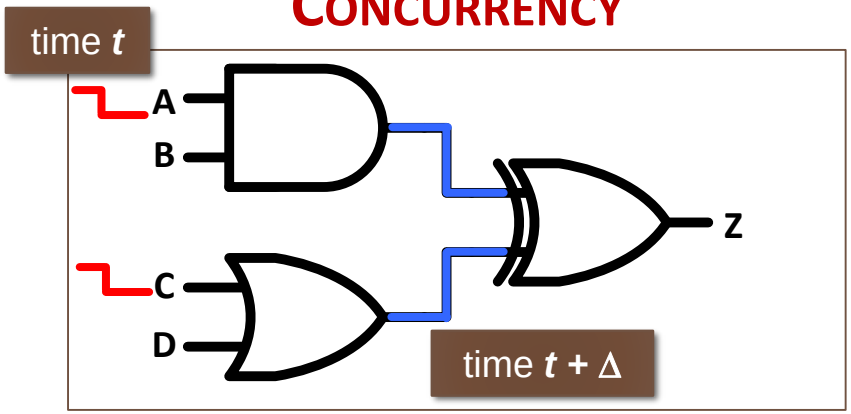
VERILOG for Sequential Circuits

©COPYRIGHT CHUA DINGJUAN. ALL RIGHTS RESERVED.

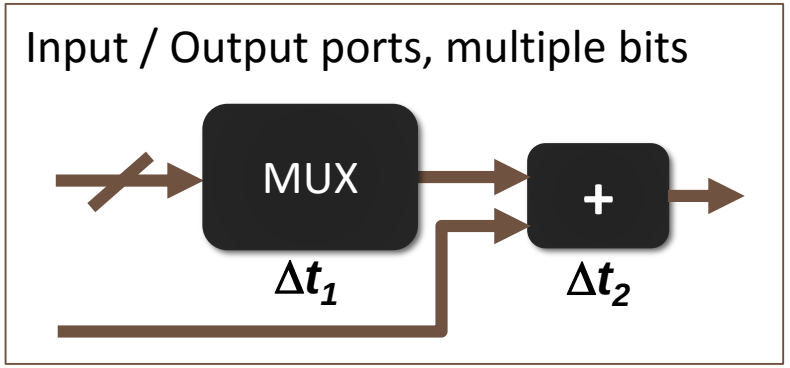
What are HDLs?

***Hardware* Description Languages** (HDLs) are programming languages for describing digital circuits and systems.

CONCURRENCY



STRUCTURE & TIME



Today, ***Verilog* and *VHDL*** are the two leading HDLs.

Combines dataflow, structural and behavioral modeling styles.

Virtually every chip (FPGA, ASIC, etc.) is designed in part using one of these two languages.

Xilinx and Altera are the two largest FPGA manufacturers.

Verilog…

Verilog is an IEEE 1364 Standard → link here

Used for ***Modeling, Simulation*** and ***Synthesis*** of digital circuits.

Advantages :

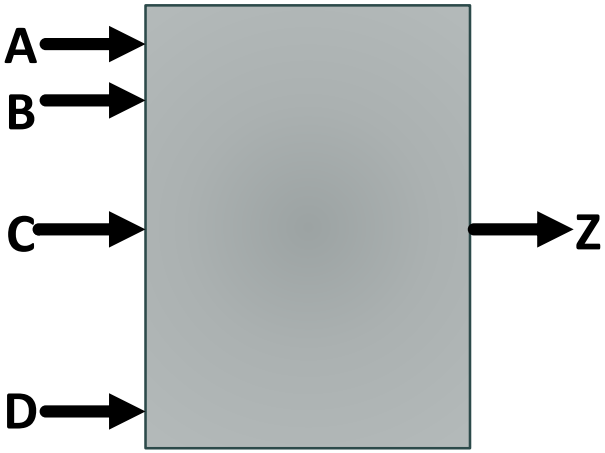
- Reduces Design Time → Cost
- Improves Design Quality
- Vendor and Technology Independence
- Easy Design Management

