Practical Considerations

In text book or ideal world

Signals change state or propagate though combinational or sequential networks

In zero time

At every turn real world signals encounter physics of practical devices

Thousands of dead physicists are there just waiting for us

If we are to design and build reliable and robust systems

We must understand when, where, how, and why

Such physical affects occur

Once we gain such understanding

Design around or compensate for potential problems

Can incorporate them into our models

To determine

How they are affecting our system

If our design approach for mitigating affects

Has proven successful

We will first look at combinational logic

Then move on to sequential circuits

Part 1 Timing in Combinational Logic – introduction

Now want to examine techniques for modeling

Real world affects

Such affects focus (result primarily from) on

Consequences of inherent parasitic devices

Such devices comprise passive components

R, L, C

Devices present in

Systems built of discrete components

Programmable logic devices

Internal to device

Getting onto or off of device

LSI and VLSI circuits

Timing and Delays

When modeling designs to study and to test real world behaviour

Must understand and work with physical world
That alters behaviour from the ideal
Such effects include signal delays

In our studies of combinational logic
We find several fundamental timing issues we need to consider
  Rise / Fall Times
  Propagation delay
  Race Conditions
These will carry forward to work with sequential circuits

We will also examine
  Potential root causes for such issues
  Affects on our circuits
  Some basic models

In later section
  Will see how to incorporate such affects into Verilog model

Let’s briefly review each of timing issues

**Fundamental Attributes**
  Textbook waveforms
    Change state in zero time
    Move through system at infinite speed
  Real-world signals not quite as efficient
Will begin our study with look at simple delays
  Such delays are first step away from textbook behaviour

**Rise Time, Fall Time, and Turn-Off Delays**
  These delays give measure of time signal takes
    To change state
      From 0 to 1 or 1 to 0
      Tristate part to cease driving

Consider the following signal
  We measure rise and fall time at
10% and 90% points

Time called rise time and fall time

Two times not always symmetrical

Specify

\[ \tau_r \text{ and } \tau_f \text{ or } \]

\[ \tau_{\text{rise}} \text{ and } \tau_{\text{fall}} \]

Problem

If rise / fall times too long

Gate no longer acts as switch instead becomes poor amplifier

Enters what is called metastable region

Will discuss in more detail shortly

Modeling Rise Time, Fall Time, and Turn-Off Delays

To incorporate the time required for a signal to change state

Verilog supports including device rise time, fall time into model

The syntax for these is given as

```
Syntax
# (rise time, fall time) device;
```

Illustrated in following code fragment

```
and #(3,4)myAnd(out, in0, in1);
```

When working with busses or simply individual signals

Connected utilizing tristate devices

Time for device output to turn-off once control is deasserted

Considered important

When working with physical devices

Turn-off time critical when trying to switch

From one device driving to a different one driving
Having multiple devices simultaneously driving bus line
  Creates drive conflict
    With resulting increase in current demand
When current supplied by power source
  Large current demand in short time
  Creates voltage transients in
    Power and ground system
  Such transients manifest as noise
    May damage the parts.

*Turn-off time* is incorporated into model
  Through simple extension of
    *Rise* and *fall time* model

The syntax for all three is given as

```
Syntax
  # (rise time, fall time, turn-off delay) device;
```

Illustrated in following code fragment

```
bufif0 #(3,4,5)myBuf0(out, in, ctl);
```

**Propagation Delay and Path Delay**

Let’s now examine movement of signals
  Along conducting path
    Can be along bus or single signal line
    Inside or outside of logical device

Look at how we model affects of real-world
**Propagation Delay**
  Time required for the affects of input signal
    To be reflected in a corresponding change in a device’s output
  Called the *propagation delay* of the part
In response to an input signal
    Time for the output to change from
    Logical 0 to a logical 1
    Often different from a state change in opposite direction

Propagation delay can easily be observed in an inverter
    If a high going signal is set as the input into the device
    Output will change to low sometime later

We measure that time at the 50% point of two signals
The parameters how we measure them and their asymmetry
Illustrated in accompanying figure

Specify delay as
    $\tau_{\text{dlh}}$ and $\tau_{\text{dhl}}$ or
    $\tau_{\text{pdlh}}$ and $\tau_{\text{pdhl}}$
Values vary with
    Logic family
    Load on device
    Medium through which signal propagating
Logic families ranking lowest to highest
    ECL
    BiCMOS
    ALS TTL
    CMOS
    ALS TTL and CMOS are comparable

