DATAFLOW MODELING

Dr.K.Sivasankaran
Associate Professor
VLSI Division
School of Electronics Engineering
VIT University

Dataflow Modeling (1/2)

- Gate level Modeling approach works well for small circuits but not for complex designs.
- Designers can design more effectively if they concentrate on implementing the function at a level of abstraction higher than gate level.
- Dataflow modeling provides a powerful way to implement a design.
- Verilog allows a circuit to be designed in terms of the data flow between registers and how a design processes data rather than instantiation of individual gates.

Dataflow Modeling (2/2)

- With gate densities on chips increasing rapidly, dataflow modeling has assumed great importance.
- Currently, automated tools are used to create a gate-level circuit from a dataflow design
- . This process is called logic synthesis.
- The data flow modeling allows the designer to concentrate on optimizing the circuit in terms of data flow.
- In the digital design community, the term RTL (Register Transfer Level) design is commonly used for a combination of dataflow modeling and behavioral modeling.

Continuous assignment (1/2)

- This level of abstraction level resembles like that of boolean equation representation of digital systems.
- The dataflow assignments are called “Continuous Assignments”.
- Continuous assignments are always active.
- Syntax: ***assign*** *#(delay) target = expression;*
- A Verilog module can contain any number of continuous assignment statements, and all staements execute concurrently.

Continuous assignment (2/2)

- LHS target -> always a net, *not a register.*
- RHS -> registers, nets or function calls.
- Delay values can be specified in terms of time units.
- Whenever an event (change of value) occurs on any operand used on the RHS of an expression, the expression is evaluated and assigned to the LHS target.

Types of continuous assignments

- Regular continuous assignment.

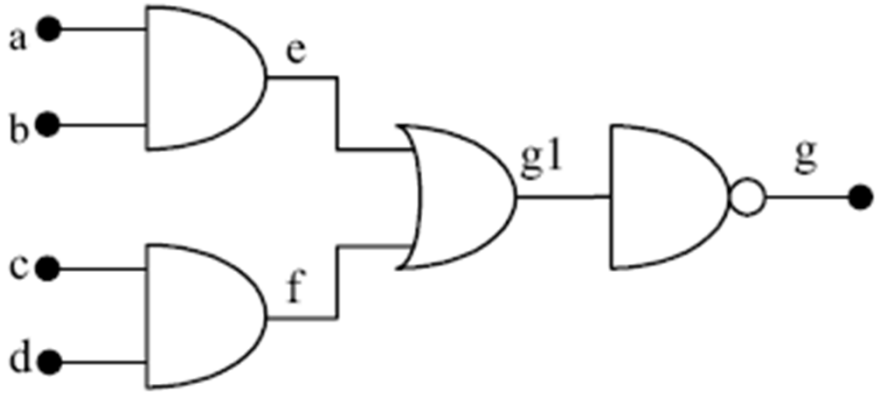
```
wire mux_out;
assign mux_out = sel_in ? b_in : a_in;
```

- Implicit continuous assignment.

```
wire mux_out = sel_in ? b_in : a_in;
```

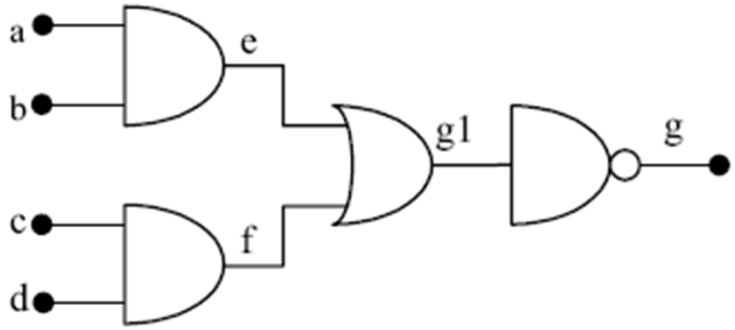
Realizing Using Multiple Continuous Assignment Statement

```
module aoi2(g,a,b,c,d);
output g;
input a,b,c,d;
wire e,f,g1,g;
assign e = a & b;
assign f = c & d;
assign g1 = e | f;
assign g = ~g1;
endmodule
```



