Practical Considerations - Introduction to Signal Behaviour in the Real World

Overview

In this lesson we will

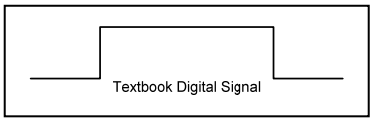
- ✓ Examine a high-level view of digital signaling and signal quality.
- ✓ Begin with an introduction to timing considerations in combinational logic.
- ✓ Examine modeling real-world combinational logic timing issues in Verilog.
- ✓ Discuss timing issues in latches and flip-flops.
- ✓ Examine and model real-world timing issues in sequential circuits.
- ✓ Discuss clocks and clock distribution.
- ✓ Introduce multiphase clocking schemes.

Introduction

In text book or ideal world

Signals change state or propagate though combinational or sequential networks

In zero time



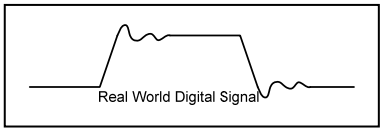
As we take first steps into real world

Begin to see that textbook models

Do not exactly match what we see on lab bench

Begin to see that quality or integrity of textbook signals

Different from we see in out circuits



Signal edges and transitions

Not as crisp

See oscillations called ringing as signals change state

Signal propagation

Meander to destination

As we look farther and deeper into real world

Discover at every turn real world signals encounter physics of practical devices

Thousands of dead physicists are there just waiting for us

Maxwell, Faraday, Lenz, Gauss and all their friends

Say welcome

Real world systems

Seem filled with black magic

Problems seem to become increasingly mysterious

As signaling frequency increases

If we are to design and build systems for today and tomorrow

That operate reliably and robustly in real world

We *must* understand when, where, how, and why
    Such physical affects occur

Once we gain such understanding
    Potentially can design around or compensate for impending problems
    Can incorporate such knowledge into our models
        To determine and test
            ✓ How such problems are affecting our system
            ✓ If our design approach for mitigating effects of real world
                Has proven successful

In this lesson
    Will take first steps into
        Real world
    Examine some issues affecting digital signal quality
    Lay foundation introduce some concepts and vocabulary
        More advanced area called signal integrity

We will begin our studies with introduction of
    Verilog constructs for modeling high-level behaviour
        In combinational logic then move on to sequential circuits

Practical Considerations – Part 1 Timing in Combinational Logic – introduction

Will begin by examining 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

Such components inherently and fundamentally present in

- Systems built of discrete components
- Programmable logic devices

  Internal to device
- Getting onto or off of device
- LSI and VLSI circuits

Initial view will be at high level

- ✓ Introduce terminology
- ✓ Illustrate macro level view of concepts

Detailed view

- ✓ Illustrate micro level view of concepts

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 among many other things

- Signal delays
- Physical variations in signal path and return
- Ground and power planes
- Impedance discontinuities
- Noise
- Crosstalk

In our studies of combinational logic

We find several fundamental timing issues

To study we need to consider basic parameters

- ✓ Rise / Fall Times
- ✓ Propagation delay
- ✓ Race Conditions
- ✓ Hazards

Need to go back to fundamentals

At the end of the day we are moving charge from one place to another

This does not happen in 0 time

Recall the fundamental relationships

$$v(t) = \frac{1}{C}\int idt$$

$$= \frac{1}{C}\int dq$$

$$v(t) = L\frac{di}{dt}$$

These will carry forward to our work with sequential circuits

We will also examine

Potential root causes for such issues

Effects on our circuits

Some basic models

In later sections

Will see how to incorporate such affects into Verilog model

Examine some tools and techniques by which we can mitigate such effects

Fundamental Attributes

Let's briefly review each of timing issues

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

Logically

From 0 to 1 or 1 to 0

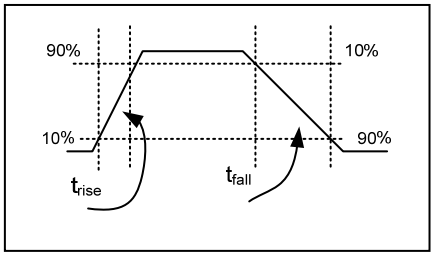
Physically

From 0 charge to some charge

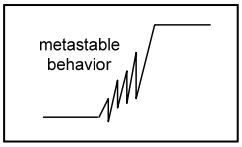
From some charge to 0 charge

Tristate part to cease driving
Switching from one driver to another
Turning driver OFF or ON

