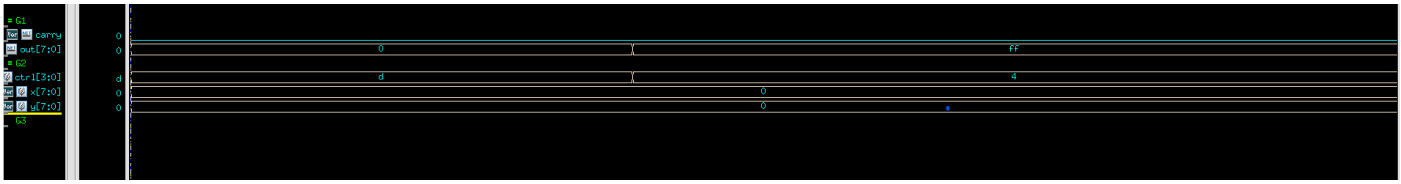
DSD_HW1_Report(b09901036 shi-lun chiu)

1.ALU

(1.) ALU_assign_tb : (screenshot of testing and nWave)

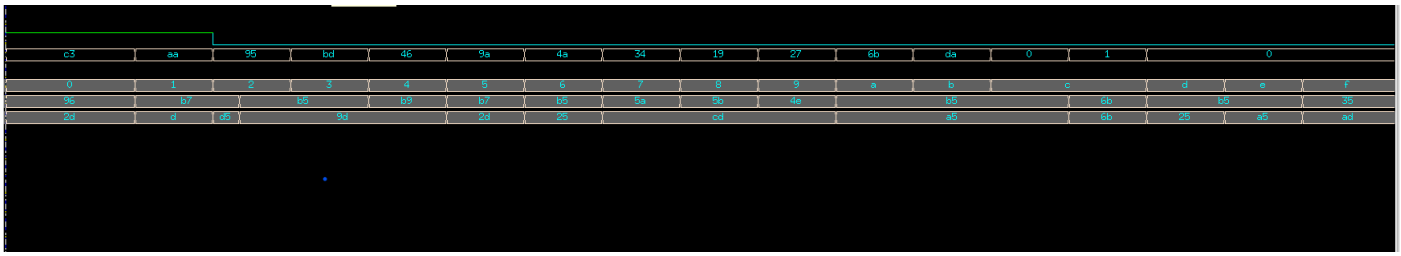
```
ash the programs that are using this file.
*Verdi* : Create FSDB file 'alu_assign.fsdb'
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
PASS --- 0100 boolean not
$finish called from file "alu_assign_tb.v", line 40.
$finish at simulation time                2500
           V C S   S i m u l a t i o n   R e p o r t
Time: 25000 ps
CPU Time:      0.700 seconds;       Data structure size:   0.0Mb
Tue Mar 28 14:59:45 2023
CPU time: .538 seconds to compile + .614 seconds to elab + .435 seconds to link + .7
45 seconds in simulation
```



(2.) ALU_assign_tb2 : (screenshot of testing and nWave)

```
alu_assign_test
0.ADD (signed)
[out = x + y]
    .... passed.
1.SUB (signed)
[out = x + y]
    .... passed.
2.BITAND
[out = and(x, y)]
    .... passed.
3.BITOR
[out = OR(x, y)]
    .... passed.
4.BITNOT
[out = ~x]
    .... passed.
5.BITXOR
[out = xor(x,y)]
    .... passed.
6.BITNOR
[out = nor(x , y)]
    .... passed.
7.SLLV
[out = y << x[2:0]
    .... passed.
8.SLRV
[out = y >> x[2:0]]
    .... passed.
```

```
    .... passed.
8.SLRV
[out = y >> x[2:0]]
    .... passed.
9.SRA
[out={x[7],x[7:1]}]
    .... passed.
10.RL
[out = {x[6:0] , x[7]}]
    .... passed.
11.RR
[out = {x[0] , x[7:1]]
    .... passed.
12.EQ
[out = (x = y )? 8'd1:8'd0]
    .... passed.
12_2.EQ
[out = (x = y )? 8'd1:8'd0]
    .... passed.
13.NOP
[out = 8'd0]
    .... passed.
14.NOP
[out = 8'd0]
    .... passed.
15.NOP
[out = 8'd0]
    .... passed.
****************************
**                        **
**   Congratulations !!   **
** All Patterns Passed!!  **
**                        **
****************************
```



