Synthesis and Timing Verification Tutorial

By

Dr. Ahmet Bindal

Computer Engineering Department
San Jose State University

A. Synthesis

This tutorial introduces the basics of **Cadence's Synthesis and Timing Verification** tool (***Ambit BuildGates Synthesis***), and how to obtain a gate-level netlist from a Verilog RTL code.
To learn more on Synthesis commands or scripts please refer to **Synthesis and Timing Verification Manual.**

(i) Getting started

(a) First, make a directory called "**synthesis"** in your home directory.

(b) Generate the following Verilog RTL files pertinent to Figure 1 in the "**synthesis"** directory.

In this figure:
The **top-level** module is ***my_design.v***
The full adder module **under** my_design.v is ***full_adder.v***
The 2-1 mux and flip-flop modules **under** my_design.v are ***mux.v*** and ***ff.v***, respectively.

Note: all the Verilog RTL files related to Figure 1 are given in **Appendix A**.

my_design:

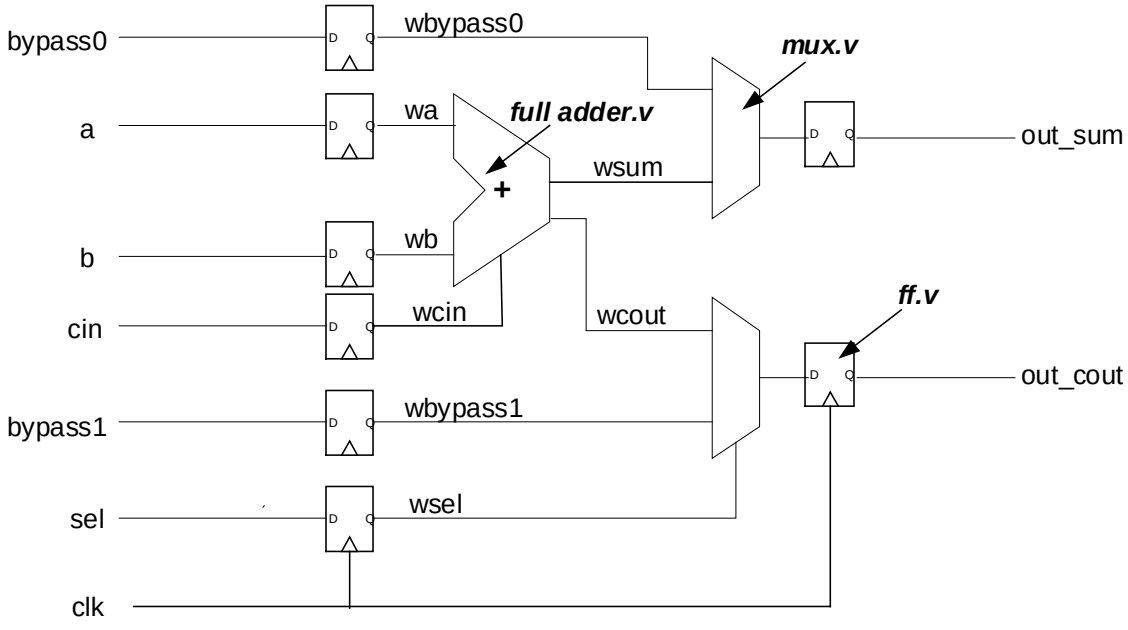


Figure 1

(ii) Prepare the "environment" template for Synthesis, "setup.tcl"

Generate the **setup.tcl** file in "**synthesis"** directory.

setup.tcl file:

```
proc setup {} {

# show all commands on the monitor as they are executed
set_global echo_commands true

# set maximum fanout limit at each synthesized gate
set_global fanout_load_limit 12

# read all the library cells used during synthesis
read_alf /apps/cadence/bg40/lib/lca500k.alf

# transform lca500kv to a name called "target_techology" for future use
set_global target_technology lca500kv

# include wireload models into library elements
read_library_update /apps/cadence/bg40/lib/lca500k.wireload3

}
```

(iii) Synthesizing the Verilog RTL code

(a) Launch ac_shell:

To launch synthesis, type **ac_shell -gui** at the prompt sign when you are still in **"synthesis"** directory.
An **"ac_shell"** window will pop up.