Consider the accompanying 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
τ_r and τ_f or
τ_{rise} and τ_{fall}



Problem
If rise / fall times too long
Gate no longer acts as switch instead becomes poor amplifier
Device can enter what is called *metastable* region
Potentially effect is
For device output to (temporarily) oscillate
Rather than crisp change of state
As shown in accompanying diagram
Will discuss phenomenon and root cause in more detail shortly



Modeling Rise Time, Fall Time, and Turn-Off Delays
In our designs
Must take rise and fall times into consideration
Models must incorporate the time required for a signal to change state
Verilog supports including device rise time, fall time into model
Other modeling tools available as well
SPICE as one example

The syntax for these is given as

Syntax
(rise time, fall time) device;

Observe the ( )

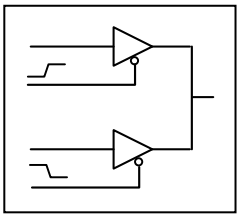
Illustrated in following code fragment

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

The rise and fall time values show up as delays in simulation
    Working with digital simulation

When working with busses or simply individual signals
    Connected utilizing tristate devices
Time for device output to
    Turn-on or turn-off when control is asserted or 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 however briefly
        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
                Lead to invalid logic signals
                    Phenomenon called ground bounce

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

    In this context
        Rise time delay τ_{pdLH}
            Rising prop delay
        Fall time delay τ_{pdHL}
            Falling prop delay

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 take high level view of 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 effects of real-world

Propagation Delay

Time required for the effects 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 or vice versa

Often different from a state change in opposite direction

Propagation delay can easily be observed in an inverter

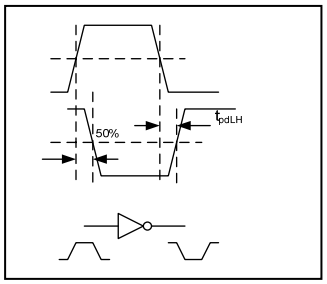
If a high going signal is set as the input into the device

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 with respect to signal change

Logical 0 to a logical 1 or vice versa

τ_{dlh} and τ_{dhl} or

τ_{pdlh} and τ_{pdhl}

Values vary with

- Logic family
- Resistive and reactive 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

Must consider

Signal duration
Prop delay ↔ propagation velocity
These are inverses

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 effect on the output
Different propagation delay models can be used
To study the behavior of combinational devices
Will use basic model looking at single device

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

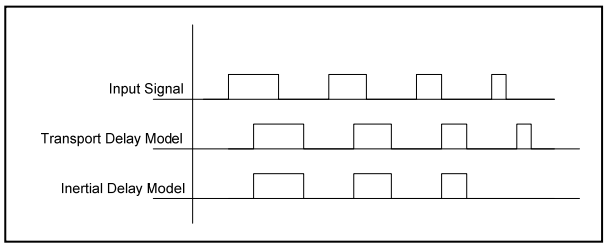
Under the *transport delay* model
Changes in input independent of duration
Are seen by the output following the specified delay

Model
Permits all signals to pass down propagation path
Regardless of duration
Transport
Assumes signaling path
Has infinite bandwidth

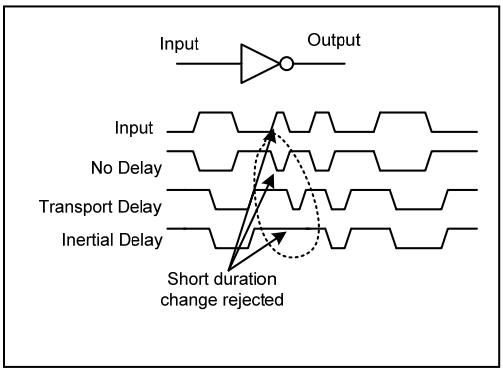
Inertial Delay

An *inertial delay* model acts somewhat like low pass filter
Signals with duration shorter than some value
Slowly changing
Inertial
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 perspective first



