天津大学

计算机设计与调试

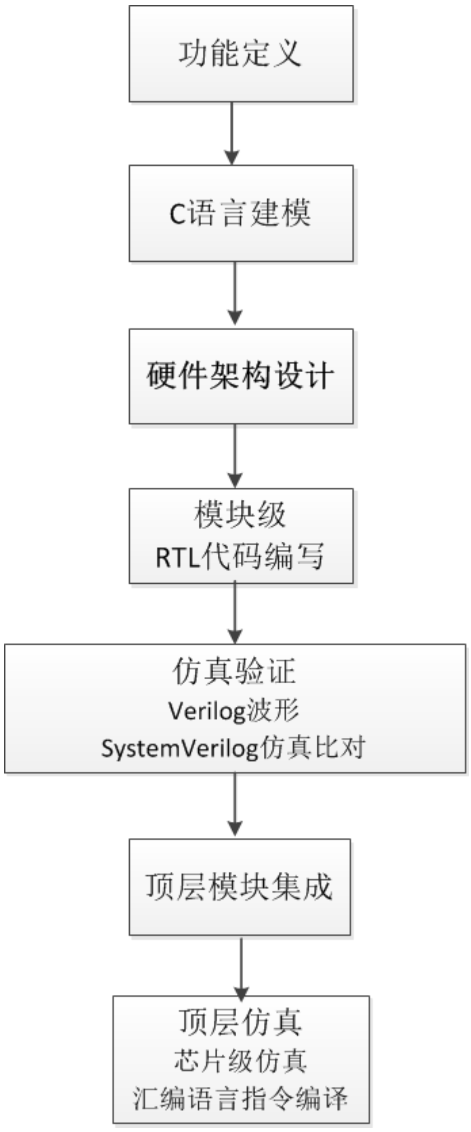
实验报告

学院	计算机科学与技术
专业	计算机科学与技术
年级	2013 级
学号	3013216096
姓名	梁振铎

2016 年 5 月 11 日

一、实验内容

1、用 C 或 C++语言，实现 VSFX 指令的功能仿真器（ISS）。
2、用 Verilog HLD 语言，实现 VSFX 指令硬件设计，即 VSFX 模块设计。
3、在基于 SystemVerilog 的自动仿真比对平台上，验证所设计的 VSFX 模块的功能正确性。
4、完成 PPC405+AltiVec 硬件设计与验证。将所设计的 VSFX 模块集成到已有的顶层设计中，完成指令功能验证。



二、 指令功能定义

1、vaddsws

有符号字向量（32 位）带饱和相加

Vector Add Signed Word Saturate *VX-form*

vaddsws VRT,VRA,VRB

4	VRT	VRA	VRB	896	
0	6	11	16	21	31

```
do i=0 to 127 by 32
    aop ← EXTS((VRA)$_{i:i+31}$)
    bop ← EXTS((VRB)$_{i:i+31}$)
    VRT$_{i:i+31}$ ← Clamp(aop $+_{int}$ bop, $-2^{31}$, $2^{31}-1$)
```

For each vector element *i* from 0 to 3, do the following.

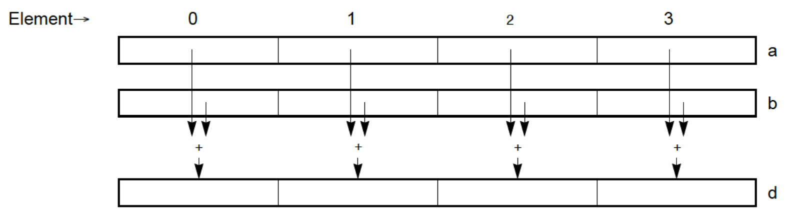
Signed-integer word element *i* in VRA is added to signed-integer word element *i* in VRB.

- If the sum is greater than $2^{31}-1$ the result saturates to $2^{31}-1$.
- If the sum is less than -2^{31} the result saturates to -2^{31}.

The low-order 32 bits of the result are placed into word element *i* of VRT.

Special Registers Altered:

SAT



d	a	b	maps to
vector unsigned int	vector unsigned int	vector unsigned int	vadduws d,a,b
	vector unsigned int	vector bool int	
	vector bool int	vector unsigned int	
vector signed int	vector signed int	vector signed int	vaddsws d,a,b
	vector signed int	vector bool int	
	vector bool int	vector signed int	

Figure 4-19. Add Saturating Four Integer Elements (32-bit)

2、vsububm

无符号字节向量（8 位）相减

Vector Subtract Unsigned Byte Modulo VX-form

vsububm VRT,VRA,VRB

4	VRT	VRA	VRB	1024
0	6	11	16	21 31

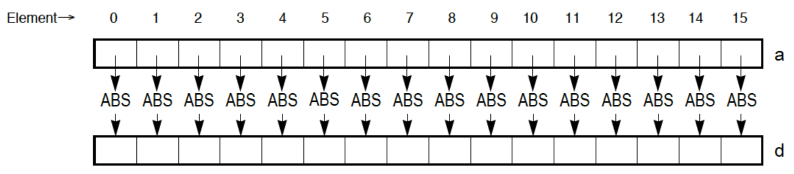
```
do i=0 to 127 by 8
    aop ← EXTZ((VRA)$_{i:i+7}$)
    bop ← EXTZ((VRB)$_{i:i+7}$)
    VRT$_{i:i+7}$ ← Chop( aop $+_{int}$ ¬bop $+_{int}$ 1, 8 )
```

For each vector element *i* from 0 to 15, do the following.

Unsigned-integer byte element *i* in VRB is subtracted from unsigned-integer byte element *i* in VRA. The low-order 8 bits of the result are placed into byte element *i* of VRT.

Special Registers Altered:

None



d	a	maps to
vector signed char	vector signed char	vspltisb z,0 vsububm t,z,a vmaxsb d,a,t

Figure 4-5. Absolute Value of Sixteen Integer Elements (8-bit)

3、vavgsh

有符号半字向量（16 位）求平均值

Vector Average Signed Halfword VX-form

vavgsh VRT,VRA,VRB

4	VRT	VRA	VRB	1346
0	6	11	16	21 … 31

```
do i=0 to 127 by 16
    aop ← EXTS((VRA)$_{i:i+15}$)
    bop ← EXTS((VRB)$_{i:i+15}$)
    VRT$_{i:i+15}$ ← Chop(( aop $+_{int}$ bop $+_{int}$ 1 ) >> 1, 16)
```

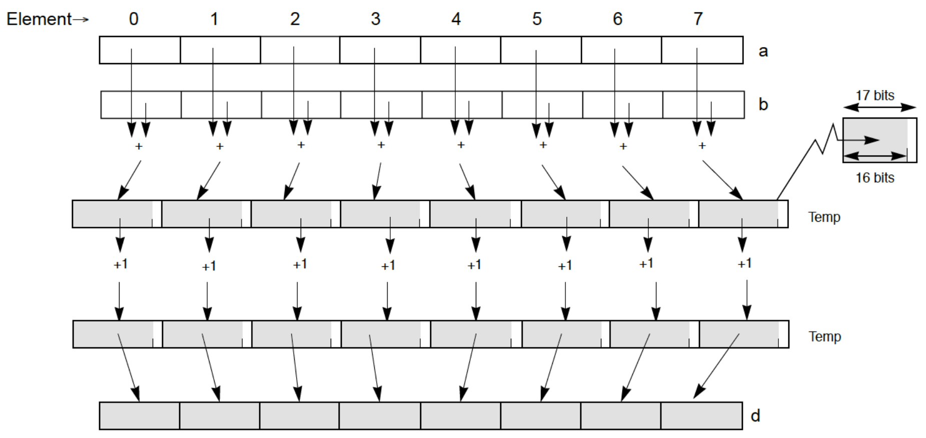
For each vector element *i* from 0 to 7, do the following.

