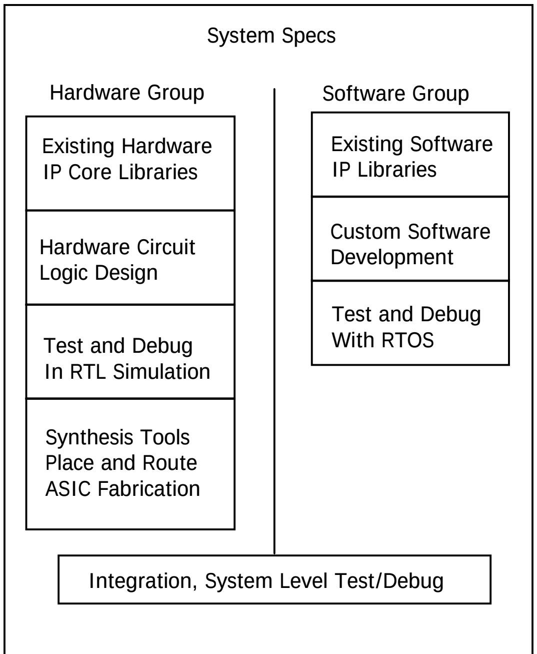
THE CHALLENGE


System Specs
Hardware Group
Software Group
Existing Hardware IP Core Libraries
Hardware Circuit Logic Design
Test and Debug In RTL Simulation
Synthesis Tools Place and Route ASIC Fabrication
Existing Software IP Libraries
Custom Software Development
Test and Debug With RTOS
Integration, System Level Test/Debug

The Hardware-Software Schism

Embedded microprocessor based approaches are the life support system of all major embedded systems. Software running on these microprocessors based provide the ability to add new functionality and avoid the costs and risk of developing dedicated hardware when software based solutions suffice.

When processors running the embedded software lack the processing speed to support CPU intensive functions, engineers have to shift CPU intensive processes to dedicated hardware function blocks.

Transforming software functions to hardware requires CPU intensive code blocks to be written in hardware-specific languages. Software programmers are not skilled in these languages, nor have the required level of understanding of hardware logic.

As a result, complex embedded system designs begin with a split of hardware and software components at the outset of the project with a different team for each. The hardware logic design generally lags software design. This affects both time-to-market and cost constraints.

To provide faster time to market and lower hardware development costs, hardware description languages continue to evolve towards higher level of descriptions, but they are still a far cry from the increasingly higher levels of abstraction software programmers want to work with in order to meet increasingly more demanding software development schedules.

ELIMINATING THE HARDWARE – SOFTWARE SCHISM

There are no easy ways for software programmers to design both “hard” and “soft” components of an embedded system in common software based high level programming languages - like Java or C, working in a standard software programming environment and without requiring deep knowledge of hardware design issues e.g. hardware programming languages.

Our soft chips technology attempts to eliminate the hardware-software schism commonly encountered in the design cycle of embedded systems. We also attempt to eliminate the need to learn hardware description languages. Patent pending technology takes software programs written in C or Java and generates low foot print executables without requiring programmers to possess any significant knowledge of embedded design.

