Exploring Trade-off's Between Centralized versus Decentralized Automotive Architectures Using a Virtual Integration Environment

Sri Kanajan
General Motors Corporation
Warren, Michigan 48090
sri.kanajan@gm.com

Haibo Zeng
University of California, Berkeley
Berkeley, CA 94720
zenghb@eecs.berkeley.edu

Claudio Pinello
General Motors Berkeley Lab
Berkeley, CA 94720
claudiopinello@cal.berkeley.edu

Alberto Sangiovanni-Vincentelli
University of California, Berkeley
Berkeley, CA 94720
alberto@eecs.berkeley.edu

Abstract

The large variety of architectural dimensions in automotive electronics design, for example, bus protocols, number of nodes, sensors and actuators interconnections and power distribution topologies, makes architecture design task a very complex but crucial design step especially for OEMs. This situation motivates the need for a design environment that accommodates the integration of a variety of models in a manner that enables the exploration of design alternatives in an efficient and seamless fashion. Exploring these design alternatives in a virtual environment and evaluating them with respect to metrics such as cost, latency, flexibility and reliability provide an important competitive advantage to OEMs and help minimize integration risks later in the design cycle. In particular, the choice of the degree of decentralization of the architecture has become a crucial issue in automotive electronics. In this paper, we demonstrate how a rigorous methodology (Platform-Based Design) and the Metropolis framework can be used to find the balance between centralized and decentralized architectures.

1. Introduction

The electronic content in an automobile is increasing at a rapid rate. In-vehicle electronics is displacing the traditional mechanical interfaces and enhancing the driving experience in every area from safety to telematics. OEMs face a number of challenges related to the integration of this electronic content. Specifically, a significant challenge lies in the selection of an appropriate architecture. The selection of an electronic architecture is presently an *ad-hoc* activity that relies heavily on the expertise of the designer. Architecture optimization is essentially driven by an architect using a qualitative and manual design approach. Traditionally, an automotive electronic architecture is highly decentralized as the components are provided by Tier 1 suppliers that develop subsystems implemented in a self-contained ECU. This approach makes integration difficult as there is little visibility from the OEM on the content of the ECUs and on their behaviors. Recently, OEMs are taking increasingly control over the architecture so that a global view of the functionality and of the implementation platform could be used to optimize a number of important metrics such as cost, reliability, fault tolerance, and performance. Reasoning at a qualitative level, an architect trades-off the level of flexibility and modularity offered by a decentralized architecture versus the cost effectiveness and low latency of a centralized architecture. Enabling a designer to make various design choices quickly and to evaluate them systematically using quantitative analysis would definitely increase efficiency in OEM engineering. Considering the role of OEMs as system integrators of a heterogeneous set of electronic subsystems fed by a complex and distributed supply chain, it is essential for the design methodology to facilitate the capture of constraints and requirements on the architecture coming from the supplier components.

This paper presents the application of the platform-based methodology and the supporting framework (Metropolis)[1] to explore the trade-off between centralized and decentralized architectures in a design case study.

1 A more detailed explanation of the modelling semantics and integration environment is available in [1].

3-9810801-0-6/DATE06 © 2006 EDAA

2. Platform-based Design and Metropolis

The basic principle of the platform-based design methodology [5] is the “orthogonalization of concerns”, i.e. separating the various aspects of a design to facilitate greater design productivity. The methodology primarily advocates separation between the following aspects of a design:

1. Function (what the system does) and architecture (how it does it)
2. Computation and communication
3. Behavior and performance

There are four basic steps in a platform-based design flow [5]:

1. Choosing the model(s) of computation (MoC) [2]
2. Modeling the functionality of the system
3. Modeling the architectural platform
4. Carrying out different mappings of functions to architectural elements to evaluate different points in the design space

