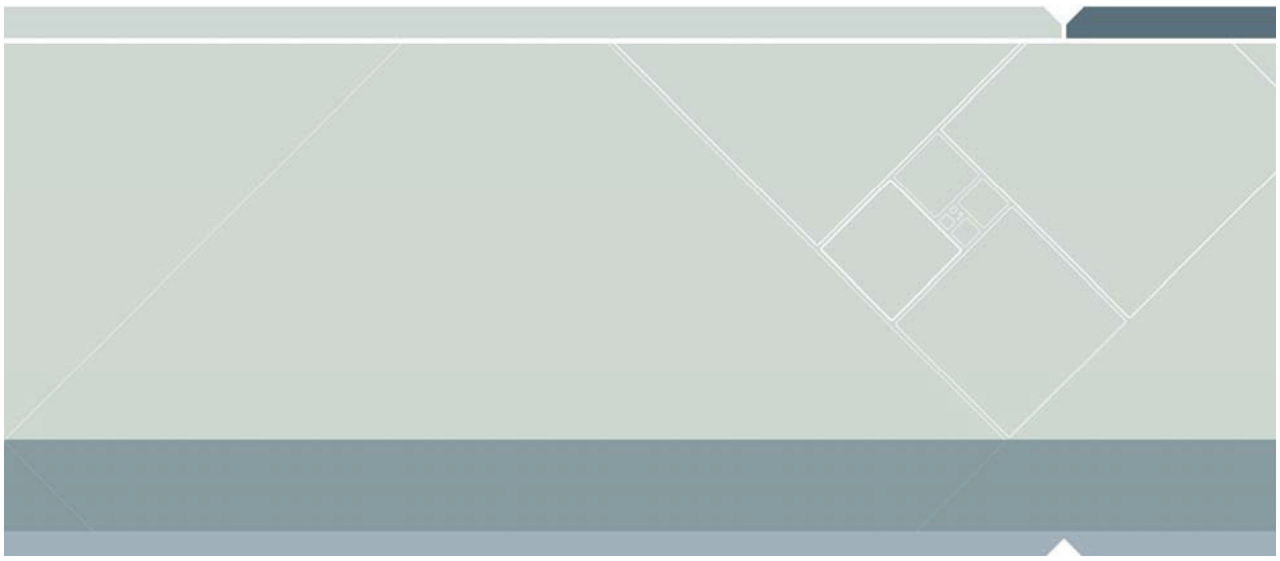


White Paper

Smart Speed Technology
Driving the Mobile Future

Document Number: SMRTSPDMFWP
Rev 1
04/2009

freescale™ semiconductor

Overview

Freescale's Smart Speed technology architecture is an intelligent integrative approach that uses hardware accelerators to offload the CPU and a Smart Speed switch to bring parallelism to the system. The 6x5 Smart Speed crossbar switch nearly eliminates wait states. This results in fewer effective cycles per instruction (eCPI) required, enabling Freescale Mobile eXtreme Convergence (MXC) cellular architecture and i.MX applications processors to drive equivalent performance to processors with higher clock speeds, but without the power consumption penalty that goes with higher operating frequencies. This paper discusses the theory and implementation of Smart Speed technology, including how the eCPI measurement can be used to compare the true processing speeds of very different processor architectures.

Contents

1 The Portable Power Problem 1

2 Device Capabilities: Power Curve Relativity 1

3 Processing Elements: Choosing an Architecture 3

3.1 General-purpose Processor Approach 3

3.2 Two-processor Approach 3

3.3 Specialized Execution Unit Approach 4

3.4 Comparing eCPI Measurements 4

3.5 eCPI Measurements in Freescale Processors 4

4 System Parallelism: Effective Communication between Execution Units 7

5 Extending Battery Life in Your Portable Design 8

6 Smart Speed Technology: Tying it All Together 9

6.1 Smart Speed Technology in a Nutshell 10

7 References 11

8 Further Reading 11

1 The Portable Power Problem

Historically, processor technology has been driven by wired devices such as the personal computer. Cranking up processor clock speeds was once adequate to support new applications and hardware, although these higher clock rates greatly increased the need for power. As mobile devices began to surge in popularity and capability, many silicon providers took the same “speeds and feeds” approach to mobile solutions, but at great expense: users could never travel too far from an alternate energy source because battery life was too short. With the rise of mobile computing, handheld devices have taken the lead in driving new processor technology. Semiconductor manufacturers’ attention has now turned to delivering better performance with minimal power drain.

