Bluespec ASIC Pilot Project

Pilot Project Overview

- Redesigned a major functional component of an existing ASIC in Bluespec
  - 2 stage arbitration block (16 nodes, 18 ports/NODE)
  - 9 Subblocks
  - ~1.55 million gates
- Integrated resulting Verilog into an existing ASIC netlist and design verification environment
- Completed synthesis, test insertion, physical layout, timing analysis on each subblock (hard macro)

Pilot Project Design Details

- 4760 lines of Bluespec VS 65459 lines of Verilog

BLOCK NAME	PILOT TOTAL GATES	ORIGINAL TOTAL GATES
UC	141000	-
UCPS (x16)	897000	856000
MC	228000	271000
FO	18000	26000
QR (x16)	163000	164000
AC	15000	0
MG	90000	42000
QR FANOUT	1000	-
PI FANOUT	1000	-
SUBTOTAL	1550000	1360000

MEMORY NAME	QUANTITY	DEPTH (WORDS)	WIDTH	# OF BITS
mc_mem	1	1024	16	16384
qr_mem	16	354	4	22656
SUBTOTAL	17			39040

Front End Issues

- Front end code had to accommodate some requirements for the backend. (Often these are vendor and design-flow specific.)
  - Design for test (DFT) required dummy SCAN ports (pragma was added)
  - Memory BIST may require connectivity from an instantiated memory core up to I/O ports at the top level of the subblock
  - Hard macro flow required bit-blasted interfaces (pragma was added)
  - To simplify verification/backend, we wanted all flops resetable. (Bluespec Verilog libraries were modified to accommodate this.)
- Bluespec scheduling can present problems
  - The designers made assumptions about the schedule that the compiler would pick. Often, correctness of the resulting design depended on these assumptions.
- Processor Interface register plumbing was entirely hidden

1

Verification/Debug Issues

- Bug turn around time can be a problem. The Bluespec flow has an added compile step.
  find bug->fix in BS code->
  recompile into Verilog->rerun sim
- Typical Verilog model for debugging becomes more difficult
  - A designer will look simultaneously at signals in the simulation waveform and the Verilog code (particularly in large chip/system designs). This becomes more cumbersome when the Verilog is compiler-generated.
  - Concise and descriptive presentation of scheduling information (and any other debugging info) would be very helpful here

Jan 16, 2003

1

Verification/Debug Issues (cont'd)

- Source/Destination naming conventions embedded in all "top-level" interfaces
  - Separate interfaces were defined for each pair of subblocks that talk to each other. These interface names include the 2 subblock names.
- Consistent use of well-defined types is important
  - In the pilot, this initially meant more compile-time errors and longer coding/unit-testing times.
  - During integration and chip-level verification, however, there were significantly fewer bugs as compared to the original project.

Jan 16, 2003

Synthesis/Physical Layout

- To accommodate the physical layout, we needed to register all signals at hard macro (subblock) boundaries
  - We restricted all interfaces to be of type either CGet/Cput or CServer/CClient (all control and data are registered with these types of interfaces).
  - Many of the subblocks had multiple wide interfaces. We ultimately needed to optimize the implementation of these interfaces to minimize flop counts.
  - Ultimately, there is still "extra" overhead incurred when these interfaces need to support full bandwidth data flow
- Multicycle Paths
  - The design required multicycle paths to minimize flop counts
  - These were encapsulated in a bluespec library

Synthesis/Physical Layout (cont'd)

- Embedding names in Verilog
  - Many backend scripts key off of instance names which are embedded in the Verilog code
  - We didn't always have the degree of control we would have liked, but the the backend was able to accommodate this
- Structural VS Logical blocks
  - We like to separate structural blocks (blocks which instantiate sub blocks) from logical blocks (blocks which contain gates/flops).
    - This simplifies synthesis scripting and budgeting between low-level blocks.
    - Compiles of structural blocks is faster (don't need to be synthesizied- can just be written out)
  - It is harder to maintain this distinction with Bluespec. ("Structural" Bluespec blocks often contain some small amount of logic.)

Synthesis/Physical Layout (cont'd)

- Bottom-up synthesis
  - Some subblocks were large enough that they required "bottom-up" synthesis. Synthesis times can blow up with top-down compiles on large blocks.
  - In the pilot, the partitioning was not ideal for this (i.e., the structural/logical distinction is blurred and interfaces were not registered). This is partially because Bluespec promotes drawing module (physical) boundaries at functional boundaries.
- Clock domain crossings (not used in pilot project)
  - We would need to generate separate Verilog modules for each domain
  - We would like standard libraries that handle resync'ing (synchronous fifoes with Grey coded pointers, etc.)

Future Improvement Ideas

- Compiler generated synthesis scripts
- Compiler generated waveform viewer config files
  - These could contain some (all?) of the bluespec types defined. This would be particularly useful for state machines.
- Synthesis/vendor specific optimized code generation
- "Hard-core/soft-shell" partitioning for synthesis
  - This means a "hard core" with all interface signals registered and a "soft shell" which instantiates the hard core and any combinational logic which surrounds it. Hard cores would be the only blocks synthesized; soft shells are always synthesized with the next highest hard core
- Mechanisms for embedding debug code in the generated Verilog (error messaging, etc.)
- Bluespec/RTL formality (equivalence) checking
- Optimizing asynchronous interfaces by making top level assertions (might be able to optimize backpressure mechanisms)