Day 1

Some understanding of SPI, I2C and UART

Day 2

why verilog, emergence of hdl, top down, bottom up approach modules

Numbers lexicon, operators,

Unsized number 23456 is a 32 bit decimal

‘H23 is a 32 bit hexadecimal number

X unknown z high impedance

12’h12x is a 12 bit hexadecimal with least signifcant bit being unknown

Negative number denoted by minus sign before size

-6’d3 an 8 bit negative number stored as 2’s complement of 3

A question mark "?" is the Verilog HDL alternative for z in the context of numbers. The ? is used to enhance readability in the casex and casez statements

string is a sequence of characters that are enclosed by double quotes

It cannot be on multiple lines. Strings are treated as a sequence of one-byte ASCII values.

Identifiers are name given to objects in design made of alpha numeric, underscore, and dolar sign and are case sensitive and cannot start with a number or a $ sign

$sign are reserved for system tasks

Escaped identifiers begin with the backslash ( \ ) character and end with whitespace characters are processed literally. An printable ascii character can be included. White space not part of identifier

two signals of equal strengths are driven on a wire, the result is unknown

two signals of strength strongl and strong0 conflict, the result is an X

Strength level particularly used for accurate modeling signal contention, MOS devices, dynamic MOS, and other low-level device

Only trireg nets can have storage strengths large, medium, and small

default value of a net is z (except the trireg net, which defaults to X )

If a net has no driver, it gets the value z.

net is not a keyword but represents a class of data types such as wire, wand, wor, tri, triand, trior, trireg

Registers retain value until another value is placed onto them.

Do not confuse the term registers in Verilog with hardware registers built from edge-triggered flip-flops in real circuits. In Verilog, the term register merely means a variable that can hold a value

Unlike a net, a register does not need a driver. Verilog registers do not need a clock as hardware registers do. Values of registers can be changed anytime in a simulation by assigning a new value to the register. default value for a reg data type is X.

Vectors can be declared at [high# : low#] or [low# : high#], but the left number in the squared brackets is always the most significant bit of the vector.

Real and integer data types time data type $time

time save-sim-time; // Define a time variable save-sim-time

initial

save-sim-time = $time; // Save the current simulation time

Simulation time is measured in terms of simulation seconds. The unit is denoted by S, the same as real time. the relationship between real time in the digital circuit and simulation time is left to the user. aka Time Scales.

Day 3

l. Multidimensional arrays are not permittegmad in Verilog

Arrays are allowed in Verilog for reg, integer, time, and vector

Arrays are accessed by <array\_name><subscript>

reg [4 : 0] port-id[0: 7] ; //Array of 8port-ids; each port-id is 5 bits wide

Note : while declaring array, index comes after identifier

Note : while defining vectors index comes before identifier

It is important not to confuse arrays with net or register vectors. A vector is a single element that is n-bits wide. On the other hand, arrays are multiple elements that are l-bit or n-bits wide.

Parameters cannot be used as variables. Parameter values for each module instance can be overridden individually at compile time

parameter port-id = 5;//Defines a constant port-id

Module definitions may be written in terms of parameters. Hardcoded numbers should be avoided.

Parameters can be changed at module instantiation or by using the defparam statement

Parameter can be changed either at instantiation using

Module\_name (strength of signal) #(token (which is the value you want to change to some another value))(Instance\_name)(Instance\_range)(port\_list)

Or using defpram

Defpram (instance\_name). (token)

You still have to instantiate the parameter module with some instance name to have the effect

Each character in a string takes up 8 bits or 1 byte

System task begin with $<keyword>

Operations like displaying on the screen, monitoring values of nets, stopping, and finishing

$display without arguments produces a new line

%% gives % and \n allows two line string \t allows tab and

%m gives the instance name of the module in which it was invoked

And will display in the format <top>.<lower instance>.<....

%g is useful for representation %v is for strength

$monitor needs to be invoked only once

If there’s more than one $monitor statement the last one will be active the previous ones are overridden

$monitoron enables monitoring $monitoroff diables it during the simulation

$stop provides stop during simulation and puts the simulation in interactive mode the user can debug and suspend simulation using the $stop statement

$finish terminates it

Compiler directives : ‘<keyword>

‘Define used to define text macros

The “`” “ quotation can be used to interpret the argument as a string.

The “``” quotation can be used to form a signal/variable name by using given argument.

The “ `\” “ quotation is used to replace the argument with an escape sequence.

‘Include allows to include entire contents of a Verilog source file in another Verilog file during compilation

' ifdef and ' timescale, are used frequently

Day 4

One can visualize a port as consisting of two units, one unit that is internal to the module another that is external to the module. The internal and external units are connected

Internally, input ports must always be of the type net. Externally, the inputs can be connected to a variable which is a reg or a net.

Internally, outputs ports can be of the type reg or net. Externally, outputs must always be connected to a net. They cannot be connected to a reg.

Internally, inout ports must always be of the type net. Externally, inout ports must always be connected to a net.

certain output ports might be simply for debugging, and you might not be interested in connecting them to the external signals. You can let a port remain unconnected by instantiating a module

Connecting Ports to External Signals

Two methods

Ordered list- instantiate same was as the port list was ordered

Connectiong ports by name- when there are a lot of ports remembering order is impractical

Gate primitive do not need a name specified and can simply be instantiated.

Buf also exists are a primitive in verilog

bufif1 bufif0 notif1 notif0 can also be used as primitives

bufifl bl (out, in, ctrl) ;

Three types of delay rise fall turn off

Turn off delay is a gate output transition to the high impedance value (2) from another value.

If the value changes to X, the minimum of the three delays is considered.

