Empirical Mapping of Abstract Executable Specifications to Networked and Closely Coupled Processing Systems

Graham R. Hellestrand

VaST Systems Technology Corporation
1250 Oakmead Pkwy, Suite 310, Sunnyvale, CA 94085 USA

Abstract. In automotive systems, (i) distributed, real-time algorithms, operating on a set of networked electronic control units (ECUs), are used to control the car (for example: the stability system); and (ii) fine grained video and audio algorithms are often mapped to an ensemble of processors in order to achieve the required performance at a known cost and, for untethered devices, with a known power budget (for example: an after-market cell phone/PDA supporting video and audio streaming – terrestrial and satellite, GPS, etc.). The mapping of complex algorithms to systems containing multiple processors that satisfy some set of optimality conditions is notoriously difficult [8]. An empirical, refutation-based (scientific) process; design of experiments technology; and multi-variate statistics [6] can be employed to help drive the optimization process. Experimentation involves performing potentially many hundreds of experiments, each of which is meticulously measured, so that the processed data can be used to make decisions about the next steps to take in modifying the system (hardware, algorithms, software mappings, input/output interfaces, etc.) in the iterative march towards an optimal system. Undertaking such an experimental regime using physical hardware is prohibitively expensive in time and resources. The use of virtual (modeled) systems, together with the statistical machinery and empirical processes, makes this process plausible and necessary. This strategy is somewhat at variance with the AutoSAR & JASPAR standardization initiatives but is consistent with the notion of services-based interfaces where, for example, the *always* best external data sources (such as satellite and terrestrial RF data) – whether captured through the car's infrastructure or a *plugged-in* external unit – should be available to whatever process is providing a requested service.

Introduction

The application of software-driven electronics in a modern automobile ranges from real-time control tasks, such as controlling stability, to multi-media processing tasks, such as decoding compressed video streams. The architectures of such disparate systems range from platforms supporting large, cooperating tasks with feedback connections – typically involving physically separated controllers connected via a communication fabric having guaranteed information delivery latencies and fault-

Automotive Software Workshop San Diego 2006
UC San Diego, California, USA

tolerant characteristics – to platforms supporting small, fine-grained algorithmic concurrency – typically, tightly coupled processor interconnected via very high performance (on silicon) communication fabrics, such as crossbar interconnects.

Where a principal component of optimization in the automotive industry is minimizing the cost of components in high-volume manufacturing, the concept of uniform software interfaces across the gamut of automotive, software-driven electronics systems appears contradictory. However, tools and methodologies that provide the same efficiencies as uniform environments for software development and deployment, yet enable individual software and hardware constituting a controller within a platform, as well as, the entire platform to be optimized, seem to provide a better fit in the environment where controllers and subsystems with a wide range of capabilities and constraints are developed for automobiles.

This paper addresses the use of virtual prototypes - high performance, timing accurate models – used in an empirical, automotive design process to engineer optimized systems for control and multi-media applications. This strategy is at variance with the AutOSAR & JASPAR standardization initiatives in that is argues for specialization of processing ensembles to provide specialized services with exogenous constraints that are atypical with respect to the other services required in a car. But it is consistent with the notion of services-based interfaces where, for example, the *always* best external data sources (such as satellite and terrestrial RF data) – whether captured through the car's infrastructure or a *plugged-in* external unit – should be available to whatever process is providing a requested service. Such an approach identifies various constraints - uniquenesses in response times (real-time critical control), computational performance (video decode), power consumption (untetherable devices) – as required capabilities meet to be matched during the mapping of abstract executable specifications to the processes and processing engines of realization, under the overall constraint of minimization of manufacturing costs.

