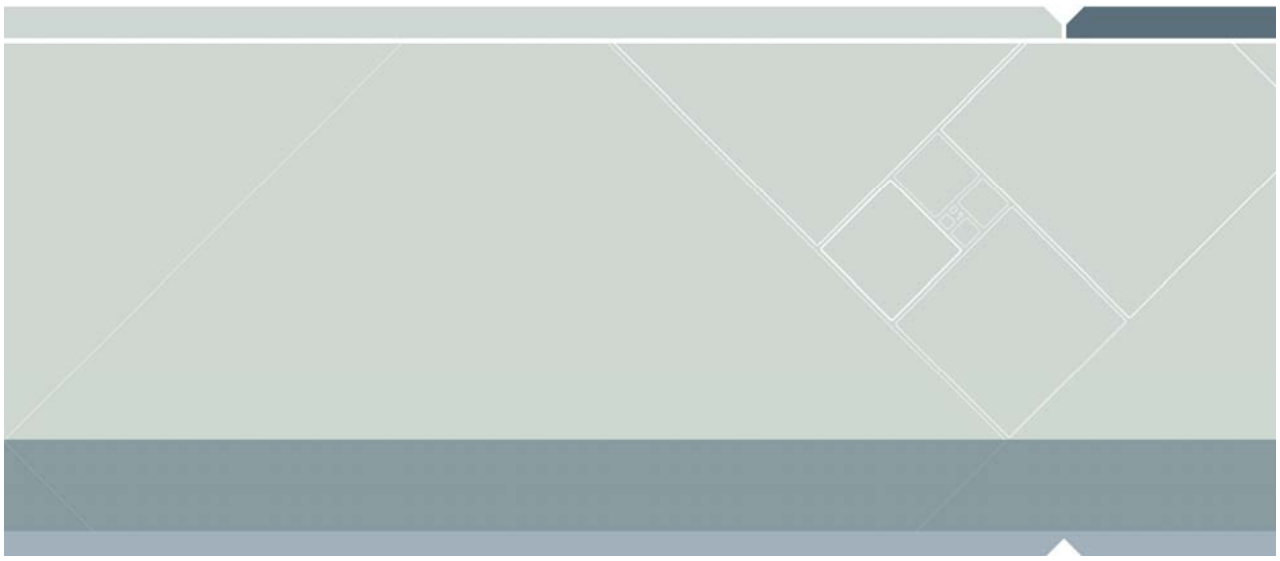


White Paper

Smart Speed™ Technology
Results of Modeling for Embedded Applications

Mike Olivarez
M.Olivarez@Freescale.com

Brian Beasley
Brian.Beasley@Freescale.com

Document Number: SMSPTECHWP
Rev 0
07/2007

freescale™ semiconductor

Overview

This paper presents the results of how Freescale used the SpecC Methodology to create architectures for the mobile device space, implementing our Smart Speed™ Technology. The solutions incorporate a generic processor architecture for code compatibility, but require additional peripherals to enable the solution's full system-level requirements. A summary of how we used the methodology to enable the system to contain the needed capabilities shows that a higher level of confidence can be introduced into a complex embedded system at higher levels of abstraction. The overview gives a short explanation of the SpecC Methodology and some examples of how we used the methodology to verify the system to meet performance, cost and market requirements, benefiting both the engineering and business aspects of creating solutions.

Contents

Reprinted with permission, this paper is a derivative of previously published work:

Olivarez, M. and Beasley, B. IFIP: International Federation for Information Processing, Volume 231, *Embedded System Design: Topics, Techniques and Trends*, eds. Rettberg, A., Zanella, M., Dömer, R., Gerstlauer, A., Rammig, F. Boston: Springer, 2007, pages 231-240. ISBN 13:978-0-387-72257-3, eISBN 13: 978-0-387-72258-0.

1 Introduction

System-level design issues have become increasingly critical as implementation technology involves more complex integrated circuits, with greater hardware integration with software. Added to this, the ever-increasing time-to-market pressures of quickly changing consumer products have shrunk market windows to sizes beyond manufacturing capabilities. A potential solution to improve the time and quality of complex systems design is to make design decisions at higher levels of abstractions. This also allows the reuse of larger design components.

The trend toward using high-level languages such as C/C++ [1, 6, 7, 8] as a basis for the next generation of modeling tools and platform methodology to encompass design reuse has come to be a great mechanism. Because there is no proven, commercially available tool suite to fully implement C to silicon, the key has been to implement the methodology using currently available means and hope that the tools industry will catch up soon. The methodology chosen, independent of any tools or language, was the SpecC Methodology. This is a well-documented method of going from the C Model of problems to creating the hardware and software needed to bring a solution to market.

This paper is limited to an overview of how Freescale used this methodology to create the architecture. The paper, which adds to the information presented in [16], covers the SpecC Methodology at a high level, breaks down some of the pieces to show how the models helped implement Freescale's wireless and mobile processors and shows how finding solutions early in the process allowed trade-offs among performance, cost and market needs.

