ADDRESSING DESIGN CHALLENGES IN HETEROGENEOUS MULTICORE EMBEDDED SYSTEMS

WARREN KURISU
DIRECTOR OF PRODUCT MANAGEMENT

WHITEPAPER

Mentor Graphics®

E M B E D D E D   S O F T W A R E

INTRODUCTION

Heterogeneous multicore systems [architectures that combine two or more different types of microprocessors (MPUs) and microcontrollers (MCUs)] are quickly becoming the defacto architecture in the embedded industry today. The quick emergence of these systems can be attributed to a number of factors.

First, global trade barriers have been broken down allowing new technology to spread more quickly and as a result, the pace of innovation has accelerated which in turn is driving an increasingly competitive global market. With the emergence of this new global market, electronics manufacturers have many new options to stay competitive and new ways to compete that were never before possible. The global competitive pressures are driving OEMs of electronic products to look for new ways to differentiate themselves by cost, time-to-market, and product differentiation. This in turn, has driven the need to consolidate embedded designs in order to maximize reuse of software, to extend existing devices with new capabilities, and to reduce the overall size, weight, and power consumption of a system. The practical outcome of this consolidation trend is that modern designs are now being architected to include complex configurations of multiple operating environments onto complex and ever-more powerful processors.

In response to these trends, semiconductor manufacturers are now creating elaborate and sophisticated system-on-chip (SoC) architectures. These new architectures include mixed types of processing cores with increased capacity to perform highly complex and sophisticated operations. This trend continues to accelerate as evidenced by the processor roadmaps made public by all major semiconductor vendors.

This paper will examine the developmental challenges surrounding heterogeneous multicore SoC design. We'll take a look at the role consolidation plays today and why many of our current procedures and practices for embedded system design must be altered if we are to accommodate heterogeneous multicore SoCs and the complex software configurations that run on them.

FROM SINGLE-CORE TO MULTICORE DESIGN

There are many types of devices on the market today that include both single-core and multicore processor architectures. Single-core designs ruled the day when embedded devices were built to do one function only. With the advent of graphical user interfaces (GUIs), graphics processing units (GPUs), faster and more powerful silicon, and the need for a system to process multiple tasks simultaneously, designers have made the transition to multicore designs. Figure 1 shows examples of purpose-built devices that use low-power, single-core microcontroller SoC architectures. Examples of such devices include electronic medical devices or digital cameras. Each of these devices provides a very specific function, and typically requires very little power.

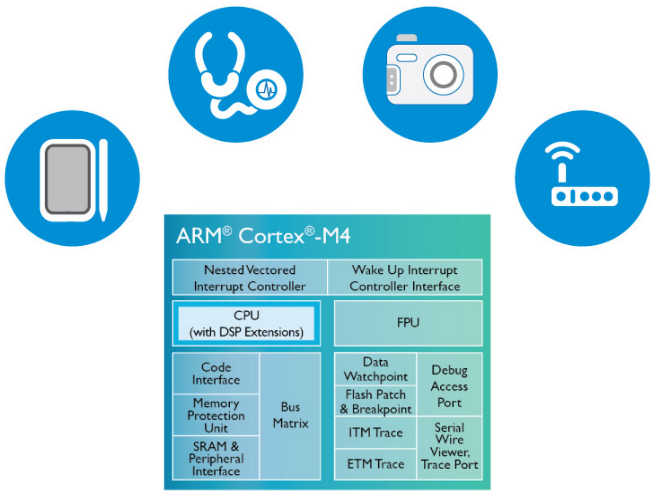
ARM® Cortex®-M4
Nested Vectored Interrupt Controller
Wake Up Interrupt Controller Interface
CPU (with DSP Extensions)
FPU
Code Interface
Memory Protection Unit
SRAM & Peripheral Interface
Bus Matrix
Data Watchpoint
Flash Patch & Breakpoint
ITM Trace
ETM Trace
Debug Access Port
Serial Wire Viewer, Trace Port

Figure 1: *Single-core MCU and end-user examples.*

Figure 2 depicts multicore devices that perform more complex functionality, which typically requires a multicore processor – and more power. Examples shown here include an automotive in-vehicle infotainment (IVI) system and a digital set-top box/gaming system. Other examples might include industrial controllers or applications needed in the mil/aero fields.