Consider 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

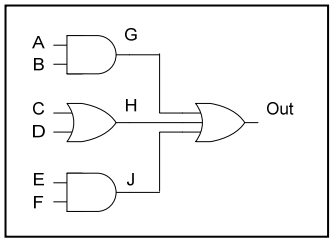
Path Delay

Such basic models do not consider *transport velocity*
Rate at which signal propagates down path
To accommodate *transport velocity*
Propose two alternate views of the signal path
Distributed
Lumped

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 bidirectional as inout
Destination pin – out or bidirectional as inout
Called *path delay*



Path delays in Verilog assigned using *specify* block

Such a block delimited by keywords

specify

endspecify

```
specify
   (A => Out) = 30;
   (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);
```

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
- ✓ Simple extension of pin-to-pin delay

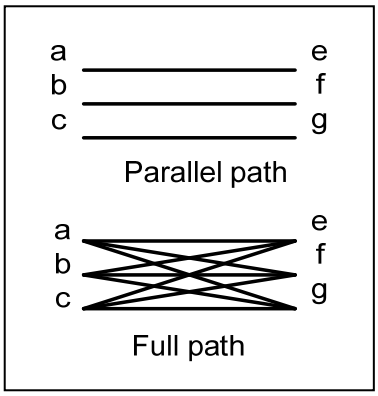
When working with vectors of signals

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: (s0, s1, s2, s3)

Destination vector *out* of 3 signals: (o0, o1, o2)

Within specify block

Operator *> distinguishes full connection semantics

Can write

Expression: (s0, s2 *> out) = delay1;
(s1, s3 *> out) = delay2;

Expressions interpreted as

Delay from s0 or s2

Through any logic block

To any output signal within output vector

Has delay equal to delay1

Delay from s1 or s3

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

```
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

```
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 effects 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 a 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 a decision point.

A *hazard* exists in any circuit

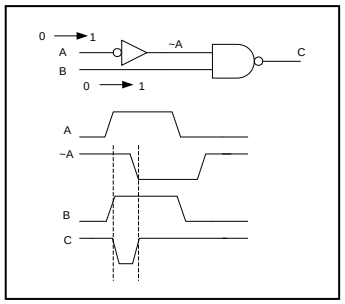
That has the possibility of producing an incorrect output

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



Let's look at simple example

This circuit has a static 1 hazard

We have a

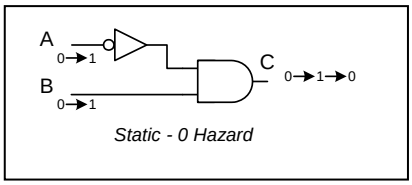
Static-0 hazard

When our circuit can produce an erroneous 1

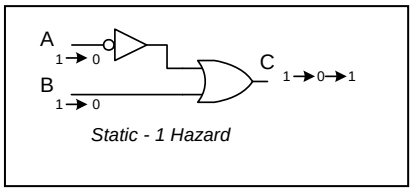
When the output should be a constant 0

Static-1 hazard under the opposite condition

The following circuits give examples of each of these hazards



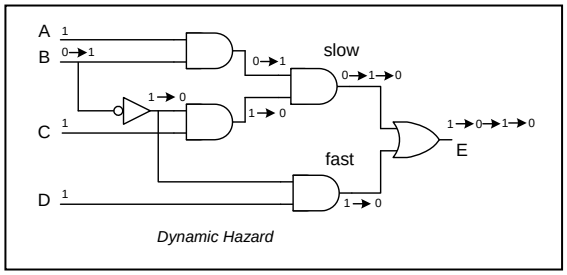
Static - 0 Hazard



Static - 1 Hazard

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

The following circuit give an examples of a dynamic hazards



Dynamic Hazard

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

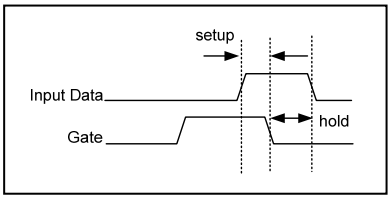
For combinational logic devices
- Major timing concerns focused on the delay of signals
  - Propagating through the devices
- Timing relationship between
  - Input data and the gate in latches
  - Clock in flip-flops
- Introduces the notions of *setup time* and *hold time*