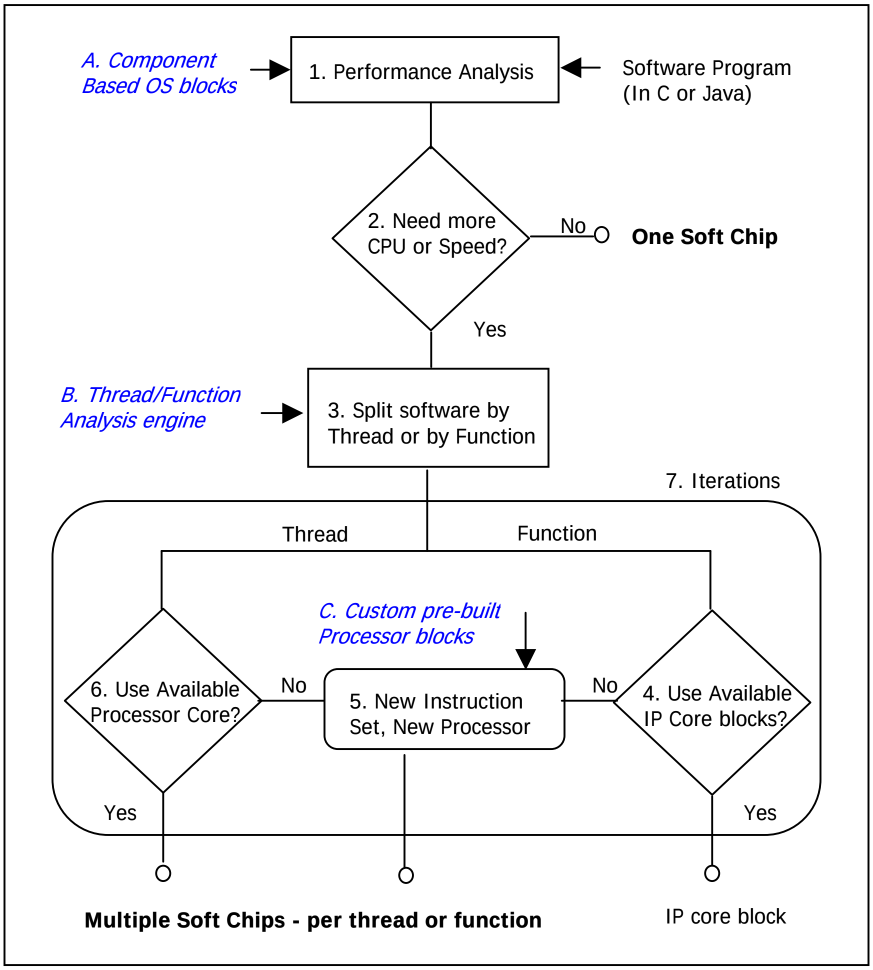
We do this analyzing software to determine the type of acceleration most appropriate for a given price performance tradeoff. Our tools then provide software-based alternatives, each with varying price/performance characteristics. For example, the software program can be automatically split to run on parallel processors or - in the event commercially available processors are not sufficient - we can suggest the best configuration of a custom processor and automatically generate one from pre-built processor blocks.

Building custom instructions from a pre-built set of instructions is significantly faster than designing code in hardware description languages because the process is automated and at a level of abstraction that software programmers are comfortable with. Time to market concerns and cost constraints increasingly favor the programmer friendly soft chips and the auto-generated custom processor approach over more complex hardware logic design tools and languages.

IDENTIFYING AND AUTOMATING PARELLELISM

A software program is compiled to produce a sequence of fetch execute cycles that call processor instructions. The instruction set is generally low level. High-level program instructions expand to 50-80 low-level processor instructions. A faster processor runs more instructions per processor but there are limitations on clock speed based on the processor design, power requirements etc. The performance of a hardware deployment is higher that running the software on microprocessors for a variety of reasons but one contributing factor is the elimination of multiple fetch cycles and the associated read-write operations for each fetch.

While at first glance, it would seem that processors couldn’t be used for significant hardware acceleration, a deeper analysis indicates that it depends on what needs to be accelerated. Consider the case of a program running 10 independent processes each of which share the same CPU resource. Parallel processing – shifting each process to another processor, with coordination between the processors - could provide a ten-fold increase in performance, in theory. This was and still is the basis for parallel processing in high performance computing applications.


A. Component Based OS blocks
1. Performance Analysis
Software Program (In C or Java)
2. Need more CPU or Speed?
No
One Soft Chip
Yes
B. Thread/Function Analysis engine
3. Split software by Thread or by Function
7. Iterations
Thread
Function
C. Custom pre-built Processor blocks
6. Use Available Processor Core?
No
5. New Instruction Set, New Processor
No
4. Use Available IP Core blocks?
Yes
Yes
Multiple Soft Chips - per thread or function
IP core block

When general-purpose processors cannot provide the performance requirements of CPU intensive processes, many types of dedicated processors have deployed to address specific applications.

