Automotive System & Software Development Challenges - Part 1

Frank Schirrmeister, Cadence Design Systems - November 05, 2013

Editor’s Note: With the case of Toyota’s killer firmware still fresh, we offer this closer look at automotive system design feature. Part Two will continue with a deeper look at specific design issues.

Today’s high-end cars contain between 70 and 100 embedded processors and run up to 100 million lines of code according to the IEEE Spectrum article “This Car runs on Code.” Specialized cars, like Indy cars can have many more sensors and data acquisition/telemetry components to optimize for racing. Integration of all the hardware and software needed to make any car perform correctly is no small task. It takes a lot of simulation, modeling, verification and IP.

This article will summarize the development challenges from analog-mixed-signal simulation to proper system configuration as well as hardware software co-design and outline some solutions that are essential to successful system development across the design chain.

How complex can design for automotive be?
To understand the complexity in design for automotive, it is important to understand the design chain a little better. For comparison, consumer products that play in the “Internet of Things (IoT)” can have a design chain that is already pretty complex. The wearable sensor that records your movements and sleep behavior transmits this information by cell phone to servers that keep it in the cloud. As part of “Big Data Analytics” information will be provided such as “your deep sleep percentage last night has been in the top 20% of all users,” or “Wednesdays seem to be your days with the least steps taken, try to make an extra effort today.”

To develop and enable this, for the main functions there are 5 company types involved.

- **IP Providers** deliver to semiconductor companies some of the building blocks such as processors or graphics cores, peripheral blocks to connect to chip interfaces and on-chip interconnect.
- **Semiconductor companies** provide the silicon that is at the core of the sensor at your wrist, cell phone, the servers holding your information, and the networks that transmit them either wirelessly using your phone or wired once your data is in your phone providers network.
- **System companies** build the actual devices involved in this chain, your wristband, cell phone and the servers that hold your information. In this particular case, the system company providing your wristband to collect movement and sleep behavior, may run the servers holding the information to run Big Data Analytics as well, either themselves or using commercial infrastructure of a **Cloud Service Provider**.
- **Independent software vendors** contribute to the software powering this scenario with the tools running on top of Android, Linux or commercial OSs such as iOS and Windows Mobile.
Finally, the necessary wireless and wired infrastructure are run by **Network Providers** that are on the top of the chain and interact directly with the end users – you and me – by enabling the devices empowering the interactions described above.

Now consider the automotive application domain. The development of the cars transporting us daily to and from work and getting our kids to soccer practice, are at least as complex as the complete development chain described above, and then are themselves part of a bigger network in which they communicate geo-syncing locations via GPS and broadcast satellites, provide in-car wireless access points, embedded links to cellular that enable services such as OnStar as well as our smart phones, will in the future talk wirelessly to the infrastructure including traffic lights (V2I – vehicle to Infrastructure) and even will coordinate with other cars (V2V – vehicle to vehicle) to coordinate who goes first across an intersection. **Development Design Chains**

**Development Design Chains**
The automotive design chain and their responsibilities are illustrated in Figure 1. Tier 2 suppliers provide semiconductor components and software to Tier 1 suppliers that take the chips and integrate them into Electronic Control Units (ECU). The ECUs are integrated into the various different sub-systems that at the end of the day make up the actual vehicle.

![Figure 1. Tier 1 to OEM Design Chain and Software Responsibilities](image)

Figure 1 also outlines the relative ownership of software development. Automotive OEMs that provide the actual car are focusing more on applications and user experience, Tier 1 providers and sub-system integrators focus on task oriented middleware while Tier 1 providers focus on standard services, ECU abstractions and complex drivers. As part of their semiconductor deliverable, semiconductor companies need to provide basic abstraction layer software such as the MCU abstraction MCAL.

Not unlike in the wireless design chain, IP providers are supplying their IP cores in automotive to semiconductor companies – Tier 2 providers - that integrate them. While in the past semiconductor providers differentiated with proprietary processor cores such as the Renesas SH, NEC V850, Infineon TriCore or the PowerPC based Freescale e200, there is definite trend towards licensed processor cores such as the M and R series from ARM. The concept of ISVs supplying software exists
as well (considered Tier 2 as well) and so do OEMs. However, the existence of Tier 1 suppliers as an additional integration step is unique to the automotive and transportation design chains and underlines its sheer complexity.

**Design Chain Development Dynamics**

Similar to the wireless design chain described earlier, in which network providers articulate requirements down the chain into system companies translating them further to semiconductor companies and IP providers, the requirements are defined from the top down in automotive as well. Driven by the need to be able to flexibly port software between different MCUs and ECUs, standards like AUTOSAR (AUTomotive Open System ARchitecture) have been developed.

AUTOSAR is jointly developed by automobile manufacturers, suppliers and tool developers. Its objective is to create and establish open standards for automotive electrics and electronics architectures that will provide a basic infrastructure to enable the component based development of automotive software, user interfaces and management for all application domains. Besides integration and easy porting of software between suppliers, some of the key goals are the standardization of basic systems functions, scalability to different vehicle and platform variants, maintainability throughout the entire product life-cycle and software updates and upgrades over the vehicle's lifetime.

By separation of concerns and introducing a layered architecture that allows decoupling of functionality from supporting hardware and software services, AUTOSAR allows OEMs to define functionality independent of implementation to execute on AUTOSAR development stacks that hide the underlying hardware architecture. A basic software layer standardizes software that does not have any functionality but offers hardware-dependent and hardware-independent services hiding the hardware to the next layer above, the so called run time environment (RTE).

