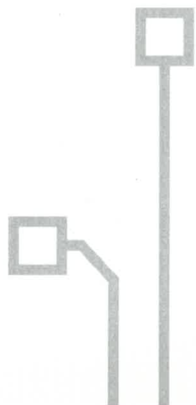
DISRUPTION IN RTL
DESIGN FOR SOCs
A NEW METHODOLOGY

DISRUPTION IN RTL
DESIGN FOR SOCs
A NEW METHODOLOGY



Disruption in RTL Design for SOCs: A New Methodology

CLAYTON CHRISTENSEN defines a "disruptive technology" as an innovation that establishes a new standard in product cost and accessibility, and gradually displaces even successful market and technology leaders. A fundamental new form of microprocessor – extensible processors – combines the dramatic productivity of complete software programmability with the exceptional performance and efficiency of optimized logic circuits. This innovation promises not just to disrupt the traditional microprocessor market, but also to change how all digital logic is designed.

Logic is a major consumer of silicon design effort. New chips are characterized by rapidly increasing logic complexity. Moore's Law scaling of silicon density makes multi-million gate designs feasible. And fierce product competition in system functionality makes these advanced silicon designs necessary. A well recognized "design gap" grows wider every year between the growth in chip complexity and productivity growth in logic design tools, shown in Figure 1.

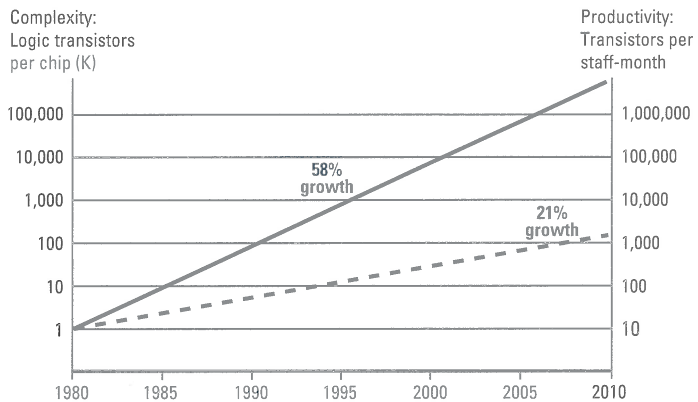


***Figure 1:** Design complexity and designer productivity*

Moreover, the market trend towards high-performance, low power systems – long battery life cell-phones, four megapixel digital cameras, fast, inexpensive color printers, high-definition digital televisions, and 3D video games – is also increasing the number of system-on-chip designs. Unless something closes the design gap, future chips will become impossible to design.

The methodology for chip logic design has not changed significantly in ten years, since the proliferation of logic synthesis for register transfer logic ("RTL") languages. The capacity of the design tools has increased, the verification speed has improved, and efforts for design reuse have yielded continuing productivity improvements. However, the challenges of the logic design problem are outstripping the available solutions. Two central problems of RTL design stand out – brutal complexity of logic verification for multi-million gate designs and the enormous time and cost penalties for design bugs. If a new methodology could remove these costs and risks, it would rapidly proliferate across system-on-chip design.

RTL is the commonly accepted way of designing SOC hardware

The conventional model of system-on-chip ("SOC") design closely follows the tradition of its predecessor: the combination of a standard microprocessor, standard memory and logic built as ASICs, shown in Figure 2. In fact, many system-on-chip designs inherit their architectures directly from earlier board-level designs. Chip-to-chip interconnect is expensive and slow, so shared buses and narrow data paths (often 32 bits wide) are typical of these board-level designs. SOC designs that are simple ports of board-level designs, often carry over these limited-width bus architectures because this backward-looking approach is the easiest way to architect an SOC. Most commonly, the processors used in these carry-overs from board-level designs are general-purpose RISC processors originally designed in the 1980s for general-purpose UNIX desktops and servers. Separate hardwired (non-programmable) ASICs implement all the logic functions that lie beyond the performance limit of the general-purpose processor.

