Quantitative Embedded System Architecture and Performance Analysis

Graham R. Hellestrand

VaST Systems Technology Corporation, Sunnyvale, CA 94070, USA

Synopsis - The Architecting of Systems

In the context of this chapter, a system is software dominated electronic engine for control, computation and communication, capable of interacting with the real world to capture data, transform it, and recommunicate it to the world. In its simplest form, the digital subsystem responsible for interpreting software, implementing algorithms and sampling and conditioning information for external communication can be represented as a Turing machine – essentially a collection of intercommunicating finite state machine s (FSMs). In its more pragmatic form it is composed of (i) a skeleton constituted from the weavings that make up its communication and infrastructure fabric, (ii) devices attached to the fabric that are structured to support the intent and effective operation of the engine, and (iii) transducers that support the interactions required between the engine and the real-world. Of course, the latter definition is more general than that obliged to describe a digital system.

Now, the architecting of the system is the structuring the skeleton, assemblages of devices, and input/output transducers to provide an engine capable of meeting its business and technical requirements. Relatively simple systems can be intuitively architected – rarely to produce an optimal system, but an acceptable one. Complex systems cannot be intuitively architected. The modern electronic engineering landscape is populated with engines that are over-engineered, unduly expensive, and plain incompetent. The architecting of optimal systems is an empirically driven, science-based discipline where hypothesis (desirable insights) about the system is set-up to be refuted and decisions are driven purely by data.

This chapter is about the quantitative architecting of complex, software dominated systems.

Critical Reactive Embedded Systems (CRES)

Within the phylum of systems, embedded systems form a large class, and critically-reactive embedded systems form a small sub-class of embedded systems.

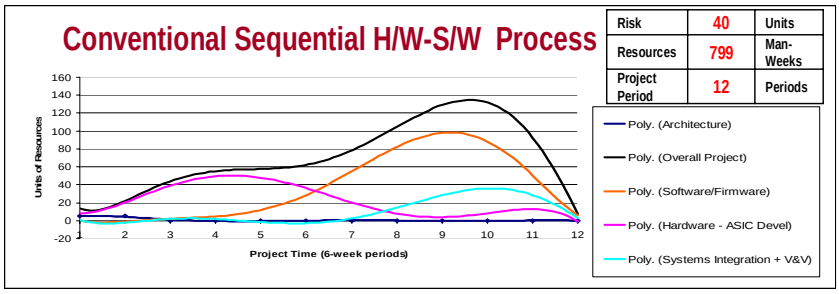
Embedded systems are control systems embedded in products and devices that are implemented using software and hardware. Apart from network connectivity and local memory, embedded systems are usually stand-alone and have prescribed interfaces for user interaction and programming - like a cell phone, camera or a home printer. About 95% of all processors produced annually are embedded in these systems. Embedded systems may be simple – a simple bus fabric connecting a single processor, memory and peripheral devices, some of which are connected, via transducers, to subsystems in the real world that collect data from and provide control capability to; others are complex as will be seen below in examples of wireless and automotive control systems.

Critically reactive embedded systems constitute that small subset of embedded control systems that have incontrovertible timing requirements that must be met. This is a severe constraint that has a far reaching impact on architecture, design, algorithm selection, development and implementation.

Architecture addresses the structure and function (behaviour and timing) of software and hardware (digital, analog and I/O transducers) of the control systems, and the mechanical, RF (radio frequency) and other devices with which the control system interacts.

The Evolving Embedded Design Process

Modern embedded systems are largely software dominated in function, complexity, engineering effort, and support. This was not always the case, until about 5 years ago, the design of the precursors of today's embedded systems are hardware dominated system in which architecture was driven as part of hardware design. The design process was sequential – hardware first, software following – and, as software began to dominate in complexity and (exponentially) in size, even while hardware increased in complexity (with a lesser index), the development times grew to unsupportable levels. The graph below shows the resource deployment (human & capital) in projects, in this case a 2.5G mobile phone, towards the end of life of the sequential design process – a process that had proven efficacious for hardware dominated designs [1].