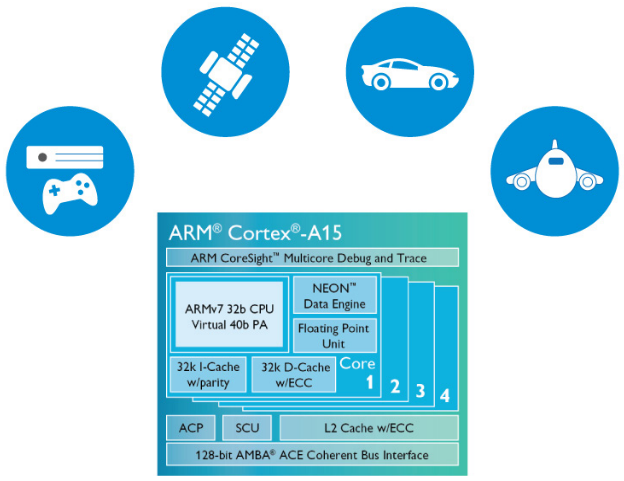

Figure 2: *Multicore MPU and end-user examples.*

EMBEDDED SYSTEMS TODAY

In order to address the challenges designers face developing multicore heterogeneous systems, it might be a good idea to take a brief look at a system in typical use today. As an example, let's use a hospital ecosystem comprised of different devices performing diverse functions. One device gathers patient data (patient data acquisition), another device aggregates and displays the retrieved data in the hospital room (patient monitor), and finally, another device allows remote users such as doctors, specialists, and lab technicians to access the patient's current status (hospital enterprise network, including remote access via the Internet).

Figure 3 represents some devices that perform specific functions such as gathering pulse and blood oxygen data, heart EKG data, and patient motion or sleep information. These devices generally run on their own low-powered microcontroller and perform a specific function which then feeds into the patient monitor system. These devices are often single-core systems using a small, real-time operating system (RTOS) or no operating system at all, running instead on bare metal.

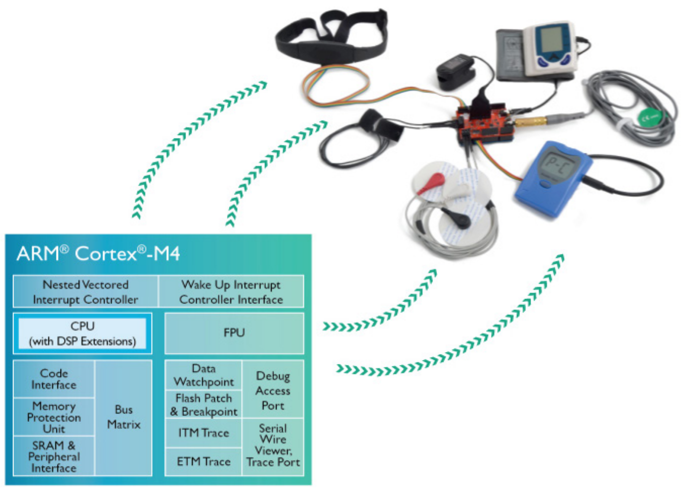
ARM® Cortex®-M4
Nested Vectored Interrupt Controller
Wake Up Interrupt Controller Interface
CPU (with DSP Extensions)
FPU
Code Interface
Memory Protection Unit
SRAM & Peripheral Interface
Bus Matrix
Data Watchpoint
Flash Patch & Breakpoint
ITM Trace
ETM Trace
Debug Access Port
Serial Wire Viewer, Trace Port

Figure 3: *Patient data acquisition system.*

The patient monitor (Figure 4) allows doctors and nurses to visually see the patient status, and adjust the parameters of devices that are connected to the patient. This device performs more complex functions such as processing the data, calculating and setting alarms and notifications, and providing a complex graphical and user interface to users. This device ties into the hospital network and may run on a general purpose operating system (GPOS) such as Android or Linux®.