The Design Process

The design process for embedded system is complex – see Figure 1 below – and has been described in [5, 8]. Starting from Marketing and Engineering Requirements Specifications, a Functional Specification is manually derived, followed by the manual construction of an Abstract Executable System Architecture. This architecture is constituted from abstract executable processes (perhaps implemented as UML or Simulink tasks) interconnected by some communication fabric (perhaps using a VaST fabric model). Considerable quantitative experimentation can be performed on the abstract model providing the processes and communication fabric are functionally and timing accurate – sufficient to identify and size communication channels and place timing constraints on them, to place requirement constraints on processing engines that will realize the abstract processes, and to judge the feasibility of a realization. In Figure 1 below, this outcome is identified as the *Physically Mappable Executable System Specification + Physical Constraints (PMESS+PC)* box. The mapping of a PMESS+PC abstract model to an Executable System Architecture (or Virtual System Prototype (VSP)) requires the trial mapping of the abstract processes into compute

engines (software and/or hardware) and the communication fabric into realizable interconnect structures (buses, crossbar fabrics, etc.). The empirical process here experiments with combinations of physically capable compute engines (software processes executing on some ensemble of processors with appropriate characteristics) and interconnect structures to satisfy the overall system constraints. There may be families of architectures that are candidates for further refinement into practical electronic systems.

The final quantitative exercise is to look within and across the identified architecture families, to use capabilities within the VSP models (such as cache size and wayness, bus bandwidth, memory types, clock domain frequencies, software mappings to processor ensembles) to identify VSPs that meet the feasibility constraints. And then, amongst the potentially hundreds of candidate VSPs to select the one or family of VSPs that satisfy the optimality constraints.

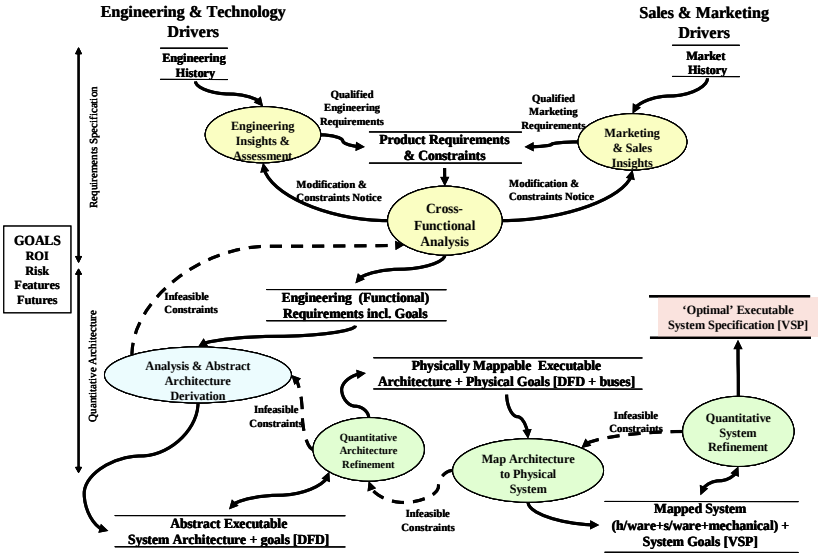


Figure 1: Requirements → Optimal Executable System Specification (VSP)

The Software engineering in such embedded systems is special – it ranges though the implementation of control code responsible for driving the hardware devices attached to an embedded system (such as: displays, keypads, MP3 players, wireless signal encoders/decoders), to power-aware database manipulations and games. All embedded software needs to be cognizant of the finiteness of physical resources present in the product controlled by the embedded system from the size and resolution of a display to the power required to perform a function, or a loop of an iteration within a function. The pervasiveness of the resource awareness constraint in embedded software development is sufficient to label this discipline uniquely – embedded software engineering.

As may be realized, the interdependence of software and hardware in embedded systems means that optimization is most meaningful when performed at system specification time and in conjunction with the Marketing and Engineering Requirements document defining the product to be developed.

Mappings

The mapping of a Functional Requirement (represented as a Data Flow Diagram) mapped to an Abstract Executable Specification composed of UML and Simulink processes, with both point-to-point message passing connections and Bus interconnections, is shown in Figure 2 [5]. The mapping of ensembles of processors to either or both closely coupled and network interconnected processors with UML and Simulink processes mapped to hardware is depicted in the 2nd mapping of Figure 2.