Disadvantages :

- Cost (Including training you and me!)
- Debugging

The Module

A piece of hardware with inputs & outputs : *module*



Module Name

Port Declaration

```
module bigbox (input a, b, c, d,
               output z);
// Here is where you work your
// magic!
endmodule
```

- Verilog is CasE-SeNSitiVe….
- Module Name : Use meaningful identifiers (~bigbox)
- Port Direction : `input`, `output`, `inout` (bidirectional)
- Port Bitwidth : `input [4:0] a , output [7:14] y`
- Don't forget the __;__!

```
y = 8'hFA
```

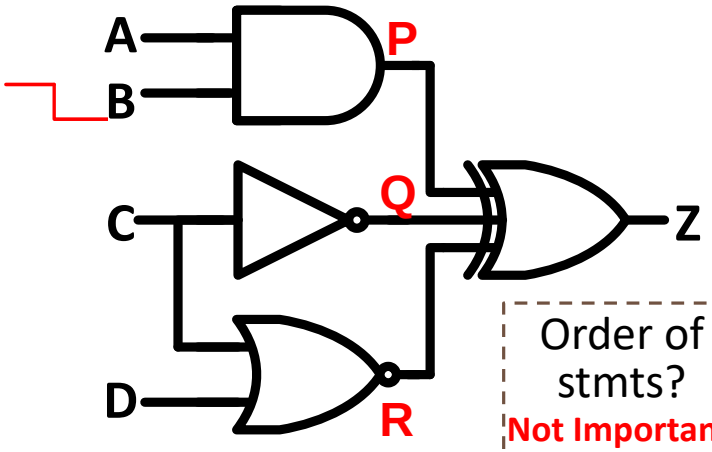
number of binary bits

Value (11111010)

h, o, d, b (radix)

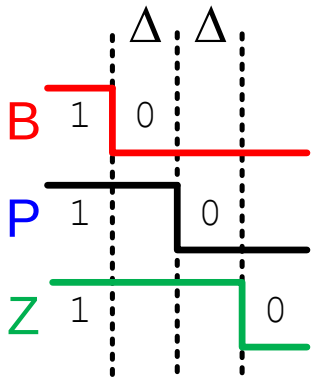
Continuous Assignment (Dataflow)

`assign` statements are used to model combinational logic



Order of stmts? **Not Important!**

```
module bigbox (input a, b, c, d,
               output z);
wire p, q, r;        internal connections

assign z = p ^ q ^ r;   1
assign q = ~c;          2
assign p = a & b;       3
assign r = ~(c | d);    4

endmodule
```



- Whenever there's an ***event*** on the RHS signal, expression is evaluated and assigned (Δ = prop. delay) ➔ *continuously monitored*
- Multiple statements can be executed in parallel (concurrently)
- **`wire`** is used to represent an internal connection

Useful Operators

- Boolean (bit-wise), logical, arithmetic, concatenation.
- Use brackets for readability, take note if **synthesizable*.