(b) Reading setup.tcl file:

Go to the prompt sign on the **"ac_shell"** window.

To have Synthesis tool read your setup.tcl file type:
ac_shell> **source setup.tcl**.
To run the script in the setup.tcl file type:
ac_shell> **setup**

(c) Reading Verilog RTL files:

In order to have Synthesis tool read your Verilog files go to the **"ac_shell"** window, and type the following:
ac_shell> **set top "my_design"**

This command assigns **"my_design"** as the top-level module in your design.

Next, load all the Verilog files into the Synthesis tool. To do this go to the **"ac_shell"** window and type the following:
ac_shell> **read_verilog my_design.v full_adder.v ff.v mux.v**

(d) Generate a netlist:

To generate a netlist corresponding to your Verilog RTL files, type the following on the **"ac_shell"** window:

ac_shell> **do_build_generic –module my_design**

(e) View the schematic of the design:

Once the gate-level netlist is generated then you can view your final schematic.
To view the schematic:
From within the module browser shown in Figure 2, double click the **my_design** module.
The schematic of **my_design** will display on the schematic window.

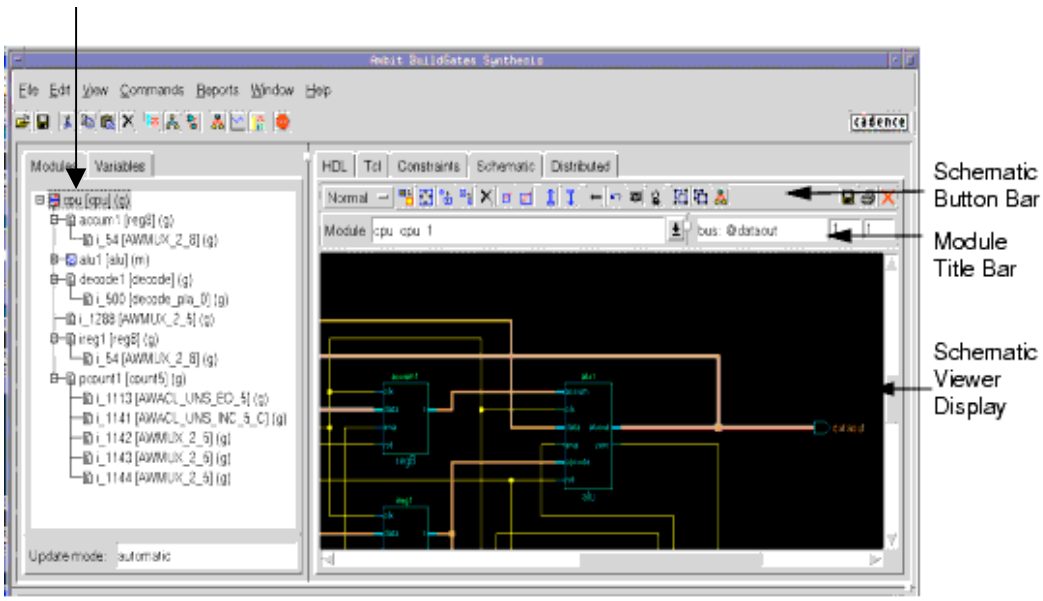


Figure 2

(f) Zoom in and out on the schematic:

To zoom in, use the left mouse button, click and hold on the schematic and move downward.
You can also use the **"zoom in"** and **"zoom out"** buttons ***right above*** the schematic window.

