GONZAGA UNIVERSITY

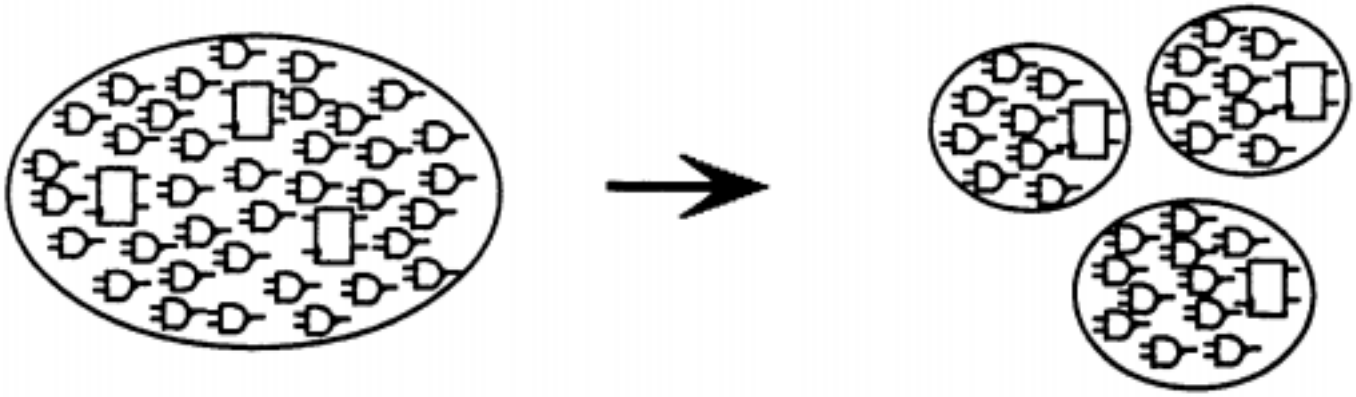
SYNOPSYS®

Synthesis: Design Partitioning

GONZAGA UNIVERSITY

Partitioning

- Partitioning is dividing a design into smaller parts



GONZAGA UNIVERSITY

Why partitioning ?

- Produce the best synthesis results (i.e., improve optimization)
- Speed up optimization run times
- Simplify the synthesis process

GONZAGA UNIVERSITY

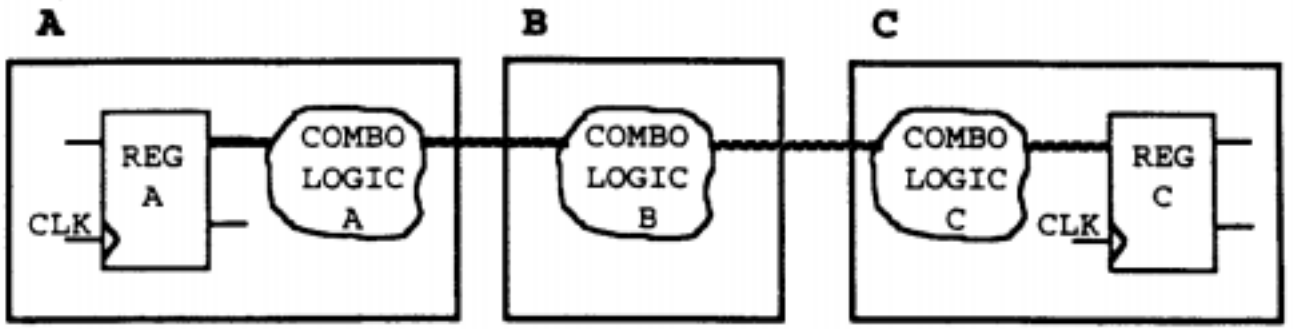
Partitioning Rules

1. Partition for Design Reuse
2. Keep related combinational logic together
3. Register the outputs
4. Partition by design goal
5. Keep sharable resources together
6. Keep user defined resources with the logic they drive
7. Isolate special functions such as pad logic, clocks, boundary scans, and asynchronous logic