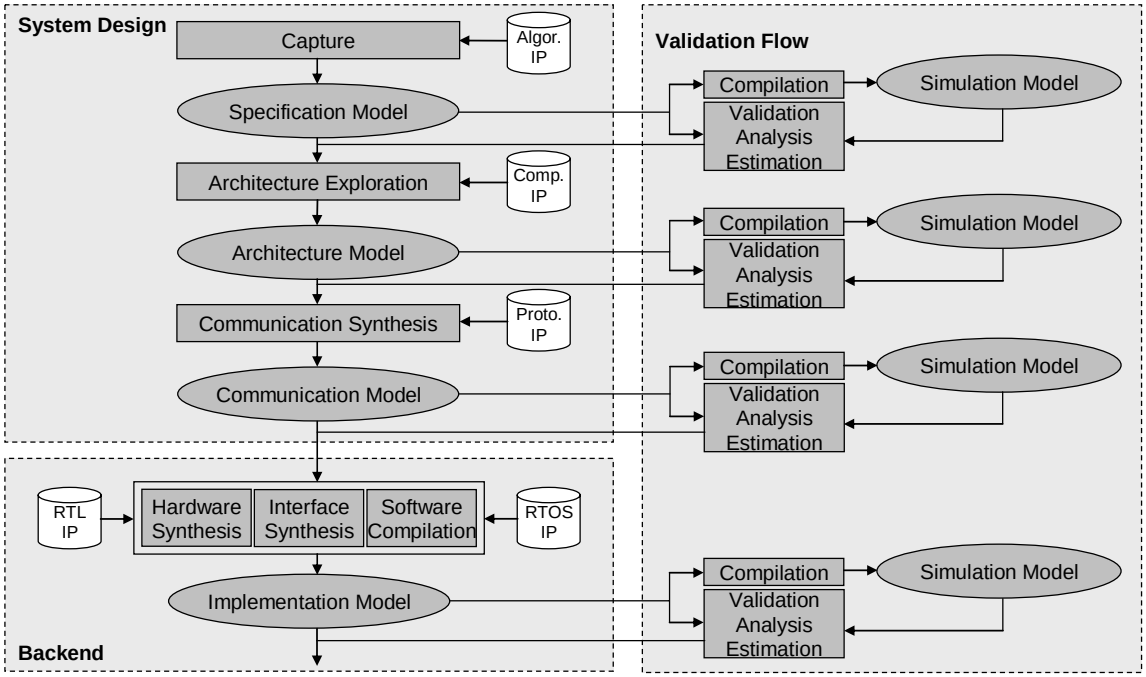
This document is organized as follows: In section 2, we explain the SpecC Methodology at a high level. Sections 3, 4 and 5 contain brief overviews of the different model types and how they impacted the system. Section 6 gives the overall benefit of using the methodology to get products to market. Conclusions are provided in section 7.

2 The SpecC Methodology

As shown in Figure 1, the SpecC Methodology has uniquely defined models for Specification, Architecture, Communication and Implementation [1, 2, 5]. These models allow the system designer to specify abstracted capabilities based on a C program of what the system needs to accomplish. Progressing through the various models reduces the amount of abstraction until one arrives at the implementation details. The Specification, Architecture and Communications models break down the system design tasks to be more manageable, yet work toward the common goal. As the system's requirements and definitions increase in complexity, it is much more important to split the tasks into parts using a method that allows them to be divided among a team of people to work on, across disciplines and geographies. As the definition of an embedded system becomes blurred—from the application-specific integrated circuit (ASIC) which performs a single task, to a fully asymmetrical parallel processing environment such as Freescale's Mobile eXtreme Convergence (MXC)[1] [11] line of cellular processors which include communications and applications processing—the ability to work more efficiently among various teams is critical to the success of the program.

[1] This document refers specifically to Freescale's MXC300-30 and MXC275-30 processors.

Figure 1.Overview of the SpecC Methodology [9].



Using well-defined models allows different team members to work independently on various tasks, yet bring the solution together successfully. The methodology's well-defined steps for simplicity allows it to function at various levels: from the specification and design of the consumer device and how the various components need to be integrated; to the chip level, how the hardware and software needs to interact and what blocks need to be defined; to the definition of a specific block and the capability it must contain. Using this top-down approach for system-level design in a systematic fashion allows smooth data interaction and helps to prioritize where resources need to be assigned to create a solution. Problems that are not on the critical path can be given a lower priority, and resources can be quickly identified and aligned using the methodology.

We can show this through examples of how the various models were applied to create sections of solutions. Though the examples are fairly small, they demonstrate how complex problems can be broken down systematically with the SpecC Methodology.

3 Specification Model

Early definition of system specifications based on requirements and use cases is critical for ensuring a system will meet its needs. As the definition of embedded systems is blurred, it becomes increasingly difficult to meet these needs, especially if the system will need to meet future requirements as well. To adjust for this complexity, it is a challenge to understand whether designing in additional capabilities will be a "value add" or simply cost-prohibitive. Using the Specification Model and well-defined use cases allows us to do a cost-benefit analysis of the system, based on how well it meets the use cases.