Propagation Delay Models
    When modeling temporal behavior in digital logic
Can must pose the question
    If a signal is entered into a combinational net or sequential device and
    State of the signal changes several times
    Before the initial value can propagate through the net
    What is the affect on the output
Two different propagation delay models can be used
    To study the behavior of combinational devices
For our initial look we propose two models
That model one aspect of delay
Defined as transport delay and inertial delay
In essence these models reflect
The bandwidth of the signaling path

Circuit’s behavior is same in both cases
If the input makes a single state change (and no others)
Before the output propagates to the output

*Transport Delay*
Under the transport delay model
Changes in input are seen by the output
Following the specified delay
Model
Permits all signals to pass down propagation path
Regardless of duration
Assumes signaling path
Has infinite bandwidth

*Inertial Delay*
An inertial delay model acts somewhat like low pass filter
Signals with duration shorter than some value
Do not pass down propagation path
Carrying bandwidth metaphor forward
Signals with frequency higher than bandwidth of signaling path
Not permitted to pass

Model refines the notion of delay
To attempt to account for the physical movement
Of electronic charge within a device
Model states
If the duration of a signal is less than a specified minimum
Signal state change will not be reflected in the device output
Such a duration is typically set to be less than or equal to
Propagation delay of the device
Following diagrams illustrates the two types of delay models

From a high level first

Propagation within a simple device

Observe that the short duration state change

Does not appear in the output waveform for the inertial model

Most practical systems behave

According to the transport model

The inertial and transport models attempt to model

Signaling path’s ability to handle

High rates of change of propagating signal

Path Delay

Such models do not consider transport velocity

Rate at which signal propagates down path

To accommodate transport velocity

Take

Distributed

Lumped

View of the path
When analyzing propagation delay must consider the path

Does signal pass through
Single part
Multiple parts
Multiple systems

We're assuming a part is
Any passive or active component
Logic gate or wire for example

Delay through module between
Source pin – in or inout
Destination pin – out or inout
Called path delay

Path delays assigned using specify block
Such a block delimited by keywords

\[
\text{specify} \\
\text{endspecify}
\]

\[
\begin{align*}
(A \Rightarrow \text{Out}) &= 30; \\
(B \Rightarrow \text{Out}) &= 12; \\
(C \Rightarrow \text{Out}) &= 18; \\
(D \Rightarrow \text{Out}) &= 15; \\
(E \Rightarrow \text{Out}) &= 6; \\
(F \Rightarrow \text{Out}) &= 9;
\end{align*}
\]

Endspecify
and \(a0(G, A, B)\);
or \(a0(H, C, D)\);
and \(a1(J, E, F)\);
or \(a3(\text{Out}, H, J, G)\);

Pin to Pin Delay
Consider accompanying simple circuit
There is different path and correspondingly
Potentially different delay
From each input ‘pin’
To output ‘pin’

Can capture these delays and differences
As shown in code fragment
Specify block is separate block in a module
Does not have to be in
initial or always block
Model above models pin-to-pin delay
**Path Delay**

As we saw earlier
Can model bus as vector of signals
Can also model in similar way
As simple extension of pin-to-pin delay

When working with vectors of signals
Between source and destination
Verilog supports two basic models
Parallel path
Full connection path

Illustrated in following graphics

**Parallel Path**
With parallel path configuration
Each signal within source vector
Connects to single signal within destination vector

Within specify block
Expression (in4 => out4) = delay;
Equivalent to
Aggregate expression
   (in4[3] => out4[3]) = delay;
   (in4[2] => out4[2]) = delay;
   (in4[1] => out4[1]) = delay;
   (in4[0] => out4[0]) = delay;
Delay specification
Quantifies or specifies delay from source signal
Along path to single signal at destination

*Full Path*
With full path configuration
Each signal within source vector
“Connected” to each signal within destination vector
Connection implicit rather than explicit
Consider
Source vector of four signals: \((s_0, s_1, s_2, s_3)\)
Destination vector *out* of 3 signals: \((o_0, o_1, o_2)\)

