

www.scdsource.com  
Thursday, May, 6th 2010

## First Look

# Forte boosts Cynthesizer's automation capabilities

By Bill Murray

08/27/09

Forte recently enhanced its Cynthesizer high level synthesis (HLS) [1] tool with new automation features for partitioning complex hierarchical systems and generating complex interfaces, as well as offering memory support upgrades, enhanced scalability, and enhanced control-based design support. SCDsource digs below the press release; looks at the tool's power management record; overviews how Fujitsu Microelectronics Europe (FME) used the tool to design a custom floating point graphics processor; and hears Forte's take on C++/SystemC versus ANSI C.

### *Automated partitioning*

What's the particular partitioning problem to be solved? According to Brett Cline, the company's vice president of marketing and sales, the problem is partitioning fixed-function hardware to implement an algorithm that is often already described as a sequential software description optimized for execution on a processor. He said "The objective of partitioning is to implement an optimized architecture with the desired large-granularity parallelism." He continued "The algorithm is often written with multiple loops iterating over large memory arrays for intermediate storage. For efficient fixed-function hardware implementation, the designer must transform the architecture into a set of interconnected finite state machines (FSMs). These FSMs process the data streams and eliminate the need for memory arrays and other large storage blocks. So, the task is to partition the design into multiple parallel execution units."

And the solution? According to Cline, Cynthesizer now has an auto-partitioning feature that uses synthesis directives to identify sections of the sequential code as partitions. The designer decides which blocks of code are to be executed in parallel. The tool automatically analyzes the data flow between partitions using a pre-verified, interface template, and implements each partition as a separate thread with its own FSM. This allows the data to flow through the multiple blocks with the desired performance and without the need for area-consuming storage. Cynthesizer also ensures that the FSMs remain synchronized, thus avoiding data loss.

The auto-partitioning tool also generates the necessary interfaces and inserts them into the SystemC code, from which they are synthesized as RTL code. Cline said "For many common dataflow patterns, this works well with minimum designer effort." However, for more complex subsystems, "the designer takes a divide-and-conquer approach using a structural hierarchy of blocks and sub-blocks" – see the FME design example below.

The interfaces encapsulate all the signal-level connectivity and protocol. So the designer executes a single binding that enables the automatic connection of multiple, separate port/signal connections, and calls transaction-level access functions to move the data. Cline said "Cynthesizer interfaces are fully-timed, so they can be synthesized and can also be used in simulation. Simulating blocks and interfaces at a high level enables the designer to verify that all blocks are working together and all of the synchronization protocols operate correctly."

## Automated interface generation

Cline said “Cynthesizer eliminates the need to manually implement the inter-block protocol and connections, signal-by-signal. For some years, we have provided a set of interfaces as CynWare intellectual property (IP). With this new product release, our new Interface Generator (IG) builds specialized interfaces automatically. Of course, the designer still has the option to write custom interfaces in SystemC.”

The Interface Generator captures the interface requirements – such as data types and buffer depths – and produces customized interface code in synthesizable SystemC, which can be used for simulation as well as synthesis. According to Cline, the Interface Generator supports interfaces such as:

- A ‘trigger-done’ interface with single and double buffers, which transmits commands from one FSM to another and receives the result.
- A circular buffer interface that enables two FSMs to simultaneously access a memory while simultaneously synchronizing to ensure that the writer stays ahead of the reader.
- A line buffer interface used in – for example – image processing. The buffer stores data from multiple rows of a 2D image and transfers it to the algorithm implementation as a series of working sets that act as a sliding window over the image.

Cline added “The interfaces incorporate transaction logging using the SystemC Verification (SCV) logging mechanisms so that the transaction sequences can be viewed directly in transaction-enabled waveform viewers such as Springsoft’s Verdi.”