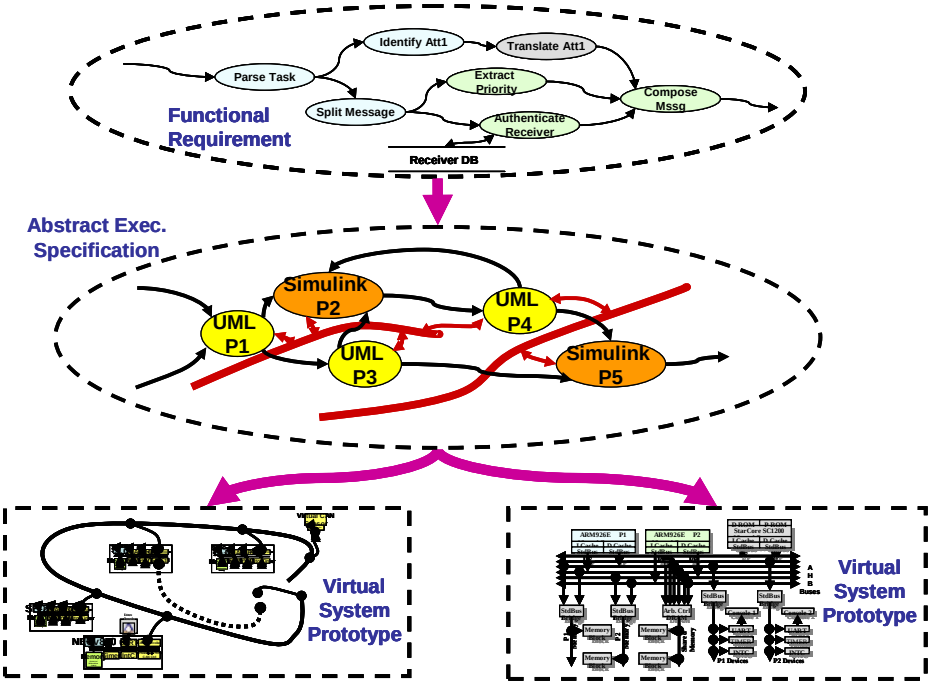


Figure 2: Mappings: Requirements → Abstract Exec. Specs → VSP

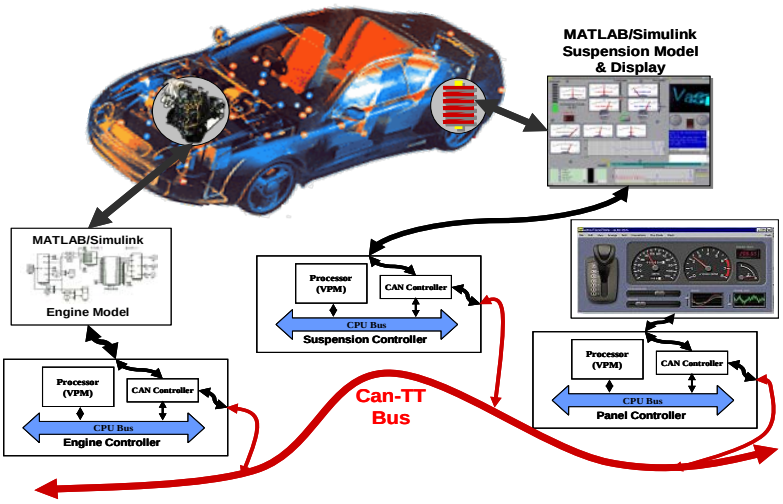
The two extreme types of target VSPs are shown in Figure 2 – Type 2: *local, nearly-autonomous computation* supported by *high latency, low bandwidth communication* (left side VSP target) AND Type 2: *highly cooperative computation* supported by *low latency, high bandwidth communication* (right side VSP target). There is a spectrum of possible mappings between these extremes. Indeed the left-side VSP target has embedded in its general high latency, low bandwidth structure, 2 nodes with local computation supported by low latency, high bandwidth communications. In general, there may be a recursive structuring of either Type within the other. The requirement for supporting the level of communication is

specified using by three factors – latency, bandwidth and frequency, which are empirically derived from the quantitative data produced by extensive experimentation with the Abstract Executable Specification. Coarsely structuring VSP targets in terms of high, medium and low levels of computation supported by high, medium and low levels of latency, bandwidth and frequency in communications generates a space of 81 architectural types. This classification provides an identification of each level of hierarchy in a complex VSP that will determine its physical characteristics in terms of computational clustering and interconnection. It is likely that within each *plane* of the hierarchy the effective classification will be determined by the communication space – a more tractable 27 architectural types. Once these determinations have been made, we find ourselves adjacent to a relatively known research and development space – that of floorplanning in VLSI design. At VaST, this is the direction of ongoing research.