There are several indices of the portending doom of this process: (i) the peak resource deployment occurring in the last quarter of the predicted design process; (ii) the software engineering requirement dominating the critical project path compounded by the strictures of the sequential design process; (iii) the decreasing time to market opportunity for globally competitive products; and (iv) market capacity to absorb technology driven products that command premium pricing with short term inelastic demand. Factors i through iii are captured as Risk, expressed in the following equation: $Risk = \sqrt{\frac{ResourceVariance}{\left(\frac{RemainingProjectTime_{@PeakResourceDeployment}}{TotalProjectTime}\right)^2}}$

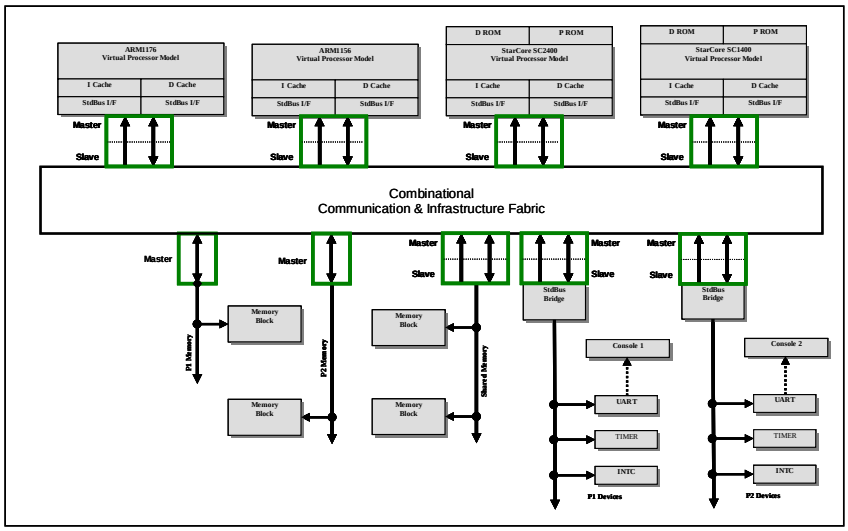
The formulation of the risk equation is motivated from the Black-Scholes option pricing equation (price of options is proportion if variance in underlying asset value), modified to reflect the critical nature of the timing of peak resource deployment in a project). The square root is a normalizing function. Although this formulation of risk is interesting, it is one such formulation. A standard measure of project risk is yet to be determined.

CRES in Wireless and Automotive

Two significant and diverse architectural examples of CRESystems are described below. The first is a controller for a hypothetical cell phone perhaps supporting multiple communications stacks simultaneously; the second is a distributed controller supporting feedback between some major subsystems in a car – this example is not hypothetical but it has been somewhat abstracted.

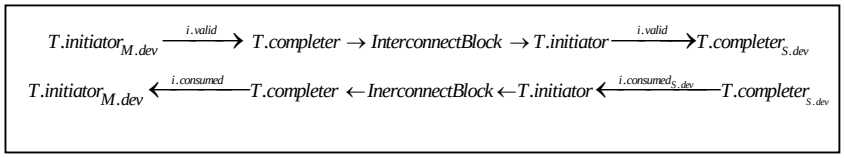
Closely-coupled multiprocessor wireless platform. This system is complex and contemporary and is implemented on a single piece of silicon. It supports two general purpose processors (an ARM1156 - perhaps for real-time control and games, and an ARM1176 - perhaps for applications processing) and two DSP processors (a StarCore SC2400 - 3-4G modem stack implementation, and an SC1400 – perhaps for a second simultaneously supported modem stack or video and/or audio decode). To support high bandwidth, low latency communication between masters devices which are processors and slave devices, such as memories is a primary requirement of such architectures. This is a major distinction between this architecture and that of the following example.

The processors (master devices) need to be connected to slave devices



(such as memory, timer, etc.) with support for various communication bandwidth and latency configurations. The requirement to connect multiple master devices to multiple slave devices gives rise to a general interconnection schema – the most known being a $m \times n$ combinational interconnect, such as cross-bar. A number of general interconnect schema supporting combinations of bandwidth and latency has been implemented, including ARM's AMBA AXI [2]. Since this is a general and critical interconnecting capability in silicon, we will briefly discuss its implementation. Firstly, where a number of metal layers are available, there are a surprising number of simple geometrical configurations of, even very wide, $m \times n$ interconnects that avoid capacitive and inductive coupling and are space economical. That is separate bundles of wires with no protocol each

