8x8 Dadda Multiplier (MAC)

by Aswin Ajayan

November 25, 2018

dependencies

compilation was done using icarus-iverilog, this can be installed in ubuntu 18 and above by

sudo apt install iverilog for other operating systems please goto
http://iverilog.wikia.com/wiki/Installation_Guide
to analyse the waveforms, gtkwave was used
can be installed in ubuntu by

sudo apt install gtkwave

A small java program, GenLoops.java was used to generate test data and certain reccuring assign statements. This doesn't pose any requirement on java runtime if you are using the txt files includes for the test benches.

Verilog code uses constructs in verilog like generate loops , and parameters, you can refer to IEEE standard for verilog for details

Top level module

An 8x8 dadda multiplier was designed and verified using verilog. the details of the top level module are as given below.

A,B - 8 bit multiplicants

M - previous result existing in Accumulator(16bits)

RES - 17 bit output.

submodules:

gen_part_products - generate the partial products of the multiplication, A*B as an 8x8 array.

processing block - takes the partial products,M and does the dadda reduction

adder16 - 16 bit Carry select adder ,

test data was randomly generated using a small program and verified the circuit using the test bench - top_level_tb(file:top_level_tb.v). Test data was loaded from files ain.txt,bin.txt,m.txt and res.txt

run.sh will compile all modules

```
1  //↩
       ////////////////////////////////////////////////////////////////////////////////////////////////////
2  //
3  //         MODULE: top_level
4  //  DESCRIPTION: top level module for dadda multiplier
5  //    IO SIGNALS: ---
6  //         AUTHOR: YOUR NAME (),
7  // ORGANIZATION:
8  //        VERSION: 1.0
9  //        CREATED: Sunday 11 November 2018 03:39:59  IST
10 //       REVISION: ---
11 //↩
       ////////////////////////////////////////////////////////////////////////////////////////////////////
12 `timescale 1ps/100fs
13 module top_level(
14         input [7:0] A,
15         input [7:0] B,
```

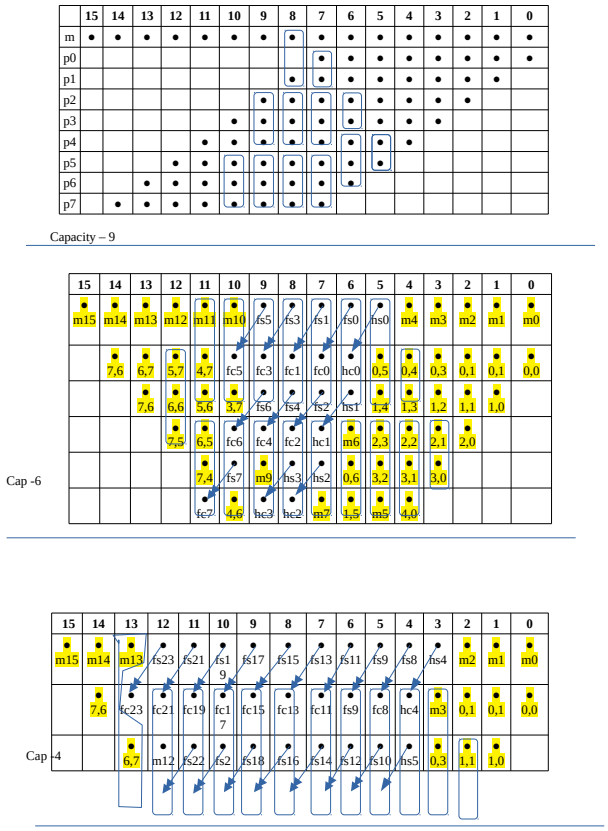

Figure 2.1:

```
16          input [15:0] M,
17          output [16:0] RES);
18
19          genvar i;
20          wire [7:0][7:0] P;
21          wire [1:0][15:0] PRE;
22          gen_part_products U1(A,B,P);
23
24          processing_block U2(P,M,PRE);
25
26          adder16 U3(PRE[1],PRE[0],1'b0,RES);
27  endmodule
```

