Kevin Hoser and Alex Schendel

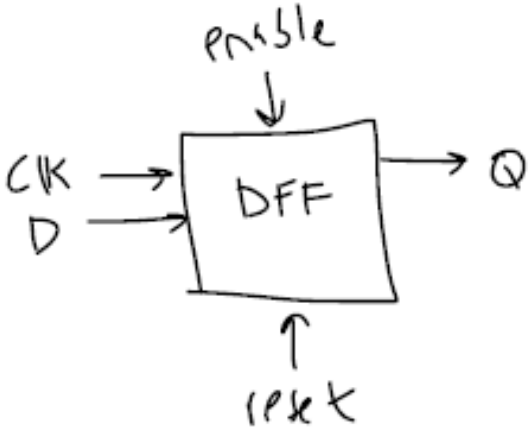
Project 3

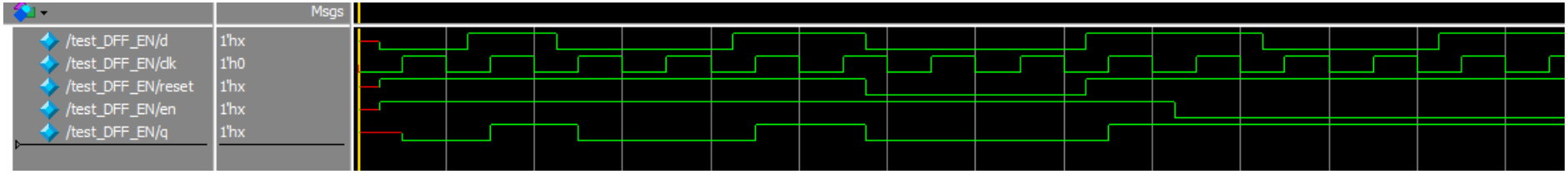
27 March 2020

EE 435

DFF w/ Enable (1 bit):

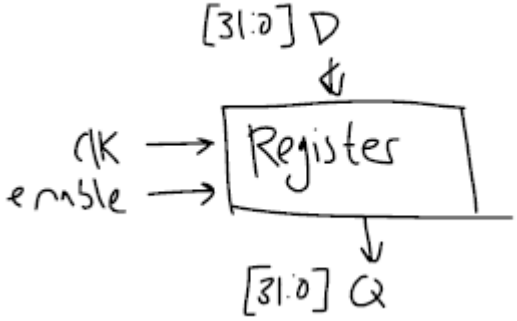
```
module DFF_EN(q, clk, d, reset, en);

output wire q;
input  d, clk, reset, en;

wire dp;

mux2_1 mux(dp, q, d, en);
DFF state(q, clk, dp, reset);

endmodule

module DFF(q, clk, d, reset);

output q;
input  d, clk, reset;

reg    q;


// always @(negedge reset or posedge clk)

always @(negedge reset or posedge clk)
   if (~reset)
      q <= 1'b0;
   else
      q <= d;

endmodule

module mux2_1 (z, i0, i1, s);

output z;
input i0, i1, s;

bufif1 (z, i1, s);
bufif0 (z, i0, s);

endmodule
```