Realizing Using Single Continuous Assignment Statement

```
module aoi2(g,a,b,c,d);
output g;
input a,b,c,d;
wire e,f,g1,g;
assign e = a & b,f = c & d, g1 = e|f, g=~g1;
endmodule
```



Realizing using Implicit assignment Statement

```
module aoi3(g,a,b,c,d);
output g;
input a,b,c,d;
wire g;
wire e = a & b;
wire f = c & d;
wire g1 = e|f;
assign g = ~g1;
endmodule
```

Assignment to Vectors

The continuous assignments are equally applicable to vectors.

A single statement can describe operations involving vectors wherever possible.

```
module add_8(a,b,c);
input[7:0]a,b;
output[7:0]c;
assign c = a + b ;
endmodule
```

Concatenation of Vectors

One can concatenate vectors, scalars, and part vectors to form other vectors.

The concatenated vector is enclosed within braces.

Commas separate the components –scalars, vectors, and part vectors.
If a and b are 8- and 4-bit wide vectors, respectively and c is a scalar

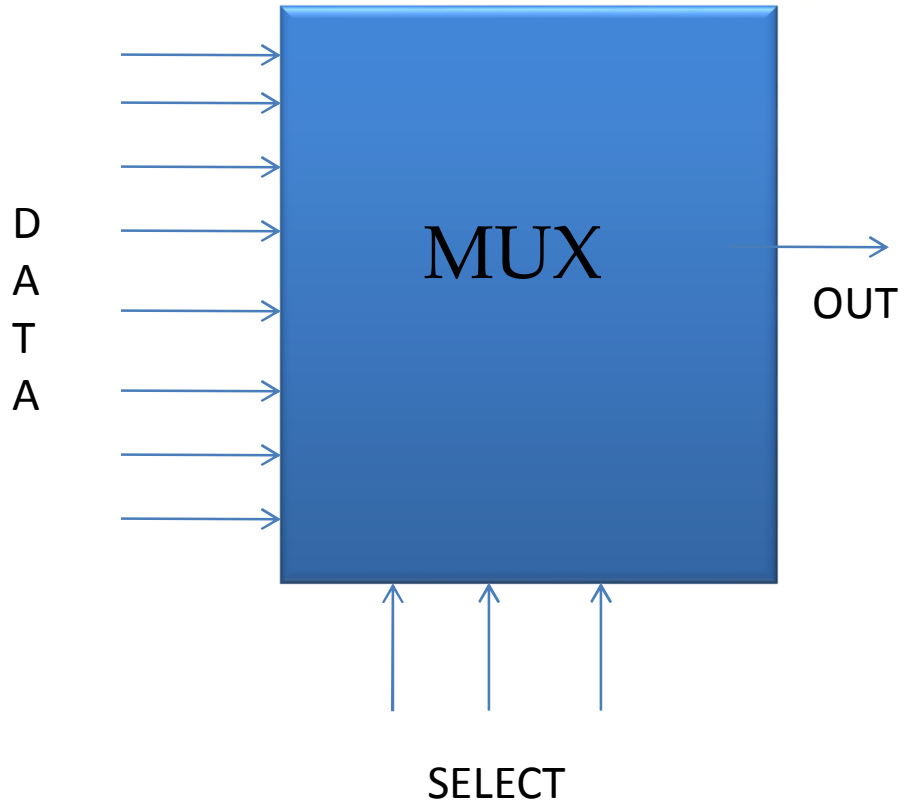
{a, b, c}

stands for a concatenated vector of 13 bits width.

```
module add_8_c(c,cco,a,b,cci);
input[7:0]a,b;
output[7:0]c;
input cci;
output cco;
assign {cco,c} = (a + b + cci);
endmodule
```