As wireless handheld devices once only thought about in science fiction become reality, the capabilities of architectures—not the speed of their clocks—will be the key to increased functionality. Consider that the computing capability that required a room full of machinery less than 30 years ago can easily fit in your pocket today. Cell phones today can replace laptop computers, just as laptops are replacing desktop computers. But to move mobile capabilities forward for the next 30 years, design philosophies must change, or else our pocket-sized mobile devices will become wired once again to large packs of energy and cooling equipment.

To achieve this goal, Freescale Semiconductor decided to rethink processor architectures. From the transistor level to memory accesses, software builds and power-saving modes, innovative thinking and engineering resulted in processors with Smart Speed technology. These processors enable wireless mobile devices to deliver longer play times with the level of performance to drive power-hungry applications, such as videoconferencing and 3-D gaming.

2 Device Capabilities: Power Curve Relativity

Because batteries have only improved their capacity about five percent every two years [1], mobile devices need to become more power efficient in every product cycle. Factors that can greatly change the power curve include these system changes:

- Efficient architecture
- Efficient design
- Intelligent power management solutions
- Energy management systems

Figures 1a, 1b and 1c illustrate these necessary changes. Figure 1a shows the energy gap between the battery and capabilities based on using a traditional architecture and just increasing the clock speed, while still using the same battery. Figure 1b shows the energy gap based on the system changes for efficient energy management listed above. As these figures show, the challenge is to build a solution that is more in line with the available energy capabilities. The two to five percent energy source increase from improving battery technology slightly decreases the burden. However, Figure 1c shows that ever-faster over-the-air speeds are enabling new power-hungry features such as mobile TV and broadcast multimedia. At some point, traditional power architectures may not be able to keep up with the features that the mobile marketplace demands.

This paper will concentrate on some of the architectural and design aspects used to close the energy gap. The expectation to continually add features without making large demands on the available energy source requires a dramatic paradigm shift from “speeds and feeds” design to Smart Speed technology.

Figure 1a: Energy gap based on system, battery and features without Smart Speed technology