Within specify block
Operator \( *> \) distinguishes full connection semantics
Can write
Expression: \((s_0, s_2 *> out) = delay1;\)
\((s_1, s_3 *> out) = delay2;\)
Expressions interpreted as
Delay from \(s_0\) or \(s_2\)
Through any logic block
To any output signal within output vector
Has delay equal to \(delay1\)
Delay from \(s_1\) or \(s_3\)
Through any logic block
To any output signal within output vector
Has delay equal to \(delay2\)

Delay specification
Quantifies or specifies delay from source signal
Along path to any signal at destination

*Local Parameter Declarations*
Earlier learned to use parameters
Instead of magic numbers
Notion extended to specify block
Language supports declaration of parameters
Within specify block using keyword `specparam`
Can modify above example by making local declaration
Within specify block

```vhs
specify
specparam A-to-OutDly = 10;
(A => Out) = A-to-OutDly;
(B => Out) = 12;
(C => Out) = 18;
(D => Out) = 15;
(E => Out) = 6;
(F => Out) = 9;
endspecify

and a0(G, A, B);
or o0(H, C, D);
and a1(J, E, F);
or a3(Out, H, J, G);
```

Conditional Path Delays
Path delays from input to output
Can be conditioned on logical state of
Individual signals
Combinations of input signals

```vhs
specify
if (in0) (in0 => out0) = 5;
if (~in0) (in0 => out0) = 10;
if (in0 & in1) (in0 => out0) = 15;
if (~(in0 & in1)) (in0 => out0) = 20;
endspecify
```

Consequences of Delays - Race Conditions and Hazards
The affects of delays are many and varied
Important to understand them
To be able to design and implement
Robust and reliable circuit or system
Simplest consequence of delays called race condition

A race condition occurs when several signals
May arrive at a circuit or common decision point (AND gate, OR gate, etc.)
At different times because of different path delays
In a circuit or system
A *critical* race occurs
When the state or output of circuit depends upon
Order in which several associated inputs arrive at the decision point

A *non-critical* race occurs
When the state or output of circuit does not depend upon
Order in which several associated inputs arrive at the decision point.

A *hazard* exists in any circuit
That has the possibility of producing an incorrect output
That we call a decoding spike or glitch

There are two types of hazards
- Static,
- Dynamic

A static hazard exists
When there is a possibility of a circuit's output producing a glitch
As the result of a race between two or more input signals
When we expect it to remain at a steady level
Based on a static analysis of the circuit function
We have a
*Static-0* hazard
When our circuit can produce an erroneous 1
When the output should be a constant 0
*Static-1* hazard under the opposite condition

A *dynamic* hazard exists
When our circuit output
May erroneously change more than once
From a single input transition

Let’s look at simple example
This circuit has a static 1 hazard
The following circuits give examples of each of these hazards.

**Static - 0 Hazard**

**Static - 1 Hazard**

**Dynamic Hazard**

**Look for the Guilty**

Resistors, Capacitors, Wires

Resistor

At the physical level we have

\[
R = \frac{\rho l}{A}
\]

As \( l \) increases / decreases

- \( R \) increases / decreases

- \( A \) increases / decreases

- \( R \) decreases / increases

Discrete Component Model

We model the resistor as illustrated

Also models wire

Intuitively

At DC - We speak of resistance

- \( L \) is short

- \( C \) is open

At AC - we now speak of impedance

- \( L \) has finite non-zero impedance

- \( C \) is finite impedance
\[ Z(\omega) = LS + R\|\frac{1}{CS} \]
\[ = LS + \frac{R}{RCS + 1} \]
\[ |Z(\omega)| = \sqrt{\frac{R^2 \left(1 - L\omega^2\right)^2 + (\omega L)^2}{1 + (RC\omega)^2}} \]

Checking the boundaries

For \( \omega = 0 \),
\[ |z(\omega)| = R \]
\( \omega \to \infty \)
\[ |z(\omega)| = L\sqrt{RC\omega} \]
Observe magnitude of \( Z \)

Begin to increase again

Because of the inductive and capacitive elements

We get a phase shift

The value is given by

\[ \phi = \phi_1 - \phi_2 \]
\[ \phi_1 = \tan^{-1}\left(\frac{L}{R} \left(\frac{\omega}{1 - L\omega^2}\right)\right) \]
\[ \phi_2 = \tan^{-1}(RC\omega) \]