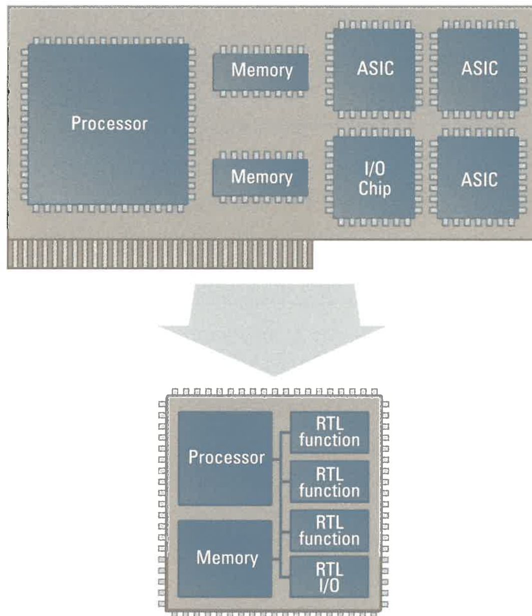


Figure 2: *Board to SOC design transition*

When all these components are combined on a single piece of silicon, clock frequency increases and power dissipation decreases. Reliability and cost often improve as well. These benefits alone can justify the investment in system-on-chip design. However, the shift to SOC integration does not automatically change the organization of the design. The architecture of these chips typically inherits the limitations imposed by the earlier board-level design. They are often organized around a single 32-bit bus because this approach saves pins – an expensive commodity in a board-level design, but much less relevant to on-chip connections. The designs often retain the rigid partitioning between a single microprocessor doing supervision tasks – running user-interfaces, real-time operating systems and high-level user tasks – and a set of logic blocks, each dedicated to a single function – data transformation, protocol processing, image manipulation or other data-intensive tasks. These architectures, derived from board-level design concepts, typically assume that communication between the logic blocks, and between logic blocks and the processor, is a bottleneck, plagued with the long latency, low clock frequency and narrow data-path width of chip-to-chip connections. Typical data bandwidth between ASIC-based logic chips and the control processor rarely reaches one hundred megabytes per second. The aggregate bandwidth among all logic functions rarely exceeds a few hundred megabytes per second.

Ironically, these bottlenecks commonly disappear in single-chip designs. While a 64-bit bus might be prohibitively expensive in board-level design, 128- and 256-bit connections, running at hundreds of megahertz, are easy to design, efficient and appropriate between adjoining blocks on an SOC. Using these wider signal paths, bandwidth between a processor and surrounding logic can exceed one gigabyte per second. Moreover, integrated circuits potentially offer much higher aggregate bandwidth. With deep sub-micron line-widths and six or more layers of metal interconnect, the theoretical cross-section bandwidth of a die 10mm on a side can approach 10^{13} bits per second (ten terabits/sec). While few practical designs will achieve this limit, it creates tremendous architecture headroom and invites a new approach to system architecture.

The traditional approach to system-on-chip design is further constrained by the origins and evolution of microprocessors. Most popular embedded microprocessors, especially 32-bit architectures, are direct descendants of desktop computer architectures of the 1980s: ARM, MIPS, 68000/ColdFire,

PowerPC, x86 and so forth. These processors were designed to serve as general-purpose application processors and were architected for implementation as standalone integrated circuits. These processors typically support only the most generic data-types (8-, 16- and 32-bit integers) and operations (integer load, store, add, shift, compare, logical-AND, etc.). This makes them well suited to a diverse mix of control-oriented applications found in computer systems. These architectures are equally good (or equally bad) at databases, spreadsheets, PC games and desktop publishing. All suffer from a common bottleneck: the need for complete generality dictates execution of an arbitrary sequence of these primitive instructions. Even the most silicon-intensive, deeply pipelined super-scalar implementation methods can rarely sustain much more than two instructions per cycle. And the harder processor designers push against this limit, the higher the cost and power per unit of useful performance extracted from the microprocessor architecture.