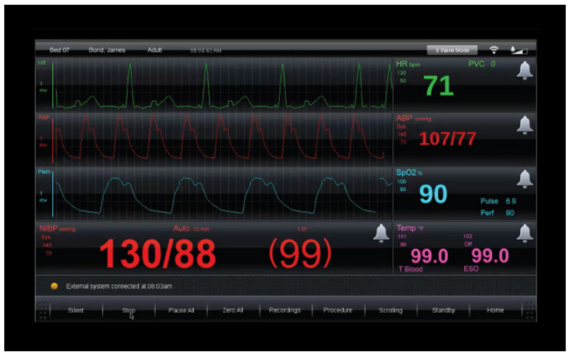

Figure 4: *A patient monitor system.*

The devices/systems depicted in Figure 5 serve as the hospital enterprise. Many of theses devices perform enterprise/network functions such as data aggregation, storage, and presentation of information across systems. The enterprise system also provides access to remote systems, work stations, and mobile devices such as iPads and mobile phones. This system, running on either homogeneous or heterogeneous multicore, typically leverages a GPOS such as Linux.

What do these terms mean?

Multicore *is a general term which simply refers to a processor with more than one core.*

Homogeneous multicore *refers to processor with more than one core, with all processing cores being identical. Simplified examples include a Windows PC which runs processors with multiple identical Intel cores, or an Apple iPad which runs an SoC with multiple identical ARM® cores.*

Heterogeneous multicore *refers to a SoC that integrates multiple non-identical cores. An example of this would be a system that integrates both a microcontroller (e.g., the ARM® Cortex®-M4 shown in Figure 1) and application cores (e.g., the ARM Cortex-A15 shown in Figure 2). Examples of heterogeneous SoCs available today are the TI Jacinto 6, Freescale Vybrid, or the Xilinx Zynq which incorporates ARM Cortex-A9 cores with an FPGA fabric.*

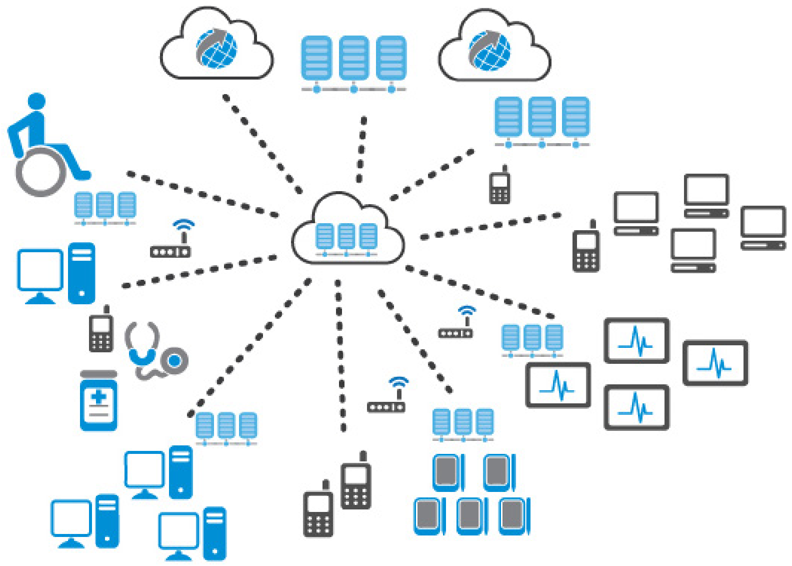

Figure 5: *Example of devices and systems within the hospital enterprise.*

Looking at the overall picture, there are three key takeaways from this hospital ecosystem example:

#1 – Each device runs its own operating system or operating environment, developed with tools that are designed for that operating environment and the particular application being implemented.

#2 – Each device runs on its own discrete processor and those processors are typically different. The type of application drives the processor selection, ranging from low-end microcontrollers to high-end application processors. Each component of the system has full ownership of all the hardware available to the component. Examples of that hardware include the processors, graphic processing units, memory, I/O, cache, etc.

#3 – The discrete components of this system are typically loosely connected. Each component boots independently and communicates with each other through messages over some physical connection. Each system component doesn't depend on what the other components do; they just need to communicate with the other components once they are booted and ready to communicate.

FROM MULTICORE TO HETEROGENEOUS

To help consolidate this hospital system, semiconductor manufacturers have responded to the demands of their customers by creating complex SoC architectures that combine heterogeneous cores and other devices; in essence, the silicon providers have been instrumental in the smooth and successful transition to heterogeneous system architectures. Further, the hospital ecosystem scenario is one example of how complex functionality of discrete devices can be consolidated onto a single SoC. An example of such an SoC is a TI OMAP5432 (Figure 6), which includes two ARM Cortex-A15 application processors, two ARM Cortex-M4 microcontrollers, an Imagination Technologies GPU, a DSP, and other processors. In addition to these different processing cores, there are also many other components on this SoC including memory, caches, I/O, security features, and more. These are the SoC architectures that enable consolidation, which in turn, allow companies to respond to the global competitive pressures.

