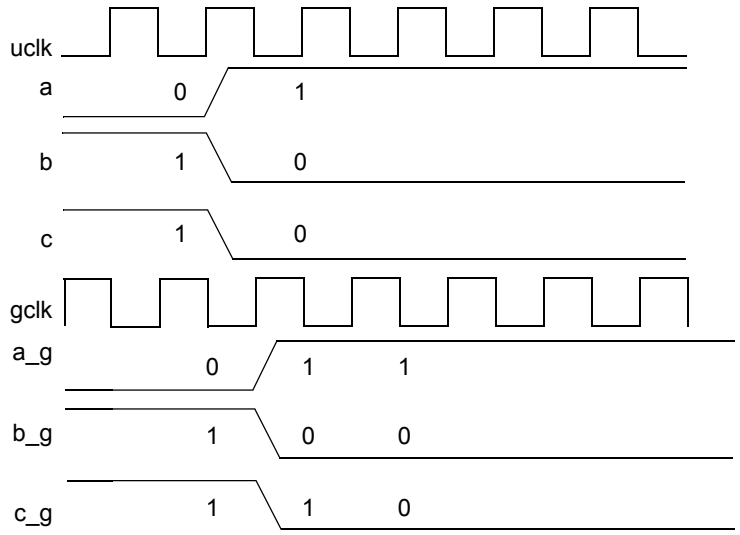
CHAPTER 1 CSL Clock Synchronizers

TABLE 1.1 Chapter Overview

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

1.1 CSL Clock Synchronizers Overview

FIGURE 1.1



- receiver clock >> sender clock
- receiver clock >= sender clock
- receiver clock <= sender clock
- receiver clock << sender clock

If there is more then one signal crossing clock domain use graycoding.
More than one bit is changing per cycle in the sequence: 000, 001, 011, 100.
Graycode - when only one bit position changes at the time - won't cause the problems.

000, 001, 011, 010, 110, 111, 101, 100
Avoid unnecessary switching of big data buses (try to keep last state if not used - power and noise issue).

Tristate all outputs and bidirects during reset.

Place idle tristate cycle between read/write on bidirects.

FIGURE 1.2

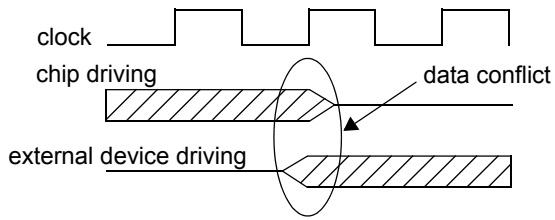
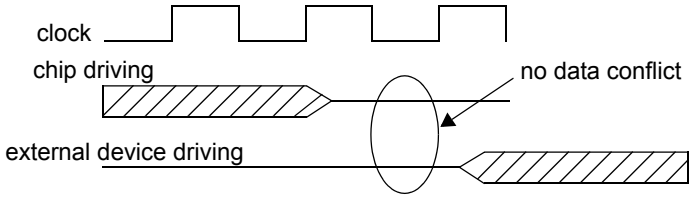


FIGURE 1.3



We believe in synchronous pipelined design.
Synchronous:

- don't use set and reset in flip flops for logical function
- no playing with the clocks
- no asynchronous feedback
- no negative edge flops
- no tristate buses
- no gated clock
- no latches
- no scannable flops functions for logical function

There may be EXCEPTIONS to this rule.
All the exceptions should be review.

```
always @ (b 0r c)
b = #1 b & c;

always @ (negedge reset_I)
a <= #1 reset_I & b;

always @ (negedge clk)
a <= #1 b;

always @ (posedge clk)
a <= #1 b & clk;

always @ (clk)
a <= #1 b;
```

1.2 CSL Clock Synchronizers Concepts

1.3 CSL Clock Synchronizers Commands summary

1.4 CSL Clock Synchronizers Commands

1.5 CSL Clock Synchronizers Examples

1.6 CSL Clock Synchronizers Checker

Move contents into the appropriate sections

Clock dmain crossing elements

If the clock domains are the same frquency but are out of phase with respect to each other then use a negatively triggered DFF on the reciever clock side to capture the data. Even if there are differences in the arrival times of the bits the individual bits should meet the setup time for the for the negatively triggered FF.

FFPE = flip flop positive edge triggered
FFNE = flip flop negative edge triggered

clka.freq == clkb.freq
clka.phase == omega * clkb.phase
which omega is the negative of the angluar frequency difference between the two clocks

```
    clka        clkb        clkb
     V           V           V
------FFPE------FFNE------FFPE
```

capture on the negative edge
If the clocks are different clock frequencies or the phase differences are very small then a clock

domain crossing FIFO should be used.Otherwise data from different clock cycles in different bits can be captured.

FIFO clock domain crossing issues that need to be handled:
meta stability
glitches
probability of a failure
Use grey coded counters for read/write counters to eliminate glitches in the counter outputs.
The glitches could be captured by the clock domain synchronizers which are used to synchronize the coutner values to the opposite clock domain.

FIFO status bits

full
The FIFO is full and cannot accept more data. Note that the full signal can assert after the HWM signal because datain flight may fill up the FIFO.

high water mark (HWM)
This signal is asserted when the FIFO fills up to or beyond the HWM. This is used to tell the upstream logic to stop sending data. The HWM is required when there is data in flight which cannot be stopped by a stall bit. An example is a switch fabric that does not have stall logic. There can be pipelines that continue to run after the receiveing unit with the clock domain FIFO stalls. The HWM tells the producer to stop sending data. The data in the pipeline is not lost because the HWM signal is generated when there is still room int he FIFO to accept more data. The HWM can be programmable.

low water mark (LWM)
This signal is asserted when the FIFO falls below the LWM. The LWM signal is used by down stream clients that need to know when the FIFO has reached a certain point. The HWM can be programmable. For example this information is useful to DMA transfer engines which need to know when there is a minimum amount of data in the FIFO. The minimum amount of data in the FIFO is equal to a transfer block size.

empty
The fifo is empty

PU = producer unit
PD = pipeline delay
RU = receiver unit

Stall signal sent to the PU from the RU. The stall signal is diriven by the HWM signal.
turn this into a figure

V |

PU====>PD=====>RU

Clock synchronization analysis

The clock file contains a list of clock name and their corresponding piece wise linear waveforms. The clocks are located in the clock file. Trace and tag the signals which comprise the clock tree(s) in the CDOM

- Identify the state elements (e.g. latches, memories, flip flops) in the tree
- find glitches in the clock gating
- Identify clock domain crossing points (CDCP)
- Identify the clock domain crossing point element types (CDCPET)

single pulse synchronizers
meta stability flip flops
FIFO CDCPET
other CDCPET/elements

If the CDCP element is a memory determine if it a is a FIFO or a SRAM or register file. If it is a FIFO then check what the address counter/increment mechanism is. Credit/debit or push/ pop.
If it is a FIFO then verify that the address counters are grey coded.
Is there a high low water mark used to control the upstream/ downstream flow control logic?

