The Case for a Configure-and-Execute Paradigm

Frank Vahid and Tony Givargis
Dept. of Computer Science and Eng.
Univ. of California, Riverside, CA 92521
{vahid,givargis}@cs.ucr.edu, www.cs.ucr.edu/~dalton

ABSTRACT

Tomorrow's silicon chips will hold more transistors than most embedded system designers could possibly use under the prevalent "describe-and-synthesize" design paradigm. Many have thus re-proposed the once popular "capture-and-simulate" paradigm, wherein pre-designed Intellectual Property software and hardware components are connected and co-simulated, to reduce this gap. However, major hurdles limit this paradigm to only very high-cost embedded systems. In this paper, we describe those hurdles and present a case for a new "configure-and-execute" paradigm for mainstream embedded systems, based on the idea of deconstructing rather than constructing systems, which takes advantage of the surplus transistors in a way that can overcome the hurdles and significantly reduce time-to-market.

Keywords: System-on-a-chip, cores, IP, methodology.

1. INTRODUCTION

Trends in silicon chip capacity indicate that next decade's chips will have massive transistor counts compared to the previous decade, implementing entire systems-on-a-chip (SOC). Unfortunately, design methods have not kept pace, resulting in a "productivity gap," resulting in underutilized transistors.

Synthesis is a commonly proposed solution. In the past, the "capture-and-simulate" paradigm [4] was prevalent, wherein one captures gates or register-transfer components in a structural schematic, and then simulates to verify correctness. Recently, the "describe-and-synthesize" design paradigm has taken hold, wherein one describes desired functionality, and then automatically synthesizes a structural schematic.

The productivity gap continues to increase, though, so many are proposing a paradigm based on design reuse. It is essentially a return to a capture-and-simulate paradigm in which the components, rather than being gates, are standard and custom processors, called Intellectual Property (IP) or more specifically "cores." However, capture-and-simulate has major problems for SOCs that didn't exist previously, which will result in high SOC design costs and thus limit SOC design to high-end niches, leaving most designers still under-utilizing potential transistors.

We can, however, still make good use of all those transistors by adopting a new paradigm, which we call *"configure-and-execute."* An embedded system designer acquires a silicon reference design for a particular application class (e.g., still video processing), and then develops his/her specification application (e.g., a digital camera) by repeatedly configuring the design and executing it in its real environment. The silicon reference design is an over-designed general-purpose working silicon SOC intended for the application class, with the SOC design costs amortized over numerous applications. When done developing, the embedded system designer can optionally spin specialized silicon, which is virtually guaranteed to be correct the first time because of the extensive in-circuit verification already done.

This paper summarizes current trends, describes problems with applying a capture-and-simulate paradigm for SOC's, introduces the configure-and-execute paradigm and argues for its use in mainstream embedded system design, and provides preliminary data for a digital camera example, developed as part of the Dalton project at UC Riverside, supporting those arguments.

CODES '99 Rome Italy

2. TRENDS

The driving trend is the existence of a huge number of transistors on chips in the near future. By 2006, an inexpensive 200mm^2 die would have 75M logic transistors, or 375M SRAM transistors [8]; a high-end 1000mm^2 die would have 400M logic transistors [18]. To illustrate the luxury of this silicon real estate, consider the following transistor counts for various cores [8]:

- 486DX4 core: 0.7 million
- Pentium/MMX: 2.8 million
- MPEG-2 encoder/decoder: 1.5/0.5 million
- 8051 microcontroller: 0.05 million

In other words, we could fit up to 7500 8051 microcontrollers on a single chip. Assuming a typical core has 0.5 million transistors, we could fit between 150 and 750 cores on a single chip.

Designers simply cannot build such complex systems under the cost and time constraints of most applications. The Semiconductor Industry Association (SIA) states that while transistors per chip have increased 58% per year, a designer's ability to use transistors has improved only 21% per year, leading to a widening productivity gap. This inability to use transistors has led to most systems not utilizing potential transistors; data from a major silicon vendor indicates that the actual number of transistors per design start has increased only 25% per year [13].

3. CURRENT PROBLEMS

The industry has generally recognized that transistors are underutilized and many seem to agree that synthesis and IP reuse will help solve the problem [18][20]. However, we argue that even these techniques will not enable designers to really utilize the large number available transistors, because cost and time requirements would still be prohibitive for most applications.