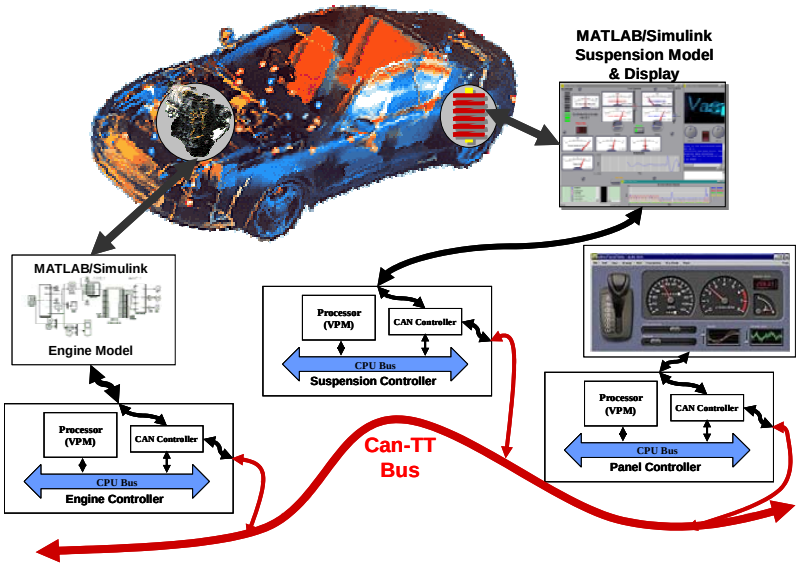
connect one (of m) ports on the interconnect block to all n target ports. Registers, multiplexers and arbitrations units can be attached to ports to effect various information transfer control strategies through the interconnect block. Secondly, to attach master and slave devices, through the interconnect block, requires a protocol for high performance (high bandwidth, low latency), point-to-point communication. On silicon, 2/4-cycle signaling elements are simple and provide the minimal control required, via a pair of cause-effect signals - such as *information valid (i.valid)* and *information consumed (i.consumed*), to implement reliable, high performance information transfer. In 4-cycle (asynchronous) signaling, the *i.valid* signal is asserted by a *transfer initiator* (producer) function (*T.initiator*), which services a master device wanting to transfer stable data to a slave device, and is transmitted to a *transfer completer* (consumer) function (*T.completer*), which services a slave device wanting to receive data from the master device. When the slave device has consumed the transferred data, the *T.completer* function asserts the *i.consumed* signal and transmits it to the *T.initiator* function which then deasserts the *i.valid* signal causing the *T.completer* function to deassert the *i.consumed* signal. Further analysis may be found in [3]. Now to control a high performance, reliable communication channel enabling a master device to transfer information (write) to a slave device connected through an $m \times n$ interconnect block, the following functional cause-effect chain is appropriate to use:



In this example the $\text{T.initiator}_{\text{M.dev}}$ is part of the master device and the $\text{T.completer}_{\text{S.dev}}$ is part of the slave device. It is assumed that for a *write* function, data accompanies the *i.valid* signals. The Interconnect Block may be arbitrarily complex with many Initiator-Completer pairs, with each pair using the same protocol. It is not necessary to merge the T.initiator and T.completer into the master and slave devices. However, when separate, some synchronization mechanism needs to bind them together in order to provide a reliable channel. The separation of devices from the communication mechanism supports a high degree of variability in configuring bandwidth and latency to meet the requirements of specific communication channels. As well, there are likely to be multiple time domains operating across platforms; the inherent asynchronous nature of the initiator-completer pairs facilitates the bridging between such domains. These at-

tributes can be used to advantage by tools that build devices and platforms – such as VaST's Peripheral Device Builder and Platform Constructor Tool, respectively.