Ways of testing :

0.

```
          err_count   =0;
        x           =8'b10010110;
        y           =8'b00101101;
        ctrl=4'd0;
```

1.

```
           x           =8'b10110111;
        y           =8'b00001101;
        ctrl=4'd1;
```

2.

```
         x           =8'b10110111;
        y           =8'b11010101;
        ctrl=4'd2;
```

3.

```
         x           =8'b10110101;
        y           =8'b10011101;
        ctrl=4'd3;
```

4.

```
 x           =8'b10111001;


        ctrl=4'd4;
```

5.

```
x           =8'b10110111;
        y           =8'b00101101;
        ctrl=4'd5;
```

6.

```
x           =8'b10110101;
        y           =8'b00100101;
        ctrl=4'd6;
```

7.

```
 x           =8'b01011010;
        y           =8'b11001101;
        ctrl=4'd7;
```

8.

```
 x           =8'b01011011;
        y           =8'b11001101;
        ctrl=4'd8;
```

9.

```
 ctrl=4'd9;
        x           =8'b01001110;
        y           =8'b11001101;
```

10.

```
x          =8'b10110101; //
       y          =8'b10100101;
       ctrl=4'd10;
```

11.

```
x          =8'b10110101; //
       y          =8'b10100101;
       ctrl=4'd11;
```

12.

```
x          =8'b10110101;
       y          =8'b10100101;
       ctrl=4'd12;
```

and

```
 x          =8'b01101011;
       y          =8'b01101011;
       ctrl=4'd12;
```

13.

```
 x          =8'b10110101;
       y          =8'b00100101;
       ctrl=4'd13;
```

14.

```
  x          =8'b10110101;
       y          =8'b10100101;
       ctrl=4'd14;
```

15.

```
 x          =8'b00110101;
       y          =8'b10101101;
       ctrl=4'd15;
```

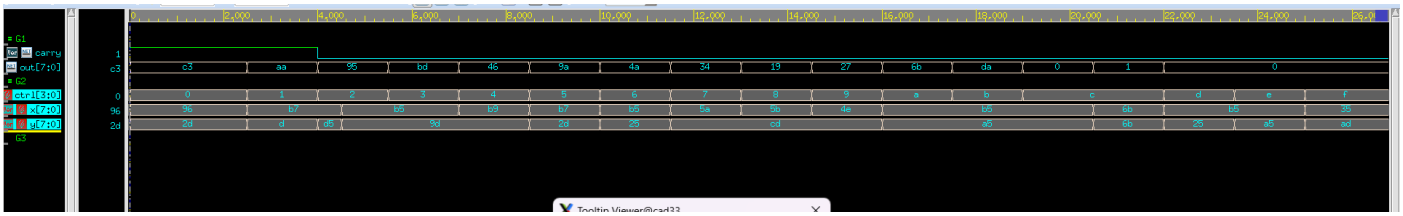
(3.) ALU_always_tb : (screenshot of testing and nWave)

```
$finish called from file "alu_always_tb.v", line 40.
$finish at simulation time                2500
           V C S   S i m u l a t i o n   R e p o r t
Time: 25000 ps
CPU Time:      0.740 seconds;       Data structure size:   0.0Mb
Tue Mar 28 15:11:06 2023
CPU time: .547 seconds to compile + .435 seconds to elab + .575 seconds to link +
84 seconds in simulation
```

