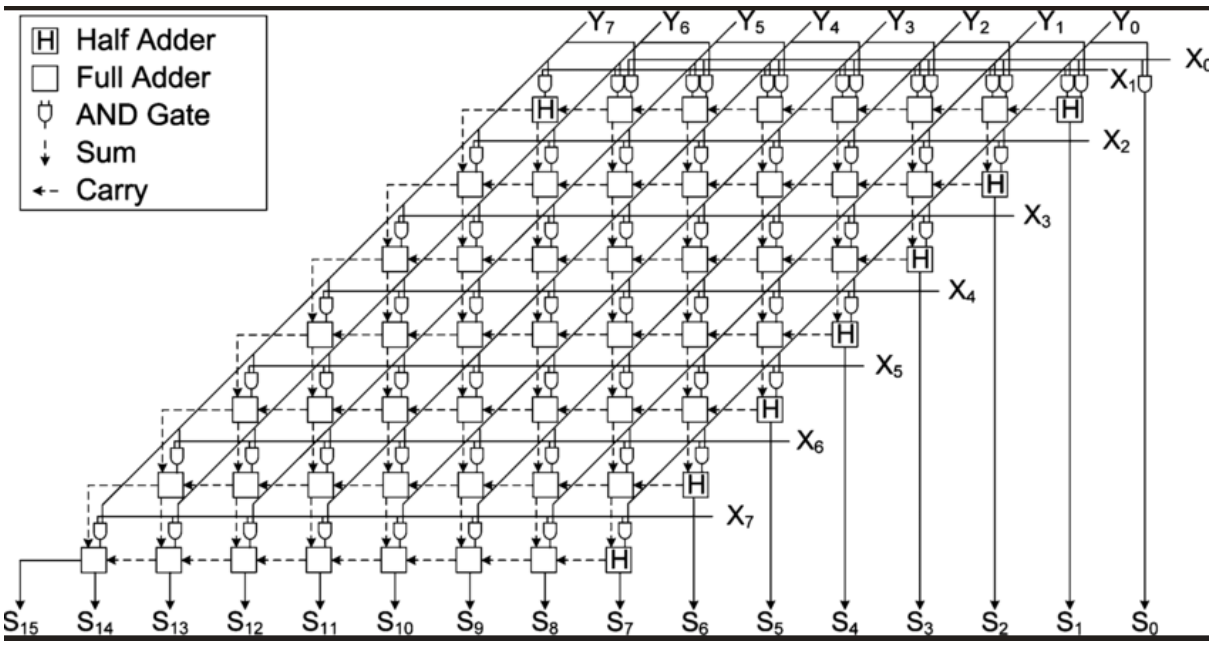
From Port	To Port	M a 1	Max Process Corner	Min Delay	Min Process Corner
A[1]	result[2]	11.943	SLOW	3.641	FAST
A[2]	result[3]	11.936	SLOW	3.859	FAST
A[5]	result[6]	11.829	SLOW	3.649	FAST
A[3]	result[3]	11.804	SLOW	4.084	FAST
X[6]	result[8]	11.651	SLOW	3.463	FAST
A[7]	result[8]	11.645	SLOW	3.243	FAST
X[2]	result[3]	11.617	SLOW	3.513	FAST
A[0]	result[2]	11.538	SLOW	3.841	FAST
X[0]	result[1]	11.198	SLOW	4.025	FAST
X[3]	result[5]	11.121	SLOW	3.261	FAST
X[4]	result[5]	11.024	SLOW	3.554	FAST
A[4]	result[5]	11.002	SLOW	3.372	FAST
X[1]	result[2]	10.997	SLOW	3.540	FAST
A[2]	result[2]	10.931	SLOW	3.797	FAST
A[1]	result[1]	10.840	SLOW	3.867	FAST
X[3]	result[4]	10.792	SLOW	3.532	FAST
X[2]	result[2]	10.792	SLOW	3.727	FAST
X[4]	result[4]	10.655	SLOW	3.722	FAST
A[4]	result[4]	10.634	SLOW	3.684	FAST
A[6]	result[7]	10.536	SLOW	3.332	FAST
A[0]	result[1]	10.510	SLOW	3.761	FAST
X[6]	result[7]	10.425	SLOW	3.333	FAST
X[1]	result[1]	10.293	SLOW	3.733	FAST
X[5]	result[5]	10.251	SLOW	3.567	FAST
X[7]	result[7]	10.236	SLOW	3.553	FAST
A[0]	result[0]	10.206	SLOW	3.616	FAST
A[5]	result[5]	10.075	SLOW	3.516	FAST