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

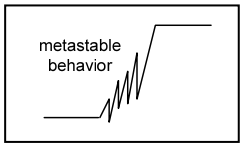
Initiate and complete the appropriate state changes

For any internal memory elements

Times are designated as τ_{setup} or τ_{su} and τ_{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

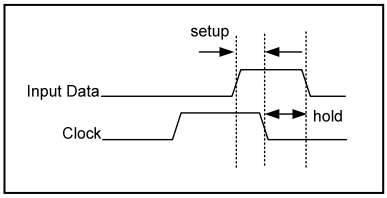
Output may enter a metastable state

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 negative 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

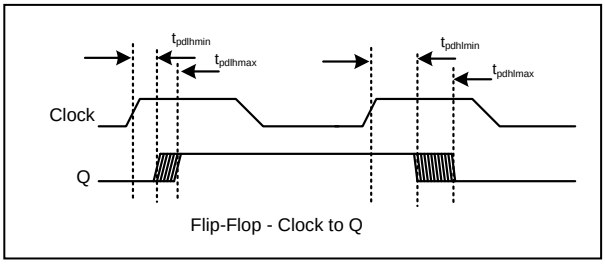
The *causative edge* of the clock or strobing signal

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

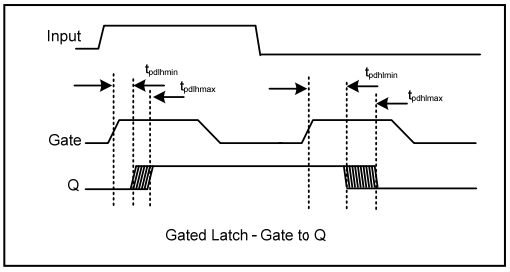
In addition to the delay from the causative edge (or level) of the gate

Latch transparency requires that the delay from input to output

When the *Gate* is enabled be specified as well

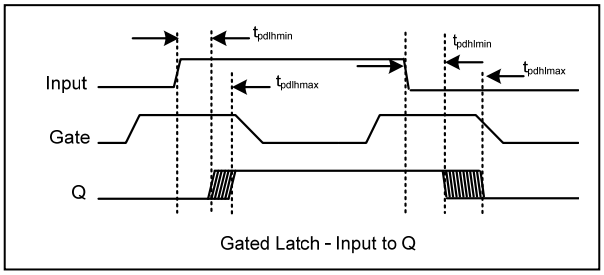
Timing diagram illustrates

Delay from leading edge of the *Gate* to the Q output of the device



Delay from input to the latch Q output

Resulting from a state change in the input follows naturally

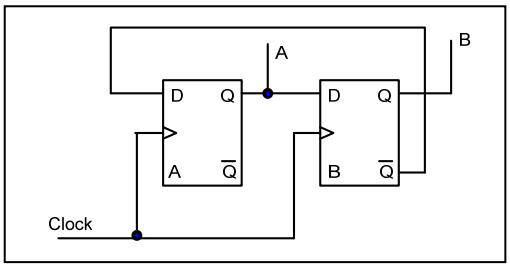


Timing Margin

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 {…0011001100…} 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

