

Techniques for Efficient Design

Dr. Paul D. Franzon

Outline

1. More on how to maximize Design Efficiency
2. Advanced Design Techniques

References

1. Smith & Franzon, Chapter 10
2. H. Bhatnagar, “Advanced ASIC Chip Synthesis Using Synopsys Design Compiler, Physical Compiler, and PrimeTime”

Outline

1. More on how to maximize Design Efficiency
 - High level structure
 - Synthesis techniques
 - Low level structure
 - Synthesis techniques
2. Advanced Design Techniques
 - Managing Multiple Clocks

Structure

High Level Structure

Maximizing Performance/ Area:

- Maximize memory bandwidth
- Maximize logic utilization
 - ◆ Is every block used every clock cycle?
 - ◆ If certain blocks have low utilization, can they be merged with other blocks and time-shared?
 - E.g. Share one multiplier, rather than build several (if design is not multiply limited)
 - Look for other examples of resource sharing at block and logic level
 - ◆ If you have a pipeline, how many “bubbles” (empty slots) are there in there typically
 - Might want to reduce combine pipeline stages into fewer stages

Structure

Mid Level Structure

- (See timing notes for detail)
- Pipelining
 - ◆ Unroll the loop
 - ◆ Insert serial registers
- Block level parallelism
- Parallelize a serial critical path
- Retiming

Structure

Low Level Structure

- Logic level parallelism (see timing notes)
- Avoid priority logic
 - ◆ Mutually exclusive cases in case statement
 - ◆ Use if ..; if ..; instead of in .. then .. else
- Adding Structure
 - ◆ e.g Arithmetic Shifter
 - Simple code:
assign shifta = {{31{a[31]},a} >> SC;
 - Vs.
assign d0 = SC[0] ? {a[31],a} >> 1 : a;
assign d1=SC[1] ? {{2{a[31]}},d0} >> 2 : d0;
Etc.
 - Detailed mux-based structure is 20% faster and is Smaller

Lower Level Strategies (cont'd)

Reducing module size by breaking module into smaller parts

- E.g. replacing a 32-bit decoder with 4 8-bit decoders
- Now have logic at both leaf level and higher up

Exploits fact that Synthesis does a better job with smaller modules

More Efficient Synthesis

Make sure the **constraints** are correctly set:

set_input_delay
set_output_delay
set_driving_cell
set_load
create_clock

- Initially, the first four of these are estimated. Later those estimates can be refined once the actual fan out, wire loading, etc. is known.

If the operating temperature range is known, and is narrower than the normal civilian or milspec range, then use the tighter range

- Gives reduced spread between setup and hold and might enable a faster and/or smaller design

More Efficient Synthesis (cont'd)

Nets that don't go through a clocked element will need a max delay constraint. (This is usually results from poor partitioning though)

- e.g. **max_delay 8ns -from in1 -to out1**

Some inputs never change (e.g. configuration pins) and some outputs are asynchronous - use **set_false_path** on these.

More Efficient Synthesis (cont'd)

When you have multiple clocks, Synopsys needs to know what signals are going between the different clocks.

e.g.

```
create_clock period 20 clock20 waveform {0 10}  
create_clock period 10 clock10 waveform {0 5}  
set_multicycle_path 2 -setup -from ff1/cp -to ff2/D  
// ff1 has a 20 MHz clock while ff1 has a 10 MHz clock
```

Timing Optimization Within Synopsys

Scenario 1: You are trying to work out the fastest possible clock speed:

- Start with a clock speed that should be achievable but only just so and do a compile
- Find out the results with
report_constraints -all_violators -verbose
report_timing -max_paths 5 // report 5 worst case timings
- Tighten the clock with **create_clock** and execute a
compile -incremental
- Never start with no clock specification or one that is way too tight or loose -- the initial optimization within Synopsys is significant in determining the final design
- Iterate on the **create_clock/compile/report loop**, saving the .db file whenever you get the best result
 - ◆ If only a few paths are creating setup violations, then you have a good chance of getting a better design.

Timing Optimization Within Synopsys (cont'd)

Scenario 2. You are having difficulty meeting required or desired timing

- After the first compile use

report_constraints -all_violators -verbose

report_timing -max_paths 5 // report 5 worst case timings

to analyze your results

- Normally Synopsys concentrates on one critical path at a time, improving it until it is no longer critical, and then moving to another. If instead you tell it to concentrate on several paths simultaneously, it might produce a better results:

group_path -name critical -critical_range 5 -to clock20

// all paths within 5ns of the most critical path are

// given a higher weight of 1.0

- If you are close (usually 2ns) then throw the computer at it:

compile incremental -map_effort high

- Re-assess your HDL code and design partitioning

Optimizing For Area

HDL Coding Style Techniques:

- **Automatic resource sharing** e.g.

```
always@(a or b or c or d or i)
begin
  z = 0; y = 0;
  if (i) z= a + b + c;
  else y = a + b + d;
end
```

- ◆ Synopsys will build 2 adders and a mux (on c/d), rather than 4 adders and bigger output muxes as long as resource sharing is turned on (the default) and the sharable code is in the **same procedural block**.
 - If the performance constraint is too tight, you might get 4 adders
- ◆ Not all resources can be shared ... see `HDL compiler for Verilog manual' if sharing does not seem to be working.

Optimizing For Area (cont'd)

Synthesis Strategies:

- After synthesis, execute **check_design** to make sure there is no unused logic.
- Use **report_resources** and refer to HDL to see if all possible resource sharing has taken place.
- **set_structure -boolean true** , before a compile might lead to a smaller design by turning on certain boolean minimizations.

For Area or Speed:

- Design by hand (using GTECH cell library directly) and place a **set_dont_touch** on it.
- This is rare and for advanced designers

Speed-Area Tradeoff

Setting and achieving aggressive speed targets can lead to a substantial increase in area

- Synthesis flattens logic and replicates it to achieve speed target
- Look at area and speed when conducting optimization experiments

FPGA Example:

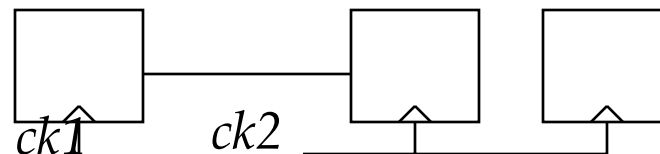
- Baseline: 65.6 MHz → 1386 LUTs, 353 registers
- Speedup: 97.6 MHz → 1538 LUTs, 358 registers

Managing Multiple Clocks

Examples:

- Different modules connected to different clocks (should relate by function of two)
- Three alternatives:
 1. Use synthesis commands to specify delay from one-flip-flop to another
 - See `set_multicycle_path` in Synopsys manual

2. Use double flopping in second clock domain



3. Use a FIFO to transfer data between the clock domains (best technique)
 - Write with input clock
 - Read with output clock
 - Have an interlock to make sure there is a delay between read and write on any one register in FIFO

Review Points

How is performance measured?

How is design efficiency measured?

What are some techniques useful at high levels?

At mid-levels?

At low levels?