CHAPTER 1 CSL Clock Tree

TABLE 1.1 Chapter Overview

1.1 CSL Clock Tree Overview
1.2 CSL Clock Tree Concepts
1.3 CSL Clock Tree Commands summary
1.4 CSL Clock Tree Commands
1.5 CSL Clock Tree Examples
1.6 CSL Clock Tree Checker

1.1 CSL Clock Tree Overview

1.2 CSL Clock Tree Concepts

- adding the clock tree to the design
- the clock tree includes enabled clock gates to control the clock for specific blocks
- the clock enable logic can be generated in the module which the clock drives OR in a central controller

1.3 CSL Clock Tree Commands summary

1.4 CSL Clock Tree Commands

csl_clock *clock_object_name;*

DESCRIPTION :

Create a csl_clock object called *clock_name*

EXAMPLE :

CSL CODE

```
// create a csl_clock object called "clock_name"
csl_clock clock_name;
```

NOTE: DP: Add a function to create negative clock and positive clock !

clock_object_name.**set_name(**“*new_clock_name*”**);**

DESCRIPTION :

Change the name of the csl clock object *clock_name* to another name.

EXAMPLE :

CSL CODE

```
//create a csl_clock object called “csl_clock_name”
csl_clock csl_clock_name;
//change the name of the “csl_clock_name object” to “new_clock_name”
csl_clock_name.set_name(“new_clock_name“);
```

clock_object_name.**set_time_base(***numeric_expresion***)**;

DESCRIPTION :

EXAMPLE :

//add an example

CSL CODE

```
   //csl code goes here
```

VERILOG CODE

```
   //verilog code goes here
```

int *clock_object_name*.**get_time_base();**

DESCRIPTION :

EXAMPLE :

//add an example

CSL CODE

```
   //csl code goes here
```

VERILOG CODE

```
   //verilog code goes here
```

```
clock_object_name.set_period(numerical_expression);
```

DESCRIPTION :

Change the period of the csl clock object *clock_name*.

EXAMPLE :

CSL CODE

```
//create a csl_clock object called “csl_clock_name”
csl_clock csl_clock_name;
/*change the value of the clock period to a new one defined by a numer-
ical expression*/
csl_clock_name.set_period(numerical_expression);
```

```
clock_object_name.set_frequency(numerical_expression);
```

DESCRIPTION :

Change the frequency of the csl clock object *clock_name*.

EXAMPLE :

CSL CODE

```
//create a csl_clock object called "csl_clock_name"
csl_clock csl_clock_name;
/*change the value of the frequency to a new one defined by a numerical
expression */
csl_clock_name.set_frequency(numerical_expression);
```

`clock_object_name.`**`jitter(`**`numerical_expression`**`);`**

DESCRIPTION :

NOTE: Make a short conclusion

Change the jitter of the csl clock object *clock_name*.
Jitter, in electronics and telecommunications, is an abrupt and unwanted variation of one or more signal characteristics, such as the interval between successive pulses, the amplitude of successive cycles, or the frequency or phase of successive cycles. Jitter is a significant factor in the design of almost all communications links (e.g. USB, PCI-e, SATA, OC-48).

Jitter can apply to a number of signal qualities (e.g. amplitude, phase, pulse width or pulse position), and can be quantified in the same terms as all time-varying signals (e.g. RMS, or peak-to-peak displacement). Also like other time-varying signals, jitter can be expressed in terms of spectral density (frequency content). Generally, very low frequency jitter is not of interest in designing systems, and the low-frequency cutoff for jitter is typically specified at 1 Hz.
For clock jitter, there are three commonly used metrics: absolute jitter, period jitter, and cycle to cycle jitter.

Absolute jitter is the absolute difference in the position of a clock's edge from where it would ideally be if the clock's frequency was perfectly constant. The absolute jitter metric is important in systems where a large number of clock sources are trying to pass data to one another (eg. SONET).

Period jitter (aka cycle jitter) is the difference between any one clock period and the ideal clock period. Accordingly, it can be thought of as the discrete-time derivative of absolute jitter. Period jitter tends to be important in synchronous circuitry like digital state machines where the error-free operation of the circuitry is limitted by the shortest possible clock period, and the performance of the circuitry is limitted by the average clock period. Hence, synchronous circuitry benefits from minimizing period jitter, so that the shortest clock period approaches the average clock period.

Cycle-to-cycle jitter is the difference in length between any two adjacent clock periods. Accordingly, it can be thought of as the discrete-time derivative of period jitter. It can be important for some types of clock generation circuitry used in microprocessors and RAM interfaces.

All of these jitter metrics are really measures of a single time-dependent quantity, and hence are related by derivatives as described above. Since they have different generation mechanisms, different circuit effects, and different measurement methodology, it is still useful to quantify them separately.

In the telecommunications world, the unit used for the above types of jitter is usually the UI (or Unit Interval) which quantifies the jitter in terms of a fraction of the ideal period of the clock. This unit is useful because it scales with clock frequency and thus allows relatively slow interconnects such as T1 to be compared to higher speed internet backbone links such as OC-192. Absolute units such as picoseconds are more common in microprocessor applications. Units of degrees and radians are also used.

If jitter has a Gaussian distribution (ie. is random), it is usually quantified using the standard deviation of this distribution (aka. RMS). Often, jitter distribution is significantly non-Gaussian. This can occur if the jitter is caused by external sources such as power supply noise. In these cases, peak-to-peak measurements are more useful. Many efforts have been made to meaningfully quantify distributions that are neither Gaussian nor have meaningful peaks (which is the case in all real jitter). All have shortcomings but most tend to be good enough for the purposes of engineering work. Note that typically, the reference point for jitter is defined such that the mean jitter is 0.