As an example, requirements capture of a 2D line buffer is shown in the screenshot in figure 1:

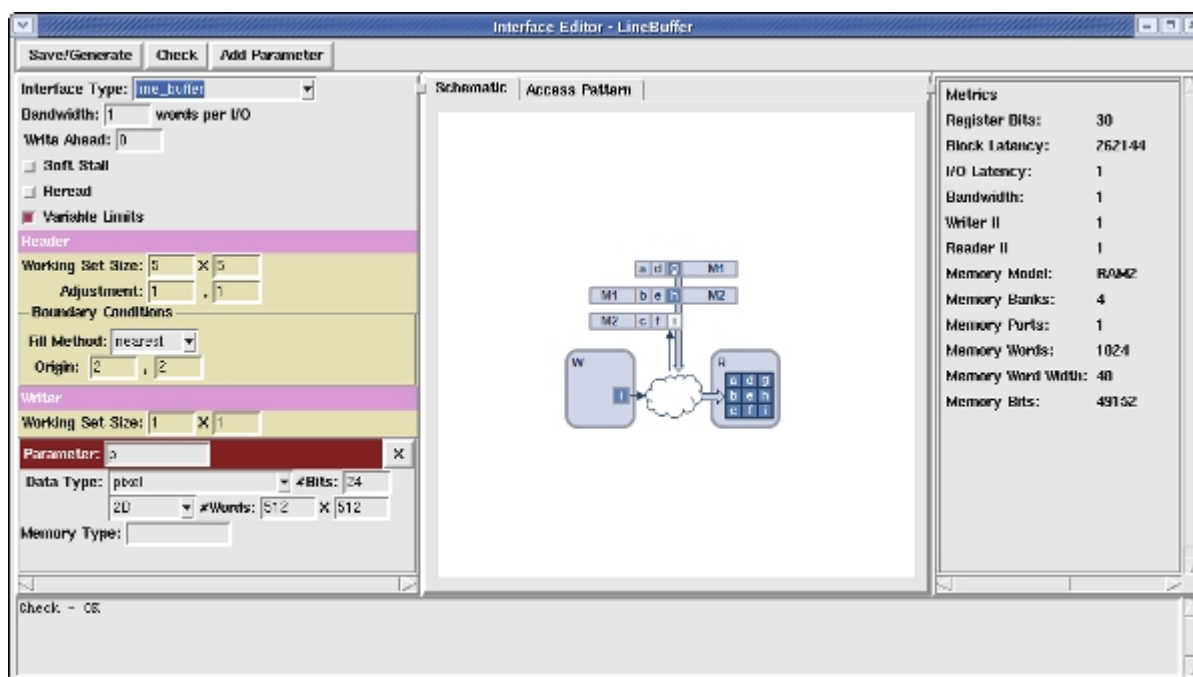


Figure 1: Interface Generator captures interface requirements of a 2D line buffer (Source: Forte)

According to Cline, the access pattern defines how the writer and reader of the line buffer interact with the data in the buffer. The access pattern also determines the minimum storage required to buffer the data, as well as the necessary address translations. The screenshot in figure 2 shows the accesses of a 5x5 “working set” from the larger 2D image. Each new access shifts the entire set by one pixel. The

user interface also defines virtual pixels that may be required by the algorithm. All of these implementation details are handled automatically by the generated SystemC interface.