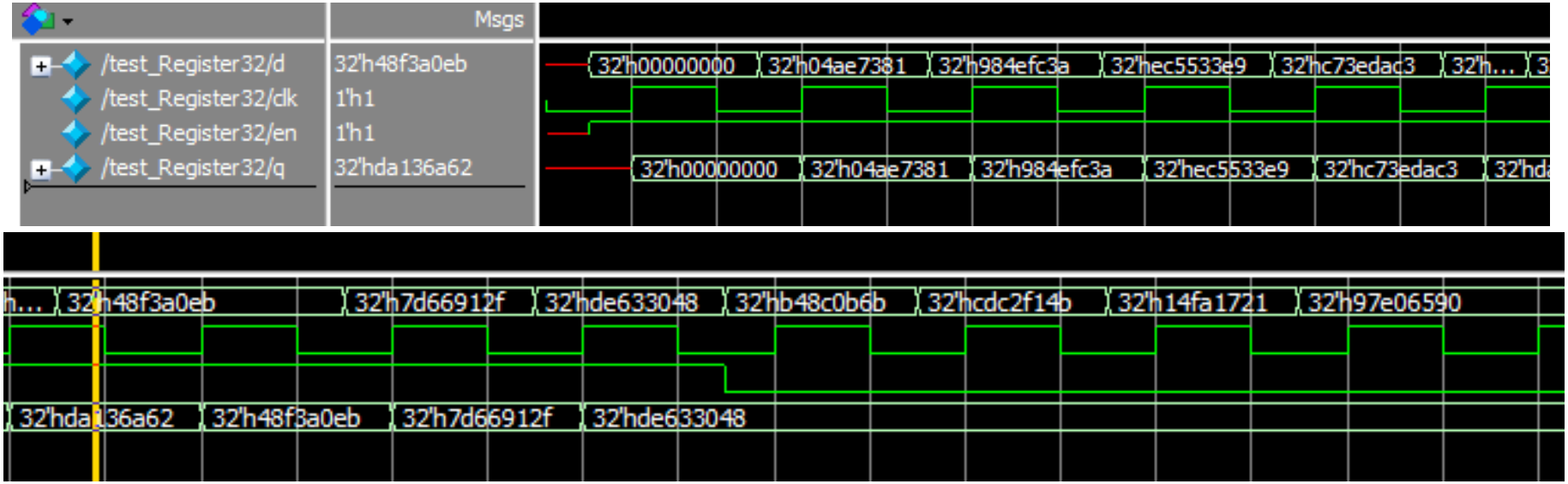




Register (32 bit):

```
module register32 (q, clk, d, en);

output [31:0] q;
input [31:0] d;
input clk, en;

DFF_EN d1[31:0] (q, clk, d, , en);

endmodule
```