In networking, in particular IP networks such as the Internet, jitter can refer to the variation (statistical dispersion) in the delay of the packets (because of routers' internal queues behaviour in certain circumstances, routing changes, etc).

EXAMPLE :

CSL CODE

```
//create a csl_clock object called "csl_clock_name"
csl_clock csl_clock_name;
/*change the value of the jitter to a new one defined by a numerical
expression*/
csl_clock_name.jitter(numerical_expression);
```

clock_object_name.**skew(***numerical_expression***);**

DESCRIPTION :

Change the skew of the csl clock object *clock_name*.

EXAMPLE :

CSL CODE

```
//create a csl_clock object called "csl_clock_name"
csl_clock csl_clock_name;
/*change the value of the skew to a new one defined by a numerical
expression*/
csl_clock_name.skew(numerical_expression);
```

clock_object_name.**add_clock_en(***signal_object_name***);**

DESCRIPTION :

Add an input signal to the clock generator named *clock_object_name* to validate.

EXAMPLE :

CSL CODE

```
// CSL example goes here
```

VERILOG CODE

```
// Verilog example goes here
```

clock_object_name.**add_xtal**(*signal_object_name*);

DESCRIPTION :

Add an input signal to the clock generator named *clock_object_name*.

EXAMPLE :

CSL CODE

```
// CSL example goes here
```

VERILOG CODE

```
// Verilog example goes here
```

`clock_object_name.add_clock_out(signal_object_name);`

DESCRIPTION :

Add an output signal to the clock generator named *clock_object_name*.

EXAMPLE :

CSL CODE

```
// CSL example goes here
```

VERILOG CODE

```
// Verilog example goes here
```

`clock_object_name.`**`add_clock_out_div2`**`(signal_object_name);`

DESCRIPTION :

Add an output signal to the clock generator named *clock_object_name* divided by 2.

EXAMPLE :

CSL CODE

```
// CSL example goes here
```

VERILOG CODE

```
// Verilog example goes here
```

`csl_clock_tree` *`clock_tree_name;`*

DESCRIPTION :

n/a

EXAMPLE :

CSL CODE

```
   //csl code goes here
```

clocktree_name.**clk_en(***clk_en_name***);**

DESCRIPTION :

n/a

EXAMPLE :

CSL CODE

```
    //csl code goes here
```

`*clocktree_name*.**clk_gate(***clk_gate_name***);**`

DESCRIPTION :

n/a

EXAMPLE :

CSL CODE

```
   //csl code goes here
```

clocktree_name.**clk_mux(***clk_mux_name***);**

DESCRIPTION :

n/a

EXAMPLE :

CSL CODE

```
    //csl code goes here
```

clocktree_name.**clk_switch(***clk_switch_name***);**

DESCRIPTION :

n/a

EXAMPLE :

CSL CODE

```
   //csl code goes here
```

clocktree_name.**clk_buffer(***clk_buffer_name***);**

DESCRIPTION :

n/a

EXAMPLE :

CSL CODE

```
   //csl code goes here
```

clocktree_name.**clk_pll(***clk_pll_name***)**;

DESCRIPTION :

n/a

EXAMPLE :

CSL CODE

```
   //csl code goes here
```

`*clocktree_name*.**clk_div**(*clk_div_name*);`

DESCRIPTION :

n/a

EXAMPLE :

CSL CODE

```
   //csl code goes here
```

clocktree_name.**clk_test_clk(***clk_test_clk_name***);**

DESCRIPTION :

n/a

EXAMPLE :

CSL CODE

```
   //csl code goes here
```

clocktree_name.**clk_xtal(***clk_xtal_name***);**

DESCRIPTION :

n/a

EXAMPLE :