Multiple controller, distributed automotive control platform: This system differs almost completely from the wireless example. Although in more recent developments, multiple processor controllers are being designed into power-train applications, in general the electronic control units in a car use relatively low speed, single processors that communicate via



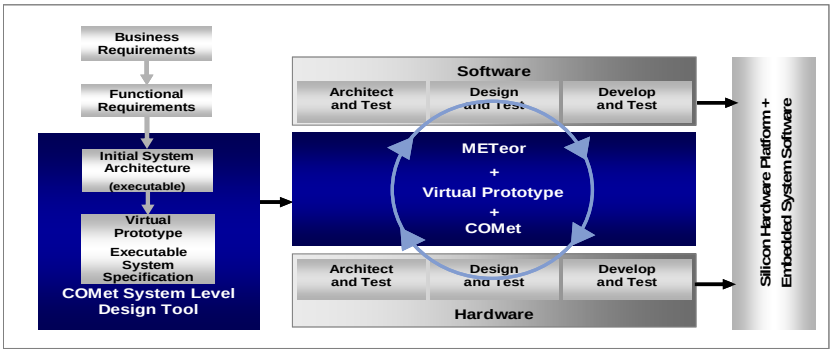
low bandwidth, high latency communications channels to provide the required critical, real-time feedback control. However, the requirement that control and safety functions in a car must meet specified hard deadlines places real-time constraints on the communication protocols. To meet this requirement the time-triggered protocols implemented on CAN-TT and FlexRay buses are increasingly being deployed.

The purpose of modeling such a system is to determine the required bandwidth and latency of the interconnect structure and the computation attributes and capabilities of the control units, under both extreme and average conditions. If the engine controller and the suspension controller each formed part of a distributed stability controller for a car, then safety, reliability and requirement to meet real-time schedules must be established under the harshest conditions.

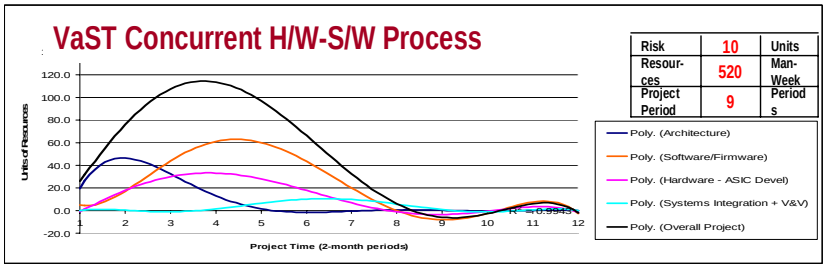
It should be noted here that experiments performed on the model are able to cover many more cases than testing using a physical artifact. This is a surprising result to most engineers. But the controllability of a model and the ability to configure extreme situations, as well as the ability to run data acquired from a physical artifact through the model, enable more comprehensive testing to be undertaken. Often this testing on the VSP is at, or better than, real time – since most subsystems in a car have relatively slow response rates to events, the VSP running on a modern PC will outperform the slow silicon used to implement the physical controller. The exception is in power train which will require multi-host simulation capability – an area in which VaST has an active research program.
The contrast between the two example gives an interesting insight into the modeling required of processor, interconnects and devices and the wide variety of characterization of objective functions and experimentation required to drive optimization. One constant, however, remains. Models need to the timing accurate and high performance to be used for architectural experimentation and for developing software for real-time control systems. Software developed on inaccurate and/or incomplete models cannot be guaranteed to run on silicon; such models cannot guarantee anything about real-time capabilities – meeting of hard schedules, power consumption, performance, bandwidth, throughput, inter alia.

The CRES Design Process

The design process underlying a modern CRES system development is empirically driven and differs from the conventional process by initially developing an executable architecture of the whole system – hardware, software, I/O transducers. Then deriving a high performance, timing accurate system golden reference model - known as a Virtual System Prototype (VSP) - to concurrently design the synthesizable hardware (including the I/O transducers) and complete development the software. A side-effect of this process is that the VSP becomes the golden test bench including for register transfer hardware designs.
VaST's version of the System Development Process is shown below:



This process has a profound effect on the efficacy of the engineering of CRES systems. The process curves below reflect the modern style of system development for the same 2.5G mobile phone discussed for the conventional embedded systems design process.



Comparing the Conventional and Concurrent design styles results in some dramatic changes. Firstly, architecture – involving hardware, software and transducers – is represented separately as a significant up-front activity that commands a lot of resources. The initial development of the architecture, requires porting target operating systems and legacy software and the development of, at least, stub drivers for projected hardware – that is, software development begins well before hardware design. The peak of the resource deployment curve now occurs in the first third of the project, reflecting the determinism derived from having a system defined and credible project plans in place early on. The net effect of the process change, for this project, was a decrease in resource requirements of about a 35%, a decrease in time to market of around 40% and a decrease in project risk of a massive 75%. These ranges of figures are consistently reported by VaST's customers for this caliber of product development.