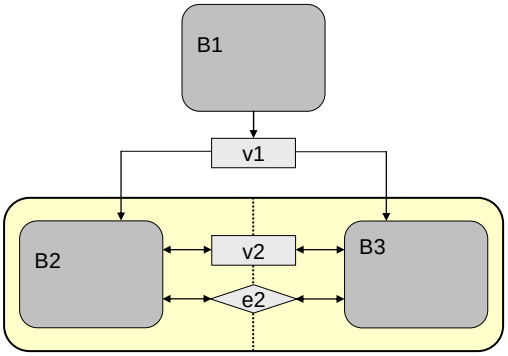
The example here is the inclusion of the floating point co-processor to the ARM1136J-S™ processor. This allows generic programs to take advantage of floating point capabilities without having to emulate them in software, thus dramatically improving the performance of algorithms used by audio, graphics and physics applications, to name a few. At the system-level definition stage, it was not known if floating point was a hard requirement. Therefore, we needed a cost analysis method to determine if the benefit would make the solution cost-prohibitive. Using the Specification Model of the SpecC Methodology allowed us to perform cost analysis. The Specification Model is a high-level model of the functionality that can test algorithmic differences without knowing the implementation details. Timing information can be

ignored while cycle approximate capabilities are used to determine the overall difference. This higher level of abstraction allows more simulations to be run quickly, to give the system designer ample data to make the trade-off analysis. An additional benefit was the ability to estimate performance capabilities of adding parallelism into the system based on the defined use cases, without incurring a large computational increase for running the models. Since the intellectual property (IP) for this case is provided by ARM, as well as simulation tools, the ARM Development Suite (ADS) was used for the basis of these tests.

The setup of the model greatly mimics the basic diagram shown in Figure 2, developed at The University of California, Irvine's (UCI) Center for Embedded Computer Systems. Behavior B1 in this case was the Color Space Conversion application used for video. Behavior B2 is generic processing for the input/output requirements. B3 represents the floating point processing of the algorithm.

There were multiple benefits of using this as a test case. One is that the mathematical algorithm is widely available [10]. The fixed point code for the function that had been previously verified is also available, so the trade-off between hardware implementation and software implementation can be easily measured. Since the code is small and primarily floating point, the results can easily be multiplied by the percentage of floating point code in a use case to determine a benefit for other use cases by knowing the tendency, without having to fully test other use cases. For instance, since the geometry processing for 3D graphics also would be primarily floating point calculations, it will mimic the benefit shown by this smaller test.

Figure 2.SpecC Methodology Specification model [9].



The results of the test are shown in Table 1 and Figure 3. An additional benefit of this kind of test that is an addition to the SpecC Methodology, is that we also can estimate power trade-offs at this point. The power can be estimated using equation 1 when the IP blocks to be used are known or can be reasonably estimated. If the necessary additional IP has not been specified, we also can determine the maximum gate count and power specifications using this method. The result is that the solution can be determined based on these factors:

1) Cost of the IP blocks and rough silicon area needed
2) Performance trade-off of the system as determined from the hardware and software
3) Power utilization of the system and impact on battery life

From Table 1, we see that the additional hardware for performing floating point calculations equates to an 825 percent performance improvement overall, greatly enhancing the floating point performance. In Figure 3, you can see the difference over software floating point; the portion shown in yellow is the percentage of compute cycles saved from the application due to the addition of hardware. Equation 1 shows the energy savings based on the cycles saved (X) and the amount of logic that is added (Y). The energy savings is independent of the technology used and additional techniques within the implementation that may save even more energy, resulting in longer battery life. From all of these factors, one can determine if the additional hardware provides a benefit that would be a "value add" to the overall system, based on the cost of adding the additional capability.

Also shown is an alternative fixed-point solution, with the results in Table 1. If it is determined that the “value add” is not a good enough benefit, software optimizations can help accomplish a similar benefit, though the floating point co-processor brings the best benefit for floating point intensive applications and dynamic range.

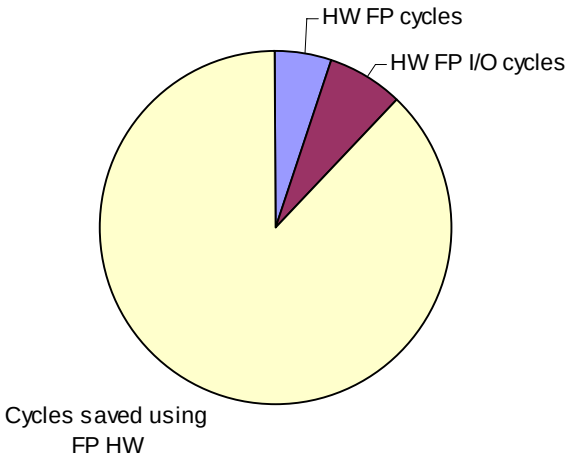
Table 1.Summary of Improvements over Software Floating Point

Improvements over Floating Point Software	Floating Point Hardware Improvement Percentage	Fixed Point Software Improvement Percentage
Application Cycles	825%	837%
Floating Point Calculation Cycles	1525%	793% [a]
Instructions Issued for Floating Point Calculations	2223%	923% [a]
Application Energy Savings	83%	88%
Floating Point Calculation Energy Savings	91%	88% [a]
Code Memory Footprint Savings	16%	16%
Arithmetic Code Development Time	15 min.	~120 min.