Synthesis follows the "describe-and-synthesize" paradigm. One first describes desired functionality in a program-like language, and then synthesizes a structural implementation. A problem is that describing functionality is error-prone – some data suggests

about 1 bug per 100 lines of code, independent of code size [1]. A similar problem has faced the software community for decades.

Therefore, synthesis will almost certainly need to be supplemented with IP reuse. However, the commonly-proposed paradigm of connecting IP together, essentially representing a return to a capture-and-simulate paradigm at a higher level of abstraction than before, has major problems of its own.

3.1 Verification -- the main problem

The main problem in building SOCs by connecting IP relates to verification of the correctness and completeness of a design's functionality. Some state that verification accounts for 1/3 to 1/2 of the system design process [18], others say it accounts for 2/3 [3][14]. Thus we see that verification is a bottleneck in system design, and the idea of building systems by connecting IP does little to address this problem.

3.1.1 Simulation

Simulation is probably the most common verification technique. It is straightforward for systems with thousands of transistors, but very difficult to perform well for million transistor systems.

The main problem with simulation is that simulating a reasonable amount of real time requires an absurdly long amount of simulation time. For example, 100 seconds of real-time for a million-transistor design requires 10+ years of RTL simulation [9][13] (cycle-based simulators help only incrementally). Such long simulation times result in typically less than one second of real time being simulated by designers. Cosimulation techniques gain perhaps an order of magnitude, which is not enough.

A second problem with simulation is that developing the system's environment for simulation (i.e., the testbench) can take enormous effort, since multi-million transistor systems often have very complex environments, unlike simpler multi-thousand transistor systems. Some claim that testbench development is the real bottleneck in verification [3]. Since most of this time is spent by the verification team trying to understand the environment and the system, there is not much potential for time reduction, and verification productivity tools like random test generators provide only incremental improvements.

A third problem is that environments often have undocumented features, which obviously can't be captured in a testbench no matter how much effort is expended.

We can conclude that *solving these three problems requires at-speed (or nearly at-speed), in-circuit verification techniques.*

3.1.2 Emulation

Emulation is commonly used to provide nearly at-speed verification. It usually consists of the use of a general-purpose FPGA (Field Programmable Gate Array) platform along with microprocessor and other components configured to act like the system being designed. However, emulation has problems too.

The first problem is that emulation requires many weeks, often over a month, to set up [3].

The second problem is that compile times are long for large designs, often lasting almost full day, and thus preventing frequent iteration among design and verification.

A third problem is that general-purpose emulators are expensive, ranging from $100,000 to $1,000,000, thus limiting their use to high-end applications, and sometimes creating a new bottleneck of different design teams having to share a single emulator.

A fourth problem is that emulators may still run 10 to 100 times slower than the eventual system [13], which may prevent proper functioning in the system's environment.

3.1.3 Silicon spins

To really begin verifying a system in its environment, we often require first silicon (i.e., a semi-custom chip) to be generated. Because of the above problems with simulation and emulation, first silicon usually still contains bugs, even after extensive simulation and emulation [13][16]. Each silicon spin can take months, and a recent study showed that the average number of spins required to verify a design was 3.5 [16], resulting in over 50% of development time occurring *after* first silicon

3.2 Other problems

Test checks manufactured chips for defects. Testing costs are projected to increase [2]: the cost per automated test equipment divided by the number of chips it tests per hour rose from $600 in 1985 to $6000 in 1998, and is estimated to rise to $100,000 in 2005. Furthermore, the test cost per transistor is also increasing (while design costs per transistor are decreasing).

Integrating cores can be a very difficult task. Even when cores have been designed to interface to the same bus, detailed timing problems can often arise, as well as load problems. Furthermore, integrating cores often requires a good understanding of the core's functionality, which can take much time. Finally, there is a problem of "compounded risks" [13], such as if there is a 98% probability of successfully integrating a core into an SOC, then the probability of successfully integrating 100 cores is only 13%.

Physical design has become "really, really, really hard" [17] in deep submicron technology. Chip design must be integrated with physical design, and designs must be created to ensure high yields. Custom techniques are becoming more necessary.

