Platform-Based System-on-Chip Design

Don Bouldin
Electrical & Computer Engineering
University of Tennessee
Knoxville, TN 37996-2100
dbouldin@tennessee.edu

Abstract

Developing a complete System-on-Chip (SoC) with a CPU core and perhaps a dozen virtual components by a fixed deadline is no easy task. Designers may encounter business and legal problems in obtaining the virtual components and may find that information is missing. By first developing a platform containing these components, designers can overcome all of these uncertainties without risking the delay of a product. Once the platform is fully operational, derivative designs in which only a few virtual components are added or dropped can be accomplished rapidly. Xilinx, Altera, Triscend and Atmel have integrated CPU cores onto their chips and now offer platform SoCs that are programmable. The issues involved in adopting this approach are discussed.

1. Introduction

Multi-million transistor integrated circuits can now be produced cost-effectively [1]. Thus, designers are faced with the challenge of creating and verifying the content of these chips as quickly as possible in order to reduce the time-to-market. It has been estimated that a one-month delay in bringing a product to market can result in a loss of ten percent of the potential revenue [2].

Hence, not all of the transistors on these chips can be customized but instead must be ported from previous designs. These reusable cores or intellectual property (IP) blocks include CPUs (like ARM, MIPS and SPARC), MPEG decompression engines, PCI bus controllers, specialized DSPs, etc. Combining several complex cores using gates and standard cells is much more manageable and quicker than designing millions of transistors one at a time.

2. Reusing Components

The myth that characterizes today's IP is that these components are blocks that have well-defined contents and interfaces. However, they are often fuzzy and hence appear more like patches in a quilt, which must be stitched together. The components cannot be assembled blindly and rapidly, but rather must be carefully pieced together to form a working system.

Therefore, design for reuse does not come free. Rather it involves much more in-depth documentation and characterization than for a design that is not intended to be reused. Based on the experiences of software engineers [3], it is estimated that preparing a component for reuse will require about 50% additional effort.

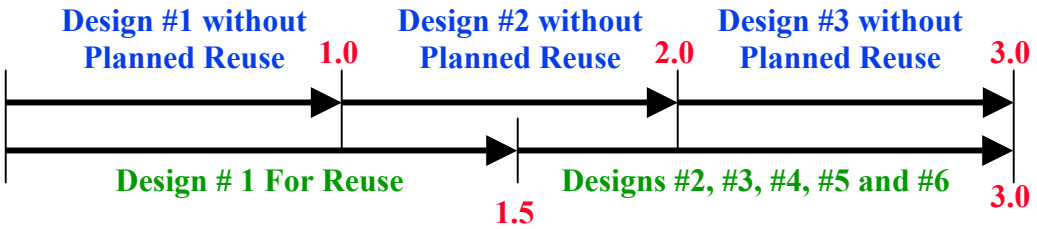


Fig. 1. Timeline Comparison of Design Approaches.

Once this has been done, the designer who is reusing the component may naively think that his design time for that component will be reduced to zero. But alas, he must take care to understand fully how the component works and how it should be integrated with other components. Again from the experiences of software engineers [3], the second design generally requires about 30% of that required to produce the component originally. Thus, the reuse is not for free but does make a significant (70% reduction) impact on the next design. A comparison of the traditional approach in which design reuse is not planned versus this new approach is depicted in Figure 1.

The information required to document soft IP consists of far more than just the source code. Also needed are: (1) functional description, (2) application intent, (3) interface specifications, (4) authors and owners, (5) packaging information, (6) **input stimuli and output responses (test bench)**, (7) tools and versions used, (8) FPGA or ASIC foundry used for fabrication, (9) size, delay and power measurements, and (10) testability features including BIST, JTAG and SCAN.

3. Design of Platform SoCs

System-on-Chip (SoC) design may involve the mixing on a single integrated circuit a microprocessor core (e.g. ARM, MIPS, SPARC), PCI bus interface, analog components and numerous digital processing functions. Figure 2 depicts a typical SoC.