- identify upstream and downstream pipeline stall/enable logic

Analyze upstream pipeline stall/enable logic. How many cycles does it take to stall the upstream pipeline enable logic? This is the criteria for setting the high and low water marks in the FIFO. this is the buffer overflow check.
Check for buffer overflow at the input to the memory/fifo and buffer overflow at the output of the memory FIFO
Check the number of meta stability FF's at the clock domain crossing point.
Check for funky/rule violator CDCP structures per the RAN document.

FIGURE 1.4

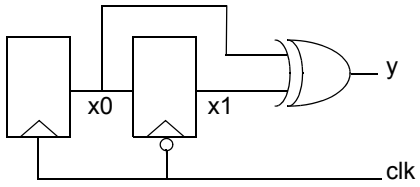


FIGURE 1.5

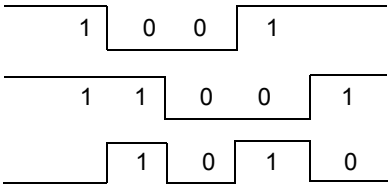


FIGURE 1.6

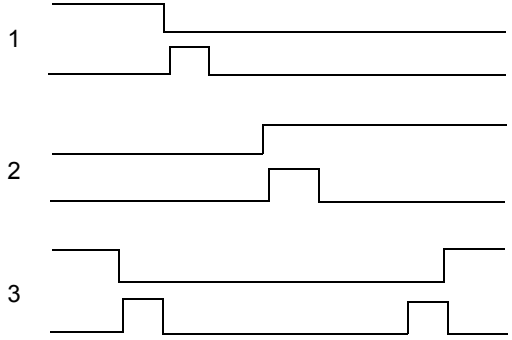


FIGURE 1.7

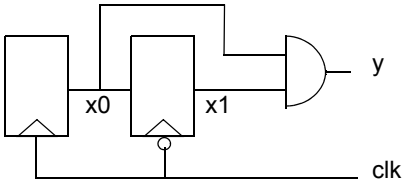
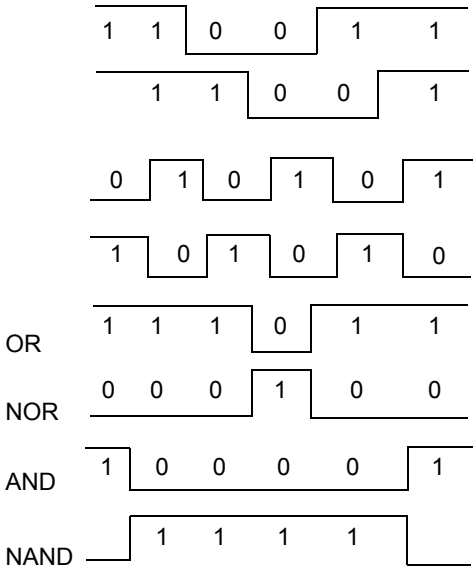


FIGURE 1.8



OMDB 45
make sections for
clock trimmer analysis

FIGURE 1.9

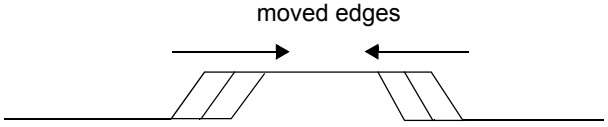
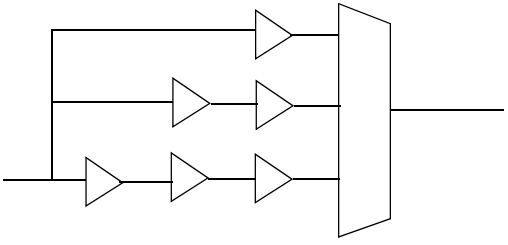


FIGURE 1.10 programmable clock delay



clk shapers analysis changes the rise/fall times

(the slope or slew rate)
transition time measured from
10-90
or
90-10

In table cross ref to 1.2 the different combinations of pos and neg edge ff's and gates is shown along w/ the problems asociated w/ the combinations

TABLE 1.2 Clock qualification logic table

0	FF	OR					X	X		
1	FFN	OR					X	X		
2	FF	NOR			X			X		
3	FFN	NOR		X	X			X		
4	FF	AND	X					X		
5	FFN	AND							X	
6	FF	NAND		X				X		X
7	FFN	NAND	X			X		X		X
	enable FF clock-ing	enable gate	clk glitc hclk pulse short ened	clk glitch glitch high	clk-stuck-low when enabled	clk stuck high when enabled	clk stuck high when enable d		clock tog-gles with glitche s when enable d	clock-inverse