The first step involves choosing one or more models of computation and abstraction levels that are appropriate for modelling both the functionality and the architectural platform. Also, it must be possible to implement the MoC on the architectural platform in an efficient manner - it must be a good match for the architectural components. A specific model of computation may be realized at different abstraction levels. The abstraction level can be chosen based on the accuracy of models that are desired. Abstract models may not accurately capture the performance of the system, whereas refined models may cause unnecessarily high overhead. The result of this step in the design flow is agreement on a set of common services at a particular abstraction level and a set of rules that determine how these services can be composed - the MoC. After this step in the design flow, both the functional and architectural modelling activities can be carried out concurrently. A functional model captures the actions the system will carry out, but does not indicate how these actions will be performed. However, the functional model does not explicitly rely on an architectural model for completeness - it is an independent executable specification of the behavioral aspects of the system. The architectural model captures how the common services will be realized using the architectural components. The performance aspects of implementing these services are captured by the architectural model. An architectural model may be parameterizable. Mapping is the step in the design flow that allows the services utilized by the functional model to be bound with the services provided by the appropriately configured architectural model. The definition of a legal mapping depends upon the specific MoC used. A mapping represents a particular point in the design space of the system. After a mapping is realized, the performance of the system can be estimated using simulation. An iterative design space exploration procedure can be carried out to explore different points in the design space.

Metropolis [1] embodies the platform-based design methodology and provides a unified framework for simulation, synthesis and verification of heterogeneous systems. The infrastructure supports many different MoCs and abstraction levels. The specification language for Metropolis is known as the ”metamodel”, as any specific model of computation can be obtained by refinement, and supports both imperative and declarative specifications for modelling functionality, architecture, and mapping. Metropolis also provides a set of backend tools, including a simulator, and interfaces with formal verification tools.

3. Case Study

3.1. Functional Architecture

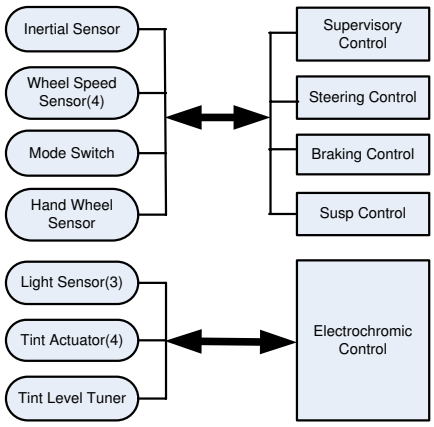

Figure 1. Functional Architecture.

The case study involves a combination of two independent functionalities - a supervisory control system that coordinates the steering, braking and suspension systems, and an electrochromic window control system. The objective of the supervisory control system is to integrate active vehicle control subsystems to provide enhanced riding and vehicle performance capabilities. The electrochromic control feature allows the driver to control the level of tint on the windows. The high-level view of the functional architecture of the two control systems is defined in Figure 1.

3.2. Architectural Model

The architectural elements that can be used to implement the functionality of the system consist of a set of ECU's, a CAN based communication bus and a set of sensors and actuators. The ECU consists of a set of software tasks, middleware, a CPU scheduler, an interrupt handler, a CAN Driver, and a CAN Controller[2]. The smart sensor/actuators have a CAN interface to inject sensor data or receive actuator commands through the bus.

In this paper, we explore alternative architectures by restricting our choice to centralized or decentralized I/O (sensor, actuator)[3] and computing. In this context, a decentralized I/O architecture has a smart I/O's that translate an analog signal into a digital message and *vice-versa* and connect onto a communication bus while a centralized I/O approach would have direct I/O connections to the ECU. For computing, we may have different ECUs each carrying out a different task needed by the system at one extreme or a single ECU carrying out the entire computation at the other extreme. The trade-offs we explore are in terms of wire lengths, number of wires(also known as cut leads), number of packaged ECU's, signal latency and flexibility.

The following architectures were modelled in Metropolis that helped analyze the timing metrics such as bus utilization and signal latency. For wire length and total number of wires, an automotive electrical layout tool called Carchitect was used [4].

3.2.1. Mixed Decentralized I/O Centralized Computing (MDICC) Architecture

This architecture shown in Figure 2,represents a mix between a partially decentralized I/O structure with the hand wheel sensors and wheel speed sensors connected onto the serial data bus and a partially centralized computing structure with the supervisor and brake features allocated to one module.

3.2.2. Centralized I/O Centralized Computing (CICC) Architecture

This architecture shown in Figure 3, features a highly centralized I/O with a centralized computing infrastructure.