Compared to computer systems, embedded systems are more diverse as a group, but are individually more specialized. A digital camera may do complex image processing, but never does SQL database queries. A network switch must handle complex communications protocols at optical interconnect speeds, but doesn't need to do 3D graphics. A consumer audio device may do complex media processing, but doesn't run COBOL applications for payroll. This creates two issues for general-purpose processors in data-intensive embedded applications. First, the poor match between the critical functions of the application (e.g. image, audio, protocol processing) and a basic integer instruction set means critical applications take more computation cycles. Generic processors are routinely bigger and slower than application-specific engines. Second, these more focused embedded applications will not take full advantage of a generic processor's broad capabilities, wasting expensive silicon resources.

A large slice of embedded systems interact closely with the "real world" or communicate complex data at high rates. These data-intensive tasks could be performed by some hypothetical general-purpose microprocessor running at tremendous speed. For many tasks, however, no such processor exists today, and the fastest available processors typically cost orders of magnitude too much, and dissipate orders of magnitude too much power. Instead, designers have traditionally turned to hard-wired circuits to perform these data-intensive functions: image manipulation, protocol processing, signal compression, encryption, etc. In the past ten years,

wide availability of logic synthesis and ASIC design tools has made register transfer level (RTL) design the standard for hardware developers. RTL-based design is reasonably efficient (compared to custom transistor-level circuit design) and can effectively exploit the intrinsic parallelism of many data-intensive problems. If similar operations are required on a whole block of data, RTL design methods – specialized operations, pipelining, replicated operators in a single data-path, replicated function units – can often achieve tens or hundreds of times the performance of a general-purpose processor. Because no RTL design tries to solve an arbitrary, generic sequential problem, it avoids the generic single processor bottle-neck. The more computationally intensive the data manipulation, the greater the potential RTL speed-up.

Large SOC designs now require tremendous resources for RTL design

The evolution of silicon technology is now bringing a new crisis to system-on-chip design. To be competitive communication, consumer and computer products require rapid increases in functionality, reliability and bandwidth, and rapid declines in cost and power consumption. All of this dictates increasing use of high-integration silicon, where much of the data-intensive function must be designed in RTL. At the same time, the design productivity gap, the increase in deep sub-micron prototyping costs and the time-to-market pressure of global electronics markets, all put intense pressure on chip designers.

A few characteristics of typical deep-submicron design illustrate the challenge:

- In a generic 0.13µ standard cell foundry process, silicon density routinely exceeds 100K usable gates per mm^2. A low cost chip ($50mm^2$ of core area) can implement 5M gates of logic. Because it can be done, someone will find a way to exploit this potential in any target application.
- In the past, silicon capacity and design automation tools limited the productive size of a block of RTL to less than 100K gates. Improved synthesis and place-and-route tools are raising that ceiling. Blocks of 500K gates are within the capacity of the tools, but design methods may not be keeping up.
- The design complexity of a typical RTL block grows much more rapidly than its gate count, and the complexity of a system increases much more rapidly than the number of constituent blocks. Verification

complexity has grown disproportionately. Many real-world design efforts report that 90% of effort is now spent on block-level or system-level verification.

- The cost of a bug is going up. Much is made of the rising cost of deep-submicron masks – a full set is approaching $1M. This, however, is just the tip of the iceberg. The combination of the larger teams required by larger design, higher staff costs, bigger NRE fees and, most importantly, lost profitability and market share, makes show-stopper bugs intolerable. Methods that reduce the occurrence such show-stoppers, or permit painless work-arounds, pay for themselves rapidly.
- All embedded systems now have significant software components. Software integration is typically the last step in the development process and routinely gets blamed for overall program delays. Earlier and faster hardware/software validation is widely viewed as a critical risk-reducer for new products.
- Standard protocols are growing rapidly in complexity. The need to conserve scarce communications spectrum, plus the inventiveness of modern protocol designers, have created complex new standards (for example: IPv6 Internet Protocol packet forwarding, G.729 voice coding, JPEG2000 image compression, MPEG4 video, Rijndael AES encryption). The new standards create serious demands for both greater flexibility and computational throughput. These require new implementation methods, compared to earlier protocols in equivalent roles (for example, IPv4 packet forwarding, G.711 voice coding, JPEG, MPEG2 and DES encryption) that could be implemented using RTL design.

