ECE 551

Digital Design And Synthesis

Fall '17

Lecture 2

- Verilog Syntax
- Structural Verilog
- Timing in Verilog

Administrative Matters

- Tutorial sessions
  - Last section tonight 6:30 in B555
  - Can also do on your own
- If you are a windows user install ModelSim Student Edition.
- OR...look at video on how to remote to Unix
- Watch Videos on mid & final sections of Lecture02
  - Quiz on these lectures next Monday in class.
- Homework #1
  - Due Next week by beginning of class

Comments in Verilog

- Commenting is important
  - In industry many other poor schmucks are going to read your code
  - Some poor schmuck (perhaps you 4 years later) are going to have to reference your code when a customer discovers a bug.
- The best comments document why you are doing what you are doing, not what you are doing.
  - Any moron who knows verilog can tell what the code is doing.
  - Comment why (motivation/thought process) you are doing that thing.

Commenting in Verilog

```
always @(posedge clk)

begin
   Sig_FF1 <= Sig          // Capture value of Sig Line in FF
   Sig_FF2 <= Sig_FF1;     // Flop Sig_FF1 to form Sig_FF2
   Sig_FF3 <= Sig_FF2;     // Flop Sig_FF2 to form Sig_FF3
end

// start_bit is ~Sig_FF2 & Sig_FF3
assign start_bit = (~Sig_FF2 && Sig_FF3) ? 1'b1 : 1'b0;
```

(Read with sarcasm)

"Thanks for the commenting the code pal. It tells me so much more than the verilog itself".

Commenting in Verilog

```
always @(posedge clk)
/***********************************************
* Sig is asynchronous and has to be double flopped  *
* for meta-stability reasons prior to use ************
*********************************/
begin
   Sig_FF1 <= Sig;
   Sig_FF2 <= Sig_FF1;  // double flopped meta-stability free
   Sig_FF3 <= Sig_FF2;  // flop again for use in edge detection
end

/***********************************************
* Start bit in protocol initiated by falling edge of Sig line *
************************************************/
assign start_bit = (~Sig_FF2 && Sig_FF3) ? 1'b1 : 1'b0;
```

Can see 2 types of comments.

Comment to end of line is //

Multi line comment starts with /* and ends with */

- This is better commenting. It tells you why stuff was done

Numbers in Verilog

- General format is: <size>'<base><number>
- Examples:
  - 4'b1101 // this is a 4-bit binary number equal to 13
  - 10'h2e7 // this is a 10-bit wide number specified in hex
- Available bases:
  - d = decimal (please only use in test benches)
  - h = hex (use this frequently)
  - b = binary (use this frequently for smaller #'s)
  - o = octal (who thinks in octal?, please avoid)

Numbers in Verilog

- Numbers can have x or z characters as values
  - x = unknown, z = High Impedance
  - 12'h13x // 12-bit number with lower 4-bits unknown
- If size is not specified then it depends on simulator/machine.
  - Always size the number for the DUT verilog
    - Why create 32-bit register if you only need 5 bits?
    - May cause compilation errors on some compilers
- Supports negative numbers as well
  - -16'h3A // this would be -3A in hex (i.e. FFC6 in 2's complement)
  - I rarely if ever use this. I prefer to work 2's complement directly

Identifiers (Signal Names)

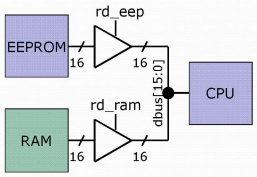
- Identifiers are the names you choose for your signals
- In a programming language you should choose descriptive variable names. In a HDL you should choose descriptive signal names.
  - Use mixed case and/or _ to delimit descriptive names.
    - ✓assign parityErr = ^serial_reg;
    - ✓nxtState = returnRegister;
  - Have a convention for signals that are active low
    - ✓Many errors occur on the interface between blocks written by 2 different people. One assumed a signal was active low, and the other assumed it was active high
    - ✓I use **_n** at the end of a signal to indicate active low
    - ✓rst_n = 1'b0 // assert reset

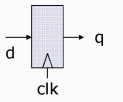
Signal Values & Strength in Verilog

- Signals can have 1 of 4 values

Value:	Meaning:
0	Logic zero, false condition
1	Logic one, true condition
x	Unknown, either uninitialized or in contention
z	High impedance. Not driven, or not connected

Need high impedance for modeling of tri-state busses

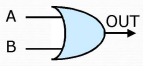




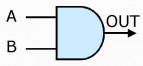
What value does q have prior to any positive clock edge?