Capacitors

At the physical level we have

\[ C = \frac{\epsilon A}{d} \]
As \( A \) increases / decreases
\( C \) increases / decreases
\( d \) increases / decreases
\( C \) decreases / increases
In a printed circuit
  Two signal traces can form the two plates of a capacitor
That capacitor appears as a parasitic device
  Between the two signal traces
In accompanying drawing
As we continue to reduce the size of a design
  Those traces are moved closer and closer together
  The distance between the plates decreases
  Thereby increasing the associated capacitance

Because the voltage across a capacitor cannot change instantaneously
  Portion of the signal originating at the logic gate on the left
  Will be coupled into the lower trace as noise
Routing any signal trace through
  Microprocessor, gate array, or programmable logic devices
  Going to produce the same affect to varying degrees

Discrete Component Model
  We model the capacitor as illustrated
Intuitively
  At DC
    L is short
    C is open
    R is finite

  At AC - the capacitor has an impedance
    L has finite non-zero impedance
    C is finite non-infinite impedance
    R is finite

\[
z = \frac{1}{Cs} + Ls + R
\]

\[
|z(\omega)| = \left( \frac{1 - LC\omega^2}{(C\omega)^2} + (RC\omega)^2 \right)^{1/2}
\]
Because of the inductive element
We get a phase shift
The value is given by

\[ \phi = \phi_1 - \phi_2 \]
\[ \phi_1 = \tan^{-1}\left( \frac{RC\omega}{1 - LC\omega^2} \right) \]
\[ \phi_2 = \frac{\pi}{2} \]

Logic Circuits and Parasitic Components
Now examine the affect of parasitic components
On the behavior of a logic circuit
We’ll use the basic logic circuit in accompanying figure for this analysis
Results extend naturally to more complex circuits
Our digital system comprises two logic devices that we model using two inverters
The source produces a typical digital signal
Such as one might find originating from
Logic gate, a bus driver
Output of more complex device such as FPGA or microprocessor
The receiver of the signal is any similar such device

First Order RC
We’ll begin with a first order model for the devices
Environment and the wire interconnecting the two devices
As shown in accompanying figure
This basic model plays a significant role
In first order analyses of typical digital circuit behavior
Vin and Vout

Related by simple voltage divider

\[
V_{out} = \left( \frac{1}{\frac{1}{Cs} + \frac{1}{RCs}} \right) V_{in}
\]

\[
= \left( \frac{1}{RCs + 1} \right) V_{in}
\]

For Vin a step

\[
V_{out} = \frac{V_{in}}{s} \left( \frac{1}{RCs + 1} \right)
\]

\[
V_{out} = V_{in} \left( \frac{1}{s} + \frac{1}{s + \frac{1}{RC}} \right)
\]

\[
V_{out} = V_{in} \left( 1 - e^{-\frac{t}{RC}} \right)
\]

Tristate Drivers

The tristate driver is commonly used in bus based applications

To enable multiple different data sources

Onto a system bus

Let’s analyze one signal of such a bus

Examine how the parasitic device can affect performance

The bus signal is presented in accompanying diagram
The capacitor models
Bus, package, and adjacent path parasitic capacitances
This value will be approximately 50pf and
Typical pull-up resistor is 10K for TTLS logic
The parasitic contributions from the interconnecting wire
Do not contribute in this analysis.

When the sending device is enabled and transmitting data
Bus capacitance and wire parasitics contribute as discussed earlier

In the circuit in the diagram
Driver has been disabled and is entering the tristate region
We model that turn-off as we did earlier

When the driving device is disabled
The driven bus is now under the control of the pull-up resistor
We model that circuit in the accompanying diagram

If the state of the bus was a logical 0 when the tristate device was disabled
The resistive pull-up voltage acts as a step input into the circuit
The signal, Vout – input to the driven device
Will increase according to the earlier equations

The equation and timing diagram follow our previous analysis

\[ V_{out} = V_{in} \left( 1 - e^{-\frac{t}{RC}} \right) \]
Second Order Series RLC

We now extend first-order interconnect model
By adding parasitic inductance

With addition of inductor
Now have a second-order circuit
Diagram shows
Extended model on left
Circuit model on right

\[
\begin{align*}
V_{\text{out}} &= \frac{1}{\frac{1}{Cs} + \frac{R}{R + LS + \frac{1}{Cs}}} V_{\text{in}} \\
&= \frac{V_{\text{in}}}{LC} \left( \frac{1}{s^2 + \frac{R}{L}s + \frac{1}{LC}} \right)
\end{align*}
\]