A consistent rule in the behavior of markets is that competitive forces drive the embrace of new technologies. In the case of electronics, Gordon Moore's prophesy, that silicon density would double roughly every 18 months, sets a grueling pace for all chip developers. This universal expectation for ever cheaper, faster transistors also invites system buyers to expect constant improvements to functionality, battery life, throughput and cost. The moment a new function is technically feasible, the race is on to deliver it. The competition is literally that intense in many markets. And in volume markets, expectations for functionality are increasingly set by the economics of silicon – functionality expands to consume the available electronics budget. Just a single CMOS process step, say from

0.18μ to 0.13μ, roughly doubles the available silicon for a given die size, and die cost. In the last five years – the period in which "system-on-chip" has become a key concept for chip designers – we've seen a roughly ten times increase in silicon capacity. Competitive pressure has pushed us to the next generation system-on-chip, characterized by dozens of functions working together, as illustrated in Figure 3.

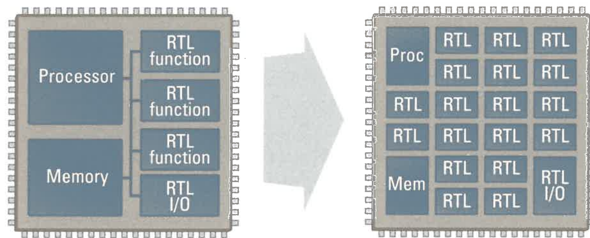


Figure 3: *SOC meets Moore's Law*

This ceaseless growth in complexity is a central dilemma for system-on-chip design. If all these logic functions could be implemented with general-purpose processor cores that were cheap and fast enough, they would be. Processor hardware is pre-designed and pre-verified, so block design based on microprocessors becomes an exercise in developing software. This approach allows developers to fix bugs literally in minutes instead of months. It also allows the addition of new features at any time in the product development, even in the field. Unfortunately, for the most computationally demanding problems, generic processor cores fall far short of the mark with respect to application throughput, cost, and power efficiency. Conversely, custom logic – especially for new complex functions or emerging standards – takes too long to design, and is too rigid. (Logic, once designed, is hard to change without incurring large verification costs.) A closer look at the make up of the typical RTL block in Figure 4 gives insight into a resolution of this paradox.

In most RTL designs, the data-path consumes the vast majority of the gates. The data-path may be as narrow as 16 or 32 bits, or as wide as hundreds of bits. It will typically contain many data registers, representing intermediate states of the computation, and will often have significant blocks of RAM, or interfaces to RAM shared with other blocks. The choice of data-path elements, and the connections among elements are