Example – Mux Inference

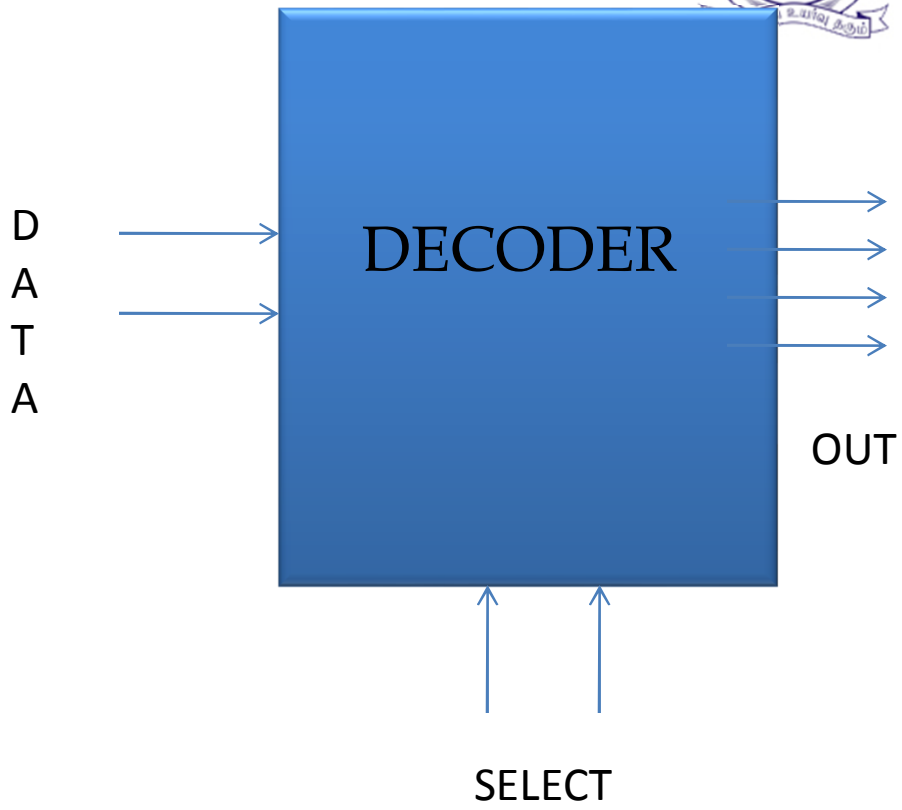
```
module generate_mux
    (data,select,out);
input [0:7] data;
input [0:2]select;
output out;
wire out;
assign out = data[select];
endmodule
```



Note: Non- constant index in expression on RHS generates MUX

Example – Decoder Inference

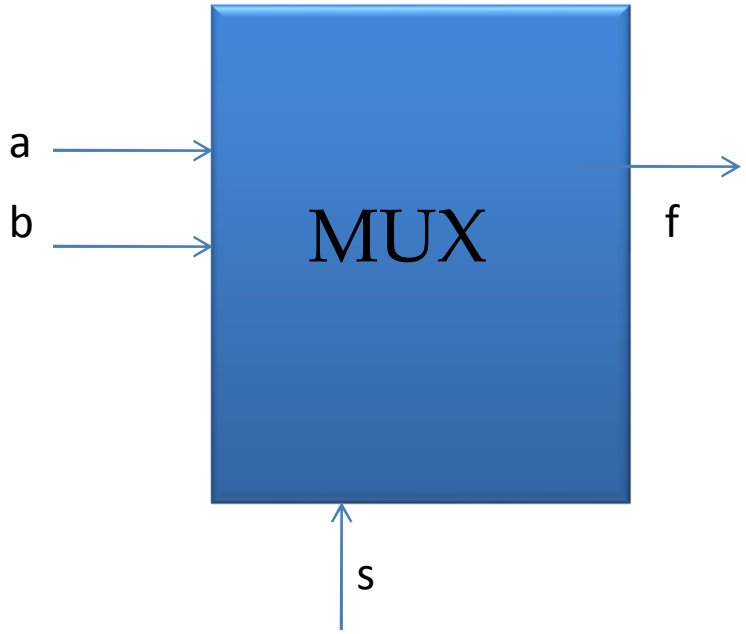
```
module generate decoder
    (data,select,out);
input  data;
input [0:1]select;
output [0:3] out;
wire [0:3] out;
assign out[select]=data;
endmodule
```