For example, many types of specialized DSP processors exist and are far more cost effective than designing dedicated hardware logic. With the decreasing price and increasing performance of today's general purpose and special purpose processors it is becoming increasingly difficult to justify hand crafted dedicated hardware logic.

Our soft chips approach attempts to build on the power inherent in parallelism by run on multiple processors, both general purpose or dedicated or as a last resort, custom processors with custom instructions generated from a sequence of base processor instructions.

To address this need, ACG has developed a base processor instruction set and tools to help analyze the "best" sequence of instructions to combine into a custom instruction for a given program segment under analysis. Based on this analysis hardware automation tools can now rapidly generate both the custom processor and the custom instructions to run in programmable logic devices.

Custom processors include extensions to existing processors. For example, an ARM core licensee has access to both the instruction set and the pre-built components. With our tools, he can determine what custom instructions need to be added to a modified ARM processor core.

The software programmer, tasked with multiple options, each with different price and performance tradeoffs can now make appropriate hardware deployment decisions within a software friendly programming environment. For each selection, the programmer can run the program on the processors and determine the performance.

THREAD/FUNCTION PARALLELISM

A critic of software-only approaches will state that hardware designed logic has higher performance than software driven processes. Not necessarily so. Consider the case of a high-speed robotic system running ten threads. If two threads use most of the CPU and are slowing down the process, then moving them to hardware frees CPU resources.

If the two "heavy" threads constitute 40% each of CPU load then moving them to hardware frees the processor to run 5 times faster (100/20), in theory. In reality it would be around 4 times faster – there is still operating system overheads that have not been accelerated. That is pretty much the fastest SYSTEM level performance achievable,- even if the hardware blocks are now running 100 times faster. Dramatic hardware acceleration does not always result in equally dramatic system level performance gains.

The system performance is thus 2.5 - but at a significant reduced cost of both development and hardware cost. Using one or more dedicated processors may improve the performance and still a lower cost than hardware development. Recall that the IP core costs for 3 general-purpose processors are small when compared to the amortized cost of dedicated hardware for small to mid scale volumes. Economies of scale favor using available processor cores and as processors technology continues to improve, the price/performance ratio continues to improve also.

Now consider moving the two threads, to two general-purpose processors identical to the main processor. The main processor is still capable of running about 4 times faster but the two slave processors are slowing the system down – they cannot run any faster than 100/40 = 2.5 times faster than a fully loaded processor.

CUSTOM INSTRUCTIONS FOR CUSTOM PROCESSORS

A critic of software-only approaches will state that dedicated hardware logic will always have higher performance than ANY processor. Not necessarily so. A piece of dedicated hardware logic is not semantically different from creation a custom processor with one instruction set. Both require and employ the same hardware building blocks: registers, ALU etc. So the basic ingredients are the same. But custom instruction is being developed with processor base instructions that are at a higher level of abstraction than the more general purpose - and more primitive - building blocks supplied by an EDA vendor.

An additional advantage of staying in the world of custom processors relates to hardware semaphores. When the IP core is separate from the processor, then interactions between the processor and the IP core block need to take place over a system bus. This may or may not be significant – if the bus is local to an SoC it may not matter as much as contentions on the main system bus. But in either case, the contention can be avoided by integration of custom instructions with the processor.