OMDB 40A gated clk analysis
NEED same 8 case with latch instead of FF

FIGURE 1.11 FF NOR

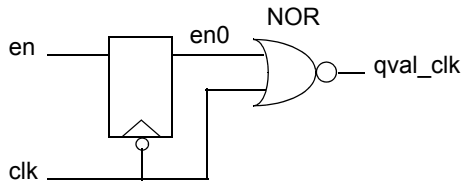


FIGURE 1.12

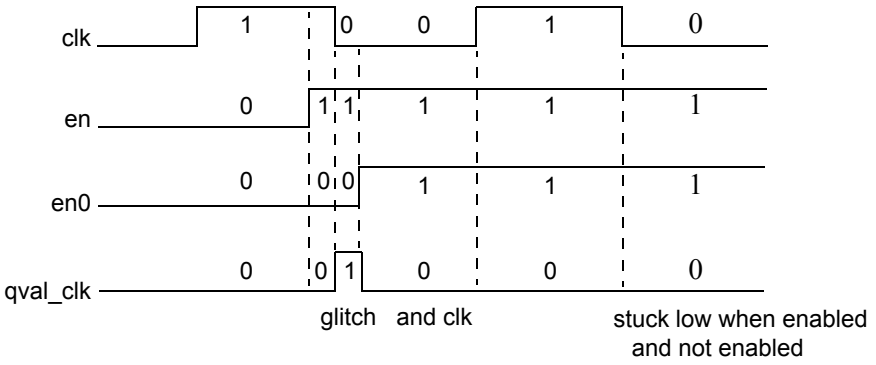


FIGURE 1.13

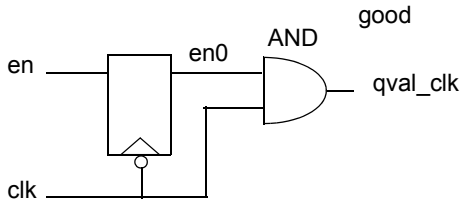
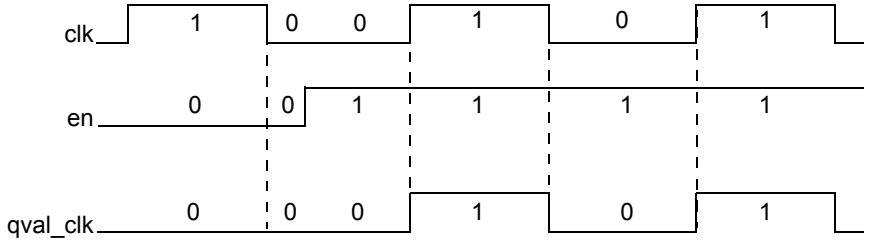


FIGURE 1.14



OMDB 41
clk analysis

FIGURE 1.15

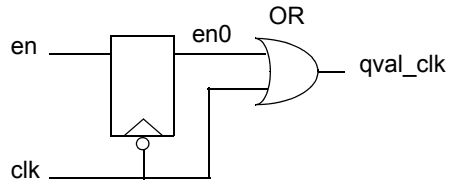


FIGURE 1.16

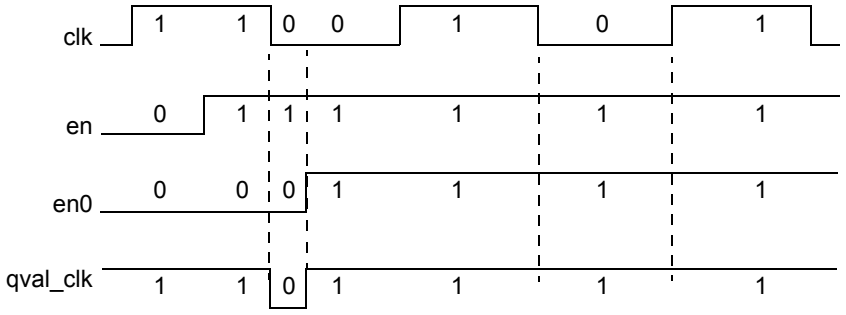
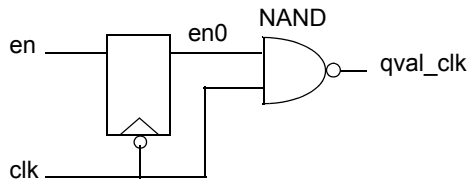
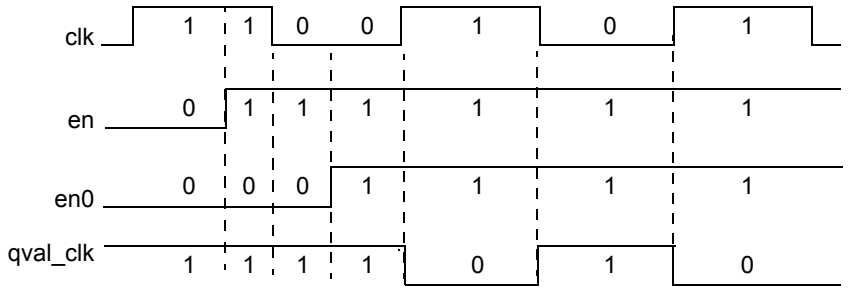


FIGURE 1.17



In the following figure a vertical line is drawn for each rising and falling edge

FIGURE 1.18



FF or OMDB 44 clk_analysis

FIGURE 1.19

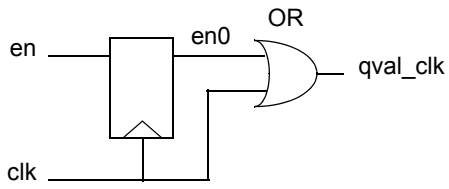


FIGURE 1.20

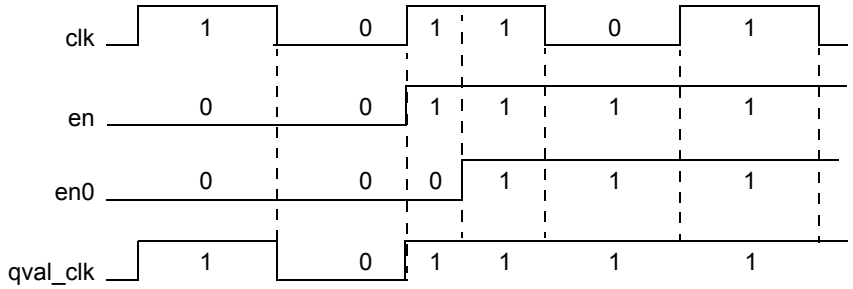


FIGURE 1.21 FF NOR

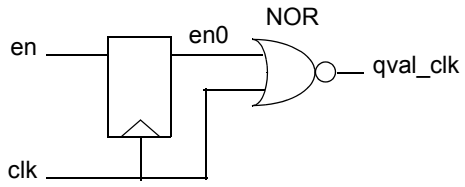
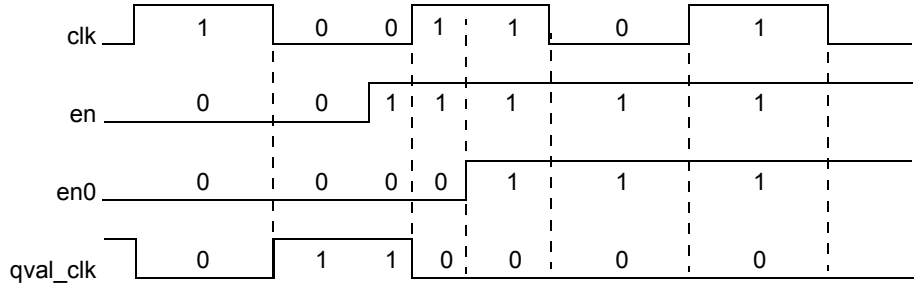