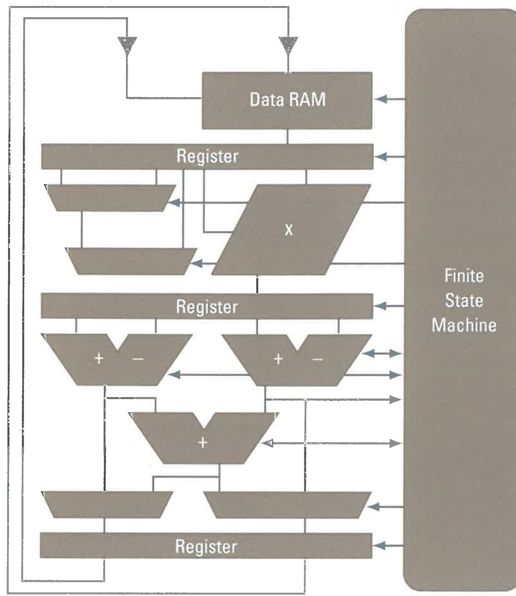


***Figure 4:** Hardwired RTL function: data-path + state machine*

often direct reflections of the fundamental data types on which the block operates. For example, a packet-processing block may directly implement a data-path that corresponds to the structure of the packet header. An image processing function may directly implement a data-path for a row or column of eight pixels from an 8x8 pixel image block. These basic data-path structures are largely independent of the finer details of the algorithm. By contrast, the finite state machine contains nothing but control details. All the nuances of the sequencing of data through the data-path, all the error conditions, all the handshakes with other blocks are captured in this subsystem. It may represent only a few percent of the gate count, but it embodies most of the design and verification risk. If a late change is made in the function of the RTL block, the change is much more likely to affect the state machine than the structure of the data-path.

One way to understand the risks associated with hardware state machines, is to examine the combinatorial complexity of verification. A state machine with N states and I inputs may have as many of N^2 "next-state" equations and each of these will be some function of the I inputs, or 2I possible input combinations. This means that at least N^2*2I input combinations

must be tried to exhaustively test all the state transitions. Typically, however, it will take many cycles to test each such transition, since it may require a number of cycles to reach the desired state, and a number of cycles to determine if the desired transition occurred. For a relatively simple state machine with 40 states and 20 inputs, truly exhaustive testing probably takes tens of billions of cycles requiring months with typical RTL simulators. Because of these exceedingly long simulation times, logic is rarely tested this completely. Even when formal verification methods are used, the basic problem remains.

RTL replacement using Xtensa is a way to get it done

Hardwired RTL design has many attractive characteristics – small area, low power, and high-throughput. However, the liabilities of RTL – difficult design, slow verification, and poor scalability to complex problems – are starting to dominate. A design methodology that retains most of the efficiency benefits of RTL, but reduces design time and risk, has a natural appeal. Application-specific processors as a replacement for complex RTL fit this need.

An application-specific processor can implement data-path operations that closely match those of RTL functions. The equivalent of the RTL data-paths are implemented using the integer pipeline of the base processor, plus additional execution units, registers and other functions added by the chip architect for a specific application. The Tensilica Instruction Extension (TIE) language – an extension of Verilog – is optimized for high-level specification of data-path functions, in the form of instruction semantics and encoding. This form of description is much more concise than RTL because it omits all sequential logic, including state machine descriptions, pipeline registers, and initialization sequences. The new processor instructions described in TIE are available to the programmer via the compiler and assembler, just like the processor's base instructions. All sequencing of operations within the data-paths is simply controlled by the program, through the existing instruction fetch, decode and execution mechanism of the processors, and are typically written in a high-level language – C and C++ – running on the extended processor.

Extended processors used as RTL-block replacements routinely use the same structures as traditional data-path-intensive RTL blocks: deep pipelines, parallel execution units, problem-specific state registers, and wide paths to local and global memories. They can sustain the same high

computation throughput and support the same low-level data interfaces as typical RTL design. The control of these data-paths, however, is very different. Cycle-by-cycle control of the data-paths is not fixed in the state transitions hardwired into the logic. Instead, the sequence of operations is explicit in the software executed by the processor. Control flow decisions are made explicitly in branches; memory references are explicit in load and store operations; sequences of computations are explicit sequences of general-purpose and application-specific computation operations. The vocabulary of the block is set by the data-path and expressed as the processor instruction set, but the overall function is determined by the program.