It is unknown: **x** (uninitialized)

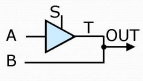
Resolving 4-Value Logic



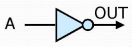
A	B	OUT
0	0	0
0	1	1
1	1	1
0	x	x
0	z	x
1	x	1
1	z	1



A	B	OUT
0	0	0
0	1	0
1	1	1
0	x	0
0	z	0
1	x	x
1	z	x



S	A	T	B	OUT
0	0	z	z	z
0	1	z	x	x
0	x	z	1	1
0	z	z	0	0
1	0	0	1	x
1	0	0	z	0
1	1	1	z	1
1	x	x	z	x
1	z	x	0	x

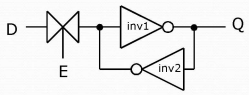


A	0	1	x	z
OUT	1	0	x	x

Signal Values & Strength in Verilog

Strength Level	Meaning:
supply	Strongest (like a supply VDD or VSS)
strong	Strong (default if not specified)
pull	Medium strength (like a pullup device)
weak	Weak, like a keeper (sustainer, leaker)
highz	High impedance

- Concept of drive strength supported to model certain CMOS circuits that utilize contention.




```
not                    inv1 (Q,n1)
not (weak0,weak1) inv2 (n1,Q)
```


Registers in Verilog

- Registers are storage nodes
  - They retain their value till a new value is assigned
  - Unlike a net (wire) they do not need a driver
  - Can be changed in simulation by assigning a new value
- Registers are not necessarily FlipFlops
  - In your DUT Verilog registers are typically FlipFlops
  - Anything assigned in an *always* or *initial* block must be assigned to a register
  - You will use registers in your testbenches, but they will not be FlipFlops

Vectors in Verilog

- Vectors are a collection of bits (i.e. 16-bit wide bus)

```
////////////////////////////////////////////////////////////////
// Define the 16-bit busses going in and out of the ALU //
////////////////////////////////////////////////////////////////
wire [15:0] src1_bus,src2_bus,dst_bus;
```

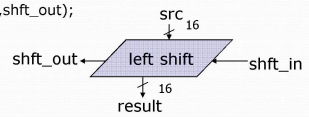
```
////////////////////////////////////////////////////////////////
// State machine has 8 states, need a 3-bit encoding //
////////////////////////////////////////////////////////////////
reg [2:0] state,nxt_state;
```

- Bus ≠ Vector (how are they different?)

Vectors in Verilog

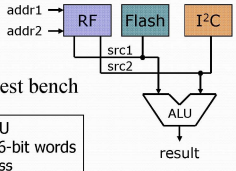
- Can select parts of a vector (single bit or a range)

```
module lft_shift(src,shft_in,result,shft_out);

input [15:0] src;
input shft_in;
output [15:0] result;
output shft_out;

////////////////////////////////////////////////////////////////////////
// Can access 15 LSB's of src with [14:0] selector                    //
// { , } is a process of concatenating two vectors to form one        //
////////////////////////////////////////////////////////////////////////
assign result = {src[14:0],shft_in};

assign shft_out = src[15];          // can access a single bit MSB with [15]

endmodule
```



Arrays & Memories

- Can have multi-dimentional arrays
  - reg [7:0] mem[0:99][0:3]; // what is this?
- Often have to model memories
  - RF, SRAM, ROM, Flash, Cache
  - Memories can be useful in your test bench




```
wire [15:0] src1,src2;       // source busses to ALU
reg [15:0] reg_file[0:31];   // Register file of 32 16-bit words
reg [4:0] addr1,addr2;       // src1 and src2 address
  .
  .
  .
src1 = reg_file[addr1];      // transfer addressed register file
                             // to src1 bus
```


Parameters & Define

- Parameters are useful to make your code more generic/flexible. More later...
- `define statement can make code more readable

```
`define idle =    2'b00;   // idle state of state machin
`define conv =    2'b01;   // in this state while A
`define avg =     2'b10;   // in this state wh                samples
  .
  .
  .
 case (state)
   `idle : if (start_conv)          conv;
           else                     = `idle

   `conv : if                       = `avg;
                                    _state = `conv;
```

Bad Example...Don't Use ` define for state assignment
Use localparam instead. ` define should be truly global thing

Parameters & Define

```
localparam idle =     2'b00;   // idle state of state machine
localparam conv =     2'b01;   // in this state while A2D converting
localparam accum =    2'b10;   // in this state while averaging samples
  .
  .
  .
 case (state)
   idle : if (start_conv) nxt_state = conv;
          else            nxt_state = idle

   conv : if (gt) nxt_state = avg;
          else    nxt_state = conv;
  .
  .
  .
```