FIGURE 1.22



OMDB 43 clock_analysis

FIGURE 1.23 FF and NAND

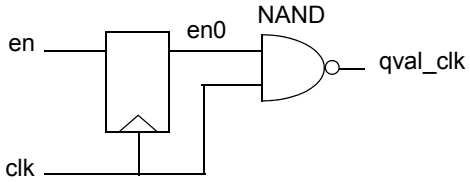


FIGURE 1.24

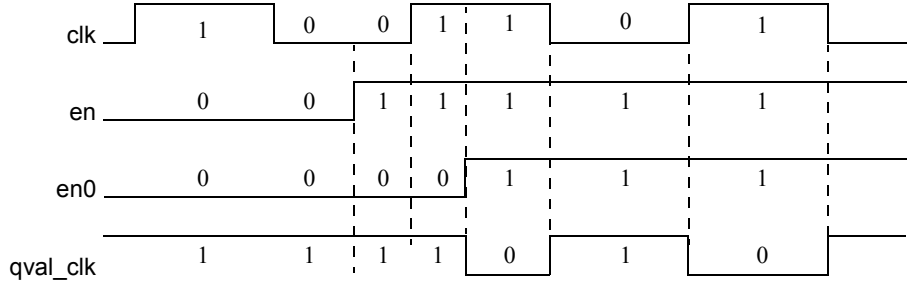


FIGURE 1.25

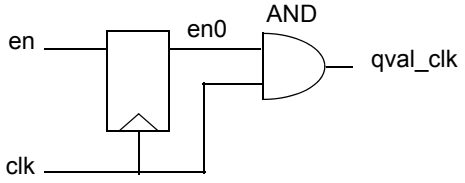
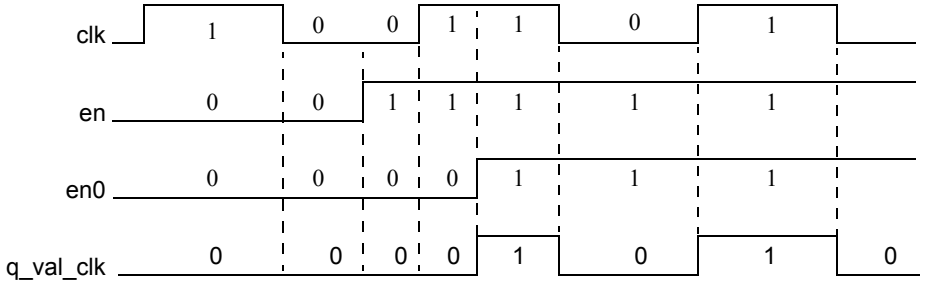
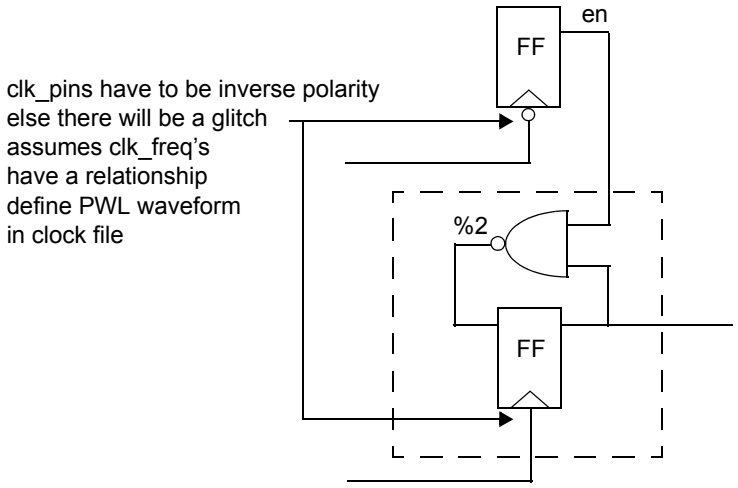


FIGURE 1.26



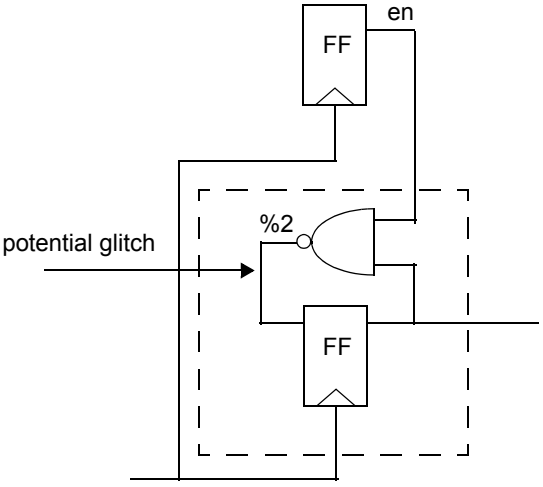
OMDB Analysis and OMDB 53
OMDB should recognize the following as a enabled clock divider and detect if there is a glitch

FIGURE 1.27



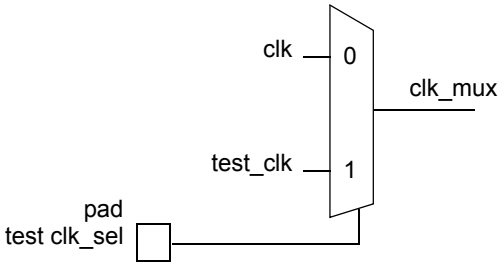
glitch case

FIGURE 1.28



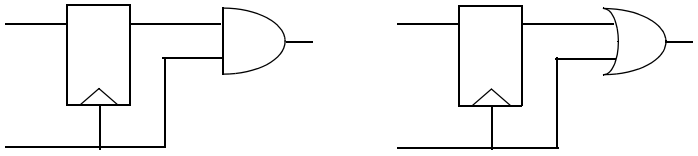
OMDB 54 clk_analysis
static clk mux select live
t hat is set during test mode
(prior to the timing on the clk)