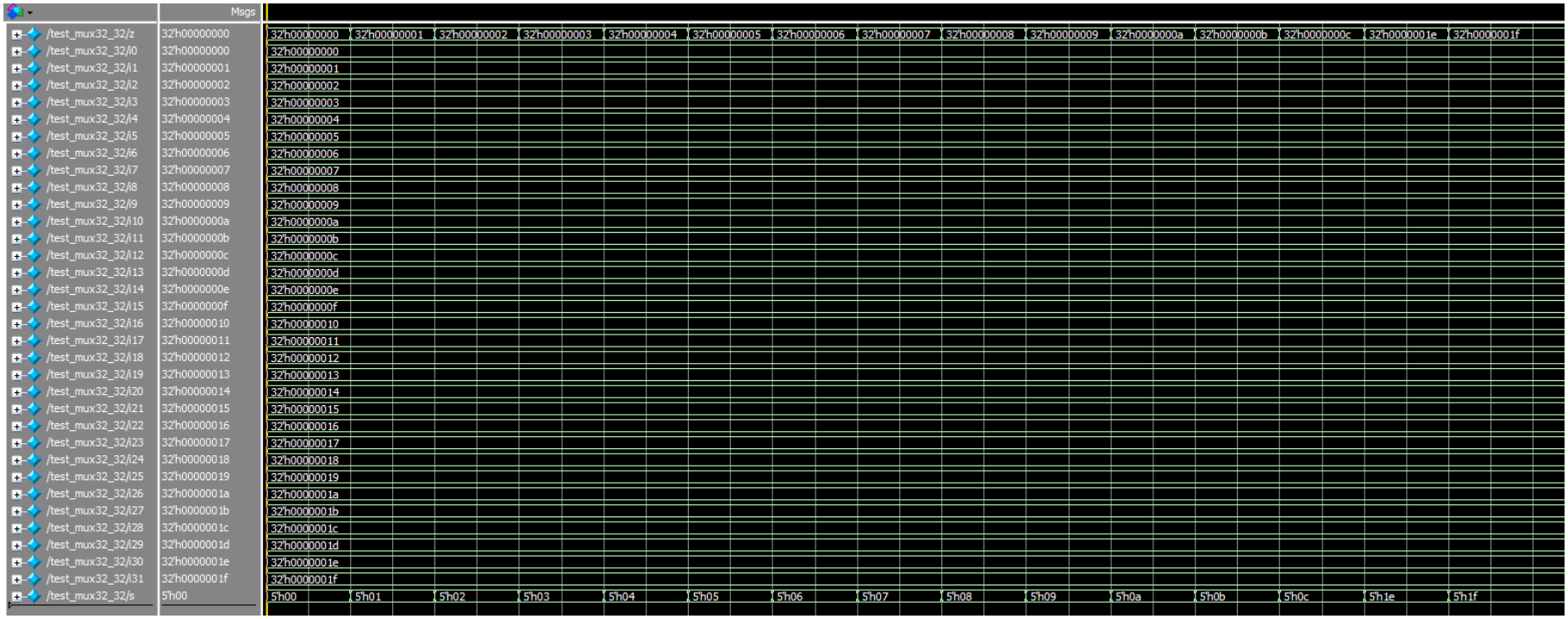


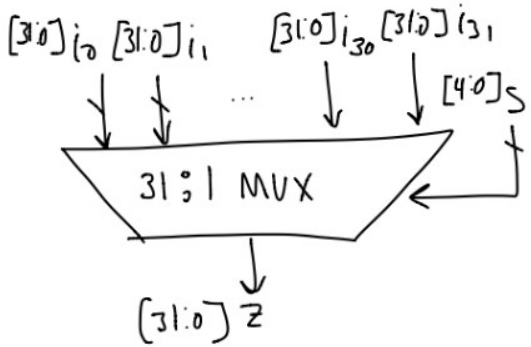


32:1 MUX (32 bit):

```
module mux32_32 (z, i0, i1, i2, i3, i4, i5, i6, i7,
                    i8, i9, i10, i11, i12, i13, i14,
i15,
                    i16, i17, i18, i19, i20, i21, i22,
i23,
                    i24, i25, i26, i27, i28, i29, i30,
i31, s);

output [31:0] z;
input [31:0] i0, i1, i2, i3, i4, i5, i6, i7,
             i8, i9, i10, i11, i12, i13, i14, i15,
             i16, i17, i18, i19, i20, i21, i22, i23,
             i24, i25, i26, i27, i28, i29, i30, i31;
input [4:0] s;

wire [31:0] a, b;

mux16_32 a1(a, i0, i1, i2, i3, i4, i5, i6, i7,
               i8, i9, i10, i11, i12, i13, i14, i15,
s[3:0]);
mux16_32 a2(b, i16, i17, i18, i19, i20, i21, i22, i23,
               i24, i25, i26, i27, i28, i29, i30, i31,
s[3:0]);
mux2_32 a3(z, a, b, s[4]);

endmodule

module mux16_32 (z, i0, i1, i2, i3, i4, i5, i6, i7,
                    i8, i9, i10, i11, i12, i13, i14,
i15, s);

output [31:0]  z;
input [31:0] i0, i1, i2, i3, i4, i5, i6, i7,
             i8, i9, i10, i11, i12, i13, i14, i15;
input [3:0] s;

wire [31:0] a, b;

mux8_32 a1(a, i0, i1, i2, i3, i4, i5, i6, i7, s[2:0]);
mux8_32 a2(b, i8, i9, i10, i11, i12, i13, i14, i15,
s[2:0]);
mux2_32 a3(z, a, b, s[3]);

endmodule


module mux8_32 (z, i0, i1, i2, i3, i4, i5, i6, i7, s);

output [31:0] z;
input [31:0] i0, i1, i2, i3, i4, i5, i6, i7;
input [2:0] s;

wire [31:0] a, b;

mux4_32 a1(a, i0, i1, i2, i3, s[0], s[1]);
mux4_32 a2(b, i4, i5, i6, i7, s[0], s[1]);
mux2_32 a3(z, a, b, s[2]);

endmodule


module mux4_32 (z, i0, i1, i2, i3, s0, s1);

output [31:0] z;
input [31:0] i0, i1, i2, i3;
input s0, s1;

wire [31:0] a, b;

mux2_32 a1(a, i0, i1, s0);
mux2_32 a2(b, i2, i3, s0);
mux2_32 a3(z, a, b, s1);

endmodule
```

```
module mux2_32 (z, i0, i1, s);

output [31:0] z;
input [31:0] i0, i1;
input s;

bufif1 buf0[31:0] (z, i1, s);
bufif0 buf1[31:0] (z, i0, s);

endmodule
```