CSL CODE

```
   //csl code goes here
```

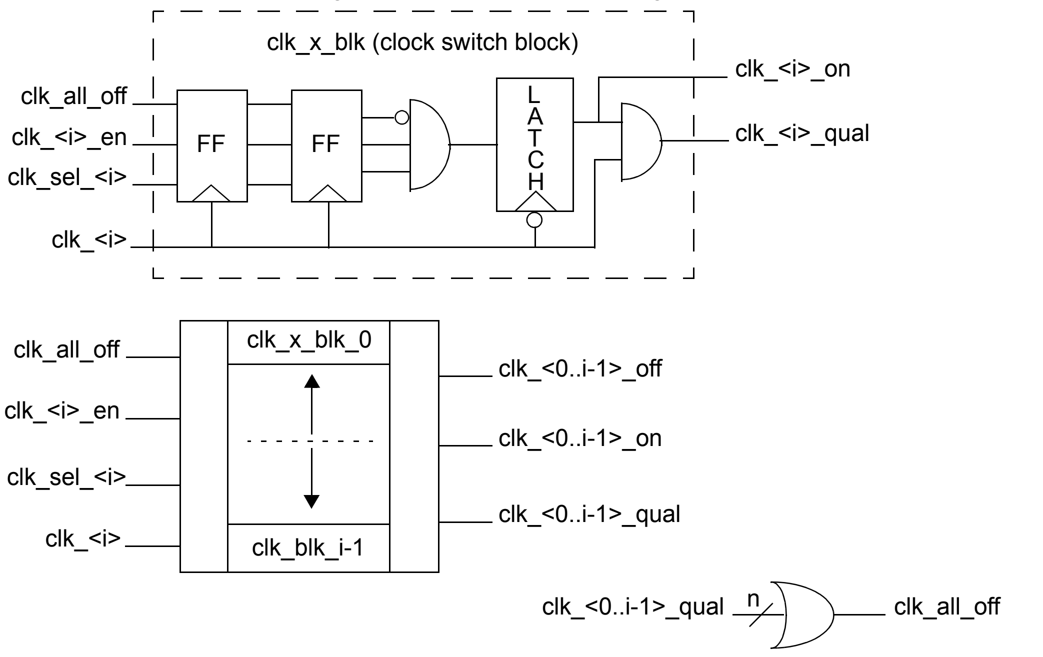
1.5 CSL Clock Tree Examples

1.6 CSL Clock Tree Checker

1.6.1 Clock switch

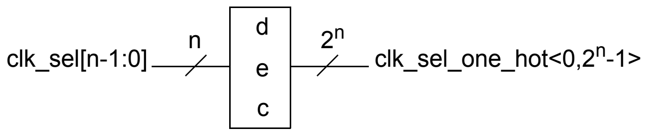
The clock switch has n clock inputs and n qualified clock outputs. The following is a description of the clock switch.
clk_<0..n+>_on all clks which are an input to the clock switch. In this example clocks clk_[0-(i-1)] are the outputs of the clock switch. Note that when the clock switch is switching between clocks that all clocks are off. When the clock switch is switching between one clock and another no clocks are enabled. Note that only one clock is on at any given point in time. There is “dead time” when switching from one clock to another.

FIGURE 1.1 Clock switch building block and top level block diagram



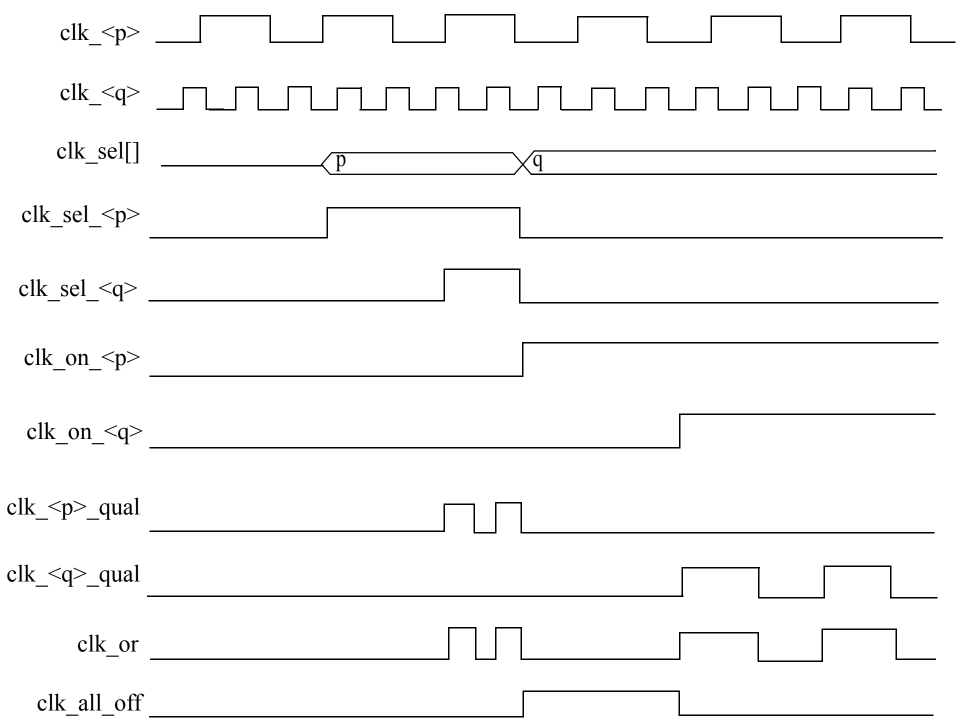
clk_<i>_qual is only high when the clk_<i>_en is high and the clk_<i> is on.
There is an external clock enable signal clk_<i>_en which is used to turn on the clock clk_<i>.
The signal clk_<i>_on indicates that the clock clk_<i> is turned on.

FIGURE 1.2 Clock select one hot decoder



There is an encoded clock select signal clk_sel[n-1:0] which is decoded into a one hot signal clk_sel_one_hot[2^n-1:0] which selects the current clock output. All clk<0,n-1> have to be off before turning clks on

FIGURE 1.3 Clock switch waveform



All clocks are at time zero. The output of the clock switch is low)no clock is selected as an output of the clock switch). Then the clk_sel line selects the clk_<p>. 2 clk_p cycle later the clk_on_<p> signal is turned on and the clk_<p>_qual signal is turned on.

1.7 csl_clock tree grammar

clk_en
clk_gate
clk_mux
clk_switch
clk_buffer
clk_pll
clk_div
clk_test_clk
clk_xtal

1.7.1 Clock enables

FIGURE 1.4 RTL view

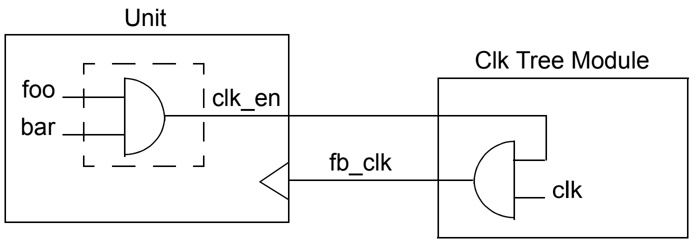
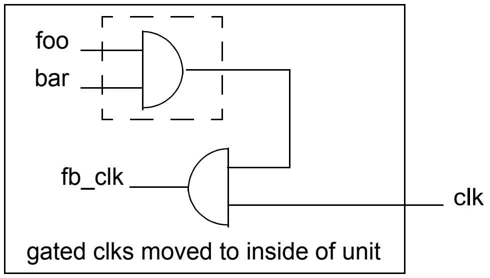
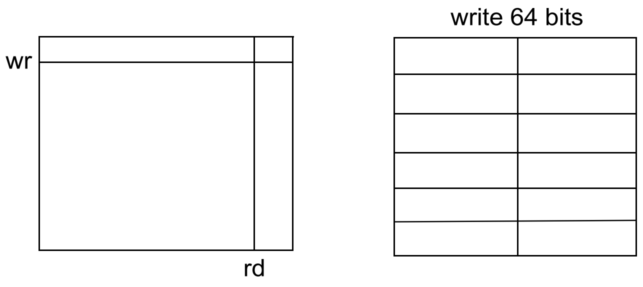