FIGURE 1.29



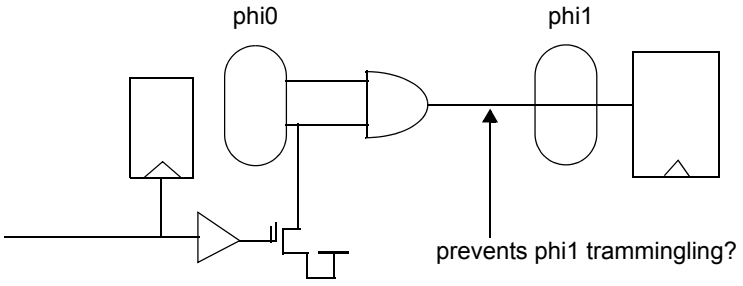
OMDB 55 clock analysis
clock tree analysis
need input waveforms PWL
analyze trees
find glitches
find enabled clks
find muxed clks

FIGURE 1.30



A clock period is divided into phi0 and phi1.

FIGURE 1.31



OMDB 56
clk domain crossing analysis
same frequency
out of pahse synchronizer

FIGURE 1.32

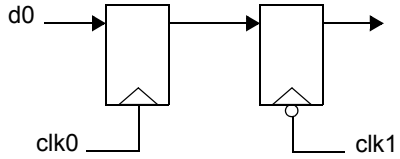


FIGURE 1.33

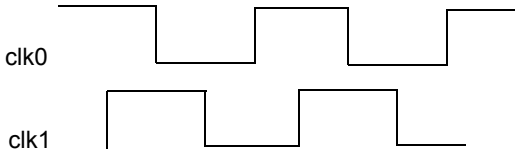


FIGURE 1.34

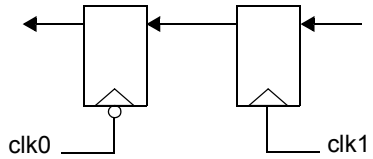
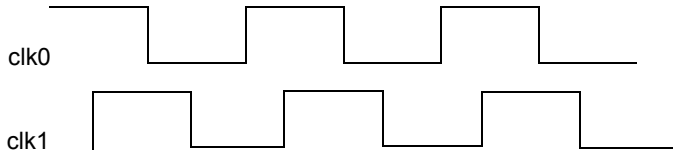
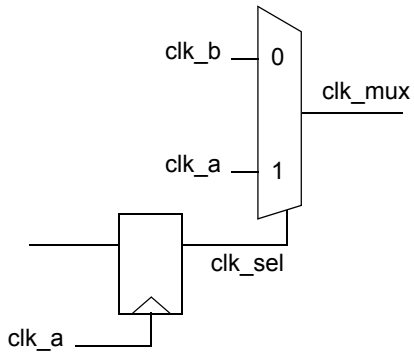


FIGURE 1.35



OMDB 57

FIGURE 1.36



tp=clkto qdelay

FIGURE 1.37 fix signal labels - match the figure above

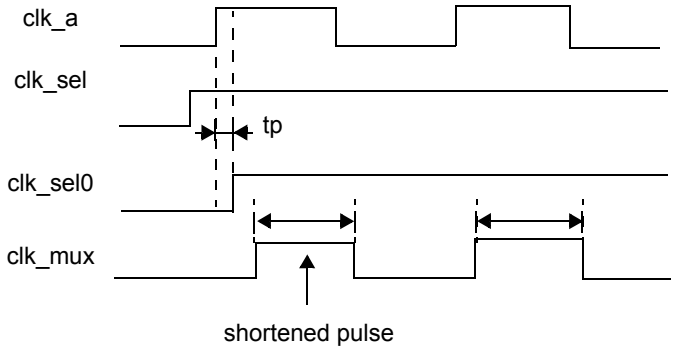


FIGURE 1.38

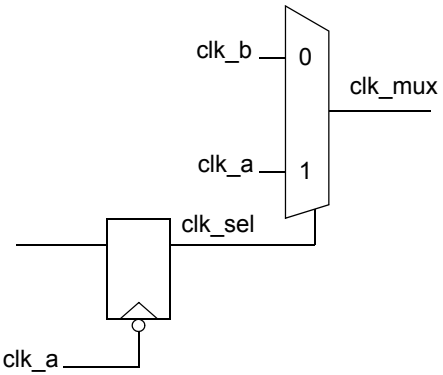


FIGURE 1.39

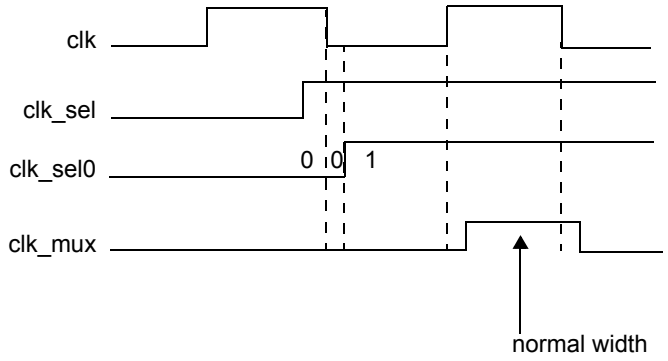


FIGURE 1.40

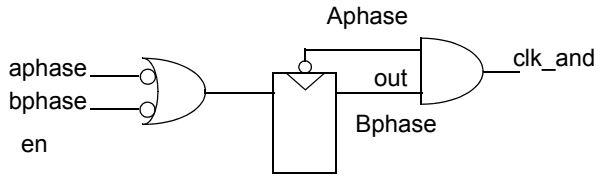


FIGURE 1.41 incomplete

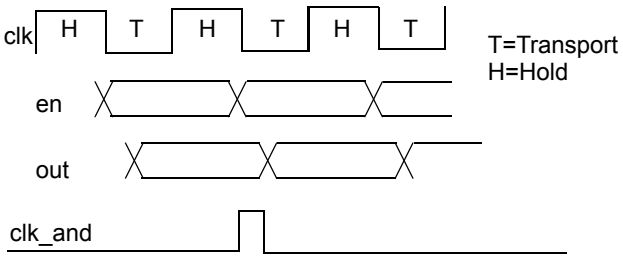
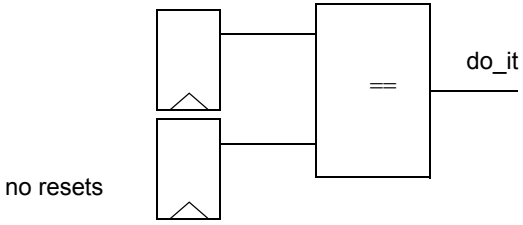