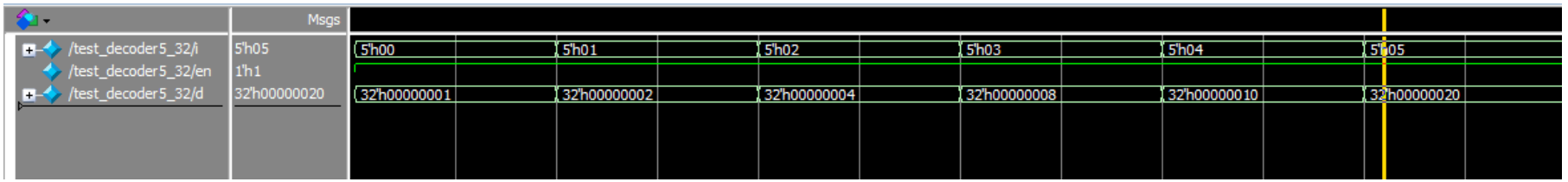


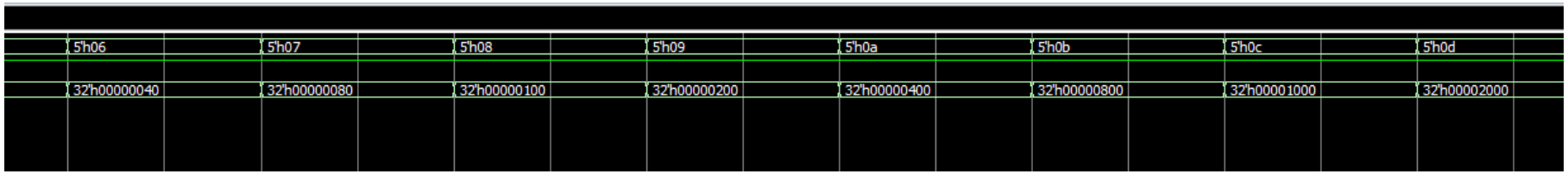


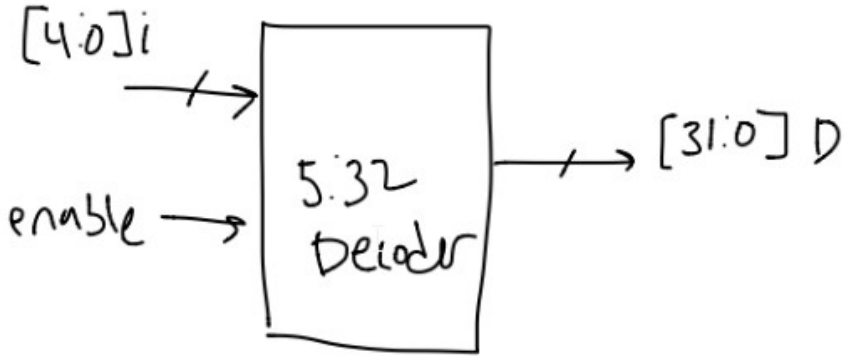
5:32 Decoder (1 bit):

```
output [31:0] d;
input [4:0] i;
input en;

wire i4_not;
wire [31:0] e;

not a1(i4_not, i[4]);

decoder4_16 a2(e[15:0], i[3:0], i4_not);
decoder4_16 a3(e[31:16], i[3:0], i[4]);

and a4(d[0], e[0], en);
and a5(d[1], e[1], en);
and a6(d[2], e[2], en);
and a7(d[3], e[3], en);
and a8(d[4], e[4], en);
and a9(d[5], e[5], en);
and a10(d[6], e[6], en);
and a11(d[7], e[7], en);
and a12(d[8], e[8], en);
and a13(d[9], e[9], en);
and a14(d[10], e[10], en);
and a15(d[11], e[11], en);
and a16(d[12], e[12], en);
and a17(d[13], e[13], en);
and a18(d[14], e[14], en);
and a19(d[15], e[15], en);

and a20(d[16], e[16], en);
and a21(d[17], e[17], en);
and a22(d[18], e[18], en);
and a23(d[19], e[19], en);
and a24(d[20], e[20], en);
and a25(d[21], e[21], en);
and a26(d[22], e[22], en);
and a27(d[23], e[23], en);
and a28(d[24], e[24], en);
and a29(d[25], e[25], en);
and a30(d[26], e[26], en);
and a31(d[27], e[27], en);
and a32(d[28], e[28], en);
and a33(d[29], e[29], en);
and a34(d[30], e[30], en);
and a35(d[31], e[31], en);

endmodule

module decoder4_16 (d, i, en);

output [15:0] d;
input [3:0] i;
input en;

wire i3_not;
wire [15:0] e;

not a1(i3_not, i[3]);

decoder3_8 a2(e[7:0], i[2:0], i3_not);
decoder3_8 a3(e[15:8], i[2:0], i[3]);

and a4(d[0], e[0], en);
and a5(d[1], e[1], en);
and a6(d[2], e[2], en);
and a7(d[3], e[3], en);
and a8(d[4], e[4], en);
and a9(d[5], e[5], en);
and a10(d[6], e[6], en);
and a11(d[7], e[7], en);
and a12(d[8], e[8], en);
and a13(d[9], e[9], en);
and a14(d[10], e[10], en);
and a15(d[11], e[11], en);
and a16(d[12], e[12], en);
and a17(d[13], e[13], en);
and a18(d[14], e[14], en);
and a19(d[15], e[15], en);
```

```
endmodule


module decoder3_8 (d, i, en);

output [7:0] d;
input [2:0] i;
input en;

wire i2_not;
wire [7:0] e;

not a1(i2_not, i[2]);

decoder2_4 a2(e[3], e[2], e[1], e[0], i[1], i[0],
i2_not);
decoder2_4 a3(e[7], e[6], e[5], e[4], i[1], i[0], i[2]);

and a4(d[0], e[0], en);
and a5(d[1], e[1], en);
and a6(d[2], e[2], en);
and a7(d[3], e[3], en);
and a8(d[4], e[4], en);
and a9(d[5], e[5], en);
and a10(d[6], e[6], en);
and a11(d[7], e[7], en);

endmodule

module decoder2_4 (d3, d2, d1, d0, i1, i0, en);

output d3, d2, d1, d0;
input i1, i0, en;

wire i1_not, i0_not;
wire e0, e1, e2, e3;

not a1(i1_not, i1);
not a2(i0_not, i0);
and a3(e0, i1_not, i0_not);
and a4(e1, i1_not, i0);
and a5(e2, i1, i0_not);
and a6(e3, i1, i0);
and a7(d0, e0, en);
and a8(d1, e1, en);
and a9(d2, e2, en);
and a10(d3, e3, en);
endmodule
```