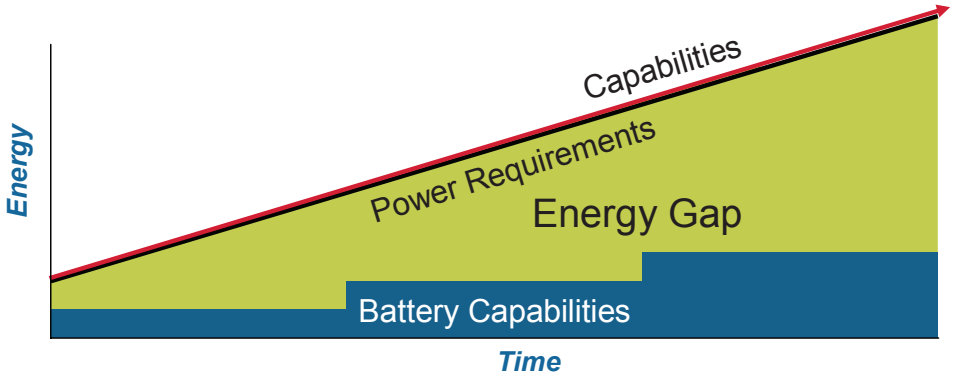


Figure 1b: Energy gap targets based on Smart Speed technology implementations

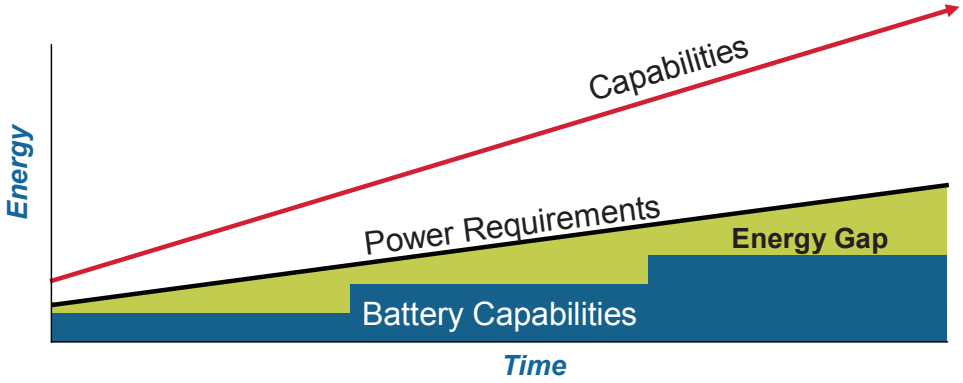
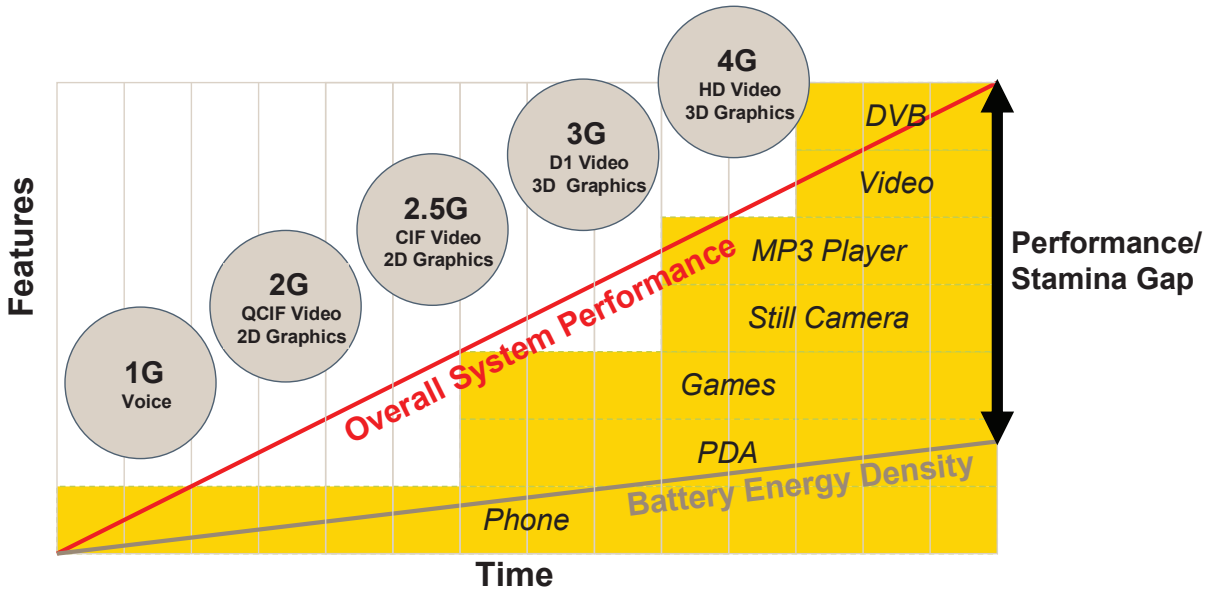


Figure 1c: Performance and energy requirements needed for mobile applications



3 Processing Elements: Choosing an Architecture

Microprocessors generally have been measured solely on clock speeds—kilohertz, megahertz and gigahertz. When processors were still fairly simple machines, this measure was close enough. Unfortunately, using clock speed as a measurement has long been outdated, and a new measurement is necessary.

If the same architecture is used between two devices, measurements such as clock speed (MHz) and instruction count (MIPS) are practical. But the diversity of portable products being designed will place many limits on those designs. As designs change with new innovations, the measurement must reflect the benefits of the innovations relative to previous offerings. The measurement of Cycles per Instruction (CPI) has been applied to measure dissimilar architectures. But even this technique falls a bit short.

This paper expands upon the proposal to add a minor modification to the CPI metric—using Effective Cycles per Instruction (eCPI) to measure the differences between architectures [9]. To show how eCPI works, we will use three different popular architecture types to demonstrate how the comparisons can be done, with results of the differences measured for some popular use cases.

3.1 General-purpose Processor Approach

Architecture A uses the philosophy that a general purpose processor is the heart of the system. To make the system faster, the clock needs to be turned up. Without going into details, as the clock speed rises, the amount of processing accomplished does not increase linearly; therefore, the eCPI does not tend to lower linearly. So, to make a processor that halves the eCPI, for example, the clock speed may need to rise by a factor of 2.5. And, to make further speed improvements, the internals may need to change dramatically to support the same instruction set architecture (ISA).

The big benefit, though, is the ease of backward compatibility. Since the same instructions are used, we only need to change application software if there are problems introduced due to timing. In many cases, the architecture can handle that. The result of running the software can produce the data for the metric, where one would need to measure the number of clock cycles used and the number of instructions needed to create the application. We can obtain the numbers using a simulator or development system, and this can lead us to the eCPI measurement of architecture A.

$$\mathrm{eCPI_A} = \frac{ClockCycles(A)}{\#Instructions(A)}$$

3.2 Two-processor Approach

Architecture B adds a second general purpose processor to the system. A general purpose processor for this purpose can consist of either a RISC- or CISC-based processor or a DSP. Since there are many types of general purpose execution units (GEU), they will require the instructions for the software to run, as well as the data, to be retrieved from memory. Some are more efficient than others, and as we mix and match we can begin to see the benefits of using eCPI.