[a] Results are not just isolated arithmetic, but also contain I/O requirements for use of the data structures, as opposed to the calculations used for floating-point arithmetic.

Figure 3. Cycles saved by using hardware floating-point vs. software floating-point. Yellow area shows compute cycles saved versus software floating point solution.

Of course in business, everyone has an obligation to control cost, and these techniques allow system designers to do their part in the analysis. The cost for the hardware and software versions can be calculated, based on the size of the code and the size of the additional hardware for the different proposed solutions. At this time, any licensing fees can be added in and amortized over the project based on the number of units to be sold to obtain the per-unit cost. From this, the project can move forward and tasks distributed across resources, or it may be determined that the “specification” must be scrubbed. Because this is an iterative process and may benefit multiple projects, using the easily changed Specification Model makes this process much easier and more efficient over previous *ad hoc* methods.



Equation 1. Power savings, based on X as the cycle reduction multiplier and Y as the additional logic adder. [1]

$$PercentEnergySavings = (1-(\frac{1}{X}*(1+Y)))*100\%$$

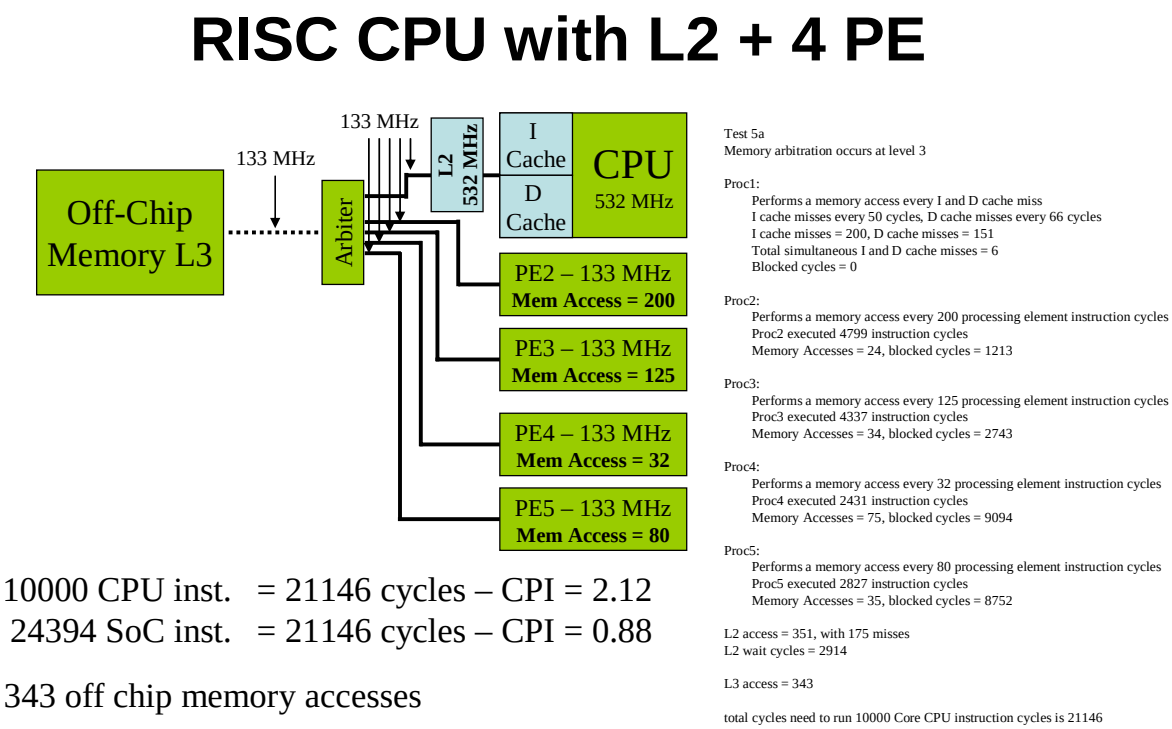
From this analysis, Freescale has added the floating point co-processor available via the ARM1136JF-S™ to enable floating-point intensive applications in its offerings such as the MXC300-30 baseband processor and i.MX31 multimedia applications processor [11, 12]. Applications such as 3D graphics, audio, gaming physics and JAVA CLDC can take advantage of the greater performance and dynamic range and quicker software development time due to the analysis performed using the Specification Model.

4 Architecture Model

Once the specification has been created and the required behaviors are being met, the next step is to decide the best way to implement. Behaviors such as whether to add a floating point unit are still a high-level trade-off, but they're made easy if the decision is whether or not to add the capability. The architecture model explores the most efficient way to implement the defined behaviors. Again, the analysis can include not only the engineering aspects of architectural exploration, but also cost, die size and time-to-market aspects.

Within the architectural exploration process of refining the specification, trade-offs of System on Chip (SoC) IP blocks can be measured for the performance impact of hardware vs. software for implementing the behavior, as well as how the various blocks are connected within the system. Various IP blocks that implement the behavior can be tested, as well as various hardware/software partitions. In many cases, reuse is most important for backward compatibility and time to market, and specific IP will be taken off-the-shelf. In other cases, new IP can be introduced with different hardware and software trade-offs. In either case, modeling the system will ensure that the behaviors are functioning so that the system specification is met [3]. Behavior performance still can be quite different in how the SoC is architected, even using the same IP blocks. The trade-off analysis used for the example in this section is the placement of the level 2 (L2) cache in the system.

