

# Synthesis Tool Strategies

Set up your designs in Synplify for performance improvement and area savings.

by Steve Pereira  
Technical Marketing  
Synplicity, Inc  
stevep@synplicity.com

You can benefit greatly from a proper synthesis strategy. Such strategies include knowing the final target architecture, knowing what coding problems could arise, and understanding what performance the periphery will require. You should also understand how to use IP. Are models available? Is cost an issue? Your initial setup can greatly affect productivity and help you achieve quicker peripheral timing closure.

In this article, I'll describe a known good strategy while using Synplify Pro tools.

## Best Practices

Setting up your design correctly can result in huge performance increases or reductions in area. The following checklist describes the best practices to use when setting up your design.

1. Include any CoreGen EDIFs or timing models for black boxes. If you use black-box IP in the design, ensure that all EDIF, NGC netlists, or timing models are provided. It is essential that the Synplify Pro tool knows the timing

requirements into and out of the box so that surrounding logic can be altered to reduce or remove criticality.

If the design has an ngc file, use the ngc2edn (provided in the Xilinx® ISE™ /bin/ directory) converter utility to produce an edn file for synthesis.

2. Ensure that the device is correct and size the design. Ensuring that the wireload models are correct for synthesis can greatly affect the resulting logic. The selection of wireload models is simply a matter of selecting the correct device and speed grade. If synthesis is performed on a different device to the final implementation, sub-optimal results are quite likely.

Selecting the correct device will also provide Synplify Pro with the accurate number of resources. One example of the importance of this is with block-select RAM mapping. Synplify orders all of the RAMs from biggest to smallest, and then starts to map the largest RAMs until there are no block RAMs left. The rest are placed into distributed RAM. If the wrong device is selected, sub-optimal mapping will occur.

Sizing the design also has a significant impact. For example, if the design uses 80% of the device, the wireload models are correct for the design. If the design consumes only a small percentage of the total resources (for example, synthesizing just a part of the design for verification), the wireload models will be inaccurate. To resolve this problem, you can assign the logic to an area group. Please see the Synplify Pro documentation for instructions on how to do this.

3. Provide accurate clock constraints. Under- or over-constraining results in reduced performance. Do not over-constrain by more than 15%. For maximum performance, ensure that there is 10% negative slack on the critical clock. This ensures that critical paths are squeezed. The Fmax field on the front panel is fine for a quick run, but do not use it if you need maximum performance. Put unrelated clocks in separate clock groups in the Synplify Pro .sdc file. If your clocks are in the same group, the Synplify Pro tool works out the worst-case setup time for the clock-to-clock paths.

Figure 1 shows a timing diagram for two clocks that are in the same clock group. Synplify rolls the clocks forward until they match up again. The tool then calculates the minimum setup time between the clocks, in this case 10 ns.

If the clocks are unrelated, there may be several hundred clock periods before the clocks match up again. This may result in the worst-case setup time being very small (100 ps). You can check the setup time in the clock relationships table in the log file. If the setup time is too short, it is best to re-constrain the clocks so that they are more related.

4. Specify timing exceptions. Provide all timing exceptions, such as false and multicycle paths, to the Synplify Pro tool. With this information, the tool can ignore these paths and concentrate on the real critical paths.
5. Constrain I/Os. If the design has I/O timing constraints, it is likely that the critical path is through the I/O block (IOB). The Synplify Pro tool sees these paths as the most critical and tries to optimize them. Usually, I/O paths physically cannot be optimized any further; as they are the most critical, the Synplify Pro tool stops optimizing the rest of the design.

A new switch has been added to the Synplify Pro 7.3 release called “use clock period for unconstrained I/O.” When enabled, the tool does not include any unconstrained I/O paths in timing optimizations.

6. Keep code generic. Keeping code generic and not locked down to a particular architecture can aid design, reuse, and portability. For example, with the DSP48 block in the Virtex™-4 architecture, you can specify generic code to implement a DSP function. The tool will map to that component(s) when possible and reduce uncertainty regarding how the function was mapped. If DSP blocks are generated, timing is better known and can speed up debug – and time to market. You can specify the register configuration and code in the opcode to drive the DSP block configuration, all with generic code.

If for some reason you wish to use a different device family, such as moving from Virtex-4 FPGAs to Virtex-II Pro FPGAs, the porting process itself should be seamless, but you may feel uncertain about implementation and logic levels.

### Good Switch Settings

- Retiming and pipelining. Enabling retiming and pipelining options can

improve your design performance by as much as 50%. Retiming attributes such as `syn_allow_retiming` let you refine your constraints by surgically applying retiming to a single register. Synplicity recommends that you enable both of these switches.

- Resource sharing. Always turn this switch on. The behavior of this switch was changed in Synplify 8.0. With the switch on, timing-driven resource sharing occurs. Non-critical logic will have resource sharing, but critical logic will not share resources (for performance reasons).
- FSM Compiler extracts and optimizes FSMs based on the number of states:
  - 2 to 4: sequential
  - 5 to 40: one-hot
  - More than 40: gray

Synplicity also recommends enabling the FSM Compiler and setting default enumerated encoding to the “default” value for VHDL designs.

- FSM Explorer timing-driven state encoding. The Synplify Pro tool automatically selects the best encoding for the specified timing. This switch is design-dependent. If the critical path starts or ends at a state machine, turn the switch on.

### Conclusion

Because synthesis tools are operating at higher levels of abstraction, synthesis optimizations can have a dramatic impact on design performance. After parsing through the HDL behavioral source code and extracting known functions (arithmetic functions, multiplexers, and memories), synthesis tools then map these functions on the target architecture features.

The tools trade off area and performance based on design constraints and tools settings; these influence the use of optimizations such as replication, merging, re-timing, and pipelining. As a result, the right tools settings in synthesis can greatly increase productivity and time to market. 🌈

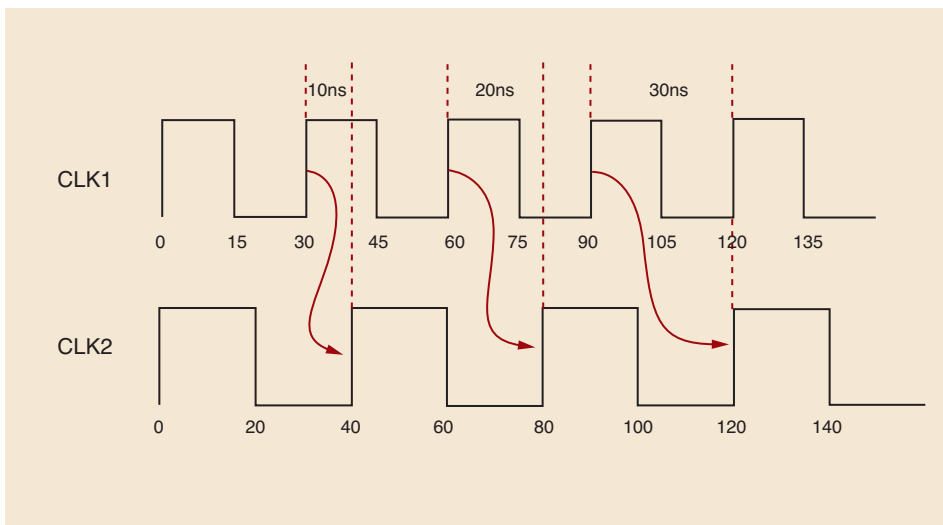


Figure 1 – Clock rolling for clocks in the same domain