Expression in denominator on right hand side
Can be written as the characteristic equation
Thus
\[
V_{\text{out}} = \frac{V_{\text{in}}}{LC} \left( \frac{1}{s^2 + 2\xi\omega_n s + \omega_n^2} \right)
\]

\[\omega_n = \frac{1}{\sqrt{LC}}\]
\[\xi = \frac{R}{2 \left( \frac{L}{C} \right)^{1/2}}\]
Recall the value of $\xi$ determines if circuit is

- Underdamped $\xi < 1$
- Critically damped $\xi = 1$
- Overdamped $\xi > 1$

$$Q = \frac{(L/C)^{1/2}}{R} = \frac{\omega_n}{R} = \frac{1}{2\xi}$$

For $V_{in}$ a step

$$v(t) := 5 - 5 \cdot \exp \left( -\frac{w}{2} \frac{w}{Q} t \right) \sin \left( \frac{1}{2} \sqrt{\frac{4Q^2 - 1}{Q}} t \right) - 5 \cdot \exp \left( -\frac{1}{2} \frac{w}{Q} t \right) \cos \left( \frac{1}{2} \sqrt{\frac{4Q^2 - 1}{Q}} t \right)$$

We can plot the behaviour of the circuit as

---

**Practical Considerations – Part 2 Timing in Latches and Flip-Flops**

For combinational logic devices

- Major timing concerns focused on the delay of signals propagating through the devices
- Timing relationship between input data and the gate in latches
  - Clock in flip-flops
- Introduces the notions of *setup time* and *hold time*

**Setup time**

- Specifies how long input signals must be present and stable before the gate or clock changes state

**Hold time**

- Specifies how long an input signal must remain stable after a specified gate or clock has changed state
Gated Latches
Setup and hold time relationships
Illustrated in the adjacent diagram
For a gated latch
That is enabled by a logical 1 on the Gate
Specification for times
Given at 50% point of each signal

The setup and hold times
Permit incoming signals to
Propagate through any input logic
Initiate and complete the appropriate state changes
For any internal memory elements
Times are designated as $\tau_{\text{setup}}$ or $\tau_{\text{su}}$ and $\tau_{\text{hold}}$.

If the setup time constraints are not met
That is if the input data changes within the setup window
Behavior of the circuit is undefined
Input
May or may not be recognized
Output may enter a metastable state
In which it oscillates for an indeterminate time
Illustrated in the accompanying diagram
As device’s internal components attempt to reach a stable state
Such oscillation can persist for several nanoseconds.

Flip-Flops
The accompanying diagram in graphically illustrates
The setup and hold time relationships
For a positive edge triggered flip-flop
Specification for times
Given at 50% point of each signal
The need for and consequences of violating the Setup and hold time constraints
Same as those for the gated latch
Propagation Delays

In combinational circuits

Propagation delay specifies interval

Following a change in state of an input signal

Effect of that change appearing on the output

Such an interval is characterized by

*minimum, typical, and maximum* values

In storage devices

Measurement is made with respect to

The *causative edge* of the clock or strobing signal

Following diagram illustrates

Minimum and maximum

Clock to Q output propagation delays for

Low to high and a high to low transition on the flip-flop output

As with combinational logic

Delays are measured at the 50% point

Between the causative and consequent edges of the signals

Two delays are generally not symmetric

Propagation delay specification for latches

Slightly more complex

In addition to the delay from the causative edge (or level) of the gate

Latch transparency requires that the delay from input to output

When the *Gate* is enabled be specified as well

Timing diagram illustrates

Delay from leading edge of the *Gate* to the Q output of the device
Delay to the latch Q output
Resulting from a state change in the input follows naturally

Timing Margin
To study the concept of timing margin
We’ll analyze the Johnson counter timing in greater detail
The two stage implementation is redrawn

If we clock the circuit
Will get the pattern \{\ldots0011001100\ldots\} on the output
If we continually increase the frequency
Pattern will repeat until at some frequency it fails
Let’s analyze the timing of the circuit
To understand what and where problems might originate
In foregoing analysis
SSI part is used
Analysis proceeds in exactly same manner
Inside of ASIC, PLD, microprocessor, or other type device