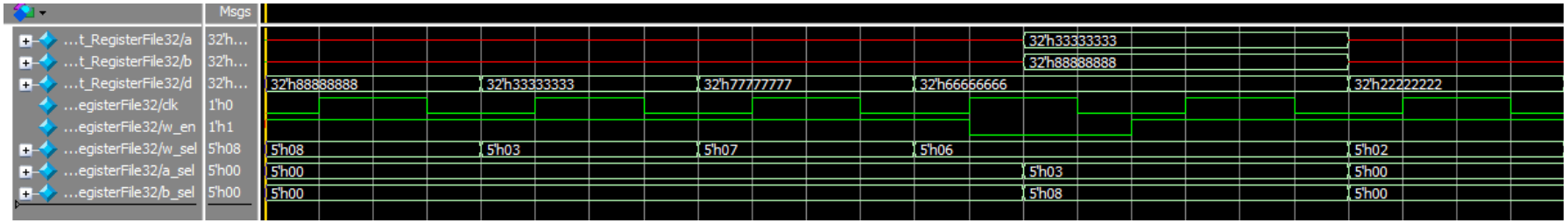


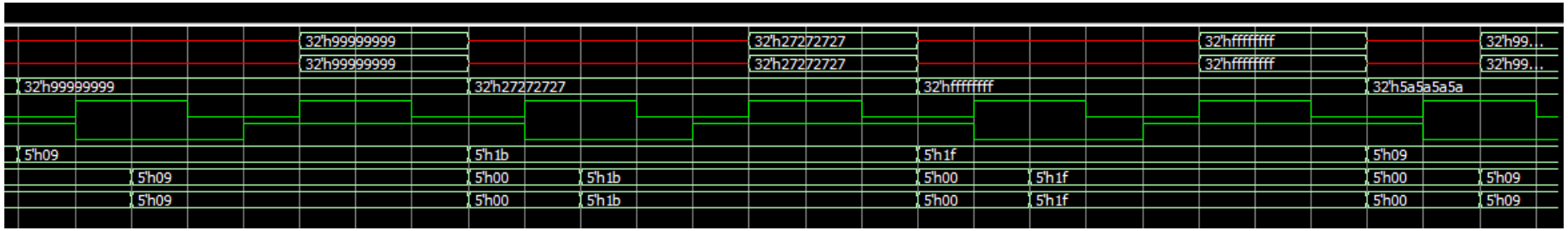


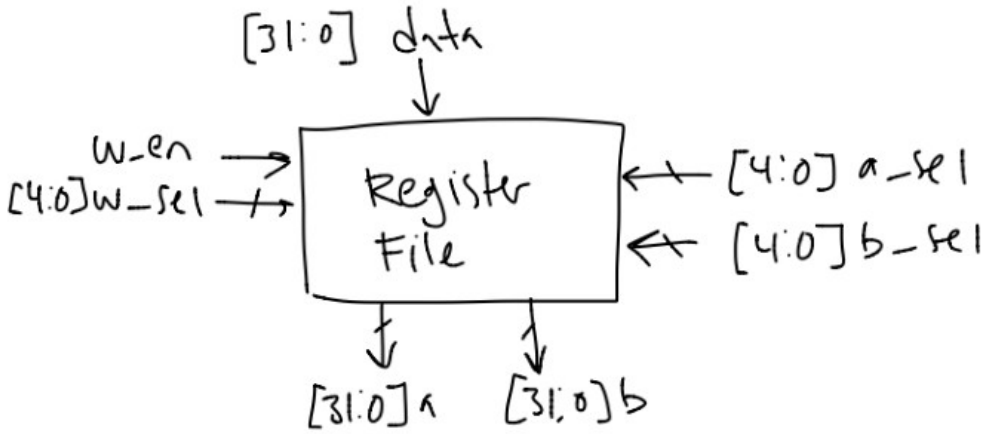

[4:0]i
enable
5:32
Decoder
[31:0] D

Register File (32 bit):

```
module registerFile32 (a, b, clk, d, w_sel, w_en, a_sel,
b_sel);

output [31:0] a, b;
input [31:0] d;
input clk, w_en;
input [4:0] w_sel, a_sel, b_sel;

wire [31:0] q [31:0];
wire [31:0] en_int;

decoder5_32 enable_select (en_int, w_sel, w_en);
register32 reg0 (q[0], clk, d, en_int[0]);
register32 reg1 (q[1], clk, d, en_int[1]);
register32 reg2 (q[2], clk, d, en_int[2]);
register32 reg3 (q[3], clk, d, en_int[3]);
register32 reg4 (q[4], clk, d, en_int[4]);
register32 reg5 (q[5], clk, d, en_int[5]);
register32 reg6 (q[6], clk, d, en_int[6]);
register32 reg7 (q[7], clk, d, en_int[7]);
register32 reg8 (q[8], clk, d, en_int[8]);
register32 reg9 (q[9], clk, d, en_int[9]);
register32 reg10 (q[10], clk, d, en_int[10]);
register32 reg11 (q[11], clk, d, en_int[11]);
register32 reg12 (q[12], clk, d, en_int[12]);
register32 reg13 (q[13], clk, d, en_int[13]);
register32 reg14 (q[14], clk, d, en_int[14]);
register32 reg15 (q[15], clk, d, en_int[15]);
register32 reg16 (q[16], clk, d, en_int[16]);
register32 reg17 (q[17], clk, d, en_int[17]);