In general, RISC machines will require more instructions to execute a job, and therefore are a good choice for the baseline architecture for determining the number of instructions it takes to perform a task. However, this is more to make the measurement graphs look nicer—the baseline can be obtained from any of the architectures. The key is to use a consistent baseline. To determine the baseline, simply run the task on the chosen baseline GEU, and record the number of clock cycles used as well as the number of instructions. In the case of architecture B, the task should be rewritten to take advantage of the two GEUs. Though this will introduce additional overhead for inter-processor communication, this should allow the task to be performed more quickly (although there are cases where this is not true). Since architecture B is now introducing some level of parallelism into the system, by taking the measurement of the number of clock cycles we can determine the relative eCPI of architecture B. The eCPI of architecture B can then be calculated to measure against architecture A.

$$\mathrm{eCPI_B} = \frac{ClockCycles(B)}{\#Instructions(A)}$$

3.3 Specialized Execution Unit Approach

In the design of architecture C, it was decided to introduce specialized execution units (SEU) in order to run the task more efficiently. Specialized execution units have the advantage of needing little to no instructions transferred from memory, and are more efficient at using memory for performing a task. They also use less power than GEUs in performing the task, thereby dramatically increasing the overall performance of the architecture. Architecture C also offers the benefits of parallelism that architecture B was able to achieve, but in most cases with less inter-processor communication. Again, as we did for architecture B, we rewrite the code to take advantage of the one or more SEUs added to the system, run the task, and record the number of clock cycles needed to perform the task. As we have done with the previous architectures, we can find the eCPI of architecture C.

$$\mathrm{eCPI_C} = \frac{ClockCycles(C)}{\#Instructions(A)}$$

3.4 Comparing eCPI Measurements

Once we have computed the eCPIs of the architectures, we make a simple comparison where the lower number is best. In general, $\mathrm{eCPI_A}$ would be closer to the 1.2 to 1.7 range. Most RISC processors for portable hardware fall into that range. Although most processors will tout that an instruction is completed each clock cycle, the eCPI will take into account the need to stall the pipeline based on requiring data from a previous instruction, and the need to wait for instructions and data to be retrieved from memory. Architecture B should allow for an eCPI closer to 1 to be achieved, though this depends on the ability to split the task among the multiple GEUs. Use caution here: if you use multiprocessor computing systems built with GEUs of the past as a benchmark, the use of more than four processors typically marked a point of diminishing returns for processing capabilities for generic operating systems, and much care is needed to ensure that the memory subsystem will be able to feed the processors. Beyond that, the inter-processor communication outweighed any gain of adding a GEU. This rule of thumb can be changed when the application and architecture is specifically tuned, but may not show a benefit in a general computing environment. This can be seen as many tasks reach diminishing returns at three processors. In a majority of cases, architecture C can achieve an eCPI of well below 1. This can be accomplished because architecture C lowers the number of memory transactions and can more efficiently handle the task being performed. Unlike the RISC or DSP architecture, it is tuned to the task and will therefore perform more of the needed computation per clock cycle.

The eCPI measures the efficiency of the architecture for the given task, but in creating a portable design, you must take the clock speed into account relative to the eCPI. If the eCPI of architecture A is 1.5, and the eCPI of architecture C is 0.25, it may be possible to use only $1/6^{th}$ the MHz rating for architecture C to equal the speed of architecture A. One still needs to take memory speed requirements into account, but in general this comparison shows how MHz and MIPS are impractical methods of measuring different processors.

A proof point of this compares architecture A with architecture C; this shows the effectiveness of the approach most quickly. From MPEG4 tests, it has been measured that the decoding of a stream on a RISC-based processing element takes about 217 million cycles to run 138 million instructions [3]. It has been further shown that a larger clip in software on the GEU takes about 2.644 seconds at 266 MHz, or 703.30 million cycles. The number of instructions would change linearly, thus requiring 447.26 million instructions.

3.5 eCPI Measurements in Freescale Processors

The same case was run on the Freescale i.MX21 applications processor integrating a Hantro MPGE4 hardware decoder, where the use case was completed in about 0.5 seconds or 133.00 million cycles [4]. The SEU was actually only running at 133 MHz in the system, and not the 266 MHz execution speed of the GEU. However, since eCPI measures the system or SoC cycles, the same measurement point in the system should be applied, and therefore the 0.5 seconds are multiplied by the 266 MHz used as the measuring point in the previous case.