Signed-integer halfword element *i* in VRA is added to signed-integer halfword element *i* in VRB. The sum is incremented by 1 and then shifted right 1 bit.

The low-order 16 bits of the result are placed into halfword element *i* of VRT.

Special Registers Altered:

None



d	a	b	maps to
vector unsigned short	vector unsigned short	vector unsigned short	vavguh d,a,b
vector signed short	vector signed short	vector signed short	vavgsh d,a,b

Figure 4-23. Average Eight Integer Elements (16-bit)

4、vcmpequh

无符号半字向量（16 位）比较是否相等

Vector Compare Equal To Unsigned Halfword *VC-form*

vcmpequh VRT,VRA,VRB (Rc=0)
vcmpequh. VRT,VRA,VRB (Rc=1)

4	VRT	VRA	VRB	Rc	70
0	6	11	16	21	22 31

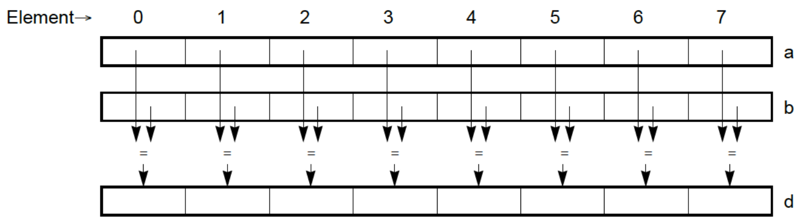
```
do i=0 to 127 by 16
   $VRT_{i:i+15} \leftarrow ((VRA)_{i:i+15} =_{int} (VRB)_{i:i+15})$ ? $^{16}1$ : $^{16}0$
if Rc=1 then do
   t ← $(VRT=^{128}1)$
   f ← $(VRT=^{128}0)$
   CR6 ← t || 0b0 || f || 0b0
```

For each vector element *i* from 0 to 7, do the following.

Unsigned-integer halfword element *i* in VRA is compared to unsigned-integer halfword element element *i* in VRB. Halfword element *i* in VRT is set to all 1s if unsigned-integer halfword element *i* in VRA is equal to unsigned-integer halfword element *i* in VRB, and is set to all 0s otherwise.

Special Registers Altered:

CR6 (if Rc=1)



d	a	b	maps to
vector bool short	vector unsigned short	vector unsigned short	vcmpequh d,a,b
	vector signed short	vector signed short	

Figure 4-28. Compare Equal of Eight Integer Elements (16-Bit)

5、vslb

字节向量（8 位）左移

Vector Shift Left Byte *VX-form*

vslb VRT,VRA,VRB

4	VRT	VRA	VRB	260
0	6	11	16	21 31

```
do i=0 to 127 by 8
    sh ← (VRB)$_{i+5:i+7}$
    VRT$_{i:i+7}$ ← (VRA)$_{i:i+7}$ << sh
```

For each vector element *i* from 0 to 15, do the following.

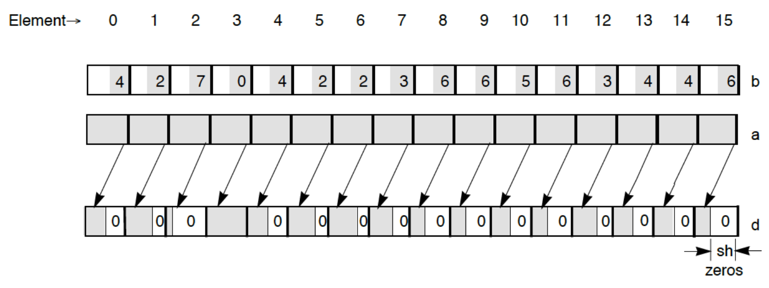
Byte element *i* in VRA is shifted left by the number of bits specified in the low-order 3 bits of byte element *i* in VRB.

- Bits shifted out of bit 0 are lost.
- Zeros are supplied to the vacated bits on the right.

The result is placed into byte element *i* of VRT.

Special Registers Altered:

None



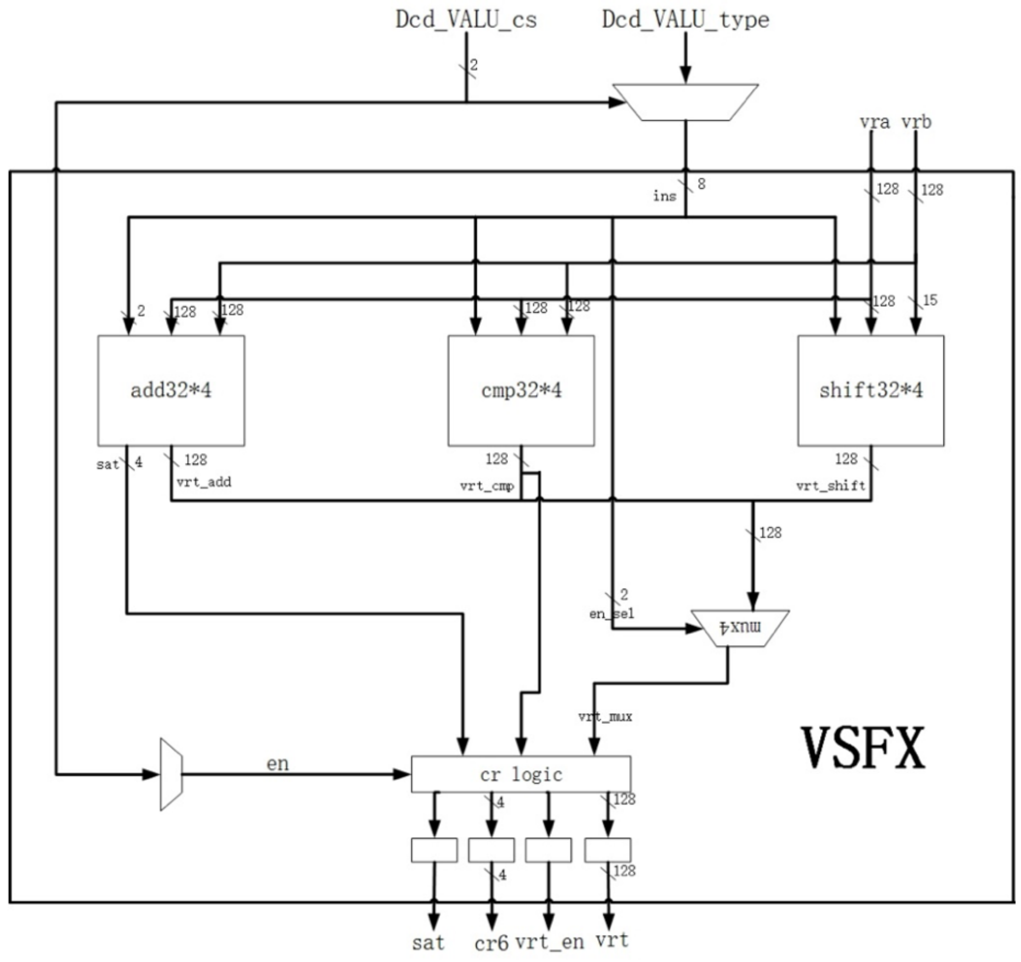
d	a	b	maps to
vector unsigned char	vector unsigned char	vector unsigned char	vslb d,a,b
vector signed char	vector signed char	vector unsigned char	