FIGURE 1.42 incomplete



reset problem
if do_it causes something to happen

```
module clken_buf (clk_out, clk_in, en_1);
output clk_out;
input clk_in, en_1;
reg clk_en;
 always @(clk_in or en_1) begin
  if (!clk_in)//transparent on low phase of clk_in
   clk_en=!en_1;
end
assign clk_out=clk_en&clk_in;
endmodule
```

FIGURE 1.43 incomplete

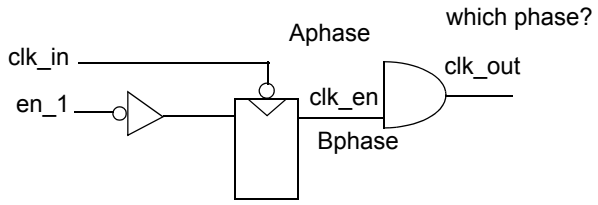
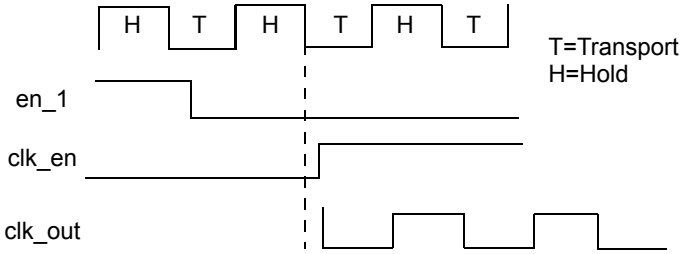


FIGURE 1.44 clk_out incomplete



RTL Clock tree analysis
RTL glitch analysis
A phase B phase analysis
clk butters
ckl latches
clk enables
asynchronous logic
-clk domain crossing
-in phase
-diff trimmers?
-multiples

FIGURE 1.45 Gated Clk

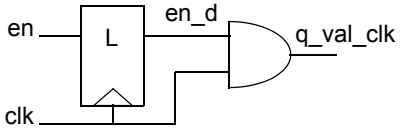


FIGURE 1.46

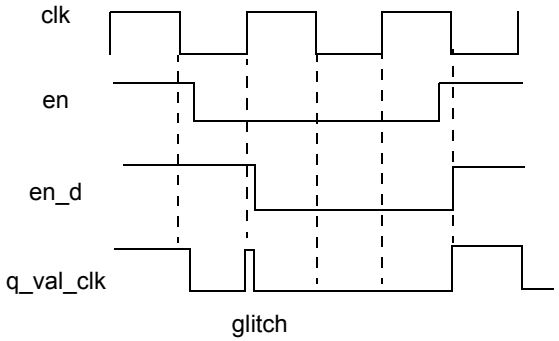


FIGURE 1.47 label signals

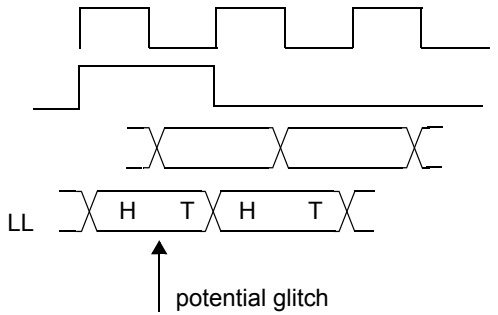
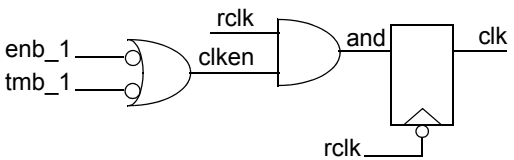


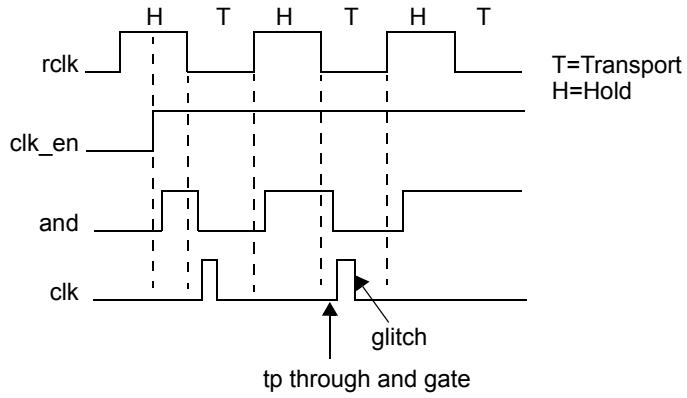
FIGURE 1.48




```
// CLK Header for gating the clock of a FF
// clk-output of the clk header
// rclk -input clk
// tmb_1- active low clock enable (in scan mode, this input is !se)
// enb_1 -active low clock enable
module  clken_buf (clk,rclk,enb_1,tmb_1);
output clk;
input rclk,enb_1,tmb-1;
reg clken;
always @ (rclk or enb_1 or tmb_1)
```

```
 if (!rclk)// latch opens a rclk low phase
  clk=clken&rclk;
assign clken=!enb_1| !tmb_1;
endmodule
```