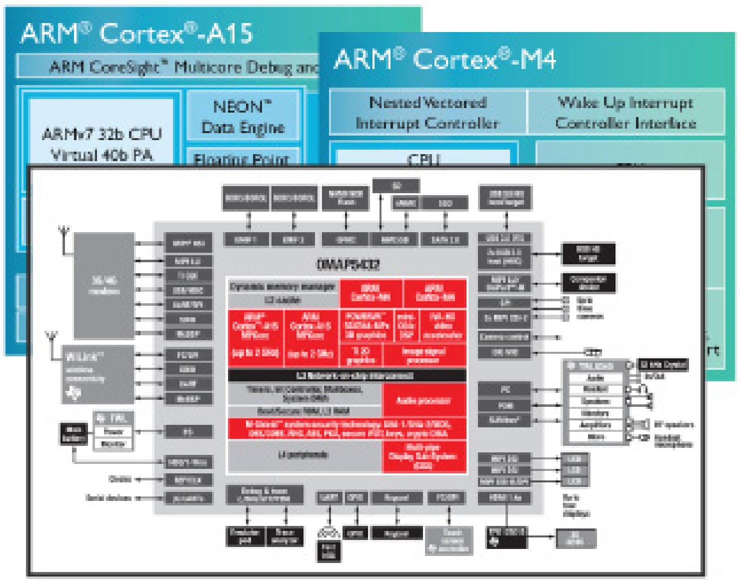
ARM® Cortex®-A15
ARM CoreSight™ Multicore Debug and
NEON™ Data Engine
ARMv7 32b CPU Virtual 40b PA
Floating Point
ARM® Cortex®-M4
Nested Vectored Interrupt Controller
Wake Up Interrupt Controller Interface
CPU
OMAP5432

Figure 6: *The TI OMAP5432 SoC, from multicore to heterogeneous environments.*

However, what embedded developers are discovering now is that with all this hardware goodness that enables consolidation, complex system development challenges are easy to surface.

DEALING WITH SYSTEM DEVELOPMENT CHALLENGES

When projects move away from discrete development of loosely coupled systems to an integrated heterogeneous environment, huge development challenges are likely to occur. These challenges did not typically exist during discrete development because developers were able to design, develop, test, and optimize within the confines of their own device, and they only needed to design and test the communication interfaces with other parts of the system.

But today with heterogeneous consolidation, there are a number of new challenges surfacing in which embedded developers must address. These include:

System architecture

With many heterogeneous cores on a SoC, one has a great many options to assign operating systems and applications to the processor cores, GPU devices, memory, I/O, and other resources that are now widely available and potentially shared. Developers need to be concerned about the architecture that optimally meets the system requirements.

Configuration

Not only do the architects need to think about the layout of the system, they need a way to configure the system. As is often the case, the initial proposed architecture may not actually perform the way the architects expected it to, so developers need to be able to rapidly re-configure the system and determine whether the system requirements can be met. Configuration can be a manual and slow process, wasting valuable development cycles.

Booting

In the discrete use case, each operating environment boots on its own hardware. In the use case on heterogeneous multicore, where multiple operating systems need to boot and typically in a particular sequence, the developer needs a framework and method to bring disparate parts of the system up in a coordinated manner according to system requirements and taking into account the shared nature of hardware on the SoC.

Debugging

When consolidating systems, developers and testers must find a way to view the system as a whole. They need to understand how each operating system and application environment is working. They need to understand where there might be shared resource contention, or saturation of processors, busses, or devices. They need a way to see how behavior in one part of the system affects, or is affected by, behavior from another part of the system. Developers need a way to optimize the overall performance of the system.

Separation

The designer needs to have some assurance that if one part of the system fails due to bad programming or malicious intent, other parts of the consolidated system will not be affected, or that the entire system is not compromised or does not fail altogether.

Device sharing

With potentially numerous OS environments (and applications consolidated into a single system), hardware devices needed to service these functions may be limited. This might require that limited hardware resources be shared. The architect and developer need a way to share these devices in a manner that ensures the individual functions are not impacted.