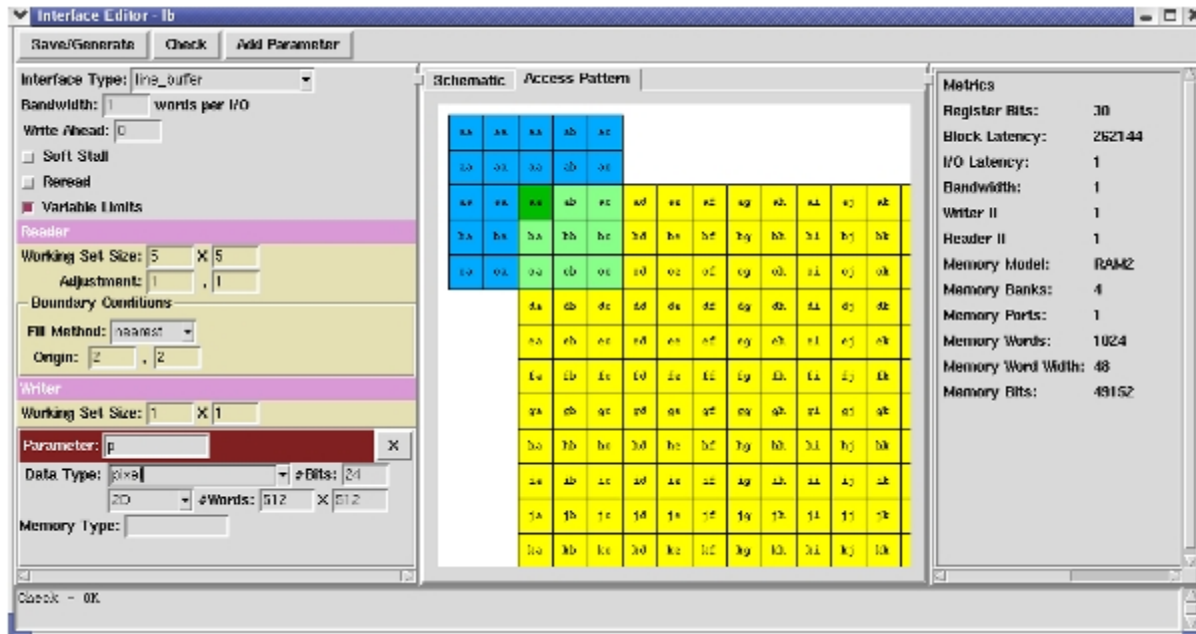


Figure 2: The access pattern for the 2D algorithm (Source: Forte)

### Memory support upgrades

Memory support features are critical because of the significant impact that they have on SoC area and performance. So, what kind of memory enhancements has Forte made?

Cline stated "We already support single and multi-port RAMs and ROMs; memories that bridge multiple clock domains; and single-port memories packaged with arbitration logic that can be shared by multiple accessing FSMs. In this release, we've added the ability to access a memory as a multidimensional array of arbitrary structures with user control of the bit ordering. This minimizes the coding changes needed to implement fixed-function hardware from an algorithm described in software. We also added contention checking to warn of shared-memory accesses that could create deadlocks and race conditions."

### Enhanced Scalability

According to Cline, the latest Synthesizer release has significantly greater synthesis capacity. He said "It can handle much larger blocks, and reduce both memory footprint and processing time. For example, one recent multifunction printer application had a single block of 600K gates that ran through Synthesizer in a matter of minutes. Synthesizer has been used to produce designs in the six-to-ten million gate range."

### Enhanced control-based design support

Before we addressed the enhancements, SCDsource wanted to know more about Synthesizer's established control synthesis capabilities. What kind of control circuitry can it synthesize? How complex?

The answers are illustrated graphically in figures 3 and 4. Figure 3 shows the basic functional block diagram of an image processing accelerator. Figure 4 shows the accelerator block in greater detail. Cline said "This is a complex, SoC-style design that communicates via a standard AHB bus, uses DMA

controllers, and complex interfaces such as line buffers, as well as deploying various algorithmic blocks. This control-heavy design can be accurately simulated using TLM interfaces for high speed, and pin-accurate interfaces to verify inter-block communications, using high-level hardware descriptions in SystemC. Synthesizer can synthesize the complete design, right down through the hierarchy."

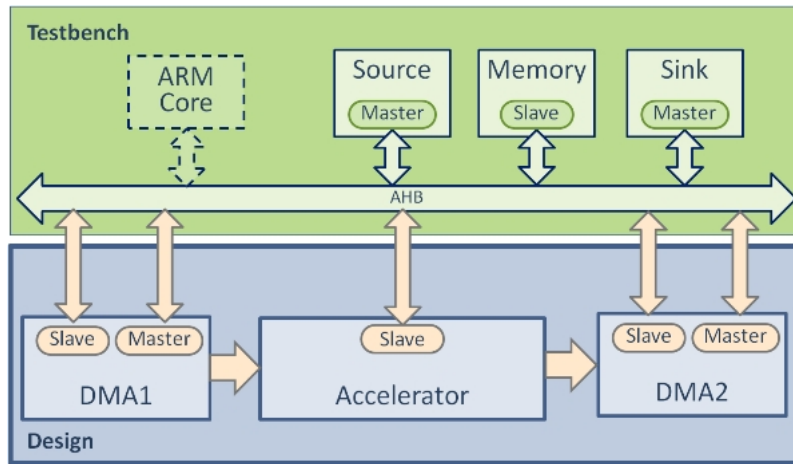


Figure 3: Typical SoC block implementation of an image processing accelerator (Source: Forte)