(4.) ALU_always_tb2 : (screenshot of testing and nWave)

```
alu_assign_test
0.ADD (signed)
[out = x + y]
    .... passed.
1.SUB (signed)
[out = x + y]
    .... passed.
2.BITAND
[out = and(x, y)]
    .... passed.
3.BITOR
[out = OR(x, y)]
    .... passed.
4.BITNOT
[out = ~x]
    .... passed.
5.BITXOR
[out = xor(x,y)]
    .... passed.
6.BITNOR
[out = nor(x , y)]
    .... passed.
7.SLLV
[out = y << x[2:0]
    .... passed.
8.SLRV
[out = y >> x[2:0]]
    .... passed.
9.SRA
[out={x[7],x[7:1]}]
    .... passed.
10.RL
[out = {x[6:0] , x[7]}]
    .... passed.
```

```
8.SLRV
[out = y >> x[2:0]]
    .... passed.
9.SRA
[out={x[7],x[7:1]}]
    .... passed.
10.RL
[out = {x[6:0] , x[7]}]
    .... passed.
11.RR
[out = {x[0] , x[7:1]]
    .... passed.
12.EQ
[out = (x = y )? 8'd1:8'd0]
    .... passed.
12_2.EQ
[out = (x = y )? 8'd1:8'd0]
    .... passed.
13.NOP
[out = 8'd0]
    .... passed.
14.NOP
[out = 8'd0]
    .... passed.
15.NOP
[out = 8'd0]
    .... passed.
**************************       /|__/|
**                      **      / 0,0  |
**   Congratulations !!   **    /_____   |
** All Patterns Passed!!  **   /^ ^ ^ \  |
**                      **  |^ ^ ^ ^ |w|
**************************   \m__m__|_|
$finish called from file "alu_always_tb2.v", line 321.
$finish at simulation time              26500
           V C S   S i m u l a t i o n   R e p o r t
Time: 265000 ps
CPU Time:      0.750 seconds;       Data structure size:   0.0Mb
Tue Mar 28 22:53:56 2023
CPU time: .591 seconds to compile + .430 seconds to elab + .642 seconds to link + .801 seconds in simulation
```



Ways of testing : same as ALU_assign_tb,the

0.

```
        err_count   =0;
      x           =8'b10010110;
      y           =8'b00101101;
      ctrl=4'd0;
```

1.

```
        x           =8'b10110111;
      y           =8'b00001101;
      ctrl=4'd1;
```

2.

```
       x           =8'b10110111;
      y           =8'b11010101;
      ctrl=4'd2;
```

3.

```
       x           =8'b10110101;
      y           =8'b10011101;
      ctrl=4'd3;
```

4.

```
x          =8'b10111001;


       ctrl=4'd4;
```

5.

```
x          =8'b10110111;
       y          =8'b00101101;
       ctrl=4'd5;
```

6.

```
x          =8'b10110101;
       y          =8'b00100101;
       ctrl=4'd6;
```

7.

```
x          =8'b01011010;
       y          =8'b11001101;
       ctrl=4'd7;
```

8.

```
x          =8'b01011011;
       y          =8'b11001101;
       ctrl=4'd8;
```

9.

```
ctrl=4'd9;
       x          =8'b01001110;
       y          =8'b11001101;
```

10.

```
x          =8'b10110101; //
       y          =8'b10100101;
       ctrl=4'd10;
```

11.

```
x          =8'b10110101; //
       y          =8'b10100101;
       ctrl=4'd11;
```

12.

```
x          =8'b10110101;
       y          =8'b10100101;
       ctrl=4'd12;
```

and

```
x          =8'b01101011;
       y          =8'b01101011;
       ctrl=4'd12;
```

13.

```
x          =8'b10110101;
```

```
        y           =8'b00100101;
        ctrl=4'd13;
```

14.

```
 x           =8'b10110101;
        y           =8'b10100101;
        ctrl=4'd14;
```