- About Baugh Wooley method:

The Baugh-Wooley algorithm is a multiplication algorithm used in digital signal processing applications. It was developed to perform binary multiplication operations quickly. Its operation can be explained with the following steps:

1. **Binary Representation:** Numbers are represented in binary format.
2. **Partial Products:** The multiplication operation is based on multiplying the individual bits of the numbers to obtain partial products.
3. **Matrix Multiplication:** The partial products are then organized into a matrix, and the final product is obtained using matrix multiplication.

The Baugh-Wooley algorithm efficiently reduces the number of partial products compared to traditional multiplication methods, making it suitable for fast and efficient multiply-accumulate (MAC) operations in digital filters. For 8x8 bit multiplication there is 7 stages of adder



PBSA for MAC

- Verilog Code:

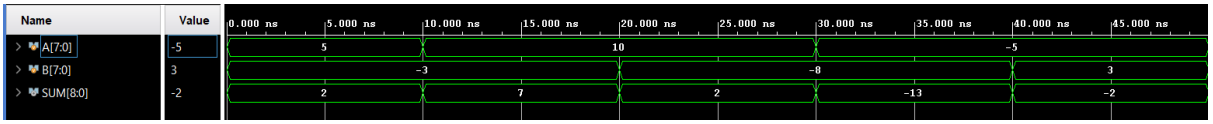
```
module PBSA #(parameter N=16)
  (input signed [N-1:0] A,
   input signed [N-1:0] B,
   output reg signed [N:0] result);

   always @* begin
      result = A + B;
   end

endmodule
```

- Testbench Code:

```
`timescale 1ns/1ns

module tb_PBSA;

  parameter N = 8;
  reg signed [N-1:0] A, B;
  wire signed [N:0] SUM;
  PBSA #(N) uut (
    .A(A),
```

```
    .B(B),
    .SUM(SUM)
  );

  reg clk = 0;
  always #5 clk = ~clk;

  initial begin
    A = 5;
    B = -3;
    #10 A = 10;
    #10 B = -8;
    #10 A = -5;
    #10 B = 3;
    #10 $finish;
  end

endmodule
```

- Simulation Result



MAC

- Verilog Code:

```
module MAC(
    input wire clk, reset,
    input signed [23:0] data, weight,
    output reg signed [19:0] result
);
    wire signed [15:0]  product [2:0];
    wire signed [15:0] sum [1:0];
    reg [1:0] count = 2'b00;

    MULTB mult_1(.A(data[7:0]), .B(weight[7:0]), .result(product[0]));
    MULTB mult_2(.A(data[15:8]), .B(weight[15:8]),
.result(product[1]));
    MULTB mult_3(.A(data[23:16]), .B(weight[23:16]),
.result(product[2]));

    PBSA s1 (.A(product[0]), .B(product[1]), .result(sum[0]));
```