High

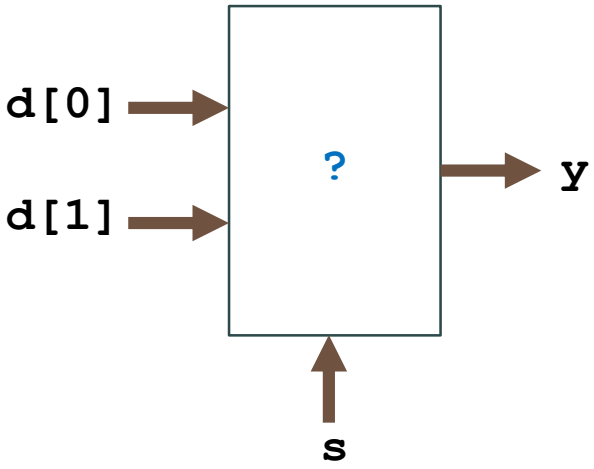
Precedence

Low

Operator	Description	Examples: a = 4'b1010, b=4'0000
`!, ~`	Logical negation, Bit-wise NOT	**!a = 0, !b =1, ~a=4'b0101, ~b=4'b1111**
`&, |, ^`	Reduction (Outputs 1-bit)	**&a = 0, |a=1, ^a = 0**
`{__,__}`	Concatenation	{b, a} = 8'b00001010
`{n{___}}`	Replication	{2 {a} } = 8'b10101010
`*, /, %,`	Multiply, *Divide, *Modulus	3 % 2 = 1, 16 % 4 = 0
`+, -`	Binary addition, subtraction	a + b = 4'b1010
`<< , >>`	Shift Zeros in Left / Right	a << 1 = 4'b0100, a >> 2 = 4'b0010
`<, <=, >, >=`	Logical Relative (1-bit output)	**(a > b) = 1**
`==, !=`	Logical Equality (1-bit output)	**(a == b)= 0** **(a != b)= 1**
`&, ^, |`	Bit-wise AND, XOR, OR	**a&b = 4'b0000** **a|b = 4'b1010**
`&&, ||`	Logical AND, OR (1-bit output)	**a&&b = 0** **a||b = 1**
`?:`	Conditional Operator	<out> = <condition> ? If_ONE : if_ZERO

Conditional Operator…

The **?:** conditional operator allows us to select the output from a set of inputs based on a condition.




```
module mystery ( input s, input [1:0] d,
                 output y);

assign y = s ? d[1] : d[0];

endmodule
```

<output> = <condition> ? If_ONE : If_ZERO

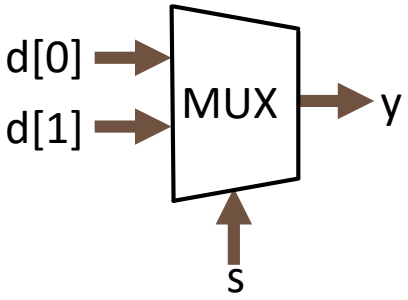
s= 1 ➔ y = d[1], s= 0 ➔ y= d[0]

- This expression is evaluated whenever there is an event on any input.
- What is this block? **- Multiplexer**

Procedural Assignment : `always`

Behavioral, higher-level description of logic.

2 assignment types : **Blocking** & **Non-Blocking**




```
module mux21( input s, input [1:0] d,
              output reg y);
```

Anything assigned in an **always** block must be declared as type **reg**

```
always @ (s, d)

begin
   if (s == 1'b0)
       y= d[0];
   else
       y= d[1];

end
endmodule
```

Conceptually, the **always** block runs o*nce* when a signal in *sensitivity list* (s,d) changes value.

Statements executed sequentially & evaluated instantaneously. ➔ Order matters!

begin and **end** behave like parentheses/brackets for conditional statements.

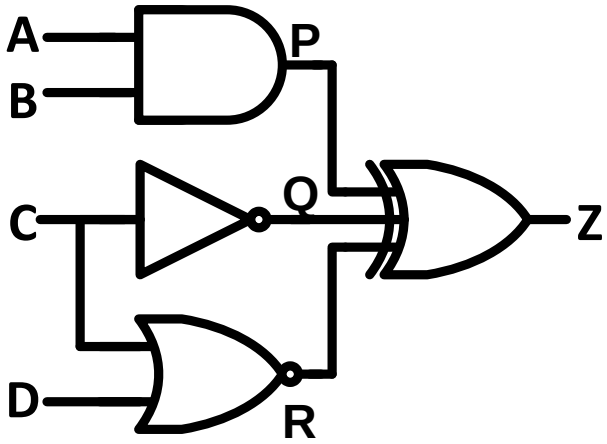
Some notes on: `always`

- **`always@(*)`** includes all signals that are read in statements.
- Statements within **`always`** block are executed ***sequentially***.
- Variables within sensitivity list are very important!
- **`if`**--**`else if`**--**`else`**, **`case`**, **`for`**, **`while`** can only be used in procedural assignments (always blocks)
- Multiple always blocks run in parallel, concurrently. (*Race)
- No ~~**`assign`**~~ in always blocks!
- If using **`posedge`** / **`negedge`**, all signals in sensitivity list needs to be specified with either **`posedge`** / **`negedge`**.