3.2.3. Centralized I/O Decentralized Computing (CIDC) Architecture

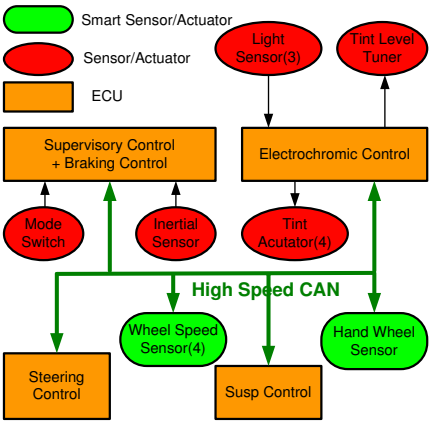

Figure 2. Mixed Decentralized I/O Centralized Computing Architecture.

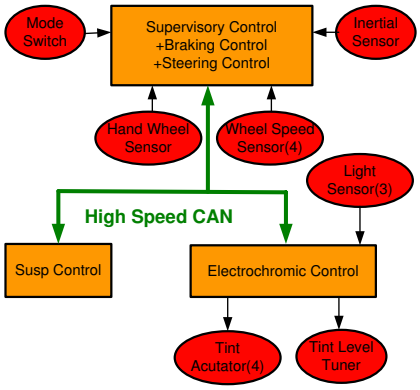

Figure 3. Centralized I/O Centralized Computing Architecture.

This architecture shown in Figure 4, features a highly decentralized computing infrastructure since each feature has a dedicated ECU. On the other hand, the I/O structure is centralized with all the I/O's directly connected to ECU's.

3.2.4. Decentralized I/O Decentralized Computing (DIDC) Architecture

This architecture shown in Figure 5, features decentralized computing and decentralized I/O.

3.3. Metrics

We analyzed the architectures with respect to the following metrics

1. Control Latency

2 For a more detailed description of this platform as modelled in Metropolis, please refer to [6].

3 The term I/O and Sensor/Actuator are used interchangeably through the paper and are synonymous in meaning.

4 Carchitect is an Architecture Layout tool developed internally at GM by Mike Allison, Alan Baum and Mira Supal. This tool should have been integrated into the Metropolis framework as a backend tool to perform layout and analysis. Due to time constraints for this paper, the modelling in Carchitect was manually derived from the representation in Metropolis. We will complete the integration soon

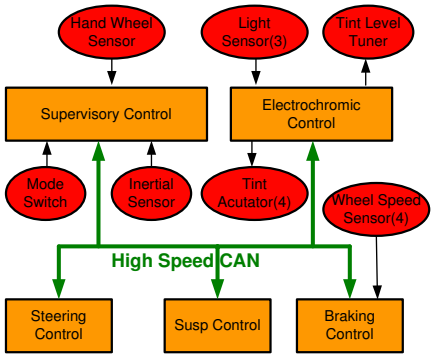

Figure 4. Centralized I/O Decentralized Computing Architecture.

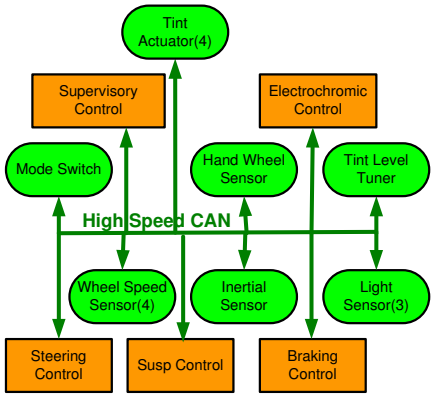

Figure 5. Decentralized I/O Decentralized Computing Architecture.

We are interested in control applications, where the electronic system typically interacts in a closed loop with the plant and the environment using sensors and actuators. An interesting measure of perfomance is given by the end-to-end latency from sensors to actuators. We measure the delay between sensing a change in the environment and the arrival of the corresponding control signal at the actuator function.

2. Geometric Metrics (number of cut-leads, wire length)

   We consider three geometric attributes: total wire length, number of cut leads[5] and of ECUs. These three attributes vary considerably according to the chosen architecture and are the greatest cost drivers for assembly and manufacturing costs.

3. Serial Data Metrics

   We compare the architecture alternatives with respect to how they use the serial bus. We measure the number of signals communicated locally versus the number of messages transmitted over the bus, the amount of data (number of bytes) in those messages, and the bus utilization (as % of the total bandwidth used for communication).