Rule 2:
Keep related combinational logic together

GONZAGA UNIVERSITY

Bad Example:



Bad Example

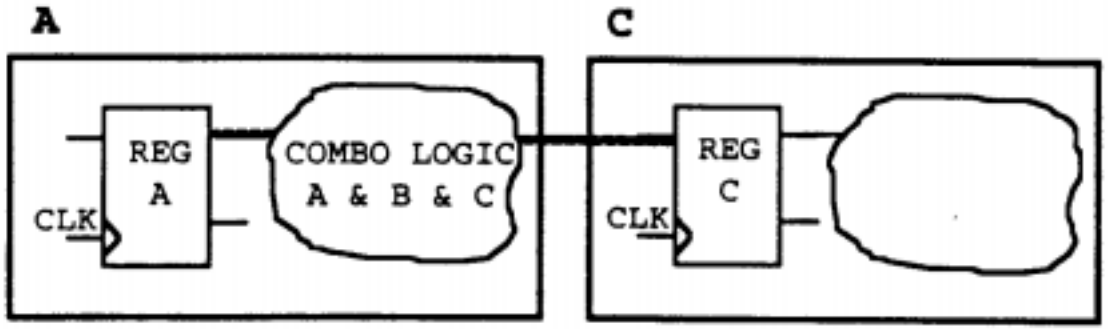
The path from Reg A to Reg C is divided between three different blocks.

- Optimization is limited because hierarchical boundaries prevent sharing of common terms.

Rule 2:
Keep related combinational logic together

GONZAGA UNIVERSITY

Better Example:



Better Example

Related combinational logic is grouped into one block; thus, all related combinational logic is at the same level of hierarchy.

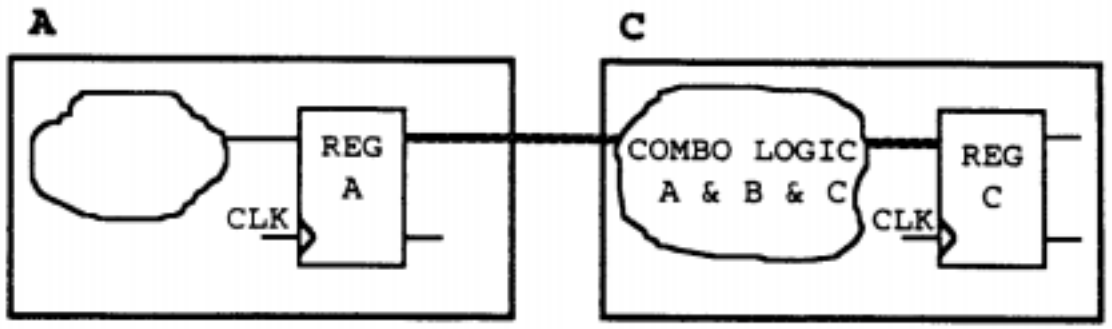
- *Combinational* optimization techniques can now be fully exploited.

Rule 2:

Keep related combinational logic together

GONZAGA UNIVERSITY

Best Example:



Best Example

Related combinational logic is grouped into the same block that contains the destination flip-flop for the combinational logic path.