Registers

- Anything assigned in an **`always`** block must be declared as type **`reg`**
- In Verilog, the term register (**`reg`**) simply means a variable that can hold a value. (cf. **`wire`**)
- Values of registers can be changed instantaneously.

Equivalence…




```
module bigbox
(input a,b,c,d, output z);

wire p, q, r;

assign q = ~c;
assign z = p ^ q ^ r;
assign p = a & b;
assign r = ~(c | d);

endmodule
```

```
module bigbox
(input a,b,c,d, output reg z);
reg p,q,r;
always @ ( *  )
   begin
      p = a & b;
      q = ~c;
      r = ~(c | d);
      z = p ^ q ^ r;
   end
endmodule
```

Blocking & Non-blocking

Verilog supports two types of assignments within **always**

= blocking assignment
- Sequential evaluation
- Immediate assignment

```
always @ (*)
begin
x = y;
z = ~x;
end
```

1) Evaluate y, assign result to x
2) Evaluate ~x, assign result to z

Behaviour	x	y	z
Initial Condition	0	0	1
y changes	0	**1**	1
x = y	**1**	1	1
z = ~x	1	1	**0**

<= non-blocking asignment
- Sequential evaluation
- *Deferred* assignment

```
always @ (*)
begin
x <= y;
z <= ~x;
end
```

1) Evaluate y, defer assignment
2) Evaluate ~x, defer assignment
3) Assign x and z with new values

Behaviour	x	y	z	Deferred
Initial Condition	0	0	1	
y changes	0	**1**	1	
x <= y	0	1	1	x <= 1
z <= ~x	0	1	1	z <= 1
Assignment	**1**	1	**1**	

Example

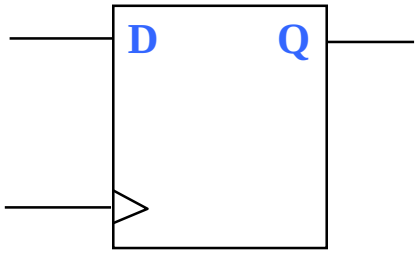
```
module example(input [2:0] A,
               output reg [2:0] V, Z, W);
always @ (A)
    begin

    V = A | 3'b001;
    Z <= V | 3'b100;
    W = Z;

    end
endmodule
```

Behaviour	A	V	Z	W	Deferred
Initial Condition	000	001	101	000	
A changes	**010**	001	101	000	
Stmt 1	010	**011**	101	000	
Stmt 2	010	011	101	000	**Z<=111**
Stmt 3	010	011	101	**101**	
Assignment	010	011	**111**	101	

An event occurs on **A** at simulation time :

- Stmt 1 is executed and V is assigned immediately
- Stmt 2 is executed and defer assignment to Z
- Stmt 3 is executed using old value of Z.
- Z is assigned.

Verilog Time! – D-FF



CLK	D	Q+
↑	0	0
↑	1	1

```
module dff ( input d, clk,
             output reg q);
```

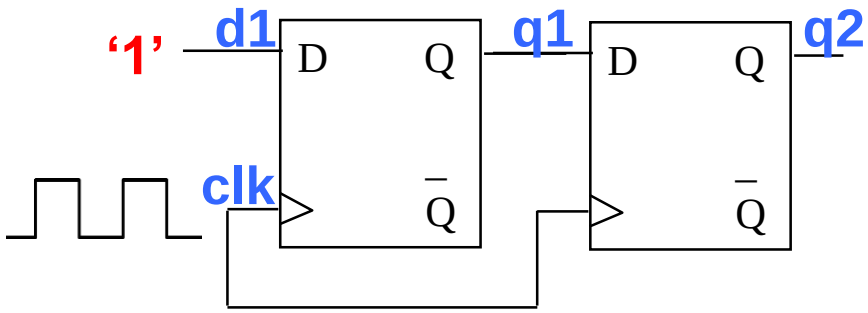
If posedge / negedge is used in the sensitivity list, ALL signals must be used with posedge / negedge.

```
always @ (posedge clk)

begin

    q <= d;        or        q = d;

end
```