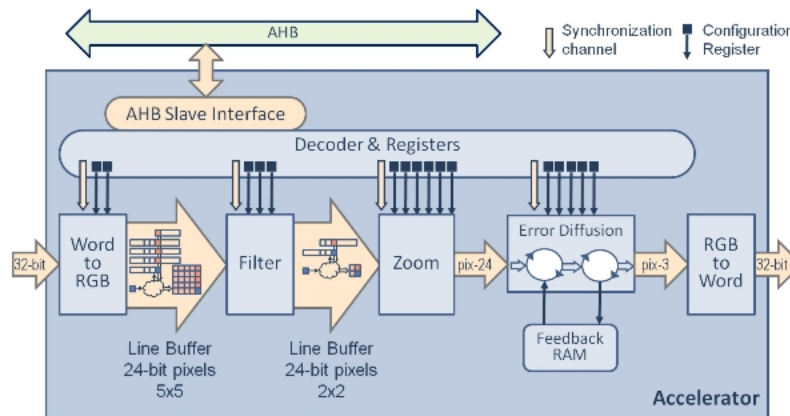


Figure 4: The accelerator functionality (Source: Forte)

So, what are the improvements in control-based design support?

Cline said "We've added a number of scheduling optimizations that achieve better results and support easier coding styles. For example, you can access an external memory from within a data flow with a simple instruction:

```
a[i] = 0;
```

where 'a' is a memory and 'i' is the index. This simple code implies the fully timed access to an external memory. The scheduler can handle single or multi-dimensional memories, and automatically inserts the datapath scheduling into the control circuitry."

Another enhancement example is in bus interface design using an address-mapping function. According to Cline, "Bus interface design typically leverages a standard protocol combined with the user's application-specific functionality. For example, the accelerator module's internal workings are controlled by a set of configuration registers and synchronization triggers. These are mapped to specific addresses in the AHB address space. When an AHB read or write transaction is addressed to the accelerator, the

accelerator must perform the standard AHB signaling, and also correctly “decode” the transaction and set or read the appropriate internal registers.”

He continued “The Cynthesizer flow enables the designer to focus on the application-specific functionality. The user needs only to write the address-to-register mapping code in untimed C, while the SystemC bus protocol module defines the I/O signaling. The module connects to the user’s mapping code using simple function calls, and together they constitute the application-specific bus interface. The bus protocol module can be written once, and re-used in multiple applications by combining it with the appropriate address-mapping function.”

How do these features stack up to the Catapult C control features announced recently by Mentor?

Cline said “The accelerator example shows that Cynthesizer’s level of control synthesis is far in advance of the generation of the simple control of sub-blocks in the fixed-function hardware implementation of a single datapath-centric algorithm. Mentor implements datatypes and its limited control support with proprietary synthesizable C++ constructs – they are actually trying to reconstruct SystemC functionality, but are doing so in a non-standard way.”

*What about power management?*

Mentor recently announced power management functionality in Catapult C. There are no power management enhancements in the latest release of Cynthesizer, so how does the tool stack up on power management?

Cline replied “Cynthesizer has had power features since the very early versions. It currently uses four key features to deliver low power designs:

- It maximizes the sharing of functional units and registers.
- It maximizes clock gating opportunities by generating RTL code that is specifically tailored for downstream tools such as Synopsys’ Power Compiler.
- It reduces memory power consumption by idling memories whenever they are not in use.
- Its rapid design exploration capability enables the user to choose the optimum area, power, and speed tradeoffs by integrating with tools such as Sequence’s Power Theater.

Cline said “Cynthesizer implements known best coding styles and design techniques. We’ve worked with customers for more than eight years to develop and refine a design and coding style that produces low power results and high quality RTL, while leveraging their investment in existing tools such as Power Compiler and Power Theater. We have a public track record in this arena.”

### **Fujitsu floating point graphics processor**

Forte illustrated the efficacy of its HLS tool with a newly-announced customer success. Fujitsu Microelectronics Europe (FME) used Cynthesizer to design a floating point processor to implement a programmable unified shader that uses the [OpenGL ES 2.0](#) shading language. The processor performs high-performance 3D rendering for applications that overlay computer graphics and video data. It is configurable to use high-level geometry, color and lighting algorithms for full-scene anti-aliased (FSAA) image generation.