(g) Hierarchical schematic views:

To see the gates in each box of the top-level schematic highlight a flip-flop with your left mouse button and double click the box. You will see the gates that make up the flip-flop. Repeat this process for the full adder and the 2-1 mux in your top-level schematic and examine each cell.
You can use the **"hierarchy up"** and **"hierarchy down"** buttons ***right above*** the schematic window to go to different cell levels in the schematic window.

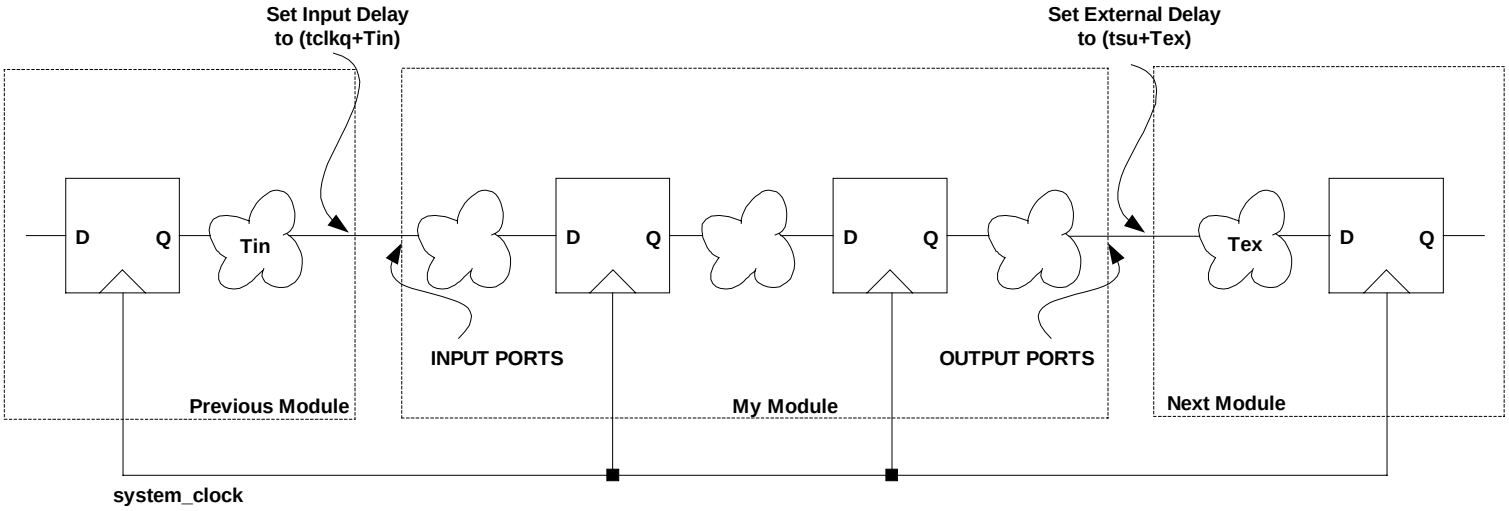
B. Timing Verification

(i) Prepare the "timing verification" template for timing verification tool, "timing.tcl"

You need to generate another template in your **synthesis** directory for timing verification after synthesizing your Verilog module(s) in order to see timing violations that this synthesized circuit may have caused. The following **timing.tcl** file is a very basic file to execute timing verification.