Three types of delay specifications are allowed. If only one delay is specified, this value is used for all transitions. If two delays are specified, they refer to the rise and fall delay values. The turn-off delay is the minimum of the two delays. If a11 three delays are specified, they refer to rise, fall, and turn-off delay values. If no delays are specified, the default value is zero

// Delay of delay-time for all transitions

and #(delay-time) al(out, il, i2);

// Rise and Fall Delay Specification.

and # (rise-val, fall-val) a2 (out, il, i2) ;

// Rise, Fall, and Turn-off Delay Specification

bufifO #(rise-val, fall-val, turnoff-val) bl (out, in, control);

Verilog provides an additional level of control for each type of delay mentioned above. For each type of delay-rise, fall, and turn-off-three values, min, typ, and max, can be specified Any one value can be chosen at the start of the simulation. Min/typ/max values are used to model devices whose delays vary within a minimum and maximum range because of the IC fabrication process variations.

typ value is the typical delay value that the designer expects the gate to have.

Specifying min, max. Typ delay

// One delay

// if +mindelays, delay= 4

// if +typdelays, delay= 5

// if +maxdelays, delay= 6

and #(4:5:6) al(out, il, i2);

// Two delays

// if +mindelays, rise= 3, fall= 5, turn-off = rnin

// if +typdelays, rise= 4, fall= 6, turn-off = rnin

// if +maxdelays, rise= 5, fall= 7, turn-off = rnin

and #(3:4:5, 5:6:7) a2(out, il, i2);

// Three delays

// if +mindelays, rise= 2 fall= 3 turn-off = 4

// if +typdelays, rise= 3 fall= 4 turn-off = 5

// if +maxdelays, rise= 4 fall= 5 turn-off = 6

and #(2:3:4, 3:4:5, 4:5:6) a3(out, il,i2);

There are other ways of invoking delays in diff simulators/operating systems in verilog-XL for a file test.v

//invoke simulation with maximum delay

> verilog test.v +maxdelays

//invoke simulation with minimum delay

> verilog test.v +mindelays r

//invoke simulation with typical delay

> verilog test.v +typdelays

If you specify delay between two operators as x the output must at minimum be observed at delay1+delay2<time of monitor

Day 5

..= assign ??<drive\_strength>?<delay><list\_of\_assignment> ;

Drive strength is optional and is defined in terms of value sets like strong1 weak0 etc

A specific delay value can also be provided

For continuous assignment The left hand side of an assignment must always be a scalar or vector net or a concatenation of scalar and vector nets

Continuous assignments are always active.

The operands on the right-hand side can be registers or nets or function calls. Registers or nets can be scalars or vectors.

Delay values can be specified for assignments in terms of time units. Delay values are used to control the time when a net is assigned the evaluated value. This feature is similar to specifying delays for gates. It is very useful in modeling timing behavior in real circuits.

// Concatenation. Left-hand side is a concatenation of a scalar

// net and a vector net.

assign {c-out, sum[3:0]) = a[3:0] + b[3:01 + c-in;

Implicit continuous assignment : Instead of declaring a net and then writing a continuous assignment on the net, Verilog provides a shortcut by which a continuous assignment can be placed on a net when it is declared

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. If in1 or in2 changes value again before 10 time units when the result propagates to out, the values of in1 and in2 at the time of recomputation are considered. This property is called inertial delay.

I.E A pulse of width less than the specified assignment delay is not propagated to the output.

Net declaration delay : A delay can be specified on a net when it is declared without putting a continuous assignment on the net.

Calculate\_parity is a functional type operand

On binary operators If any operand bit has a value X, then the result of the entire expression is X.

Negative numbers are represented as 2's complement internally

Designers should avoid negative numbers of the type ass> ' in expressions because they are converted to unsigned 2's complement numbers and hence yield unexpected results.

//Do not use numbers of type <sss> '<base><nnn>

-Id10 / 5

// Is equivalent (2's complement of 10) /5 = (Z3' - 10) /5

// where 32 is the default machine word width.

// This evaluates to an incorrect and unexpected result

Logical operators work on creating true (1) or false (0) or x output

Logical operators take variables or expressions as operands.

If any operand bit is X or z, it is equivalent to X

Relational if any operand is x output is x

Equality operators can have output even if one operator is x for case equality or case inequality

In equality/inequality operator as long as all bit(s) match output is 1 or 0 if not

In bitwise operation if one operand is longer than other the shorter is extended with zeros

Z is treated as an x in bitwise operation

Understand the difference between logical ! and bitwise ~

Reduction works on a single operand

For concatenation use curly brackets to replicate inside curly make another an x time curly bracket on the inside gives the said replication

Conditional operators works like a mux

Conditional operators can be nested

Unary operators have the highest precedenc

Unary

Multiply, Divide, Modulus

Add subtract

Shift

Reduction

Logical

Conditional lowest precedence

By encapsulating functionality inside a module, we can replace the gate-level module with a dataflow module without affecting the other modules in the simulation. This is a very powerful feature of Verilog.

N bit ripple carry adder will have 2n gatelevels propagation time can be a limiting factor

CLA Adder propagation delay is reduced to four gate levels, irrespective of the number of bits in the adder

Day 6

Some review of questions

Day 7

Always and initial cannot be nested

the simulation must be halted inside an initial statement

Procedural assignments update values of reg, integer, real, or time variables. These are unlike continuous assignments

<assignment>

::<value1>=<expression>

The left-hand side of a procedural assignment can be one of the following: A reg, integer, real, or time register variable or a memory element A bit select of these variables (e.g., addr[Ol) A part select of these variables (e.g., addr[31:161) A concatenation of any of the above

Procedural are two types blocking and non blocking

Blocking statment = blocks any parallel assignment using delays

Non blocking <= statemnts with sequential delays don’t block each other

reg2 <= @(negedge clock) in2 A in3; // assignments can be made this way

Using non blocking after concurrent transfer of data in any common event is recommended

d.

