Semifore

Taming Design Complexity with Semifore – A Case Study

1 The Challenge

Electronic system design is undergoing a major shift. For decades, the design process was primarily focused on a monolithic system on chip (SoC). All functions could be integrated there and moving to the next process node provided the required increase in performance and reduction in power and area to remain competitive.

Today, things are different. The SoC has been replaced with a system of chips and chiplets, integrated in a complex and sophisticated package. Many of the devices in these new systems are purpose-built devices performing specific tasks, often related to AI. The complexity of the hardware and the associated software stack that interacts with it has become a substantial design challenge.

One of the key drivers for this step-function increase in design complexity has to do with who is doing the design. While semiconductor companies are still a major force in the industry, an increasing number of system OEMs are now designing chips as well. The best way to achieve differentiation with silicon-enabled software is to control the whole process. And so, there are new names in the semiconductor ecosystem. Google, Amazon, Facebook, Alibaba, and Microsoft are just a few worth mentioning.

These organizations are now driving the development of highly complex hardware/software systems and consuming a fair percentage of advanced wafer fab capacity on a worldwide basis.

2 What's Needed?

What is needed is a holistic view of the process.. The interaction between the hardware and software elements of the design needs to be understood and optimized as early as possible to ensure a successful outcome.

These tools must maintain architectural integrity throughout silicon implementation. This increases productivity. For the balance of the discussion, we'll focus on how the hardware and software interact. This is a critical part of the design process. Errors in the hardware/software interface become exponentially harder to detect and fix as the design matures. If a bug escapes to the final product, future software updates may not be possible. The stakes are very high.

Tools used to address these challenges must be designed for early expression of the "nuances" of chip hardware register and memory mapping requirements. Something like standard SystemRDL, but much more expressive. There is a need to automate the management of large numbers of hardware registers and fields (millions) – where manual methods are slow and error-prone.

And most critical, hardware/software interface bugs need to be identified before the hardware register implementation is too committed to change.

3 A Case Study

A large system OEM shared its experiences with taming design complexity at the Design Automation Conference. The team characterized the problem as follows:

The project was a custom co-processor designed. The design contained over 300,000 CSRs (control/status registers), comprised of about 900,000 fields with over 500,000 lines of CSR Verilog RTL representing around 5% of the total die area. The team was managing over 600 discrete CSR specification files in various standard formats.

Regarding coherency, multiple specification formats were required due to the needs of downstream processes and partners. Examples include:

- Synthesizable Verilog RTL for verification and physical design
- Header files for software, firmware, design, and verification
- Data structure definitions for software, firmware, modeling, and verification
- UVM RAL (register abstraction layer) models for verification
- Microsoft Word and HTML documents for internal use, and internal and external partners

This team's primary observation was that manually keeping of all the various CSR specification files up to date was inefficient, costly, tedious, and error-prone.

Based on these design requirements, the team chose to deploy Semifore's CSRSpec™ language and CSRCompiler™ to address their project needs. The reasons for their choice were summarized as follows:

3.1 CSRSpec Language – Benefits

- C-like syntax
- C-like macro preprocessor
- Parameterizable templates
- Arrays of objects (registers, memories, groups)
- References (links)
- Conditionals, loops, and case statements
- Enumerations
- Fields wider than word width span multiple addresses
- Address map hierarchies easily composed using external references and "address map linking"

3.2 CSRCompiler – Benefits

- Rich set of command-line and control file options to enable customization and flexibility
- Free tools such as RALGEN and RGM did not provide all the various output formats needed (e.g., Microsoft Word, HTML, IP-XACT)
- Fast: full-chip HTML output in ***19 seconds*** *(over 300K registers)*

3.3 Overview of Features of the Methodology

An automated flow based on an enhanced make utility, a set of shared makefiles, and shared CSRCompiler control files was configured. Compile avoidance was based not on timestamps, but lexical analysis of CSRSpec source files and checksum-based signatures. Dependencies were inferred on-the-fly, eliminating rule dependency errors and makefile maintenance costs. Also, a common set of makefiles ensured project-wide uniformity and ease of use.

Build required only one simple command. Rule design ensured that outputs for dependent higher-level address maps were automatically built, and cron-driven full builds ensured version-controlled derived outputs that were coherent with the source CSRSpec files.

3.4 Results

Using this methodology, the number of lines of manually maintained CSR specifications were reduced across all formats from over 13 million to 225 thousand—a compression factor of about 60x. Use of CSRSpec templates and address map linking reduced source code copy-paste errors and coherency problems. For example, changing one line of CSRSpec in the “ID” template source accurately and consistently updated over 250 template instances.

The team’s assessment was that, without a single-source approach, more costly and error-prone redundant entry and maintenance would have been required. The team pointed out that a single-source approach eliminated file coherency issues and maintained referential integrity. All this improved productivity and decreased time to market in the team’s view.

4 Summary

We have shared the real experiences of a design team at a large system OEM to tame their design complexity. The bottom line from their discussion was that the explosion in control/status register design size can be mitigated using a single-source, single-flow approach.

Positive results were achieved by:

- confining human capture and maintenance to single set of CSRSpec source files
- relying on CSRCompiler to produce the desired specifications automatically and faithfully in various standard output formats
- effective use of hierarchy, shared templates and macros, and shared tooling

The use of a single set of CSRSpec source files also eliminated coherency challenge. CSRCompiler, driven by the make flow, ensured that the various derived output files were faithfully kept up to date.

If you'd like to explore how Semifore can help with your next design project, please contact us.

Semifore, Inc.
(650) 960 0200
sales@semifore.com
www.semifore.com