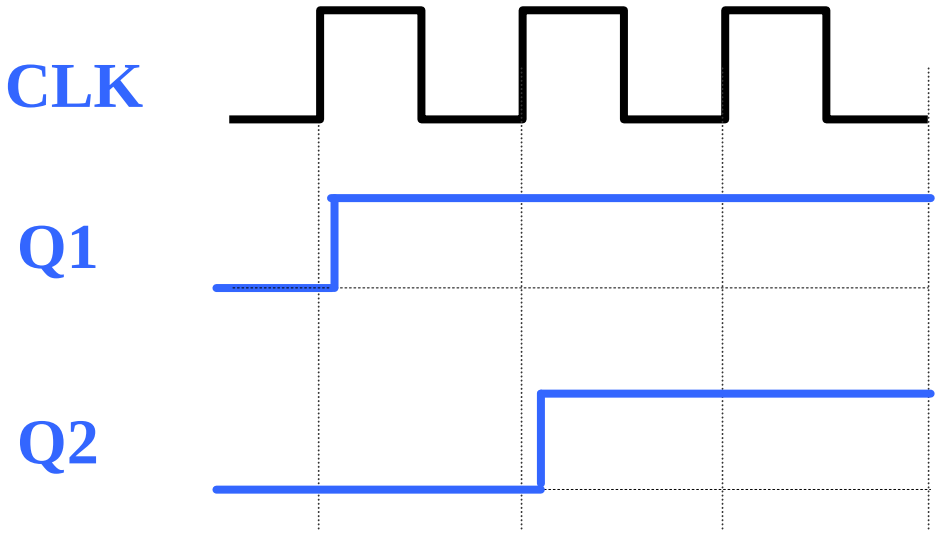
No difference since there is only one line!

```
endmodule
```

Two D Flip-Flops…

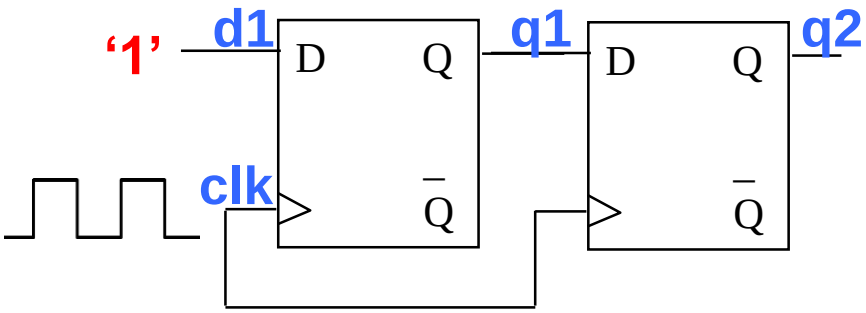


CLK	D	Q^+
↑	0	0
↑	1	1



Behaviour	q1	q2
Reset FFs	0	0
@ 1st rising edge	1	0
@ 2nd rising edge	1	1

Two D Flip-Flops… and Verilog!



Behaviour	q1	q2
Reset FFs	0	0
@ 1st rising edge	1	0
@ 2nd rising edge	1	1

```
always @ (posedge clk)
begin

q1 = d1;
q2 = q1;

end
```

Behaviour	q1	q2
Reset FFs	0	0
@ 1st rising edge	1	1
@ 2nd rising edge	1	1

```
always @ (posedge clk)
begin

q1 <= d1;
q2 <= q1;

end
```