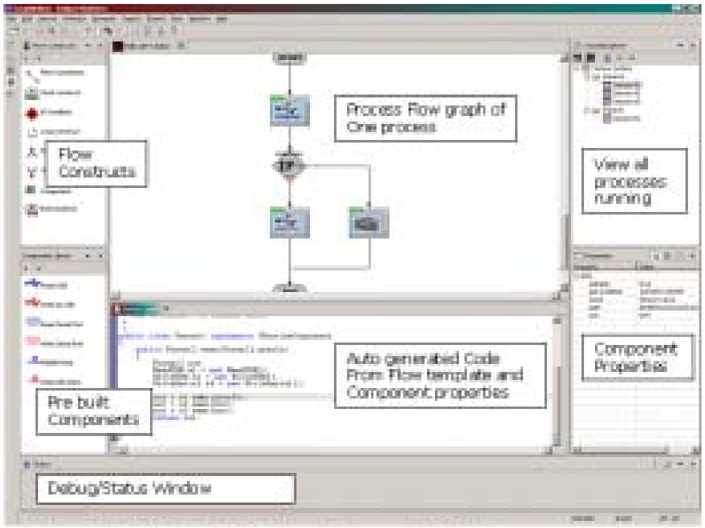
After the software code that needs acceleration has been identified, analysis tools map the code to a sequence of gate logic circuits based on the machine code sequence based on the code compiled for a target processor. Since some basic processor building blocks are needed to build upon, ACG developed a base processor instruction set for the purposes of being able to generate higher level custom instructions from them. We also developed a simulator to identify the base processor instructions most used and the custom processor instructions needed (derived from those base instructions and other hardware building blocks).

The purpose of this work is to test an hypothesis: given a set of processor instructions and other common processor building blocks, a custom processor specification may be generated swiftly through automated program analysis. We have tested this hypothesis on our target processor instruction set. However the analysis and instruction generation engine is not generic. For example, the core engine is applicable to MIPS, re-configurable core users, if they wished to build program specific custom processors with MIPS building blocks.

SOFT CHIPS DEVELOPMENT FRAMEWORK

Motivation: There are no easy ways for software programmers to design both “hard” and “soft” components of an embedded system in common software based high level programming languages - like Java or C, working in a standard software programming environment and without requiring deep knowledge of hardware design issues e.g. hardware programming languages.

Solution: Patent pending technology takes software programs written in C or Java and generates low foot print executables without requiring programmers to possess any significant knowledge of embedded design.


Process Flow graph of One process
Flow Constructs
View all processes running
Component Properties
Auto generated Code From Flow template and Component properties
Pre built Components
Debug/Status Window

- Built on top of IBM's Extensible Development Framework (ECLIPSE).
- Any process in embedded system may be modeled as a set of flow graphs
- View program flow and analyze performance down to each node in flow graph.
- Help customer analyze price/performance hardware/software tradeoffs

Features and Benefits: Distinctive features and benefits of our automated conversion to soft chips include:

Soft Chips technology: analyses program output machine code and generates a small footprint C code created, with identified essential system services fused with it. Benefits: Rapid embedded software development; Algorithms are validated in a standard programming environment; lower resource overheads for device; No OS licensing; use low cost, low power 8/16 bit processors.

Multiple Soft Chips: Further performance gains are achieved with extension to our soft chips technology. Should one CPU be insufficient, we can analyze the software to determine if program threads are separable to be able to run on multiple processors. We then generate soft chips - OS-less executables - for the threads, taking care to include thread synchronization mechanisms etc.

Software centric: Soft chips technology is thus a software centric approach to deploying software more cost effectively on one or more processors to engender the highest possible performance at the lowest cost, while still staying in a software centric paradigm. The technology is applicable to a wide variety of host processors, both general-purpose processors and custom processors.

Custom Processors: In the case of custom processors the soft chips development framework helps analyze the type of instruction sets best needed to run the program thread most efficiently. It can also help generate the minimal set of macro instructions that the custom processor would need to run this program. Based on this information, the hardware designer can now present to the customer alternatives regarding custom processor design vs. using standard commercially available processor cores.

Custom Instructions: ACG supports internally developed hardware logic for a representative instruction set. These hardware logic instructions may be combined with other instructions to create new macro instructions. Macro instructions compress the number of fetch-execute cycles to one, dramatically increasing the performance. Selecting the type of instruction sets needed and the level of granularity (e.g. combination of instructions) is driven by flexibility and performance trade-offs specific to the application.