Note: Non- constant index in expression on LHS generates DECODER

Example

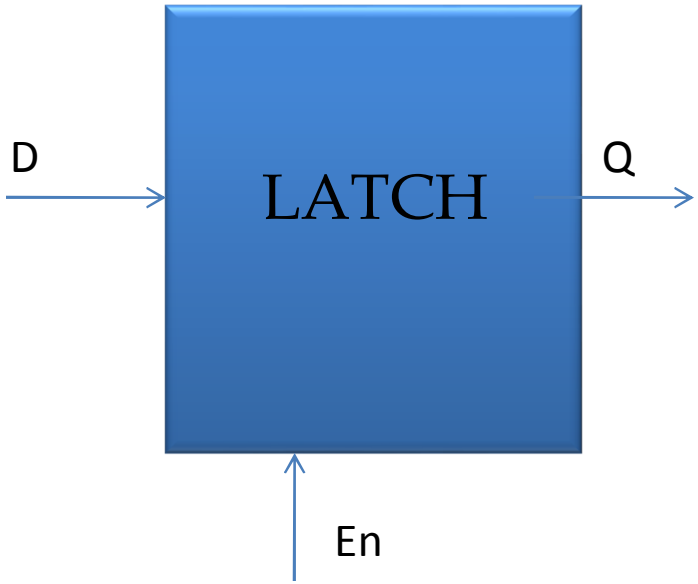
```
module generate_mux
    (a,b,f,s);
input  a,b;
input s;
output  f;
wire  f;
assign f = s? a : b;
endmodule
```



Note: Conditional Operator generate MUX

Example

```
module latch (D,Q,En);
input  D,En;
output Q;
wire  Q;
assign Q=En?D:Q;
endmodule
```



Note: Using "assign" to describe sequential logic using conditional operator.
Cyclic dependency of net also infers latch.

Ways to specifying delays

Delay values control the time between the change in a right hand-side operand and when the new value is assigned to the left-hand side.

- Regular assignment delay
- Implicit continuous assignment delay
- Net declaration delay

Regular assignment delay

- The delay value is specified after the keyword **assign**.
- Any change in values of in1 or in2 will result in a delay of 10 time units before recomputation of the expression in1 & in2, and the result will be assigned to out.

Example:

```
module and_ex(and_out , a_in, b_in);
input a_in, b_in;
output and_out;
wire  and_out;
assign #10 and_out = a_in & b_in;
endmodule
```

Implicit continuous assignment delay

An equivalent method is to use an implicit continuous assignment to specify both a delay and an assignment on the net.

Example:

```
module and_ex(and_out , a_in, b_in);
input a_in, b_in;
output and_out;
wire  #10 and_out = a_in & b_in;
endmodule
```

Net declaration delay

- A delay can be specified on a net when it is declared without putting a continuous assignment on the net.
- If a delay is specified on a net out, then any value change applied to the net out is delayed accordingly.

Example:

```
module and_ex(and_out , a_in, b_in);
input a_in, b_in;
output and_out;
wire  #10 and_out;
assign and_out = a_in & b_in;
endmodule
```