Table1: eCPI measurements based on MPEG4 decode use case

	Architecture A	Architecture C
Cycles required (millions)	703.30	133.00
Instructions executed (millions)	447.26	447.26
eCPI	1.57	0.30

Equation 1: Effective MHz rating of Architecture C with SEU for MPEG4 decode

$$1.57\ (eCPI_A) / 0.30\ (eCPI_C) * 266\ MHz = 1392\ MHz\ (equivalence)$$

Using this use case, it can be shown that the i.MX21 SoC has the same capability of a 1.4 GHz GEU. Of course, the i.MX21 processor has additional SEUs that incorporate capabilities such as video pre- and post-processing and video encoding, that when used all together would rival GEUs of much higher MHz ratings. In addition, this use case does not recognize that the CPU is essentially idle, so yet more processing can be done in parallel, without an impact to the cycles required, though the number of instructions executed could increase greatly. It also shows that since the duty cycle can be decreased to 20 percent, the system can be put into low power modes to save energy. Within Freescale's Mobile eXtreme Convergence (MXC) and i.MX families of processors, many SEUs are added to decrease the eCPI of the system, for functions such as video pre- and post-processing, video processing, graphics and baseband functions. The ability to provide parallelism in the system using a combination of GEU and SEU for performance and flexibility aids these families of products to close the energy gap for the capabilities required in mobile consumer devices.

This analysis can be extended further as newer family members have superior capabilities than that of the i.MX21 processor. The i.MX31 applications processor, for instance, has the ability to encode VGA MPEG4 video, and the MXC91321 and i.MX21 processors also have the ability to encode CIF MPEG4. Using the same method as before, we develop Table 2, based on CIF encoding. It has been measured that for an encode case the CPU can perform about 311 million instructions in 421 million cycles. This is based on a QCIF video encode that would use the on-chip cache more efficiently than would be the case for a larger CIF video used in the follow-on measurement, thus skewing the eCPI of architecture A downward. Even with this change in the favor of Architecture A, we can see the benefit of the SEU as shown in Table 2 and Equation 2.

Table 2: eCPI measurements based on MPEG4 encode use case

	Architecture A	Architecture C
Cycles Required (millions)	2797.26	313.88
Instructions Executed (millions)	2066.38	2066.38
eCPI	1.35	0.15

Equation 2: Effective MHz rating of Architecture C with SEU for MPEG4 encode

$$1.35\ (eCPI_A) / 0.15\ (eCPI_C) * 266\ MHz = 2394\ MHz\ (equivalence)$$

As can be seen from this more complex use case, the benefit of an architecture like Architecture C is that its performance is equivalent to a 2.4 GHz CPU. Going to higher video encoding capabilities would further increase the difference in eCPI, thus demonstrating a much more efficient architecture for this use case. In addition, steps to increase the video quality that are optional for MPEG4 have been added to the hardware in the i.MX31 such as deblocking and deringing along with the needed color space conversion. These would further increase the difference in eCPI, resulting in a much higher equivalence rating. Using all the SEUs within each processor of the MXC and i.MX family would greatly enhance the overall equivalence ratings when used in parallel.

In addition, it can be seen that even more generic, programmable SEUs can achieve a similar benefit. For this, a test based on a floating point intensive routine run on the ARM11™ simulator was created [5]. The test used the Color

Space Conversion equations as defined in [6] to generate the needed C floating point routines. The input and output was fairly minimal to concentrate more on the floating point routines. This allows for the output to be essentially generic. Once you have a percentage of the code that would use the routine, you can get the overall benefit of the addition of the floating point unit by multiplying the percentage of the code receiving the benefit by the inverse of the percentage increase, and add back the percentage of the code which did not receive the benefit. From the example of data obtained from this test case, it can be seen that the additional floating point hardware gave an 825 percent performance improvement over floating point software. If 50 percent of the code in a test case is floating point-intensive, the benefit can be obtained by using Equation 3 and the data of Table 3.

Table 3: Summary of improvements over software floating-point [5]

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 % *
Instructions issued for floating-point calculations	2223 %	923 % *
Application energy savings	83 %	88 %
Floating point calculation energy savings	91 %	88 %*
Code memory footprint savings	16 %	16 %
Arithmetic code development time	15 min.	~120 min.

* 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.