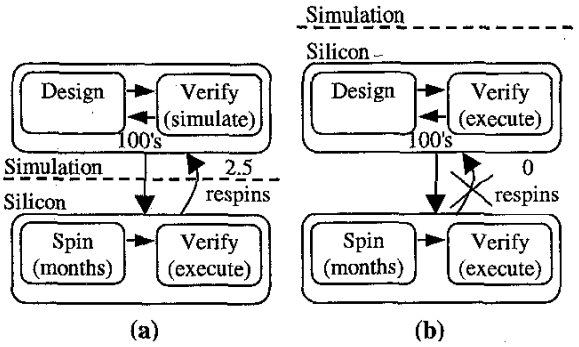
Time-to-market constraints continue to shrink, with average time from product conception to delivery reduced to 8 months [11], with further reduction likely. Such crushing constraints greatly increase the need for correct silicon on the first spin.

3.3 Summary of problems

Building SOCs, even with extensive IP reuse, is time-consuming and costly, because simulation covers a small fraction of real-time and can't address undocumented environment features, emulation requires much time and is expensive, and silicon spins require months. Furthermore, testing costs, integration and physical design problems, and tight time-to-market constraints add to the difficulties in building SOCs. Large SOCs will therefore be limited to a very small percentage of designs [5].

Mainstream embedded system designers thus will build systems that greatly underutilize potential transistors. Even today, potential transistors are underutilized by a factor of ten [13]. This underutilization will likely get worse as chip capacity grows and the productivity gap widens; if the current gap continues its rate of growth, designers in 2006 will underutilize potential transistors by a factor of 68, and in 2012 by a factor of 287.

Figure 1: (a) Simulation-based development leads to several respins, (b) Silicon-based development may eliminate respins.



4. CONFIGURE-AND-EXECUTE

Since most designers can't build systems that take advantage of potential transistors, we can instead use a different paradigm that makes use of those transistors to greatly reduce time-to-market. The paradigm is based on two points. First, designs for different products within the same application class have similar hardware architectures. Second, *deconstructing* (configuring and adding to or deleting from) an existing design that subsumes one's desired design is much easier than constructing a design from scratch (as evidenced by many designers starting from past designs in practice). These points lead us to a "configure-and-execute" paradigm, involving providers and users of reference designs.

A *reference design provider* is a company with extensive SOC expertise that builds a silicon chip, implementing a reference design, for a given application class. A reference design is an over-designed working system for an application class, containing more cores than would likely ever be needed by any particular application, and built to be easily configured and modified to implement most instances of applications of its class. The discussion of the earlier sections demonstrated that there is plenty of room on the chip for such extra cores. The high cost of building a reference design could be amortized over a large number of applications in the class. Reference design builders would likely make extensive use of advanced synthesis, verification, testing, and physical-design techniques.

A *reference design user* is a typical embedded system designer, who acquires a reference design and then configures it for a specific application. The user thus has working silicon from the very start of the design process, running at real speeds and executing in a real environment. If the user decides to generate specialized silicon for a given configuration in order to reduce chip costs, power, size, etc., then first-time correct silicon is quite likely since extensive verification in the system's real environment has already been performed. Figure 1 illustrates the time-to-market advantage gained by the user. Whereas a simulation-based paradigm typically requires respins, the configure-and-execute paradigm is carried out on real silicon, and thus respins are very unlikely.

4.1 Reference designs– "fig chips"

As described above, a key aspect of configure-and-execute is the existence of configurable reference design chips, or what we refer to as "fig chips." A fig chip is a fabricated and packaged silicon integrated circuit having a configurable architecture targeted to a particular class of embedded computing systems. A typical fig chip would include microprocessors, digital signal processors, memories, buses, field-programmable logic, and pins that provide access to the internal bus for extendibility (ideally providing for cascading). Furthermore, a fig chip would include a large number of cores specific to its application class.

Fig chips are classifiable into prototype-oriented and product-oriented chips. *Prototype-oriented* fig chips are very general devices intended to cover as broad an application class as possible. These chips would likely be far too large, expensive, and consume too much power to actually be used in a final product. They would likely have nearly the maximum number of transistors and pins (e.g., 3000 pins in the year 2006 [9]) as technology allows, costing perhaps hundreds of dollars each, and consuming perhaps 100 watts of power. They would be used during development, after which a product-oriented fig chip or specialized chip would be used in the final product.