FIGURE 1.49 RTL Analysis tool



Don't drive clk pin with data pin

FIGURE 1.50

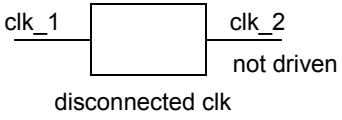


FIGURE 1.51

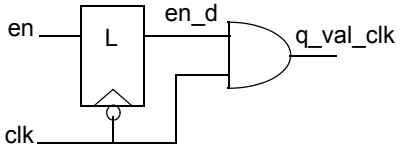
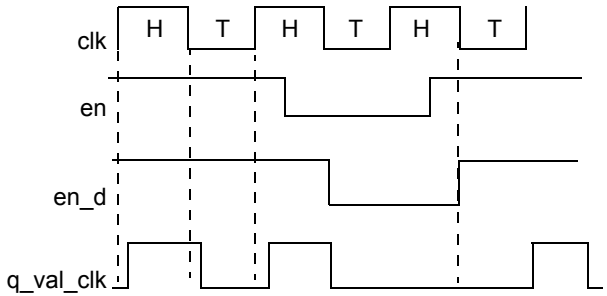
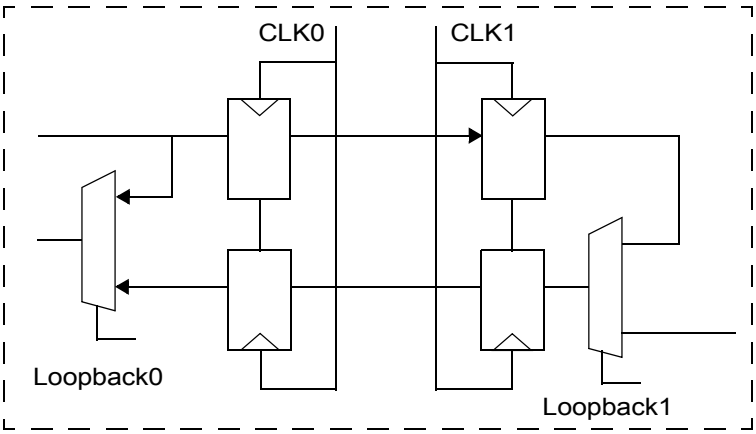


FIGURE 1.52



1.7 CSL Clock Tree Analysis

FIGURE 1.53 A debug mechanism to go across clock interfaces. label signals and correct fonts

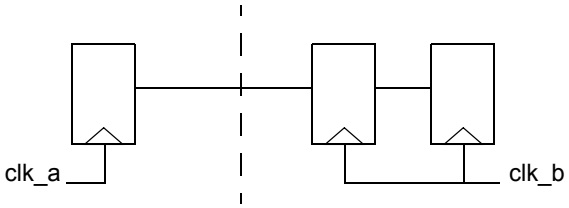


1.7.1 Instantiating Synchronizers:

Synchronizers: devices which are used to synchronize data going from one clock domain to another clock domain.The genclksyn should be able to identify constructs which would cause meta stability, and we should be able to insert constructs which will prevent meta stability.

FIGURE 1.54 This Figure shows 3 registers, and the left is a flip flop, the dashed line represent the boundary between 2 different clock domains and the 2 flip flops on the right are

the 2 synchronizers in the second clock domain which prevent meta stability from providing us a with a bit air raid.



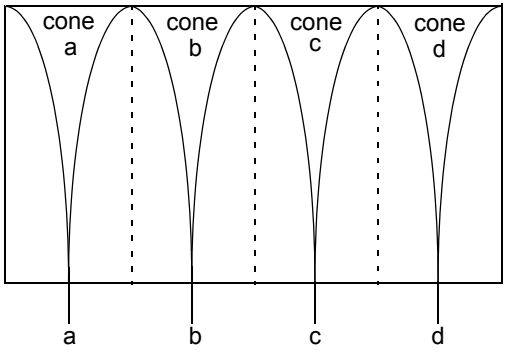
There are different kinds of synchronizers, the tool will check the synchronizers inside the RTL and makes sure that they have been instantiated correctly and that they function correctly, and we also can check if they are : Mesochronous or Pleisochronous

ck 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.

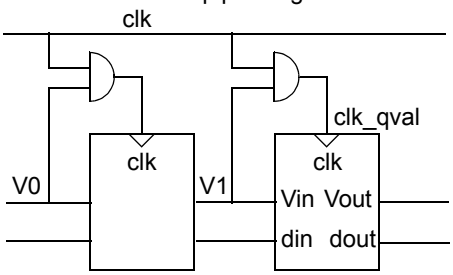
FIGURE 1.55



shell >
Don’t place gates

place clusters

FIGURE 1.56 micro pipelining



valid clk gating?

each pipeline turns on when it's needed.

<START OF ADDED FROM TRANSCRIBED SECTION>

2006_02_24_clk_dom_cross.fm

FIGURE 1.57

- receiver clock >> sender clock
- receiver clock >= sender clock
- receiver clock <= sender clock
- receiver clock << sender clock

If there is more then one signal crossing clock domain use graycoding.
More than one bit is changing per cycle in the sequence: 000, 001, 011, 100.
Graycode - when only one bit position changes at the time - won't cause the problems.

000, 001, 011, 010, 110, 111, 101, 100

2006_02_24_clk_dom_cross1.fm

Avoid unnecessary switching of big data buses (try to keep last state if not used - power and noise issue).

Tristate all outputs and bidirects during reset.

Place idle tristate cycle between read/write on bidirects.

FIGURE 1.58

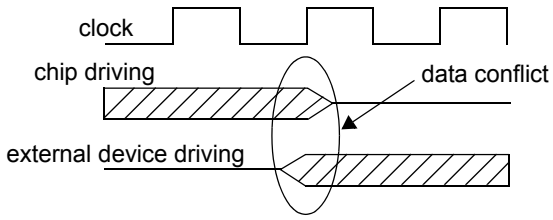
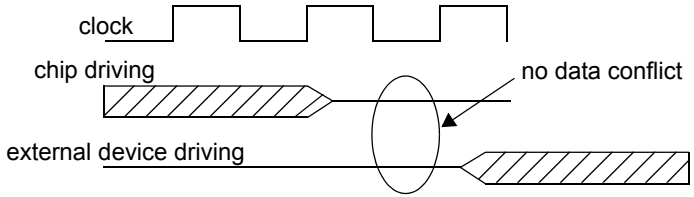