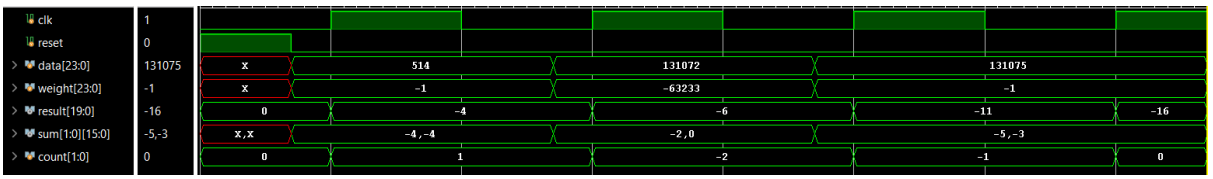
```
    PBSA s2 (.A(product[2]), .B(sum[0]), .result(sum[1]));

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            result <= 20'b0;
            count <= 2'b00;
        end else begin
            result <= result +  sum[1];
            count <= count + 1'b1;
        end
    end

endmodule
```

- Testbench Code(student id 040200203):

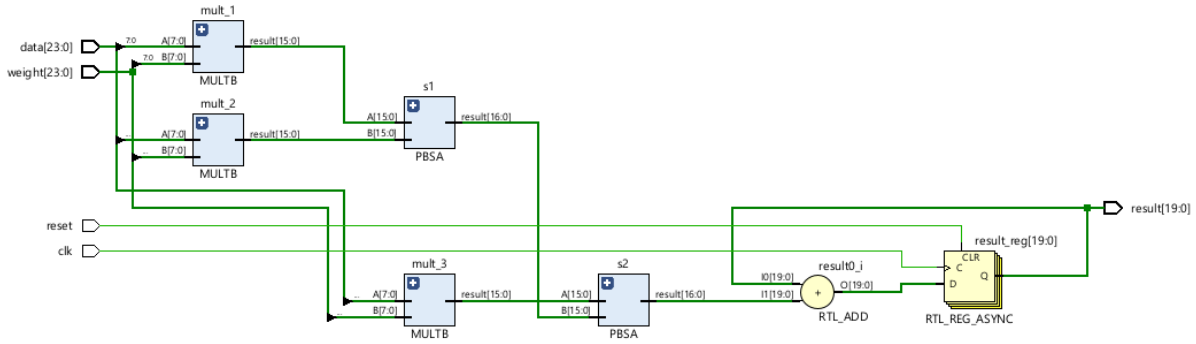
```
`timescale 1ns/1ns

module MAC_tb;
  parameter CLK_PERIOD = 20;
  parameter N = 24;
  reg clk = 0;
  reg reset = 0;
  reg signed [N-1:0] data, weight;
  wire signed [19:0] result;
  MAC uut (
    .clk(clk),
    .reset(reset),
    .data(data),
    .weight(weight),
    .result(result)
  );

  always #((CLK_PERIOD)/2) clk = ~clk;
  initial begin
    reset = 1;
    #7 reset = 0;

    data = 24'b0000_0000_0000_0010_0000_0010;
    weight = 24'b1111_1111_1111_1111_1111_1111;
    #20;
    data = 24'b0000_0010_0000_0000_0000_0000;
    weight = 24'b1111_1111_0000_1000_1111_1111;
```

```
    #20;
    data = 24'b0000_0010_0000_0000_0000_0011;
    weight = 24'b1111_1111_1111_1111_1111_1111;
    #20;
    #10 $finish;
  end

endmodule
```