Application to Distributed Control Systems

An example of a practical outcome can be discussed for a typical system found in the automotive industry – see [1], [2], [3] and [4]. The connection of two interlinked control subsystems with a display subsystem in a car – see Figure 3. Although the mappings are not shown, the mapped implementations of two abstract specifications – the engine control and the suspension control subsystems - are shown as controllers connected to the MatLab models of the engine and suspension, respectively that they are controlling. The control is implemented in terms of both software and hardware within the controllers.

The control of this virtual system was modelled in Simulink and then manually mapped to software to be executed by the electronic control modules connected via a CAN network. UML can also be used for this purpose. The complex models of the engine and the suspension subsystems were written in Matlab/Simulink. The



availability of high performance and accurate interconnect models - system and CAN bus models in this case - enabled accurate modeling of interconnect and communications as well as processing – including the control effected by the feedback paths.

This model was not used to support experimentation at the abstract level. However, implementing that step will enable the computing of bounds on the set of models and the interconnection of models that are of interest in the later, finer grained optimization investigations – thereby limiting the investigative effort across a potentially huge abstract architecture space. Having data to support required interconnection bandwidths, communication latency tolerances, and computational performance requirements to meet real-time deadlines, is of inestimable value in limiting the search space.

Two Indicative Experiments

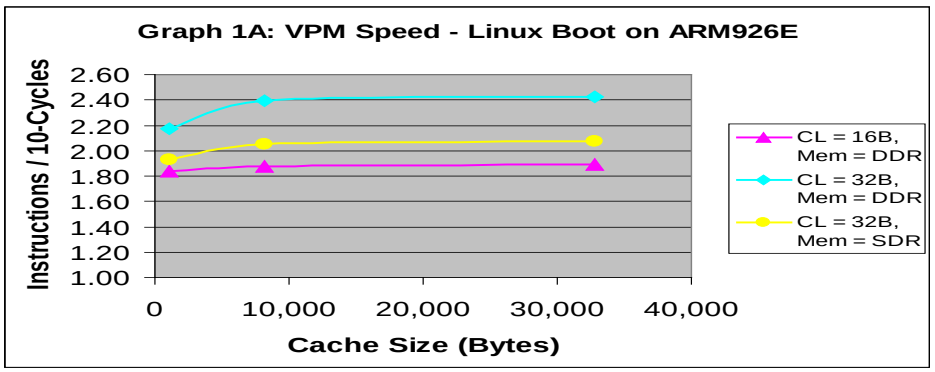
Two simple experiments are discussed below that demonstrate (i) the ability to measure and discriminate between computational capabilities that satisfy functional requirements (such as power consumed, performance and cost), and (ii) the ability to measure communication capabilities and determine their adequacy to support requirements (including performance and ability to meet real-time schedules). These are elements of a bigger empirical process that will eventually provide the data that will help drive the process described in Figure 1.

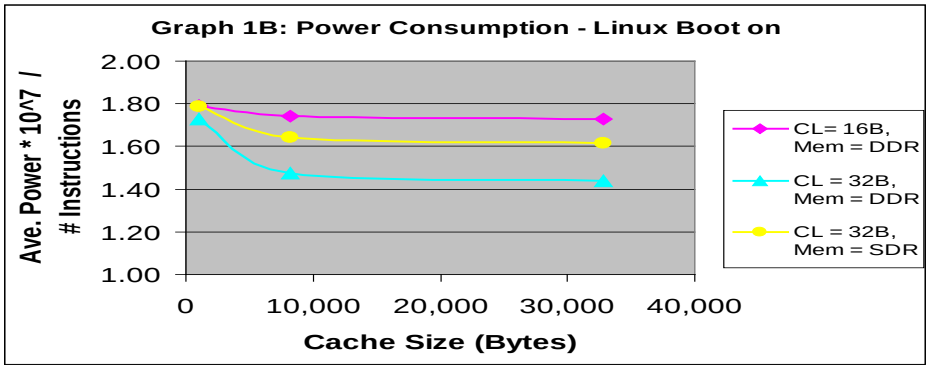
Towards Computational Optimization: Matching Computational Capability to Computational Requirements