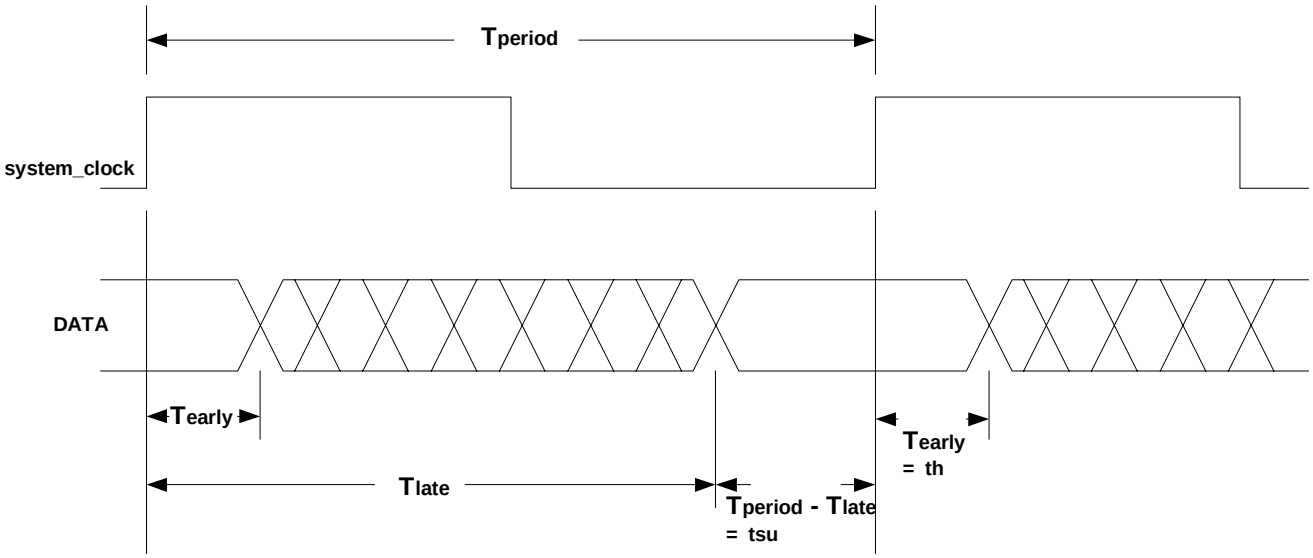
timing.tcl file:

```
proc timing { } {


#  Defining an ideal clock
#  ******************
# -waveform {leading_edge   trailing_edge}
# -period: the value of the period
# "ideal_clock" is the name of the clock
# -clock: specifies the name of the ideal clock
# -pos: the positive edge of the ideal clock
# -neg: the negative edge of the ideal clock
# ***********************************
set_clock ideal_clock -waveform {0 4} -period 10
set_clock_root -clock ideal_clock -pos module_clock

# Source all_inputs
# ***************
proc all_inputs {} {find -port -input -noclocks "*"}

# Source all_outputs
# ****************
proc all_outputs {} {find -port -output "*"}

# Defining the set-up and hold times for all input(s) with respect to ideal_clock
# -early refers to a set-up time value for your input(s)
# -late refers to a hold-time value for your input(s)
# ***************************************
set_input_delay -clock ideal_clock -early 0.1 [all_inputs]
set_input_delay -clock ideal_clock -late  0.2 [all_inputs]

# Defining the set-up time for the next module's input ports
set_external_delay 0.0 -clock ideal_clock [all_outputs]

# Defining the drive (output) resistance of your input(s)
set_drive_resistance 0 [all_inputs]

}
```



Previous Module must supply the **earliest** and **latest** *data* **arrival** times to my **INPUT PORTS** with respect to clock. These data **arrival** times constitute my *input delays*, and they have to be specified by **set_delay_input** command in my **timing.tcl** file.

Next Module must supply the **earliest** and **latest required** *data* times at my **OUTPUT PORTS** with respect to clock. These **required** data times constitute my external output delays, and they have to be specified by **set_external _delay** command in my **timing.tcl** file.



EXAMPLE:

set_input_delay -clock system_clock -early (tclkq + Tin_min) -late (tclkq + Tin_max) -rise [all_inputs]

set_external_delay -clock system_clock -early (tclkq) -late (Tperiod - tsu - Tex_max) -rise [all_outputs]

(ii) Define the top-level module for timing verification

Just like what you have done for Synthesis you need to define the top-level module for timing verification.
Go to the **"ac_shell"** window and type the following:

ac_shell> **set_top_timing_module $top**
ac_shell> **set_current_module $top**

set_top_timing_module is a command that identifies the module, **$top**, to be used in subsequent steps to apply timing constraints.
The module, **$top**, was the variable, **my_design,** in an earlier command.

set_current_module sets the module, **$top**, as the top-level *current* module.

