EE2001 - Digital systems lab

Vinita Vasudevan

2s complement

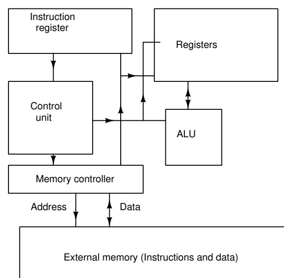
- The decimal value of a n-bit binary number is $\sum_{i=0}^{n-1} a_i 2^i$.
- If a negative number is represented in the twos complement form, its decimal value is $-(2^n - \sum_{i=0}^{n-1} a_i 2^i)$.
- In practice it is obtained by first subtracting the number from $2^n - 1$ (each bit is complemented) and then adding 1 to the number.
- For a positive number $a_{n-1} = 0$ and for a negative number $a_{n-1} = 1$.
- Positive number the decimal value is $\sum_{i=0}^{n-2} a_i 2^i$. Negative number: $-(2^n - 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i) = -2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$

2s complement

- In general, the value is $-a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$
- if A is an n bit number $a_{n-1}a_{n-2}\cdots a_o$, we need $A - A = 0$. If the sum is also n bits, throw away the carry out from the MSB
- If the sum is $n + 1$ bits? Do a sign extension. Verify that the decimal value of the number does not change with sign extension.
- How do you deal with fractions?

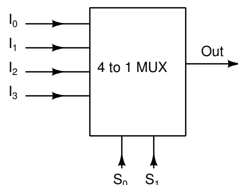
Functional Simulation: Behavioural Modelling

High level model specifying circuit function, without getting into actual implementation



Need to verify funcationality/execution behaviour of various instructions, before going to gate level or even data flow level modelling. Easier to write a C programe to verify functionality, using loops, case statements etc.

Behavioural modelling also useful for components.
Example: Multiplexer



C program

```
if (!S0 && !S1)
{
  Out = I0
}
else if (!S0 && S1)
{
  Out = I1
}
 else if (S0 && !S1)
{
  Out = I2
}
 else
{
  Out = I3
}
```

$$Out = \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1 \bar{S}_0 I_2 + S_1 S_0 I_3$$

Data flow modelling using Verilog

```
module mux4( I0, I1, I2, I3, s, out);

input I0, I1, I2, I3 ;
input[1:0] s;
output out
wire t0, t1;

 assign t0 = (~s[1] & I0) | (s[1] & I2);
 assign t1 = (~s[1] & I1) | (s[1] & I3);
 assign out = (~s[0] & t0) | (s[0] & t1);

endmodule
```

```
module mux4( I0, I1, I2, I3, s, out);

input I0, I1, I2, I3 ;
input[1:0] s;
output out;

 assign out = (s == 0)? I0:
              (s == 1)? I1:
              (s == 2)? I2:
              (s == 3)? I3: 1'bx;

endmodule
```

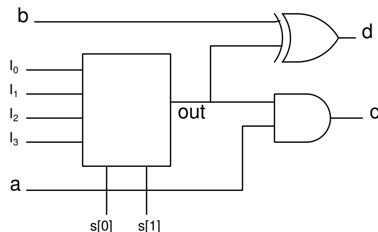
Behavioural Modelling

```
module mux4( I0, I1, I2, I3, s, out);

input I0, I1, I2, I3 ;
input[1:0] s;
output out;
reg out;

 always@(I0 or I1 or I2 or I3 or s)
begin
  if (s == 0)
    out = I0;
  else if (s == 1)
    out = I1;
  else if (s == 2)
    out = I2;
  else if (s == 3)
    out = I3;
end

endmodule
```

- Code is similar to C
- The arguments to the always block constitute the sensitivity list. Any change in these inputs will cause the always block to execute
- The statements within the always block are executed sequentially
- In order to assign values to it, "out" has to be declared as "reg" (similar to what we did in test benches).

Can mix models

```
module circuit( I0, I1, I2, I3, s, a, b, c, d);

input I0, I1, I2, I3 ;
input a, b;
input[1:0] s;
output c, d;
reg out;

 always@(I0 or I1 or I2 or I3 or s)
begin
case( s )
  0: out = I0;
  1: out = I1;
  2: out = I3;
  3: out = I3;
end

assign c = out & a;

xor x1(d, out, b);

endmodule
```



Experiment 3: Behavioural modelling using Verilog

Objective:Model circuits using behavioural models.

- Try out the behavioural and data flow model of a 4-to-1 MUX.
- Model a 1-to-4 demultiplexer using a behavioural model.
- Repeat the experiment using a data-flow model model.
- A and B are 4-bit inputs and the output of the circuit is $MAX(A, B)$. You can use a mix of data flow and behavioural models if you wish.

In all three cases, write a test bench to test the circuit.