In contrast, *product-oriented* fig chips are intended for use in final products. Though still more general than a particular application instance, they would be small, cheap, and consume a small enough amount of power to be used in final products. In fact, because of their production in large quantities, and potential for optimized configuration, their power consumption could be even lower than obtainable by a typical custom SOC designer.

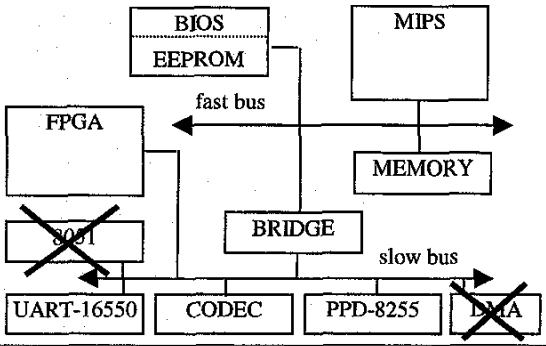
An example of a fig chip for control systems (e.g., closed-loop automatic control) would include a microprocessor, digital signal processor, cache, memory, direct-memory access controller, and a two-level bus (processor-local and peripheral). The peripheral bus would link a large number of cores specific to control systems, such as microcontrollers, numerous analog-digital converters, pulse-width modulators, counters, timers, serial communication devices (UARTs), and numerous blocks of field-programmable logic.

Field-programmable logic on a fig chip serves two purposes. The first purpose is to provide for use of cores that are not already on the fig chip. In this case, the ideal core would be specifically optimized for the programmable logic, typically resulting in only a factor of 10 slowdown or less [21] compared to a core implemented as an ASIC (application-specific integrated circuit). Otherwise, the core could be synthesized and mapped to the logic. The second purpose is to implement custom logic, which is estimated by Dataquest to occupy only about 10% of an SOC, with the remaining 90% of the SOC being made up of cores [6].

A key aspect of a fig chip is that it is designed for development and debugging, implying that its internal registers should be controllable and observable, and step-by-step execution should be supported. Fortunately, scan technology can be used to provide such features [13]. Fig chips would thus come with debug environments providing software control of the execution of the system from a development workstation or PC.

It is important to point out that a fig chip would be a complete working system. This means that any operating systems would be pre-installed, all drivers for controlling peripherals would be included, and template software would be running on the microprocessor exercising these items.

Figure 2: A sample reference design.



4.2 Configure

After acquiring a fig chip, a designer configures the chip for his/her own purposes. This implies that the fig chip must be designed with configuration in mind. Drivers must be easily turned off. Cores must have settable parameters. Cores should be capable of being shut down if they are not needed (thus consuming no power). Software templates obviously must be easily modifiable. For prototype-oriented fig chips, additional parameters must be settable, such as cache-size (and associativity, replacement policy, etc.), bus size, memory size, etc. A fig chip designed with configuration in mind can make designing a system much easier than constructing a system.

4.3 Execute

The designer has working silicon from the start of the design process. The silicon may run at speed, or perhaps nearly at-speed if numerous FPGA cores were used and result in a system slowdown compared to an ASIC design. The designer makes numerous iterations among configuring the design and executing it, thus supporting the spiral model of development.

4.4 Specialize

When using a prototype-oriented fig chip, the designer will want to eventually generate a specialized chip for use in the final product. This specialized chip is not custom, but rather a subset of the original fig chip. Selected parameters during configuration become "hard-coded" in the specialized chip. For example, the specialized chip may have a smaller cache with a particular associativity, a smaller bus with bus-invert, omitted or added cores, and programmable logic possibly replaced by custom logic. Specialization will differ depending on the relative importance of optimizing power, performance, size, cost, etc.

Because the system is extensively verified in its real environment during development, first-pass correct silicon is very likely.

5. RELATED WORK

Many of the concepts in this paper are based on the "Rapid Silicon Prototyping" concepts described by Payne [13] of VLSI Technology Inc., which has already built reference design chips for several application classes [19], and some tools for specialization, such as a cache configuration tool. Reference designs for various applications classes are beginning to appear, such as customizable networking chips.