A side note is that for this process to perform, the VSP needs to be timing accurate: clock cycle accurate for processor; clock edge accurate for transactions through buses and bus bridges; and timing accurate for peripheral devices and I/O transducers. The objective is to get the model running with less than 1% timing variation from the silicon – with recognition that loads – particularly bus, memory and device loads – will affect the silicon timing when compared with the synthesized register transfer hardware design. Concomitantly, VSPs also need to be high performance: 50-200+MIPS for the processor, 1.5 million transactions/sec for buses and bridges, and peripheral devices depending what they need to do. If the VSP is slow, the experimentation at the architectural end will be incomplete, likely seri-

ously so, resulting in irreparably deficient specifications in competitive markets.

Specification which are incompetent in either timing or function and become the golden reference, will problematically drive the software and hardware engineer processes. The effect is likely worse than the intuitive engineering process that drives the old hardware dominated design and that has been well documented [4] in producing late and functionally incomplete designs (more than 50% of the time), project cancellations (20% of the time), product respins (more than 40% of the time), and grossly over-engineered products. As products become more complex, these numbers climb. The Architecture Driven Design Process is the first huge step in addressing this problem. Quantitative driving of this process further refines it to produce optimal control systems from optimal Virtual System Prototypes. This is the subject of the next Section.

Quantitative Systems Architecture

The use of empiricism in developing optimal software is rare. The complexity of processor centric, electronic systems that control modern products requires a systematic approach to developing systems (software and hardware) in order to deliver an optimal fit for an intended product. When engineering is critical to survival in global, competitive markets, building optimal systems is a requirement – over engineering cost time to market and money; under engineering fails to achieve the intended purpose of the product; and worst, under engineer by intuitively over-engineering selected subsystems – a remarkably frequent case – costs time and money and still fails to produce a competent product [5].

The intuitive optimization of systems – architecture, software design, hardware design, and interfaces – has largely been driven from hardware design. Since hardware designers have rarely been required to understand, the software that would run on their architectures, they tended to speculatively over-engineer perceived *performance critical* subsystems. In relatively simple systems, this history driven design style produced working architectures of unknown optimality. This approach to systems design has largely been spectacularly unsuccessful for complex designs. The ability to support data-driven decision making early in the system development process is one of the underlying drivers of building models of systems that are timing accurate and high performance.

It is known that software and algorithms have a 1st order effect on an embedded system's performance; similarly hardware (platform) design has a

1st order effect on the system. This is difficult to reconcile with practice, when next-generation product planning often focuses on processor micro-architecture, regardless of the fact that 5%-8% improvement gained from iterative micro architecture development affects the system negligibly - a 2nd or 3rd order effect.

Deriving best-case, average and worst-case system performance from simulation using VSPs enables analyses of system variability leading to the identification of factors having the most significant impact on variability. These factors are prognostic as well as diagnostic and they can be used to drive the optimization of systems. In order to use such factors to drive the optimization process, a great number of experiments – involving many simulation runs - on various configurations of a system may be required. The design of experiment methodology [6] helps by providing a statistically valid mechanism for dramatically reducing the number of experiments needed to be performed.

Since VSPs are used to directly execute software, including hard real-time code, during development and debugging, trace information (streamed from non-perturbing probes inserted into the model) - including response latencies, power consumption, speed between markers, frequency of function calls, etc. - is produced alongside the usual debug data and hence is available to software and systems engineers as a normal part of the edit-compile-execute-debug software development cycle. This changes the perspective of where optimization should occur – as a normal part of the development cycle, not as a post development clean-up.