Figure 4. Architecture simulation with L2 dedicated to CPU.

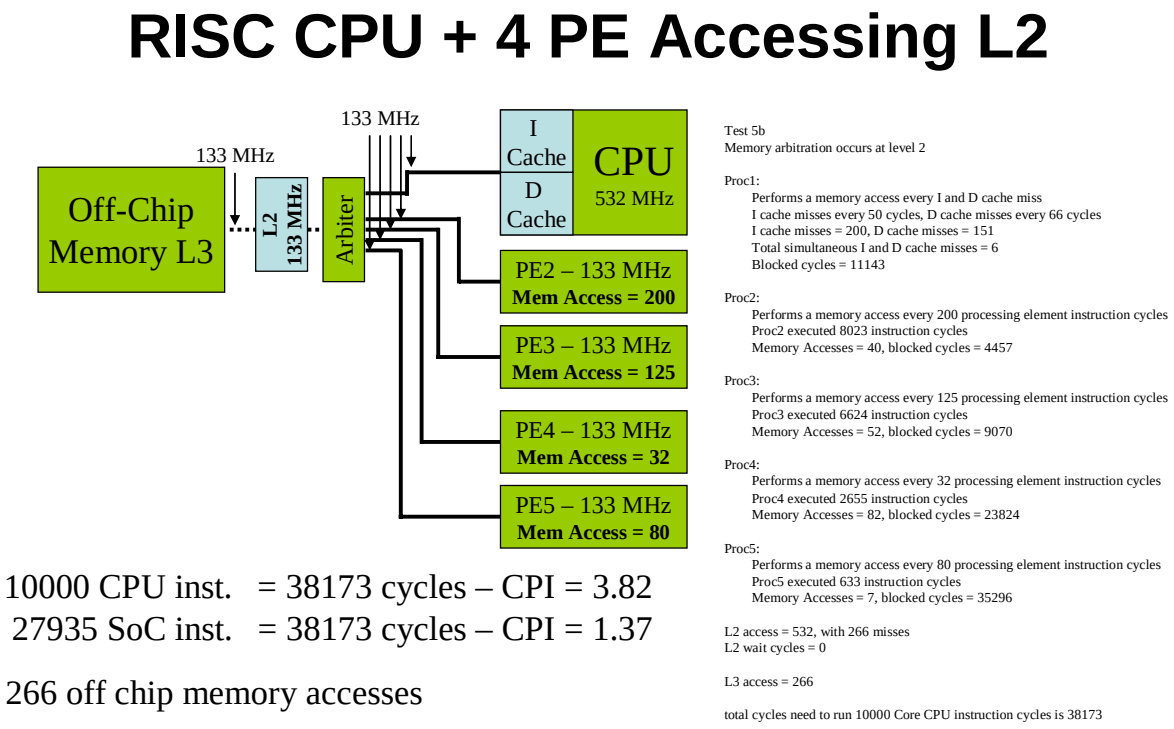


Architectural exploration was performed to see if the L2 cache subsystem should be dedicated to the ARM processor, or if it should be shared across multiple processing elements (PE) within the system. The characteristics for the memory subsystem were set up based on the times required for data to be obtained from the level 1 (L1, instruction and data, I-cache and D-cache respectively) cache of the CPU and level 2 (L2) cache accesses. The statistical model used 30

percent of the instructions as memory access (10 percent stores, 20 percent loads overall). The I-cache was set to a 98 percent hit rate, the D-cache was set to a 95 percent hit rate, while the L2 used a 50 percent hit rate. The memory arbitration scheme was done using priority arbitration, where the priority was set based on the processing element number, as shown in Figure 4 and Figure 5.

The simulation looked at the cycles per instruction (CPI) that the ARM processor should be able to achieve, as well as the theoretical CPI (tCPI) of the SoC, based on each of the PEs having different definitions of instructions. Since the two architectures are using the same types of PEs, the measurement is a good comparison of the two architectures. If the two systems were using different PEs, the effective CPI (eCPI) could be calculated based on simulations of test cases as defined in [4][2]. Because of the difference in CPU speed and its higher memory bandwidth needs compared to the other PEs used in the test, dedicating the L2 to the CPU gave the best performance, as shown by the lower CPI numbers for both the CPU and the SoC tCPI. The test was also run with a faster memory interface, which could be accomplished using double data rate memory. This improved on the numbers given in Figure 5, but still wasn't as efficient as what was accomplished using the architecture shown in Figure 4, when the L2 cache is dedicated to the ARM processor.

Figure 5. Architecture simulation with shared L2.