The essential specifications on the 74ALS74 - D type flip-flop are given as
\[ \tau_{\text{PD LH}} = 5-16\text{ns} \]
\[ \tau_{\text{PD HL}} = 7-8\text{ns} \]
\[ \tau_{\text{su}} = 16\text{ns} \]

The timing diagram illustrates
Clock and the Q output of the A flip-flop
For two changes in the value of the state variable

![Timing Diagram](image)

When the setup time is violated
Circuit will not behave as designed
May behave in unpredictable ways
To understand the circuit timing constraints consider 2 cases
The foregoing analysis assumes no signal delay caused by
Either parasitic devices or the board layout.

Case 1 – Low to High Transition of QA
Clock period = \( t_{\text{pdlH}} + t_{\text{su}} + \text{slack}_0 \)
From the timing diagram,
In the limit as \( \text{slack}_0 \) approaches 0
The clock period approaches a minimum
Clock period = \( t_{\text{pdlH}} + t_{\text{su}} \)
Under such a condition and with the minimum value for \( \tau_{\text{pdlH}} \)
The maximum frequency for the counter will be
\[ F_{\text{max}} = \frac{1}{(5 + 16) \times 10^{-9} \text{ sec.}} = 48 MHz \]

If \( \tau_{pdLH} \) is at its maximum value
Maximum frequency for the counter will be
\[ F_{\text{max}} = \frac{1}{(16 + 16) \times 10^{-9} \text{ sec.}} = 31.3 MHz \]

Case 2 – High to Low Transition of QA

Clock period = \( t_{pdHL} + t_{su} + \text{slack}_1 \)

From the timing diagram
In the limit, as \( \text{slack}_1 \) approaches 0,
Clock period approaches a minimum

Clock period = \( t_{pdHL} + t_{su} \)

Under such a condition and with the minimum value for \( \tau_{pdHL} \)
Maximum frequency for the counter will be
\[ F_{\text{max}} = \frac{1}{(7 + 16) \times 10^{-9} \text{ sec.}} = 43.5 MHz \]

If \( \tau_{pdHL} \) is at its maximum value
Maximum frequency for the counter will be
\[ F_{\text{max}} = \frac{1}{(18 + 16) \times 10^{-9} \text{ sec.}} = 29.4 MHz \]

Based upon the calculations above
Maximum clock frequency that can be used for the circuit is 29.4 MHz

To ensure reliable operation with any individual SN74ALS74 device
When designing
One must always consider worst case values
Then make an educated evaluation of how far to carry such analysis
If carried too far it’s possible to prove
No design will ever work properly
Checking Boundaries

Verilog HDL
- Provides some tools to aid in analyzing timing issues
- In sequential circuits

Setup and Hold Times
- When working with clocked sequential circuits
- There are several important parameters
  - Must be met
  - Must be verified in the design

As we discussed earlier
- Data to be clocked into sequential circuit
  - Must be stable – unchanging
  - Minimum time before sampling edge

Relevant in two major contexts
- Inside system under design
- With respect to signals coming in to system
  - From external world

- Both cases can lead to metastability
  - If timing not met
  - Such minimum time denoted setup time

Setup time constraints apply to
- Temporal region prior to sampling edge

Equally important is hold time
- Parameter specifies amount of time
  - Input data must be stable – unchanging
  - Minimum time following sampling edge

Although somewhat less important than other two parameters
- Minimum width of pulse can be important
- Found in
Control signals
Latching enable signals
Strobes

These values are given in the accompanying diagram

The Verilog language supports
Test and confirmation of each of these
Setup and hold constraints are given in `specify` block
Illustrated in following code fragment

```verilog
specify
$width(posedge clock, pulseWidth);
$setup(data, posedge clock, setupTime);
$hold(posedge clock, data, holdTime);
endspecify
```

For the data and clock illustrated in figure above

Signal Skew
As system operating frequency and bus speeds increases
Signal skew
   Across signaling wave front
   Between several signals

Becoming increasingly important

Action based upon signals cannot be taken
   Until certain that all have reached stable value
Immediate effect
   Operating speeds must be reduced to accommodate
Following figure illustrates such skew between two signals.