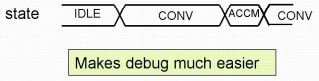
- Read Cumming's paper on use of define & param. Posted on web under "reference"

17

System Verilog…Better Yet

- System verilog adds an enumerated type.

```
typedef enum reg [1:0] { IDLE, CONV, ACCM } state_t;

state_t state, nxt_state;                // declare state and nxt_state signals
```



Makes debug much easier

```
case (state)
  IDLE : if (strt_cnv) begin
       clr_dac = 1;
       clr_smpl = 1;
       nxt_state = CONV;
  end
  CONV : if (!gt) begin
       inc_dac = 1;
       nxt_state = CONV;
    . . .
```

18

Useful System Tasks

- $display ➔ Like printf in C. Useful for testbenches and debug

```
$display("At time %t count = %h",$time,cnt);
```

- $stop ➔ Stops simulation and allows you to still probe signals and debug
- $finish ➔ completely stops simulation, simulator relinquishes control of thread.
- Also useful is `include for including code from another file (like a header file)
- Read about these features of verilog in your text

19

Full Adder: Structural

```
module half_add (X, Y, S, C);

input X, Y ;
output S, C ;

xor SUM (S, X, Y);
and CARRY (C, X, Y);

endmodule
```

```
module full_add (A, B, CI, S, CO) ;

input A, B, CI ;
output S, CO ;

wire S1, C1, C2;

// build full adder from 2 half-adders
half_add PARTSUM (A, B, S1, C1),
         SUM (S1, CI, S, C2);

// … add an OR gate for the carry
or CARRY (CO, C2, C1);

endmodule
```

20

Full Adder: RTL/Dataflow

```
module fa_rtl (A, B, CI, S, CO) ;

input A, B, CI ;
output S, CO ;

// use continuous assignments
assign S = A ^  B ^ CI;
assign C0 = (A & B) | (A & CI) | (B & CI);

endmodule
```

Data flow Verilog is often very concise and still easy to read

Works great for most boolean and even datapath descriptions

21

Full Adder: Behavioral

- Circuit "reacts" to given events (for simulation)
  - Actually list of signal changes that affect output

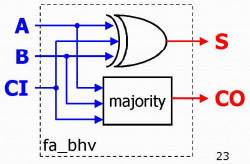
```
module fa_bhv (A, B, CI, S, CO) ;

input A, B, CI;
output S, CO;
reg S, CO;              // assignment made in an always block
                        // must be made to registers
// use procedural assignments
always@(A or B or CI)
  begin
     S = A ^ B ^ CI;
     CO = (A & B) | (A & CI) | (B & CI);
  end
endmodule
```

22

Full Adder: Behavioral

- IN SIMULATION
  - When A, B, or C change, S and CO are recalculated
- IN REALITY
  - Combinational logic – no "waiting" for the trigger
  - *Constantly* computing - think transistors and gates!
  - Same hardware created for all three types of verilog

```
always@(A or B or CI)
  begin
     S = A ^ B ^ CI;
     CO = (A & B) | (A & CI) | (B & CI);
  end
```



23

Structural Basics: Primitives

- Build design up from the gate/flip-flop/latch level
  - Structural verilog is a netlist of blocks and connectivity
- Verilog provides a set of gate primitives
  - and, nand, or, nor, xor, xnor, not, buf, bufif1, etc.
  - Combinational building blocks for structural design
  - Known "behavior"
  - Cannot access "inside" description
- Can also model at the transistor level
  - Most people don't, we won't

24

Primitives

- No declarations - can only be instantiated
- Output port appears before input ports
- Optionally specify: instance name and/or delay (discuss delay later)

```
and N25 (Z, A, B, C);     // name specified
and #10 (Z, A, B, X),
        (X, C, D, E);     // delay specified, 2 gates
and #10 N30 (Z, A, B); // name and delay specified
```

25

Syntax For Structural Verilog

- First declare the interface to the module
  - Module keyword, module name
  - Port names/types/sizes
- Next, declare any internal wires using "wire"
  - wire [3:0] partialsum;
- Then *instantiate* the primitives/submodules
  - Indicate which signal is on which port
- ModelSim Example with ring oscillator

26

Hierarchy

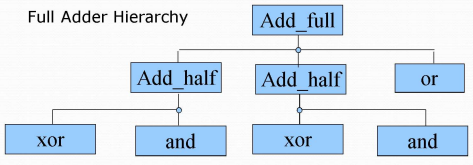
- Any Verilog design you do will be a module
- This includes testbenches!