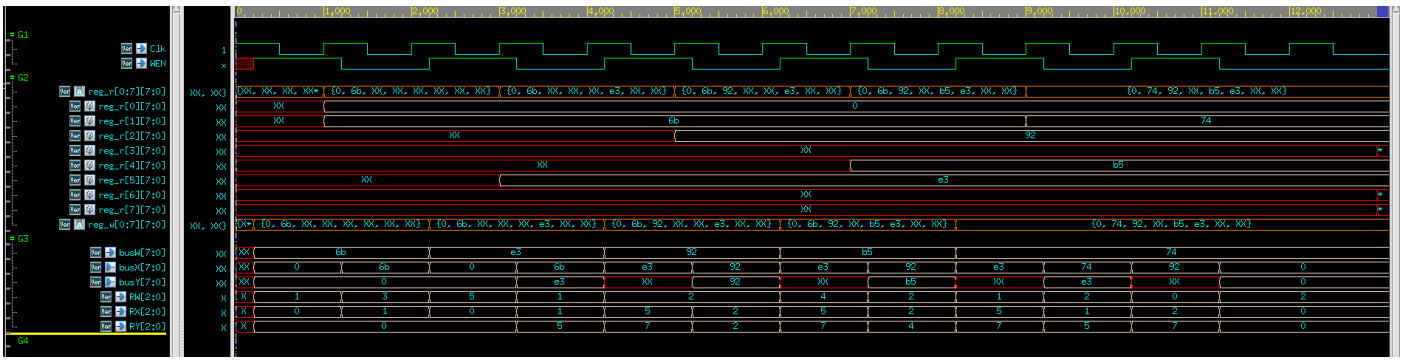
15.

```
 x           =8'b00110101;
        y           =8'b10101101;
        ctrl=4'd15;
```

2. 8 X 8 register file

(1.) screenshot of testing and nWave

```
 8 X 8 bits register file test
1: store  A=107 in REG#1
    .... passed.
2: store  B=227 in REG#5
    .... passed.
3: store  C=227 in REG#2
    .... passed.
4: store  D=181 in REG#4  and look back C=146 in REG#2
    .... passed.
5: restore multiplicand E=116 in REG#1
    .... passed.
6: test #reg0
    .... passed.
****************************          /|__/|
**                        **        / 0,0  |
**   Congratulations !!   **      /_____   |
** All Patterns Passed!!  **     /^ ^ ^ \  |
**                        **    |^ ^ ^ ^ |w|
****************************     \m___m__|_|
$finish called from file "register_file_tb.v", line 175.
$finish at simulation time                13000
           V C S   S i m u l a t i o n   R e p o r t
Time: 130000 ps
CPU Time:      0.720 seconds;       Data structure size:   0.0Mb
```