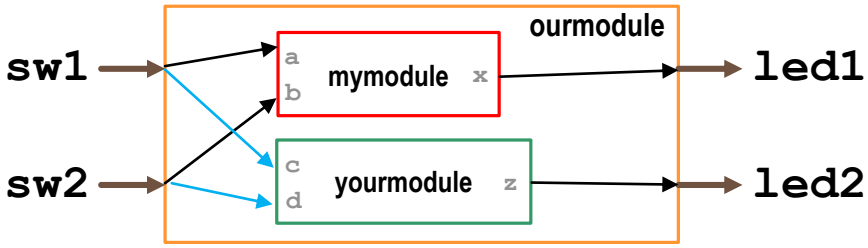
Behaviour	q1	q2
Reset FFs	0	0
@ 1st rising edge	1	0
@ 2nd rising edge	1	1

Structural Modeling

- For modular designs, the top design is often specified as interconnected blocks.
- Two examples below demonstrate port connection by position / name.

```
module mymodule (input a, b, output x);
...
endmodule
```

```
module yourmodule (input c, d, output z);
…
endmodule
```



Port Connection by Position

```
module ourmodule (input sw1, sw2,
                  output led1, led2);

mymodule M1 (sw1, sw2, led1);
yourmodule M2 (sw1, sw2, led2);

endmodule
```

Port Connection by Name

```
module ourmodule (input sw1, sw2,
                  output led1, led2);

mymodule M1 (  .a (sw1),
               .b (sw2),
               .x (led1) );

yourmodule M2( .z(led2),
               .c (sw1),
               .d(sw2) );

endmodule
```

Basic Guidelines…

#1: When modeling sequential logic, use nonblocking assignments.

#2: When modeling simple combinational logic, use continuous assignments (assign).

#3: When modeling complex combinational logic, use blocking assignments in an always block.

#3: When modeling both sequential and combinational logic within the same always block, use nonblocking assignments.

#4: Do not mix blocking and nonblocking assignments in the same always block.

#5: Do not make assignments to the same variable from more than one always block.

Summary

- Operators (`~`, `*`, `/`, `+`, `-`, `&`, `^`, `|`)
- Continuous assignments (`assign`)
- Procedural assignments (`always`)
  - Blocking assignment (`=`)
  - Non-blocking assignment (`<=`)
- Modeling of multiple D Flip-flops
- Structural Modeling

Try this!

`assign` is used for

In continuous assignments, the code is executed

The code in the always block is executed when

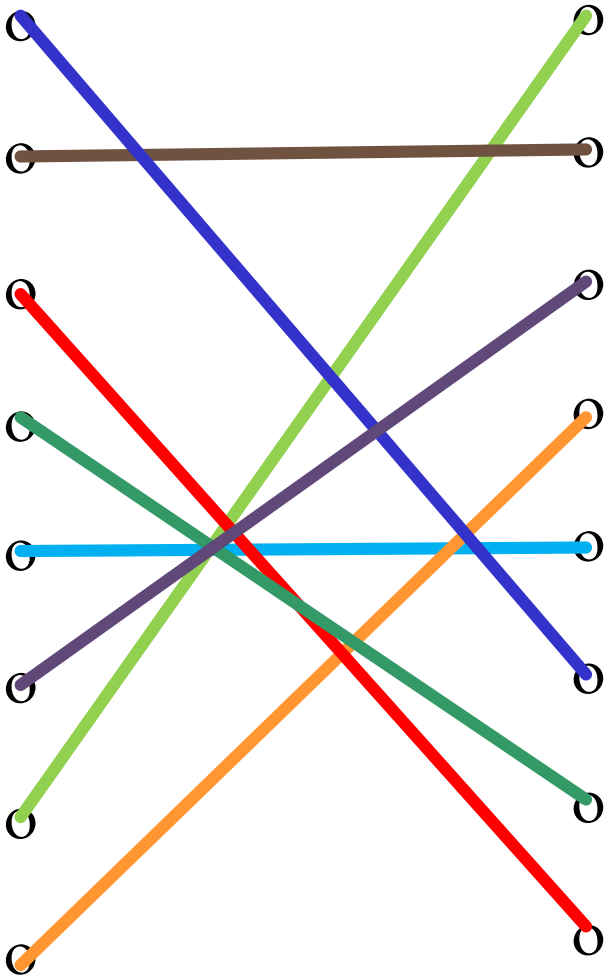
`always` is used for

<= is a

`endmodule` Is always paired with

The sensitivity list follows the

Code in `always` block is executed



`always @.`

when any RHS signal changes

`module`

sequentially.

non-blocking procedural assignment.

continuous assignments.

procedural assignments.