Equation 3: Use case improvement percentage calculation

$$CasePercentCyclesUsed = (1 - PercentCodeImproved\ ) + (\frac{PercentCodeImproved}{ImprovementPercentage})$$

So, for a system that is 50 percent floating point-intensive, Equation 3 would show:

$$CasePercentCyclesUsed = (1 - 50\%) + (\frac{50\%}{1525\%})$$

$$CasePercentCyclesUsed = (50\%) + (3\%) = 53\%$$

For this use case, the number of effective cycles is held constant to compute a relative CPI of this use case, which would be 0.53. From the tests, it was shown that the test code used 24.3 million cycles to perform 19.8 million instructions. This gives the CPU-only architecture an eCPI of 1.23. The CPU with floating point was based on 2.9 million cycles for an eCPI of 0.15, which equates to the *ImprovementPercentage* of 1525 percent as shown in the table. So the use case from a 50 percent floating point-intensive algorithm would only use the 53 percent cycles, using either measurement method. This allows for the eCPI to be calculated by multiple means.

From the examples shown, we see that SEUs can be either hardwired state machines or programmable engines to achieve a benefit. Neither of the cases shown really took advantage of parallelism, but it can be deduced from the final test that if the floating point unit is used in parallel to the CPU, the 50 percent CPU usage would be the dominating factor, and could drive the *CasePercentCyclesUsed* to be 50 percent.

4 System Parallelism: Effective Communication between Execution Units

In looking at the eCPI, parallelism in the system was briefly discussed. In the previously defined architectures, there are various levels of system parallelism that can be taken into account. In the case of architecture A, the GEU will most likely be a pipelined architecture. For example, if it is a RISC processor, it may likely have fetch, decode, execute, memory and write stages. Each stage performs its portion of the task while all the other stages perform their portion. The processor can effectively perform five parallel functions, one for each stage. Other GEUs may have more pipeline stages, further increasing the parallel capabilities. Once the pipeline is filled, the processor should be able to complete an instruction each clock cycle. Of course, if the pipeline needs to be flushed, as may happen for a branch instruction, the penalty of having to refill the pipeline will occur. This same kind of parallelism can also happen within an SEU. But of course, parallelism within either the GEU or SEU is just scratching the surface.

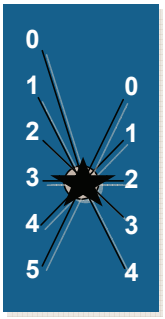
A greater benefit of parallelism is that once multiple execution units (EU)—which can be either GEUs or SEUs—are put in a system, tasks can be completed much more quickly and efficiently. Multiple pipeline stages are effective for increasing system clock speed, since less of the computation is performed within the shorter amount of time, but the best that can be hoped for is still one instruction per clock cycle. Multiple EUs takes us to the next level where it is possible to complete more than one instruction per clock cycle, and thus achieve an eCPI of less than one as shown in the eCPI calculation above. But having the execution units in the system alone is not enough to ensure that all the EUs are functioning to full potential.

In order to achieve the full potential of the various EUs, a system structure must be created to support such a system. A bus structure that only allows a single transaction at a time can cause the other EUs to sit idle in wait states, defeating the point of having multiple EUs. To gain the benefits of parallelism, the bus architecture must be built to support parallelism. There are a variety of methods to accomplish this goal, one of which is to add a crossbar switch, which creates point-to-point access between bus masters and slaves. The crossbar switch allows all of the items connected on one side (master side) to talk to all of the items connected on the other side (slave side). This allows for multiple transactions to occur simultaneously—up to a number that matches the side with fewer connections. So, if the bus speed is set to 133 MHz, and the crossbar switch supports up to five simultaneous transactions, the crossbar can achieve the effective throughput of a 665 MHz bus. This in turn allows the EUs to consume more data, thus realizing and optimizing the benefits of multiple execution units.

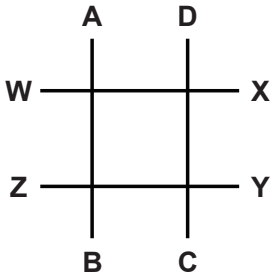
Other bus topologies can also be implemented to achieve parallelism. An easy, but in many ways less efficient method is to introduce multiple buses within the system. The architect should take great care to ensure items that need to transfer data to each other most often share the same bus. But that tends to be difficult, so to make it more efficient, the multiple buses can be configured into a mesh network. This allows for multiple paths to be created among various EUs. By necessity, this topology also adds overhead to ensure the best possible data path is chosen based on distance and load. The maximum speed of the mesh network can be calculated by adding the total number of networks that make up the mesh. This method of calculation is accurate only if the transactions are connected on the same network. It may be invalid if a perpendicular network is also used to enable data to reach its destination, since this is now using more than one network of the mesh to transmit the data. For instance, from Figure 2 b, transactions across W-A, Z-B, Y-C, and X-D will allow four simultaneous transactions, but once a connection such as A-B, C-D W-X and Y-Z is used, only two simultaneous transactions are possible. Using perpendicular networks decreases the total possible throughput and makes the routing more complex.