FIGURE 1.5 Layout



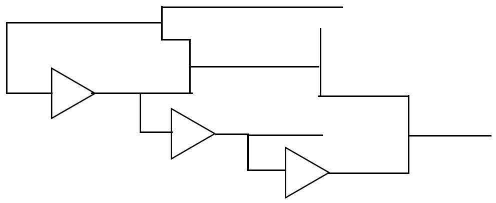
shapers and trimmers
clk tree generator with testing tested ?
transceiver on memory chip clk bypass cypress
Need test circuit to move the 10 arival
Need PLL
Need on chip sampler

FIGURE 1.6



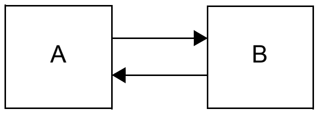
13 x (13 x 5)
8 x (8 x 5)
4 x 3 x (8 x (3 x 5)

FIGURE 1.7 Programable delay circuit



50 million bit celling satt?mers
pixel replication function
lots of problems with Virage and Artisan ram models
need mbist

FIGURE 1.8 Copy back and forth



checkpoint/restore
copy A -> B
copy B -> A
swap A -> B
swap B <- A

- need fast double bit error connection
- delay lines need to be robust
- compiling on delay lines causes timing problem -> data dependent
- need new model which integrates chip assembly and memory model instantiation

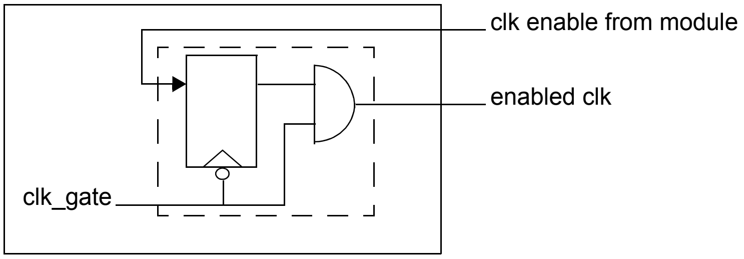
other memory applications:

FIFO
switches
BCH code system: 2^5 = 32

- add a bit: 2^6 = 64
- add a bit: 2^7 = 128

Fermat
Gabis
Vander Monde
Gauss
Newton

FIGURE 1.9 Clk network module



configure module

FIGURE 1.10 PLL configuration

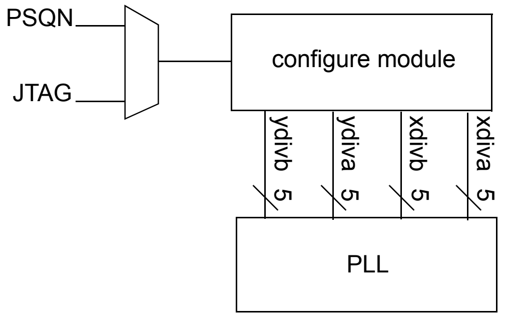


FIGURE 1.11 Equation for PLL frequency

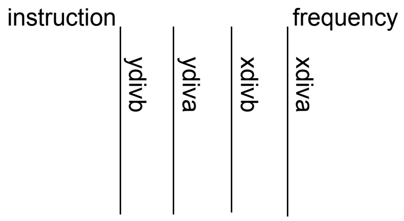
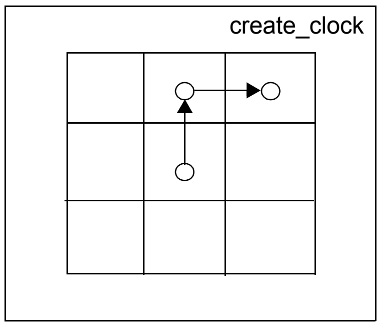


FIGURE 1.12 Create_clock

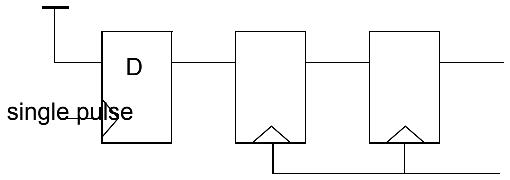


<move to clock synch>

1.7.2 csl_clk_sync

The following circuit is used to synchronize a pulse generated in one clock domain into a nother clock domain. Note that the single pulse signal drives the clock pin and the data pin (D) is tied to the Vdd.

FIGURE 1.13 Pulse synchronization

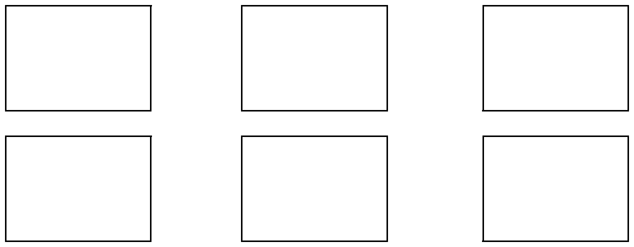


use gray code for counters which cross clk domains
gray code checker (“The Art of Electronic” , Horowitz)
</move to clock synch>

clk network	pads, pad macros, pll/dll/bypass, clk gates
reset network	
probe network	
port network	
memory bus/switch network	
memories	