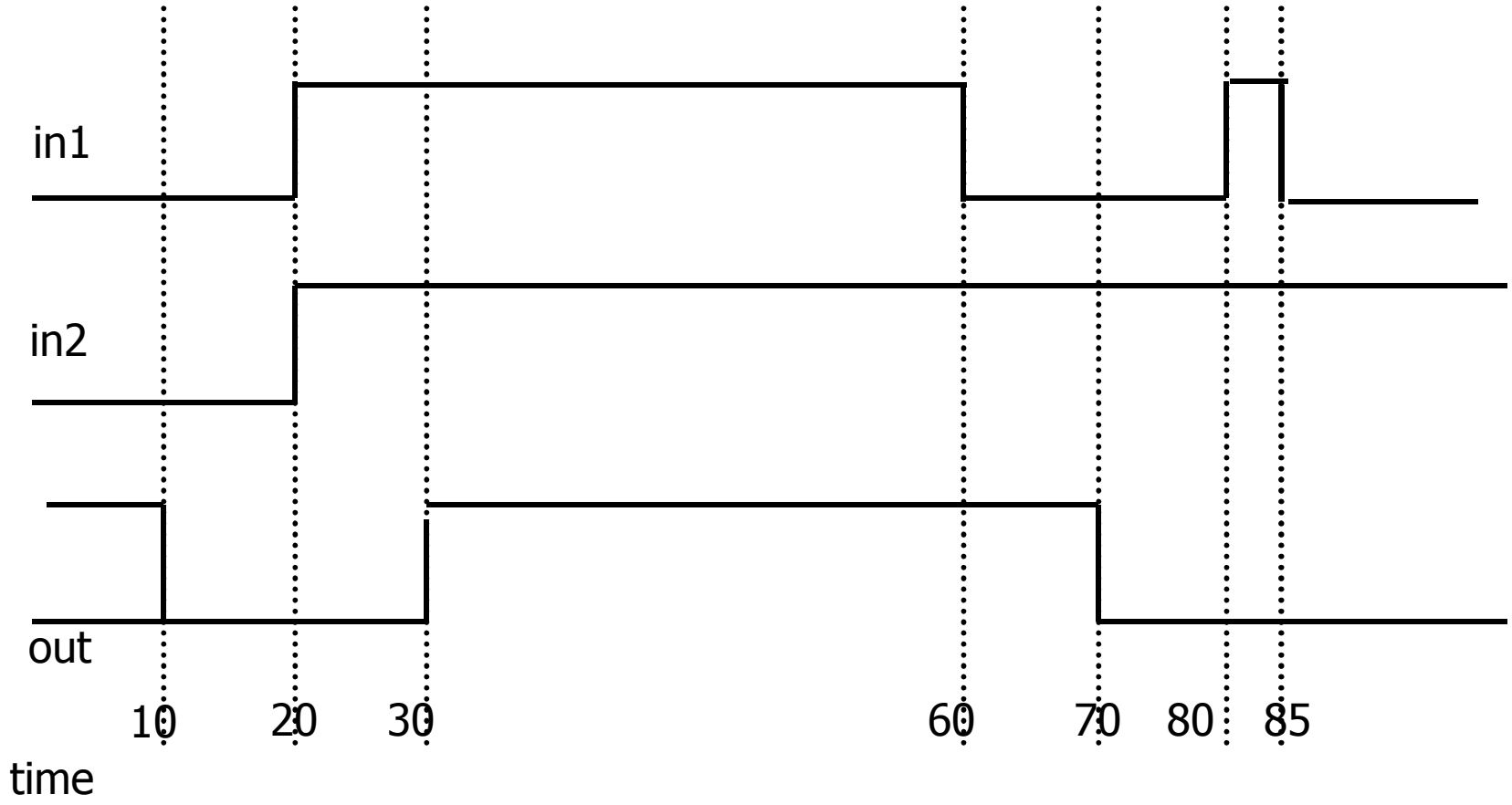
Inertial delay

```
wire out;
assign #10 out = i1 & i2; // Regular assignment delay
```

- The inertial delay of real circuits is modeled through regular assignment delay.

- Any event on the RHS signals which is not lasting for the amount of inertial delay specified will not have any effect on the LHS target.

Inertial delay



Some Valid Statements

```
assign outp = (p == 4'b1111);

if (load && (select == 2'b01)) …….

assign a = b >> 1;

assign a = b << 3;

assign f = {a, b};

assign f = {a, 3'b101, b};

assign f = {x[2], y[0], a};

assign f = { 4{a} }; // replicate four times

assign f = {2'b10, 3{2'b01}, x};
```