4. Flexibility

   Flexibility is defined as the degree of resilience an established architecture has with respect to specific future design changes. To evaluate flexibility, we define 11 different incremental design scenarios. Each architecture is evaluated with respect to these scenarios in terms of whether the architecture can accommodate the design scenario without any additional cost. The scoring is binary where 1 is for an architecture that can accommodate the scenario without cost and a 0 for an architecture that incurs a cost or at least changes in the hardware due to the design scenario.

   The following were the scenarios we used:

   - Removal of the electrochromic control feature including its corresponding S/A. I.e. a vehicle that has only the active stability control features.
   - Removal of two wheel speed sensors.
   - Removal of the light sensors. I.e. a feed forward control strategy for the electrochromic windows.
   - Removal of the active suspension control feature.
   - Removal of the active steer control feature.
   - Change of geometric location of one of the light sensors from the rear of the vehicle to the top.
   - Addition of a new node on the HS CAN bus housing a feature that requires wheel speed data.
   - Addition of a new node on the HS CAN bus housing a feature that requires mode switch.
   - Addition of a new node on the HS CAN bus housing a feature that requires light sensor data.
   - Change in data flow where the braking functionality requires the mode switch signal as well.
   - Change of geometric position of the inertial measurement sensor.

   We assume that each of these design scenarios have equal probability of occurrence, hence the weighting of the score is uniform.

4. Experimental Results

4.1. Control Latency

At the top of Table 1 we report the control latencies (in milliseconds). The first two rows report the time when a

5 A cut lead is defined as a contiguous point-to-point wire connection.

change in the sensed Steering Wheel Angle leads to a new value of the commanded Active Front Steering Angle reaching the destination middleware and then the actuator function. Similarly, the next two rows report the latency incurred from a change in the sensed lighting conditions to the corresponding actuation of the ElectroChromic Windows.

The general trend is that on a centralized architecture it is easier to achieve shorter latencies. There are at least two reasons for this

- communication over serial busses is generally slower than communication local to an ECU;
- it is possible to control the relative order of execution of functions within an ECU. Whereas, across ECU's in an unsynchronized communication network, there is no way of ensuring sampling alignment across the receiver and transmitter.

In distributed architectures, the lack of synchronization among ECUs leads to scenarios where the sampling alignment is so unfavorable that it introduces one-period delays at each sample. For example, looking at the first two rows of the last column we see that the control signal arrived at the destination middleware just past the periodic activation of the actuator function, namely at time 11.6182ms vs 10.0014ms. Therefore, the updated value will be read at the next periodic activation of the actuator, namely at time 20.0014ms.

Conversely, looking at the latencies of the Electrochromic Windows for the first three architectures, controlling the offsets can easily minimize the Control Latency.

4.2. Geometric Attributes

From the layout analysis (Table 1), architecture 3 (DIDC) had the shortest wire length and the least number of cut leads since the distributed I/O enabled the I/O's to be connected to the nearest bus wire rather than directly to an ECU located further away. Architecture 2 (CIDC) on the other hand had the longest wire length due to the geometric location of the ECU's with respect to the I/O's. Architecture 1 (CICC), although benefiting from the shorter wire length due to the fewer number of ECU's, suffered from a relatively large cut lead count due to all its I/O's being centralized. Architecture 0 (MDICC) is pretty much in between, with 49 cut leads and a wire length of 143.5.

4.3. Serial Data

In the results Table 1 there is a section comparing the architecture alternatives with respect to the serial bus usage. Clearly, the more distributed architectures make more extensive use of the bus.

4.4. Flexibility

Architecture 3 (DIDC) obviously was the most flexible since every feature and I/O was an independent component. Architecture 0 (MDICC) was resilient to 6 out of the 11 changes since its wheel speed sensors, hand wheel sensors and steering and suspension subsystems were modular. Based on the scenario's, it turned out that these features and I/O were the most sensitive to change.

5. Conclusion and Future Work

We analyzed a set of alternative architectures for an automotive case study using Platform-based Design and the supporting Metropolis environment. These architectures were evaluated by four different metrics, i.e. control latency, geometric attributes, serial data attributes and system flexibility. The overall results favor the architecture that is a mix between a centralized electrochromic window control system and a decentralized vehicle supervisory control system (MDICC). This architecture supports 6 out of the 11 design scenarios for flexibility, has low latency and bus utilization, relatively short wire length and small cut lead count. This was due to the architecture housing the window application being quite decoupled from the architecture of the supervisory control feature. Since wheel speed sensors and hand wheel sensors were sensitive to future changes and were geometrically complicated to route, having these as distributed I/O, was a key attribute contributing to the quality of the MDICC architecture.

