REFERENCES

1. Michael J. Flynn, and Stuart F. Oberman, (2001). *Advanced Computer Arithmetic Design*, John Wiley & Sons Inc., USA.

2. Samir Palnitkar, (2003). *Verilog HDL: A Guide to Digital Design and Synthesis*, 2nd Edition, Prentice Hall, USA.

3. Wai-Kai Chen, (2003). *Logic Design*, CRC Press, USA.

4. Keshab K. Parhi, (1999). *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley & Sons Inc., USA.

5. Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic, (2004). *Digital Integrated Circuits: A Design Perspective*, 2nd Edition, Prentice Hall, USA.

6. John P. Uyemura, (2002). *Introduction to VLSI Circuits and Systems*, John Wiley & Sons Inc., USA.

7. Michael John Sebastian Smith, (2003). *Application-Specific Integrated Circuits*, Addison Wesley, USA.

8. Wayne Wolf, (2004). *Modern VLSI Design: Systems on Silicon*, 2nd Edition, Prentice Hall, USA.

9. M. Michael Vai, (2002). *VLSI Design*, CRC Press, USA.

10. Jacob Millman, and Arvin Grabel, (2003). *Microelectronics*, 2nd Edition, McGraw Hill, USA.

11. Pascal C. H. Meier, Rob A. Rutenbar, and L. Richard Carley. *Exploring Multiplier Architecture and Layout for Low Power*. Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh.

12. Pedram Mokrian, G. Michael Howard, Graham Jullien, and Majid Ahmadi. *On the use of 4:2 Compressors for Partial Product Reduction*. University of Windsor and University of Calgary.

13. Moises E. Robinson and Earl Swartzlander, Jr.. *A Reduction Scheme to Optimize the Wallace Multiplier*. Department of Electrical and Computer Engineering, University of Texas, USA.

14. Bryan W. Stiles and Earl Swartzlander, Jr.. *Pipelined Parallel Multiplier Implementation*. Department of Electrical and Computer Engineering, University of Texas, USA.

15. Lakshmanan, Masuri Othman and Mohamad Alauddin Mohd. Ali, (2002). *High Performance Parallel Multiplier using Wallace-Booth Algorithm*. Signal Processing Research Group, Universiti Kebangsaan Malaysia (UKM), Malaysia.

16. Oscar Gustafsson,Andrew G. Dempster, and Lars Wanhammar. *Multiplier Blocks using Carry-save Adders*. Department of Electrical Engineering, Linkoping University, Sweden and University of Westminster, UK.

17. David Harris, (2000). *Structural Design with Verilog*. Harvey Mudd College.

18. Synopsys Inc., (2005). *Coding Guidelines for Datapath Synthesis*. USA.

19. Wikipedia Encylopedia, (2006), Multiplier, http://en.wikipedia.org/wiki/Multiplier, 15 December 2006.

20. Wikipedia Encylopedia, (2007), Flip-flop, http://en.wikipedia.org/wiki/Flip-flop_(electronics) , 7 March 2007.

21. Altera, (2006), Quartus II Introduction Using Verilog Design, http://www.altera.com/education/univ/materials/manual/labs/tut_quartus_intro_verilog.pdf, 28 December 2006.

APPENDICES

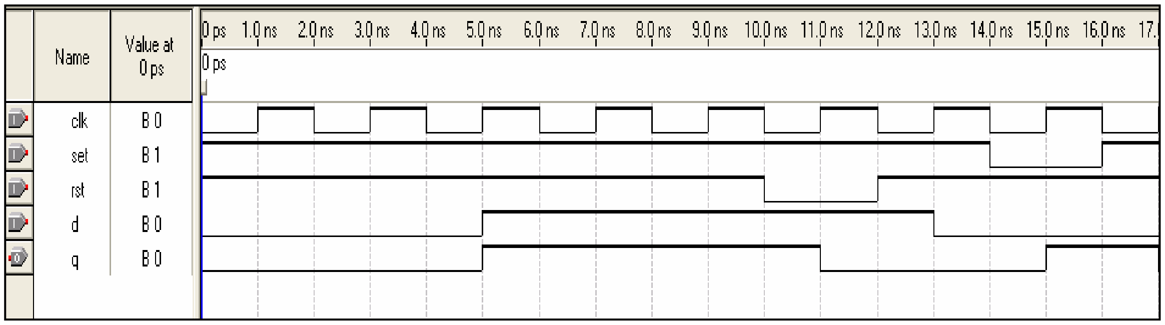
Appendix A

Source Code, Simulation Result and RTL View for a 1-bit D Flip-flop with Synchronous Set and Reset

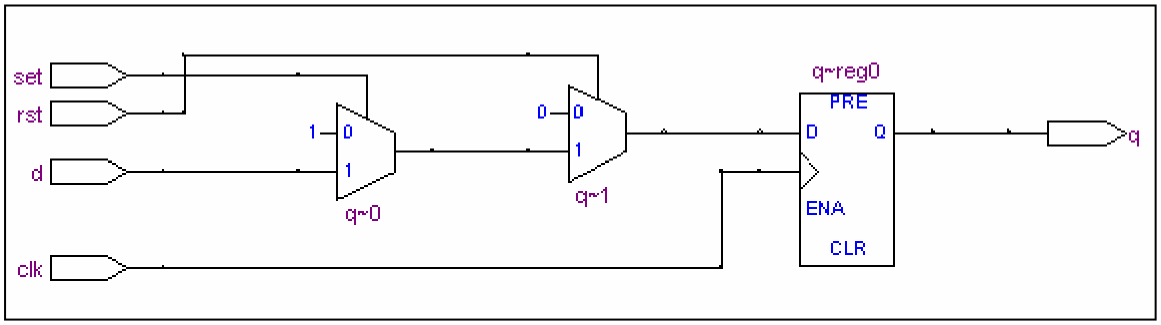
Source Code for a 1-bit D flip-flop with synchronous set and reset:

```
//Define a 1-bit D flip-flop with synchronous set and reset
module d_ff (q, clk, d, set, rst);

//I/O port declarations
output q;
input clk, d, set, rst;
reg q;

always @ (posedge clk)
        begin
                if (rst = = 0) q = 0;
                else if(set = = 0) q = 1;
                else q = d;
        end

endmodule
```

Simulation Result for a 1-bit D flip-flop with synchronous set and reset:



RTL View for a 1-bit D flip-flop with synchronous set and reset:



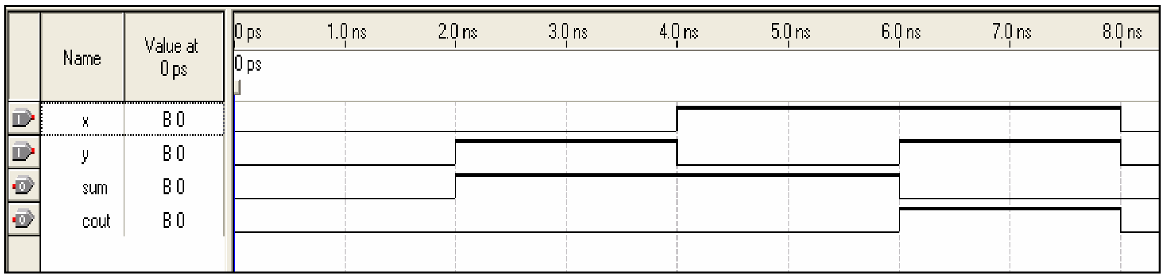
Appendix B

Source Code, Simulation Result and RTL View for a 1-bit Conventional Half Adder

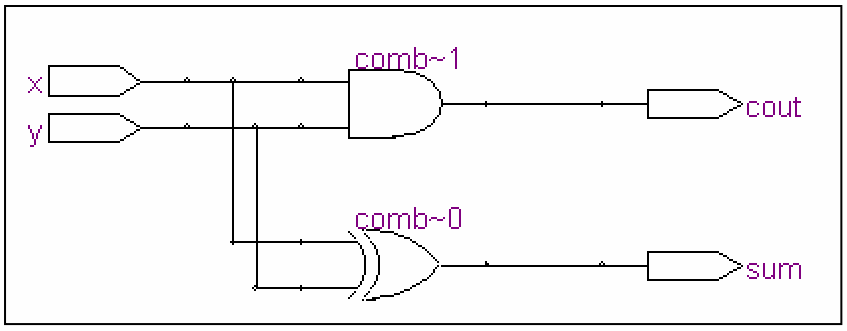
Source Code for a 1-bit Conventional Half Adder:

```
//Define a 1-bit half adder
module halfadd (sum, cout, x, y);

//I/O port declarations
output sum, cout;
input x, y;

//Instantiate logic gate primitives
xor (sum, x, y);
and (cout, x, y);

endmodule
```

Simulation Result for a 1-bit Conventional Half Adder:



RTL View for a 1-bit Conventional Half Adder:



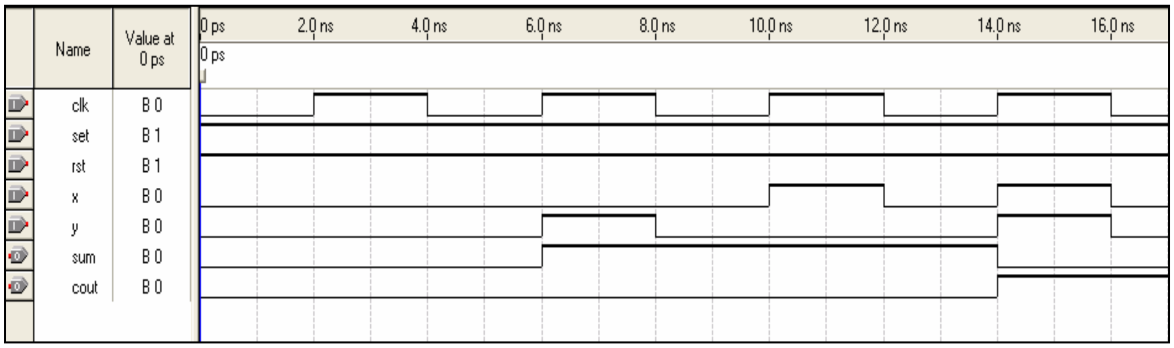
Appendix C

Source Code, Simulation Result and RTL View for a 1-bit Pipeline Half Adder

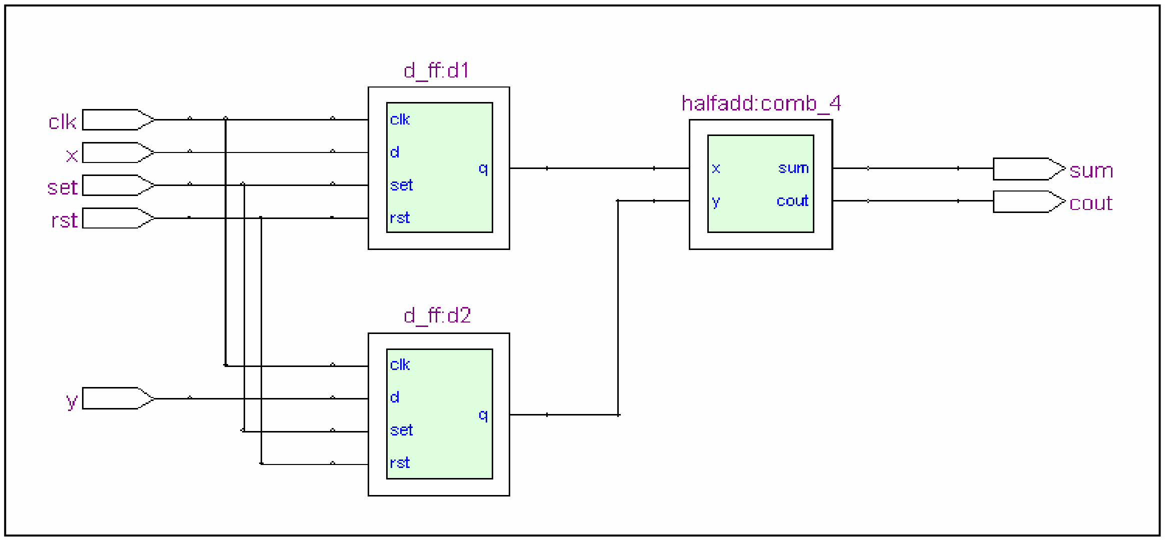
Source Code for a 1-bit Pipeline Half Adder:

```
//Define a pipeline 1-bit half adder
module halfadd_pl (sum, cout, clk, set, rst, x, y);

//I/O port declarations
output sum, cout;
input clk, set, rst, x, y;

//Internal nets
wire x1, y1;

//Instantiate logic gate primitives
d_ff d1 (x1, clk, x, set, rst);
d_ff d2 (y1, clk, y, set, rst);

halfadd (sum, cout, x1, y1);

endmodule
```

Simulation Result for a 1-bit Pipeline Half Adder:



RTL View for a 1-bit Pipeline Half Adder:



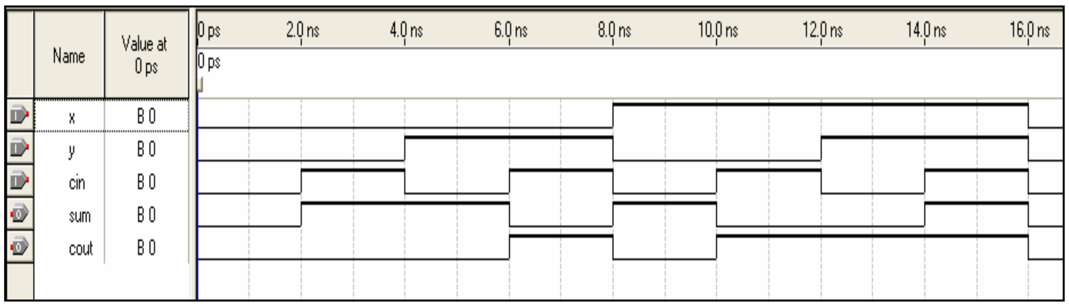
Appendix D

Source Code, Simulation Result and RTL View for a 1-bit Conventional Full Adder

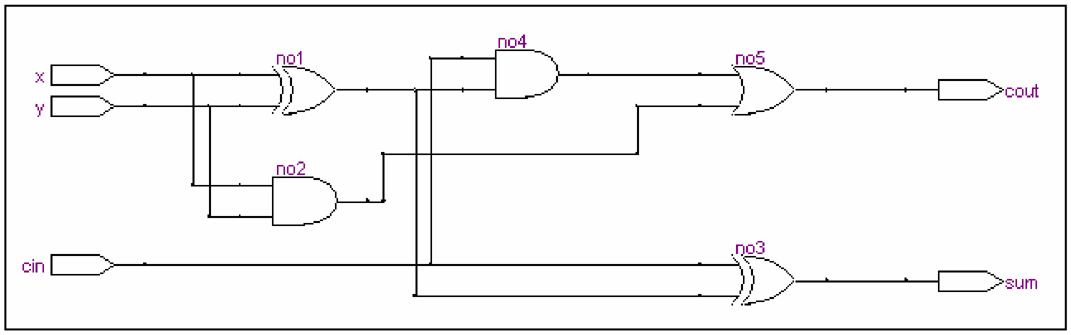
Source Code for a 1-bit Conventional Full Adder:

```
//Define a 1-bit full adder
module fulladd (sum, cout, x, y, cin);

//I/O port declarations
output sum, cout;
input x, y, cin;

//Internal nets
wire s1, s2, c1;

//Instantiate logic gate primitives
xor no1 (s1, x, y);
and no2 (c1, x, y);
xor no3 (sum, s1, cin);
and no4 (s2, s1, cin);
or no5 (cout, s2, c1);

endmodule
```

Simulation Result for a 1-bit Conventional Full Adder:



RTL View for a 1-bit Conventional Full Adder:



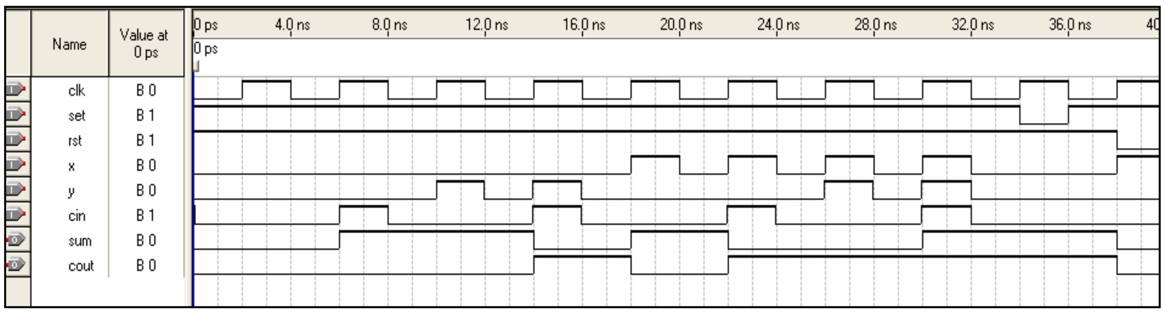
Appendix E

Source Code, Simulation Result and RTL View for A 1-bit Pipeline Full Adder

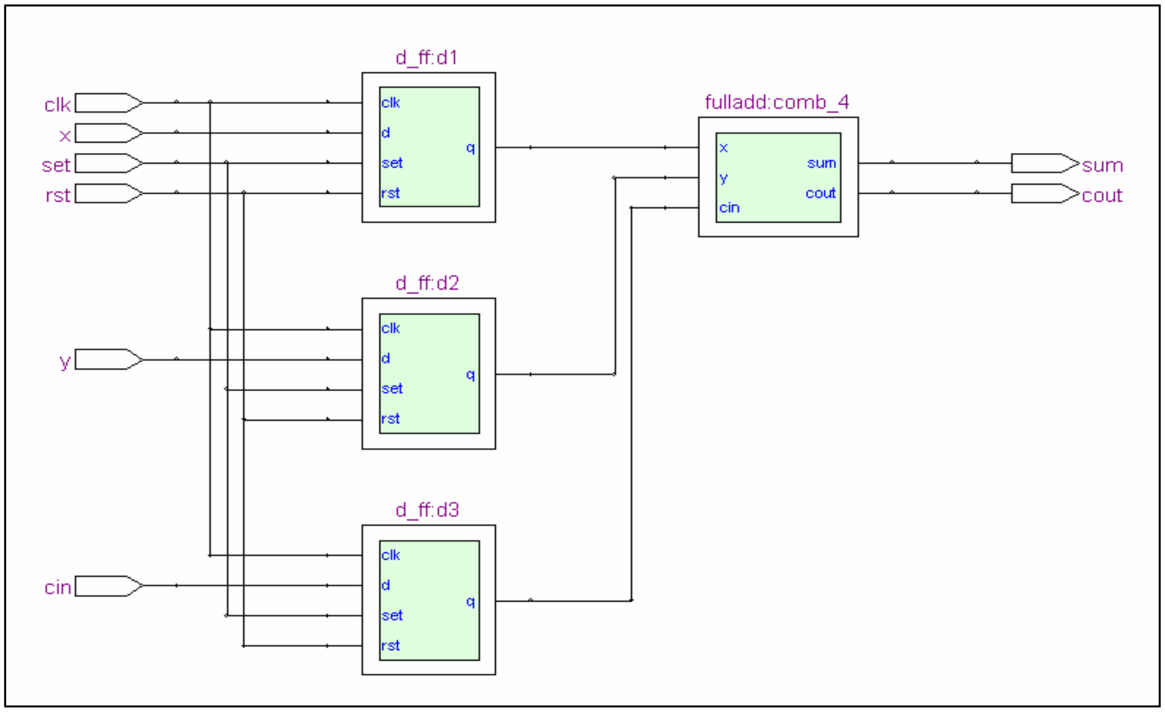
Source Code for a 1-bit Pipeline Full Adder:

```
//Define a pipeline 1-bit full adder
module fulladd_pl (sum, cout, clk, set, rst, x, y, cin);

//I/O port declarations
output sum, cout;
input clk, set, rst, x, y, cin;

//Internal nets
wire x1, y1, cin1;

//Instantiate logic gate primitives
d_ff d1 (x1, clk, x, set, rst);
d_ff d2 (y1, clk, y, set, rst);
d_ff d3 (cin1, clk, cin, set, rst);

fulladd (sum, cout, x1, y1, cin1);

endmodule
```

Simulation Result for a 1-bit Pipeline Full Adder:



RTL View for a 1-bit Pipeline Full Adder:



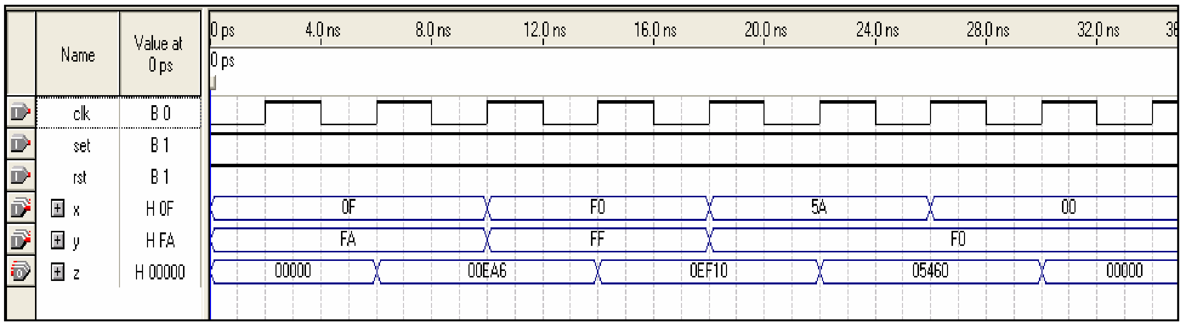
Appendix F

Source Code, Simulation Result and RTL View for Conventional High Speed 8-bits x 8-bits Wallace Tree Multiplier

Source Code for Conventional High Speed 8-bits x 8-bits Wallace Tree Multiplier:

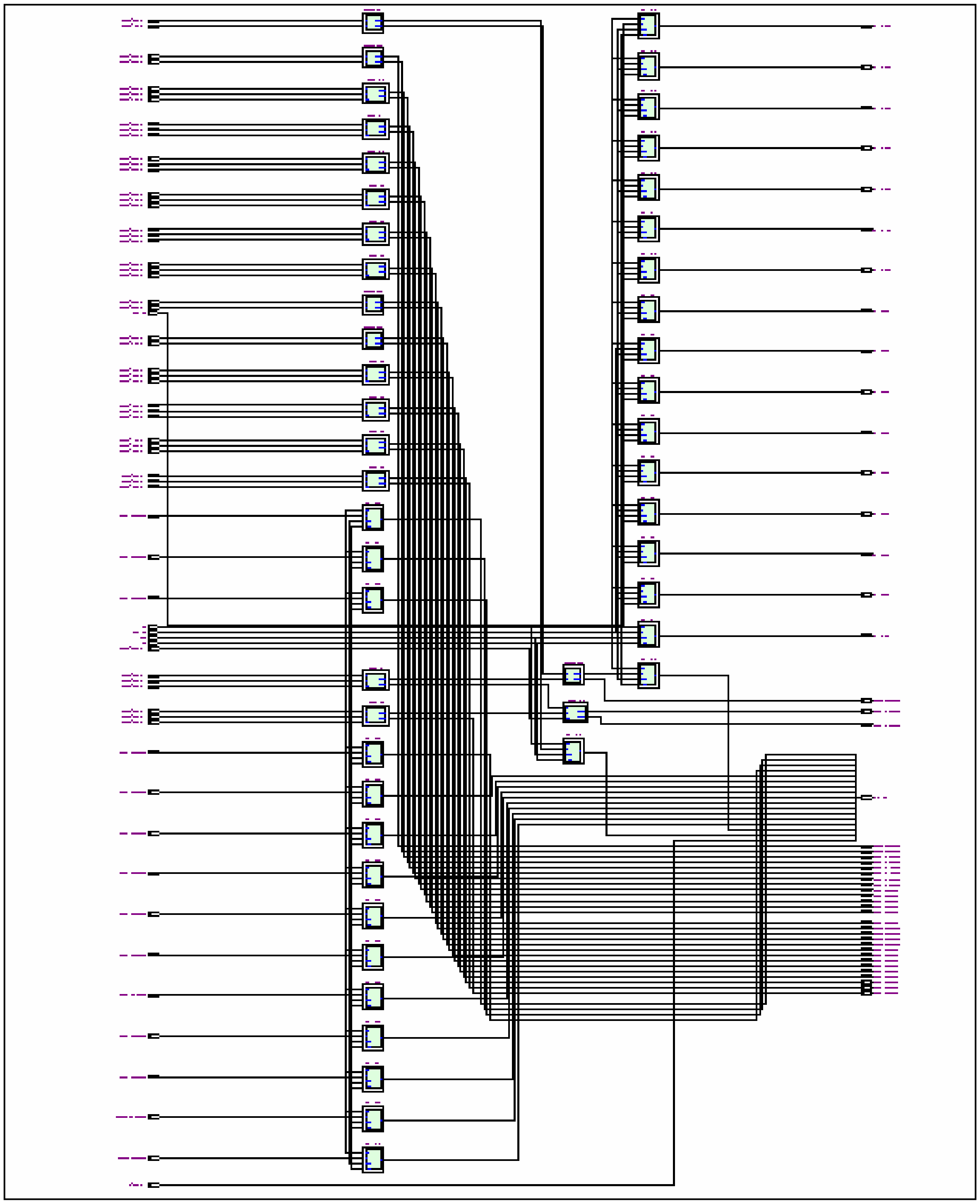
```
// Define an 8-bits x 8-bits Conventional Wallace Tree multiplier
module wallace_t(z, clk, set, rst, x, y);

// I/O port declarations
output [16:0] z;
input [7:0] x, y;
input clk, set, rst;

// Internal nets
reg [15:1] sh, ch;
reg [49:1] sf, cf;
reg [16:1] w;

// Instantiate an 8-bits x 8-bits Wallace Tree multiplier

d_ff d1 (w[1], clk, x[0], set, rst);
d_ff d2 (w[2], clk, x[1], set, rst);
d_ff d3 (w[3], clk, x[2], set, rst);
d_ff d4 (w[4], clk, x[3], set, rst);
d_ff d5 (w[5], clk, x[4], set, rst);
d_ff d6 (w[6], clk, x[5], set, rst);
d_ff d7 (w[7], clk, x[6], set, rst);
d_ff d8 (w[8], clk, x[7], set, rst);

d_ff d9 (w[9], clk, y[0], set, rst);
d_ff d10 (w[10], clk, y[1], set, rst);
d_ff d11 (w[11], clk, y[2], set, rst);
d_ff d12 (w[12], clk, y[3], set, rst);
d_ff d13 (w[13], clk, y[4], set, rst);
d_ff d14 (w[14], clk, y[5], set, rst);
d_ff d15 (w[15], clk, y[6], set, rst);
d_ff d16 (w[16], clk, y[7], set, rst);

halfadd ha1 (sh[1], ch[1], (w[1] & w[10]), (w[2] & w[9]));
fulladd fa1 (sf[1], cf[1], (w[1] & w[11]), (w[2] & w[10]), (w[3] & w[9]));
fulladd fa2 (sf[2], cf[2], (w[2] & w[11]), (w[3] & w[10]), (w[4] & w[9]));
fulladd fa3 (sf[3], cf[3], (w[3] & w[11]), (w[4] & w[10]), (w[5] & w[9]));
fulladd fa4 (sf[4], cf[4], (w[4] & w[11]), (w[5] & w[10]), (w[6] & w[9]));
fulladd fa5 (sf[5], cf[5], (w[5] & w[11]), (w[6] & w[10]), (w[7] & w[9]));
fulladd fa6 (sf[6], cf[6], (w[6] & w[11]), (w[7] & w[10]), (w[8] & w[9]));
halfadd ha2 (sh[2], ch[2], (w[7] & w[11]), (w[8] & w[10]));
halfadd ha3 (sh[3], ch[3], (w[1] & w[13]), (w[2] & w[12]));
fulladd fa7 (sf[7], cf[7], (w[1] & w[14]), (w[2] & w[13]), (w[3] & w[12]));
fulladd fa8 (sf[8], cf[8], (w[2] & w[14]), (w[3] & w[13]), (w[4] & w[12]));
fulladd fa9 (sf[9], cf[9], (w[3] & w[14]), (w[4] & w[13]), (w[5] & w[12]));
fulladd fa10 (sf[10], cf[10], (w[4] & w[14]), (w[5] & w[13]), (w[6] & w[12]));
fulladd fa11 (sf[11], cf[11], (w[5] & w[14]), (w[6] & w[13]), (w[7] & w[12]));
fulladd fa12 (sf[12], cf[12], (w[6] & w[14]), (w[7] & w[13]), (w[8] & w[12]));
halfadd ha4 (sh[4], ch[4], (w[7] & w[14]), (w[8] & w[13]));

halfadd ha5 (sh[5], ch[5], ch[1], sf[1]);
fulladd fa13 (sf[13], cf[13], cf[1], sf[2], (w[1] & w[12]));
fulladd fa14 (sf[14], cf[14], cf[2], sf[3], sh[3]);
fulladd fa15 (sf[15], cf[15], cf[3], sf[4], sf[7]);
fulladd fa16 (sf[16], cf[16], cf[4], sf[5], sf[8]);
fulladd fa17 (sf[17], cf[17], cf[5], sf[6], sf[9]);
fulladd fa18 (sf[18], cf[18], cf[6], sh[2], sf[10]);
fulladd fa19 (sf[19], cf[19], ch[2], sf[11], (w[8] & w[11]));

halfadd ha6 (sh[6], ch[6], cf[7], (w[1] & w[15]));
fulladd fa20 (sf[20], cf[20], cf[8], (w[1] & w[16]), (w[2] & w[15]));
fulladd fa21 (sf[21], cf[21], cf[9], (w[2] & w[16]), (w[3] & w[15]));
fulladd fa22 (sf[22], cf[22], cf[10], (w[3] & w[16]), (w[4] & w[15]));
fulladd fa23 (sf[23], cf[23], cf[11], (w[4] & w[16]), (w[5] & w[15]));
fulladd fa24 (sf[24], cf[24], cf[12], (w[5] & w[16]), (w[6] & w[15]));
fulladd fa25 (sf[25], cf[25], ch[4], (w[6] & w[16]), (w[7] & w[15]));
halfadd ha7 (sh[7], ch[7], (w[7] & w[16]), (w[8] & w[15]));

halfadd ha8 (sh[8], ch[8], ch[5], sf[13]);
halfadd ha9 (sh[9], ch[9], cf[13], sf[14]);
fulladd fa26 (sf[26], cf[26], cf[14], sf[15], ch[3]);
fulladd fa27 (sf[27], cf[27], cf[15], sf[16], sh[6]);
fulladd fa28 (sf[28], cf[28], cf[16], sf[17], sf[20]);
fulladd fa29 (sf[29], cf[29], cf[17], sf[18], sf[21]);
fulladd fa30 (sf[30], cf[30], cf[18], sf[19], sf[22]);
fulladd fa31 (sf[31], cf[31], cf[19], sf[12], sf[23]);
halfadd ha10 (sh[10], ch[10], sh[4], sf[24]);
halfadd ha11 (sh[11], ch[11], (w[8] & w[14]), sf[25]);

halfadd ha12 (sh[12], ch[12], ch[8], sh[9]);
halfadd ha13 (sh[13], ch[13], ch[9], sf[26]);
halfadd ha14 (sh[14], ch[14], cf[26], sf[27]);
fulladd fa32 (sf[32], cf[32], cf[27], sf[28], ch[6]);
fulladd fa33 (sf[33], cf[33], cf[28], sf[29], cf[20]);
fulladd fa34 (sf[34], cf[34], cf[29], sf[30], cf[21]);
fulladd fa35 (sf[35], cf[35], cf[30], sf[31], cf[22]);
fulladd fa36 (sf[36], cf[36], cf[31], sh[10], cf[23]);
fulladd fa37 (sf[37], cf[37], ch[10], sh[11], cf[24]);
fulladd fa38 (sf[38], cf[38], ch[11], sh[7], cf[25]);
halfadd ha15 (sh[15], ch[15], ch[7], (w[8] & w[16]));

fulladd fa39 (sf[39], cf[39], 1'b0, ch[12], sh[13]);
fulladd fa40 (sf[40], cf[40], cf[39], ch[13], sh[14]);
fulladd fa41 (sf[41], cf[41], cf[40], ch[14], sf[32]);
fulladd fa42 (sf[42], cf[42], cf[41], cf[32], sf[33]);
fulladd fa43 (sf[43], cf[43], cf[42], cf[33], sf[34]);
fulladd fa44 (sf[44], cf[44], cf[43], cf[34], sf[35]);
fulladd fa45 (sf[45], cf[45], cf[44], cf[35], sf[36]);
fulladd fa46 (sf[46], cf[46], cf[45], cf[36], sf[37]);
fulladd fa47 (sf[47], cf[47], cf[46], cf[37], sf[38]);
fulladd fa48 (sf[48], cf[48], cf[47], cf[38], sh[15]);
fulladd fa49 (sf[49], cf[49], cf[48], ch[15], 1'b0);

assign z[0] = w[1] & w[9];

d_ff d17 (z[1], clk, sh[1], set, rst);
d_ff d18 (z[2], clk, sh[5], set, rst);
d_ff d19 (z[3], clk, sh[8], set, rst);
d_ff d20 (z[4], clk, sh[12], set, rst);
d_ff d21 (z[5], clk, sf[39], set, rst);
d_ff d22 (z[6], clk, sf[40], set, rst);
d_ff d23 (z[7], clk, sf[41], set, rst);
d_ff d24 (z[8], clk, sf[42], set, rst);
d_ff d25 (z[9], clk, sf[43], set, rst);
d_ff d26 (z[10], clk, sf[44], set, rst);
d_ff d27 (z[11], clk, sf[45], set, rst);
d_ff d28 (z[12], clk, sf[46], set, rst);
d_ff d29 (z[13], clk, sf[47], set, rst);
d_ff d30 (z[14], clk, sf[48], set, rst);
d_ff d31 (z[15], clk, sf[49], set, rst);
d_ff d32 (z[16], clk, cf[49], set, rst);

endmodule
```

Simulation Result for Conventional High Speed 8-bits x 8-bits Wallace Tree Multiplier:



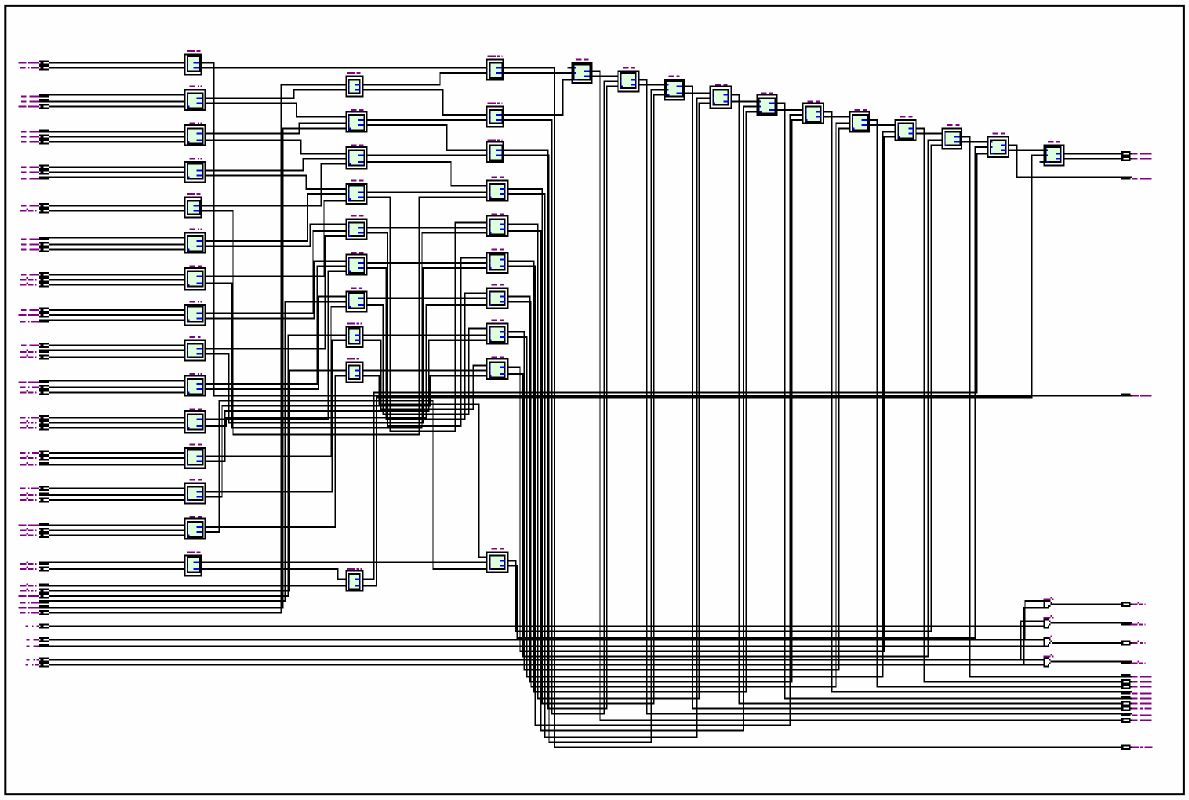
RTL View for Conventional High Speed 8-bits x 8-bits Wallace Tree Multiplier:

Page 1 of 4:



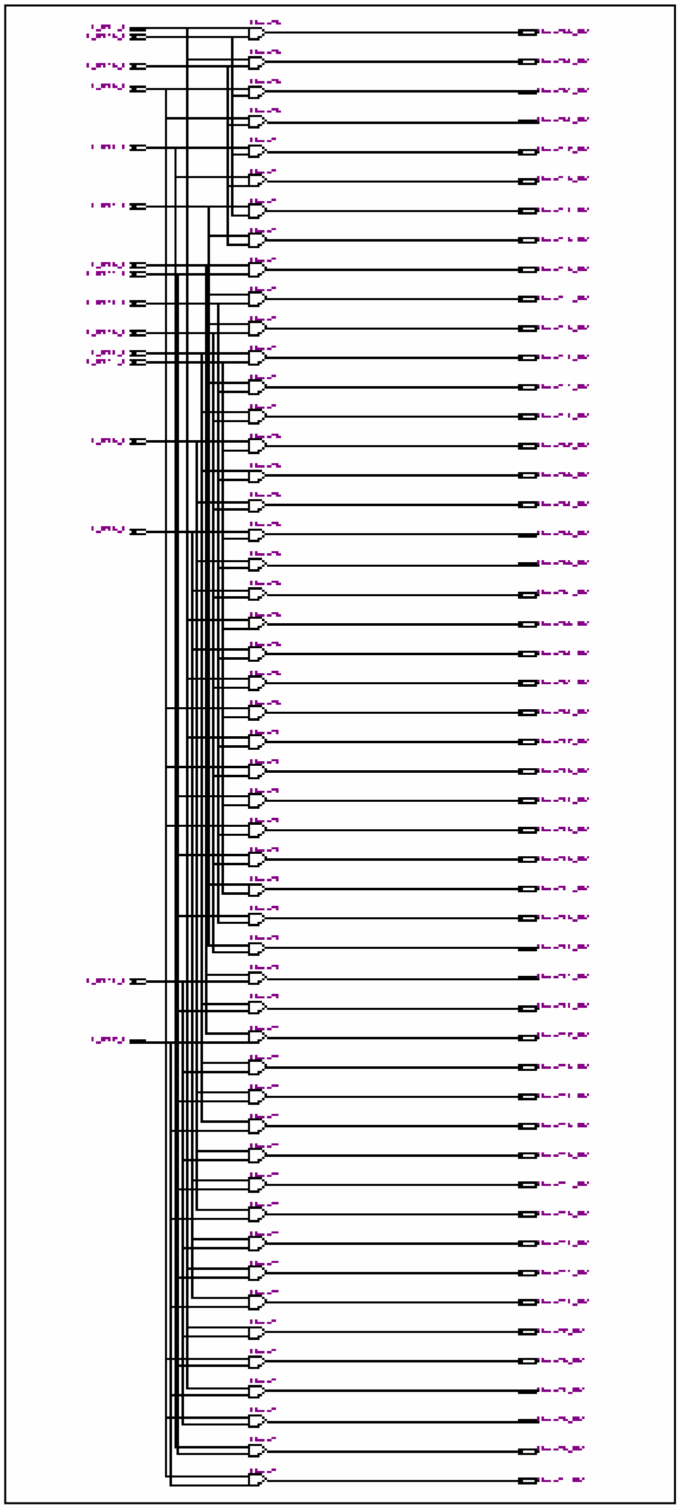
RTL View for Conventional High Speed 8-bits x 8-bits Wallace Tree Multiplier:

Page 2 of 4:



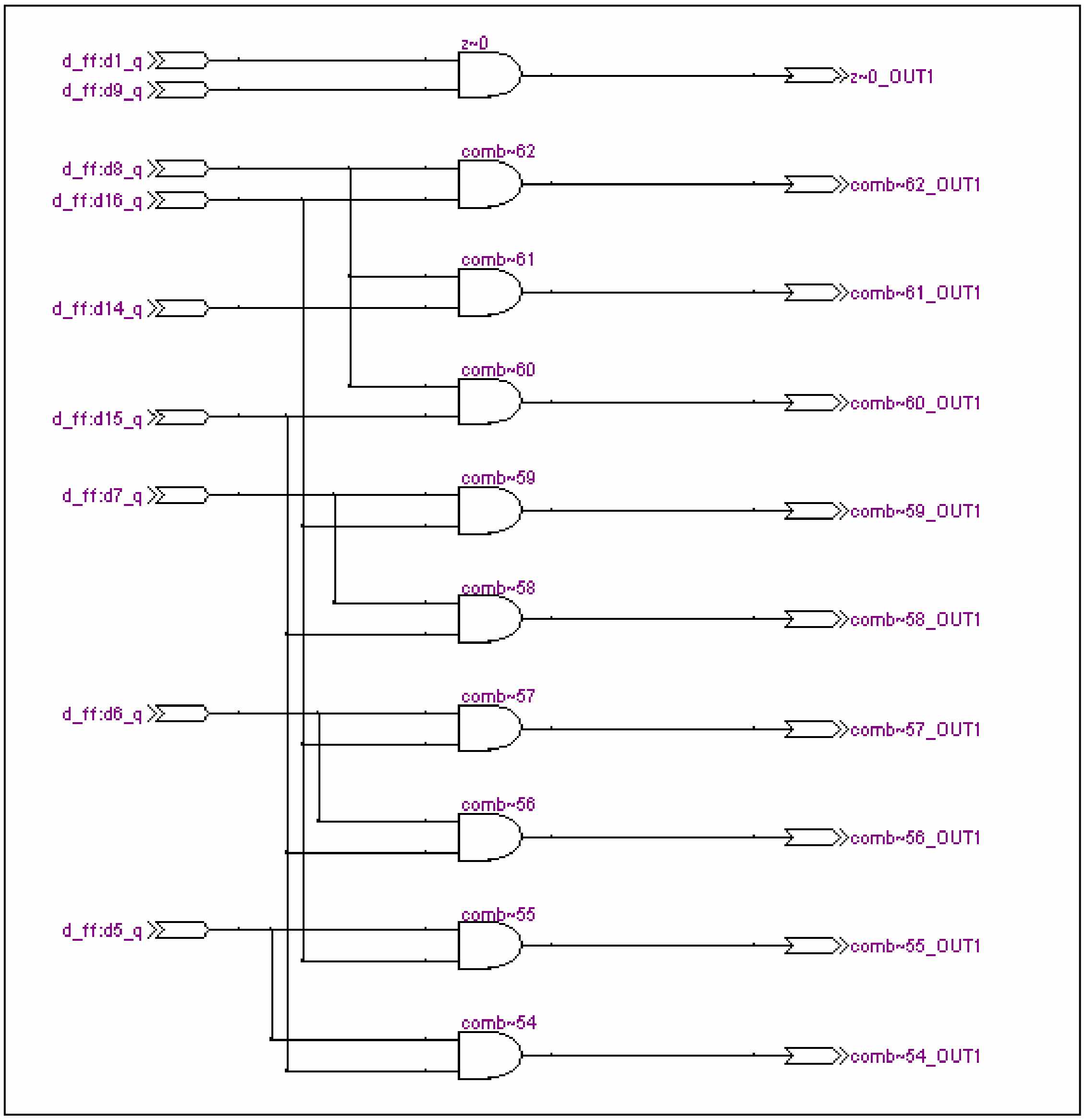
RTL View for Conventional High Speed 8-bits x 8-bits Wallace Tree Multiplier:

Page 3 of 4:



RTL View for Conventional High Speed 8-bits x 8-bits Wallace Tree Multiplier:

Page 4 of 4:

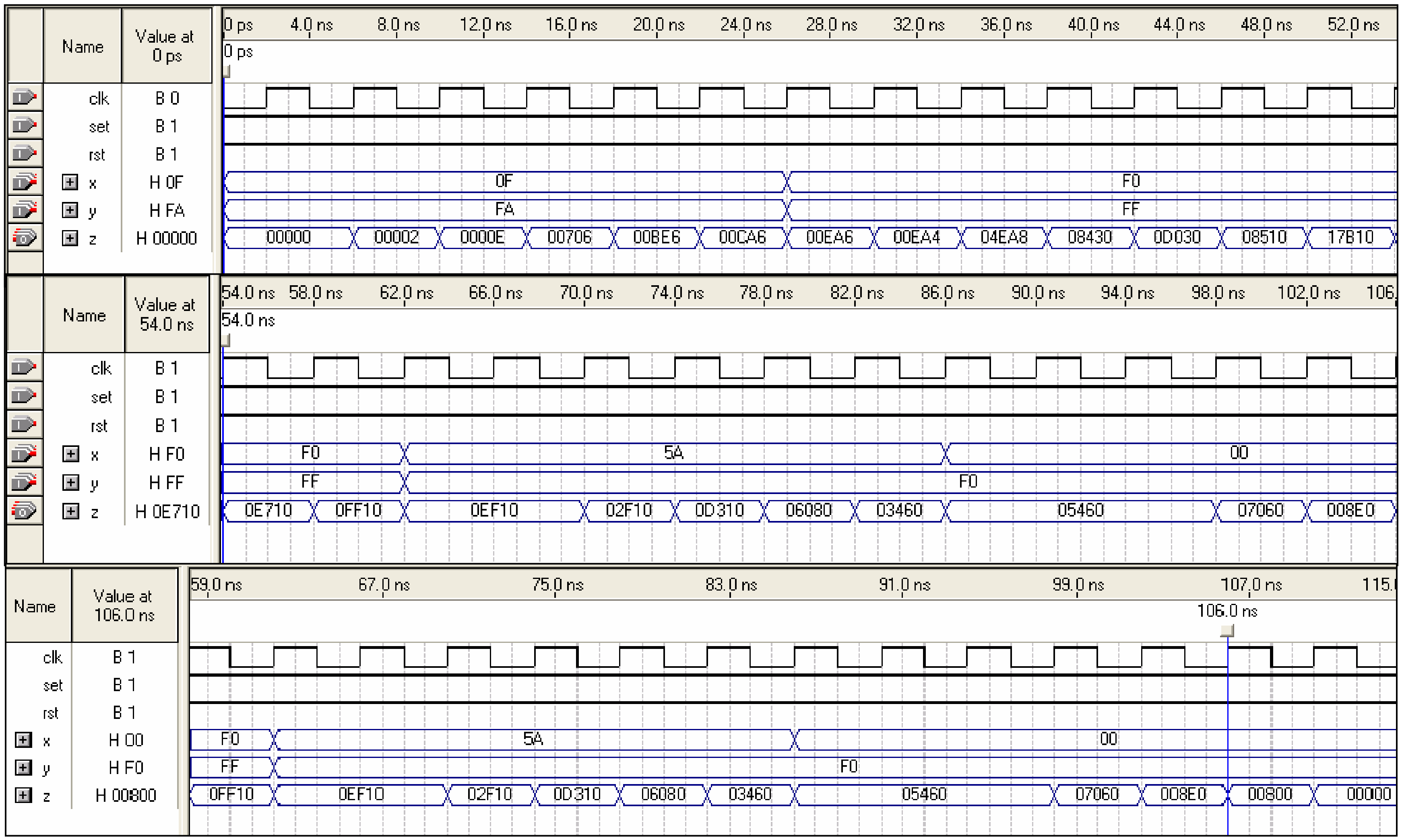


Appendix G

Source Code, Simulation Result and RTL View for Pipeline High Speed 8-bits x 8-bits Wallace Tree Multiplier

Source Code for Pipeline High Speed 8-bits x 8-bits Wallace Tree Multiplier:

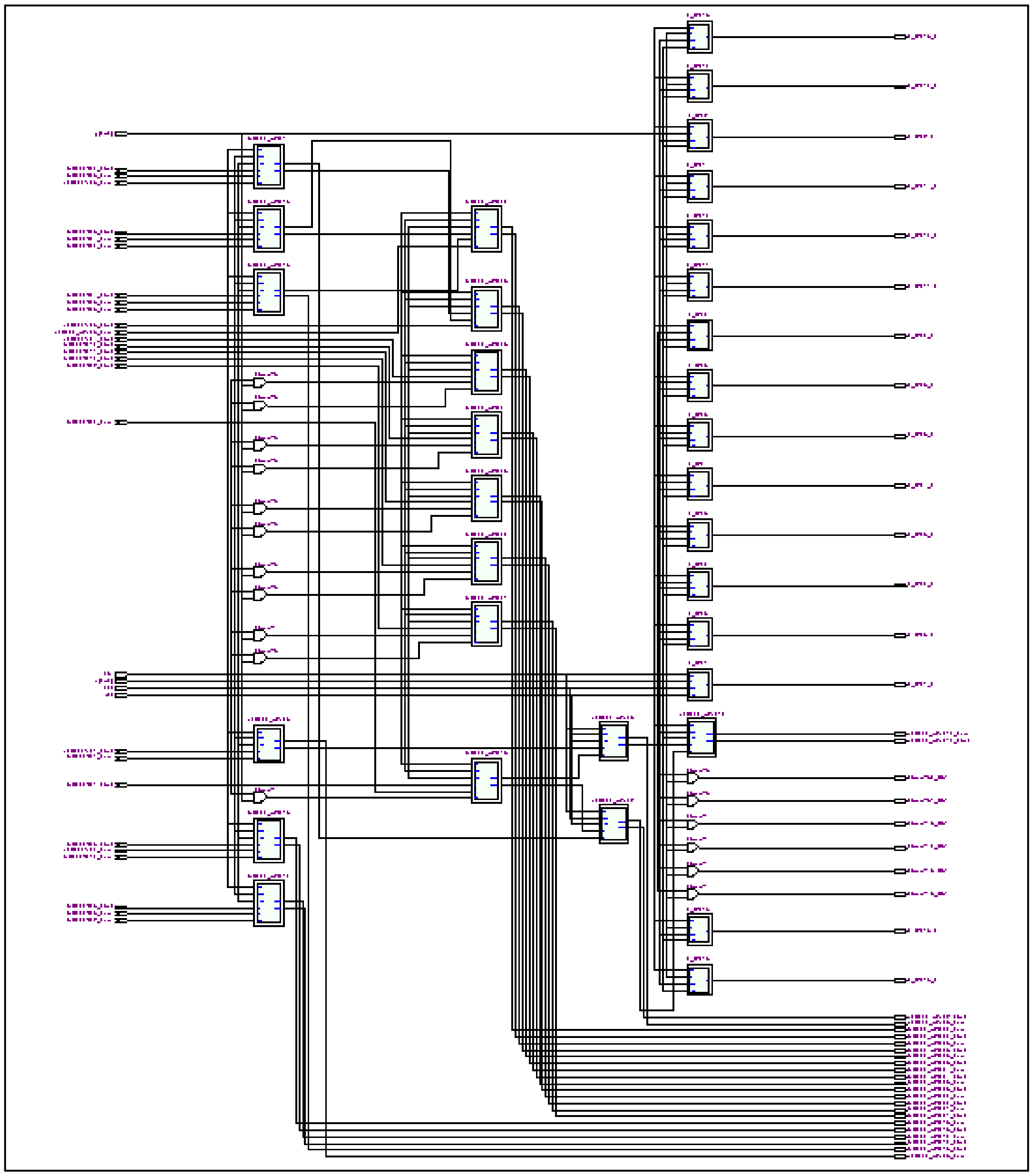
```
// Define an 8-bits x 8-bits Pipeline Wallace Tree multiplier
module wallace_t_pl(z, clk, set, rst, x, y);

// I/O port declarations
output [16:0] z;
input [7:0]  x, y;
input clk, set, rst;

// Internal nets
reg [15:1] sh, ch;
reg [49:1] sf, cf;
reg [16:1] w;

// Instantiate an 8-bits x 8-bits Wallace Tree multiplier

d_ff d1 (w[1], clk, x[0], set, rst);
d_ff d2 (w[2], clk, x[1], set, rst);
d_ff d3 (w[3], clk, x[2], set, rst);
d_ff d4 (w[4], clk, x[3], set, rst);
d_ff d5 (w[5], clk, x[4], set, rst);
d_ff d6 (w[6], clk, x[5], set, rst);
d_ff d7 (w[7], clk, x[6], set, rst);
d_ff d8 (w[8], clk, x[7], set, rst);

d_ff d9 (w[9], clk, y[0], set, rst);
d_ff d10 (w[10], clk, y[1], set, rst);
d_ff d11 (w[11], clk, y[2], set, rst);
d_ff d12 (w[12], clk, y[3], set, rst);
d_ff d13 (w[13], clk, y[4], set, rst);
d_ff d14 (w[14], clk, y[5], set, rst);
d_ff d15 (w[15], clk, y[6], set, rst);
d_ff d16 (w[16], clk, y[7], set, rst);

halfadd ha1 (sh[1], ch[1], (w[1] & w[10]), (w[2] & w[9]));
fulladd fa1 (sf[1], cf[1], (w[1] & w[11]), (w[2] & w[10]), (w[3] & w[9]));
fulladd fa2 (sf[2], cf[2], (w[2] & w[11]), (w[3] & w[10]), (w[4] & w[9]));
fulladd fa3 (sf[3], cf[3], (w[3] & w[11]), (w[4] & w[10]), (w[5] & w[9]));
fulladd fa4 (sf[4], cf[4], (w[4] & w[11]), (w[5] & w[10]), (w[6] & w[9]));
fulladd fa5 (sf[5], cf[5], (w[5] & w[11]), (w[6] & w[10]), (w[7] & w[9]));
fulladd fa6 (sf[6], cf[6], (w[6] & w[11]), (w[7] & w[10]), (w[8] & w[9]));
halfadd ha2 (sh[2], ch[2], (w[7] & w[11]), (w[8] & w[10]));
halfadd ha3 (sh[3], ch[3], (w[1] & w[13]), (w[2] & w[12]));
fulladd fa7 (sf[7], cf[7], (w[1] & w[14]), (w[2] & w[13]), (w[3] & w[12]));
fulladd fa8 (sf[8], cf[8], (w[2] & w[14]), (w[3] & w[13]), (w[4] & w[12]));
fulladd fa9 (sf[9], cf[9], (w[3] & w[14]), (w[4] & w[13]), (w[5] & w[12]));
fulladd fa10 (sf[10], cf[10], (w[4] & w[14]), (w[5] & w[13]), (w[6] & w[12]));
fulladd fa11 (sf[11], cf[11], (w[5] & w[14]), (w[6] & w[13]), (w[7] & w[12]));
fulladd fa12 (sf[12], cf[12], (w[6] & w[14]), (w[7] & w[13]), (w[8] & w[12]));
halfadd ha4 (sh[4], ch[4], (w[7] & w[14]), (w[8] & w[13]));

halfadd_pl ha5 (sh[5], ch[5], clk, set, rst, ch[1], sf[1]);
fulladd_pl fa13 (sf[13], cf[13], clk, set, rst, cf[1], sf[2], (x[0] & y[3]));
fulladd_pl fa14 (sf[14], cf[14], clk, set, rst, cf[2], sf[3], sh[3]);
fulladd_pl fa15 (sf[15], cf[15], clk, set, rst, cf[3], sf[4], sf[7]);
fulladd_pl fa16 (sf[16], cf[16], clk, set, rst, cf[4], sf[5], sf[8]);
fulladd_pl fa17 (sf[17], cf[17], clk, set, rst, cf[5], sf[6], sf[9]);
fulladd_pl fa18 (sf[18], cf[18], clk, set, rst, cf[6], sh[2], sf[10]);
fulladd_pl fa19 (sf[19], cf[19], clk, set, rst, ch[2], sf[11], (x[7] & y[2]));

halfadd_pl ha6 (sh[6], ch[6], clk, set, rst, cf[7], (x[0] & y[6]));
fulladd_pl fa20 (sf[20], cf[20], clk, set, rst, cf[8], (x[0] & y[7]), (x[1] & y[6]));
fulladd_pl fa21 (sf[21], cf[21], clk, set, rst, cf[9], (x[1] & y[7]), (x[2] & y[6]));
fulladd_pl fa22 (sf[22], cf[22], clk, set, rst, cf[10], (x[2] & y[7]), (x[3] & y[6]));
fulladd_pl fa23 (sf[23], cf[23], clk, set, rst, cf[11], (x[3] & y[7]), (x[4] & y[6]));
fulladd_pl fa24 (sf[24], cf[24], clk, set, rst, cf[12], (x[4] & y[7]), (x[5] & y[6]));
fulladd_pl fa25 (sf[25], cf[25], clk, set, rst, ch[4], (x[5] & y[7]), (x[6] & y[6]));
halfadd ha7 (sh[7], ch[7], (w[7] & w[16]), (w[8] & w[15]));

halfadd_pl ha8 (sh[8], ch[8], clk, set, rst, ch[5], sf[13]);
halfadd_pl ha9 (sh[9], ch[9], clk, set, rst, cf[13], sf[14]);
fulladd_pl fa26 (sf[26], cf[26], clk, set, rst, cf[14], sf[15], ch[3]);
fulladd_pl fa27 (sf[27], cf[27], clk, set, rst, cf[15], sf[16], sh[6]);
fulladd_pl fa28 (sf[28], cf[28], clk, set, rst, cf[16], sf[17], sf[20]);
fulladd_pl fa29 (sf[29], cf[29], clk, set, rst, cf[17], sf[18], sf[21]);
fulladd_pl fa30 (sf[30], cf[30], clk, set, rst, cf[18], sf[19], sf[22]);
fulladd_pl fa31 (sf[31], cf[31], clk, set, rst, cf[19], sf[12], sf[23]);
halfadd_pl ha10 (sh[10], ch[10], clk, set, rst, sh[4], sf[24]);
halfadd_pl ha11 (sh[11], ch[11], clk, set, rst, (x[7] & y[5]), sf[25]);

halfadd_pl ha12 (sh[12], ch[12], clk, set, rst, ch[8], sh[9]);
halfadd_pl ha13 (sh[13], ch[13], clk, set, rst, ch[9], sf[26]);
halfadd_pl ha14 (sh[14], ch[14], clk, set, rst, cf[26], sf[27]);
fulladd_pl fa32 (sf[32], cf[32], clk, set, rst, cf[27], sf[28], ch[6]);
fulladd_pl fa33 (sf[33], cf[33], clk, set, rst, cf[28], sf[29], cf[20]);
fulladd_pl fa34 (sf[34], cf[34], clk, set, rst, cf[29], sf[30], cf[21]);
fulladd_pl fa35 (sf[35], cf[35], clk, set, rst, cf[30], sf[31], cf[22]);
fulladd_pl fa36 (sf[36], cf[36], clk, set, rst, cf[31], sh[10], cf[23]);
fulladd_pl fa37 (sf[37], cf[37], clk, set, rst, ch[10], sh[11], cf[24]);
fulladd_pl fa38 (sf[38], cf[38], clk, set, rst, ch[11], sh[7], cf[25]);
halfadd_pl ha15 (sh[15], ch[15], clk, set, rst, ch[7], (x[7] & y[7]));

fulladd_pl fa39 (sf[39], cf[39], clk, set, rst, 1'b0, ch[12], sh[13]);
fulladd_pl fa40 (sf[40], cf[40], clk, set, rst, cf[39], ch[13], sh[14]);
fulladd_pl fa41 (sf[41], cf[41], clk, set, rst, cf[40], ch[14], sf[32]);
fulladd_pl fa42 (sf[42], cf[42], clk, set, rst, cf[41], cf[32], sf[33]);
fulladd_pl fa43 (sf[43], cf[43], clk, set, rst, cf[42], cf[33], sf[34]);
fulladd_pl fa44 (sf[44], cf[44], clk, set, rst, cf[43], cf[34], sf[35]);
fulladd_pl fa45 (sf[45], cf[45], clk, set, rst, cf[44], cf[35], sf[36]);
fulladd_pl fa46 (sf[46], cf[46], clk, set, rst, cf[45], cf[36], sf[37]);
fulladd_pl fa47 (sf[47], cf[47], clk, set, rst, cf[46], cf[37], sf[38]);
fulladd_pl fa48 (sf[48], cf[48], clk, set, rst, cf[47], cf[38], sh[15]);
fulladd_pl fa49 (sf[49], cf[49], clk, set, rst, cf[48], ch[15], 1'b0);

assign z[0] = w[1] & w[9];

d_ff d17 (z[1], clk, sh[1], set, rst);
d_ff d18 (z[2], clk, sh[5], set, rst);
d_ff d19 (z[3], clk, sh[8], set, rst);
d_ff d20 (z[4], clk, sh[12], set, rst);
d_ff d21 (z[5], clk, sf[39], set, rst);
d_ff d22 (z[6], clk, sf[40], set, rst);
d_ff d23 (z[7], clk, sf[41], set, rst);
d_ff d24 (z[8], clk, sf[42], set, rst);
d_ff d25 (z[9], clk, sf[43], set, rst);
d_ff d26 (z[10], clk, sf[44], set, rst);
d_ff d27 (z[11], clk, sf[45], set, rst);
d_ff d28 (z[12], clk, sf[46], set, rst);
d_ff d29 (z[13], clk, sf[47], set, rst);
d_ff d30 (z[14], clk, sf[48], set, rst);
d_ff d31 (z[15], clk, sf[49], set, rst);
d_ff d32 (z[16], clk, cf[49], set, rst);

endmodule
```



Simulation result of the multiplication process by the high speed 8-bits x 8-bits Wallace Tree multiplier with pipelining

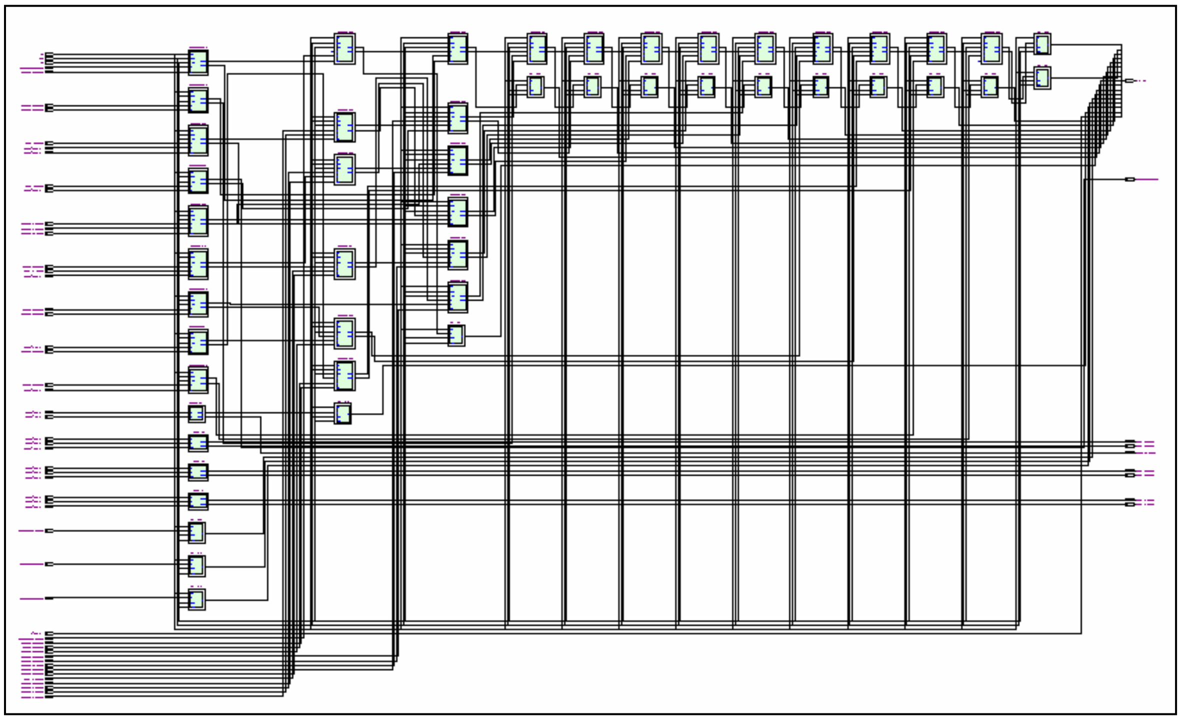
RTL View for Pipeline High Speed 8-bits x 8-bits Wallace Tree Multiplier:

Page 1 0f 4:



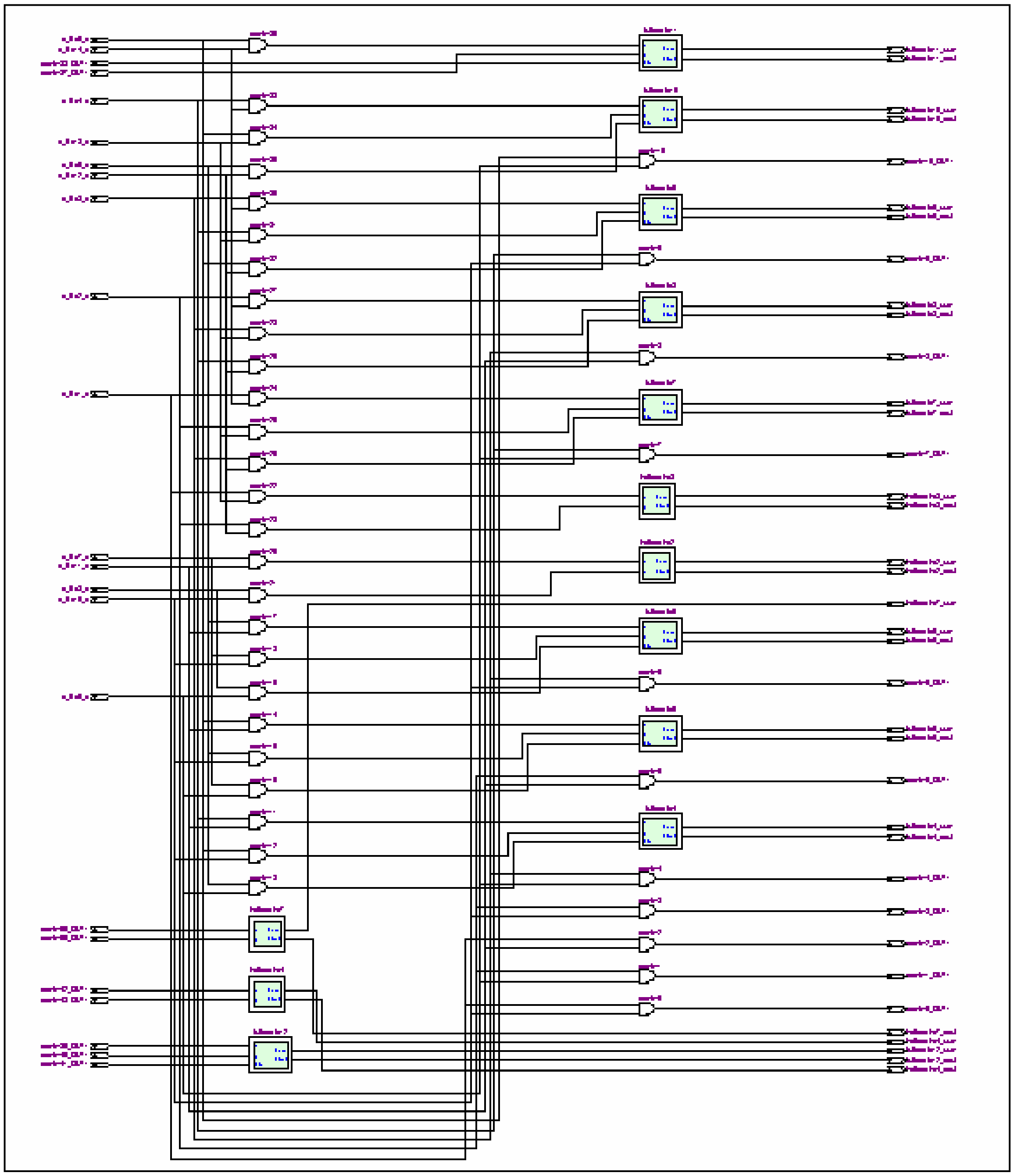
RTL View for Pipeline High Speed 8-bits x 8-bits Wallace Tree Multiplier:

Page 2 of 4



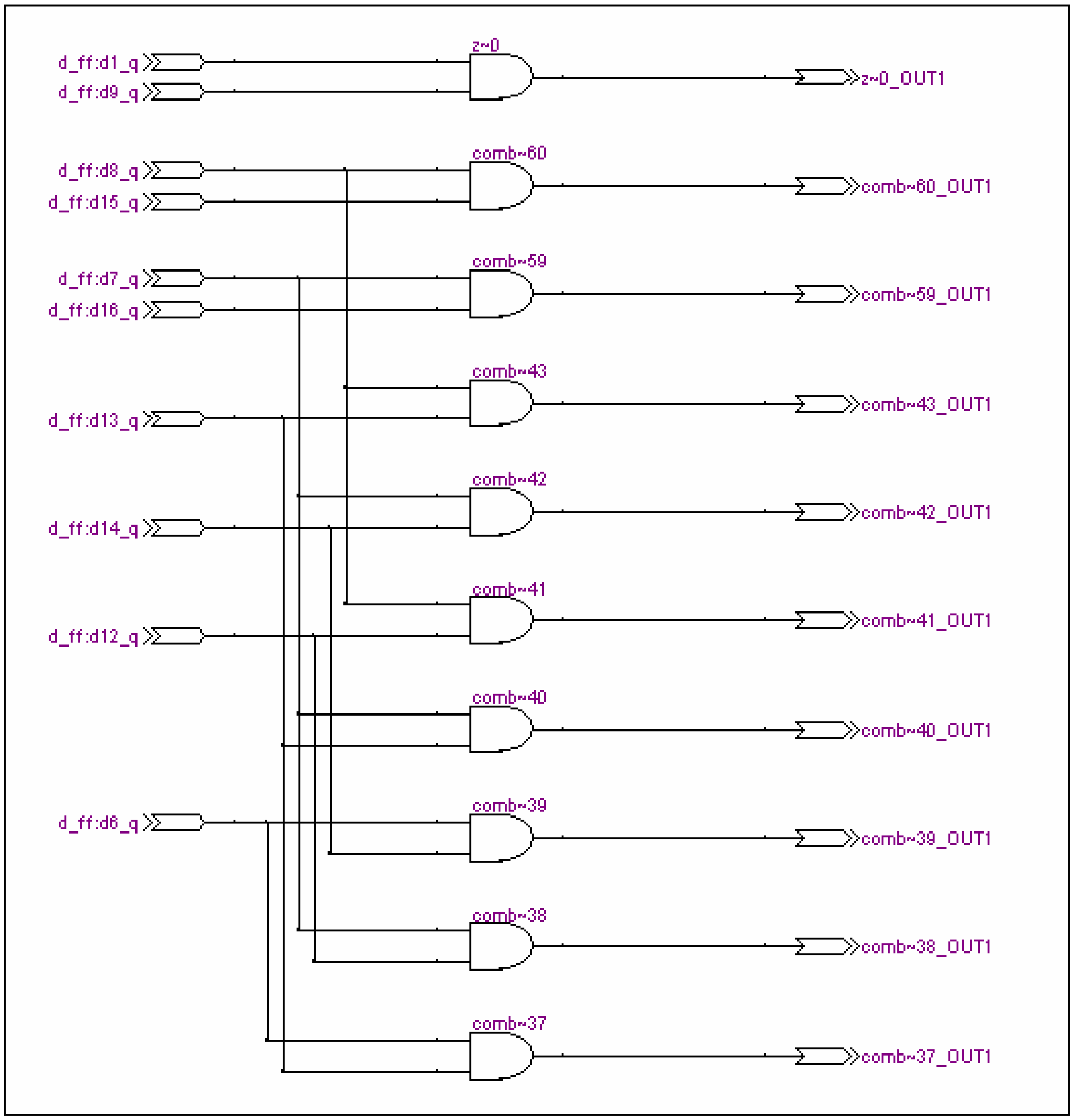
RTL View for Pipeline High Speed 8-bits x 8-bits Wallace Tree Multiplier:

Page 3 of 4



RTL View for Pipeline High Speed 8-bits x 8-bits Wallace Tree Multiplier:

Page 4 of 4


d_ff:d1_q
d_ff:d9_q
z~0
z~0_OUT1
d_ff:d8_q
d_ff:d15_q
comb~60
comb~60_OUT1
d_ff:d7_q
d_ff:d16_q
comb~59
comb~59_OUT1
d_ff:d13_q
comb~43
comb~43_OUT1
d_ff:d14_q
comb~42
comb~42_OUT1
d_ff:d12_q
comb~41
comb~41_OUT1
comb~40
comb~40_OUT1
d_ff:d6_q
comb~39
comb~39_OUT1
comb~38
comb~38_OUT1
comb~37
comb~37_OUT1