<move to GUI>
modules as boxes

FIGURE 1.14

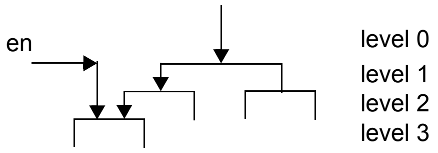


</move to GUI>

1.7.3 Clock tree construction

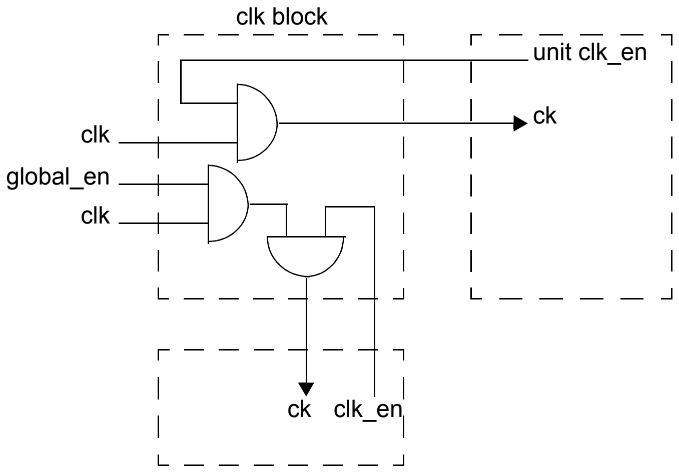
A clock tree can be enabled in different tree levels in the following figure each of the levels and each of the branches in each level can have its own gated clock

FIGURE 1.15 Hierarchical clock tree



The clock enable is generated in the module and sent to the clock gen block. Put some real clock enables in this figure (w/ FFs and latches)

FIGURE 1.16 Clock enables for the module

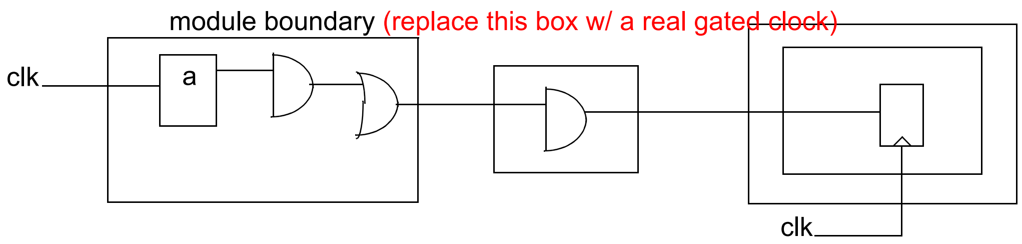


```
level0 (.inclk (clk), .outclk (clkaa)
level0buf ()
level0and () //number of fanouts
level1and ()
level2and ()
level3and ()
clkbuf_10_fo16  // (level and fanout)
```

1.7.4 Inserting RTL

- clk trees gatedclks
- clk muxes
- clk enables (latches)

FIGURE 1.17 Module level clock enables

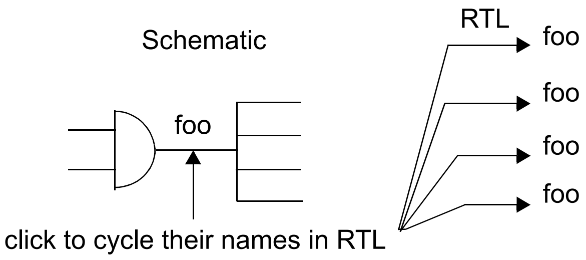

module boundary (replace this box w/ a real gated ~~clock~~)
clk
a
clk

Rules for checking clk domain

- clk_mux
- clk_or
- clk_and
- clk_FF

<move to GUI>
traverse clk tree
turn on test clk or sys clk
set the values of the top level pin
cycle based sim DFT

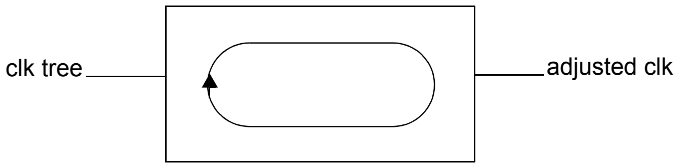
FIGURE 1.18



</move to GUI>
csim clk file
clk declaration
clk pwl waveform
clk phase relationship between 2 clocks
clk_phase_diff <clk0> <clk1> <time>

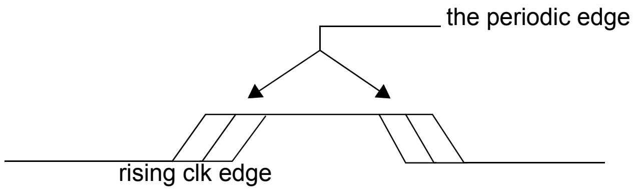
1.7.5 Feedback loop filter

FIGURE 1.19 Adjusts clock edge based on skew



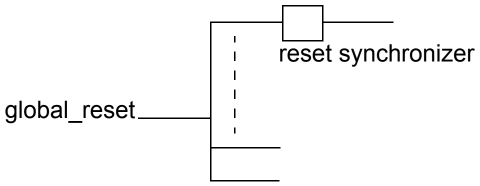
Digital Training loop

FIGURE 1.20 Counts the number of edges that fall +/- the periodic edge



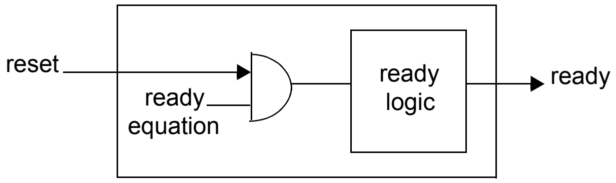
Artisan RAMS 500 MHz if you are carefull
clk gating for SRAMS

FIGURE 1.21 Global reset tree



1.7.6 Module reset csl_interconnect

FIGURE 1.22 Module reset



ready logic detects module ready conditions

1.7.7 Clock Specification file

The clock specification file has clock names, their associated frequencies and a piece wise linear