register32 reg18 (q[18], clk, d, en_int[18]);
register32 reg19 (q[19], clk, d, en_int[19]);
register32 reg20 (q[20], clk, d, en_int[20]);
register32 reg21 (q[21], clk, d, en_int[21]);
register32 reg22 (q[22], clk, d, en_int[22]);
register32 reg23 (q[23], clk, d, en_int[23]);
register32 reg24 (q[24], clk, d, en_int[24]);
register32 reg25 (q[25], clk, d, en_int[25]);
register32 reg26 (q[26], clk, d, en_int[26]);
register32 reg27 (q[27], clk, d, en_int[27]);
register32 reg28 (q[28], clk, d, en_int[28]);
register32 reg29 (q[29], clk, d, en_int[29]);
register32 reg30 (q[30], clk, d, en_int[30]);
register32 reg31 (q[31], clk, d, en_int[31]);

mux32_32 muxA (a, q[0], q[1], q[2], q[3], q[4], q[5],
q[6], q[7],
                q[8], q[9], q[10], q[11], q[12],
q[13], q[14], q[15],
                q[16], q[17], q[18], q[19], q[20],
q[21], q[22], q[23],
                q[24], q[25], q[26], q[27], q[28],
q[29], q[30], q[31], a_sel);
mux32_32 muxB (b, q[0], q[1], q[2], q[3], q[4], q[5],
q[6], q[7],
                q[8], q[9], q[10], q[11], q[12],
q[13], q[14], q[15],
                q[16], q[17], q[18], q[19], q[20],
q[21], q[22], q[23],
                q[24], q[25], q[26], q[27], q[28],
q[29], q[30], q[31], b_sel);

endmodule
```