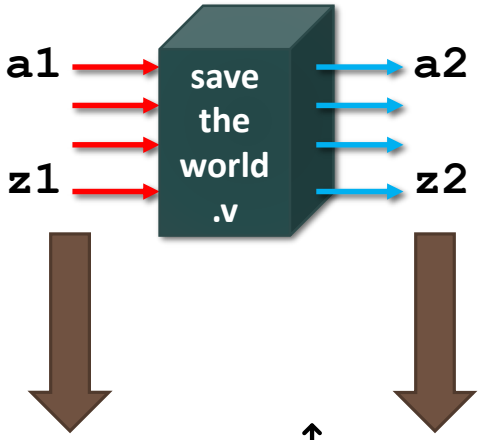
a signal in the sensitivity list changes.

Verilog : Simulation & Synthesis

SELF - READING

What is Simulation?

```
module savetheworld (input a1, … z1,
                     output a2, …,z2);
  ……. ……. ……. ……. …….
  ……. ……. ……. ……. …….
  ……. ……. ……. ……. …….
endmodule
```



How do we know our design actually works?

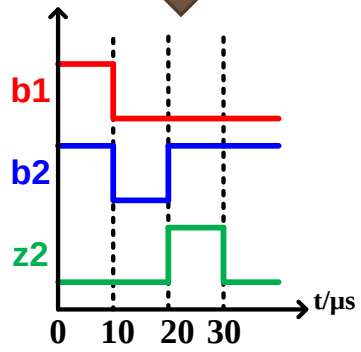
- Functional Simulation (Xilinx)

Verilog Code

```
module
……. …….
……. …….
endmodule
```

Test Bench

```
call module
a1=9;
b1=1;
//wait for 10u
#10
b1=0;
```



Method

- Designer applies input values to the code
- Simulator produces corresponding outputs in truth tables / timing diagrams
- Simulators usually assume negligible propagation gate delays.

What is Synthesis?

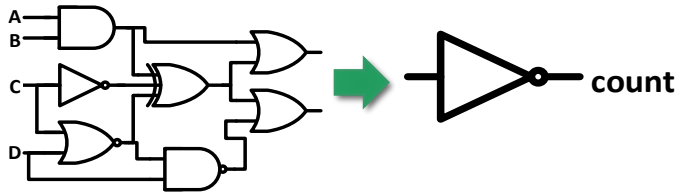
Now that our design is working, time to save the world.

Synthesis :

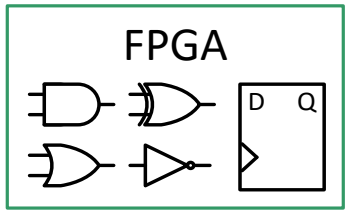
1) Translation
   - Code is transformed into hardware (gates & wires).
2) Optimization
   - Minimizes the amount of hardware required.
3) Mapping
   - Implements hardware on target device.

```
module savetheworld
(input a1,…z1, output a2,…,z2);
        ……. ……. ……. …….
          count <= count + 1 ;
endmodule
```

Translation + Optimization



Mapping



"This looks like a NOT gate. My FPGA can do some of these…"

Summary

- Operators (`~`, `*`, `/`, `+`, `-`, `&`, `^`, `|`)
- Continuous assignments (`assign`)
- Procedural assignments (`always`)
  - Blocking assignment (`=`)
  - Non-blocking assignment (`<=`)
- Modeling of multiple D Flip-flops
- Structural Modeling