Numerous researchers have proposed combining microprocessors and field-programmable logic; a summary of some of the work can be found in [7]. Rabaey has proposed an architecture of reconfigurable heterogenous processors connected via a communication network [15]. Fig chips differ in that they would possess a large number of parameterized cores.

Meerbergen [10] has described the need for "silicon platforms" in building chips for consumer products at Philips, in harmony with the ideas presented in this paper.

The reader may notice that the configure-and-execute paradigm actually has existed for many years in microcontroller-based system design approaches. In particular, a microcontroller is essentially an early form of SOC; it has a microprocessor plus numerous peripherals (like timers and UARTs) on chip and pre-integrated. Microcontroller providers create different microcontrollers for different application classes (e.g., TV applications, automotive, etc.), with each class having different on-chip peripherals. Providers amortize their design costs over large numbers of chips, resulting in chips available for just a few dollars. Development environments and in-circuit emulators also exist for microcontrollers, allowing step-by-step execution and internal register control/observation. Microcontroller users configure the device for a particular application instance.

6. EXAMPLE

We have developed an early version of a product-oriented reference design validating some claims of this paper. The design is captured in RTL VHDL and synthesized by Synopsys. Figure 2 shows the basic architecture, consisting of a (partial) 32-bit MIPS processor core connected to RAM and EEPROM (for program memory) via a high-speed MIPS bus. Peripheral cores are connected to a low speed ISA bus. The two buses are joined with a bridge core. The peripheral include a (partial) CODEC for data compression/decompression, a UART for serial data transmission and reception, a programmable peripheral interface controller for parallel I/O, a DMA core for direct memory access by other cores, and an 8051 core for miscellaneous software functions. The reference design would house an FPGA for custom hardware. The design also provides logic for loading configuration data, e.g., loading a MIPS program into the internal EEPROM, and shutting off inactive cores by writing the "disable" command to the control register of that core. Table 1 provides some statistics on the design. Because silicon was not generated, only 15 person-months were required to build this design. However, it is interesting to note that *integrating* the various cores occupied a substantial portion of time in building the design. Most integration problems were low-level timing mismatches in among cores resulting in corrupted bus data.

Table 1: Statistics for designing the reference design.

Lines of RTL Code	18,572
Transistor Count	888,740
Synthesis Time	12 hours
Max Clock Speed	100 MHz
Core Design Time	10 person-months
Core Integration Time	5 person-months

Given this reference design, we then set out to build a digital camera system. The digital camera requires a CCD preprocessor core, not already part of the reference design, for image-capture, thus requiring use of the FPGA. We could capture the remaining camera functionality using the existing cores, with the 8051 and DMA cores unused, as shown in Figure 2. Thus, design consisted of modifying the existing reference design software to implement the digital camera application, and of describing and synthesizing the CCD core onto the FPGA and integrating it with the other cores. Table 2 shows that the CCD core was designed in a couple weeks and required another couple weeks to integrate. The complete example is available at [22].

Table 2: Statistics for using the reference design.

CCD Lines of Code	526
CCD Gate Count (FPGA)	28,415
CCD Synthesis Time	3 hours
CCD Core Design Time	0.6 person-months
CCD Core Integration Time	0.7 person-months
System's Silicon Execution (1 frame)	.00035 sec
Silicon/FPGA Execution (1 frame)	0.0029 sec
RT-Simulation (1 frame)	299 sec
Gate-Simulation (1 frame)	28,080 sec

The CCD core synthesized for the FPGA could run at 20MHz, compared with 100 MHz for the reference design. Compared to specialized silicon, the reference design with FPGA would run between 5 and 10 times slower. As shown in the table, simulation times are many orders of magnitude slower. A co-simulator would help – our instruction-set simulation runs at 2 MHz, but remaining cores would still run at much slower rates.

In summary, configuring the reference design required only a small amount of design, enabled nearly at-speed execution of the system for verification, and eliminated much of the time-consuming integration time. As this example was a relatively small (1 million transistors) SOC, these factors would likely magnify significantly for larger SOCs.

7. LIMITATIONS AND FUTURE WORK

A key limitation of the configure-and-execute paradigm is the fact that simulation is good at catching some bugs that emulation and actual silicon are not good at catching [3]. Part of this is due to the fact that, even though we can control the chip such that we can step through it execution, we often cannot control the real environment, making a simulated environment desirable. A second limitation is that a system may not match a particular application class and thus have no reference design.