Inter-Processor Communication (IPC)

The integration of many applications onto a single SoC requires a means to allow those applications to communicate with each other and with the system. Given the heterogeneous nature of consolidated systems, this drives the requirement for an IPC framework that scales across different applications including open source software and proprietary software environments where the protection of IP is paramount.

Security

With discrete architectures, system architects can integrate a separate function within the embedded device that connects to the outside world and isolates any external malicious attack from the rest of the system. During consolidation, those security functions could be consolidated onto a heterogeneous multicore SoC, but the shared nature of hardware and other devices on the SoC creates additional challenges to contain the attack.

FINDING THE ANSWERS

As you can see, the challenges are tremendous and very complex when it comes to heterogeneous consolidation, which is why Mentor Graphics has developed an industry-first solution that delivers a comprehensive commercial set of runtime environments and tools for heterogeneous multicore development.

This solution responds to the complex development challenges that today's developer faces when developing heterogeneous systems on modern heterogeneous SoC architectures. Key points of the Mentor multicore heterogeneous offering include:

CONSOLIDATION OF THE SYSTEM INCLUDING DISCRETE COMPONENTS

Mentor allows developers to configure and deploy multiple operating systems and applications across heterogeneous processors. Today, developers moving to heterogeneous architectures are trying to map existing development methodologies used for creating discrete components to heterogeneous design. This does not work. Configuration of the system needs to be done in the context of the system, since these discrete components are now being consolidated onto a single SoC with many shared hardware components.

SIMPLIFIED BOOTING

Booting a heterogeneous system is also not as simple as booting an OS on a dedicated processor. One needs a way to manage the booting of operating systems across the various cores, and to manage the applications that run on those processors. For example, performance requirements may dictate a certain boot order of the components. Mentor provides a set of capabilities to manage the booting of operating systems and applications across heterogeneous cores. In the Mentor multicore offering, this is addressed through the support for the remote processor framework (remoteproc) which can be used for Mentor® Embedded Linux®, Nucleus® RTOS, and even bare metal implementations.

SYSTEM-WIDE COMMUNICATION

In discrete loosely coupled systems, the only communication required was at the application level over a physical connection such as serial or Ethernet. With consolidated heterogeneous systems, the system provides shared resources that can be used for communication, but no framework exists to enable the communication across heterogeneous components running on heterogeneous processors. The Mentor offering includes a "clean-room" implementation of rpmsg/VirtIO enabling developers to architect systems with functions that can seamlessly communicate between open source components and proprietary environments in a way that protects the proprietary environments from exposure to open source licensing concerns.

VISUALIZATION DEEP INTO THE SYSTEM

Developers need to be able to visualize how the heterogeneous components interact with each other in consolidated heterogeneous systems. Because the systems are consolidated on shared hardware, the chances of running into resource contention and bottlenecks are greatly increased. Developers need tools that can help them identify those contentions and bottlenecks, and to enable them to quickly find solutions to the problems. In the Mentor heterogeneous offering, Mentor Graphics® Sourcery™ CodeBench (which includes Mentor's Sourcery™ Analyzer) is integrated in a way that allows various OS and hypervisor runtimes and the applications running on them to be visualized on a single common timeframe allowing developers to have a system-wide view of how the various components are behaving. Further, they can see how various components affect each other, and how shared resources can be oversubscribed or undersubscribed. Essentially, developers can analyze and optimize the entire system.

DEVELOPMENT TOOLS FOR OPEN SOURCE AND PROPRIETARY ENVIRONMENTS

Consolidation increasingly includes a mixing of open source and proprietary environments on a single heterogeneous SoC. This drives the requirement for the development tools, booting mechanisms, IPC frameworks, and visualization tools to work seamlessly across both open source and proprietary operating systems and applications.

THE MENTOR GRAPHICS HETEROGENEOUS SOLUTION: A WALK-THROUGH

Step 1:
Let's first start by focusing on the TI Jacinto 6 hardware in the middle of Figure 7. This simple example includes the use of two ARM Cortex-A15 application processors and two ARM Cortex-M4 application processors.

Step 2:
Notice that one configuration might be to run a single instance of Linux on each of those processors, as shown by the two green blocks (Mentor® Embedded Linux®) at the top left of this diagram.