Figure 4-103. Shift Bits Left in Sixteen Integer Elements (8-Bit)

三、C++建模

```
#include "ppc_altivec.h"

signed int Clamp
(signed long long x, signed int y, signed int z){
    if(x<y) return y;
    if(x>z) return z;
    return x;
}

template <typename T>
T Chop(T x, T y){
    return x & ((1<<y)-1);
}

void vaddsws (ppc_avr_t &r, ppc_avr_t &a, ppc_avr_t &b){
    for(int i=0;i<4;i++) {
        r.s32[i] = Clamp((long long)a.s32[i] +
            (long long)b.s32[i], -(1<<31), (1<<31)-1);
    }
}
void vsububm (ppc_avr_t &r, ppc_avr_t &a, ppc_avr_t &b){
    for(int i=0;i<16;i++) {
        r.u8[i] =
        (unsigned char)Chop(a.u8[i] + (~b.u8[i]) + 1, 8);
    }
}
void vavgsh  (ppc_avr_t &r, ppc_avr_t &a, ppc_avr_t &b){
    for(int i=0;i<8;i++) {
        r.s16[i] =
        (signed short)Chop((a.s16[i] + b.s16[i] + 1)>>1, 16);
    }
}
void vcmpequh(ppc_avr_t &r, ppc_avr_t &a, ppc_avr_t &b){
    for(int i=0;i<8;i++) {
        r.u16[i] =
        (unsigned short)(a.u16[i] == b.u16[i] ? 0xffff : 0x0);
    }
}
void vslb    (ppc_avr_t &r, ppc_avr_t &a, ppc_avr_t &b){
    for(int i=0;i<16;i++) {
        int sh = b.s8[i]&0x7;
        r.s8[i] = a.s8[i] << sh;
    }
}
```

四、硬件架构设计


Dcd_VALU_cs
Dcd_VALU_type
2
vra
vrb
ins
8
128
128
2
128
128
128
128
128
15
add32*4
cmp32*4
shift32*4
sat
4
128
vrt_add
128
vrt_cmp
128
vrt_shift
128
2
en_sel
mux4
vrt_mux
VSFX
en
cr logic
4
128
4
128
sat
cr6
vrt_en
vrt

五、模块级 RTL 代码

1、vaddsws

```
module vaddsws( vra, vrb, vrt, sat );

input  [31 : 0] vra;
input  [31 : 0] vrb;
output [31 : 0] vrt;
output          sat;

wire   [31 : 0] sum;
wire   [31 : 0] vrt;
wire            sat;

assign {sat, sum}    = vra + vrb;
assign vrt = (vra[31]^vrb[31]) ? sum : (vra[31]&&vrb[31] ?
                        (sum[31] ? sum : 32'h80000000) :
                        (sum[31] ? 32'h7fffffff : sum));

endmodule
```

2、vsububm

```
module vsububm( vra, vrb, vrt );

input  [31 : 0] vra;
input  [31 : 0] vrb;
output [31 : 0] vrt;

wire   [31 : 0] vrt;
wire   [ 7 : 0] op0;
wire   [ 7 : 0] op1;
wire   [ 7 : 0] op2;
wire   [ 7 : 0] op3;

assign op0    =     (vra[ 7: 0] + ~vrb[ 7: 0] + 1) & 8'hff;
assign op1    =     (vra[15: 8] + ~vrb[15: 8] + 1) & 8'hff;
assign op2    =     (vra[23:16] + ~vrb[23:16] + 1) & 8'hff;
assign op3    =     (vra[31:24] + ~vrb[31:24] + 1) & 8'hff;
assign vrt    =     {op3, op2, op1, op0};

endmodule
```

3、vavgsh

```
module vavgsh( vra, vrb, vrt );

input  [31 : 0] vra;
input  [31 : 0] vrb;
output [31 : 0] vrt;

wire   [31 : 0] vrt;
wire   [15 : 0] op0;
wire   [15 : 0] op1;

assign op0    =     ((vra[15: 0] + vrb[15: 0] + 1) >> 1);
assign op1    =     ((vra[31:16] + vrb[31:16] + 1) >> 1);
assign vrt    =
     {(vra[31]^vrb[31] ? ~op1[15] : op1[15]), op1[14:0],
      (vra[15]^vrb[15] ? ~op0[15] : op0[15]), op0[14:0]};

endmodule
```

4、vcmpequh

```
module vcmpequh(vra, vrb, vrt);

input  [31 : 0] vra;
input  [31 : 0] vrb;
output [31 : 0] vrt;

wire   [31 : 0] vrt;

assign vrt[15 :  0] = ( vra[15: 0] == vrb[15: 0] ) ?
                        16'hffff : 16'h0000;
assign vrt[31 : 16] = ( vra[31:16] == vrb[31:16] ) ?
                        16'hffff : 16'h0000;

endmodule
```