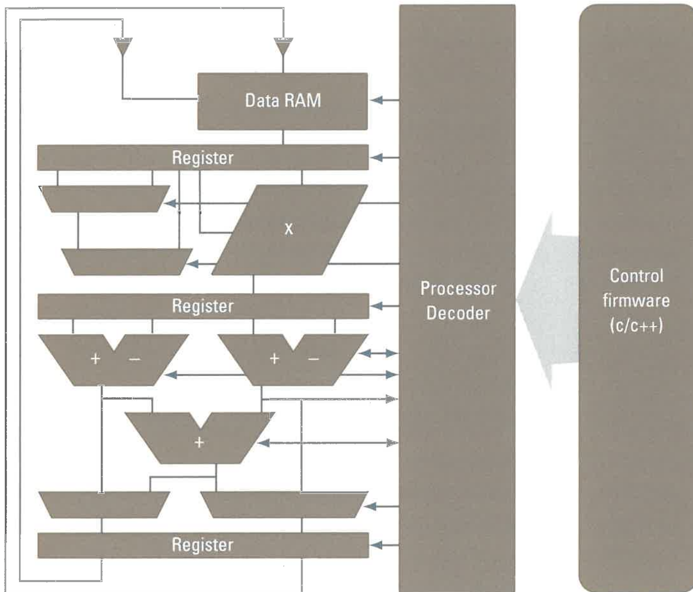


***Figure 5:** Programmable function: data-path + processor + software*

This transition from hardwired state machine to program control has several important implications:

1. **Flexibility.** Chip developers, system builders, and, when appropriate, end-users can change the block's function just by changing the program. This means new functions, bug fixes and performance upgrades can potentially be made at any time, including changes made to products already in the field.

2. **Software-based development.** Developers can use sophisticated, low-cost software development methods for implementation of most chip features. PC-based native code development and source-level debug with graphical user interfaces make development, enhancement and maintenance of functions significantly easier. Developers routinely report, for example, that software-style real-time debug tools give much higher visibility and much faster bug fixes than typical hardware simulation methods.

3. **Fast, complete system modeling.** RTL simulation is slow. For a 10 million gate design, even a modern software simulator may not exceed a few cycles per second. Only the most basic interface verification can be achieved. Simulation of realistic data sets is impossible. Even mixed-mode models suffer from the RTL simulation bottleneck. By contrast, processor simulations, including cycle- and bit-exact support for processor extensions, run at hundreds of thousands of cycles per second, per processor, so the simulation of even a complex system, implemented with large numbers of processors will typically run at least a thousand times faster than the RTL equivalent.

4. **Unification of control and data.** No modern system consists solely of hardwired logic. There is always some processor and software function, though it may be restricted to simple initialization, user-interface or error handling. Moving RTL functions into a processor removes the artificial distinction between control and data processing. This unification typically simplifies development and eliminates unnecessary interface hardware, driver software and communication bottlenecks.

5. **Time-to-market.** Moving critical functions from RTL to application-specific processors simplifies the design, accelerates system modeling and pulls in finalization of hardware. Application-specific processors easily accommodate changes to standards because details do not get cast in stone, but are instead implemented in runtime software. For the traditional RTL flow, prototypes cannot be built until every detail

of every logic function has been designed and verified. Almost any error, even in rare state transitions, mandates a new prototype run, with million dollar costs and months of delay. When the processor implements the critical data functions, the hardware prototypes can be safely built as soon as the basic data-path functions are fixed. All control functions can be implemented in software in simulator or hardware emulation during prototype fabrication, or directly on the prototype hardware as soon as it is ready.

Most importantly, perhaps, migration from RTL design to application-specific processors boosts the productivity of the engineering team. It reduces both the engineering manpower for RTL development and for verification. It sharply cuts risks of fatal logic bugs and permits graceful recovery when a bug is discovered. It accelerates the start of software bring-up and reduces the uncertainty typically surrounding hardware-software integration. It frees up low-level resources for more highly leveraged work in invention of new functions, development of improved system-level architectures and support of new customers and markets. The more complex the RTL task, the greater the impact of this advantages.