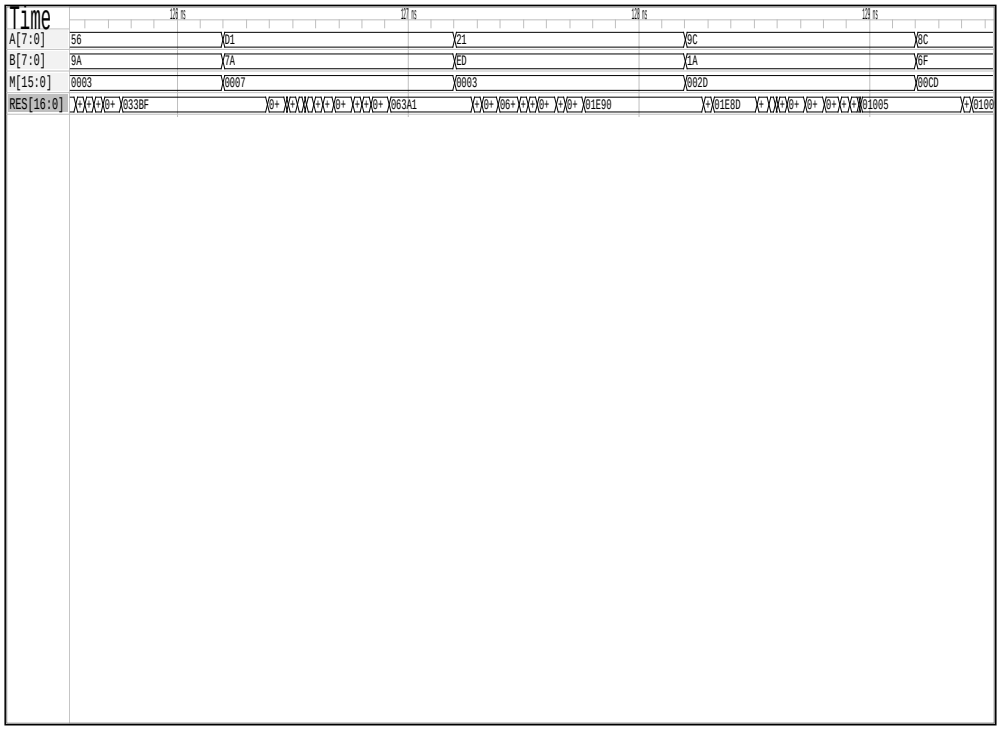
Time
A[7:0]
B[7:0]
M[15:0]
RES[16:0]
56
D1
21
9C
8C
9A
7A
ED
1A
6F
0003
0007
0003
002D
00CD
033BF
063A1
01E90
01E8D
01005

Figure 2.2:

Sheet1

input A	input B	**B**	Result
A = d5	B = f6	M = 005b	RES = 0cd09
A = a2	B = b0	M = 0000	RES = 06f60
A = 85	B = ef	M = 008f	RES = 07cba
A = aa	B = a5	M = 0070	RES = 06e02
A = 74	B = 2c	M = 001f	RES = 0140f
A = 7d	B = fa	M = 0055	RES = 07a67
A = 58	B = 4d	M = 0014	RES = 01a8c
A = 7c	B = 22	M = 00a1	RES = 01119
A = 30	B = 6c	M = 002f	RES = 0146f
A = 56	B = ac	M = 0059	RES = 03a21
A = 86	B = 52	M = 002d	RES = 02b19
A = 3d	B = 81	M = 00d3	RES = 01f90
A = 19	B = 91	M = 0002	RES = 00e2b
A = 11	B = 19	M = 0080	RES = 00229
A = ba	B = 95	M = 004f	RES = 06c91
A = fb	B = 73	M = 005f	RES = 07120
A = 73	B = 27	M = 0056	RES = 011db