Ways of testing :

```
  `timescale 1ns/10ps
`define CYCLE  10
`define HCYCLE  5
```

```
module register_file_tb;
    // port declaration for design-under-test
    reg Clk, WEN;
    reg  [2:0] RW, RX, RY;
    reg  [7:0] busW;
    wire [7:0] busX, busY;

    // instantiate the design-under-test
    register_file rf(
        Clk  ,
        WEN  ,
        RW   ,
        busW ,
        RX   ,
        RY   ,
        busX ,
        busY
    );

    // write your test pattern here
     initial begin
       $fsdbDumpfile("register_file.fsdb");
       $fsdbDumpvars (0, register_file_tb , "+mda");

    end

     // clock generation
    always#(`HCYCLE) Clk = ~Clk;

    //test pattern
    parameter A ={4'd6,4'd11};
    parameter B ={4'd14,4'd3};
    parameter C ={4'd9,4'd2};
    parameter D ={4'd11,4'd5};
    parameter E ={4'd7,4'd4};

    //simulation count
    integer err_count;
    initial begin

        Clk       = 1'b1;
        err_count = 0;
```

```
        $display( " 8 X 8 bits register file test" );

        //------------------2 cycle testbench---------
        #(`CYCLE*0.2)
        $display( "1: store  A=%3d in REG#1 " , A );
        WEN = 1'b1 ; RW = 3'd1 ; busW = A ; RX = 3'd0 ;
        RY = 3'd0 ;
        #(`CYCLE*0.8)
        #(`CYCLE*0.2)
        WEN = 1'b0 ; RW = 3'd3 ;  RX = 3'd1 ;  RY = 3'd0 ;
        #(`CYCLE*0.3)
        if( busX==A ) $display( "    .... passed." );
        else begin
            err_count = err_count+1;
            $display( "    .... failed, design(%b) != expected(%b)", busX, A );
        end
        #(`HCYCLE)

        //------------------2 cycle testbench---------
        #(`CYCLE*0.2)
        $display( "2: store  B=%3d in REG#5 " , B );
        WEN = 1'b1 ; RW = 3'd5 ; busW = B ; RX = 3'd0 ;
        RY = 3'd0 ;
        #(`CYCLE*0.8)
        #(`CYCLE*0.2)
        WEN = 1'b0 ; RW = 3'd1 ;  RX = 3'd1 ;  RY = 3'd5 ;
        #(`CYCLE*0.3)
        if( busY==B ) $display( "    .... passed." );
        else begin
            err_count = err_count+1;
            $display( "    .... failed, design(%b) != expected(%b)", busX, A );
        end
        #(`HCYCLE)

        //------------------2 cycle testbench---------
        #(`CYCLE*0.2)
        $display( "3: store  C=%3d in REG#2 " , B );
        WEN = 1'b1 ; RW = 3'd2 ; busW = C ; RX = 3'd5 ;
        RY = 3'd7 ;
        #(`CYCLE*0.8)
        #(`CYCLE*0.2)
        WEN = 1'b0 ; RW = 3'd2 ;  RX = 3'd2 ;  RY = 3'd2 ;
```

```
        #(`CYCLE*0.3)
        if( busY==C & busX == C ) $display( "    .... passed." );
        else begin
            err_count = err_count+1;
            $display( "    .... failed, designX(%b) != expected(%b)", busX, C );
            $display( "    .... failed, designY(%b) != expected(%b)", busY, C );
        end
        #(`HCYCLE)

        //------------------2 cycle testbench---------
        #(`CYCLE*0.2)
        $display( "4: store  D=%3d in REG#4  and look back C=%3d in REG#2" , D ,C);
        WEN = 1'b1 ; RW = 3'd4 ; busW = D ; RX = 3'd5 ;
        RY = 3'd7 ;
        #(`CYCLE*0.8)
        #(`CYCLE*0.2)
        WEN = 1'b0 ; RW = 3'd2 ;  RX = 3'd2 ;  RY = 3'd4 ;
        #(`CYCLE*0.3)
        if(  busY == D && busX ==C ) $display( "    .... passed." );
        else begin
            err_count = err_count+1;
            $display( "    .... failed, designX(%b) != expected(%b)", busX, C );
            $display( "    .... failed, designY(%b) != expected(%b)", busY, D );
        end
        #(`HCYCLE)

         //------------------2 cycle testbench---------
        #(`CYCLE*0.2)
        $display( "5: restore multiplicand E=%3d in REG#1 " , E );
        WEN = 1'b1 ; RW = 3'd1 ; busW = E ; RX = 3'd5 ;
        RY = 3'd7 ;
        #(`CYCLE*0.8)
        #(`CYCLE*0.2)
        WEN = 1'b0 ; RW = 3'd2 ;  RX = 3'd1 ;  RY = 3'd5 ;
        #(`CYCLE*0.3)
        if( busX == E ) $display( "    .... passed." );
        else begin
            err_count = err_count+1;
            $display( "    .... failed, design(%b) != expected(%b)", busX, E);
        end
        #(`HCYCLE)
```