Verilog supports:
- Setting limits on skew
- Recognition skew when it occurs
- Reporting such violations

Skew constraints are given in *specify* block. Illustrated in following code fragment:

```verilog
specify
  $width(posedge clock, pulseWidth);
  $setup(data, posedge clock, setupTime);
  $hold(posedge clock, data, holdTime);
  $skew(posedge signalA, posedge signalB, skewLimit);
endspecify
```

For the two signals illustrated in figure above.

**Practical Considerations – Part 3 Clocks and Clock Distribution**

**The Source**
- The clock system or time base in a digital system is an essential component in ensuring proper operation.
- For certain hard real-time applications, having the proper time base is critical to meeting the timing specifications.
- There are four fundamental parameters that one should consider when designing or selecting a clock system or time base.

For the basic clock, these parameters are:
- Frequency and frequency range
- Rise times and Fall Times
- Stability
- Precision

**Frequency**
- Often start out with a clock source that is a higher frequency than necessary
  - Then can use ripple counters to divide down
Higher frequency
To a number of lower frequencies
Remember, because ripple counters are asynchronous
One should never decode any of the state variable combinations
To generate a specific frequency

Decoding spikes
Will occur
Will have enough energy
To clock flip-flops and latches at the wrong times

Phase Locked Loop
Building a higher frequency from a lower one
Done using a phase locked loop
Basic block diagram appears as drawn

Feedback signal comprises the output of a voltage controlled oscillator (VCO)
That is controlled by the output voltage
When the phase difference between
Input signal and the output of the VCO is zero
System has locked onto the input frequency
Output of the phase detector will be zero
Difference in phase appears as an error voltage
Filtered
By the low pass filter shown
Amplified
Used to provide an input voltage to the VCO
Output of the VCO
Can now serve as the clock to the system time base
Input signal to the PLL
Any of the encoded data streams

Rate Multiplier
A scheme to select a fractional portion of a clock signal
Called a rate multiplier
Block diagram for such a circuit is given below

A rate multiplier
Simply a combinational logic block
Combined with an N bit counter
Period of the counter will be $2^N$
Counter will cycle through all of its states every $2^N$ clock pulses
N bit selector also permits $2^N$ combinations
For each of the $2^N$ combinations on the selector input
Device will output that many pulses
Thus if N is 4
The counter will be 4 bits and there will be 4 selector lines
If the counter is a binary counter and the selector pattern is ‘0101’
Binary 5
Then for every 16 clock pulses coming into the counter
There will be 5 output pulses
The output frequency will be 5/16 of the input frequency
The design of the rate multiplier is such that
Selected number of output pulses
Evenly distributed across the period of the clock
Illustrated in the timing diagram
For a selector pattern of binary 5
If using a high frequency oscillator as the primary clock source in a design
    But a portion of the application requires
    Significantly lower frequency
    Ripple counter can provide a very effective means of developing signal
    Twelve to fourteen stage ripple counters
    Commonly found as a single MSI part
    By using such a counter
    Can easily divide high frequency source by as much as 16K
    When doing so
    One must be aware that there will be a substantial skew
        Between the edge of the input signal
        Resulting edge of any of the lower frequency signals
        Because of the ripple delay through the device

The application of such a divider
    Illustrated in the following logic diagram

Affect of the edge skew is reflected in the subsequent timing diagram
    For an asynchronous binary up counter

The timing diagram illustrates the propagation delay for the first four stages
    As the counter changes from a count of binary seven to binary eight

Observe that the interstage delay
    Accumulates as subsequent flip-flops change state
    Here the change in state of the third stage
Delayed by four propagation delays
Following the causative event in the first stage
Doesn’t take too much imagination to visualize
Situation in which the least significant stage
May have changed states twice
Before the n\(^{th}\) stage is able to change

Precision and Stability
Simplest kinds of clock sources
Use resistor and capacitor combinations to set their output frequency
While such devices may be perfectly reasonable for
Controlling windshield wipers or a door bell
Should never be used in critical hard real-time applications
Capacitors have
Wide tolerances
Subject to
Humidity, brownout, low voltage levels, number of environmental affects

Crystal based sources
Generally the best solution
For stable and accurate timing signals
When using such devices
One can either start with the basic crystal
Then design the analog electronics
Necessary to implement the desired oscillator
Buy a prepackaged oscillator

Each crystal or oscillator has
Different stability and accuracy specifications

If the application demands greater accuracy and stability
Than is available with standard devices
Next step is to use temperature compensated designs
Such sources utilize a small heater
Minimize the affects of temperature variation of the oscillator.
Designing a Clock System

Single Phase Clocks

A single phase clock should start with the crystal oscillator. Such an approach gives stability and repeatability to any clocking and timing that needs to be in the embedded application. Variations on the accompanying circuit should never be used.