We have assumed in this experiment the existence of a process, identified in the Functional Requirement Specification, which has been mapped to a single software task which is to be mapped onto a single processor platform (electronic control unit (ECU)). The experimentation will occur within that platform and involve determining better matching of original requirements to implementation capabilities, driven by trying to optimize an objective function with parameters performance, power and cost. This may involve local changes, such as matching local processor caches to task *working set* behaviours, modifying the memory hierarchy capabilities in the local ECU, modifying clock frequencies, etc. If this fails to match the capabilities to the requirements, then more substantial structural change is required, such as introducing a more powerful processor, mapping algorithms that have failed to meet real-time constraints and possess intrinsic concurrency to a cluster of processors in mapping the inner-iterations of such algorithms to hardware accelerators, etc.
This pattern of local (usually parametric) changes being tried before the more substantial structural changes mimics the ways that engineers behave when confronted with such issues – conservatively. Where data is used to determine the success or failure of meeting requirements, the techniques in the *Design of Experiments* [7] statistics can be used to pareto order the parameters and structural

changes that can then be applied to wring the greatest affect, as determined by the optimizing function, with the least traversal of the design space.

This experiment uses a typical, single processor VSP, in this case an ARM926E, with instruction and data ports attached to individual AMBA AHB buses; a single instruction and data memory is attached to each of the AHB buses via an arbitrating bus bridge; a timer, interrupt controller and UART are attached to a separate AHB bus which is bridged to the AHB bus attached to the processor's data port; and the interrupt and other relevant signal lines are connected as expected. The software tasks running on the VSP was Montvista's 2.1 version of the Linux kernel [9]. This





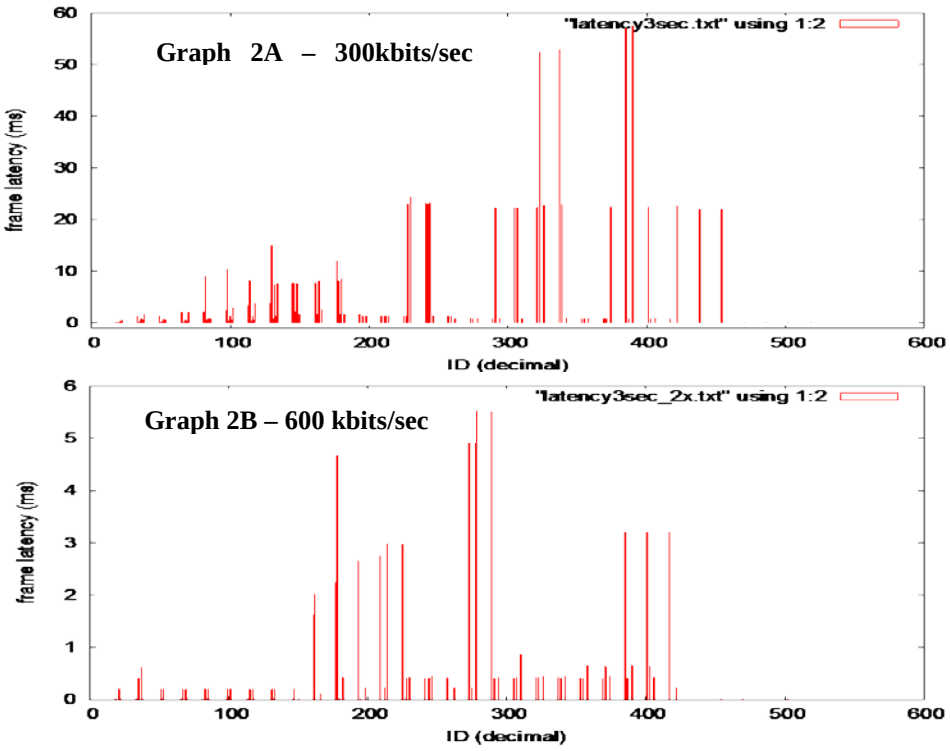
was used in order to examine the behaviour of the VSP under significant uncached as well as cached loads. The measurement made were power consumed and performance – that is, instructions per 10—cycles (IP10-C) of the processor clock. These measures were charted separately and are shown in Graphs 1A and 1B, above. IP10-C is a direct result from the VSP. The computation of power is described briefly in the Appendix 1 of this paper.