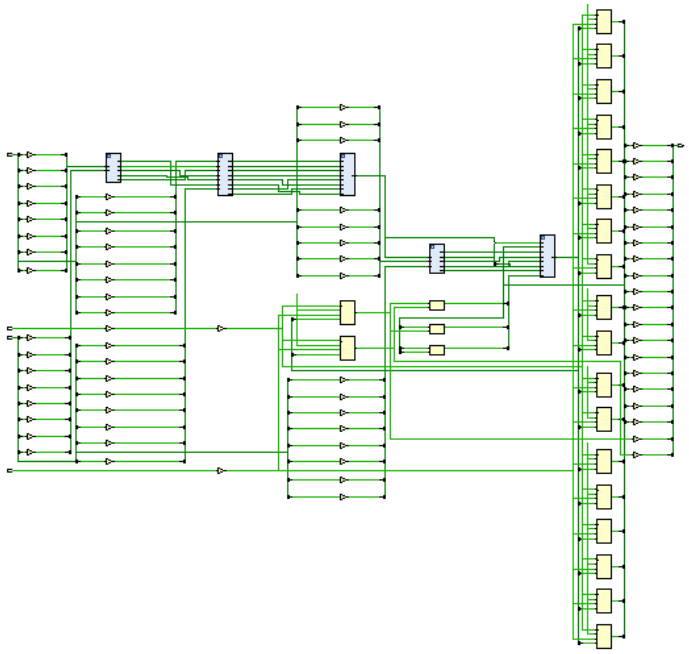
- Simulation Result



- RTL Schematic

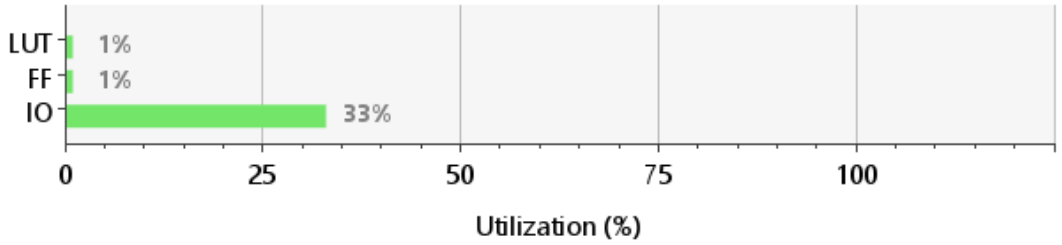


- Technology Schematic



- Utilization and timing Reports
  231 LUTs are used and 20 flip-flop is used. maximum delay is 10.5

Resource	Utilization	Available	Utilization %
LUT	232	63400	0.37
FF	20	126800	0.02
IO	70	210	33.33



Pad	Max Delay	Max Edge	Max Process Corner	Min Delay	Min Edge	Min Process Corner	Edge Skew
result[0]	10.383	Rise	SLOW	3.250	Rise	FAST	0.406
result[1]	10.401	Rise	SLOW	3.269	Rise	FAST	0.423
result[2]	10.500	Rise	SLOW	3.272	Rise	FAST	0.522
result[3]	10.496	Rise	SLOW	3.284	Rise	FAST	0.518
result[4]	10.368	Rise	SLOW	3.244	Rise	FAST	0.390
result[5]	10.220	Rise	SLOW	3.200	Rise	FAST	0.242
result[6]	10.391	Rise	SLOW	3.249	Rise	FAST	0.414
result[7]	10.351	Rise	SLOW	3.250	Rise	FAST	0.373
result[8]	10.207	Rise	SLOW	3.193	Rise	FAST	0.229
result[9]	10.200	Rise	SLOW	3.189	Rise	FAST	0.223
result[10]	10.314	Rise	SLOW	3.205	Rise	FAST	0.336
result[11]	10.271	Rise	SLOW	3.202	Rise	FAST	0.293
result[12]	10.153	Rise	SLOW	3.148	Rise	FAST	0.175
result[13]	9.978	Rise	SLOW	3.077	Rise	FAST	0.000
result[14]	10.127	Rise	SLOW	3.125	Rise	FAST	0.149
result[15]	10.253	Rise	SLOW	3.179	Rise	FAST	0.275
result[16]	10.275	Rise	SLOW	3.204	Rise	FAST	0.297
result[17]	10.157	Rise	SLOW	3.145	Rise	FAST	0.180
result[18]	10.146	Rise	SLOW	3.133	Rise	FAST	0.168
result[19]	10.298	Rise	SLOW	3.197	Rise	FAST	0.320
Worst Case Summary	10.500	Rise	SLOW	3.077	Rise	FAST	0.522