5、vslb

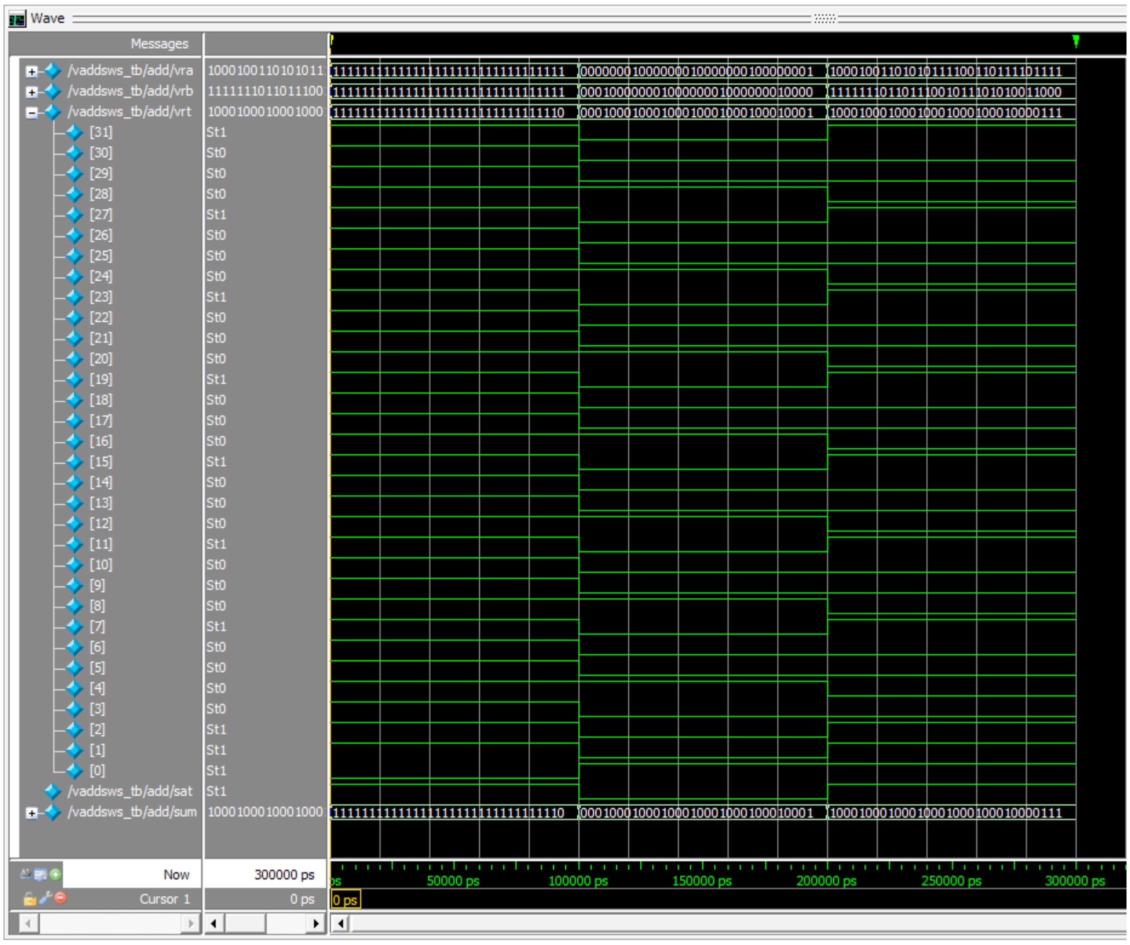
```
module vslb( vra, vrb, vrt );

input       [31: 0] vra;
input       [31: 0] vrb;
output      [31: 0] vrt;

wire        [31: 0] vrt;
wire        [15: 0] res0;
wire        [15: 0] res1;
wire        [15: 0] res2;
wire        [15: 0] res3;
wire        [31: 0] sh;

assign sh    =    vrb & 32'b00000111000001110000011100000111;
assign res0    =    vra[ 7: 0] << sh[ 7: 0];
assign res1    =    vra[15: 8] << sh[15: 8];
assign res2    =    vra[23:16] << sh[23:16];
assign res3    =    vra[31:24] << sh[31:24];
assign vrt     =    {res3[ 7: 0], res2[ 7: 0],
                     res1[ 7: 0], res0[ 7: 0]};

endmodule
```

六、仿真验证

1、vaddsws

测试向量：

```
vra: 32'hffffffff;
     32'h01010101;
     32'h89abcdef;
vrb: 32'hffffffff;
     32'h10101010;
     32'hfedcba98;
```

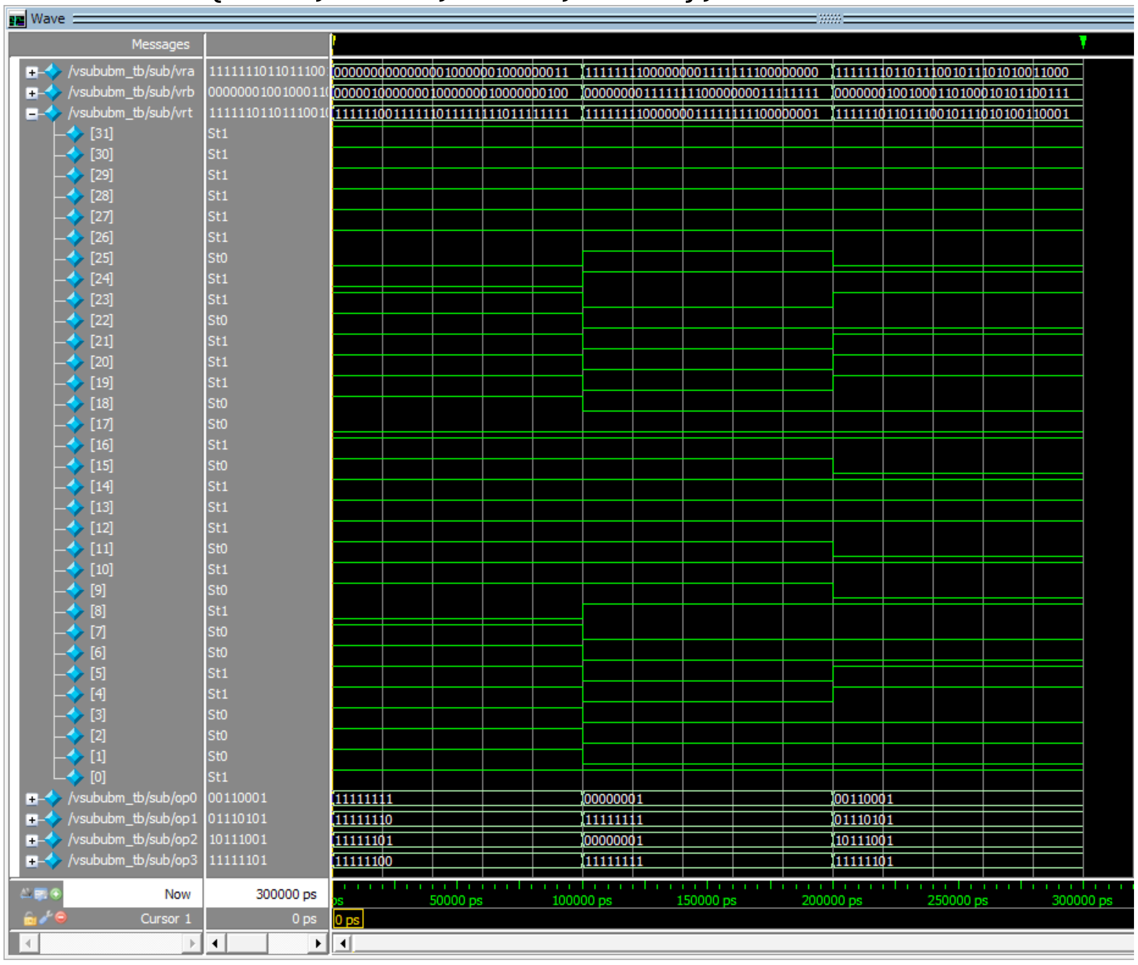


```
Wave
Messages
/vaddsws_tb/add/vra   1000100110101011  11111111111111111111111111111111 | 00000001000000010000000100000001 | 10001001101010111100110111101111
/vaddsws_tb/add/vrb   1111111011011100  11111111111111111111111111111111 | 00010000000100000001000000010000 | 11111110110111001011101010011000
/vaddsws_tb/add/vrt   1000100010001000  11111111111111111111111111111110 | 00010001000100010001000100010001 | 10001000100010001000100010000111
  [31]  St1
  [30]  St0
  [29]  St0
  [28]  St0
  [27]  St1
  [26]  St0
  [25]  St0
  [24]  St0
  [23]  St1
  [22]  St0
  [21]  St0
  [20]  St0
  [19]  St1
  [18]  St0
  [17]  St0
  [16]  St0
  [15]  St1
  [14]  St0
  [13]  St0
  [12]  St0
  [11]  St1
  [10]  St0
  [9]   St0
  [8]   St0
  [7]   St1
  [6]   St0
  [5]   St0
  [4]   St0
  [3]   St0
  [2]   St1
  [1]   St1
  [0]   St1
/vaddsws_tb/add/sat   St1
/vaddsws_tb/add/sum   1000100010001000  11111111111111111111111111111110 | 00010001000100010001000100010001 | 10001000100010001000100010000111

Now       300000 ps    ps  50000 ps  100000 ps  150000 ps  200000 ps  250000 ps  300000 ps
Cursor 1  0 ps         0 ps
```