Optimization of Systems

Event-Based, Objective Functions

In an event driven simulation environment, an objective function can be expressed as a function whose parameters are functions each characterizing contributions to the objective function of one of the components constituting the system, viz. CPUs, buses, bus bridges, memories and peripheral devices. The parameter functions themselves have parameters that are functions of simulation event types sourced from the various event activities that occur in a VSP during simulation. In general, an objective function is a function of functions of functions of events and has the form of Equation 1, below. This

Equation 1:

$$
\begin{aligned}
F_{Power}(\,| & f_{CPU}(\Theta_{cc=0..cn} \circ f_{CPU_{cc}}(\Theta_{CEvType=1..cet} \circ g_{CPU_{cc,CEvType}}(\Theta_{CEvCnt=scecn..tcecn} \circ Event_{CPU_{cc,CEvType,CEvCnt}})),\\
& f_{Bus}(\Theta_{bc=0..bcn} \circ f_{Bus_{bc}}(\Theta_{BEvType=1..bet} \circ g_{Bus_{bc,BEvType}}(\Theta_{BEvCnt=sbecn..tbecn} \circ Event_{Bus_{bc,BEvType,BEvCnt}})),\\
& f_{BusBridge}(\Theta_{bbc=0..bbcn} \circ f_{BBus_{bbc}}(\Theta_{BBEvType=1..bbet} \circ g_{BBus_{bbc,BBEvType}}(\Theta_{BBEvCnt=sbbecn..tbbecn} \circ Event_{BBus_{bc,BBEvType,BBEvCnt}})),\\
& f_{Mem}(\Theta_{mc=0..mcn} \circ f_{Mem_{mc}}(\Theta_{MEvType=1..met} \circ g_{Mem_{mc,MEvType}}(\Theta_{MEvCnt=smecn..tmecn} \circ Event_{Mem_{mc,MEvType,MEvCnt}})),\\
& f_{Dev}(\Theta_{dc=0..cn} \circ f_{Dev_{dc}}(\Theta_{DEvType=1..det} \circ g_{Dev_{dc,DEvType}}(\Theta_{DEvCnt=sdecn..tdecn} \circ Event_{Dev_{dc,DEvType,DEvCnt}})))
\end{aligned}
$$

$$
where\ f_{CPU_k}(\Theta_{EvType=1..et} \circ g_{CPUk,EvType}(...)) = f_{CPU_k}(g_{CPU_k,1}(...), g_{CPU_k,2}(...), ..., g_{CPU_k,et}(...))
$$

is actually a generic function that computer power consumption.

Binding the Generic Power Function to a Specific Architecture

A variant of the wireless architecture discussed above was instrumented. One experiment investigated the power consumed by the ARM1156 – the 3 other processors were held in *reset* for the duration of this simple experiment [7]. The basic function computed is Instant Power which calculates the total energy consumed over some period of time or some number of events (such as cycles).

The functions computed that are useful for optimization purposes are:

- Maximum power consumed, over a particular period (maximum of the instant powers)
- Average power consumed over the whole experiment.

A specific function used to compute instant power per k-cycles is given in the Equation 2:

$$Equation\ 2:$$
$$f_{Power} = .W_{Pipe} \times f_{Pipe} + W_{Instr} \times f_{Instr} + W_{Cache} \times f_{Cache} + W_{TLB} \times f_{TLB} + W_{RegAcc} \times f_{RegAcc} + W_{MemAcc} \times f_{MemAcc} + W_{PeriphAcc} \times f_{PeriphAcc}$$
$$where.$$
$$f_{Instr} = .2 \times f_{Instr,jmp} + 2 \times f_{Instr,except} + 0 \times f_{Instr,ctrl} + 12 \times f_{Instr,coproc_{15}} + 0 \times f_{Instr,LdSt} + f_{Instr,arith} + f_{Instr,other}$$
$$and.$$
$$f_{Instr,i} = .\sum (instructions\ of\ type_i\ in\ k-cycles)$$

Similar functions occur for f_{Pipe}, f_{Cache}, f_{TLB}, f_{RegAcc}, f_{MemAcc}, $f_{PeriphAcc}$ and the weights for the constituent *accumulating* functions are given in Table 1, and the weights (W_i) for each of the classes of functions contributing to f_{Power} have been set to the constant function 1 in this study.