- Interface ("black box" representation)
  - Module name, ports
- Definition
  - Describe functionality of the block
- Instantiation
  - Use the module inside another module

27

Hierarchy

- Build up a module from smaller pieces
  - Primitives
  - Other modules (which may contain other modules)
- Architecture: typically top-down
- Design & Verification: typically bottom-up



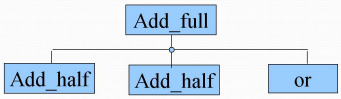
28

Add_half Module


```
module Add_half(c_out, sum, a, b);
    output sum, c_out;
    input a, b;

    xor sum_bit(sum, a, b);
    and carry_bit(c_out, a, b);
endmodule
```

29

Add_full Module




```
module Add_full(c_out, sum, a, b, c_in) ;
    output sum, c_out;
    input a, b, c_in;
    wire w1, w2, w3;

    Add_half AH1(.sum(w1), .c_out(w2), .a(a), .b(b));
    Add_half AH2(.sum(sum), .c_out(w3), .a(c_in), .b(w1));
    or carry_bit(c_out, w2, w3);
endmodule
```

30

Can Mix Styles In Hierarchy!

```
module Add_half_bhv(c_out, sum, a, b);
    output reg sum, c_out;
    input a, b;
    always @(a, b) begin
        sum = a ^ b;
        c_out = a & b;
    end
endmodule
```

```
module Add_full_mix(c_out, sum, a, b, c_in) ;
    output sum, c_out;
    input a, b, c_in;
    wire w1, w2, w3;
    Add_half_bhv AH1(.sum(w1), .c_out(w2),
                     .a(a), .b(b));
    Add_half_bhv AH2(.sum(sum), .c_out(w3),
                     .a(c_in), .b(w1));
    assign c_out = w2 | w3;
endmodule
```

31

Hierarchy And Scope

- Parent cannot access "internal" signals of child
- If you need a signal, must make a port!

Example:
Detecting overflow

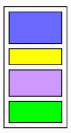
Overflow =
cout XOR cout6

Must output
overflow or cout6!

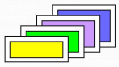
```
module add8bit(cout, sum, a, b, cin);
    output [7:0] sum;
    output cout;
    input [7:0] a, b,cin;
    wire cout0, cout1,… cout6;
    FA A0(cout0, sum[0], a[0], b[0], cin);
    FA A1(cout1, sum[1], a[1], b[1], cout0);
    …
    FA A7(cout, sum[7], a[7], b[7], cout6);
endmodule
```

32

Hierarchy And Source Code



- Can have all modules in a single file
  - Module order doesn't matter!
  - Good for small designs
  - Not so good for bigger ones
  - Not so good for module reuse (cut & paste)
- Can break up modules into multiple files
  - Helps with organization
  - Lets you find a specific module easily
  - Good for module reuse (add file to project)



Structural Verilog: Connections

- Positional or Connect by reference
  - Can be okay in some situations
    - Designs with very few ports
    - Interchangeable input ports (and/or/xor gate inputs)
  - Gets confusing for large #s of ports

```
module dec_2_4_en (A, E_n, D);

input [1:0] A;
input E_n;
output [3:0] D;
  .
  .
  .
```

```
wire [1:0] X;
wire W_n;
wire [3:0] word;

// instantiate decoder
dec_2_4_en DX (X,  W_n,  word);
```

Partial code instatiating decoder

Structural Verilog: Connections

- Explicit or Connect by name method
  - Helps avoid "misconnections"
  - Don't have to remember port order
  - Can be easier to read
  - .<port name>(<signal name>)

```
module dec_2_4_en (A, E_n, D);

input [1:0] A;
input E_n;
output [3:0] D;
  .
  .
  .
```

```
wire [1:0] X;
wire W_n;
wire [3:0] word;

// instantiate decoder
dec_2_4_en DX (.A(X), .E_n(W_n),
.D(word));
```

Partial code instatiating decoder

Empty Port Connections

- Example: module dec_2_4_en(A, E_n, D);
  - dec_2_4_en DX (.A(X), .D(word)); // E_n is high impedence
  - dec_2_4_en DX (.A(X), .E_n(W_n) ,); // Outputs D[3:0] unused.
- General rules
  - Empty input ports => high impedance state (z)
  - Empty output ports => output not used
- Specify all input ports anyway!
  - Z as an input is very bad...why?
- Helps if no connection to output port name but leave empty:
  - dec_2_4_en DX(.A(X[3:2]), .E_n(W_n), .D());