Cost Effective Migration: Our Eclipse based framework supports deployment alternatives with easy migration from small footprint “soft” chips running on one processor to multiple soft chips running on multiple processors - that may also include higher performance custom processor packages. Benefits: Cost effective life cycle support for high performance embedded systems, working with one standard "programmer-friendly" framework.

For more information, or a live demonstration of our system, please contact: fdacosta@advancedcybernetics.com

■ FAQ ON SOFT CHIPS TECHNOLOGY

Q 01: What does this technology do?
This technology enables rapid automated conversion from high level languages to highly compact encapsulations of code we call soft chips. Soft chip technology removes the overhead of the OS and associated costs (CPU, Memory) resulting a compact low power self - standing executable. Thread and function level analysis automatically generate multiple soft chips to run on a multiple processor System-on-Chip. Further performance gains may be achieved with Custom Processor extension. Our Custom processor technology automatically generates co-processor specifications to run a thread or function of the program. The processor - along with custom instructions- is generated with a pre-built set of processor building blocks.

Q 02: How are Custom processors generated?
Programs are compiled for a special purpose virtual processor as the target processor. Output machine code is analyzed and a custom processor created to run one thread in the program. Verilog is automatically generated for the processor. The program fetch-execute flow and performance is verified on an off-the-shelf FPGA interfaced with the tool as part of an automated verification procedure. The processor is generated with a pre-built set of processor building blocks. While ACG has developed a set of base instructions and tools to generate custom processors, the core technology is applicable to re-configurable processor modules supplied by ARC, MIPS et al.

Q 03: High level languages to Verilog translators exist. Why Custom Processors?
Verilog compilers take a hardware centric approach: they produce the functional equivalent of a piece of software in a hardware circuit. Many software features cannot be supported in this approach: multi-dimensional arrays, recursion, polymorphism, dynamic objects, features related to object oriented programming e.g. invoking methods through interfaces. Conversion to direct hardware also has higher gate counts because of loop unrolling. Conversely, we retain a software centric approach to code conversion: the fetch-execute sequence is directly related to program flow: programs running in a development environment execute in the same manner on the custom processors. We never leave the software paradigm: we create a "better" processor for higher performance.

Q 04: Verification is a bottleneck in all chip design. How do you address verification?
Verification in our world is verification of the behavior of the custom processor running the program thread. We retain the Fetch-execute cycle paradigm central to software running on processors. We verify that program flow is the same for any soft chip in our system – regardless of the type of processor it runs on. A custom processor is a subset from a family of pre-written and rigorously tested components. The fetch-execute cycle of the program is also immediately tested through an interface from the development environment to FPGA test boards. The behavior of fetch-execute cycle used to run the program on a general-purpose microprocessor is compared with that of the fetch-execute cycle of the program running on the custom processor.

Q 05: Does one set of tools produce both soft chips and custom processors?
Yes. Our IBM Eclipse based IDE extensions support deployment alternatives with easy migration from license free small footprint "soft" chips to higher performance, co-processor cores. Benefits: Cost effective lifecycle support within one programming environment, supporting multiple programming languages but operating on one object code base, the compiler output.

RELATED READING

http://www.mips.com/content/Products/Cores/32-BitCores: "The MIPS32™ M4K™ core was designed for multi-CPU SOCs, which are becoming increasingly popular in next-generation consumer, networking and broadband applications. The M4K is a flexible and high-performance, yet surprisingly small and low power core that offers the highest performance density of any 32-bit synthesizable core, enabling designers to meet the high system throughput demands of multi-CPU SOCs while controlling silicon cost."

