8-BIT DADDA MULTIPLIER

DESCRIPTION:

“dvsd_8216m3” is a 8 bit multiplier which uses Dadda-tree algorithm to multiply two 8-bit numbers to produce 16-bit number.

Input A [MSB:LSB]: a7 a6 a5 a4 a3 a2 a1 a0

Input B [MSB:LSB] : b7 b6 b5 b4 b3 b2 b1 b0

Output M [MSB:LSB] : m15 m14 m13 m12 m11 m10 m9 m8 m7 m6 m5 m4 m3 m2 m1 m0

MODULES:

1. dvsd_8216m3
2. Half Adder
3. 3:2 Compressor
4. CMOS AND Gate
5. CMOS XOR-XNOR Gate
6. CMOS 2-input Multiplexer
7. CMOS Inverter

Code

```
`timescale 1ns/1ps

module dvsd_8216m3(
  input   a0, a1, a2, a3, a4, a5, a6, a7,
  input   b0, b1, b2, b3, b4, b5, b6, b7,
  output  m0, m1, m2, m3, m4, m5, m6, m7,
  output  m8, m9, m10, m11, m12, m13, m14, m15
);

  wire w00,w10,w20,w30,w40,w50,w60,w70;
  wire w01,w11,w21,w31,w41,w51,w61,w71;
  wire w02,w12,w22,w32,w42,w52,w62,w72;
  wire w03,w13,w23,w33,w43,w53,w63,w73;
  wire w04,w14,w24,w34,w44,w54,w64,w74;
  wire w05,w15,w25,w35,w45,w55,w65,w75;
  wire w06,w16,w26,w36,w46,w56,w66,w76;
  wire w07,w17,w27,w37,w47,w57,w67,w77;

// ============================================================
// Product Generation
// ============================================================

  and1b AND0(a0, b0, w00);
  and1b AND1(a1, b0, w10);
  and1b AND2(a2, b0, w20);
  and1b AND3(a3, b0, w30);
  and1b AND4(a4, b0, w40);
  and1b AND5(a5, b0, w50);
  and1b AND6(a6, b0, w60);
  and1b AND7(a7, b0, w70);

  and1b AND8(a0, b1, w01);
  and1b AND9(a1, b1, w11);
  and1b AND10(a2, b1, w21);
  and1b AND11(a3, b1, w31);
  and1b AND12(a4, b1, w41);
  and1b AND13(a5, b1, w51);
  and1b AND14(a6, b1, w61);
  and1b AND15(a7, b1, w71);

  and1b AND16(a0, b2, w02);
  and1b AND17(a1, b2, w12);
  and1b AND18(a2, b2, w22);
  and1b AND19(a3, b2, w32);
  and1b AND20(a4, b2, w42);
  and1b AND21(a5, b2, w52);
  and1b AND22(a6, b2, w62);
  and1b AND23(a7, b2, w72);
```

```
  and1b AND24(a0, b3, w03);
  and1b AND25(a1, b3, w13);
  and1b AND26(a2, b3, w23);
  and1b AND27(a3, b3, w33);
  and1b AND28(a4, b3, w43);
  and1b AND29(a5, b3, w53);
  and1b AND30(a6, b3, w63);
  and1b AND31(a7, b3, w73);

  and1b AND32(a0, b4, w04);
  and1b AND33(a1, b4, w14);
  and1b AND34(a2, b4, w24);
  and1b AND35(a3, b4, w34);
  and1b AND36(a4, b4, w44);
  and1b AND37(a5, b4, w54);
  and1b AND38(a6, b4, w64);
  and1b AND39(a7, b4, w74);

  and1b AND40(a0, b5, w05);
  and1b AND41(a1, b5, w15);
  and1b AND42(a2, b5, w25);
  and1b AND43(a3, b5, w35);
  and1b AND44(a4, b5, w45);
  and1b AND45(a5, b5, w55);
  and1b AND46(a6, b5, w65);
  and1b AND47(a7, b5, w75);

  and1b AND48(a0, b6, w06);
  and1b AND49(a1, b6, w16);
  and1b AND50(a2, b6, w26);
  and1b AND51(a3, b6, w36);
  and1b AND52(a4, b6, w46);
  and1b AND53(a5, b6, w56);
  and1b AND54(a6, b6, w66);
  and1b AND55(a7, b6, w76);

  and1b AND56(a0, b7, w07);
  and1b AND57(a1, b7, w17);
  and1b AND58(a2, b7, w27);
  and1b AND59(a3, b7, w37);
  and1b AND60(a4, b7, w47);
  and1b AND61(a5, b7, w57);
  and1b AND62(a6, b7, w67);
  and1b AND63(a7, b7, w77);

//============================================================
// Stage 1D
```

```
//============================================================

  wire us1,us2,us3,us4,us5,us6;
  wire uc1,uc2,uc3,uc4,uc5,uc6;

  cmos_halfadder HA1(w60,w51,us1,uc1);
  cmos_halfadder HA2(w43,w34,us2,uc2);
  compressor3to2 CMP1(w70,w61,w52,us3,uc3);
  cmos_halfadder HA3(w44,w35,us4,uc4);
  compressor3to2 CMP2(w71,w62,w53,us5,uc5);
  compressor3to2 CMP3(w72,w63,w54,us6,uc6);

// ============================================================
// Stage 2D
// ============================================================

  wire vs0,vs1,vs2,vs3,vs4,vs5,vs6,vs7,vs8,vs9,vs10,vs11,vs12,vs13;
  wire vc0,vc1,vc2,vc3,vc4,vc5,vc6,vc7,vc8,vc9,vc10,vc11,vc12,vc13;

  cmos_halfadder HA4(w40,w31,vs0,vc0);
  compressor3to2 CMP4(w50,w41,w32,vs1,vc1);
  cmos_halfadder HA5(w23,w14,vs2,vc2);
  compressor3to2 CMP5(us1,w42,w33,vs3,vc3);
  compressor3to2 CMP6(w24,w15,w06,vs4,vc4);
  compressor3to2 CMP7(uc1,us2,us3,vs5,vc5);
  compressor3to2 CMP8(w25,w16,w07,vs6,vc6);
  compressor3to2 CMP9(uc3,uc2,us5,vs7,vc7);
  compressor3to2 CMP10(us4,w26,w17,vs8,vc8);
  compressor3to2 CMP11(uc5,uc4,us6,vs9,vc9);
  compressor3to2 CMP12(w45,w36,w27,vs10,vc10);
  compressor3to2 CMP13(uc6,w73,w64,vs11,vc11);
  compressor3to2 CMP14(w55,w46,w37,vs12,vc12);
  compressor3to2 CMP15(w74,w65,w56,vs13,vc13);

// ============================================================
// Stage 3D
// ============================================================

  wire ts0,ts1,ts2,ts3,ts4,ts5,ts6,ts7,ts8,ts9;