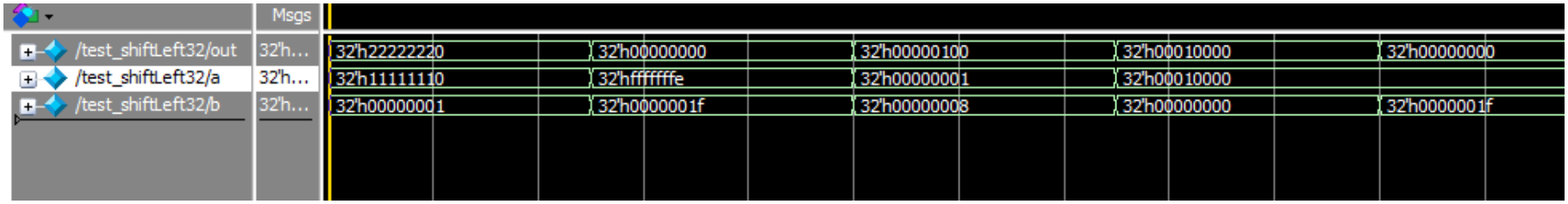

Msgs
...t_RegisterFile32/a
...t_RegisterFile32/b
...t_RegisterFile32/d
...egisterFile32/clk
...egisterFile32/w_en
...egisterFile32/w_sel
...egisterFile32/a_sel
...egisterFile32/b_sel
32'h88888888
32'h33333333
32'h77777777
32'h66666666
32'h22222222
5'h08
5'h03
5'h07
5'h06
5'h02
5'h00
5'h03
5'h08

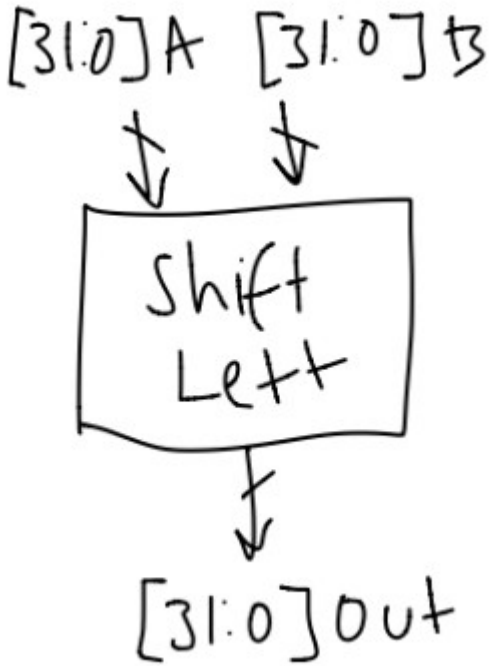

32'h99999999
32'h27272727
32'hffffffff
32'h5a5a5a5a
5'h09
5'h1b
5'h1f
5'h00


[31:0] data
w_en
[4:0] w_sel
Register File
[4:0] a_sel
[4:0] b_sel
[31:0] a
[31:0] b

Shift Left (32 bit):

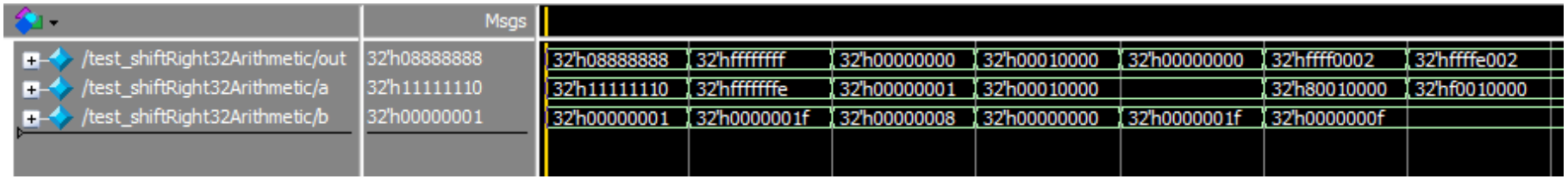
```
module shiftLeft32 (out, a, b);

output [31:0] out;
input [31:0] a, b;

assign out = a << b;

endmodule
```