Nonblocking assignments can be used effectively to model concurrent data transfers because the final result is not dependent on the order in which the assignments are evaluated

On the downside, nonblocking assignments can potentially cause a degradation in the simulator performance and increase in memory usage.

Zero delay control is a method to ensure that a statement is executed last, after all other statements in that simulation time are executed. This is used to eliminate race conditions. However, if there are multiple zero delay statements, the order between them is nondeterministic.

x= #0 y+1;

The event does not hold any data. A named event is declared by the keyword event. An event is triggered by the symbol ->. The triggering oi the event is recognized by the symbol @.

always

wait (count-enable) #20 count = count + 1;

In the above example, the value of count-enable is monitored continuously. If count-enable is 0, the statement is not entered. If it is logical I, the statement count = count + 1 is executed after 20 time units. If count-enable stays at 1, count will be incremented every 20 time units.

In case expression default has to be provided

Each of statementl, statement2 ..., default-statement can be a single statement or a block of multiple statements. A block of multiple statements must be grouped by keywords begin and end

In case stmt if two or more cases cause same block to be executed then they can be seperated by commas before the :

Casex or casez is used where we are not concerned with the values of certain variable the case x treats all x and z as dont cares case z treats all z as dont cares its used to make operation where desinger only cares about certain values easier

Loops while, for, repeat, forever

In while Any logical expression can be specified with these operators. If multiple statements are to be executed in the loop, they must be grouped typically using keywords begin and end.

for loops are generally used when there is a fixed beginning and end to the loop. If the loop is simply looping on a certain condition, it is better to use the while Ioop.

A repeat construct cannot be used to loop on a general logical expression. A while loop is used for that purpose. A repeat construct must contain a number, which can be a constant, a variable or a signal value. However, if the number is a variable or signal value, it is evaluated only when the loop starts and not during the loop execution.

Forever loop p is equivalent to a while loop with an expression that always evaluates to true

Parallel and sequential blocks

Sequential blocks are executed normally where one statment follows another and delyas are relative to the previous stmts executed

In parallel fork and join statemnets are used to concurrently start a block similar to non blocking assignment

Parallel blocks provide a mechanism to execute statements in parallel. However, it is important to be careful with parallel blocks because of implicit race conditions that might arise if two statements that affect the same variable complete at the same time

All statements start at simulation time 0. The order in which the statements will execute is not known

Sequential and parallel blocks can be nested

Blocks can be given name

Local variables can be declared for the named block

Named blocks are a part of the design hierarchy. Variables in a named block can be accessed by using hierarchical name referencing

Named blocks can be disabled, i.e., their execution can be stopped.

Day 8

Task and functions

Task can enable other tasks and function. Function cant

Task start at non zero sim time. Function start at 0 sim time

Task can have delay, event, timing control. Function can’t.

Task may not have any input, may have multiple input,output, inout arguments

Function must have one input at least and may have multiple input

Task do not return any value but can pass mutliple output through output or inout arguments

Function always return a single vlaue they do not have any output or inout argument

Tasks and function are defined in a local module, and are privy to only that module

Functions are used in combinational ckt that execute at 0 sim time and have only output and are generally used for calculations

Tasks or functions cannot have wires. Tasks and functions contain behavioral statements only

Tasks and functions do not contain always or initial statements but are called from always blocks, initial blocks, or other tasks and functions.

A task can be used when

There are delay, timing, or event control constructs in the procedure. The procedure has zero or more than one output arguments. The procedure has no input arguments.

input, inout, and output used for 1/0 arguments in a task are the same as the keywords used to declare ports in modules, difference being, Ports are used to connect external signals to the module. I/O arguments in a task are used to pass values to and from the task.

There are some peculiarities of functions. When a function is declared, a register with name <name\_of\_function> is declared implicitly inside Verilog

The output is passed back to this register

Function is invoked by specifying name and the input arguments

Return value is placed where the function was invoked

Optional <range\_or\_type>specifies the width of the internal register default is 1 bit

Useful modelling techniques

Assign, deassign, force, release

Identify system tasks for file output, displaying hierarchy, strobing, random number generation, memo y initialization, and value change dump.

Procedural continuous assignments override the effect of regular procedural assignments and are normally used for controlled periods of time.

If an initial/default assignment for a value was made, followed by a new assignment this assigment can be deassigned using deassign statement to revert to the previous default/initlal set of assignments

Example 9-1 for reference

force and release statements are typically used in the interactive debugging process

It is recommended that force and release statements not be used inside design blocks. They should appear only in stimulus or as debug statements.

Ex 9-1 in force release for reference

Force and release can be used on nets as well

You can force a different assignment at the net for a small period of time

If multiple parameters are defined in the module, during module instantiation they can be overridden by specifying the new values in the same order as the parameter declarations in the module

the designer can specify that the particular portion of the code be compiled only if a certain flag is set. This is called conditional compilation

A designer might also want to execute certain parts of the Verilog design only when a flag is set at run time. This is called conditional execution.

The ' ifdef statement can appear anywhere in the design. A designer can conditionally compile statements, modules, blocks, declarations, and other compiler directives. The 'else statement is optional. A maximum of one 'else statement can accompany the 'ifdef. An 'ifdef is always closed by a corresponding ' endif

The conditional compile flag can be set by using the 'define statement inside the Verilog file. In the example above, we could define the flags by defining text macros TEST and ADD-B2 at compile time by using the 'define statement. The Verilog compiler simply skips the portion if the conditional compile flag is not set. A

boolean expression, such as TEST 88 ADD-B2, is not allowed with the ' ifdef statement.