(PWL).
description of the clk waveform

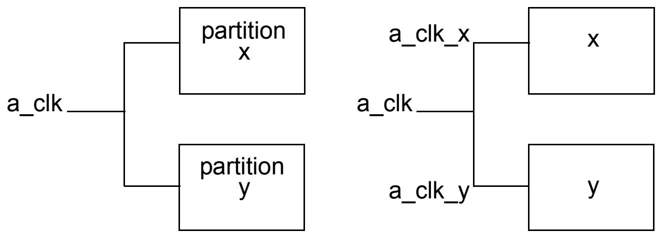
Example:

```
clock_spec <clk_name><period><PWL>
PWL : "{" # " , " # "}"
period : # {, #} *<time_scale>
time_scale: ms, ns, ps
```

1.7.8 Clock net creation rules

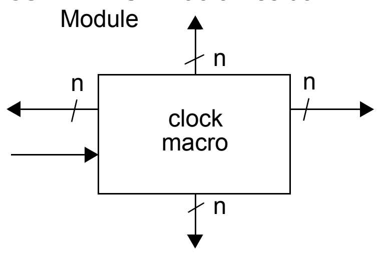
If a clock is used by more than one physical partition then the clock should split into two names. The layout team can then route the two different clock nets separately.

FIGURE 1.23 Partition level clock tree



```
x (.a_clk (a_clk));
y (.a_clk (a_clk));
assign a_clk_x = a_clk;
assign a_clk_y = a_clk;
x (.a_clk_x (a_clk_x));
y (.a_clk_y (a_clk_y));
```

FIGURE 1.24 Clk macro insertion

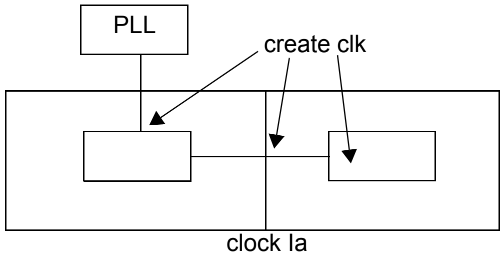


1.7.9 clk tree instructions/directives

```
   build_balanced_clk_tree (.root(<root>)
                                                .children (<list_of_children>)
```

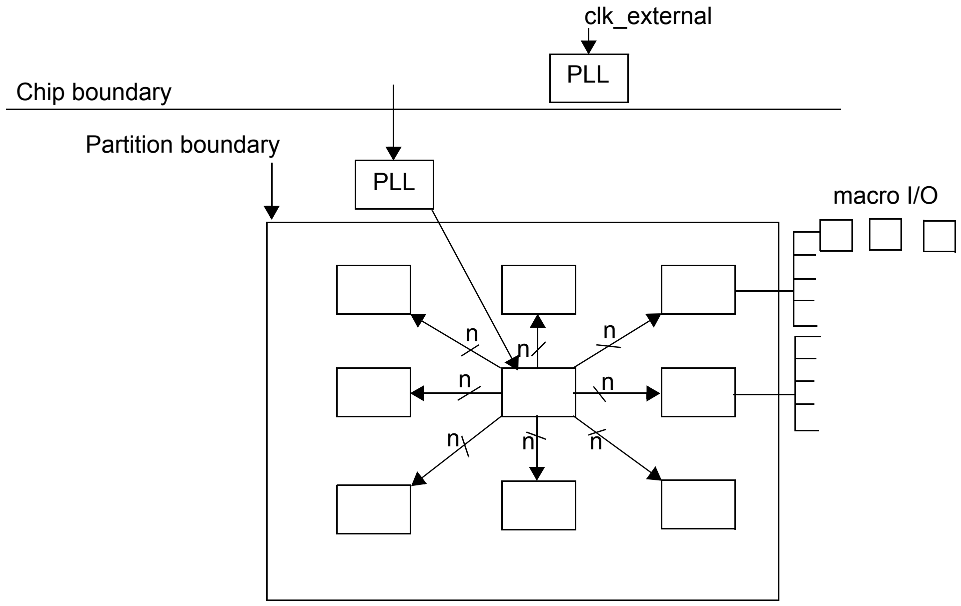
The generation/placement of the clk network, macrocells and memory cell performance network (PSQR network)
with Floorplaning and constraint generation DC primetime packages

FIGURE 1.25



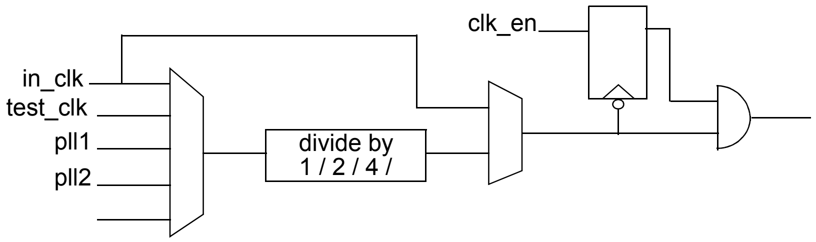
1.7.10 Clk Tree on chip clk_distribution

FIGURE 1.26 Clock tree



clk_delay_cell phase shift (mux which selectes different delays)
clk_shaper_cell moves rising/falling edges with respect to each other

FIGURE 1.27 clock logic after PLL



1.7.11 Clock tree analysis

if <name>_clk and <name>_clk_en are present in the net list then <name>clk_en should be defined under all conditions

- pre reset
- during reset
- after reset
- after all reset values propagate

Violation:
if clk is defined 0/1 and clk_en is X or Z then clk is not a fanout of clk_en and clk_en should either not be present (clk is not enabled) or clk needs to be gated by clk_en.

clk and clk_en naming convention
<name>_clk // clk and clken which are associated should ues the same prefix
<name>_clken // name and clk and clken suffixes

Optional
clks derived from other clks should use the other clocks name as prefix

```
ha_clk -> ha_new_clk = ha_clk
                       ha_new_clk_en
```