Using the Architecture Model, it was shown that the L2 cache was a greater benefit to the much faster processing element. A secondary result was that the number of external memory accesses was reduced by allowing all the PEs to use the on-chip memory. Lowering the overall number of off-chip memory access could save energy, but the difference was not shown to be great enough to warrant the decrease in performance for this case. In a separate case where the dominate PE was not significantly faster than the other PEs, it was shown that sharing the on-chip cache memory was more beneficial for overall performance and lowered the number of off-chip memory accesses, thus conserving

[2] tCPI based on statistical simulations and is an approximate measure of what the architecture may achieve. eCPI is based on measured test cases using actual applications.

precious system energy. Even though the PEs may not change, the subtle changes in behaviors defined through architecture exploration can significantly affect the overall system capabilities.

The Architecture Model allows quick trade-offs between different configurations, implementing the behaviors defined by the Specification Model. Quickly changing between scheduling and priorities allows the system designer to make architectural decisions, while abstracting out information such as the communication protocol and overall bus structure. This method can be used hierarchically so that the problem is more manageable and can be split between different architects and teams.

The configuration of Figure 4 and the techniques shown have been used in Freescale's MXC and i.MX applications processing solutions. In other architectures, the L2 cache has been configured in a shared method, because the processing elements within those configurations do not have as dominant an element as the ARM11 CPU in the configuration shown in Figure 4. The Architectural Model showed how making the trade-offs can produce a higher-performing system, even if the IP blocks have already been chosen, as well as the well-documented hardware/software trade-off analysis.

5 Communications Model

The Communications Model introduces less-abstracted system timing to get a true performance estimate of behaviors. The Specification and Architecture models can use only a rough estimate of timing, and didn't include any overhead that the bus structures would introduce. When including the various IP blocks of the system, one must also take into account the interface portion, because various IP vendors may not use the same bus interface for all blocks. The Communications Model enables this to be taken into account.

As the various PEs are added to the system, arbitration schemes and bus structures have a large performance impact, as well as cost and time-to-market implications. The easiest method is to connect every processing element (including the memories) to a single bus. The bus will need to arbitrate among all the elements and will only be as fast as the slowest element. The best performance would have to be a mesh network of buses, so that every processing element has a point-to-point connection with every other element at the rate of the fastest element. This latter method is too expensive and wastes many of the resources added to the system.

Freescale's MXC and i.MX products use various bus structures to maximize performance and cost. For the high-performance processing elements and the memory interface, a Smart Speed switch allows the faster processing elements to communicate on a point-to-point basis, with fast access to memory. These processing elements have a standardized bus interface definition, to ensure ease of connectivity between standard IP blocks with the highest performance. The slower peripherals use a slower, more cost-effective bus and are interfaced to the shared resource. This method allows slower devices to be efficiently integrated into the system without affecting the performance of the more timing- and bandwidth-critical components.

Since the two domains still need to be able to share resources such as memory, and allow inter-process communication, transducers were added to the system to close the gaps. The two primary standards used in this case are ARM's AMBA AHB bus protocol for high-speed communications, and the Freescale Intellectual Property Interface (IPI) structures for the less performance-critical elements.[3] Bridges were created between different bus architectures, and modeled for performance using an internal modeling tool based on SystemC [6]. A result of the modeling was the creation of the Smart Speed switch, which is an AHB-compliant crossbar that allows up to five master elements to talk to slave elements simultaneously, giving the relative ability to enable data throughput of a 665 MHz bus.

A key design parameter for the mobile space is to complete use cases efficiently without wasting resources. The best method for lowering clock speed and saving precious battery life is to use parallelism in the system, and dedicated processing elements, as was shown previously with the floating point example. Looking at the complex behaviors of the system, parallelism will need to exist not only by the various PEs working simultaneously, but also in communications,

[3] The Freescale IPI standard includes a range of bus structures for different speed peripherals. The slower speed bus is used, as it connects the non-speed critical processing elements for the system mentioned in this document.

so that the PEs are not starved. By using the Communications Model and adding the requirements for the protocol and bridging between protocols, one can determine various system bus configurations. Through such use case modeling and deciding on the PEs for the architecture, it was determined that there was sufficient high bandwidth or low-latency peripherals to warrant a switch which allows up to five masters. Such PEs can be represented by the ARM processor, L2 cache, image processing unit, multi-channel DMA unit, memory controller, high speed I/O, graphics processor and bridges to lower bandwidth/latency PEs, though all of these may not be a master. Complex use cases can be used to best define the communications requirements for such systems.

Of course, the parallelism doesn't come for free. There are costs in silicon space and energy consumption to be factored. Equation 1 can be used once again to estimate the costs over a previously used solution. The ability to quickly configure and change the model allows the communications structures to be changed to meet desired cost targets, before getting too deep into implementation details.

Figure 6.Smart Speed switch configuration.

