

Synplicity Synplify -- FPGA Synthesis

Wish List

5/18/01

Bill Lenihan
Raytheon Systems Co. (formerly Hughes Aircraft Company)
2000 E. Imperial Hwy., R2-V514
El Segundo, CA 90245
310-334-8324
wlenihan@west.raytheon.com

Unless otherwise noted:
Target technology is Xilinx Virtex and derivatives;
Source code language is Verilog.

req #	Feature
1	When there is more than one verilog modules in the list of files read into the project's source code directory -- users need an easy & reliable way to pick the top level design to be synthesized. Supposedly the file at the bottom of the list is the top level, but I've found that sometimes dragging the file I want to the bottom isn't enough, nor is clicking on the "change result file (edif)" button. The last resort is going into the "Project -- Implementation Options -- Implementation Results -- Result File Name", but this is too cumbersome. Why can't dragging the right file to the bottom be enough to reliably declare the top level to be synthesized?
2	It's easy to pick the target technology with the "change target" button, but when the user instantiates device-specific macros (i.e., clock DLLs or Block RAM in Xilinx Virtex), they have to go to "add source" and traverse up & down their directory structure to grab the right technology file from C:\synplicity\synplify\lib\xilinx. Why can't the synthesis tool just know to look in the right file if it encounters such macros? Why does the user have to read in a remote file that is already "under the hood" of Synplify?
3	It seems that the only way to control the "replicate logic" feature is to set the Fanout Guide in the Implementation Options. Is there another way? Does the user have any control over the "wireload model" if the synthesis timing prediction doesn't match the actual P&R tool timing results? Can the user control the 'replicability' of one part of the design separate from other parts of the design?
4	I want to create a large ROM by inference using a case statement and direct Synplify to implement it using the Block RAM(s) (not LUTs or other CLB logic). i.e., the pragma /* synthesis syn_romstyle = "block_rom" */ would work for Xilinx, as well as Altera. [Note: since Xilinx BRAM is synchronous, the always block surrounding the case statement would fire on the posedge of clock, not the address.] See case # 24086.
5	Can a synthesis pragma equivalent to /* synthesis xc_map="lut" */

req #	Feature
	<p>be made that will work for CPLD macrocells? i.e.,</p> <pre>/* synthesis xc_map="macrocell" */</pre> <p>See case # 23323.</p>
6	<p>We often have HDL-based IP that is created by other groups within our company. Sometimes this IP is targeted to other FPGA Synthesis tools (i.e., Synopsys or Exemplar), including the use of synthesis pragmas/directives in the code. Such code will produce non-optimal (or in some cases useless) results if read into Synplify in the HDL format, since Synopsys or Exemplar pragmas are different than Synplicity.</p> <p>If Synplify could read in EDIF netlists as well as Verilog & VHDL, then the IP owners could synthesize using their tool and deliver their IP as EDIF netlists without their customers having to deal with incompatible synthesis pragmas/directives.</p>
7	<p>I need to create a fast, N-bit-wide bi-directional bus in some of my FPGA designs. By 'fast' I mean not only launching data-out & capturing data-in delays (small prop delay from pad to data registers), but also fast in turning the bus around from read-to-write or write-to-read, preferably in 1 clock cycle (i.e., I need very small prop delay in tri-state control).</p> <p>To meet this requirement, I need the N-bit data-in registers, the N-bit data-out registers, and the N-bit tri-state control registers to all be pushed into the I/O cells of the FPGA. This is easy to do with the N-bit data-in & data-out registers, but invariably the N-bit tri-state control register is optimized down to 1 bit (since all N bits are logically the same) and the register that produces this 1 bit must be in the core logic not the I/O cells. This slows down the turnaround of the bus and forces me to insert wait states in the controller, degrading performance.</p> <p>We need a way to keep an N-bit tri-state control register even if all N-bits are logically the same.</p>
8	<p>Help files / documentation seems incomplete when it comes to synthesis pragmas concerning CPLDs. Example: for Xilinx devices, xc_fast & xc_slow are mentioned as being valid for xc4000 only, but I found that they also work for xc9500 devices, too.</p>
9	<p>In Xilinx Virtex designs, the Xilinx P&R tool takes shift register structures and tries to fit them into the SRL macro (shift registers implemented in 1 16-bit-LUT as opposed to regular FDCE's) if the user wants to implement the shift register in regular FDCE's (i.e., for pipelining purposes to distribute the bits of the register from one side of the chip to the other), he must make sure there is a reset or preset in place and that it is wired somewhere that won't get stripped out (dummy pin on I/O or bit in a bus' memory map). Since this may be hard to understand and pass along as other people inherit / use the design, it would be better to implement in a synthesis pragma. Can Synplicity & Xilinx work to make a clean solution to this?</p>
10	<p>I would like to suggest that the on-line help files should differentiate more clearly that the "block_ram & no_rw_check" are seperate pragmas and don't need to be concatenated</p> <p>is now:</p> <pre>synthesis syn_ramstyle = "registers"</pre>

req #	Feature	
	synthesis syn_ramstyle = "select_ram"	
	synthesis syn_ramstyle = "block_ram"	
	synthesis syn_ramstyle = "block_ram & no_rw_check"	
	should be:	
	x	V
	c	I
	4	R
	0	T
	0	E
	0	X
	synthesis syn_ramstyle = "registers"	x x 'RAM' from registers & combinatorial logic
	synthesis syn_ramstyle = "select_ram"	x x distributed RAM via LUT's (default)
	synthesis syn_ramstyle = "block_ram"	x Block RAM, with read/write address conflict resolution logic
	synthesis syn_ramstyle = "no_rw_check"	x Block RAM, with NO read/write address conflict resolution logic

- 11 I'm hoping to find some kind of synthesis pragma to define "origin points" for multiple submodules (whose innards have already been RLOCed), so that I can pack them close together. My brute force alternative is to create one giant submodule w/ all the appropriate RLOCs, but this doesn't lend itself well to (a) easy understanding by others, (b) modular design, heirarchy, etc.,