(iii) Run the timing constraint script, timing.tcl

Just like what you have done for Synthesis you need to run the script file(s) for timing verification.
To run the timing.tcl file, go to the **"ac_shell"** window and type the following:
ac_shell> **source timing.tcl**

To run the timing.tcl script type the following in the **"ac_shell"** window.
ac_shell> **timing**

(iv) Prepare the "timing report" template for timing verification tool, "report.tcl"

You need to generate another file under "synthesis" directory that manages all the timing related reports of your circuit following a timing verification step.

report.tcl file:

```
proc report {} {

mkdir report
mkdir netlist

report_timing > report/timing.rpt
report_area -hier -cell > report/area.rpt
report_hierarchy > report/hierarchy.rpt

write_verilog -hier netlist/my_design.net

}
```

As you can see this script generates a **"report"** and a **"netlist"** directory under **"synthesis"** directory. It subsequently forms 3 files, ***timing.rpt***, ***area.rpt*** and ***hierarchy.rpt*** under **"report"** directory, and 1 file, ***my_design.net*** under **"netlist"** directory.

(v) Generating the reports

To generate the timing reports, compile the report.tcl script by typing:
ac_shell> **source report.tcl**

Furthermore, run the script by typing:
ac_shell> **report**

Refer to **Appendix B** for more detail on the timing report.

(vi) View the reports

You can use the ***"Tcl"*** button on **"ac_shell"** window or a UNIX window to view the **timing.rpt.**
Note that, by default the report lists the ***most critical path*** in the design.
In the timing.rpt, search for ***slack time***. If it is a negative slack, it means a time violation.
Experiment by changing the clock period to avoid the timing violation if there is one.

Hint: Editing the timing constraint file, **timing.tcl**, can correct the violation.

Also view the ***area.rpt*** and ***hierarchy.rpt*** in the **"report"** directory along with ***my_design.net*** in the **"netlist"** directory. Understand the contents of each file.

C. Optimization

One last thing you should experiment is to optimize your gate-level netlist to reduce the propagation delay between the flip-flop boundaries and perhaps to reduce its real estate foot print on the chip (circuit area).

(a) Viewing unoptimized cells after the synthesis:

Before starting to optimize the top-level schematic, **my_design**, view the contents of your **unoptimized** full adder.

On the top-level schematic, **my_design**, find your full adder, and double click the **full_adder** module, view and print your **unoptimized** full adder schematic.

(b) Optimize the design:

To Optimize the design, type:
ac_shell> **do_optimize**

(c) Re-view the schematic of the design:

Now, *double click* the newly created **full_adder** module.
Print and compare the difference between the unoptimized and optimized full adder schematics in terms of propagation delays and area.

(d) Exit ac_shell:

Type:
ac_shell> **exit**.

Appendix A

Verilog RTL files:

(a) The top level-module, my_design.v:

```
module my_design (bypass0, bypass1, module_clock, rst, a, b, cin, sel, out_sum,
out_cout);

input bypass0, bypass1, module_clock, rst, a, b, cin, sel;
output out_sum, out_cout;

wire wbypass0, wbypass1, wa, wb, wcin, wsel, wout_sum, wout_cout;

ff ff1 (.clk(module_clock), .rst(rst), .d(bypass0),   .q(wbypass0));
ff ff2 (.clk(module_clock), .rst(rst), .d(bypass1),   .q(wbypass1));
ff ff3 (.clk(module_clock), .rst(rst), .d(a),        .q(wa));
ff ff4 (.clk(module_clock), .rst(rst), .d(b),        .q(wb));
ff ff5 (.clk(module_clock), .rst(rst), .d(cin),      .q(wcin));
ff ff6 (.clk(module_clock), .rst(rst), .d(sel),      .q(wsel));
ff ff7 (.clk(module_clock), .rst(rst), .d(wout_sum),  .q(out_sum));
ff ff8 (.clk(module_clock), .rst(rst), .d(wout_cout), .q(out_cout));

full_adder fa (.a(wa), .b(wb), .cin(wcin), .sum(wsum), .cout(wcout));

mux  mx0 ( .a(wbypass0), .b(wsum),  .sel(wsel), .out(wout_sum));
mux  mx1 ( .a(wbypass1), .b(wcout), .sel(wsel), .out(wout_cout));

endmodule
```

(b) full_adder.v:

```
module full_adder (a, b, cin, sum, cout);


input  a, b, cin;
output sum, cout;

wire cout = (a & b) | (cin & (a | b));
wire sum  = a ^ b ^ cin;

endmodule
```

(c) ff.v:

```
module ff (clk, rst, d, q);

input  clk, rst, d;
output  q;
reg  q;

always @(posedge clk)
        begin
                if(rst)
                                q = 0;
                else
                                q = d;
        end
endmodule
```

(d) mux.v:

```
module mux ( a, b, sel, out);

input a, b, sel;
output out;

wire out = sel ? a : b;

endmodule
```

Appendix B

To understand the timing report, first ***right-click*** on your mouse button when you are on the schematic window. Select "worst path" to highlight the most timing critical path as shown in Figure 3 and compare this with your timing report.