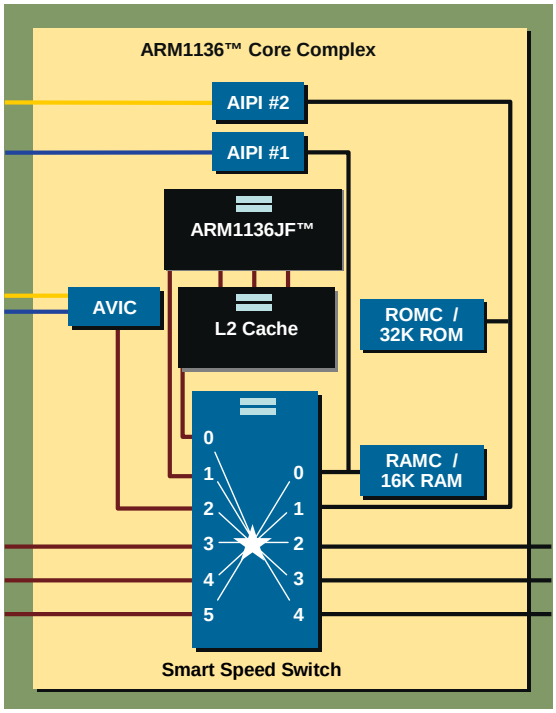


Figure 6 shows the heart of the MXC300-30 and i.MX31 systems, which allow true parallel processing by the architecture because of internal system communications. The ARM1136JF-S core can internally process items in parallel via the SIMD engine, as well as process integer and floating point calculations in parallel. The L2 cache can support simultaneous reads and writes of data, adding further system parallelism, and the multiple ports of the Smart Speed Switch allow parallel processing among the various PEs. By running complex use cases such as video telephony or multi-player online gaming through the Communications Model of the system, a configuration was created to meet the performance, cost and fluid market requirements the system will need to fill.

6 Creating Smart Speed Technology

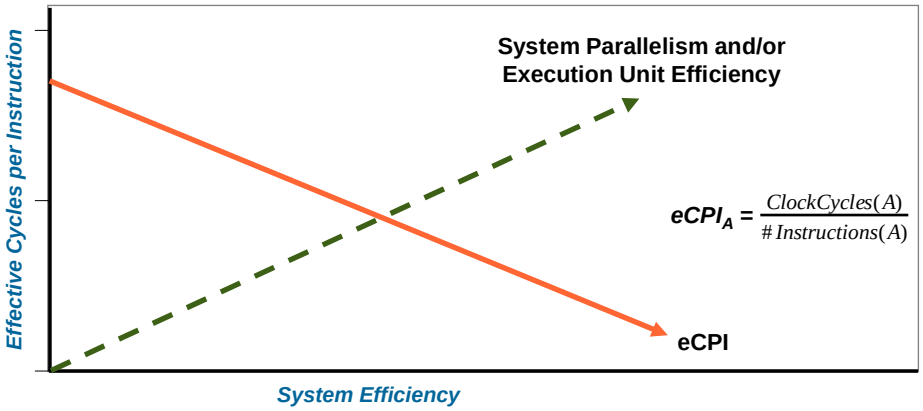
Smart Speed Technology is the brand that defines a system solution based on providing enough performance for use cases without wasting resources. Unlike the personal computing space, where processors are run as fast as possible even if the use case defines an idle state, thus wasting resources, processors for the mobile space must be able to conserve as much resources as possible without degrading the user experience. Keys to accomplish this goal are to begin with well-defined use cases that create the foundation of the solution, and to add some use cases that are good candidates for future requirements. From the use cases, the Specification Model can be built, tested and verified. As

capabilities are added that could drain the precious limited energy resource of the battery, additional capabilities to limit power consumption can be added. In the case of MXC and i.MX processors, additional power-saving capabilities such as Dynamic Voltage and Frequency Scaling and Dynamic Process Temperature Compensation have been added to conserve energy.

Process and design techniques can limit power consumption, but the ability to limit power use must begin at much higher levels of abstraction, such as what was shown in the Specification Model scenario. Additional system acceleration is a key requirement for keeping CPU speed down while maintaining the ability to perform complex use cases. The ability to make well-tested hardware/software trade-offs at high levels of abstraction leads to the ability to achieve parallelism in the system that can most efficiently perform the use cases. Using the Specification and Architecture models allows for lowering the system's eCPI and making the trade-off between silicon space for clock speed and power use, resulting in greater system efficiency [4].

Figure 7 visually summarizes the goal of using parallel execution to enhance system performance, rather than just speeding up a single execution unit. Energy can then be conserved by cutting the power to execution units that are not needed in the currently running use cases.

Figure 7. eCPI vs. System parallelism and efficiency.



Benchmarking can be run on the final solution to verify and improve the methodology. Through published data from Synchromesh Computing's benchmarking of the i.MX31 processor [13, 14], we see that the modeling of the system created accurate trade-off and specification analysis. From the floating point tests, it was found that adding the floating point unit (FPU) increased the capability around 1000 percent, where the modeling showed an 825 percent increase. Because different tests were used for the benchmarks, which had a different floating point to I/O overhead ratio than what was used for the model test, this can be considered accurate system modeling. Though the difference is 17 percent greater efficiency by the final solution over what was projected, it can be seen that the data is "Close enough for all practical purposes" [15] to make the decisions at a high level of abstraction. This affirms the use of the Specification Model as a tool for making high-level system trade-offs. Power was not practically measured to understand if there was a difference in the i.MX31 processor based on the use of the FPU alone, but when compared to a similar architecture without the FPU, the system showed a large power savings by the i.MX31 processor. Unfortunately, there are too many other variables to also take into account to understand if this is solely due to the FPU. A better way to measure power from the silicon solution and isolate smaller subsections of the chip is not available. However, we can see that the overall system goals for performance and cost that the overall price/performance targets specified were met.

