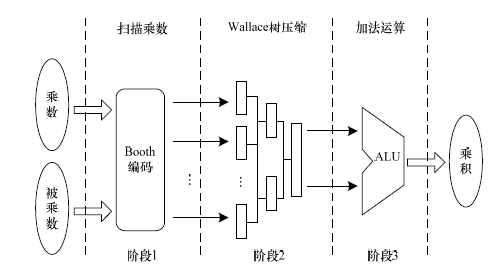
异步乘法器实验报告

张里蒙 胡凯 王世英

1.实验原理

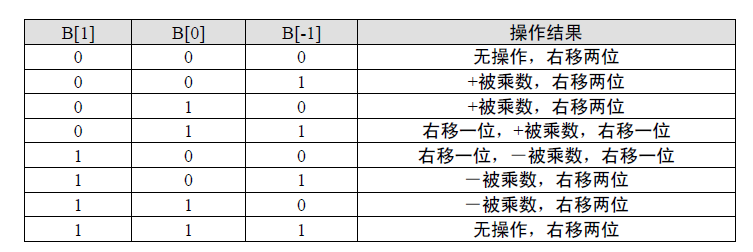
基于Booth 编码的乘法器处理过程大致相同，分为3 个阶段。第1 个阶段是扫描乘数进行Booth 编码产生若干个部分积，第2 个阶段将部分积输入Wallace 压缩树并行压缩至2 个操作数，第3 个阶段采用专门的加法器将2 个操作数相加得到最后的乘积结果。



2.实验设计

***booth编码模块（booth\_recoder.v）***

booth编码的原理如下图所示：



module booth\_recoder(

multiplicand,

code,

pp,

s

);

parameter BITWIDTH = 16;

input [BITWIDTH-1:0] multiplicand;

input [2:0] code;

output [BITWIDTH:0] pp;

output s;

reg [BITWIDTH:0] pp;

reg s;

always @ (multiplicand or code)

case(code)

3'b000:pp = {(BITWIDTH+1){1'b0}};//+0

3'b001:pp = {multiplicand[15],multiplicand}; //+M

3'b010:pp = {multiplicand[15],multiplicand}; //+M

3'b011:pp = {multiplicand,1'b0}; //+2M

3'b100:pp = {~multiplicand,1'b1};//-2M

3'b101:pp = {~multiplicand[15],(~multiplicand)};//-M

3'b110:pp = {~multiplicand[15],(~multiplicand)};//-M

3'b111:pp = {(BITWIDTH+1){1'b1}};//-0

endcase

always @ (multiplicand or code)

case(code[2])

1'b0:s = 1'b0;//+0,+M,+2M

1'b1:s = 1'b1;//-2M,-M,-0

endcase

endmodule

***部分积产生模块（bfj\_9.v）***

bfj\_9 j\_9(

.multiplicand1(multiplicand1),

.multiplier1(multiplier1),

.s0(s0),.s1(s1),.s2(s2),.s3(s3),.s4(s4),

.s5(s5),.s6(s6),.s7(s7),.s8(s8),

.j0(j0),.j1(j1),

.j2(j2),.j3(j3),

.j4(j4),.j5(j5),

.j6(j6),.j7(j7),

.j8(j8)

);

assign tmp\_prod=j0000+{j1000[25:0],6'b000000}+{s7,1'b0,s6,1'b0,s5,1'b0,s4,1'b0,s3,1'b0,s2,1'b0,s1,1'b0,s0};

//partial product accumulator

assign tp1 = j0+{j1,1'b0,s0};

assign tmp\_prod[1:0] = tp1[1:0];

assign tp2 = tp1[BITWIDTH+5:2] + {j2,1'b0,s1};

assign tmp\_prod[3:2] = tp2[1:0];

assign tp3 = tp2[BITWIDTH+5:2] + {j3,1'b0,s2};

assign tmp\_prod[5:4] = tp3[1:0];

assign tp4 = tp3[BITWIDTH+5:2] + {j4,1'b0,s3};

assign tmp\_prod[7:6] = tp4[1:0];

assign tp5 = tp4[BITWIDTH+5:2] + {j5,1'b0,s4};

assign tmp\_prod[9:8] = tp5[1:0];

assign tp6 = tp5[BITWIDTH+5:2] + {j6,1'b0,s5};

assign tmp\_prod[11:10] = tp6[1:0];

assign tp7 = tp6[BITWIDTH+5:2] + {j7,1'b0,s6};

assign tmp\_prod[13:12] = tp7[1:0];

assign tp8 = tp7[BITWIDTH+5:2] + {j8[BITWIDTH-1],1'b0,s7};

assign tmp\_prod[2\*BITWIDTH-1:14] = tp8[BITWIDTH+1:0];

always@ (posedge clk or posedge reset or posedge clk)

if(reset) product <=0;

else if(e0^e1) product<={1'b1,~tmp\_prod[2\*BITWIDTH-2:0]+1'b1};

else product <= tmp\_prod;

endmodule

***压缩模块用两个3-2压缩来组成4-2压缩（booth\_mul.v）***

module booth\_mul(multiplicand,multiplier,product,clk,reset);

parameter BITWIDTH = 16;

input clk,reset;

input [BITWIDTH-1:0] multiplier;

input [BITWIDTH-1:0] multiplicand;

output [2\*BITWIDTH-1:0] product;

reg [2\*BITWIDTH-1:0] product;

reg [15:0] multiplicand1,multiplier1;

wire [8:0] ss;

wire a,e0,e1;

wire s0,s1,s2,s3,s4,s5,s6,s7,s8;

wire [2\*BITWIDTH-1:0] tmp\_prod;

wire [BITWIDTH+3:0] tp8;

wire [BITWIDTH+5:0] tp1,tp2,tp3,tp4,tp5,tp6,tp7;

wire [BITWIDTH+3:0] j0;

wire [BITWIDTH+2:0] j1,j2,j3,j4,j5,j6;

wire [BITWIDTH+1:0] j7;

wire [BITWIDTH:0] j8;

assign e0=multiplicand[15];

assign e1=multiplier[15];

assign a= e0|e1;

always@ (posedge clk or posedge reset or negedge clk)

begin

if(reset)

product <=0;

else

begin

if(a)

begin

if(multiplicand[15]) multiplicand1 <= ~multiplicand+1'b1;

else multiplicand1 <= multiplicand;

if(multiplier[15]) multiplier1 <= ~multiplier+1'b1;

else multiplier1 <= multiplier;

end

else

{multiplicand1,multiplier1}<={multiplicand,multiplier};

end

end

***测试文件***

module tb\_booth\_mul();

reg signed [15:0] multiplicand;

reg signed [15:0] multiplier;

wire [31:0] product;

parameter BITWIDTH = 16;

reg clock,reset;

initial

begin

reset<=0;

clock <= 1'b0 ;

forever

#5 clock <= ~clock ;

end

always @(posedge clock)

begin

#5

multiplicand <= $random($random);

multiplier <= $random($random);

end

目前完成到16\*16乘法器部分，控制电路没成功加进去。