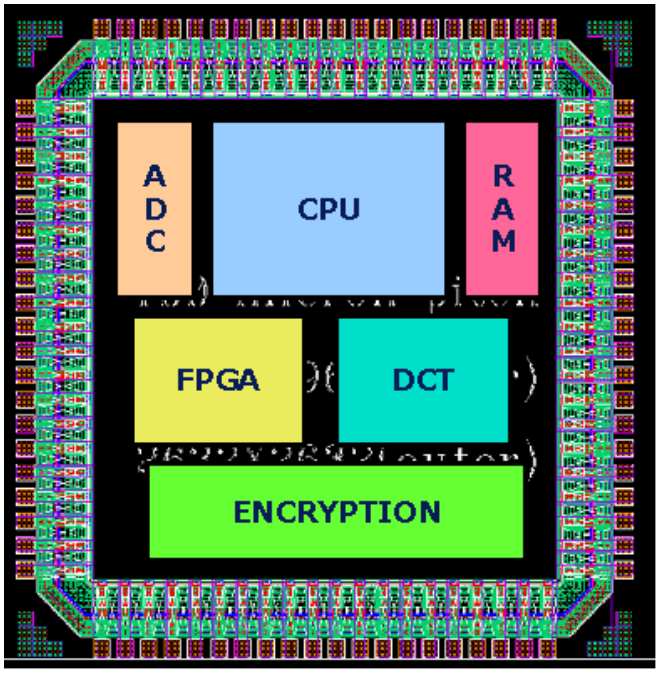


Fig. 2. Typical SoC

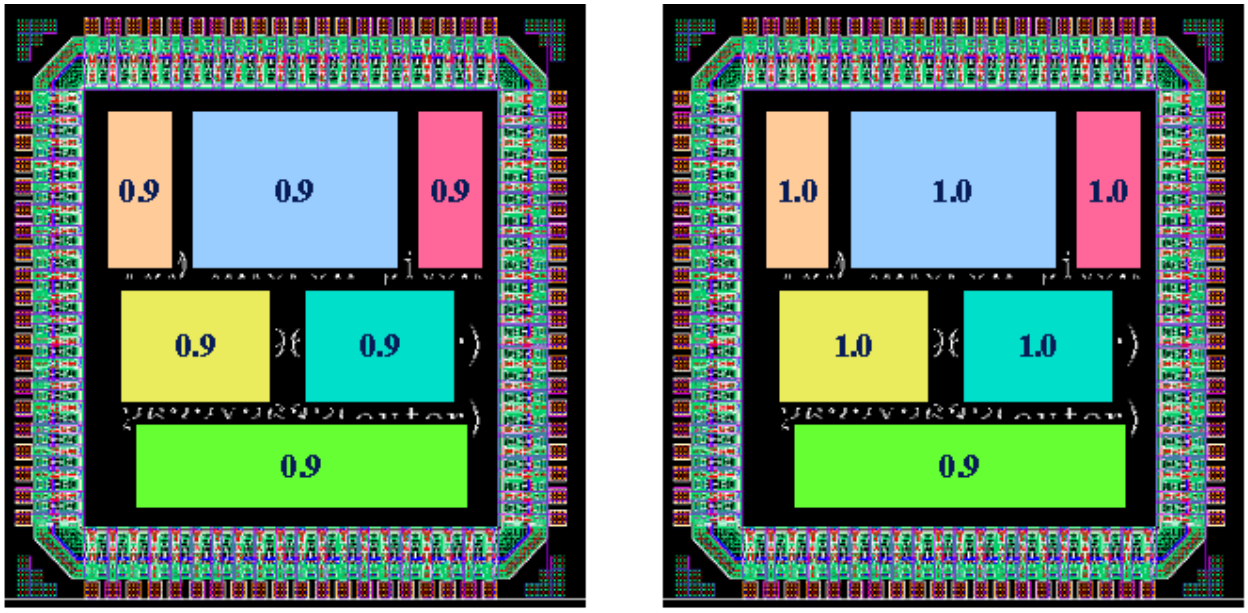
Designers are increasingly reusing significant portions of previous designs to reduce the time to market which generally results in greater revenue for the product. Reuse of previous designs has been occurring for decades. Initially, only simple library cells were implemented with reuse in mind and this continues today. In the past few years, major functions have been implemented as virtual components. Since these may have been developed by designers in other companies, their reuse involves a combination of effort and risk in a new design. To minimize these, some organizations are internally standardizing on a set of virtual components and any associated software to develop their own SoC platforms.

Platform-based design allows an organization to develop a complete SoC that is central to its product line. Once the SoC platform is fully operational, derivative designs in which only a few virtual components are added or dropped can be accomplished rapidly.

Developing a complete SoC with ten or more virtual components by a fixed deadline is no easy task. Designers may encounter business and legal problems in obtaining the virtual components and may find that information is missing. Developing a platform permits designers to overcome all of these uncertainties without risking the delay of a product.

Figure 3 depicts a comparison of using unproven versus proven components. If the probability of being correct is 0.9 for each component and their interconnection, then the probability that the entire SoC will be correct is only 0.5. Thus, a significant amount of time must be taken for verification to achieve first-pass success. On the other hand, if all of the components except a new one and its interconnection are already proven, then the probability of the entire SoC being correct is improved to 0.8. Thus, far less time need be allocated for verification. **It is this contrast which serves as the compelling motivation to adopt a platform-based design approach.**

Moreover, a platform SoC also provides software developers with working silicon they can use. The organization can market the platform SoC to customers as a demonstration of what can be done and even provide the customers with the opportunity to commence their own product development using the existing SoC. Whenever the customer determines that it is worthwhile to have a derivative design customized for his product needs, the platform SoC designers add or subtract a small number of virtual components and revise the associated software. The derivative design can likely be done in less than six months from concept to production.