Table 1: Power: Function Types, Event & Weighting Functions

Function Types	Events	Weight Functions
Pipeline	ibase	6.0
Instruction Types	ijmp	2.0
	iexcept	2.0
	ictrl	0
	icoproc	12.0
	iundefs	0
	imemrd	0
	imemwt	0
	imemrw	0
	iarith	1.0
	iother	1.0
Caches (I&D)	Cache_lookup	$f_{i\text{-}dcache}$(size, ways)
	icache_hit	iCache-lookup + f_{icache}(line size, decode)
	icache_miss	Icache_lookup
	dcache_hit	Dcache_lookup + f_{dcache}(size, ways, line size)

	dcache_miss	Dcache_lookup
	line_fill	0
TLB	tlb_miss	30.0
Register	regfile_access	1.0
Memory (incl. bus transactions)	membus_transaction	50.0
Periph Device (incl. bus transactions)	periphbus_reg_access	50.0

In industry studies, the *accumulating* function might be replaced with individual functions relevant to computing power in ways not considered for the simple examples of this paper. Such functions can include history and implementation dependent technology functions. Similarly, the weights (W_i) may be more complex functions – for example, the cache hit weights are functions of cache structure (size, wayness, policies).

The Use and Interpretation of Event-Based Optimization Functions

Event Bindings are simple to implement, typically a pointer to a function and a history buffer of events. However, the extraction of data from register transfer (RT) models or representative samples of the silicon is difficult, time consuming and subject to experimental errors. The inability to map an event in a behavioural model to an observable data point in the silicon or RT model further complicates the building of accurate tables.

Events, in event-based simulation, are associated with aggregate underlying behaviour of the circuits that the semantics of events are intended to describe. The number sources of Event Types generated by all types of components in a complex platform may number in the hundreds; the number sources of Events Instances (events caused by the simulation of instances of typed components instantiated in a platform) numbers in the thousands, possibly many thousands. Since component instances, even though morphologically similar or identical, may have different electronic instantiations and be affected by local circuit connectivity, the objective function table may remain large. To set appropriate Event Bindings for entries in the Objective Function Table, the knowledge and skills of the silicon vendors are required.

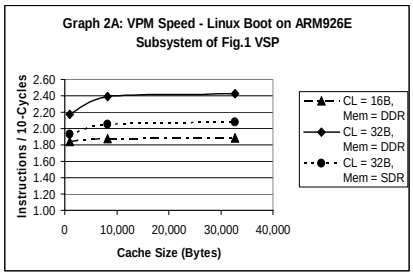
Since the precise physical association of an event with some circuit implementation that it models is not necessarily obvious and, in many cases, unlikely to be independent of circuit effects associated with other events. The ascription of local physical semantics to an event is more in the nature of a verisimilitude.

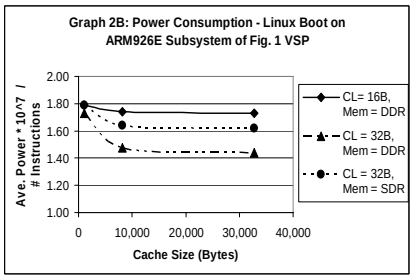
The higher the level of event-based, behavioural abstraction, in general, the weaker is the physical (or structural) connection to implementation. This is a good thing for high performance modeling and, with forethought, does not compromise timing accuracy at an agreed level of granularity – say clock-

cycle level. However, it does mean that more sophisticated mechanisms are required to efficiently predict aggregate underlying hardware concepts, such as power. In addition, various event types and instances may correlate highly across a variety of behaviours - for instance, cache misses with processor initiated bus traffic, data path stalling and power consumption. The elimination of dependence between event instance sources, that are the potential *independent* variables in a statistical analysis, should have the effect of condensing the number of variables required to explain a behaviour, or to be used in the prediction of behaviour as well as reduce the propensity for over estimation using data extracted from these models.

An Example of Quantitative Investigation

The Speed – instructions/10-cycles (IPC_{10}) and relative Power Consumption of 9 structural variants of an experimental VSP similar to that used to describe the power calculation above were computed while booting Linux. The variants were selected from the full set of variants determined by - cache size: 1k, 8k, 32k; cache line: 16B, 32B; Memory configured as DDR (1^{st} word delayed 5-cycles, 2^{nd} word available per ½ cycle) and SDR (1^{st} word delayed 5-cycles, 2^{nd} word available per 1 cycle); bus data width 4bytes. The results are shown in the Graphs 2A & 2B.