2、vsububm

测试向量：

```
vra: {8'h0, 8'h1, 8'h2, 8'h3};
     {8'hff, 8'h0, 8'hff, 8'h0};
     {8'hfe, 8'hdc, 8'hba, 8'h98};
vrb: {8'h4, 8'h4, 8'h4, 8'h4};
     {8'h0, 8'hff, 8'h0, 8'hff};
     {8'h01, 8'h23, 8'h45, 8'h67};
```

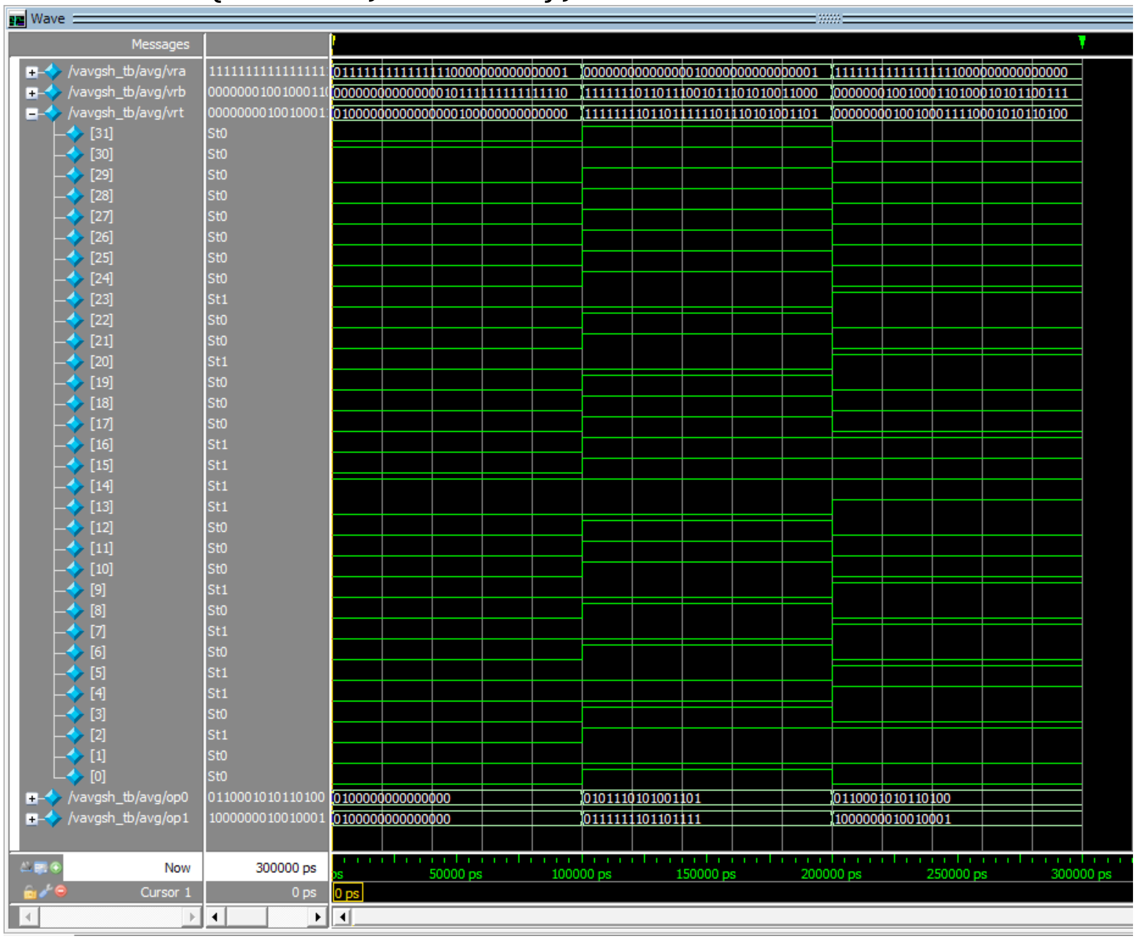


```
Wave
Messages
/vsububm_tb/sub/vra  111111101101110 0000000000000000100000001000000011  11111111000000001111111100000000  11111110110111001011101010011000
/vsububm_tb/sub/vrb  000000010010001 00000100000000100000001000000100   00000000111111110000000011111111  00000001001000110100010101100111
/vsububm_tb/sub/vrt  111111011011100 11111100111111011111111011111111   11111111000000011111111100000001  11111101101110010111010100110001
  [31] St1
  [30] St1
  [29] St1
  [28] St1
  [27] St1
  [26] St1
  [25] St0
  [24] St1
  [23] St1
  [22] St0
  [21] St1
  [20] St1
  [19] St1
  [18] St0
  [17] St0
  [16] St1
  [15] St0
  [14] St1
  [13] St1
  [12] St1
  [11] St0
  [10] St1
  [9]  St0
  [8]  St1
  [7]  St0
  [6]  St0
  [5]  St1
  [4]  St1
  [3]  St0
  [2]  St0
  [1]  St0
  [0]  St1
/vsububm_tb/sub/op0  00110001  11111111  00000001  00110001
/vsububm_tb/sub/op1  01110101  11111110  11111111  01110101
/vsububm_tb/sub/op2  10111001  11111101  00000001  10111001
/vsububm_tb/sub/op3  11111101  11111100  11111111  11111101
Now       300000 ps   ps  50000 ps  100000 ps  150000 ps  200000 ps  250000 ps  300000 ps
Cursor 1  0 ps        0 ps
```

3、vavgsh

测试向量：

```
vra: {16'h7fff, 16'h0001};
     {16'h0001, 16'h0001};
     {16'hffff, 16'h8000};
vrb: {16'h0001, 16'h7ffe};
     {16'hfedc, 16'hba98};
     {16'h0123, 16'h4567};
```