Interconnect 0.9 so
0.9x0.9x0.9x0.9x0.9x0.9x0.9 = 0.5

Interconnect 0.9 so
1.0x1.0x1.0x1.0x1.0x0.9x0.9 = 0.8

Fig. 3. Comparison of Component Risk.

4. Existing Platform SoCs

Several organizations are using this platform-based design approach. Philips Semiconductors has developed a digital video platform SoC intended for set-top boxes. The SoC includes a 32-bit MIPS microprocessor core plus Philips' own Trimedia core and an MPEG-2 decoder. Interface circuitry for PCI, UART and USB are also included [4].

Tality, the design services spin-off of Cadence, has developed two SoC platforms. One includes both an ARM microprocessor core coupled with the popular OAK digital signal processor. Another platform is targeted for the bluetooth wireless market [5].

Infineon has developed a triple-mode SoC platform for wireless applications. It includes a 32-bit microcontroller and a digital signal processor [6].

FPGA vendors have integrated CPU cores onto their chips and now offer platform SoCs that are programmable. Triscend offers both the 8032 8-bit microcontroller and the 32- bit ARM7TDMI core with its programmable logic family [7].

Xilinx offers a choice of the IBM PowerPC, Intel's StrongARM and Pentium class, and QED processors for its Virtex and Spartan product line. The PowerPC 405 is embedded as a hard-core in the Virtex-II architecture and can operate at 300 MHz to produce 420 Dhrystone MIPs [8].

Atmel offers the popular 8051, ARM and AVR 8-bit microcontrollers embedded in its programmable product line. A field-programmable system-level integrated circuit starter kit is available at very low cost. [9].

Altera offers a choice of CPU cores including the ARM, MIPS Technologies and Altera's internally developed Nios embedded processor [10].

5. Developing Open SoCs

A database of commercial IP blocks is now on-line [11]. Thus, SoC developers can identify reusable components to be integrated into their platforms. Business and legal issues must be pursued individually for each component but the development of a SoC is undoubtedly facilitated.

Free IP blocks are also available on-line [12]. These include USB 2.0 and Ethernet MAC interfaces as well as DES/AES encryption blocks and microcontrollers. A 32-bit SPARC-V8 that was developed by the European Space Agency is also available on-line [13]. The source codes for all of these can be downloaded by anyone at no charge.

Thus, universities and individuals can and are developing open SoCs to serve as education and research platforms. In our advanced graduate electives at the University of Tennessee [14], the initial course in a two-semester sequence provides the students with the opportunity to learn how to synthesize small pieces of HDL source code into FPGAs. In the second semester, larger projects are assigned that require a team of generally four students to implement. The application requirements are first presented in narrative form and the team members must partition the design into manageable modules. Each module is the responsibility of an individual to capture in VHDL, synthesize and simulate as well as verify in an FPGA. Once each student believes his design is "known good", the team then integrates the components into a single-chip ASIC. Obviously, any deficiencies not already corrected by individual designers must be dealt with during this integration or design with reuse phase. It is not unusual for a student to recognize that the quality of his component or his documentation is substandard and hence some redesign or additional documentation is performed until the full system works properly.

Projects following the model just described are intended to provide individual students with the experience of designing *for* reuse and the team of designers with the experience of design *with* reuse. Example projects completed or underway include: Wavelet Image Compression, Huffman Encoding, LZ Data Compression, Discrete Cosine Transform, Fast Fourier Transform, CORDIC 2-D Vector Rotation, Automatic Target Recognition, Constant False Alarm Rate, Data Encryption, and Boolean Satisfiability.

All of the examples given so far have been for developing soft IP using a HDL for implementation using FPGAs or single-chip ASICs. These are appropriate for advanced graduate electives which are targeted at developing system-on-a-chip designers. For senior capstone courses, projects must generally involve integrating existing hardware and software components with only a limited amount of time available for creating new components.

6. Conclusions

Developing a platform SoC with a CPU core and perhaps a dozen virtual components removes the uncertainties about the individual components and their interconnection such that derivative designs can be accomplished rapidly. Programmable SoCs and commercial and open cores can be exploited in this endeavor.

ACKNOWLEDGMENTS

The author gratefully acknowledges the support of DARPA/AFRL grant F30602-01-2-0562. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies of the U.S.Government.

REFERENCES

[1] http://www.itrs.net.ntrs/publntrs.nsf

[2] http://www.aw.com/catalog/academic/product/0,4096,0201500221-DES,00.html

[3] http://aemp.eeel.nist.gov/reuse/

[4] http://www.nexperia.com/

[5] http://www.cadence.com/company/pr/pr00-bluetooth_soc.html

[6] http://www.infineon.com/

[7] http://www.triscend.com/

[8] http://www.xilinx.com/

[9] http://www.atmel.com/

[10] http://www.altera.com/

[11] http://www.design-reuse.com/

[12] http://www.opencores.org/

[13] http://www.gaisler.com/

[14] http://vlsi1.engr.utk.edu/ece/bouldin_courses