2.1 code

```
1  //↩
       ////////////////////////////////////////////////////////////////////////////////
2  //
3  //         MODULE: top_level_tb
4  //  DESCRIPTION: tset bench for 8*8 dadda
5  //   IO SIGNALS: ---
6  //         AUTHOR: YOUR NAME (),
7  // ORGANIZATION:
8  //        VERSION: 1.0
9  //        CREATED: Sunday 11 November 2018 04:05:03  IST
10 //       REVISION: ---
11 //↩
       ////////////////////////////////////////////////////////////////////////////////
12
13 `timescale 1ps/100fs
14 module top_level_tb(); //testbench doesnt have any inputs or↩
       outputs
15         reg [7:0] A;     //inputs are takens as reqisters ( ↩
               they need to hold the value)
16         reg [7:0] B;
17         reg[15:0] M;
18         wire [16:0] RES; //outputs are takens as wires in tb↩
                .
19         reg[7:0] ain_array[0:250];
20         reg[7:0] bin_array[0:250];
21         reg[16:0] res_array[0:250];
22         reg[15:0] M_array[0:250];
23         top_level dut(.*); //since all the inputs to the dut↩
               are the wires of same name
24         integer i;
25         initial begin
26                 $dumpfile("top_level_tb.vcd");
27                 $dumpvars(0,top_level_tb);                ↩
                                  //first argument is the level of↩
                       debugging
28                                                                                  //↩
                                                                                      level↩
                                                                                       ↩
                                                                                      0↩
                                                                                       ↩
                                                                                      will↩
                                                                                       ↩
                                                                                      log↩
                                                                                       ↩
```

```
all↩
 ↩
the↩
 ↩
variab
 ↩
even↩
 ↩
in↩
```

```
29
30                  $readmemb("ain.txt",ain_array);
31                  $readmemb("bin.txt",bin_array);
32                  $readmemb("res.txt",res_array);
33                  $readmemb("m.txt",M_array);
34                  //sub modules
35                  //whereas level 1 will log only the ones in ↩
                        the top module
36                  A = 8'h0;
37                  B = 8'h0;
38                  M = 16'h0;
39                  #2000;
40                  A = 8'hff;
41                  B = 8'haa;
42                  #2000;
43                  B = 8'hff;
44                  #200;
45                  #1000;
46
47                  M = 16'h02;
48                  $display("starting....");
49
50                  for(i = 0;i<250;i = i+1)begin
51                          A = ain_array[i];
52                          B = bin_array[i];
53                          M = M_array[i];
54                          #1000;
55                          $display("A = %h, B = %h, M = %h  , ↩
                                RES = %h",A,B,M,RES);
56                          if(RES != res_array[i])
57                                  $display("error");
58                          else
59                                  $display("test passed");
60
61                  end
62
63          end
64  endmodule
```

```
`timescale 1ps/100fs
module gen_part_products(
        input [7:0] A,
        input [7:0] B,
        output[7:0][7:0] P);     //portlist can be 2D array in verilog
        genvar i;
        generate
                for(i = 0; i < 8; i = i +1) begin:part_product
                        assign P[i][0] = A[0] & B[i] ;
                        assign P[i][1] = A[1] & B[i] ;
                        assign P[i][2] = A[2] & B[i] ;
                        assign P[i][3] = A[3] & B[i] ;
                        assign P[i][4] = A[4] & B[i] ;
                        assign P[i][5] = A[5] & B[i] ;
                        assign P[i][6] = A[6] & B[i] ;
                        assign P[i][7] = A[7] & B[i] ;
                end
        endgenerate
endmodule


////////////////////////////////////////////////////////////////////////////////////////////////////
//
//       MODULE: adder16