  wire tc0,tc1,tc2,tc3,tc4,tc5,tc6,tc7,tc8,tc9;

  cmos_halfadder HA6(w30,w21,ts0,tc0);
  compressor3to2 CMP16(vs0,w22,vs13,ts1,tc1);
  compressor3to2 CMP17(vc0,vs1,vs2,ts2,tc2);
  compressor3to2 CMP18(vc1,vc2,vs3,ts3,tc3);
  compressor3to2 CMP19(vc3,vc4,vs5,ts4,tc4);
  compressor3to2 CMP20(vc5,vc6,vs7,ts5,tc5);
  compressor3to2 CMP21(vc7,vc8,vs9,ts6,tc6);
```

```
  compressor3to2 CMP22(vc9,vc10,vs11,ts7,tc7);
  compressor3to2 CMP23(vc11,vc12,vs13,ts8,tc8);
  compressor3to2 CMP24(vc13,w75,w66,ts9,tc9);

// ============================================================
// Stage 3D
// ============================================================

  wire rs0,rs1,rs2,rs3,rs4,rs5,rs6,rs7,rs8,rs9,rs10,rs11;
  wire rc0,rc1,rc2,rc3,rc4,rc5,rc6,rc7,rc8,rc9,rc10,rc11;

  cmos_halfadder HA7(w20,w11,rs0,rc0);
  compressor3to2 CMP25(ts0,w12,w03,rs1,rc1);
  compressor3to2 CMP26(tc0,ts1,w04,rs2,rc2);
  compressor3to2 CMP27(tc1,ts2,w05,rs3,rc3);
  compressor3to2 CMP28(tc2,ts3,vs4,rs4,rc4);
  compressor3to2 CMP29(tc3,ts4,vs6,rs5,rc5);
  compressor3to2 CMP30(tc4,ts5,vs8,rs6,rc6);
  compressor3to2 CMP31(tc5,ts6,vs10,rs7,rc7);
  compressor3to2 CMP32(tc6,ts7,vs12,rs8,rc8);
  compressor3to2 CMP33(tc7,ts8,w47,rs9,rc9);
  compressor3to2 CMP34(tc8,ts9,w57,rs10,rc10);
  compressor3to2 CMP35(tc9,w76,w67,rs11,rc11);

// ============================================================
// Final Addition
// ============================================================

  wire mc1,mc2,mc3,mc4,mc5,mc6,mc7,mc8,mc9,mc10,mc11,mc12,mc13,mc14;
  wire ms1,ms2,ms3,ms4,ms5,ms6,ms7,ms8,ms9,ms10,ms11,ms12,ms13,ms14;

  cmos_halfadder HA8(w10,w01,ms1,mc1);
  compressor3to2 CMP36(rs0,w02,mc1,ms2,mc2);
  compressor3to2 CMP37(rc0,rs1,mc2,ms3,mc3);
  compressor3to2 CMP38(rc1,rs2,mc3,ms4,mc4);
  compressor3to2 CMP39(rc2,rs3,mc4,ms5,mc5);
  compressor3to2 CMP40(rc3,rs4,mc5,ms6,mc6);
  compressor3to2 CMP41(rc4,rs5,mc6,ms7,mc7);
  compressor3to2 CMP42(rc5,rs6,mc7,ms8,mc8);
  compressor3to2 CMP43(rc6,rs7,mc8,ms9,mc9);
  compressor3to2 CMP44(rc7,rs8,mc9,ms10,mc10);
  compressor3to2 CMP45(rc8,rs9,mc10,ms11,mc11);
  compressor3to2 CMP46(rc9,rs10,mc11,ms12,mc12);
  compressor3to2 CMP47(rc10,rs11,mc12,ms13,mc13);
  compressor3to2 CMP48(rc11,w77,mc13,ms14,mc14);

// ============================================================
// Outputs
```

```
// ============================================================

  assign m0   = w00;
  assign m1   = ms1;
  assign m2   = ms2;
  assign m3   = ms3;
  assign m4   = ms4;
  assign m5   = ms5;
  assign m6   = ms6;
  assign m7   = ms6;
  assign m8   = ms7;
  assign m9   = ms8;
  assign m10  = ms9;
  assign m11  = ms10;
  assign m12  = ms11;
  assign m13  = ms12;
  assign m14  = ms13;
  assign m15  = mc14;
endmodule

// ============================================================
//
// Half Adder
//
// ============================================================

module cmos_halfadder(
  input a,
  input b,
  output sum,
  output carry
);
  assign sum = a + b;
  assign carry = a & b;

endmodule

// ============================================================
//
// 3:2 Compressor
//
// ============================================================

module compressor3to2(
  input a0,
  input a1,
  input a2,
  output sout,
```

```
  output cout
);

  assign cout = (a0 * a1) + (a2 * (a0 ^ a1));
  assign sout = ((a0 ^ a1) * ~a2) + (~(a0 ^ a1) * a2);

endmodule

// ============================================================
//
// CMOS AND Gate
//
// ============================================================

module and1b(
  input a1,
  input a2,
  output y
);
  assign y = a1 & a2;

endmodule

// ============================================================
//
// CMOS 2-input Multiplexer
//
// ============================================================

module mux1b(
  input i0,   // input 1
  input i1,   // input 2
  input s,    // input select
  output y    // output
);
  assign y = (~s * i0) + (s * i1);
endmodule

// ============================================================
//
// CMOS XOR-XNOR Gate
//
// ============================================================

module xor_xnor1b(
  input a,
  input b,
  output xor_o,
```

```
  output xnor_o
);
  assign xor_o = a ^ b;
  assign xnor_o = ~(a ^ b);
endmodule

// ============================================================
//
// CMOS 1-bit Inverter
//
// ============================================================

module inverter1b(
  input in,
  output out
);
  assign out = ~in;
endmodule
```