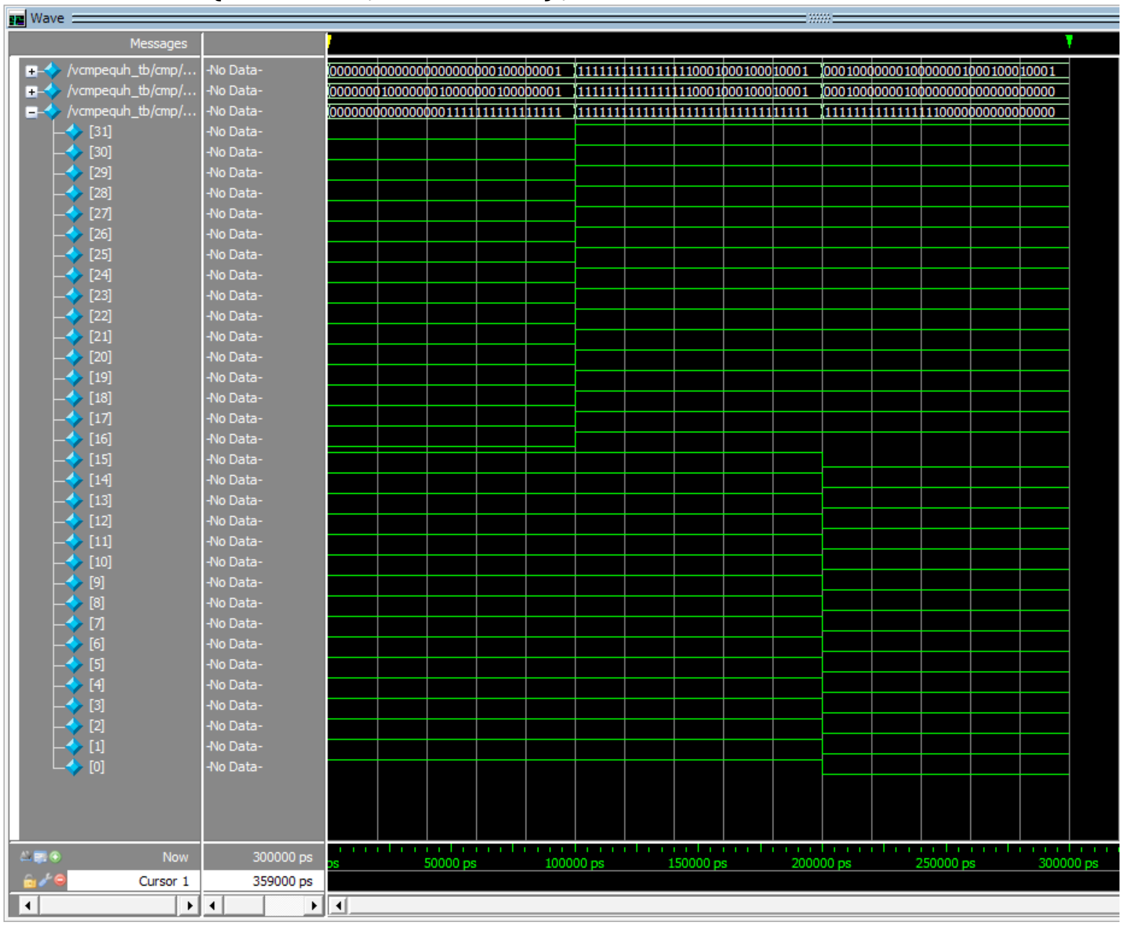


```
Wave
Messages
/vavgsh_tb/avg/vra   1111111111111111  01111111111111111000000000000001  00000000000000010000000000000001  11111111111111111000000000000000
/vavgsh_tb/avg/vrb   0000000100100011  00000000000000010111111111111110  11111110110111001011101010011000  00000001001000110100010101100111
/vavgsh_tb/avg/vrt   0000000010010001  01000000000000000100000000000000  11111111011011111110111010101101  00000000100100011110001010110100
  [31]  St0
  [30]  St0
  [29]  St0
  [28]  St0
  [27]  St0
  [26]  St0
  [25]  St0
  [24]  St0
  [23]  St1
  [22]  St0
  [21]  St0
  [20]  St1
  [19]  St0
  [18]  St0
  [17]  St0
  [16]  St1
  [15]  St1
  [14]  St1
  [13]  St1
  [12]  St0
  [11]  St0
  [10]  St0
  [9]   St1
  [8]   St0
  [7]   St1
  [6]   St0
  [5]   St1
  [4]   St1
  [3]   St0
  [2]   St1
  [1]   St0
  [0]   St0
/vavgsh_tb/avg/op0   0110001010110100  0100000000000000  0101110101001101  0110001010110100
/vavgsh_tb/avg/op1   1000000010010001  0100000000000000  0111111101101111  1000000010010001

Now       300000 ps   ps  50000 ps  100000 ps  150000 ps  200000 ps  250000 ps  300000 ps
Cursor 1  0 ps        0 ps
```

4、vcmpequh

测试向量：

```
vra: {16'h0000, 16'h0101};
     {16'hffff, 16'h1111};
     {16'h1010, 16'h1111};
vrb: {16'h0101, 16'h0101};
     {16'hffff, 16'h1111};
     {16'h1010, 16'h0000};
```