http://www.itweek.co.uk/News/1131464 "MIPS also revealed a new 32-bit core designed to optimize silicon on chip (SOC) designs implemented with multiple processor cores. The company's MK4 multiprocessing core is the first to allow developers to insert user-defined instruction logic, which extends specific instructions for proprietary header-analysis tasks. "

http://www.arc.com/products/soc/microprocessors "ARC offers a complete range of synthesizable microprocessor core solutions, ranging from the powerful 32-bit ARC family of processors, to legacy 8-bit general-purpose and Intel 8086/80186 architectures. The ARC processor can readily be tailored through easy-to-use, integrated, hardware and software tool chains to meet your stringent requirements for computational performance, digital signal processing, I/O throughput, power consumption, silicon area and cost."

http://www.eetimes.com/story/OEG20000321S0009 "In current system-on-chip (SoC) applications, chip designers are integrating several processor cores onto a single piece of silicon. This has been seen as a very important advance in the technology, especially for communications systems, where demands for speed and performance are ever increasing. Of course, the designs are never simple and require the integration of several intellectual property (IP) components, millions of transistors and assorted peripherals for each processor core. Ideally, the chip requires a system-level approach for modeling and verification to define the architecture of the SoC. It also requires specification of the components, including types of processors, number of processors, required trade-offs between hardware and software, type of bus and method for sharing memory and other peripherals."

http://www.eetimes.com/story/OEG20010323S0071 "But while some laud the ability to build a customized processor, reconfigurable processors remain a hard sell to the vast majority of embedded hardware engineers who are either unfamiliar with reconfigurable designs or uncertain about abandoning the traditional processor development model. Foremost on their minds are questions about the trade-offs of adding gates vs. code optimization and verification and test"

http://www.eedesign.com/silicon/OEG20030819S0029 "Most concurrent programming problems can be attributed to a lack of proper synchronization in the access of shared resources (e.g. CPU and bus cycles, memory, and various devices). The problems are manifested in the form of data corruptions, race conditions, deadlocks, stalls, and starvation. The occurrence of these problems is often unpredictable and hard to reproduce."

http://www.eetimes.com/story/OEG20030912S0038 "A sea change is happening in the way software developers create and debug the software for many complex systems-on-silicon, the software-rich chips where a large part of the system is implemented as software as opposed to specialized silicon blocks. Traditionally, software development happened on a hardware surrogate for the chip-to-be; now, more and more developers are transitioning to developing software on a software model of the chip-to-be. Such a model is known as a virtual platform."

http://www.eetimes.com/story/OEG20031121S0033 " Now that whole systems can be put on a single piece of silicon, IC design is as much a software design issue as it was a hardware design problem before. A typical system-on-chip (SoC) design has two or more processors, memory, one or more dedicated subsystems specific to the application and a complex, sometimes hierarchical communications system between them. As such, these chips contain multiprocessor real-time operating systems, complete with I/O drivers, utilities and diagnostic subsystems, along with their specific software applications. The exploding complexity of software in these new SoC designs has already resulted in the software design costs exceeding the hardware design costs for some of the more complex designs. The problem now is how to manage that software cost. So where can improvements be made?"

http://www.eetimes.com/in_focus/embedded_systems/OEG20020531S0026
" It should come as no surprise that the software debugger is also affected by reconfigurabilty. Although supporting subtractive changes is trivial, supporting multiple disparate processing cores is a challenge technologically (synchronizing independent processors) and economically (a small number of developer seats use a specific combination of cores). For these reasons, one of the first-generation multi-core debug architectures is based on the single-core debugger technology already available. To manage the activities of multiple single-core debuggers, a middleware mechanism or mediator needs to be put in place. For each debugger, the mediator creates the illusion that that debugger has total control over the processor it is attached to — something that was taken for granted when single core debuggers were designed. The mediator manages three tasks: execution environment initialization, launching debugger instances and defining and managing the debug topology — the start/stop relations between processing cores."

http://www.commsdesign.com/design_center/3gwireless/design_corner/OEG20020531S0029 "When the design involves a heterogeneous mix of different vendors' processors, such as ARM cores and DSP cores, the debug environment also must cope with inherent differences, like bus structures and data flow characteristics. In such environments, more complex interactions are common between the microcontroller and processor and the DSP."