Often, in a single simulation, delay values in one module need to be defined by using certain time unit, e.g., 1 p, and delay values in another module need to be defined by using a different time unit, e.g., 100 ns. Verilog HDL allows the reference time unit for modules to be specified with the ' timescale compiler directive. Usage: ' timescale / The specifies the unit of measurement for times and delays. The specifies the precision to which the delays are rounded off during simulation. Only 1, 10, and 100 are valid integers for specifying time unit and time precision

Output from Verilog normally goes to the standard output and the file verilog.log. It is possible to redirect the output of Verilog to a chosen file.

$fopen(“<name\_of\_file>”);

Usage: <file\_handle>=$fopen(“<name\_of\_file>”);

Compiler directives like ‘ifdef works with ‘define similarly ‘ifndef works in opposition and is included only if the macros is not defined

A macros declared using ‘define can be removed using ‘undef

$random function call returns a 32-bit random number each time it is called. The random number is a signed integer; it can be positive or negative. Therefore, if 2-bit random varialbe is desired, you need a modulo operation as follows.

reg [1:0] R;

R = $random % 2;

Above example will generate random value between -1 to 1. If only positive is needed, use concatenation operator as follows.

reg [1:0] R;

R = {$random} % 2;

Or use $urandom for unsigned numbers

Memory reading

$readmemb (“<file\_name>”,<memory\_name>,<start\_addr>,<end\_addr>);

$readmemb (“<file\_name>”,<memory\_name>); is mandatory

Default is wherever start addr is and finish at wherever end addr is

Ex 9-11 for reference

.dat files can be read to read a file from end to end

Initial

Begin

$readmemb(“<file\_name>”,<mem\_name>);

For (i=0, i<mem\_size; i=i+1)

$display (“memory [%d] = %b”, i, memory[i]);

End

A value change dump (VCD) is an ASCII file that contains information about simulation time, scope and signal definitions, and signal value changes in the simulation run. All signals or a selected set of signals in a design can be written to a VCD file during simulation. Postprocessing tools can take the VCD file as input and visually display hierarchical information, signal values, and signal waveforms

System tasks for vcb are

$dumpvars to dump instance signals

$dumpfile name of vcd file

$dumpon,$dumpoff starting or ending dumping

$dumpall generating checkpoints

Ex to dumpvariables

$dumpvars(1,top) //1 is for hierarchy 1 means dump 1 level below hierarchy

If many other statements are executed in the same time unit as the $display task, the order in which the statements and the $display task are executed is nondeterministic

$strobe is used, it is always executed after all other assignment statements in the same time unit have executed. Thus, $strobe provides a synchronization mechanism to ensure that data is displayed only after all other assignment statements, which change the data in that time step, have executed.

Day 9

static timing verification. Designers first do a pure functional verification and then verify timing separately with a static timing verification tool. The main advantage of static verification is that it can verify timing, orders of magnitude more quickly than timing simulation

Delay models

Distributed delay specified per element basis simply assigning delay to individual gates

Gate value change after the delay value or at individual assign statement

assign #5 e = a & b; stil is distributed cause its on the single element C

Lumped delay

Specified per module basis

Specified as a single delay on the output of the module

Cumulative delay of all paths is lumped in at one location

Basically the entire delay is summarised at one point, normally at the end of the datapath or behav path

Pin to pin delay

For different path to path between element delays

Although pin-to-pin delays are very detailed, for large circuits they are easier to model than distributed delays because the designer writing delay models needs to know only the I/O pins of the module rather than the internals of the module.

Pin-to-pin delays are also known as path delay  
  
Path delay modelling

Specify blocks

A delay between a source (input or inout) pin and a destination (output or inout) pin of a module is called a module path delay.

Path delays are assigned using specify or end specify statements between them form the specify block

specify

(a => out) = 9;

(b => out) = 9;

(c => out) = 11;

(d => out) = 11;

Endspecify

The specify block is a separate block in the module and does not appear under any other block, such as initial or always

every path delay statement has a source field and a destination field.

Usage is

(<source\_field>)=><destination\_field>)=<delay\_value>;

In a parallel connection, each bit in source field connects to its corresponding bit in the destination field. If the source and the destination fields are vectors, they must have the same number of bits; otherwise, there is a mismatch. Thus, a parallel connection specifies delays from each bit in source to each bit in destination.

(a => out) = 9;

//the above statement is shorthand notation

//for four bit-to-bit connection statements

(a[0] => out[0]) = 9;

(a[1] => out[1) = 9;

In full connection its possible to specify delay even among non parallel data

(<source\_field> \*> <destination\_field>) = <delay>;

specify

(a,b \*> out) = 9;

The full connection is particularly useful for specifying a delay between each bit of an input vector and every bit in the output vector when bit width of the vectors is large

//a[31:0] is a 32-bit vector and out[l5:0] is a 16-bit vector

//Delay of 9 between each bit of a and every bit of out

specify

( a \*> out) = 9;

// you would need 32 X 16 = 352 parallel connection

// statements to accomplish the same result! Why?

endspecify

Specparam declared for use inside a specify block Instead of using hardcoded delay numbers to specify pin-to-pin delays, it is common to define specify parameters by using specparam and then to use those parameters inside the specify block

Specify

//define parameters inside the specify block

specparam d-to-q = 9;

specparam clk-to-q = 11;

(d=>q)= d\_to\_q;

(clk=>q)=clk\_to\_q;

Endspecify

Note that specify parameters are used only inside their own specify block. They are not general-purpose parameters that are declared by the keyword parameter

It is recommended that all pin-to-pin delay values be expressed in terms of specify parameters instead of hardcoded numbers

Based on the states of input signals to a circuit, the pin-to-pin delays might change. Verilog allows path delays to be assigned conditionally, based on the value of the signals in the circuit.