Future work is extensive, including the development of an environment to support configuration of fig chips, including exploring the numerous combinations of parameter values to find the best settings for a given application and set of constraints. We are also working on developing an object-oriented simulation model of a reference design to support early performance and power analysis, and provide a means for building virtual system prototypes. These activities form the Dalton project at UCR [22].

8. CONCLUSIONS

The current situation and trends indicate that designers will not be able to utilize potential transistors, and even an IP-based reuse paradigm is not likely to bridge the gap because of major problems related to simulation time, emulation time and expense, repeated silicon spins, and other problems. Rather than trying to enable designers to use all those transistors for their system's functionality, we propose instead to use them to reduce time-to-market, by using a "configure-and-execute" paradigm. Extensive research into the design, parameterization, configuration, and debugging of reference designs is thus needed, as well as techniques for optimization of configuration parameters for specialized silicon generation.

9. ACKNOWLEDGEMENTS

This work was supported by NSF Grant CCR-9811164 and by a Design Automation Conference Graduate Scholarship. We thank Roman Lysecky for his extra efforts on the reference design.

10. REFERENCES

[1] K. Baty, DAC preview, EE Times, June 8, 1998, Issue 1011.
[2] B. Bottoms. The Third Millennium's Test Dilemma, IEEE Design and Test, Oct-Dec 1998, pp 7-11.
[3] A. Evans et al. Functional Verification of Large ASICs. Design Automation Conference, 1998.
[4] D. Gajski, N. Dutt, A. Wu, S. Lin. High-Level Synthesis: Introduction to Chip and System Design. Kluwer Academic Publishers, 1991.
[5] A. de Geus. The IC Hits Midlife Crisis – From Silicon to Systems, EE Times, Sep 30, 1998, Issue 1028.
[6] R. Gupta and Y. Zorian. Introducing Core-Based System Design, IEEE Design & Test, Vol. 14, No. 4, Oct-Dec 1997, pp. 15-25.
[7] J. Hauser and J. Wawizynek. Garp: A MIPS Processor with a Reconfigurable Coprocessor, IEEE Symposium on Field-Programmable Custom Computing Machines, 1997.
[8] K. Kiefendorff. Transistor Budgets Go Ballistic. Microprocessor Report, Volume 12, Number 10, August 3, 1998, 14-18.
[9] J. Kumar, Prototyping the M68060 for Concurrent Verification, IEEE Design & Test, Jan-Mar 1997, pp. 34-43.
[10] J. van Meerbergen, A. Timmer, J. Leijten, F. Harmsze, M. Strik. Experiences with System Level Design for Consumer ICs, VLSI'98, pp 17-22.
[11] Midyear forecast -- CEO Perspectives, EE Times, May 27, 1998, Issue 1009.
[12] S. Ortiz. New Chips Move Networking onto Silicon. IEEE Computer, Feb 1999.
[13] B. Payne. Rapid Silicon Prototyping: Paradigm for Custom System-on-a-Chip Design, http://www.vlsi.com/velocity.
[14] J. Quigley. DAC Preview, EE Times, June 8, 1998, Issue 1011.
[15] J. Rabaey, A. Abnous, Y. Ichikawa, K. Seno, M. Wan. Heterogeneous Reconfigurable Systems, IEEE Workshop on Signal Processing Systems, 1997, pp 24-34.
[16] Van Rootselaar, Panel summaries, IEEE Design & Test, 15:2, 1998, pp 5-7.
[17] M. Sarrafzadeh. Confab calls for flexible fab processes – layout gap could repeal Moore's law, EE Times, April 13, 1998, Issue 1002.
[18] Semiconductor Industry Association Roadmap 1997, http://notes.sematech.org/ntrs/PublNTRS.nsf.
[19] Velocity product information, VLSI Technology Inc., http://www.vlsi.com/velocity.
[20] Virtual Socket Interface Association Architecture Document, http://www.vsi.org.
[21] Xilinx LogiCORE product information, Xilinx Corp., http://www.xilinx.com.
[22] The UCR Dalton project: http://www.cs.ucr.edu/~dalton.