The boot sequence of Linux spends more than 50% of its time executing with the I&D caches disabled. Linux performs initialization of the cache after the Initial Program Load, kernel load and the device driver installations. Once the operating system has booted and the idle loop is executing, the behaviour of the VSP is much the same as its behaviour running Viterbi – that is the working-set size is compatible with any cache size. As is also expected, in an environment where the working set size of the target code greatly exceeds the cache size, the impact of the memory hierarchy on power and speed is considerable. For booting Linux, the settings of the ARM926E VSP subsystem: cache size (32 kbytes), cache line size

(32bytes), and Memory (DDR) yield minimum power consumption and maximum VSP speed.
To minimize cost, as well, a cache size (I&D) of 16 kbytes would proportionally reduce silicon cost by about 30% and adversely affect both power and speed by about 1%. To further optimize for cost, cache sizes of 8 kbytes will yield a further ~25% reduction in silicon with a worsening in power consumption and speed of 5%-10%. The overall results are expected, but the quantification of the results cannot be derived intuitively – these can only be gotten from a timing accurate model that has sufficient performance to enable large, representative software loads to be run. These results should be used to drive the optimization of the VSP – hardware, software and IO interfaces.

Summary and Conclusion

The use of high performance, timing accurate models of multi-processor and distributed systems is part of the standard methodology for building complex critical reactive embedded electronic control systems. The requirement to develop software prior to the design of hardware and to comprehensively test the device, preclude the architecting and development of such systems physically. The use of virtual system prototypes (VSP) as the golden reference for system development – software and hardware – is the logical outcome of this discussion.
In the quest for developing systems that work and solve the problems they are design to solve, empirical experimentation is the only methodology humans have evolved to ensure either desired outcomes or reasons for the failure to produce a desirable outcome. In this methodology, the refutation of carefully constructed hypotheses is the centre piece that needs to drive the quantitative engineering process – this is just the scientific process. Intuitive design, however intelligent, is no substitute for engineering decision making driven by hard data.
If hypothesis building concerns speed, power consumption, reaction time, latency, meeting real-time schedules, etc. the model needs to be timing accurate (processor, buses, bus bridges and devices). If the extensive execution of software is an intrinsic part of the empirical experimentation, then the model needs to have high performance across all components.
Optimizing systems with complex objective functions is not intuitive. Complex tradeoffs between hardware structure and the software and algorithms that are executed on the hardware cannot be done by ratiocination or formal analysis alone, the acquisition of data as part of well-formed ex-

periments refuting thoughtfully constructed hypotheses (ratiocination) enables decision making driven by results. Optimization comes from considering hardware, software and IO interfaces together – not separately.
The process driven by VSP as the golden architecture itself drives radical change in the engineering systems of companies. This change offers huge savings in resources - human and capital, time-to-market, and fitness, safety and reliability for use of the resulting products. The reduction of project risk is of overwhelming importance. In the situation where advanced engineering companies with high infrastructure costs and inflexible decision making are unable to compete with newer entrants having lower costs and greater flexibility, the VSP driven architecture process offers a strategic mode where intelligent, data driven activity dictates architectures that can then be contracted out for piecemeal realization – without divulging the consolidated intellectual property constituting the system or the decision making process that resulted in the optimal family of designs. Not just a desirable strategy but a mandatory one.

References

[1] Hellestrand, G.R. The Engineering of Supersystems. IEEE Computer, 38, 1(Jan 2005), 103-105.

[2] ARM AMBA AXI Protocol Specification. 2004. www.arm.com

[3] Hellestrand, G.R. Event, Causality, Uncertainty and Control. 221-227, Proc. 2nd Asia Pacific Conf. on Hardware Description Languages (APCHDL'94) Toyohashi, Japan, 24-25 Oct. 1994

[4] Venture Data Corporation. Collett International Report on Embedded Systems. 2004.

[5] Hellestrand, G.R. Systems Architecture: The Empirical Way – Abstract Architectures to 'Optimal' Systems. ACM Conf. Proc. EmSoft2005, Sept 2005, Jersey City, NY.

[6] Montgomery, D.C. Design and Analysis of Experiments. 5th Ed. John Wiley & Sons, NY, 2001.

[7] Hellestrand, G.R., Seddighnezhad, M and Brogan, J.E. Profiles in Power: Optimizing Real-Time Systems for Power as well as Speed, Response Latency & Cost. Power Aware Real-time Conference (PARC2005), Sept 2005, Jersey City, NY.