

Synopsys and Apple

Apple Computer Designs Interactive TV Set-Top Box ASIC With Behavioral Compiler

Design engineer David Black joined Apple Computer's Interactive Television group in December 1994, as the lead designer for a set-top box ASIC.

The design goal was to develop a chip (multimedia mixer ASIC) that mixed video and graphics into a single stream of data for output to a television set. Although a prototype of the ASIC existed, the project was behind schedule and needed to be completed as quickly as possible in order to make implementation trials with British Telecom. An engineer with fifteen years of experience in logic design and EDA tools, Black found that the interactive video and proprietary network interface involved in this project represented a new design area for him. Essentially a one-man design team, Black was also responsible for setting up the EDA environment for the project in Apple's Austin, Texas office.

"After a month, I was no longer asking 'how do I code this?'; instead, I could concentrate on the design functionality."

David Black, Design Engineer, Apple Computer

Synopsys

Solution

- Behavioral Compiler™
- DesignWare® Developer
- Design Compiler™
- Test Compiler™

Benefits

- Manage design complexity and reduces design time
- Helps designers understand and improve performance/area
- Easier to specify, understand, debug and reuse. Typically 10x-20x faster simulation
- Delivers high quality of results

In Apple's concept, the set-top box (Pippin) is an interactive service that replaces pay-per-view and VCR tape rentals. Additionally, set-top boxes provide internet access and true interactive consumer directed home shopping. As a result, the design project Black faced was considerably complex. Initially, he estimated that the desired functionality would require doubling the complexity of the prototype. Moreover, since the prototype had been implemented in a Xilinx FPGA in an asynchronous design, Black knew he'd essentially have to start from the ground

up, developing design specifications before coding in a synchronous design approach. The final 54 MHz ASIC consisted of approximately 100k gates (65k logic and 38k memory), with 51 modules and six levels of hierarchy. Ninety percent of the logic was generated from a behavioral specification using Synopsys' Behavioral Compiler."

Based on the project timeline and design constraints, Black selected Synopsys' Behavioral Compiler. "Behavioral Compiler promised to provide a higher level of abstraction, enabling

easier coding and faster simulation," Black explained. "This would improve the design cycle time, allowing me to focus less on lower level details and more on the desired functionality." Since the video processing portion of the design was a natural pipeline, he felt that Behavioral Compiler's automatic pipeline optimization features would be extremely handy. In addition to Behavioral Compiler, Black chose to use Synopsys' DesignWare Developer™, Design Compiler™, Test Compiler™ and DesignWare IP library to make up the EDA environment for the project.

coding

Getting Up to Speed on Behavioral Compiler

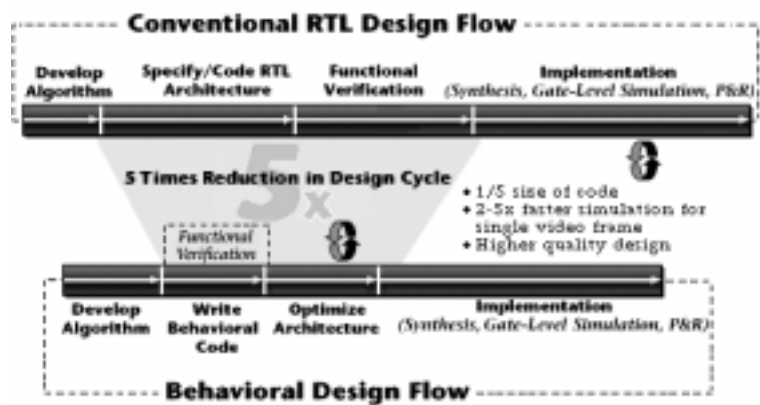
Black attended a one-week training class on Behavioral Compiler for Verilog held locally in Austin. "A nice feature of the course for me was that the main example was a video processing design," he commented. "The course used a substantially-sized example instead of the typical dozen gates so often encountered in classes." After the training class, Black immediately started writing code and running it through the compiler. It took him approximately two to four weeks to feel he had an intuitive understanding of the tool and the behavioral approach to coding. "After a month, I was no longer asking 'how do I code this?'; instead, I could concentrate on the design functionality," he explained.

"I was about five times as productive using Behavioral Compiler than using a conventional RTL approach."