Testbench for dvsd_8216m3

```
module dvsd_8216m3_tb();
  reg a0, a1, a2, a3, a4, a5, a6, a7;
  reg b0, b1, b2, b3, b4, b5, b6, b7;
  wire m0, m1, m2, m3, m4, m5, m6, m7;
  wire m8, m9, m10, m11, m12, m13, m14, m15;

  reg clock;
  integer i;

  always begin
    #5 clock = !clock;
  end

  initial begin
    clock = 0;
    a0 = 0;
    a1 = 0;
    a2 = 0;
    a3 = 0;
    a4 = 0;
    a5 = 0;
    a6 = 0;
    a7 = 0;
    b0 = 0;
    b1 = 0;
    b2 = 0;
    b3 = 0;
    b4 = 0;
    b5 = 0;
    b6 = 0;
    b7 = 0;
  end

  integer ran1, ran2;

  initial begin
    $dumpfile("dvsd_8216m3_tb.vcd");
    $dumpvars(0, dvsd_8216m3_tb);
    for(i=0; i<10; i=i+1) begin
      @(posedge clock);
      ran1 = {$random} % 256;
      {a0, a1, a2, a3, a4, a5, a6, a7} = ran1;
      ran2 = {$random} % 256;
      {b0, b1, b2, b3, b4, b5, b6, b7} = ran2;
    end
  end

  initial begin
```

```
    $timeformat(-9,4," ns",9);
    $display("time\t\ta\t\tb\t\tm");
    $monitor(
      "%0t",$realtime,"\t",a7,a6,a5,a4," ",a3,a2,a1,a0,"\t",
      b7,b6,b5,b4," ",b3,b2,b1,b0,
      "\t",m15,m14,m13,m12," ",m11,m10,m9,m8," ",m7,m6,m5,m4," ",m3,m2,m1
    );
    #50 $finish;
  end

  dvsd_8216m3 MULTIPLIER(
  .a0(a0),
  .a1(a1),
  .a2(a2),
  .a3(a3),
  .a4(a4),
  .a5(a5),
  .a6(a6),
  .a7(a7),
  .b0(b0),
  .b1(b1),
  .b2(b2),
  .b3(b3),
  .b4(b4),
  .b5(b5),
  .b6(b6),
  .b7(b7),
  .m0(m0),
  .m1(m1),
  .m2(m2),
  .m3(m3),
  .m4(m4),
  .m5(m5),
  .m6(m6),
  .m7(m7),
  .m8(m8),
  .m9(m9),
  .m10(m10),
  .m11(m11),
  .m12(m12),
  .m13(m13),
  .m14(m14),
  .m15(m15)
 );
endmodule
```