Module Port List

- Multiple ways to declare the ports of a module

```
module Add_half(c_out, sum, a, b);
    output sum, c_out;
    input a, b;
    ...
endmodule
```

```
module Add_half(output c_out, sum,
                input a, b);
    ...
endmodule
```


Module Port List

- Multiple ways to declare the ports of a module (example2)

```
module xor_8bit(out, a, b);
    output [7:0] out;
    input [7:0] a, b;
    ...
endmodule
```

```
module xor_8bit(output [7:0] out,  input [7:0] a, b);
                ...
endmodule
```


Why Know Structural Verilog?

- Code you write to be synthesized will almost all be dataflow or behavioral
- You will write your test bench primarily in behavioral
- What needs structural Verilog?
  - Building hierarchy (instantiating blocks to form higher level functional blocks)
  - There are somethings you can't trust to synthesis, and need to instantiate library gates structurally and use *`don't touch* directives for synthesis.
    - ✓ Intentional clock gating
    - ✓ Cross coupled SR latches
  - Synthesis tools output structural verilog (gate level netlist). You need to be able to read this output.

Timing Controls For Simulation

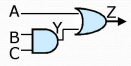
- Can put "delays" in a Verilog design
  - Gates, wires, & behavioral statements
- Delays are useful for Simulation only!
  - Used to approximate "real" operation while simulating
  - Used to control testbench
- SYNTHESIS
  - Synthesis tool IGNORES these timing controls
    - ✓Cannot tell a gate to wait 1.5 nanoseconds
    - ✓Delay is a result of physical properties
  - Only timing (easily) controlled is on *clock-cycle* basis
    - ✓Can tell synthesizer to attempt to meet cycle-time restriction

Zero Delay vs. Unit Delay

- When no timing controls specified: zero delay
  - Unrealistic – even electrons take time to move
  - OUT is updated same time A and/or B change:
    and Ao(OUT, A, B)
- Unit delay often used
  - Not accurate either, but closer...
  - "Depth" of circuit does affect speed!
  - Easier to see how changes propagate through circuit
  - OUT is updated 1 "unit" after A and/or B change:
    and #1 Ao(OUT, A, B);

41

Zero/Unit Delay Example



Zero Delay:
Y and Z change at same "time" as A, B, and C!

Unit Delay:
Y changes 1 unit after B, C

Unit Delay:
Z changes 1 unit after A, Y

Zero Delay

T	A	B	C	Y	Z
0	0	0	0	0	0
1	0	0	1	0	0
2	0	1	0	0	0
3	0	1	1	1	1
4	1	0	0	0	1
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1
8	0	0	0	0	0
9	0	0	1	0	0
10	0	1	0	0	0
11	0	1	1	1	1
12	1	0	0	0	1
13	1	0	1	0	1
14	1	1	0	0	1
15	1	1	1	0	1

Unit Delay

T	A	B	C	Y	Z
0	0	1	0	x	x
1	0	1	0	0	x
2	0	1	0	0	0
3	0	1	1	0	0
4	0	1	1	1	0
5	0	1	1	1	1
6	1	0	0	1	1
7	1	0	0	0	1
8	1	1	1	0	1
9	1	1	1	1	1
10	1	0	0	1	1
11	1	0	0	0	1
12	0	1	0	0	1
13	0	1	0	0	0
14	0	1	1	0	0
15	0	1	1	1	0
16	0	1	1	1	1

42

Types Of Delays

- Inertial Delay (Gates)
  - Suppresses pulses shorter than delay amount
  - In reality, gates need to have inputs held a certain time before output is accurate
  - This models that behavior
- Transport Delay (Nets)
  - "Time of flight" from source to sink
  - Short pulses transmitted
- Not critical for our project, however, in industry
  - After APR an SDF is applied for accurate simulation
  - Then corner simulations are run to ensure design robust