Initially, Black planned to limit behavioral compilation to just the video processing sections of the design. Since this represented a major portion of the code, it more than justified using Behavioral Compiler. After taking the training course and beginning the initial coding, Black discovered that many more modules would benefit from the behavioral coding approach. "Since by design practice, our design was fundamentally synchronous, then anywhere a state machine was present, an opportunity for behavioral code existed," he explained. The result? A reduction in code size that, in turn, reduced the chance of error. "Instead of ending up with ten to twenty pages of RTL code that was error prone and difficult to follow," Black said, "I had a small amount of compact code that I could just look at and understand what was going on." Black estimates that his code using Behavioral Compiler is approximately five times smaller than it would have been using conventional RTL.

Exploring Design Architectures

Since this project involved a quick-as-possible schedule, Black focused on fast implementation and minimized his architectural investigations. He did, however, make some architectural trade-offs in pipelining in the video processing portion of the design. "One of Behavioral Compiler's strengths is its ability to automatically pipeline code, which allowed me to explore trade-offs in latency," Black explained. His initial interval was fixed by the pixel clock rate, so latency was the only variable he could play with. In the end, since area was a concern in the design, he used the largest latency available to maximize resource sharing resulting in an area efficient architecture.



Achieving Significant Time Savings with Behavioral Compiler

In designing the set-top box ASIC, Black spent a considerable amount of time up front developing the specifications. After the design was fully documented, he estimates that coding only took him about one month using Behavioral Compiler. "Coding in RTL could have easily doubled the time I spent coding—and it would have been quite a bit more difficult to verify," Black commented.

The tool's higher level of coding abstraction allowed Black to focus on design functionality and ignore the more mundane aspects of compilation. With

Behavioral Compiler, he was able to make his timing goals on the first compile. This considerably shortened the design cycle and eliminated timing as an issue. "For me, productivity is being able to concentrate on the design and functionality, rather than on the tools," Black explained. "In this sense, I was about five times as productive using Behavioral Compiler than using a conventional RTL approach."

Cutting Simulation Time in Half

In addition, the tool provided significant time savings in the simulation phase. Midway through the project, Apple hired an external contractor to create a board-level test environment

to verify various blocks of the design from the pin-level out. By chance, the contractor had recently finished an RTL video simulation project. In testing the Apple code, the contractor found that Behavioral Compiler cut the simulation time at least in half. Using conventional RTL, simulation took about ten hours to do a single screen (equivalent to one 1/30-second frame). With Behavioral Compiler, that time was reduced to just five hours. "And that was without doing any tricks to improve simulation," Black added.

"One of Behavioral Compiler's strengths is its ability to automatically pipeline code, which allowed me to explore trade-offs in latency."

Resulting in Flexible, High Quality Code

Black's final code to tape out was solid, high quality code that the ASIC vendor could easily fabricate. Perhaps more importantly, the high level of abstraction available with Behavioral Compiler resulted in code that can be easily changed. "The set-top box is still in trials, and its debut in the marketplace is still a few years away," Black explained. "As a result, there are still a lot of open questions regarding how the data will be transmitted, for example. Over coax, ATM, a satellite downlink—all of these are possibilities." Currently, Apple adapts the set-top box to whatever transmission technology is being used in the trials. With Behavioral Compiler, Black can easily alter the code to meet changing specifications—without having to perform a significant redesign.

Looking to the Future

Black plans to use Behavioral Compiler on his next project. "It really is an excellent product," he commented. "I'd hate to be forced to go back to RTL only as a coding approach." Black believes that designers of any experience level can use Behavioral Compiler and that novice engineers straight out of college might find it easiest to adapt to the behavioral coding approach. It wouldn't be difficult for more experienced engineers to change their coding style, either, he maintains. "Behavioral coding is what people do now when they develop Verilog testbenches."

Black sees a definite use for Behavioral Compiler, especially in situations when time-to-market is crucial. "This product has definitely improved my productivity."

"Instead of ending up with ten to twenty pages of RTL code that was error prone and difficult to follow, I had a small amount of compact code that I could just look at and understand what was going on." David Black, Design Engineer, Apple Computer

For more information on Synopsys Professional Services call your local Synopsys sales representative, or visit us on the Web at www.synopsys.com/psg

For information related to products, training or support services please visit us on the Web at www.synopsys.com. For sales assistance, please call 1.650.584.5000.



700 East Middlefield Road, Mountain View, CA 94043 T 650 584 5000 www.synopsys.com

Synopsys, the Synopsys logo and Designware are registered trademarks and Behavioral Compiler, Design Compiler and Test Compiler are trademarks of Synopsys, Inc. All other products or service names mentioned herein are trademarks of their respective holders and should be treated as such. Printed in the U.S.A.

©2001 Synopsys, Inc. 3/01.TM.WO