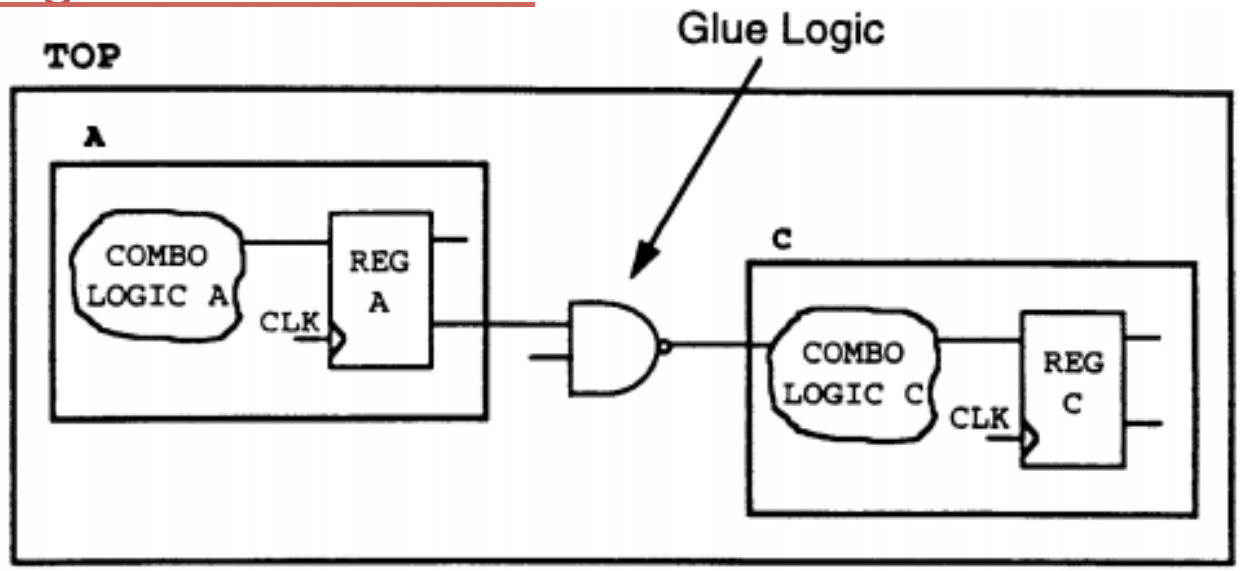
- Allows for improved *sequential mapping* during optimization (no hierarchical boundaries between combinational and sequential logic).
- Simplifies the description of the timing interface.

Rule 2:

Keep related combinational logic together

GONZAGA UNIVERSITY

No glue logic between blocks



Bad Example

A NAND gate was added to the TOP level block description, to "bridge" the two instantiated lower-level blocks.

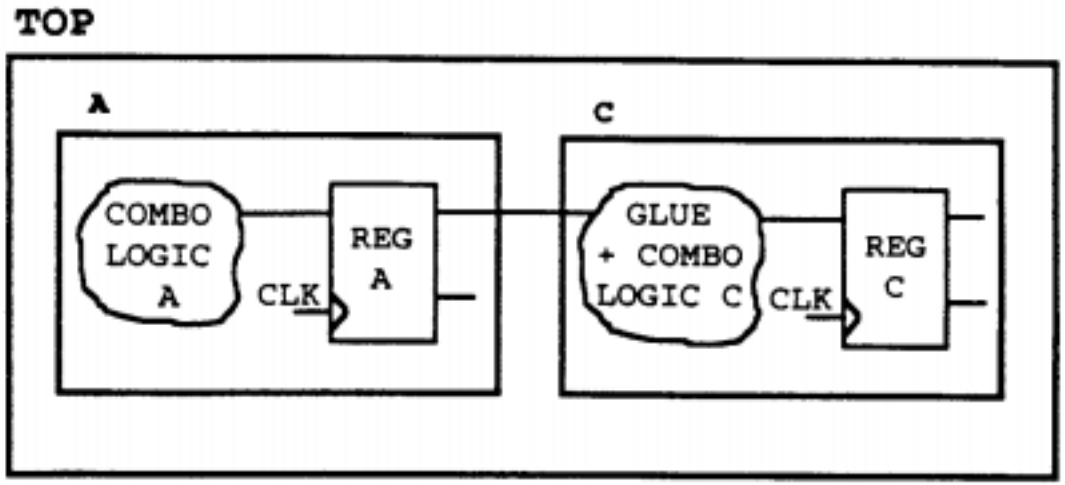
- Optimization is limited because the glue logic cannot be "passed" to the lower-level blocks where it might be "absorbed."

