Synplify Extends Timing Constraint

by Jim Tatsukawa, Partner Programs Manager, Synplicity Inc., jimt@synplicity.com

Synplicity has expanded its Synthesis Constraint Optimization Environment (SCOPE) to allow you to characterize the timing of macrofunctions not synthesized in Synplify. These new constraints integrate SCOPE more tightly for mixed mode design entry than any other FPGA constraint solution. SCOPE provides total control of your synthesis results by using an innovative multi-level constraint approach. The addition of the timing constraints for mixed mode entry complements the existing timing constraint.

New Synplify 3.0C Timing Constraints:

- Black Box Propagation Delays
- Black Box Setup Delays
- Black Box Clock to Output Delays

Existing Timing Constraint Features:

- Clock Frequency
- Input Delays
- Output Delays
- Delays To Registers
- Delays From Registers
- Multicycle Paths
- Improve Timing Constraint
- Route Timing Constraint

> *"Synplify allows you to define timing constraints that automatically control synthesis to meet the system requirements."*

When synthesizing a design, logic can often be made faster at the expense of more logic. The most direct method of evaluating these "area vs. performance" trade-offs is by analyzing the performance of the synthesized logic. Synplify allows you to define timing constraints that automatically control synthesis to meet the system requirements.

Consider the Xilinx cores and macrofunctions that have been carefully structured to fit into your design. Most synthesis tools do not allow you to specify the timing characteristics of these functions when incorporated into your design. Therefore, the synthesis tool does not understand whether inputs and outputs are registered or combinatorial. Additionally, the delays inside these block are unknown, leading to paths that become over- or under-constrained.

If a design is under-constrained, the logic will not be synthesized to map to the optimal amount of logic and will perform slower than required. If a design is over-constrained, synthesis compromises on design performance and area to achieve the goals. The over-constrained design may either be larger than required, or may be slower for the overall design.

Defining Hierarchy with the Black_Box Attribute

Synplify supports mixed mode design entry by instantiating components and attaching the "black_box" timing attribute. The black_box attribute allows the integration of schematics, LogiBlox, COREgen, Xilinx Core Solutions, as well as any other design that is not to be synthesized in VHDL or Verilog.

To use the black_box attribute create a stub for the macrofunction (logic content will be ignored). The stub must declare the ports and the port directions. By placing the "black_box" synthesis directive just before the semicolon in the module declaration, Synplify will ignore the internal logic. The body of the code then instantiates the component. The netlist that Synplify generates will be combined with the other design files when compiled in the Alliance Series tools.

Control For Mixed Mode Entry

Timing Constraints For Black_Box Modules

After the black_box module has been specified, Synplify allows the full timing characterization of the black_box module through the use of three types of timing attributes that are attached to the black_box definition. The following summaries describe how to effectively use these timing constraints to fully characterize your design.

- **Combinatorial delays** through the black_box module are defined by the syn_tpd attribute. It specifies the delays from the inputs of the black_box module to the outputs. Delays are given in nanoseconds.

```
Syntax: syn_tpd1="{input or input
 bus}->{output or output
 bus}={propagation delay in ns}"
```

- **Registered inputs** for the black_box module use the syn_tsu attribute to specify the setup time required for the inputs relative to the clock. Synplify will then recognize the black_box module as the destination of register to register propagation delays. Many designs use both rising edge and falling edge clocks. Synplify allows you to specify the rising edge and falling edge clocks. The syn_tsu specifies falling edge clocks by the additional use of the "!" character.

```
Syntax: syn_tsu[0-9]="{input or
 input bus}->{clock input}=
 {propagation delay in ns}"

Syntax: syn_tsu[0-9]="{input or
 input bus}->!{clock input}=
 {propagation delay in ns}"
```

> *"Synplify supports mixed mode design entry by instantiating components and attaching the "black_box" timing attribute."*

- **Registered outputs** for the black_box module use the syn_tco attribute to specify the clock to output delays within the black_box module. The syn_tco additionally specifies the black_box module as the source of register to register propagation delays. The syn_tco attribute supports the rising or falling edge clock specification as in the syn_tsu.

```
Syntax: syn_tco[0-9]="{clock
 input}->{output or output
 bus}={propagation delay in ns}"

Syntax: syn_tco[0-9]="!{clock
 input}->{output or output
 bus}={propagation delay in ns}"
```

◆

Example Design

```
module ram32x4(z, d, addr, we, clk);
/* synthesis black_box
     syn_tpd1="addr[3:0]->z[3:0]=8.0"
     syn_tsu1="addr[3:0]->clk=2.0"
     syn_tsu2="we->clk=3.0"
 */
output [3:0] z;
input [3:0] d;
input [3:0] addr;
input we;
input clk;

endmodule

/*
 * Build a bigger ram
 */
```

```
module ram64x4(z, d, addr, we, clk);
output [3:0] z;
input [3:0] d;
input [4:0] addr;
input we /* xsynthesis
 syn_input_delay=10.0 */;
input clk;

wire [3:0] za, zb;

wire wea = we & ~addr[4];
wire web = we & addr[4];
ram32x4 r1 (za, d, addr[3:0], wea, clk);
ram32x4 r2 (zb, d, addr[3:0], web, clk);
assign z = addr[4] ? zb : za;

endmodule ◆
```