1.7.12 Clock network checker rule set

1. clocks can't be assigned to another clock
Not allowed in RTL:

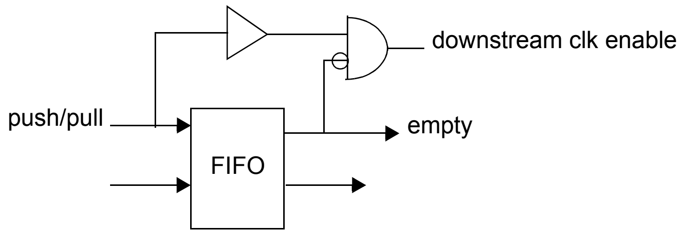
```
assign clka=clkb;
```

low power: turn off the clks

1.7.13 FIFO w/ clock qualified by FIFO not empty signal

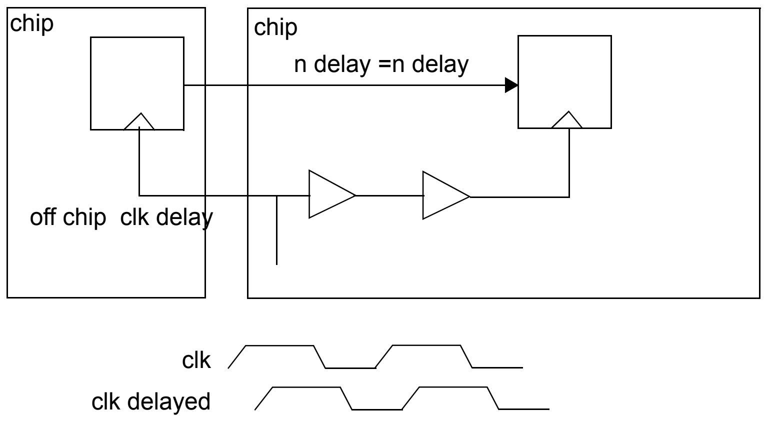
The downstream pipeline is enabled with the fifo not empty signal and a delayed version of the fifo push or pull signals

FIGURE 1.28 FIFO w/ clock qualified by FIFO not empty signal



fifo_idle=(empty & !push) latency estimates

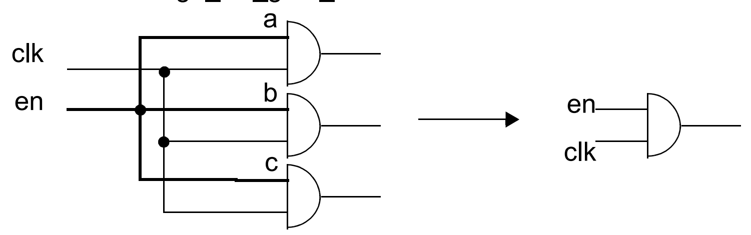
FIGURE 1.29 Clk latency estimates



clk tree

- merge_clk_gate_enables
- split_clk_gate_enables

FIGURE 1.30 merge_clk_gate_enables



report_ckl_gates -reset
with reset
add_reset_clk_gates -reset <resetname>

Building RTL CLK trees

Gated clock tree
The interconnect for a chip design contains data signals, control signals, clk signals and reset signals. In a chip some of the blocks that will not be used will be disabled. This can be done by building a clk tree which has a gated number of clocks (e.g. and or mux)
Building clock tree is imoprtant. we need to have attributes inside gen_interconnect which will tell if a module or a block is a clock enabled or not. Gen_interconnect will build a tree that includes clk enabled buffers which will turn that block on or off either explicitly through some kind of control bit or through of valid bit.

Checking RTL clock trees and state elements

RTL is an acronym for register transfer level hardware description language.

Inserting clock trees into RTL

CSLC will do the following

- be able to build balanced clock trees which have no skew
- be able to control when the clocks go off and on in modules to reduce the power
- add clock de-skewing circuits to “trim“ the clock
- define a set of rules for cecking the clock domain
  - clock mux
  - clock or
  - clock flip-flop
  - clock and
- define a set of gates for the modules (e.g. clk mux, clk flip flop, clk enable)
- be able to insert a test clock into the tree that’s why we need the muxes
  - Gated CLKs
  - CLK muxes

- CLK enables (latches)
- traverse cloc tree
- turn on test clock or sys clock
- set the values of the top level pin
- cycle based sim- DFT

<move to clock_sync>
Clock gates

FIGURE 1.31 Generated clocks

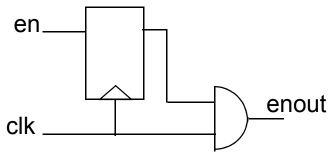


FIGURE 1.32 Generated clocks

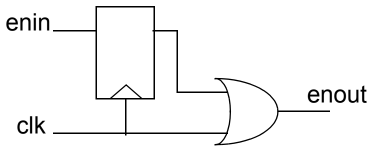


FIGURE 1.33 Clock mux cell

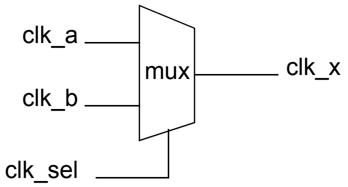


FIGURE 1.34 Clock glitches

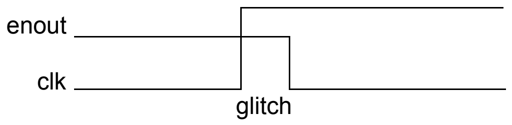


FIGURE 1.35 Clock wave form

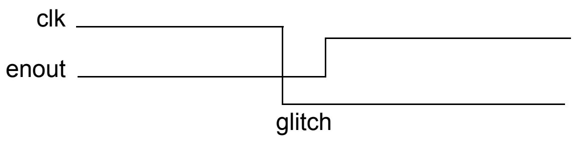
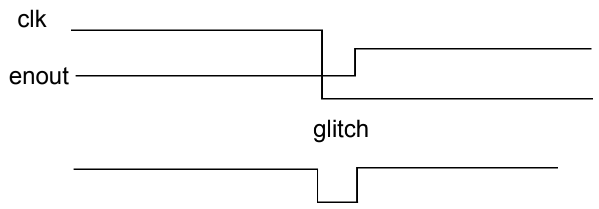