A conditional path delay is expressed with the if conditional statement. The operands can be scalar or vector module input or inout ports or their bit-selects or part-selects, locally defined registers or nets or their bit-selects or part-selects, or compile time constants (constant numbers and specify block parameters)

The else construct cannot be used. Conditional path delays are also known as state dependent path delays(SDPD

Specify

If (a) (a=>out)=9;

If (~a)(a=>out)=10;

if (b & C) (b => out) = 9;

if (-(b & C)) (b => out) = 13;

if ({c,d) == 2'b01)

(c,d \*> out) = 11;

if ({c,d) != 2'b01

(c,d \*> out) = 13;

Endspecify

Rise,fall, turn off delays in delay specification

specparam t-rise = 9, t-fall = 13, t-turnoff = 11;

(clk => q) = (t-rise, t-fall, t-turnoff);

//specify six delays.

//Delays are specified in order

//for transitions 0->l, 1->0, 0->z, z->l, 1->z, z->0. Order

//must be followed strictly.

specparam t-01 = 9, t-10 = 13, t-Oz = 11;

specparam t-zl = 9, t-lz = 11, t-zO = 13;

(clk => q) = (t - 01, t-10, t-Oz, t-zl, t-lz, t-zO);

//Delays are specified in order //for transitions 0->l, 1->0, 0->z, z->l, 1-zz, z->0 / / 0->X, X->l, l->X, X->o, X->z, z->X. //Order must be followed strictly.

specparam t-01 = 9, t-10 = 13, t-Oz = 11;

specparam t-zl = 9, t-lz = 11, t-zO = 13;

specparam t-Ox = 4, t-xl = 13, t-lx = 5;

specparam t-xO = 9, t-xz = 11, t-zx = 7;

(clk => q) = (t-01, t-10, t-Oz, t-zl, t-lz, t-zO, t-Ox, t-xl, t-lx, t-xo, t-xz, t-zx );

Min, max, typ delay in delay specifications

specparam t-rise = 8:9:10, t-fall = 12:13:14, t-turnoff = 10:11:12;

(clk => q) = (t-rise, t-fall, t-turnoff) ;

min, typical and max values can be typically invoked with the runtime option +mindelays, +tmdelays, or +maxdelays

Default is the typical delay value

Handling x transitions

Verilog uses the pessimistic method to compute delays for transitions to the X state. The pessimistic approach dictates that if X transition delays are not explicitly specified, Transitions from X to a known state should take maximum possible time Transition from a known state to X should take the minimum possible time

0->X min(t-01, t-Oz) = 9

l->X min(t-10, t-lz) = 11

z->X min(t-z0, t-zl) = 9

X->O max(t-10, t-zO) = 13

X->l max(t-01, t-zl) = 9

X->z max(t-lz, t-Oz) = 11

Timing checks

o set up timing checks to see if any timing constraints are violated during simulation. Timing verification is particularly important for timing critical, highspeed sequential circuits like microprocessors.

three most common timing checks tasks: $setup, $hold, and $width

All timing checks must be inside the specify blocks only.

$setup and $hold checks

Used in seq elements like edge triggered ff

the setup time is the minimum time the data must arrive before the active clock edge.

The hold time is the minimum time the data cannot change after the active clock edge.

reference-event Signal that establishes a reference for monitoring the da ta-even t signal

da ta-event Signal that is monitored for violation

limit Minimum time required for hold of data event

$setup (<data\_event>,<reference\_event>,<limit>);

/data is being checked for violations //Violation reported if Tposedge-clk - Tdata < 3

Specify

$setup(data, posedge clock, 3);

endspecify

$hold (<reference\_event>,<data>,<limit>)

Violation is reported if ( Tdata-event - Treference-event ) < limit.

//Violation reported if Tdata - Tposedge-clk < 5

Specify

$hold(posedge clear, data, 5);

endspecify

$width Sometimes it is necessary to check the width of a pulse.

The system task $width is used to check that the width of a pulse meets the minimum width requirement.

$width(<reference\_event>,<limti>);

reference-event Edge-triggered event (edge transition of a signal)

limit Minimum width of the pulse

The data-event is not specified explicitly for $width but is derived as the next opposite edge of the reference-event signal. Thus, the $width task checks the time between the transition of a signal value to next opposite transition in the signal value.

Violation is reported if ( Tdata-event - Treference-event

//the next negedge of clear is the data-event //Violation reported if Tdata - TClk < 6

Specify

$width (posedge clock, 6 ) ;

Endspecify

Delay back annotation refer to the OVI Standard Delay File (SDF) Format Manual

1. The designer writes the RTL description and then performs functional simulation.

2. The RTL description is converted to a gate-level netlist by a logic synthesis tool.

3. The designer obtains prelayout estimates of delays in the chip by using a delay calculator and information about the IC fabrication process. Then, the designer does timing simulation or static timing verification of the gate-level netlist, using these preliminary values to check that the gate-level netlist meets timing constraints.

4. The gate-level netlist is then converted to layout by a place and route tool. The postlayout delay values are computed from the resistance (R) and capacitance (C) information in the layout. The R and C information

5. The post-layout delay values are back-annotated to modify the delay estimates for the gate-level netlist. Timing simulation or static timing verification is run again on the gate-level netlist to check if timing constraints are still satisfied.

6. If design changes are required to meet the timing constraints, the designer has to go back to the RTL level, optimize the design for timing, and then repeat Step 2 through Step 5.