//  DESCRIPTION: 16 bit adder (carry_out select )
//    IO SIGNALS: A,B,O,Cin
//       AUTHOR: YOUR NAME (),
// ORGANIZATION:
//       VERSION: 1.0
//       CREATED: Monday 12 November 2018 12:30:50  IST
//      REVISION: ---
////////////////////////////////////////////////////////////////////////////////////////////////////

`timescale 1ps/100fs
module CSA_block #(parameter width = 4)(
        input [width -1:0]A,
        input [width -1:0]B,
        output[1:0][width -1:0]SUM,
        output[1:0]c_out);

        genvar i;
        wire [1:0][width - 1:0]carry_out;
        full_adder fa0(A[0],B[0],1'b0,SUM[0][0],carry_out[0][0]);
        full_adder fa1(A[0],B[0],1'b1,SUM[1][0],carry_out
```

```
            [1][0]);
23          generate
24                  for (i = 1; i < width -1 ; i = i + 1) begin:↩
                        gen_CSA_block
25                          full_adder fa_carry_out_zero(A[i],B[↩
                                i],carry_out[0][i-1],SUM[0][i],↩
                                carry_out[0][i]);
26                          full_adder fa_carry_out_one(A[i],B[i↩
                                ],carry_out[1][i-1],SUM[1][i],↩
                                carry_out[1][i]);
27                  end
28          endgenerate
29
30          full_adder fa_carry_out_zero_last(A[width-1],B[width↩
                -1],carry_out[0][width-2],SUM[0][width-1],c_out↩
                [0]);
31          full_adder fa_carry_out_one_last(A[width-1],B[width↩
                -1],carry_out[1][width-2],SUM[1][width-1],c_out↩
                [1]);
32  endmodule
33
34  module selector #(parameter width = 4)(
35          input [1:0][width -1:0]SUM,
36          input [1:0]c_out,
37          input c_in,
38          output [width -1:0]SUM_OUT,
39          output CARRY_OUT);
40
41
42          genvar i;
43
44          assign SUM_OUT = c_in?SUM[1]:SUM[0];
45          assign CARRY_OUT = c_in?c_out[1]:c_out[0];
46  endmodule
47
48  module adder16(
49          input [15:0]A,
50          input [15:0]B,
51          input c_in,
52          output [16:0]SUM);
53
54          wire [2:0][1:0] block_carry_out;
55          wire [1:0][3:0] sum_block1;
56          wire [1:0][4:0] sum_block2;
57          wire [1:0][5:0] sum_block3;
58          wire [3:0]cout_inter;
59          wire cout_0;
60          //carry select adder is implemented in stages of 1 ,↩
                4 , 5 , 6
61          full_adder fa0_in16(A[0],B[0],c_in,SUM[0],cout_0);
```

```
62          CSA_block #(.width(4)) block_1(A[4:1],B[4:1],↩
                sum_block1,block_carry_out[0]);
63          CSA_block #(.width(5)) block_2(A[9:5],B[9:5],↩
                sum_block2,block_carry_out[1]);
64          CSA_block #(.width(6)) block_3(A[15:10],B[15:10],↩
                sum_block3,block_carry_out[2]);
65
66
67          selector #(.width(4)) sel0(sum_block1,↩
                block_carry_out[0],cout_0,SUM[4:1],cout_inter↩
                [0]);
68          selector #(.width(5)) sel1(sum_block2,↩
                block_carry_out[1],cout_inter[0],SUM[9:5],↩
                cout_inter[1]);
69          selector #(.width(6)) sel2(sum_block3,↩
                block_carry_out[2],cout_inter[1],SUM[15:10],SUM↩
                [16]);
70
71
72  endmodule
73
74
75  //module CSA_block_tb(); //testbench doesnt have any inputs ↩
        or outputs
76  //      reg [7:0] A;    //inputs are takens as reqisters ( ↩
        they need to hold the value)
77  //      reg [7:0] B;
78  //      wire [1:0][7:0]SUM; //outputs are takens as wires in↩
         tb .
79  //      wire [1:0]c_out;
80  //      CSA_block #(.width(8)) dut(.*); //since all the ↩
        inputs to the dut are the wires of same name
81  //      initial begin
82  //              $dumpfile("CSA_block.vcd");
83  //              $dumpvars(0,CSA_block_tb);                ↩
            //first argument is the level of debugging
84  //                                                        ↩
                    //level 0 will log all the variable even↩
         in
85  //              //sub modules
86  //              //whereas level 1 will log only the ones in ↩
        the top module
87  //              A = 8'h0;
88  //              B = 8'hff;
89  //              #2000;
90  //              A = 8'hff;
91  //              B = 8'haa;
92  //              #2000;
93  //              B = 8'hff;
94  //              #200;
```

```
95 //                    #1000;
96 //        end
97 //endmodule
```