The results of about 20 simple experiments were very interesting. Under the mixed uncached and cached loads presented by booting the Linux operating system (takes about 30 seconds on the VSP), we varied instruction and data cache line sizes and the type of memory - from slow single data rate (SDR) memory, to fast double data rate (DDR) memory – and then measure performance (IP10-C) and power consumption. As may be reasonably expected (even though under the set of complex conditions of this simple experiment, the unexpected occurs relatively frequently!), as cache line

size is increased and memory is accelerated, the IP10-C number increases and the power number decreases. It is clear that, for the experiments measured, that the largest cache line size coupled with the fastest memory produces an optimum platform with capabilities that maximize performance and minimize power consumption for the execution load presented by the Linux boot. Note that we did not explicitly report the frequencies of interrupts, or work done by the interrupt controller and the UART, since we were primarily interested in the behaviour of the processor and the memory hierarchy. Power was computed from processor micro-architecture, transaction bus activity, and memory activity.
It is clear that complete platform activity can be measured and reported and that this sort of empirical data is fundamental to driving decisions about how to produce optimal systems that incorporate software, hardware and linkages with the external (modeled) world.

Towards Communication Optimization: Determining Adequacy of Communications Capability to meet the Communications Requirements

This is a relatively sophisticated experiment that examines patterns of prioritized packet communication on a high latency, low bandwidth serial bus – Controller Area Network (CAN). By automatically identifying patterns of communication over extended period of time, for example, that fail to enable (i) a task running on a



processor connected to the bus to meet a real-time deadline (latency failure) or (ii) a

periodic communication with some priority generated by a task to be scheduled within the period of the communication requirement (bandwidth failure), a local remedy may be applied or, if unsuccessful, a structural change may be applied by moving to a more propitious communications capability class in the identified space of 27 classes of communication capabilities.

The CAN bus described in the experiment has 6 nodes with each node transmitting a number of messages within the period of the experiment. The message has a Frame Identifier that defines the priority of the message and maps between a statically assigned – at set-up time - source node and destination node both attached to the CAN bus. Frame ID=0 has the highest priority and transmits messages 0x00, 0x20, 0x40, …, where in our case the bottom 4-bits defines the Frame ID. Each node should only transmit a unique set of IDs which guarantees the uniqueness of the message priority and integrity of the arbitration system. The CAN bus has a clock with frequency of 300k cycles/sec.
The latency, defined as:

(Start time of frame ID when it wins bus arbitration) –
(Start time of frame ID for its first attempt to win bus arbitration)

was measured for each message sent from a node and graphed in Graph 2A. Graph 2A shows that a few messages - and not the lowest priority - get stuck waiting to go on the bus at times. This means that the latency of some messages exceeds the periodicity of their expected transmission thus causing a real-time schedule to be missed. These are the types of things that break networks.
The bus clearly does not have the bandwidth to enable a guarantee to be attached to the VSP certifying that it will not miss real-time schedules. To test this assertion, we doubled the bandwidth by doubling the clock frequency of the CAN bus to 600k cycles/sec. Graph 2B shows the result. Note the vertical axes of Graphs 2A and 2B differ to enable a better view of the traffic at 600k bits/sec. The result of this change is that for the same traffic, no messages get stuck and the real-time constraint, as far as communications is concerned, can be met. Interestingly, and not immediately obviously, doubling the frequency more than decreases the latency by ½.