Rule 2:

Keep related combinational logic together

GONZAGA UNIVERSITY

No glue logic between blocks

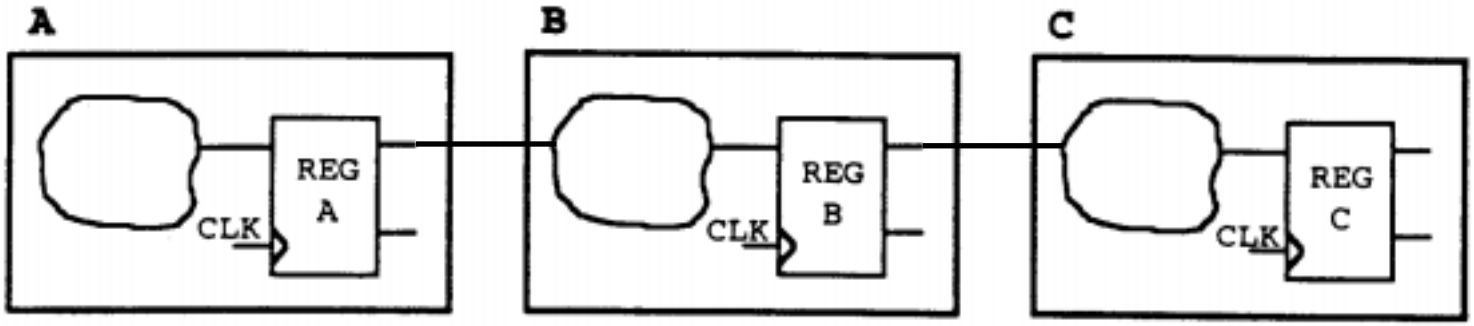


Good Example

Merge glue logic into the related combinational logic description of the lower-level architectural statements.

- The merged glue logic can now be optimized away.

Rule 3:
Register all outputs



Good Example

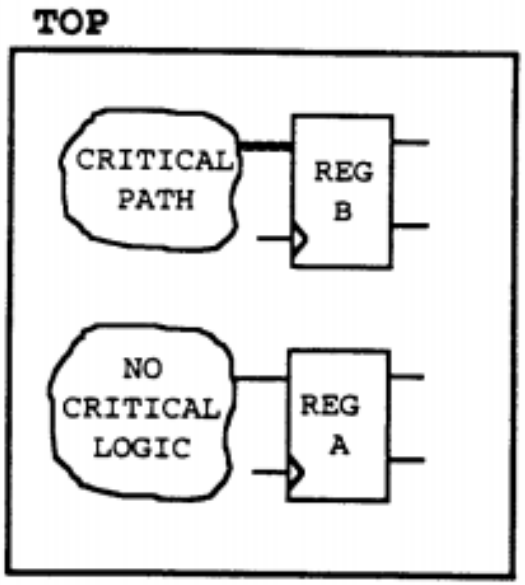
For each block within a design, register the signals exiting the block.

- Simplifies the synthesis design environment: all inputs of each block arrive within the same relative delay.