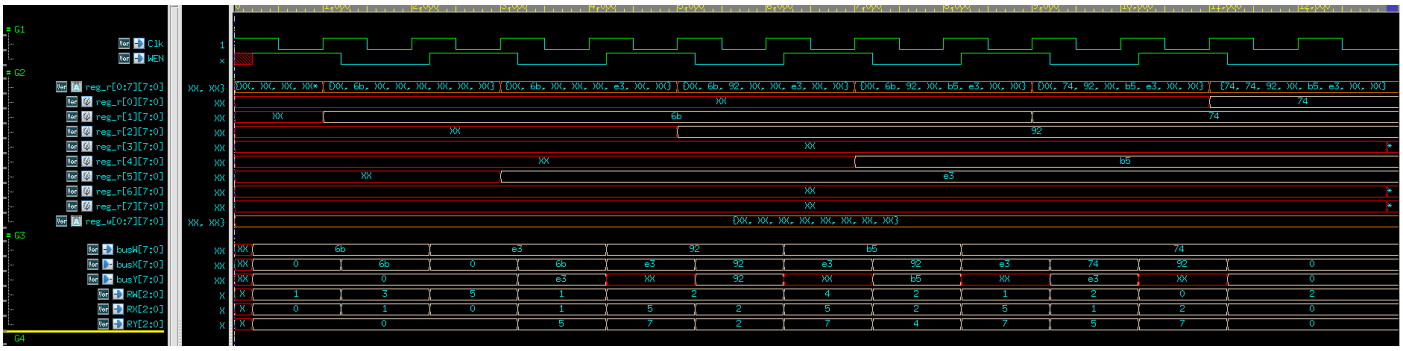
```
     //------------------2 cycle testbench---------
    #(`CYCLE*0.2)
    $display( "6: test #reg0 "  );
    WEN = 1'b1 ; RW = 3'd0 ; busW = E ; RX = 3'd2 ;
    RY = 3'd7 ;
    #(`CYCLE*0.8)
    #(`CYCLE*0.2)
    WEN = 1'b0 ; RW = 3'd2 ;  RX = 3'd0 ;  RY = 3'd0 ;
    #(`CYCLE*0.3)
    if( busX == 8'd0 ) $display( "    .... passed." );
    else begin
        err_count = err_count+1;
        $display( "    .... failed, design(%b) != expected(%b)", busX,8'd0 );
    end
    #(`HCYCLE)



if( err_count==0 ) begin
$display("**************************       /|__/|");
$display("**                      **      / O,O  |");
$display("**   Congratulations !!   **    /_____   |");
$display("** All Patterns Passed!!  **   /^ ^ ^ \\  |");
$display("**                      **  |^ ^ ^ ^ |w|");
$display("**************************   \\m___m__|_|");
end
else begin
$display("************************** ");
$display("    Failed ,and ...        ");
$display("............... . ..... /´ /)");
$display("............... ..,../¯ ..//");
$display("............... ..,/¯ ..//");
$display("............... ./... .//");
$display("............/´¯/' ...'/´¯`•¸");
$display(".........//.../... ./... ..../¨¯|.");
$display("........('(...´(... ..... ,~/'...')");
$display(".........|......... ..... ../..../");
$display(".........|.... ..... ..... _.•´");
$display("............|..... ..... ..(");
$display("..............|..... ..... ...|.");

$display("    Total %2d Errors ...     ", err_count );
$display("************************** ");
```

```
    end
     #(`CYCLE) $finish;
    end
endmodule
```

3. simple Caculator

(1)

```
6: set REG#6 = (B[3]==1)? 8'b11111111: 8'b00000000
    [REG#2 = sra REG#2]
    [REG#6 = and 0000_0001 REG#2]
    [REG#6 = sub REG#0 REG#6]
    .... passed.
7: shift & summation, REG#7 = A*B[0]
    [REG#3 = and REG#1 REG#3]
    [REG#7 = add REG#0 REG#3]
    .... passed.
8: shift & summation, REG#7 = A*B[1:0]
    [REG#4 = and REG#1 REG#4]
    [REG#4 = sll 0000_0001 REG#4]
    [REG#7 = add REG#7 REG#4]
    .... passed.
9: shift & summation, REG#7 = A*B[2:0]
    [REG#5 = and REG#1 REG#5]
    [REG#5 = sll 0000_0010 REG#5]
    [REG#7 = add REG#7 REG#5]
    .... passed.
                300
10:shift & summation, REG#7 = A*B
    [REG#6 = and REG#1 REG#6]
    [REG#6 = sll 0000_0011 REG#6]
    [REG#7 = add REG#7 REG#6]
    .... passed.
Calculation restuls: 13 * 12 =  156
****************************        /|__/|
**                        **      / 0,0  |
**   Congratulations !!   **     /______  |
** All Patterns Passed!!  **    /^ ^ ^ \  |
**                        **   |^ ^ ^ ^ |w|
****************************    \m__m__|_|
```



(2) use it given testbench

what I found

In ALU I know that case can be only use in always block ,and assign can only be used by wire announcement .

In register file I knew that in sequential circuit can use clock gating to prevent unwanted result

The last simple calculator is relatively simple if ALU and register files are finished.