Hierarchy and Partitioning

Dr. Paul D. Franzon

Objectives
1. How to specify Hierarchy
2. Design Partitioning – what is a good hierarchy

Motivation
• Good hierarchy greatly simplifies and speeds up synthesis

References
1. Smith and Franzon, Chapter 10, Sections 10.1, 10.2
**Specifying Hierarchy in Verilog**

```verilog
module top (clock, data_in, ..., data_out);
    input clock;
    input [7:0] data_in;
    output [7:0] data_out;
    // outputs of declared modules type wire or tri
    chiplet1 u1 (.clock(clock), .Din(data_in), .Dout(data_out));
    chiplet u2 (.clock ...);
endmodule
```
module chiplet1 (clock, Din, Dout);
  input  clock;
  input [7:0] Din;
  output [7:0] Dout;
  wire [7:0] Dout;
  wire control;

  dataUnit u1 (.clock (clock), .DatIn(Din),
                .control(ConIn), .DatOut(Dout));
  controller u2 (.clock(clock),
                 .control(ConOut), ... )

endmodule
Hierarchy (cont’d)

- Logic ONLY in leaf modules
- Signal Name (has to be type wire or tri)
- Port name inside module
- Forbidden (No glue logic above leaf modules)

Module name
Instance Name

- top
- chiplet1
- controller
- clock
- data_in
- Din
- clock
- ConOut
- ConIn
- clock
- DatOut
- DatIn
- u1
- dataUnit
- u2
- control

© 2013 Dr. Paul D. Franzon, www.ece.ncsu.edu/erl/faculty/paulf.html
Digital ASIC design

Partitioning a Design

ie. Deciding what to put in each module.

General Rule:
Make the synthesized units reasonably small/medium sized while keeping them as sensible synthesis targets.

Why?
- Synthesis is performed serially on modules or module groups
- Synthesis run time $\propto e^{gate\text{-}count}$
- Hence two 1,000 gate modules synthesize faster than one 2,000 gate module (if highly interconnected internally)
Sensible Synthesized Units

Synthesized unit = module or module sub-hierarchy that is synthesized as single unit

Sensible constraints:

- Critical path contained within synthesized unit
- Every path from input to output must pass through a register
- Sharable resources within synthesized unit
  - Must be within same procedural block for automatic resource sharing
- One synthesis strategy only
  - E.g. Separate FSM, as has a different synthesis strategy
- One clock if at all possible
- Registered outputs if it fits in with timing plan
  - Important to register outputs if they are connected to someone else’s design
- Add internal structure where “good structures” can be human specified
- All logic at leaf cell modules only
  - i.e. No “glue” logic
- HUMAN READABLE AND UNDERSTANDABLE
Sharable Resources

Example:

```verilog
always@(*)
  if (A) B = C+D; else B = C+E;

SHOULD build one adder and one mux, rather than two adders and one mux.

It won’t if coded as

assign B1 = C+D;
assign B2 = C+E;
assign B = A ? B1 : B2;
```
Simplified Partitioning Example

Notes:
- circles = combinational logic
- bar instanced twice as U2 and U3
Problems with this Partitioning

Problems/Issues:

U1: \textit{Note, could not register O/P due to feedback logic.}

U2: \textit{U2 and U3 are different instances of the same module. U2 has to be faster due to connection to O/P logic in U1.}

U3: \textit{No problem or issue.}

U4: \textit{No FFs, so unconstrained timing paths from input to output.}
Possible Fixes

Problems/Issues:

U1

U2:

U3:

U4:

U5:

U2 and U3 have to be synthesized separately.

Combine by rewriting or in synthesis
Digital ASIC design

Synthesis Script To Address Problems/Issues

Write top level Verilog module (ignoring details of inputs and outputs):
module top ();
...
endmodule;

Synthesis Script Extract:
(instead of current compile):

......
// on worst_case cells/conditions:
characterize -constraints {U1}
current_design foo compile
current_design top
group {U4 U5} -design_name pets -cell_name U10
characterize -constraints {U10}
current_design pets compile
current_design top

Characterize calculates input and output delays due to connected logic. Determines input_delay and driving_cell

Creates new module “pets”
Script (cont’d)

uniquify -cell U3 -new_name bar2
characterize -constraints {U2}
current_design bar
compile
current_design top
characterize -constraints {U3}
current_design bar2
compile
current_design top
report_timing

Creates temporary module name for U3 so it can be synthesized separately from U2

If report timing specifies a critical path that spans multiple modules, then you should revisit partitioning or group those together and resynthesize the grouped module.
Questions on Script

Is the area of the logic in the timing path from U1 to U2 optimal?

Not necessarily, as the sub-paths were separately optimized, and U1 was synthesized with U2 logic unoptimized. How fixed?

Could run characterize & compile incremental on U1 again to improve.

Why should every path in a synthesized unit contain a register?

Otherwise the timing is unconstrained.

Why should outputs that interface with other designers be registers, if possible?

So the other designer can just assume a FF for set_input_delay and set_driving_cell in script.
**Partitioning (cont’d)**

If your hierarchy is such that the leaf cell modules are the desired synthesis units, and there is no need to optimize logic across module boundaries, then just use:

```plaintext
current_design top
compile
```

- This automatically synthesizes the leaf cell modules
- **Note**, `current_design` **is the most recent module read unless you tell Synopsys otherwise**

You should have **NO GLUE LOGIC** between synthesized units

- Otherwise you have to expand the size of the synthesized unit to include that logic, or (less desirably) use group and flatten to create a “super module”
Motion Estimator

Hierarchy:

- Not synthesized
- Synthesized
Example – Motion Estimator

Then run the script:

```python
current_design = top_without_mem
compile
```

This will:

- Compile PE with automatic characterize-constraints on IO
- Compile Comparator with automatic characterize-constraints
- Compile Controller with automatic characterize constraints
- Assemble hierarchical netlist
Exercise

What is wrong with this partitioning?

current_design = top
ungroup -all -flatten
compile

Logic will be suboptimal. Either combine modules or split here.
Exercise – glue logic

Consider this code extract

module foo ( ... )
  bar u1 (clock, fred, wilma, barney);
  snafu u2 (clock, george, rosey, jane);
  assign bambam = wilma & george;
endmodule

What is wrong with this? Glue logic above the leaf modules
How is it fixed? current_design = foo
               ungroup -all -flatten
               compile

Or redesign so and gate in a module
Summary - Partitioning

Remember:

Partition the design into the modestly sized modules that
- Entirely contain the critical paths
- Have flip-flops in all paths from input to output
- Have FFs for all outputs (as much as practical)
- Contain sharable logic
- All logic in leaf modules
- Make sense from a design and human readability perspective