Figure 2: Networking topologies supporting parallelism

2a. Star topology



2b. Mesh topology



Along with the single bus, these are just building blocks that can be used within a system. The system may contain multiple instances of these depending on need and the ability to isolate the data between the various execution units. Key to greater parallelism is to make sure the EUs that exchange data most have a clear path with as few interruptions as possible.

Of course, the true potential of the bus topology depends upon limiting how many EUs are requesting the services of the same EUs or other resources. This is especially true for the resource that is the interface to the memory subsystem. Clearly, in a system that is using multiple I/O devices, a bus topology that supports parallelism greatly improves the total system throughput, having the greatest effect on lowering the overall eCPI.

5 Extending Battery Life in Your Portable Design

Unlike other systems, portable designs need to take into account a limited energy source. Battery life is critical, and if it is not being enhanced by the choice of EUs in the architecture, then adding certain EUs can be counterproductive. One of the things to look at is the overall affect on the following power equation:

$$P = cv^2 f$$

As we see from the power equation, the easiest way to lower the overall power requirements without having to change the process technology which would change the capacitance (c) and voltage (v), is to lower the frequency (f) of the system. This can also result in less heat being generated by the device, allowing the portable design to forgo heat sinks and fans. As we have seen, this can be done as we add EUs to the system and gain parallelism. As this is being done, it is necessary to find out what benefits are gained by adding the EUs with respect to the amount of energy that can be saved. The problem faced is how one can estimate the benefits with respect to energy even before the various EUs are available to put into conventional power measuring tools. This can be considered at the silicon level of the processor, or at the board level when designing a portable solution. To obtain a good relative measurement, the following equation is proposed to find the percentage of energy savings an EU can provide.

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

In this equation, X is a cycle reduction multiplier, and Y is the additional logic adder. The best way to see how this works is to give an example. If we take a GEU and want to add an SEU to the system, we can run test cases to see how much performance improvement can be obtained. By running the case in the GEU, just as we did to find the eCPI of architecture A before, we can get a baseline. We can then run the case as we did for architecture C with the SEU and find the performance improvement. Again, we can use the data from Table 3 for this use case. When this was done for a test that had heavy floating point content, it was found that the GEU alone required about eight times the number of clock cycles to process the given task than what was required when the SEU was added, making X equal to 8. Assuming that adding the SEU to the system will add a logic block about a third the size of the GEU to perform the task, Y will equal 0.33. The numbers can then be plugged into the equation.

$$PercentEnergySavings = (1 - (\frac{1}{8} * (1 + 0.33))) * 100\% = 83.38\%$$

So, by adding the SEU to the system and performing tasks that used the SEU, the overall energy saving estimate is about 83 percent, even though the overall size of the silicon was increased. After building the system, it was found that the computational savings of the SEU was closer to 11 times using the GEU, resulting in an even greater energy savings. But with the SEUs not being used all the time, and the total amount of silicon being added, care must be taken so that the benefits of multiple EUs in parallel are achieved without wasting energy. To do this, multiple power saving modes can be implemented in the system to ensure peak performance is realized, with minimal energy consumption.

Since we have a limited energy source to consume in a portable design, we can use different power saving techniques to extend the useful time of the portable design. Part of this is to use fast transistors in the critical speed path of the design, but use low-power transistors wherever possible to achieve power savings. An active well-biasing technique can be incorporated so that the transistors can achieve the best power for the performance needed. When the transistor is needed for speed, it will usually incur a larger energy leakage; otherwise, it can be tuned to save leakage energy. Active well-biasing is the technique that allows this to happen automatically, resulting in an efficient use of energy. Other techniques include creating separate power domains within the design, so that the areas that are not being used at the current time may be shut off. For example, when an SEU is added to the system, it may not be used all the time. When not in use, the power to that portion may be shut down in order to achieve the desired energy savings.