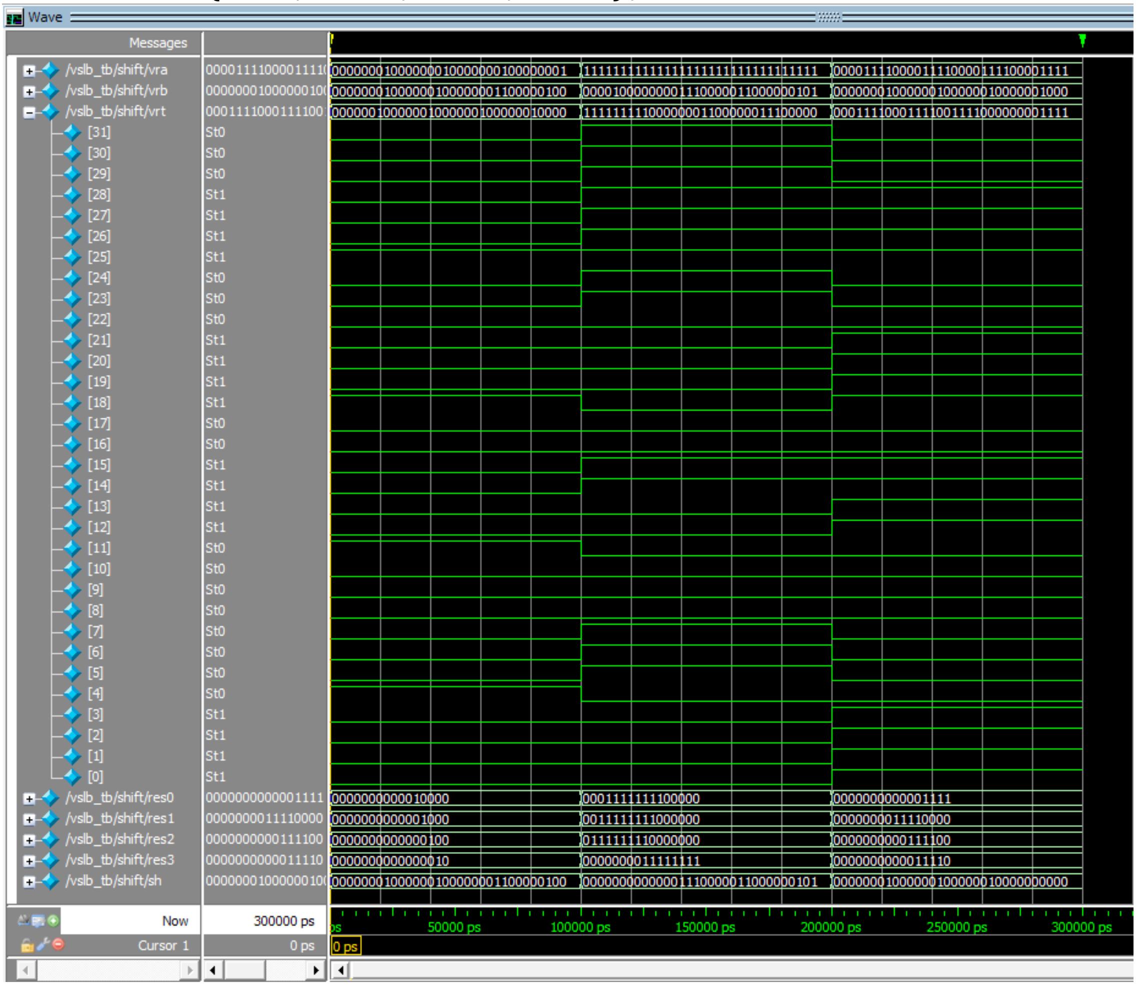


```
/vcmpequh_tb/cmp/...  -No Data-  000000000000000000000000100000001 | 11111111111111111000100010001 | 0001000000010000000100010001
/vcmpequh_tb/cmp/...  -No Data-  000000010000000100000001000000001 | 11111111111111111000100010001 | 000100000001000000000000000000
/vcmpequh_tb/cmp/...  -No Data-  0000000000000000011111111111111 | 1111111111111111111111111111111 | 111111111111111000000000000000
[31] ... [0]          -No Data-
Now       300000 ps
Cursor 1  359000 ps
ps  50000 ps  100000 ps  150000 ps  200000 ps  250000 ps  300000 ps
```

5、vslb

测试向量：

```
vra: {8'h1, 8'h1, 8'h1, 8'h1};
     {8'hff, 8'hff, 8'hff, 8'hff};
     {8'hf, 8'hf, 8'hf, 8'hf};
vrb: {8'h1, 8'h2, 8'h3, 8'h4};
     {8'h8, 8'h7, 8'h6, 8'h5};
     {8'h1, 8'h2, 8'h4, 8'h8};
```



```
Wave
Messages
/vslb_tb/shift/vra    0000111100001111(   0000000100000001000000010000000 1   11111111111111111111111111111111   00001111000011110000111100001111
/vslb_tb/shift/vrb    0000000100000010(   00000001000000100000001100000100   00001000000001110000011000000101   00000001000000100000010000001000
/vslb_tb/shift/vrt    000111100011110O    000000100000010000001000000 10000  11111111100000000110000001 1100000   00011110001111001111000000001111
  [31] St0
  [30] St0
  [29] St0
  [28] St1
  [27] St1
  [26] St1
  [25] St1
  [24] St0
  [23] St0
  [22] St0
  [21] St1
  [20] St1
  [19] St1
  [18] St1
  [17] St0
  [16] St0
  [15] St1
  [14] St1
  [13] St1
  [12] St1
  [11] St0
  [10] St0
  [9] St0
  [8] St0
  [7] St0
  [6] St0
  [5] St0
  [4] St0
  [3] St1
  [2] St1
  [1] St1
  [0] St1
/vslb_tb/shift/res0   0000000000001111    0000000000010000   0001111111100000   0000000000001111
/vslb_tb/shift/res1   0000000011110000    0000000000001000   0011111111000000   0000000011110000
/vslb_tb/shift/res2   0000000000111100    0000000000000100   0111111110000000   0000000000111100
/vslb_tb/shift/res3   0000000000011110    0000000000000010   0000000011111111   0000000000011110
/vslb_tb/shift/sh     0000000100000010(   00000001000000100000001100000100   00000000000001110000011000000101   00000001000000100000010000000000

Now       300000 ps   ps  50000 ps  100000 ps  150000 ps  200000 ps  250000 ps  300000 ps
Cursor 1  0 ps        0 ps
```