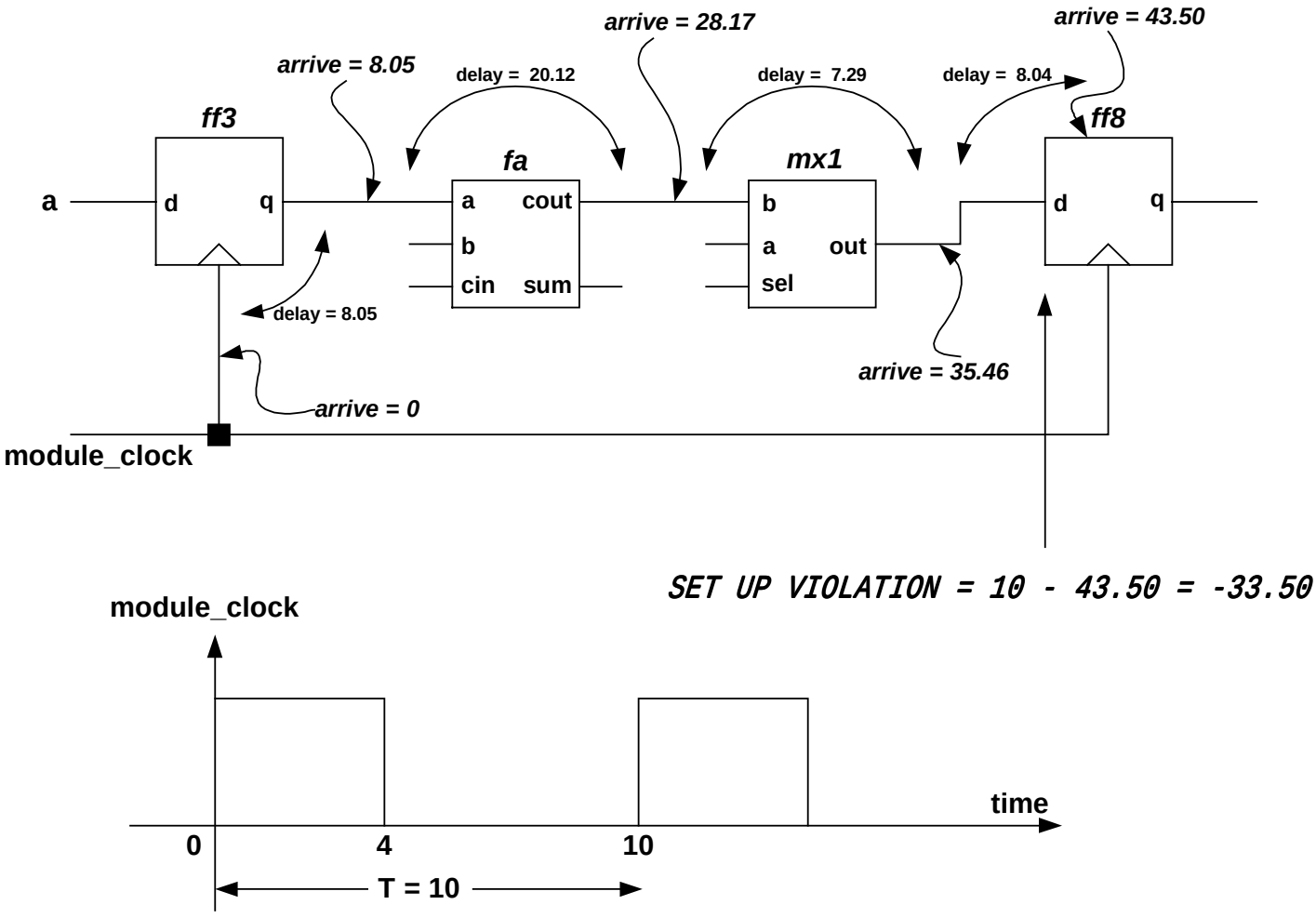


Figure 3

Now it is time to open the **timing.rpt** file under "synthesis/report" directory and examine its contents.

timing.rpt file:

Report	report_timing
Options	> report/timing.rpt
Date	20021014.121316
Tool	ac_shell
Release	v4.0-s008
Version	Apr 20 2001 04:20:50
Module	my_design
Timing	LATE
Slew Propagation	WORST
Operating Condition	NOM
PVT Mode	max
Tree Type	balanced
Process	1.00
Voltage	3.30
Temperature	25.00
time unit	1.00 ns
capacitance unit	1.00 pF
resistance unit	1.00 kOhm

Path 1: VIOLATED Setup Check with Pin ff8/q_reg/CLK

Endpoint: ff8/q_reg/D (v) checked with leading edge of 'ideal_clock'
Beginpoint: ff3/q_reg/Q (v) triggered by leading edge of 'ideal_clock'
Other End Arrival Time 0.00
- ***Setup 0.00***
+ Phase Shift 10.00
= ***Required Time 10.00***
- ***Arrival Time 43.50***
= ***Slack Time -33.50***

Instance	Arc	Cell	Delay	Arrival Time	Required Time
	module_clock ^			0.00	-33.50
ff3	clk ^	ff		0.00	-33.50
ff3/q_reg	CLK ^ -> Q v	ATL_MACRO_FF	8.05	8.05	-25.44
ff3	q v	ff		8.05	-25.44
fa	a v	full_adder		8.05	-25.44
fa/i_87	I0 v -> O0 v	ATL_OR	7.04	15.10	-18.40
fa/i_88	I1 v -> O0 v	ATL_AND	6.54	21.64	-11.86
fa/i_89	I1 v -> O0 v	ATL_OR	6.54	28.17	-5.32
fa	cout v	full_adder		28.17	-5.32
mx1	b v	mux		28.17	-5.32
mx1/i_80	I0 v -> O0 v	ATL_MUX_21	7.29	35.46	1.96
mx1	out v	mux		35.46	1.96
ff8	d v	ff		35.46	1.96
ff8/i_67	I0 v -> O0 v	ATL_MUX_21	8.04	43.50	10.00
ff8/q_reg	D v	ATL_MACRO_FF	0.00	43.50	10.00