In general though, due to the locations of the sensors and actuators, a centralized I/O configuration naturally caused a larger wire length since each I/O wire had to be routed all the way back to the target ECU. On the other hand, the decentralized I/O allowed the designer to route the signal coming from the I/O to the nearest bus connection, hence reducing the total wire length significantly. This is a sure advantage since wiring issues contribute for a large part to the assembly costs and warranty issues. The trade-off though is clearly the reduced latency in a centralized architecture while the decentralized architecture supports a geometrically distributed sensor actuator configuration and is generally more flexible.

This paper highlights the need for a virtual integration environment that allows the architect to take advantage of the architectural degrees of freedom and efficiently analyze the impact of the changes. Here, Metropolis was used as the master modeling environment while the back end tools such as the geometric layout design tool performed the analysis for other metrics. Changes in the design can easily be

	Architecture 0 (MDICC)	Architecture 1 (CICC)	Architecture 2 (CIDC)	Architecture 3 (DIDC)
Latencies				
SteeringWheel Angle latency (ms), arrival at destination middleware	11.3962	0.008	11.3962	11.6182
SteeringWheel Angle latency (ms), read by destination process	20.0037	10.0081	20.0038	20.0014
Light Adjusting latency (ms), arrival at destination middleware	0.2007	0.2007	0.2007	17.6645
Light Adjusting latency (ms), read by destination process	0.3003	0.3003	0.3003	20.0001
Geometric Attributes				
Cut leads	49	63	53	20
ECUs	9	3	5	20
Total Wire Length (grid-point)	143.5	153.5	165.5	106.5
Serial Data				
Local Signals	35	51	23	0
CAN Msgs	19	13	24	37
Total Msg DATA Length(bytes)	119	87	159	208
Bus Utilization	36.90%	26.14%	48.00%	68.06%
Flexibility	6	2	4	11

Table 1. Results Table

propagated through the integrated environment and quantitatively analyzed. The methodology and the environment are key enablers in providing a decision support system for OEM's to make good architectural choices early on in the design cycle and manage complexity.

Dependability, albeit an important aspect, was not considered in this paper. Intuitively, a decentralized system offers more availability than a centralized system that inherently has a single point of failure. Our group is also working on a design environment for automotive architecture exploration for safety systems [3][4]. We plan to extend the case study to include the dependability metric. In addition, a cost model that captures the impact of architectural decisions on the entire lifecycle of an architecture would be instrumental in giving the architect an informed way of navigating the design space. Another interesting avenue for research would be to explore the trade-off between a centralized communication infrastructure (e.g. a central backbone bus topology) and a decentralized topology (e.g. a meshed type topology).

6. Acknowledgements

The authors gratefully acknowledge Arkadeb Ghosal, Wei Zheng and Abhijit Davare from UC Berkeley for their advice on cost and flexibility analysis, and the Metropolis methodology. Paolo Giusto and Max Chiodo from the GM Berkeley Labs were instrumental in developing the architecture exploration methodology for automotive systems.

References

[1] F. Balarin, L. Lavagno, C. Passerone, A. L. Sangiovanni-Vincentelli, M. Sgroi, and Y. Watanabe. Modeling and designing heterogeneous systems. In *Concurrency and Hardware Design, Advances in Petri Nets*, pages 228–273, London, UK, 2002. Springer-Verlag.

[2] E. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computations. *IEEE Transactions on Computer Aided Design Integrated Circuits*, 12(17):1217–1229, December 1998.

[3] M. McKelvin, G. Eirea, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. A formal approach to fault tree synthesis for the analysis of distributed fault tolerant systems. *EMSOFT*, 2005.

[4] C. Pinello, L. Carloni, and A. Sangiovanni-Vincentelli. Fault-tolerant deployment of embedded software for cost-sensitive real-time feedback-control applications. In *Proceedings of Design Automation and Test in Europe*, Paris, February 2004.

[5] A. Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign*, February 2002.

[6] H. Zeng, S. Sonalkar, S. Kanajan, C. Pinello, A. Davare, and A. Sangiovanni-Vincentelli. Design space exploration of automotive platforms in metropolis. *Accepted by SAE Congress*, 2006.