A couple of caveats, however, should be noted. For all the attractive properties of application-specific processors, they are not always the right choice. Three cases stand out:

- **Small, fixed state machines:** Some logic tasks are too trivial to warrant a processor. A state machine with just a handful of states, no data storage, no computation and little risk of bugs or changes could, of course, be implemented as short sequence of instructions. However, if the function requires less than a few dozen states, or less than a few thousand total gates, a processor would plainly be inefficient. Bit-serial engines, such as simple UARTs, fall into this category.
- **Simple data buffering:** Similarly, some logic tasks amount to no more than storage control. A FIFO controller built with a RAM and some wrapper logic, can be emulated via memory operations within a processor, but a simple FIFO is faster, simpler and commonly available in standard design libraries.
- **Very deep pipelines:** Some computation problems have so much regularity and so little state-machine control, that a single very deep pipeline is the ideal implementation. All data passes through exactly the same sequence of operations with modest control conditions or

data dependencies. The common examples – 3-D graphics and magnetic disk read channel chips – sometimes have pipelines hundreds of clock stages deep. Application-specific processors could be used to control such pipelines, but the benefits of instruction-by-instruction control would be of less obvious help in these applications. More complex or irregular computations dictate shorter pipelines so operations sequences can be more finely controlled. In contrast, application-specific processors are optimized to support pipelines of up ten or twenty stages in depth. Even in these very data-intensive cases, application-specific processors may eventually prove effective. The application-specific data-paths typically dominate the silicon area, so the overhead for processor-based implementation over state-machine-based implementation is modest. These computations are often highly parallel, so multiple processors can be fully used. In these cases, systems built around multiple processors with these shorter pipelines will often achieve nearly the same efficiency as RTL with very deep pipelines, but with greater versatility.

The migration of functions between software and hard-wired logic over time is well known. In fact, during early pre-standards exploration, processor-based implementations are common, even for simple standards that clearly allow efficient logic only implementations. Some common standards that have followed this path include popular video codecs such as MPEG2, 3G wireless protocols such as W-CDMA and encryption and security algorithms such as SSL and triple-DES. The speed of this migration, however, has been limited by the large gap in performance and design ease between software-based and RTL-based development. The emergence of application-specific processors creates a new path, quick and easy enough for development and refinement of new protocols and standards, yet efficient enough in silicon area and power to support very high volume deployment.

Conclusion

The steady, rapid improvement in silicon density is opening the door to a wide spectrum of new digital products, from optical-rate routers to multi-megapixel digital cameras. The difficulty of designing millions of gates of new logic for each system-on-chip, however, threatens to frustrate this potential. A new design methodology – configuration and programming of application-specific processors in place of register-transfer-level logic design – offers important benefits.