τ_{PDLH} = 5-16ns

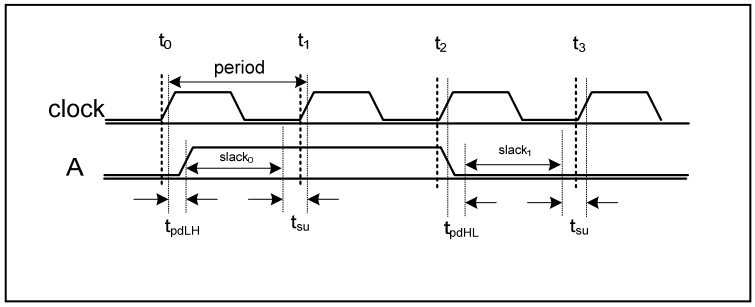
τ_{PDHL} = 7-18ns

τ_{su} = 16ns

The timing diagram illustrates

Clock and the Q output of the A flip-flop

For two changes in the value of the state variable



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 Q_A

Clock period = $t_{pdLH} + t_{su} + slack_0$

From the timing diagram

In the limit as $slack_0$ approaches 0

The clock period approaches a minimum

Clock period = $t_{pdLH} + t_{su}$

Under such a condition and with the minimum value for τ_{pdLH}

The maximum frequency for the counter will be

$$F_{\max} = \frac{1}{(5+16)x10^{-9}\ \text{sec}.}$$

$$= 48MHz$$

If τ_{pdLH}, is at its maximum value

Maximum frequency for the counter will be

$$F_{\max} = \frac{1}{(16+16)x10^{-9}\ \text{sec}.}$$

$$= 31.3MHz$$

Case 2 – High to Low Transition of Q_A

Clock period = $t_{pdHL} + t_{su} + slack_1$

From the timing diagram

In the limit, as $slack_1$ approaches 0,

Clock period approaches a minimum

Clock period = $t_{pdHL} + t_{su}$

Under such a condition and with the minimum value for τ_{pdHL}

Maximum frequency for the counter will be

$$F_{max} = \frac{1}{(7+16)x10^{-9}\ \text{sec.}}$$
$$= 43.5MHz$$

If τ_{pdHL}, is at its maximum value

Maximum frequency for the counter will be

$$F_{max} = \frac{1}{(18+16)x10^{-9}\ \text{sec.}}$$
$$= 29.4MHz$$

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

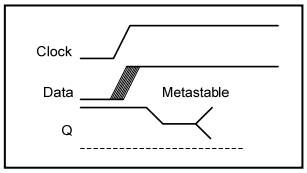
As we discussed earlier

Data to be clocked into sequential circuit

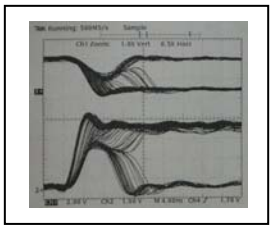
Must be stable – unchanging

Minimum time before sampling edge

If data not given sufficient time to
    Settle
    Propagate sufficiently into system
Can lead to phenomenon called *metastability*



Metastability is ability of digital system
    To exist in unstable equilibrium
        For unbounded time
In such a state
    System may not be able to settle into stable state
        Within time necessary to ensure proper operation
Such a situation illustrated in accompanying diagrams



Metastable states are inherent features of
    Asynchronous systems
    Systems with multiple independent clocks

Relevant in two major contexts
- ✓ Inside system under design
    Both single and multiple clock systems
- ✓ 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

Metastability is a significant issue in design of digital systems
    Becoming increasingly important with growing demand
        Distributed systems
            Integrated and discrete components
                FPGAs
                Multiple core or microprocessor based systems
    Such states are a possibility wherever asynchronous inputs occur

Problem is manageable because

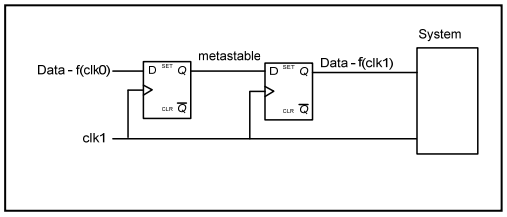
Probability of a metastable state persisting longer than specified duration

Exponentially declining function of *t*

In electronic devices, the probability of such an "undecided" state

While always possible is infinitesimal

Basic synchronizer given in following diagram



Data coming into system

Generated with respect to clock clk0

Synchronized to clock clk1

Assumptions

- Metastable condition of shorter duration than period of clk1
- D input to second FF meets setup time