A standard format called the Standard Delay Format (SDF) is popularly used for back-annotation

Switch Level Modeling

rare cases designers will choose to design the leaf-level modules, using transistors

Verilog HDL currently provides only digital design capability with logic values o, I, X, z, and the drive strengths associated with them. There is no analog capability. Thus, in Verilog HDL, transistors are also known switches that either conduct or are open

Mos switches defined using pmos nmos keyword

nmos nl (out, data, control) ; //instantiate a nmos switch

Cmos c1(out,data, ncontrol, pcontrol)

When the ncontrol signal is 1 and pcontrol signal is o, the switch conducts. If ncontrol signal is o and pcontrol is I, the output of the switch is high impedance value

Bidirection switches Conduct from drain to source

Either side of the device can be the driver signal

Trans,transif0,transif1

The tran switch acts as a buffer between the two signals inoutl and inout2. Either inoutl or inout2 can be the driver signal. The tranifO switch connects the two signals inoutl and inout2 only if the control signal is logical 0.

Transif0 conducts if control signal is 0

tran tl(inout1, inout2);

tranifO t1(inoutl, inout2, control);

Bidirectional switches are typically used to provide isolation between buses or signals.

The power (Vdd, logic I) and Ground (Vss, logic o) sources are needed when transistor-level circuits are designed. Power and ground sources are defined with keywords supply1 and supplyo

Resistive switches have higher source-to-drain impedance than regular switches and reduce the strength of signals passing through them.

rnmos rpmos //resistive nrnos and pmos switches

rcmos //resistive cmos switch

rtran rtranif0 rtranif1

Diff btw regular and resistive switches their source-to-drain impedances and the way they pass signal strengths.

Resistive devices have a high source-to-drain impedance. Regular switches have a low source-to-drain impedance. Resistive switches reduce signal strengths when signals pass through them. exception is that if the input is of strength supply, the output is of strength strong

Rise,fall, turnoff delay in switches same as regular specification

On pmos,nmos,rpmos,rnmos, cmos, cmos

Tran,rtran don’t have any delay specification allowed

Transif0 transif1 can have turnon and turnoff delay

With either zero delay,

same specification for both turnon and off tranifO #(3) T(inout1, inout2, control);

, turn on and turn off specific delay tranifl #(1,2) tl(inout1, inout2, control);

Specify block in switch delays is the same as regular specify blocks for switches

Stimulus is the same way

The make an n to 1 mux you need n cmos devices and n=2^select lines

Cmos ff has 2 cmos and 2 not gates

Clk is control to 2 cmos

Cmos taking the inp D is on if clk is 1

Cmos has a bubble at the control, works if clk is 0

User defined primitives UDP

These primitives are self-contained and do not instantiate other modules or primitives. UDPs are instantiated exactly like gate-level primitives.

Combinational A good example is a 4-to-1 multiplexer

and sequential e latches and flip-flops

Primitive definition

Primitive (udp name)   
 (output terminal name) // only one is allowed

(input terminal name);

//terminal declaration

Output (output terminal name)

Input (input terminal name)

Reg (output terminal name); (optional for seq udp)

// UDP initialization

Initial (output terminal name )= (value);

//udp table

Table

(table entries)

Endtable

Endprimitive

UDPs can take only scalar input terminals (1 bit) multiple inputs allowed

Ome scalar output output appears first multiple output not allowed

Seq udp output must also be declared as reg

Seq udp can be initialized with an initial statement

Udp table can only contain 1,0, or x

Does not handle z values and are treated as x values

Udp are defined at same level as modules, can’t be defined inside modules but can be instantiated inside modules are instantiated like gate primitives

Udp doesn’t support inout ports

The values in a state table entry must appear in the same order as they appear in the input terminal list.

2. Inputs and output are separated by a ":".

3. A state table entry ends with a ";".

All possible outcomes have to be defined in udp where the product of the output is known must be explicitly specified

Udp doesn’t handle the case when all inputs are x

Tables can be shortened using ? that display don’t cares

It is important to note that the state table becomes large very quickly as the number of inputs increases. Memory requirements to simulate UDPs increase exponentially with the number of inputs to the UDP. However, UDPs offer a convenient feature to implement an arbitrary function whose truth table is known, without extracting actual logic and by using logic gates to implement the circuit.

State table using seq udp

Format is slightly diff

(input 1)(input2 ).... (input n): ( current state) : (next state);

The input specification of state table entries can be in terms of input levels or edge transitions.

If a sequential UDP is sensitive to input levels, it is called a level-sensitive sequential UDR If a sequential UDP is sensitive to edge transitions on inputs, it is called an edge-sensitive sequential UDP.

only one initial statement allowed in seq udp

A - in seq udp means it retains the previous value

In edge sensitive udp

10 for a value means on transition of 10 the output value is affected

1x,01,x1, 1?, 0?,x?,?? Type of transition also exist

Only one edge specification is allowed per table. More than one edge specification in a single table entry

(01) (10) 0 : ? : 1 ; //illegal;two edge transitions in an entry

Udp shorthand symbols

? = 0,1,x can’t be used at output field

B = 0,1 can’t be used at output field

* = no change can only be used at output field

R = (0,1) rising edge of signal

F = (1,0) falling edge of signal

p= (01),(0x) or (x1) potential rising edge

n=(10),(x0) or (1x) potential falling edge of signal

\* = (??) any value change of signal

When designing a functional block, it is important to decide whether to model it as a module or as a user-defined primitive.

UDPs model functionality only. They do not model timing or process technology (such as CMOS, TTL, ECL)

The primary purpose of a UDP is to define in a simple and concise form the functional portion of a block. A module is always used to model a complete block that has timing and process technology.

A block can modeled as a UDP only if it has exactly one output terminal

A UDP is typically implemented as a lookup table in memory. As the number of inputs increases, the number of table entries grows exponentially

UDPs are not always the appropriate method to design a block. Sometimes it is easier to design blocks as a module. For example, it is not advisable to design an 8-to-1 multiplexer as a UDP because of the large number of table entries

The UDP state table should be specified as completely as possible. All possible input combinations for which the output is known should be covered. If a certain combination of inputs is not specified, the default output value for that combination will be X.