The communication experiment shows graphically how capabilities of communication structures can be varied to determine experimentally whether a communications requirement can be satisfied for particular set of communications in a particular circumstance.

Summary

This paper has use an ad hoc, empirical process to analyze and modify Virtual System Prototypes – pre-silicon, high performance, timing accurate models of systems – so that they meet requirement specifications. The process and statistics used to drive the process [8] enable quantitative data to be used to make critical architecture decision well before hardware is available. The methodology argues that optimizing subsystems in a car will lead to specialization of the hardware and software. This is at variance with JASPAR and AutoSAR, where it is desired for all hardware to have a

unified interface to all applications. Clearly, this constraint will lead to sub-optimal systems.

The process outlined in Figure 1, is VaST's roadmap for the future in driving the empirical design of systems. We expect to supply tools to enable joint architecture teams composed of software and hardware, as well as specialized architects, to confidently specify, model, validate systems and then to iterate in order to realized optimal systems. This is a grand challenge – the first 1,000 paces have been taken!

Acknowledgements

I would like to thank Casey Alford for setting up and performing the CAN bus and other experiments and then interpreting the results. Similarly, Mahdi Seddighnezhad, Jim Brogan and Casey Alford were instrumental in producing data for the single processor system and discussing and reiterating the results. VaST has formed a research group to investigate quantitative architecture, called QArk. The charter of QArk is to pursue quantitative architecture research and then to define tools and capabilities that need to be defined and built to support our customers in the use of this powerful methodology and technology.

References

1. Shigematsu, Takashi. Software Quality Management Applied to Automotive Embedded Systems. SAE 2002-21-0017.
2. Kawana, S., Noritaka, O, Shigematsu, T and Nagai, Yoshiyuki. Empirical Approach for Reliable Assurance of Vehicle Software. Automotive Software Workshop San Diego, Feb. 2004.
3. Winters, F.J., Mielenze, C. and Hellestrand, G.R. Design Process Changes Enabling Rapid Development. Proc. Convergence 2004 P-387, Oct 2004, 613-624, Society of Automotive Engineers, Warrendale, PA.
4. Frischkorn, H-G. Automotive Software – The Silent Revolution. Keynote. Automotive Software Workshop San Diego, Feb. 2004.
5. Hellestrand, G.R. Systems Architecture: The Empirical Way – Abstract Architecture to Optimal Systems. Proc. 5th ACM Intl. Conf. on Embedded Systems, Sept. 2005, Jersey City, NJ, 147-158, ACM 100059.
6. Hair, J.F., Anderson, R.E, Tatham, R.L and Black, W.C. Multivariate Date Analysis with Readings. 4th Ed. Prentice-Hall International, Inc., NJ (1995).
7. Montgomery, D.C. Design and Analysis of Experiments. 5th Ed. John Wiley & Sons, NY, 2001
8. Hellestrand, G.R. The Engineering of Supersystems. IEEE Computer, 38, 1 (Jan 2005), 103-105.
9. www.montavista.com

Appendix 1 – Computing Power

We instrumented the VSP containing the single ARM925E core described above. 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). A simplified function used to compute instant power per k-cycles is given in the Equation 1, below.

$$Equation 1:$$
$$f_{Power} = .W_{Pipe} \times f_{Pipe} + W_{Instr} \times f_{Instr} + W_{Cache} \times f_{Cache} + W_{TLB} \times f_{TLB} + W_{\mathrm{Re}\,gAcc} \times f_{\mathrm{Re}\,gAcc} + W_{MemAcc} \times f_{MemAcc} + W_{PeriphAcc} \times f_{PeriphAcc}$$
$$where.$$
$$f_{Instr} = .2 \times f_{Instr,jmp} + 2 \times f_{Instr,except} + 0 \times f_{Instr,ctrl} + 12 \times f_{Instr,coproc_{15}} + 0 \times f_{Instr,LdSt} + f_{Instr,arith} + f_{Instr,other}$$
$$and.$$
$$f_{Instr,i} = .\sum(instructions\ of\ type_i\ in\ k-cycles)$$

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.

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. 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).

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