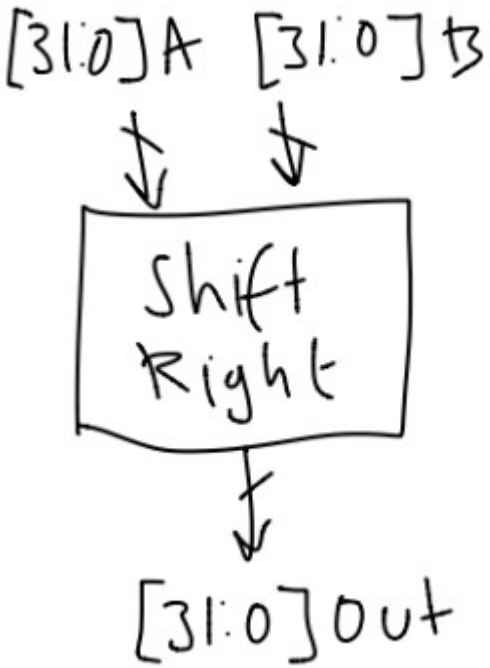




Shift Right (32 bit):

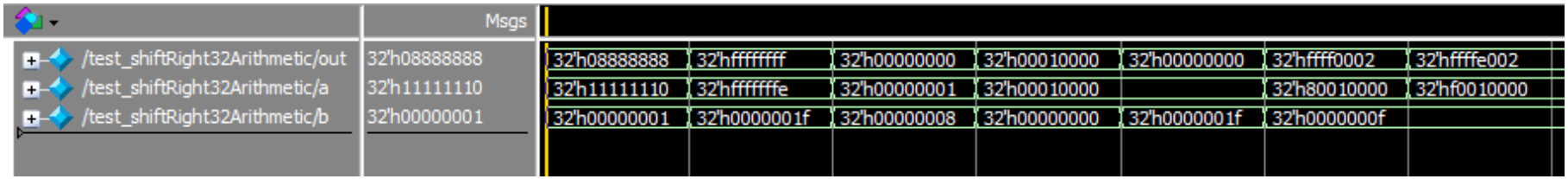
```
module shiftRight32 (out, a, b);

output [31:0] out;
input [31:0] a, b;

assign out = a >> b;

endmodule
```

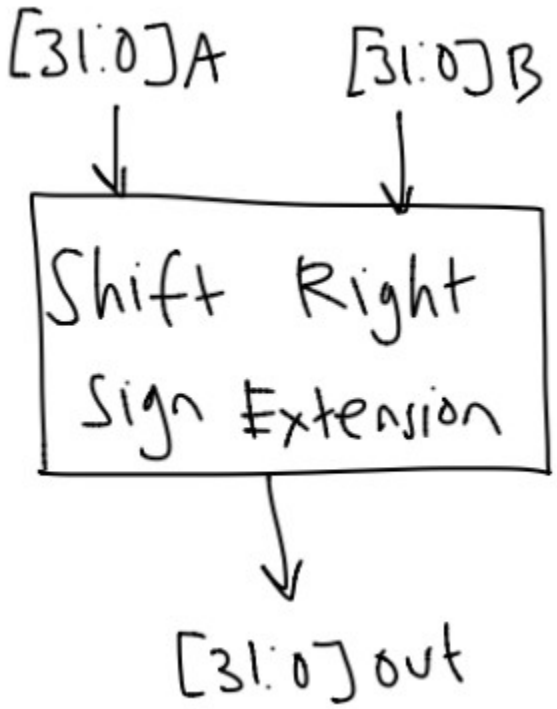




Shift Right with Sign Extension (32 bit):

```
module shiftRight32Arithmetic (out, a, b);

output [31:0] out;
input signed [31:0] a, b;

assign out = a >>> b;

endmodule
```





ALU (32 bit):

```
module ALU(out, a, b, op);

output [31:0] out;
input [31:0] a, b;
input [3:0] op;

wire [31:0] add, sub, my_and, my_or, my_xor, my_xnor,
            my_shl, my_shr, my_sra;

full_adder_32 adder(, add, a, b, 1'b0);
full_subtractor_32 subber(, sub, a, b, 1'b0);
and32 ander(my_and, a, b);
or32 orer(my_or, a, b);
xor32 xorer(my_xor, a, b);
xnor32 xnorer(my_xnor, a, b);
shiftLeft32 shiftLefter(my_shl, a, b);
shiftRight32 shiftRighter(my_shr, a, b);
shiftRight32Arithmetic shiftRightAer(my_sra, a, b);

mux16_32 muxer(out, add, sub, , , , , , ,
    my_and, my_or, my_xor, my_xnor, my_shl, my_shr, my_sra, , op);

endmodule
```