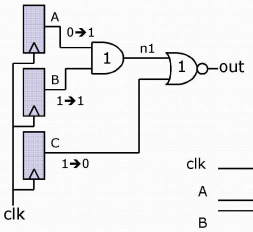
43

Delay Examples

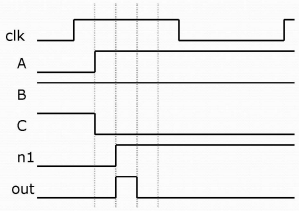
```
wire  #5  net_1;              // 5 unit transport delay

and #4 (z_out, x_in, y_in);   // 4 unit inertial delay
assign #3 z_out = a & b;      // 3 unit inertial delay

wire #2 z_out;                // 2 unit transport delay
and #3 (z_out, x_in, y_in);   // 3 for gate, 2 for wire

wire #3 c;                    // 3 unit transport delay
assign #5 c = a & b;          // 5 for assign, 3 for wire
```

44

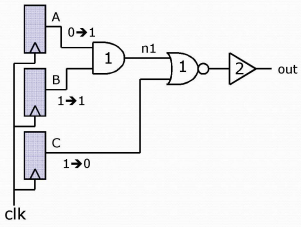
Combinational Logic will Generate Pulses



- Have 3 Flops with a unit delay clk2q
- Feeds simple combinational logic with unit gate delays
- A,B,C make the transitions shown
- What does "out" do?



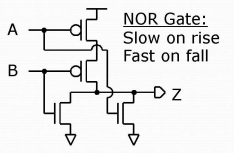
Inertial Delay (tries to model gate delay)

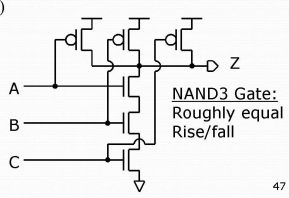


- If we add a buffer with a 2 unit delay to the end of this circuit, now what happens?
- Remember the inertial gate delay will filter any pulse shorter than the specified delay…The pulse is eaten.

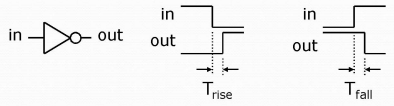
Gate Delay Variation (Verilog Handles This)

- Gates don't have same delay in a rising transition vs falling
- Gates delays are not constant
  - Vary with PVT (Process, Voltage, Temperature)
  - You will hear reference to PVT corners
    - ✓(fff,typ,sss) = (min,typ,max)



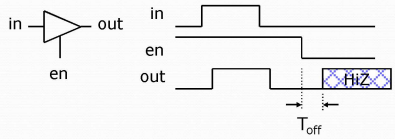


Rise/Fall/Turn Off



Referenced to what the output of the gate is doing

not #(2,1) inv1(out,in); // not gate with 2 unit rise and 1 unit fall



Turn Off delay = Time from cntrl to high impedance

bufif1 #(1,1,2) tri1(out,in,en); // tri-state gate with unit rise/fall and 2 unit off

Min/Typ/Max Delay

- Speed, Speed, Speed ....
  - If you need to ensure speed of your circuit you want to perform your worst case analysis with the longest (max) delays.
- Perhaps more dangerous is a min-delay case. Race condition. If circuit has a race that is not met, it will fail at any clock speed.
  - To ensure your circuit is immune to race conditions (i.e. clock skew) you want to perform your worst case analysis with shortest (min) delays.

49

Min/Typ/Max

- Verilog supports different timing sets.

```
and #(1:2:3) g1(out,a,b);          // 1 ns min, 2ns typical, 3ns max delay
```

- Can specify min/typ/max for rise and fall separate)

```
and #(2:3:4, 1:2:3) g1(out,a,b);   // gate has different rise,fall for min:typ:max
```

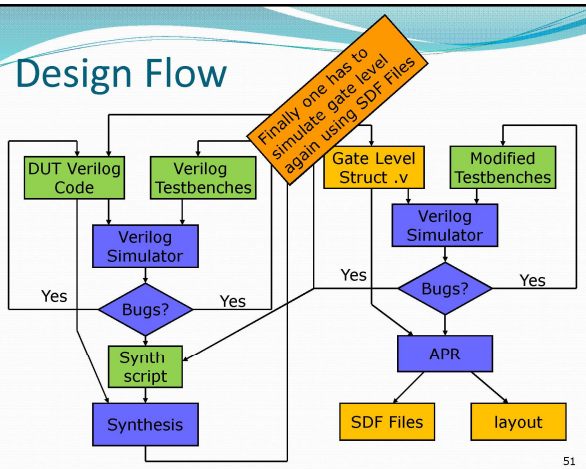
- Selection of timing set can is typically done in simulation environment

```
% verilog test.v +maxdelays
```

Invoke command line verilog engine (like Verilog XL) selecting the maxdelay time set.

50

Design Flow



51