Design is trying to do a digital job with analog parts. Adding the R and C as shown can significantly affect the rise and fall times of the input signals to the buffer. As the transition times increase, so does the probability of the circuit becoming metastable.

Multiple Phase Clocks

It is frequently the case in contemporary digital systems that more precise resolution in time is required than can be achieved with a single phase clock. Consider the basic clock waveform:

There are four places within a single clock period that a decision can be made:
1. The rising edge,
2. The falling edge,
3. The high level, and
4. The low level.

One cannot tell the difference in time between the edge at 1a and that at 1b. Best resolution we have is a half period.

Simplest multiple phase clock generator given in accompanying diagram:

The circuit generates two non-overlapping clock signals as output.
The structural Verilog model for the clock generator
Given in following code module

```verilog
module ClockTwoPhase(phase2, phase1, clk, por);
  // declare inputs and outputs
  input       clk, por;
  output      phase2, phase1;
  reg         pullUp;

  initial
    pullUp = 1;

  // build clock
  not inv0(nclk, clk);
  DFF f0(qF0, qBarF0, qBarF0, nclk, pullUp, por);
  and andP1(phase1, qF0, clk);
  and andP2(phase2, qBarF0, clk);
endmodule
```

The timing diagram for the generator is given as follows

We can now see that over a two clock cycle interval
Have eight different places that a logical decision can be made

Further the edge at $t_0$ is distinguishable from the edge at $t_1$
Same holds true for the remaining edges and levels along the two phases

Using such a scheme can now
Use $P_2$ as a causative edge and $P_1$ as a sampling edge
$P_2$ will affect an event or state change
Interval between $P_2$ and $P_1$ should be sufficient
For all changing and propagating signals to settle
Before they are acted upon by the logic clocked by P1

Although the circuit contains a race
As illustrated in the logic diagram

Race is biased towards path2
There can never be a decoding spike
On either of the two AND gates generating P1 and P2

Another effective and flexible way to produce a multiple clock phase time base
Use a Johnson counter

By decoding each of the four states in the counting sequence
Of two stage Johnson counter
Can generate four different phased clocks
Structural Verilog model follows

```verilog
module ClockFourPhase(phase4, phase3, phase2, phase1, clk, por);

// declare inputs and outputs
input       clk, por;
output      phase4, phase3, phase2, phase1;
reg         pullUp;

initial
    pullUp = 1;

// build clock
not inv0(nclk, clk);
DFF f0(qF0, qBarF0, qBarF1, nclk, pullUp, por);
DFF f1(qF1, qBarF1, qF0, nclk, pullUp, por);

// build the four phases
and andP1(phase1, qBarF0, qBarF1);
and andP2(phase2, qF0, qBarF1);
and andP3(phase3, qF0, qF1);
and andP4(phase4, qBarF0, qF1);

endmodule
```

The timing diagram illustrates the four different clock phases

![Timing Diagram](image)

By incorporating additional phases

Have increased the control that we have over

Placement or sampling of events in time
More than Four Phases

Expanding the time base beyond four phases
We can continue to build on the Johnson counter
If such a design is utilized
Disconnected subgraph for each case will have to be managed

An alternate approach is to utilize a delay based scheme
Such as a tapped delay line
Advantage of such an approach
One can use a lower frequency clock
Johnson counter used in the previous design
Requires a base clock frequency
Four times the frequency of the phases

Multiple Clocks vs. Multiple Phases
The major advantage of multiple phases
When compared to multiple clocks
All phases are derived from the same fundamental frequency
Clock noise can be filtered out much more easily
With multiple independent clocks
Although all may be using the same frequency
All are running asynchronous to each other

Gating the Clock
The general rule of thumb is never do it
Because of the high potential for hazards
If gating becomes essential
Thoroughly understand the timing
Change the control logic
Only when clock in such a state
That it cannot result in change on the gate output
For example, the two-phase clock discussed earlier