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 that other two parameters

Minimum *width* of pulse can be important

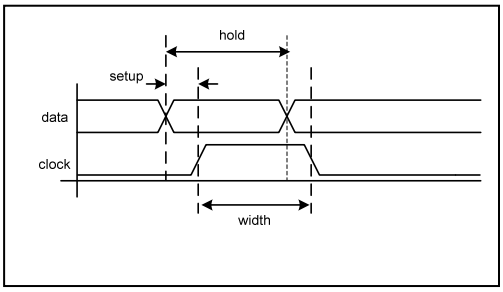
Found in

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

```
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

Note - these are system calls or checks

Violation reported by system

Reported at runtime as timing violation

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

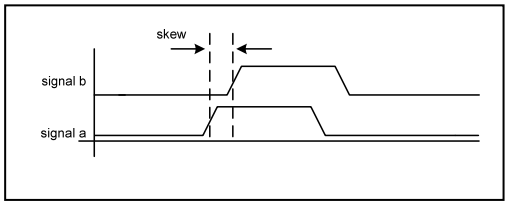
Until certain that all have reached stable value

Immediate effect

Operating speeds must be reduced to accommodate

Accompanying 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

Which adds the skew system task to the specify block

```
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

Once again these are system calls

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
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

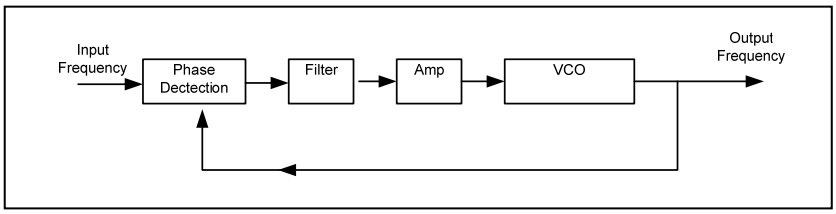
Decoding spikes
Will occur
Will have enough energy
To clock flip-flops and latches at the wrong times

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

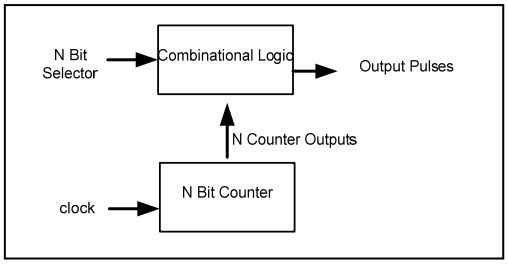
Rate Multiplier

A scheme to select a fractional portion of a clock signal

Called a *rate multiplier*

Block diagram for such a circuit

Given in accompanying diagram



A rate multiplier

Simply a combinational logic block

Combined with an N bit counter

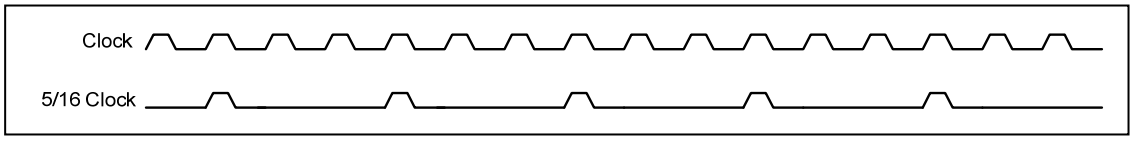
Period of the counter will be 2^N

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



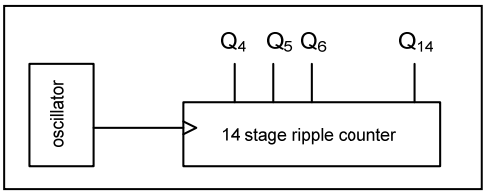
If using a high frequency oscillator as the primary clock source in a design
But a portion of the application requires
Significantly lower frequency

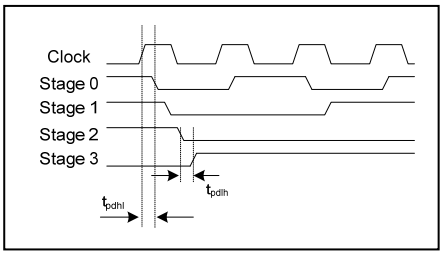
Ripple counter can provide a very effective means of developing signal
Twelve to fourteen stage ripple counters
Commonly found as a single MSI part

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

Effect 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

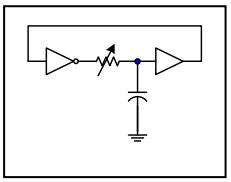
When using such devices
One can either start with the basic crystal
Then design the analog electronics
Necessary to implement the desired oscillator
Buy a prepackaged oscillator

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 effects of temperature variation of the oscillator.