Level-sensitive entries take precedence over edge sensitive entries. If an edge-sensitive and level-sensitive entry clash on the same inputs, the output is determined by the level-sensitive entry because it has precedence over the edge-sensitive entry.

Day 10

The system has a set of pre defined system routines but sometimes to optimize we need use the custom capabilities of the verilog lang by defining our own system tasks and functions.

To do this, the designers need to interact with the internal representation of the design and the simulation environment in the Verilog simulator. The Programming Language Interface (PLO provides a set of interface routines to read internal data representation, write to internal data representation, and extract information about the simulation environment. User-defined system tasks and functions can be created with this predefined set of PLI interface routines.

As a part of PLI, a new interface called the Verilog Procedural Interface (VPI) has been defined to provide object-oriented access to Verilog HDL object. VPI routines are a superset of the functionality of acc- and tf\_ routines. VPI routines begin with vpi\_

user-defined system tasks can also be invoked in the design and stimulus

The internal design representation is typically in the Verilog simulator proprietary format and is incomprehensible to the designer

PLI can be used to write special-purpose or customized output display routines. Waveform viewers can use this file to generate waveforms, logic connectivity, source level browsers, and hierarchy information.

A user defined task bascially acts like a function in C that is linked to the PLI library

Theoretically you can write any system task to implement whatever

o broad classes of PLI library routines: access routines and utility routines

Access routines provide access to information about the internal data representation; they allow the user C routine to traverse the data structure and extract information about the design. Utility routines are mainly used for passing data across the Verilog/Programming Language Boundary and for miscellaneous housekeeping functions

Acc routines are used to Read information about a particular object from the internal data representation Write information about a particular object into the internal data representation

Access routines use the concept of a handle to access an object. Handles are predefined data types that point to specific objects in the design. Any information about the object can be obtained once the object handle is This is similar to the concept of file handles for accessing files in C programs. An object handle identifier is declared with the keyword handle.

Logic synthesis

logic synthesis is the process of converting a high-level description of the design into an optimized gate-level representation, given a standard cell library and certain design constraints

Remember that we are providing a cycle-by-cycle RTL description of the circuit. Hence, there are restrictions on the way these constructs are used for the logic synthesis tool

For example, the while and forever loops must be broken by a @ (posedge clock) or @ (negedge clock) statement to enforce cycle-by-cycle behavior and to prevent combinational feedback

Also, the initial construct is not supported by logic synthesis tools. Instead, the designer must use a reset mechanism to initialize the signals in the circuit It is recommended that all signal widths and variable widths be explicitly specified

Only operators such as a== and ! == that are related to X and z are not allowed, because equality with X and z does not have much meaning in logic synthesis

it is recommended that you use parentheses to group logic the way you want it to appear. If you rely on operator precedence, logic synthesis tools might produce undesirable logic structure

Note a circuit produced by assign a[3:0]=[3:0]b+[3:0]c|[3:0]d ;

Will create 4 identical circuits of assign a=b+c|d;

conditional operator ? is used, a multiplexer circuit is inferred.

If else, case stmt also lead to a mux being synthesized

For loops

The for loops can be used to build cascaded combinational logic. For example, the following for loop builds an &bit full adder.

c=c\_cin

For (i=0; i<=7; i=i+1)

{

c,sum[i]=a[i]+b[i]+c[i];

}

c\_out=c;

c=ab+bc+ca

sum=a^b^c;

{c,sum}=a+b+cin;

The always statement can be used to infer sequential and combinational logic. For sequential logic, the always statement must be controlled by the change in the value of a clock signal clk.

Always @ (posedge clk)

q=d;

Infers an edge sensitive d ff

However

using always @(clk or d)

If (clk)

q=d;

Infers a level sensitive d ff

For combinational logic, the always statement must be triggered by a signal other than the clk, reset, or preset

Functions infer the combinational logic that was inferred on them

Day 11

Design constraints such as area, timing, and power are not considered in the translation proces

Various technology independent boolean logic optimization techniques are used. This process is called logic optimization

Day 12

For tech dependent optimization the synthesis tool takes the internal representation and implements the representation in gates, using the cells provided in the technology library. In other words, the design is mapped to the desired target technology.

Inverse relationship exists between area and timing

Timing improvement requires parallelized ckt, which means larger ckt.

To build smaller ckt, timing has to be compromised

For very high speed circuits like microprocessors, vendor technology libraries may yield nonoptimal results. Instead, design groups obtain information about the fabrication process used by the vendor, for example, 0.65 micron CMOS proces

Functional verification

Identical stimulus is run with the original RTL and synthesized gate-level descriptions of the design. The output is compared to find any mismatches.

The same stimulus is applied to both the RTL description in Example 14-1 and the synthesized gate-level description in Example 14-2, and the simulation output is compared for mismatches. However, there is an additional consideration. The gate-level description is in terms of library cells VAND, VNAND, etc. Verilog simulators do not understand the meaning of these cells. Thus, to simulate the gate-level description, a simulation library, abc-100.v, must be provided by ABC Inc. The simulation library must describe cells VAND, VNAND, etc., in terms of Verilog HDL primitives and, nand, etc

The stimulus is applied to the RTL description and the gate-level description. A typical invocation in Verilog-XL is shown below

//Apply stimulus to RTL description

> verilog stimu1us.v mag-c0mpare.v

//Apply stimulus to gate-level description.