The RTE handles the information exchange between the application software components and connects the application software components to the appropriate hardware they execute on. The RTE decouples the application software components from the hardware and serves together with the basic software as a virtual function bus that separates with specific connections the application software components from themselves. The application layer is the layer where the actual functionality is developed. It is composed of application software components that interact with the run time environment.

The intended net effect of AUTOSAR in automotive is that OEMs and sub-system providers can define functionality independent of the implementation. The functionality can then be automatically mapped to AUTOSAR compliant applications. This is somewhat different from the wireless design chain in which requirements are still articulated using language based specifications and executable specifications are still an exception.

**Intra-Vehicle Networking and Different Types of Automotive Software**

Market researchers categorize car electronics into the segments “safety systems,” “powertrain & body electronics,” “infotainment” and “driver assistance.” Figure 2 illustrates different functions in a car and their requirements on how to deal with faults. It is clear that from a development perspective for hardware and software, different rules apply depending on whether a particular function is essential and needs to be fault functional, whether it is important enough to be fail safe or whether it is more fault tolerant as most of the telematics/infotainment functions are.
Different levels of in-car networks
The different levels of in-car networks – all of them in use - illustrate that the complexity of automotive developments is in itself as elaborate as other application domains are even if one includes for example the cellular network in the wireless application domain. For instance, in today’s cars you can find at least five different network types in use.

The **Controller Area Network (CAN)** is the most spread network in the car running at 1Mbps dealing with active safety, body and chassis, enabling functions like millimeter wave radar, dashboard, brake control and seat belt. Together these are part of the collision detection system.

The **Local Interconnect Network (LIN)** is a low cost bus for body applications at 19.2 KBauds with UART interfaces. It can be used for elements like the multifunction keyless system combining door control, light control, door mirror control and others. It is used for unlocking the doors when the driver approaches his vehicle and carries a key card, flashing the head and hazard lights, illuminating the interior and adjusting mirrors and seats to pre-set positions as defined by the driver, all indicating that the doors are unlocked.

The **MOST** bus for “media oriented systems transport” is designed for multimedia using optical fiber for up to 150 Mb/s. Ethernet, originally mainly used for diagnostics, is after the required extensions have been implemented, now also used for multimedia applications like transmitting the rear end video when parking.

**FlexRay** is a deterministic and secure high performance bus at 10Mbps mainly used in X-by-wire, Automatic Driver Assist (ADAS) and high performance applications. Good example applications are brake-by-wire applications combining brake and brake pedal control.

Just like different fault characteristics are important drivers to define software development requirements, they are a similarly important consideration to choose the appropriate interconnect.

**Traditional Development Techniques and Automotive**
There is actually not one single development engine, neither hardware-based or software-based, that serves all required use models and user requirements for hardware and software verification. Often
only a combination of engines will help users to get their verification and software development challenges done most efficiently.

Virtual prototypes are transaction-level representations of the hardware, able to execute the same code that will be loaded on the actual hardware (.hex and .elf files), and are often executing at well above 50 MIPS on x86-based hosts running Windows or Linux. To the software developer, virtual prototypes look just like the hardware because the registers are represented correctly, while functionality is accurate but abstracted. However, they are available earlier and offer a development environment that is observable, controllable, distributable, scalable and repeatable. They can be operated with the same tool chain of compiler and debugger as the hardware itself.

**Virtual prototypes**

Virtual prototypes are addressing the general trend that more functionality and effort is moving into software. There are a couple of requirements that are special in automotive. Designs have a very long life time and often hundreds of variants. As discussed above, the design chain is very heterogeneous and collaborative and deals with severe safety concerns, some of them leading to standard’s requirements like ISO 26262.

Combined with the long life time of designs in automotive, developers have to consider a very sophisticated legacy of hardware, software and processes. Outside the infotainment area, the designs itself are reactive and real-time systems that are closely interacting with the physical environment. To consider real world effects, testing needs to be performed with actual hardware of the ECU and the car.

For the actual chip design, IP providers and semiconductor companies use RTL simulation that executes the same hardware representation that is later fed into logic synthesis and implementation. This is the main vehicle for hardware verification and it executes in the Hertz range, but it is fully accurate as the RTL becomes the golden model for implementation, allowing detailed debug.

Acceleration/emulation executes parts of the design using specialized hardware—verification computing platforms—into which the RTL is mapped automatically and for which the hardware debug is as capable as in RTL simulation. While it is well adopted within IP providers and semiconductor providers, the development cycles in most applications have been long enough for Tier 1 suppliers and OEMs to rather rely on the actual silicon or FPGA based prototypes. However, shortening design cycles and increased requirements for testing, make emulation applicable for those users as well.

Interfaces to the outside world such as Ethernet, can be made using rate adapters. In acceleration, part of the design (often the test bench) resides in the host, while the verification computing platform executes the design under development. As indicated by the name, the primary use case is acceleration of simulation. In-circuit emulation takes the full design including test benches and maps it into the verification computing platform, allowing much higher speed and enabling hardware/software co-development, ranging all the way up into the megahertz range.

FPGA-based prototyping uses an array of FPGAs into which the design is mapped directly. Due to the need to partition the design, re-map it to a different implementation technology, and re-verify that the result is still exactly what the incoming RTL represented, the bring-up of an FPGA-based prototype can be cumbersome and take months (as opposed to hours or minutes in emulation) and debug is mostly an offline process. In exchange, speeds will go into the tens of megahertz range, making software development a realistic use case.
Also see:

- [Automotive System & Software Development Challenges - Part 2](#)

*Part 2 of this article will outline some specific design challenges around analog mixed signal and hardware software co-development, as well as discuss usage of virtualization in automotive.*

Read more [about the author](#).