Additional benchmarks based on the memory subsystem were also run, as well as the basis of overall application performance. The addition and placement of the L2 cache, as shown by the Architecture Model of the system, made a large impact. A more simplistic test is to turn the L2 cache on and off in a development system and run a case that uses the full system capabilities. For instance, a video case visually conveys that the L2 has a large impact on performance. It can also be deduced that much of the system power savings can be attributed to the L2 cache, because it lowers the access cycles to external memory and allows the system to run less to complete the same task. This minor example

gives an overview of the benefits of modeling at this level of abstraction. Additional modeling was used much more extensively in the more complex cellular baseband architecture of the MXC family of processors.

Although no specific benchmark test was used to measure it, the overall increase in system performance can be greatly attributed to bus architecture as a result of using the Communications Model. Much of the testing on larger test kernels, in addition to EEMBC testing for the i.MX31 processor, used tests that were highly dependent on bandwidth as it did computation, such as the Consumermark™ tests (26.6 Consumermark score) and STREAM Benchmark results [14]. The ability to efficiently perform these tests would allow one to deduce that the bus structures added greatly to the system efficiency as much as the processing elements added for computational capabilities.

Based on these results, we achieved Smart Speed technology capabilities with the MXC and i.MX architectures defined. The use cases were met, with ample performance to allow new use cases to be defined, increasing the lifetime of the system in the marketplace. The low-power constraints were achieved based on the tests run, but can be lowered more by using additional techniques such as Dynamic Voltage and Frequency Scaling and Dynamic Process Temperature Compensation. The current benchmark tests are unable to take advantage of these capabilities since they are targeted solely for performance. Future benchmarks, which will likely be better architected for power consumption, will have the ability to benchmark this functionality.

An additional benefit of the methodology is that it allows significant amounts of reuse of the abstraction levels, as well as the physical structures that make up the chips. This paper covers MXC and i.MX31 processors, which were concurrently designed. The core of the applications processing components were similar enough that many of the same model components and tests were used across the projects. Differences introduced from the different use cases defined by different market segments were easily adapted. This allowed for significant amounts of reuse in the modeling and physical aspects of the projects, reducing the needed resources as compared to three separate simultaneous projects, enabling them to get to market more quickly. Costs were also reduced as changes in the design phase were minimized from the previous “paper spec” method, extending Smart Speed Technology beyond the hardware and software used in the market space.

7 Conclusion

From the techniques shown and the benchmark results of the final product, we see that the modeling effort used to design the MXC and i.MX architecture allowed us to use Smart Speed Technology very effectively. The solutions can meet the use cases computationally, yet still achieve low clock speeds and energy efficiency for mobile systems. This achievement can be attributed directly to the SpecC Methodology for performance, with additional analysis for power and cost to create a successful market solution for mobile consumer devices.

8 References

[1] D. Gajski, J. Zhu et al. “SpecC: Specification Language and Design Methodology”, Kluwer Academic Publishers, 2000

[2] http://www.cecs.uci.edu/~specc

[3] L. Cai, M. Olivarez, D. Gajski, “Introduction of System Level Architecture Exploration Using the SpecC Methodology”, International Symposium on Systems and Circuits, May 2001

[4] M. Olivarez, B. Beasley, “Use Effective Cycles/Instruction as a True CPU Benchmark”, Portable Design, May 2005

[5] A. Gerstlauer, R. Dömer, D. Gajski, “Modeling and Design With SpecC”, Motorola In-house Seminar, April 2001

[6] http://www.systemc.org

[7] E.A. Lee et al., “Overview of The Ptolemy Project”, UC, Berkeley, March 2001

[8] F. Balarin et al. “Hardware-software Co-design of Embedded Systems, The POLIS Approach”, Kluwer Academic Publishers, April 1997

[9] A. Gerstlauer, R. Dömer, D. Gajski, “System Design: a Practical Guide with SpecC”, Kluwer Academic Publishers, 2001

[10] V. Bhaskaran and K. Konstantinides, “Image and Video Compression Standards: Algorithms and Architectures, Second Edition”, Kluwer Academic Publishers, 1997

[11] http://www.freescale.com/mxc

[12] http://www.freescale.com/imx

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

[14] “The Freescale Semiconductor i.MX31 Processor: Cool, Powerful”, Synchromesh Computing, 2005

[15] S. Nichols, “Laws of Engineering”, Various Lectures: University of Texas at Austin,1987-

[16] A. Rettberg, M. Zanella, R. Dömer, A. Gerstlauer, F. Rammig (Eds.), "Embedded System Design: Topics, Techniques and Trends", Springer, 2007.

How to Reach Us:

Home Page:
www.freescale.com

e-Mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
1-800-521-6274
480-768-2130
support@freescale.com

Europe, Middle East and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright license granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc.
All other product or service names are the property of their respective owners. ARM, ARM1136J-S, ARM1136JF-S and ARM Development Suite are trademarks of ARM Ltd. Consumermark is a trademark of EEMBC.
© Freescale Semiconductor, Inc. 2007.

Document Number: SMSPTECHWP
REV 0