Rule 4:
Partition by Design Goal

GONZAGA UNIVERSITY

Bad example



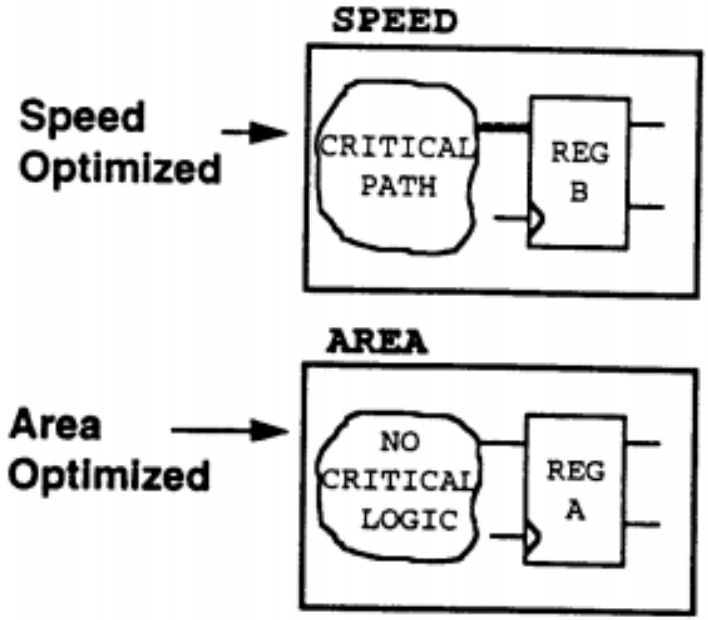
Bad Example

REG B is in the critical path, but REG A is not.

- Optimization is limited because the designer cannot isolate parts of a block and optimize them solely for area or for speed.
- The entire block is optimized using the *same* optimization technique(s).

Rule 4:
Partition by Design Goal

Good example



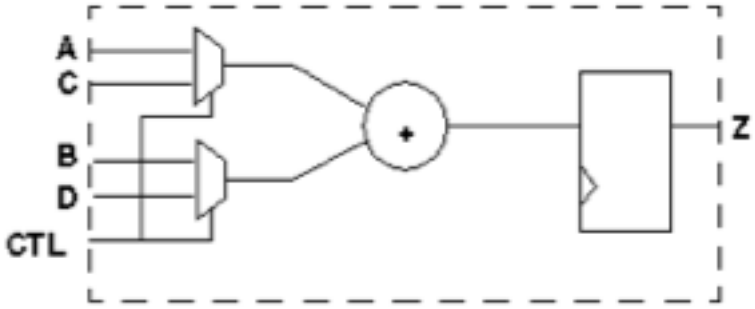
Good Examples

Use different entities or modules to partition the design into separate blocks.

- Designer can now perform appropriate optimization technique(s) on each block.

Rule 5:
Keep sharable resources together

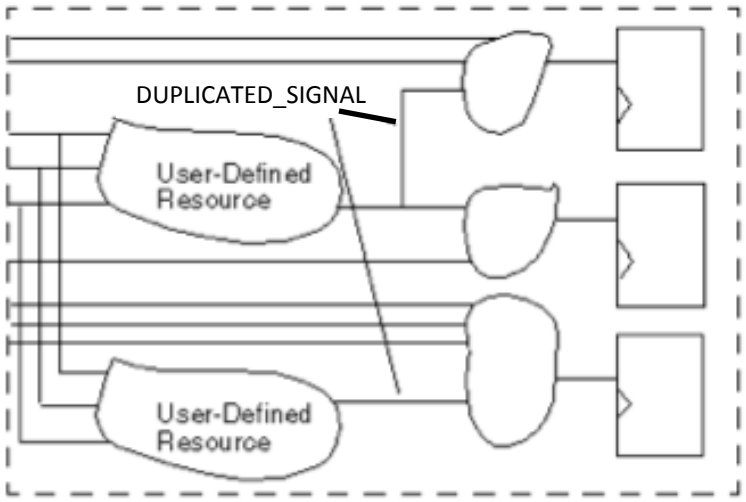
- It is possible to share large resources, such as adders or multipliers, but resource sharing can occur only if the resources belong to the same VHDL process (or Verilog always block).



Rule 6: Keeping User-Defined Resources with the Logic they Drive

- User-defined resources are functions, procedures, or macro cells.
- Keeping these resources with the logic they drive, gives you the flexibility to split the load by manually inserting multiple instantiations of a user-defined resource if timing goals cannot be achieved with a single instantiation.

Duplicating User-Defined Resources



Rule 7: Isolating Special Functions

- Isolate special functions (such as I/O pads, clock generation circuitry, boundary-scan logic, and asynchronous logic) from the core logic.