Designing a Clock System

Single Phase Clocks



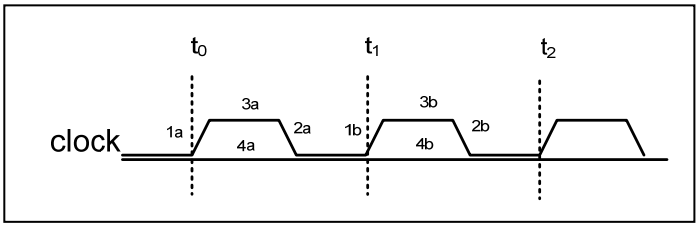
A single phase clock should start with the crystal oscillator
Such an approach gives
Stability and repeatability to any clocking and timing
That needs to be in the embedded application.
Variations on the accompanying circuit should never be used

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

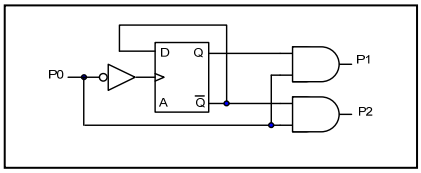
It is frequently case in contemporary digital systems
That more precise resolution in time is required
Than can be achieved with a single phase clock

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.



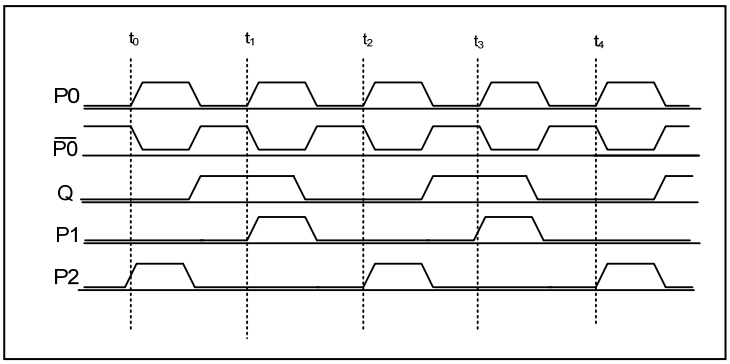
One cannot tell the difference in time
Between the edge at 1a and that at 1b

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

```
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:

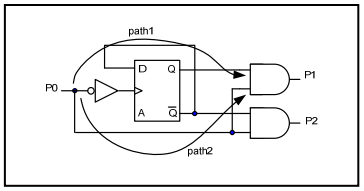


We can now see that over a two clock cycle interval
Have eight different places that a logical decision can be made

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 P2 as a causative edge and P1 as a sampling edge
P2 will affect an event or state change
Interval between P2 and P1 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

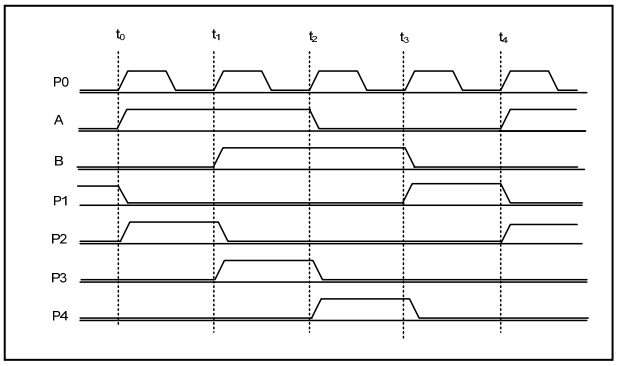


Another effective and flexible way to produce a multiple clock phase time base
Use a Johnson counter

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

```
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



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

Summary

In this lesson we

- ✓ Examine a high-level view of digital signaling and signal quality.
- ✓ Began with an introduction to timing considerations in combinational logic.
- ✓ Examined modeling real-world combinational logic timing issues in Verilog.
- ✓ Discussed timing issues in latches and flip-flops.
- ✓ Examined and model real-world timing issues in sequential circuits.
- ✓ Discussed clocks and clock distribution.
- ✓ Introduced multiphase clocking schemes.