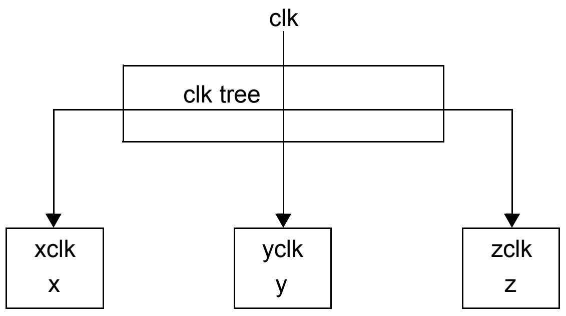
FIGURE 1.36



- section on Tristate buses CLK drivers for mutually exclusivity (mutex)

Transmission block takes an input signal and replicates the signal and routes the replicated signals to thier destinations.

FIGURE 1.37



```
virtual distribution_block {
in (clk);
prefix (use_module_name);
distribution_list; // optional buffer
}
```

</move to clock_sync>
<BEGIN OF ADD SECTION>

- adding the clock tree to the design
  - the clock tree includes enabled clock gates to control the clock for specific blocks
  - the clock enable logic can be generated in the module which the clock drives or in a central controller

</END OF ADD SECTION>

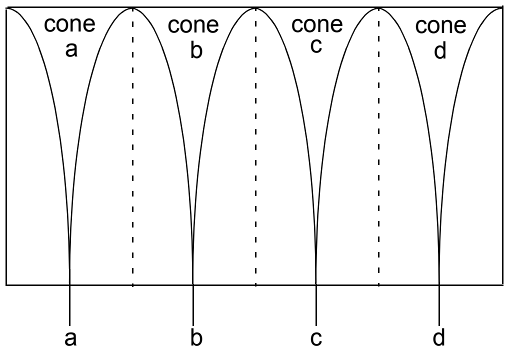
<ADD>
k clk synchronizers
ck for glitches

ck for inverse relationships
same (re ==~ we);
inverse (re, we);
gain based synth
logic structure
complex gates
global step

- slew
- fanout sensitive

Verilog 2001
System Verilog
optimize in parallel using threads.

FIGURE 1.38

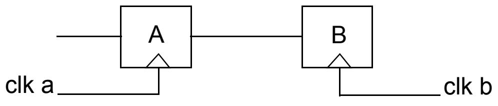


shell >
Don't place gates
place clusters

</ADD>

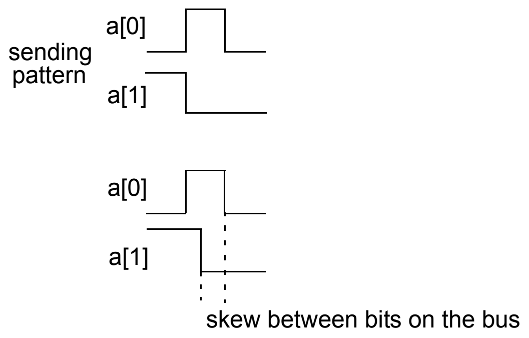
<ADD>
Timing

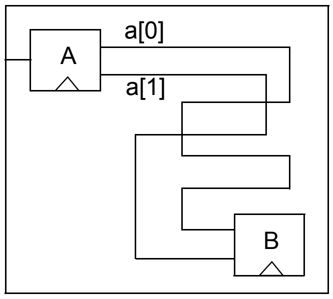
FIGURE 1.39



The length of the wire between A and B should not exceed the driver clock period. Otherwise the bits on the wire can arrive at B and the bits may be from different cycles due to different wire lengths.
"Line up the bits"

FIGURE 1.40two bit bus

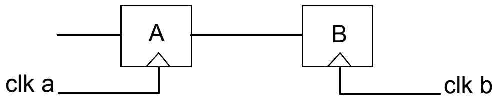




hold ck
A's effective clock period > longest travel time of a bit on "a" bus.

Timing
Hold check clcks in phase
clkb = 2clka
clka 1 1 1 1
clkb 1 1 1 1 1

FIGURE 1.41



clka 1 1 1 1
clkb 1 1 1 1

clks same frequencies but out of phase
clka 1 1 1 1
clkb   1 1 1 1
what is the minimum amount that they must be out of phase?
what are the ways to capture the data?

FIGURE 1.42negedge solution

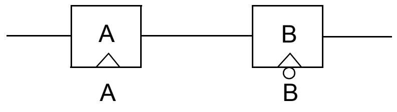
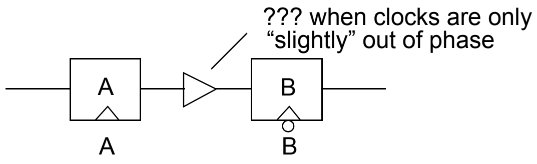


FIGURE 1.43delay solution

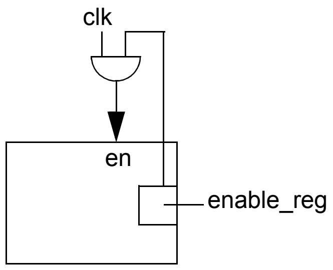


</ADD>

1.7.14 Unit level power management

A signal at the input turns on the pipeline when data is ready.

<new>


clk
en
enable_reg