The company applied a divide-and-conquer approach to the FPP design (see figure 5). The control and arithmetic blocks each consist of a dozen sub-blocks, each of which was synthesized individually. The sub-blocks were selected according to defined P2P protocol transmission points in the design, and bounded by the need to reduce complexity and, therewith, synthesis time.

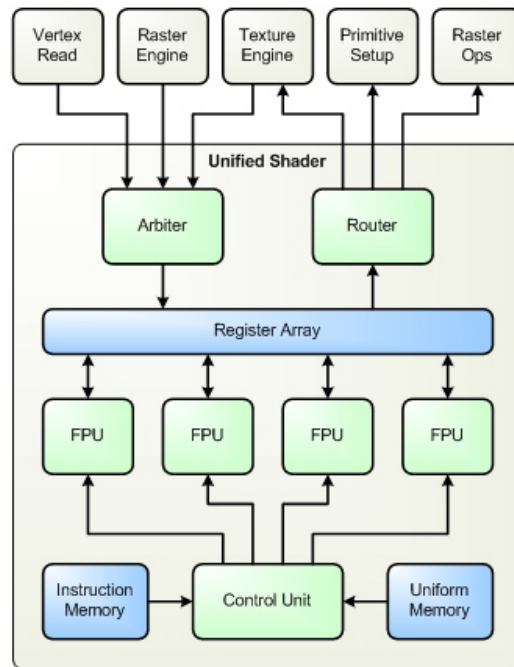


Figure 5: Synthesizer synthesized both control and arithmetic units in this FPP (Source: Fujitsu)

In an email swap with Raimund Sönning, FME's manager, hardware development, SCDsource learned that:

- The controller block was described in SystemC with 8K lines of code (LoC), from which the tool synthesized 348K Verilog RTL LoC.
- The arithmetic block was described in SystemC with 5K LoC, from which the tool synthesized 155K Verilog RTL LoC.
- A further 12K lines of SystemC code – capturing, for example, data type definitions – were shared.
- There was no IP reuse – everything was designed from scratch.

Sönning said “Architectural changes were significantly faster in SystemC than in RTL – particularly when changing data formats and types that are routed through and/or used in multiple parts of the design.”

What were the specific SystemC attributes that persuaded FME to use it? According to Sönning, they are:

- A single source for the hardware description and software development model.
- Better scalability and encapsulation via the higher abstraction level of functional code.
- Greater flexibility for test bench design.
- A widespread and accepted standard, with a particularly focus on TLM issues.

Sönning cited simulation speed versus RTL as a significant plus. FME's software engineers used the cycle-accurate SystemC models for software development, but would like untimed or loosely-timed models for greater simulation speed.

Sönning declined to say how much effort was expended in writing the SystemC code, but stated in the press release that a manual RTL design would have required significantly more time and effort.

## C++/SystemC versus ANSI C

So, back to the old, old question. How does SystemC-driven HLS stack up against C-driven HLS?

Cline said "Unlike pure ANSI C/C++, SystemC allows the designer to specify both control and datapath behavior within the semantics of the language, enabling the creation and verification of a fully accurate, high level simulation model." He continued, "In contrast, ANSI C is a sequential, untimed language with no explicit parallelism. It has no bit-accuracy and it cannot capture hierarchy adequately. That's why – going back to my earlier statement – Mentor is using proprietary, non-standard, synthesizable C++ constructs to reconstruct SystemC functionality."

Nonetheless, many industry standard algorithms – such as graphics, encryption and data compression algorithms – are expressed in ANSI C. And Mentor claims considerable market success. Doesn't it make sense to synthesize from ANSI C?

Cline said "Our customers utilize their C algorithms using SystemC extensions such as the `sc_uint<range>` datatype. The effort is the same as using the datatypes for C-driven HLS tools. You can think of Cynthesizer as handling ANSI C algorithms with a SystemC wrapper." He continued "ANSI C is a successful software programming language, but SystemC was devised with real hardware design in mind. Cynthesizer can model, synthesize and simulate the entire design at a high level of abstraction."

## References

[1] ESL Design and Verification by B. Bailey, G. Martin and A. Piziali. Morgan Kaufmann, 2007. [Chapter 11: Hardware Implementation](#).

## Further Reading

[Mentor Catapult C synthesizes control and power management](#)

All materials on this site Copyright © 2007-2009 Tech Source Media, Inc. All Rights Reserved | [Privacy Statement](#)