七、顶层模块集成

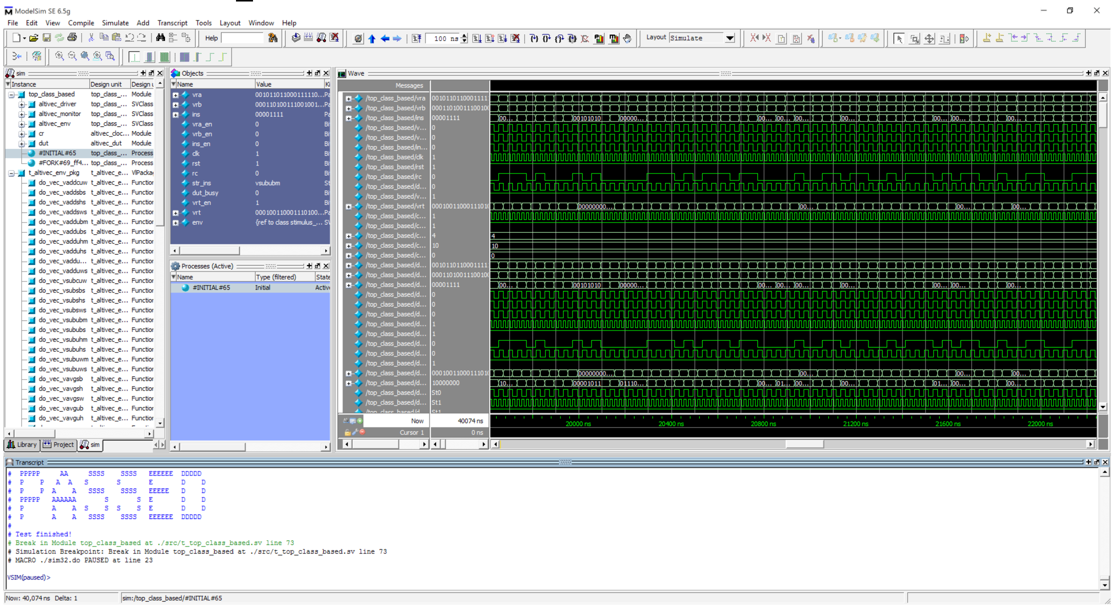
```
module vsfx_top(clk, en, vra, vrb, ins, vrt_en, vrt, sat, cr6);

input            clk;
input            en;
input   [127 : 0] vra;
input   [127 : 0] vrb;
input   [  7 : 0] ins;
output           vrt_en;
output  [127 : 0] vrt;
output           sat;
output  [  3 : 0] cr6;

reg      [127 : 0] vrt;
reg                sat;
reg      [  3:   0] cr6;
reg                vrt_en;
wire     [127 : 0] add;
wire     [127 : 0] sub;
wire     [127 : 0] avg;
wire     [127 : 0] cmp;
wire     [127 : 0] shift;
wire     sat0, sat1, sat2, sat3;

vaddsws add0( .vra(vra[ 31:  0]), .vrb(vrb[ 31:  0]),
              .vrt(add[ 31:  0]), .sat(sat0) );
vaddsws add1( .vra(vra[ 63: 32]), .vrb(vrb[ 63: 32]),
              .vrt(add[ 63: 32]), .sat(sat1) );
vaddsws add2( .vra(vra[ 95: 64]), .vrb(vrb[ 95: 64]),
              .vrt(add[ 95: 64]), .sat(sat2) );
```

```
vaddsws add3( .vra(vra[127: 96]), .vrb(vrb[127: 96]),
              .vrt(add[127: 96]), .sat(sat3) );

vsububm sub0( .vra(vra[ 31:  0]), .vrb(vrb[ 31:  0]),
              .vrt(sub[ 31:  0]) );
vsububm sub1( .vra(vra[ 63: 32]), .vrb(vrb[ 63: 32]),
              .vrt(sub[ 63: 32]) );
vsububm sub2( .vra(vra[ 95: 64]), .vrb(vrb[ 95: 64]),
              .vrt(sub[ 95: 64]) );
vsububm sub3( .vra(vra[127: 96]), .vrb(vrb[127: 96]),
              .vrt(sub[127: 96]) );

vavgsh avg0( .vra(vra[ 31:  0]), .vrb(vrb[ 31:  0]),
             .vrt(avg[ 31:  0]) );
vavgsh avg1( .vra(vra[ 63: 32]), .vrb(vrb[ 63: 32]),
             .vrt(avg[ 63: 32]) );
vavgsh avg2( .vra(vra[ 95: 64]), .vrb(vrb[ 95: 64]),
             .vrt(avg[ 95: 64]) );
vavgsh avg3( .vra(vra[127: 96]), .vrb(vrb[127: 96]),
             .vrt(avg[127: 96]) );

vcmpequh cmp0( .vra(vra[ 31:  0]), .vrb(vrb[ 31:  0]),
               .vrt(cmp[ 31:  0]) );
vcmpequh cmp1( .vra(vra[ 63: 32]), .vrb(vrb[ 63: 32]),
               .vrt(cmp[ 63: 32]) );
vcmpequh cmp2( .vra(vra[ 95: 64]), .vrb(vrb[ 95: 64]),
               .vrt(cmp[ 95: 64]) );
vcmpequh cmp3( .vra(vra[127: 96]), .vrb(vrb[127: 96]),
               .vrt(cmp[127: 96]) );
```

```

vslb shift0( .vra(vra[ 31:  0]), .vrb(vrb[ 31:  0]),
             .vrt(shift[ 31:  0]) );
vslb shift1( .vra(vra[ 63: 32]), .vrb(vrb[ 63: 32]),
             .vrt(shift[ 63: 32]) );
vslb shift2( .vra(vra[ 95: 64]), .vrb(vrb[ 95: 64]),
             .vrt(shift[ 95: 64]) );
vslb shift3( .vra(vra[127: 96]), .vrb(vrb[127: 96]),
             .vrt(shift[127: 96]) );

always @(posedge clk)
begin
         if (ins == 8'b01110000) //vaddsws
    begin
        vrt <= add;
        sat <= sat0 | sat1 | sat2 | sat3;
    end
    else if (ins == 8'b10000000) //vsububm
        vrt <= sub;
    else if (ins == 8'b10101001) //vavgsh
        vrt <= avg;
    else if (ins == 8'b00001011) //vcmpequh
        vrt <= cmp;
    else if (ins == 8'b00100010) //vslb
        vrt <= shift;

    vrt_en <= en;
end

endmodule
```

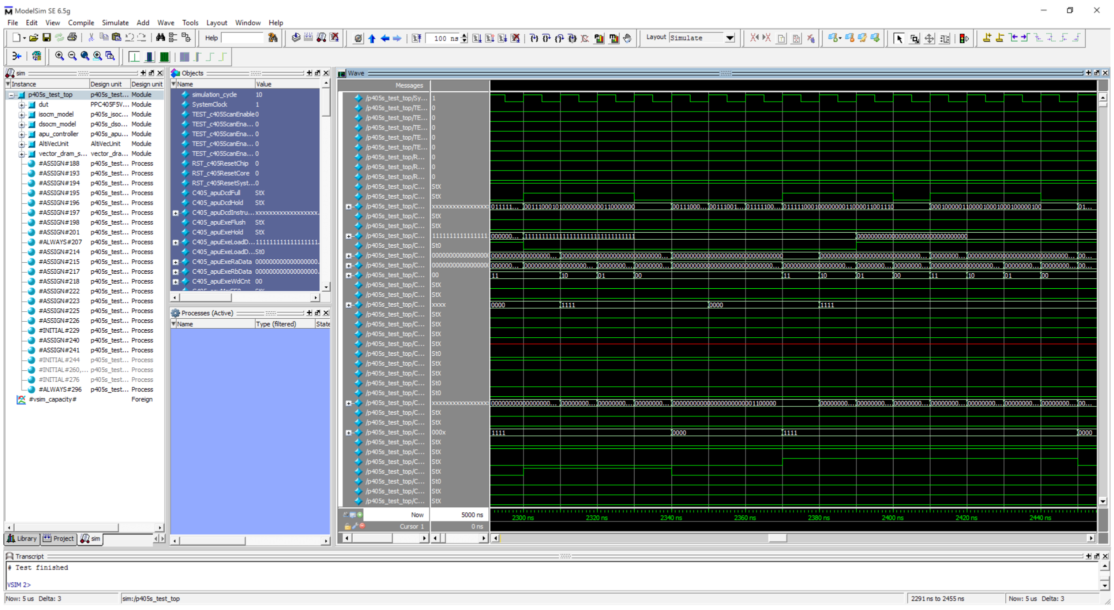
八、顶层仿真

运行./module_verification/sim32.bat 结果如下：



```
#  PPPPP    AA    SSSS   SSSS   EEEEEE  DDDDD
#  P    P  A  A  S      S      E       D    D
#  P    P A    A  SSSS   SSSS   EEEEE   D    D
#  PPPPP  AAAAAA      S      S E       D    D
#  P      A    A S    S S    S E       D    D
#  P      A    A  SSSS   SSSS   EEEEEE  DDDDD
#
# Test finished!
# Break in Module top_class_based at ./src/t_top_class_based.sv line 73
# Simulation Breakpoint: Break in Module top_class_based at ./src/t_top_class_based.sv line 73
# MACRO ./sim32.do PAUSED at line 23

VSIM(paused)>
```

运行./sim/sim.bat 结果如下：



```
# Test finished

VSIM 2>
```

对比./sim/out_result 中的两个文件结果一致，说明模块功能正确。