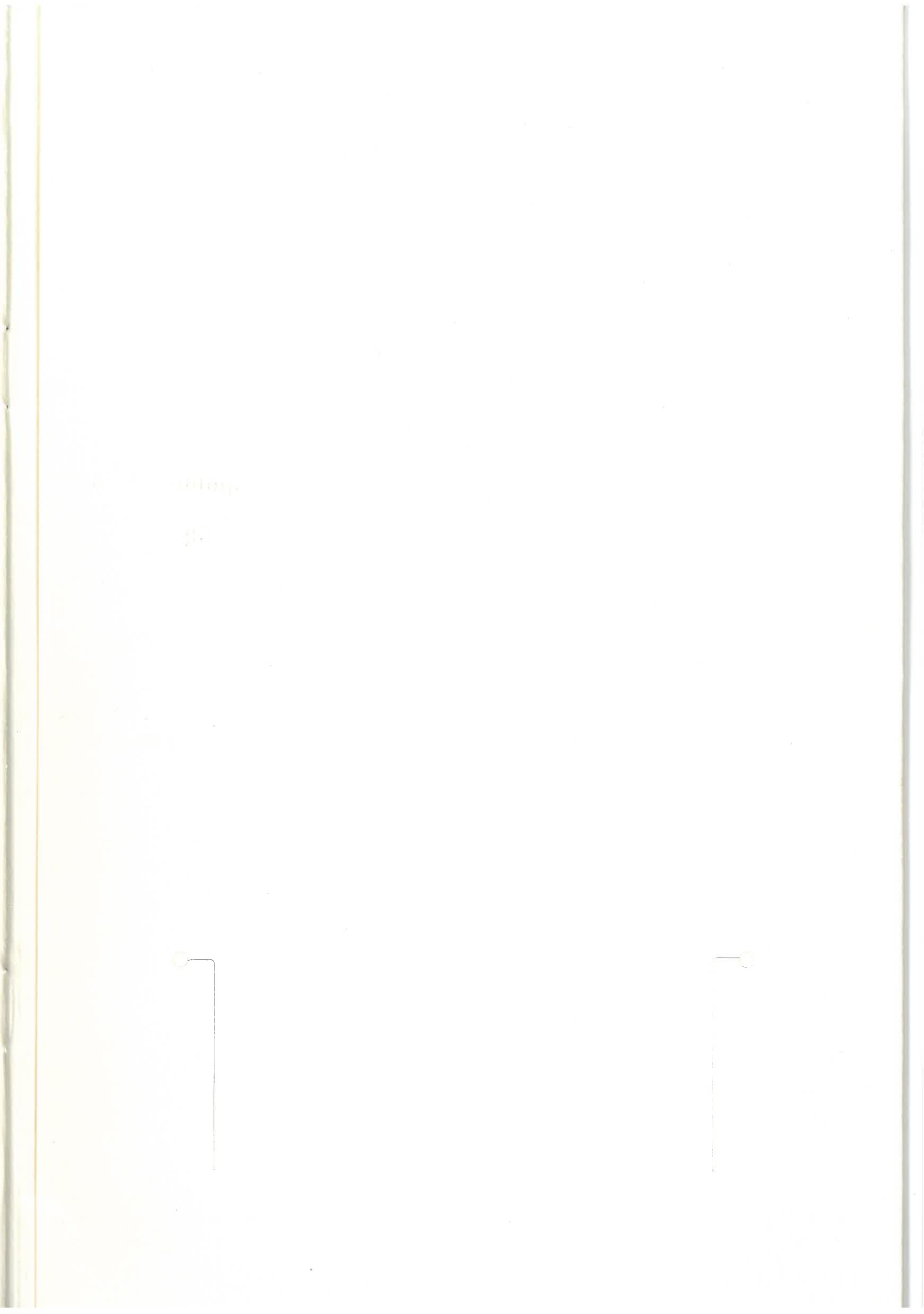
The net result is a fundamental disruption of the traditional decade-old methodology for RTL-based logic design. This new methodology allows each of the increasing number of functional blocks in a system-on-chip to be replaced by an analogous processor with similar throughput and cost, but much greater flexibility. It reduces the design time and design risk, because of simpler specification, much faster verification, and seamless functional changes at any point in the product development flow. It liberates engineers from routine and repetitive low-level RTL tasks, so they can attack new functions and higher-level system optimizations.

Another result is gradual obsolescence of the RTL-centric design approach of the 1990s. Languages like Verilog and VHDL will remain important only as intermediate data formats, used for process portability and low-level validation of design integration. Design reuse in large organizations and market transactions for silicon intellectual property can now shift from RTL representations to formal descriptions of processor configurations.

This transition even holds some promise to change the fundamental economics of silicon development. As this new methodology takes hold, as did RTL design only ten years ago, the design of very complex chips will get easier and less expensive. Pervasive programmability will also expand the number of applications and customers each design can support. This means greater silicon product longevity and better return on the development investment.

This broad use of extensible processors may also drive the fundamental disruption of the embedded microprocessor market. These very small, very fast processors will be built in such large numbers, and used by such a wide range of engineers, that the popularity of traditional processors will be seriously undermined. The lower cost, easier integration, and more-than-adequate general-purpose performance will all tend to make the extensible processor an increasingly attractive alternative. As extensible processors become broadly understood as basic building blocks, they may displace legacy processors in most general-purpose processing tasks. These processors promise to significantly simplify the whole system-on-chip design and programming challenge and to ultimately displace current leading processor architectures. Christensen's model of disruptive innovation is at work in microprocessors.

© 2002 Tensilica, Inc.



tensilica