//Include simulation library "abc-100.v" using the -v option

> verilog stimu1us.v mag-c0mpare.gv -v abc-100.v

One technique is to write a high-level architectural description in C++. The output 298 Verilog HDL: A Guide to Digital Design and Synthesis obtained by executing the high-level architectural description is compared against the simulation output of the RTL or the gate-level description

The gate-level netlist is typically checked for timing by use of timing simulation or by a static timing verifier. If any timing constraints are violated, the designer must either redesign part of the RTL or make trade-offs in design constraints for logic synthesis.

Use meaningful names for signals and variables

Avoid mixing positive and negative edge-triggered flip-flops Mixing positive and negative edge-triggered flip-flops may introduce inverters and buffers into the clock tree. This is often undesirable because clock skews are introduced in the circuit.

Use basic building blocks vs. Use continuous assign statements

. Continuous assign statements are very concise way of representing the fun&ionality and generally do a good job of generating random logic. However, the final logic structure is not necessarily symmetrical. Instantiation of basic building blocks creates symmetric designs, and the logic synthesis tool is able to optimize smaller modules more effectively. However, instantiation of building blocks is not a concise way to describe the design; it inhibits retargeting to alternate technologies, and generally there is a degradation in simulator performance.

Instantiate multiplexers vs. Use if-else or case statements

ifelse and case statements are frequently synthesized to multiplexers in hardware. If a structured implementation is needed, it is better to implement a block directly by using multiplexers, because if -else or case statements can cause undesired random logic to be generated by the synthesis tool. Instantiating a multiplexer gives better control and faster synthesis, but it has the disadvantage of technology dependence and a longer RTL description. On the other hand, if-else and case statements can represent multiplexers very concisely and are used to create technology-independent RTL descriptions.

Use parentheses to optimize logic structure The designer can control the final structure of logic by using parentheses to group logic. Using parentheses also improves readability of the Verilog description.

//translates to 3 adders in series

out=a+b+c+d;

//translates to2 adders in parallel with one final adder to sum results

out = (a + b) + (C + d) ;

Use arithmetic operators \*, l, and % vs. Design building blocks

Multiply, divide, and modulo operators are very expensive to implement in terms of logic and area

Be careful with multiple assignments to the same variable Multiple assignments to the same variable can cause undesired logic to be generated. The previous assignment might be ignored, and only the last assignment would be used.

//two assignments to the same variable

always @ (posedge clk)

if(load1) q <= al;

always @ (posedge elk)

if(load2) q <= a2;

The synthesis tool infers two flip-flops with the outputs anded together to produce the q output. The designer needs to be careful about such situations.

Define if-else or case statements explicitly Branches for all possible conditions must be specified in the if-else or case statements. Otherwise, level-sensitive latches may be inferred

instead of multiplexers. R

Horizontal partitioning Use bit slices to give the logic synthesis tool a smaller block to optimize. This is called horizontal partitioning. It reduces complexity of the problem and produces more optimal results for each block. For example, instead of directly designing a 16-bit ALU, design a 4-bit ALU and build the 16-bit ALU with four 4-bit ALUs.

downside of horizontal partitioning is that global minima can often be different local minima. Thus, by use of bit slices, each block is optimized individually, but there may be some global redundancies that the synthesis tool may not be able to eliminate.

Vertical partitioning implies that the functionality of a block is divided into smaller submodules. This is different from horizontal partitioning. In horizontal partitioning, all blocks do the same function. In vertical partitioning, each block does a different function. Assume that the 4-bit ALU described earlier is a fourfunction ALU with functions add, subtract, shift right, and shift left.

For logic synthesis it is important to create hierarchy by partitioning a large block into separate functional sub-blocks. A design is best synthesized if levels of hierarchy are created and smaller blocks are synthesized individually. Creating modules that contain a lot of functionality can cause logic synthesis to produce suboptimal designs

Parallelizing the design structure

Converting seq into parallel structure uses more resources providing more speed

Storage strengths are used to model charge storage in trireg type nets

If two signals with same known value and different strength drive the same net, the signal with the higher strength wins.

When two signals with opposite value and same strength combine, the resulting value is X.

tri is denotes nets that have multiple drivers

module mux(out, a, b, control) ;

output out;

input a, b, control;

tri out;

wire a, b, control;

bufifO bl (out, a, control) ; //drives a when control = 0; z otherwise

bufifl b2 (out, b, control) ; //drives b when control = 1; z otherwise

Endmodule

The net is driven by bl and b2 in a complementary manner. When bl drives a, b2 is tristated; when b2 drives b, bl is tristated. Thus, there is no logic contention. If there is contention on a tri net, it is resolved by using strength levels. If there are two signals of opposite values and same strength, the resulting value of the tri net is X.

Trireg

Keyword trireg is used to model nets having capacitance that stores values. The default strength for trireg nets is medium.

Driven state-At least one driver drives a 0, 1, or X value on the net. The value is continuously stored in the trireg net. It takes the strength of the driver. Capacitive state. All drivers on the net have high impedance (z) value. The net holds the last driven value The strength is small, medium, or large (default is medium)

trireg (large) out;

wire a, control;

bufifl (out, a, control);

Keywords tri0 and tri1 are used to model resistive pulldown and pullup devices. A trio net has a value o if nothing is driving the net. Similarly, tril net has a value 1 if nothing is driving the net. The default strength is pull.

When there is logic contention, if we simply use a tri net, we will get an X. This could be indicative of a design problem. However, sometimes the designer needs to resolve the final logic value when there are multiple drivers on the net, without using strength levels

Nets wand perform the and operation on multiple driver logic values. If any value is 0, the value of the net wand is 0. Net wor performs the or operation on multiple driver values. If any value is I, the net wor is I. Nets triand and trior have the same syntax and function as the nets wor and wand. The example below explains the function.

Please test the PLI routines