FIGURE 1.59



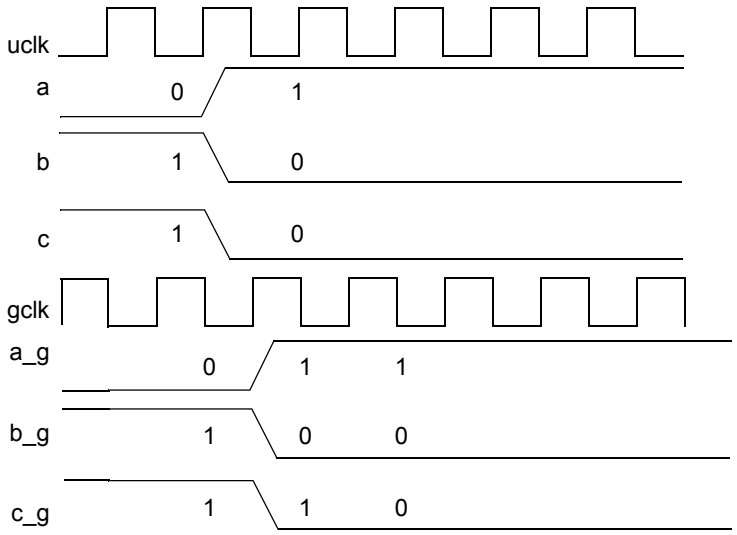
We believe in synchronous pipelined design.
Synchronous:

- don't use set and reset in flip flops for logical function
- no playing with the clocks
- no asynchronous feedback
- no negative edge flops
- no tristate buses
- no gated clock
- no latches
- no scannable flops functions for logical function

There may be EXCEPTIONS to this rule.
All the exceptions should be review.

```
always @ (b 0r c)
b = #1 b & c;

always @ (negedge reset_I)
a <= #1 reset_I & b;

always @ (negedge clk)
a <= #1 b;

always @ (posedge clk)
a <= #1 b & clk;

always @ (clk)
a <= #1 b;
```

FIGURE 1.60



- receiver clock >> sender clock
- receiver clock >= sender clock
- receiver clock <= sender clock
- receiver clock << sender clock

If there is more then one signal crossing clock domain use graycoding.
More than one bit is changing per cycle in the sequence: 000, 001, 011, 100.
Graycode - when only one bit position changes at the time - won't cause the problems.

000, 001, 011, 010, 110, 111, 101, 100

<ADD>

FIGURE 1.61glitches

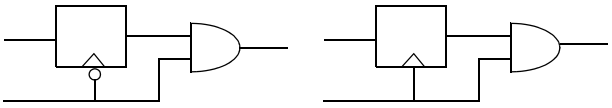
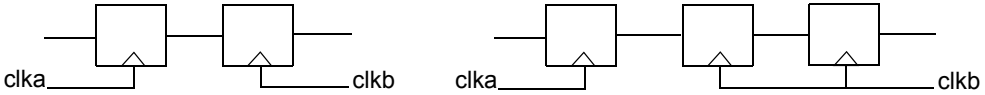


FIGURE 1.62meta stability



</ADD>
</END OF ADDED FROM TRANSCRIBED SECTION>

<ADD>
//from fifo
Metastability equation
mean time between failure(MTBF) equation
Metastability analogy: Imagine a ball bouncing at the top of a hill with two equal slopes. There are two cases which describe what happens to the ball:
A) ball can roll down either slope
B) ball bounces back and forth at the top of the hill

FIGURE 1.63 Metastability

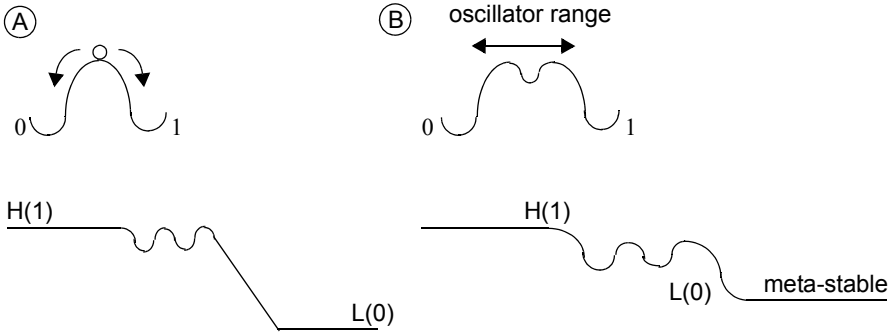
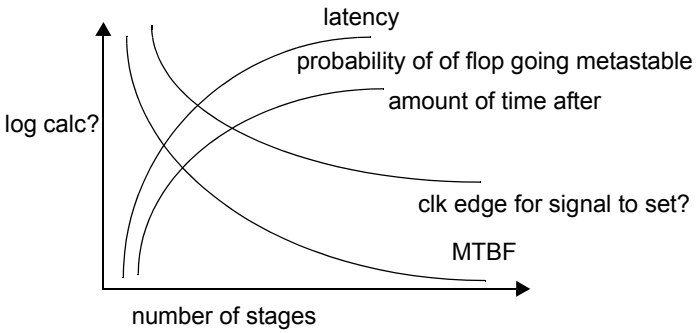


FIGURE 1.64



The outputs are then combined to generate a keep out signal that controls the multiplexer of a two-register t othat of figure ????. If the predicted clock tell into the keep-out region one cycle ago, then the actual clock is in the keep-out region during the current cycle. In this case the upper flip-flop of

the two register synchronizer may be in a metastable state. to sampling this flip-flop, the keep out signal controls the multiplexer to select the lower flip flop
In essence the circuit operates by using the time advance of the clock predictor to cancel the time delay of the synchronizer wait. The result is that a reliable synchronized keep-out signal is generated simultaneously with the sampling event it corresponds to.

1.7.2 General Purpose Asynchronous Synchronizers

The synchronization methods described in Section 10.3 cannot be applied if events on the external signal are aperiodic, or if the clock is not periodic. In this case, the synchronization cannot be performed in advance by predicting the timing of events. Rather, synchronization must be performed explicitly in response to each event, introducing a synchronization wait into the latency of the data stream.

1.7.3 Waiting Synchronizer

The most straightforward method of synchronizing a truly asynchronous event stream is to use the waiting synchronizer illustrated in figure 10-6 and discussed in Section 10.3.1.4. As described, this approach introduces a synchronization delay tp typically one or more clock cycles, and has a non-zero probability of synchronization failure.
</ADD>