Another option is to use less power to a subsystem. For example, since the GEU is not needed as much because of the abilities of the SEUs that have been added to the system, the voltage and frequency of the GEU may be lowered, thus requiring much less power. Of course, since the SEUs allow the system to complete the job more quickly and efficiently, the processor may not be needed all the time, so the entire processor may be powered down.

These techniques have been implemented in various MXC and i.MX processors from Freescale. In [2,7], more data can be found on the accompanying power management solutions that make up the full solution for energy-efficient solutions for mobile devices. This includes the hardware to provide the power and switch between the voltages needed for the various power states, including powering down the various domains and switching the frequency as needed. In addition, Freescale's eXtreme Energy Conservation [8] software enhances the overall efficiency of the system by adding configuration and control of the energy management.

6 Smart Speed Technology: Tying it All Together

Now that the groundwork has been laid, we see what Smart Speed technology means. Instead of the old way of thinking, of just building a processor with a faster clock to get speed, the speed is now determined by the set of tasks to be performed. From the set of tasks, it can be determined which EUs are needed to make the system work more efficiently. One principle of Smart Speed technology is to have a system that works smarter, not harder. By using efficient (and possibly) multiple EUs in the system, we can achieve this principle. But even efficient EUs aren't enough if they are in a wait state.

The second principle of Smart Speed technology is to have the various EUs working in parallel. As we saw before, pipelining is one method of parallelism. When using multiple EUs to perform a task, another level of parallelism is invoked to make sure the various portions of the task are broken down. This is to ensure that the EUs can be used in a parallel fashion, instead of them doing a task and shutting down while the next element is performing the next step of the task. Otherwise, we are defeating the purpose of having the multiple EUs. Another key aspect is being able to feed the multiple EUs in the system with an efficient bus structure. The common shared bus is not efficient, because multiple EUs are trying to send the data between themselves. As we have seen, other bus structures such as the crossbar or mesh allow for true system parallelism to happen.

The third principle of Smart Speed technology introduced here is efficient energy use. As we have seen, we can easily show the performance gain in the system by measuring the eCPI of the solution, but in addition, we can estimate the power savings that can be gained by adding the EUs. We can now make tradeoffs between power and performance at early stages of system design to maximize the resources that the portable design can use. Once resources are maximized we can apply other power saving techniques such as power gating, dynamic voltage and frequency scaling,

and active well-biasing to ensure efficient utilization of the limited energy source. This results in “performance with stamina”—higher performance with longer usage time.

6.1 Smart Speed Technology in a Nutshell

The paradigm has forever shifted from running the clock faster to achieve performance. “Performance with stamina” is now the mantra. Smart Speed technology is driving the industry toward processors that optimize mobile device performance and maximize battery life.

Smart Speed can be summarized in these three principles:

- Work smarter, not harder.
- Use parallelism instead of brute force.
- Use the limited energy source efficiently.

By using these simple principles, Smart Speed technology can enable portable devices to run longer, retain smaller form factors and support more innovative applications without substantial increases in battery power. A shift in thought away from raw power and towards intelligent use of resources is critical to support the current and coming generations of small, smart wireless devices.

7 References

[1] C. Chakrabarti, “Low Power System Design: A High-Level Perspective”, Department of Electrical Engineering, Arizona State University, 2006.

[2] C. Chun, A. Barth, “eXtreme Energy Conservation for Mobile Communications: The Key to Power Management, AM105”, America’s Freescale Technology Forum, 2006.

[3] C. Chu, P. Kritzinger, “MPEG4 Decoder: ARM9 Benchmark Results and Analysis”, October 2001.

[4] “The Freescale Semiconductor i.MX21 Processor”, Synchromesh Computing, October 2004.

[5] M. Olivarez, “Benefits of Adding a Floating Point Co-Processor to the ARM11™ Platform Complex”, November 2002.

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

[7] M. Olivarez, “Improved Scalable Power-Management Solutions”, Spring Processor Forum, May 2006.

[8] “eXtreme Energy conservation: Advanced Power-Saving Software for Wireless Devices”, Freescale Semiconductor, Inc., February 2006.

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

8 Further Reading

The authors suggest the following books for further reading.

System Design: A Practical Guide with SpecC, by Gerstlauer, Dömer, Peng, and Gajski, available from Kluwer Academic Publishers.

SpecC: Specification Language and Methodology, by Gajski, Zhu, Dömer, Gerstlauer, and Zhao, available from Kluwer Academic Publishers.

Computer Organization and Design: The Hardware/Software Interface, by Patterson and Hennessy, available from Morgan Kaufmann Publishers, Inc.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +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 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: SMRTSPDMFWP
REV 1

freescale semiconductor