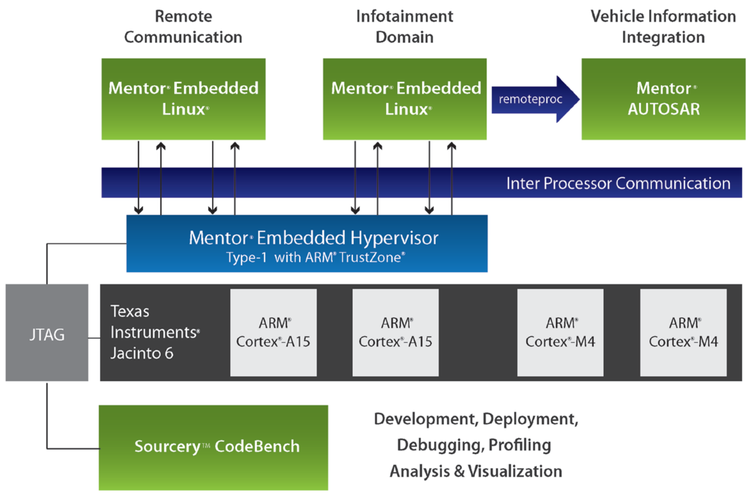
Remote Communication
Infotainment Domain
Vehicle Information Integration
Mentor® Embedded Linux®
Mentor® Embedded Linux®
remoteproc
Mentor® AUTOSAR
Inter Processor Communication
Mentor® Embedded Hypervisor
Type-1 with ARM® TrustZone®
JTAG
Texas Instruments® Jacinto 6
ARM® Cortex®-A15
ARM® Cortex®-A15
ARM® Cortex®-M4
ARM® Cortex®-M4
Sourcery™ CodeBench
Development, Deployment, Debugging, Profiling Analysis & Visualization

Figure 7: *The Mentor Graphics comprehensive heterogeneous multicore offering.*

Step 3:
Now look at the hardware box again and notice there is an ARM Cortex-M4 microcontroller on the right. A typical configuration might be to run a real-time operating system, such as the Mentor Nucleus® RTOS on this core. The Mentor solution provides a mechanism by which the Linux instance in the middle can remotely load the Nucleus RTOS and its applications on the Cortex-M4 core, and boot that core. This is an example of the earlier point about booting in a coordinated manner across cores.

Step 4:
We now have two Linux instances and their applications running on each of the ARM Cortex-A15 cores, and a Nucleus RTOS instance and its application running on the ARM Cortex-M4 core. The applications need to communicate with each other. Mentor provides the Inter-Process Communication (IPC) mechanism denoted by the broad horizontal box below the OS boxes. This rpmsg IPC mechanism allows both the open source components and the proprietary components to communicate with each other within a common, seamless IPC framework.

Step 5:
Note the bottom boxes labeled Sourcery™ CodeBench and Sourcery™ Analyzer. These are the common tools that allow developers to configure, develop, deploy, debug and analyze the system. Recall that in the discrete device development scenario, the tools are typically disparate, inconsistent, and unconnected due to the specific environment being used for each discrete development effort. In the Mentor solution, this common tool environment provides the capabilities to successfully develop consistently in a consolidated heterogeneous environment, with a system-level perspective.

Step 6:
Finally, locate the box labeled Mentor® Embedded Hypervisor. Embedded systems design has been adopting, and continues to increasingly adopt, virtualization technologies into designs, such as the Mentor Embedded Type-1 Hypervisor. These technologies are typically used to provide and enforce separation between OS instances, partition and enforce access to hardware (e.g., memory, I/O, devices) and also provide sharing (virtualization) of hardware. Recall previously in this paper about the challenge of ensuring separation in consolidated systems so that one OS instance failing due to bad programming or malicious intent doesn't corrupt the entire system? This is one critical function of a hypervisor. All of the capabilities addressed in this example work with or without a hypervisor, providing developers the maximum flexibility of choice and options to meet their specific design goals.

CONCLUSION

Driven by business trends consolidation of systems with heterogeneous operating environments on heterogeneous SoC architectures are on the doorstep. OEMs, device manufactures, and design teams need to get on the right side of this curve. No doubt, traditional development methodologies will be disrupted as more heterogeneous systems are introduced.

Mentor Graphics is uniquely positioned to support the breadth of solutions across heterogeneous cores. The company takes great pride in claiming that it is the only embedded systems supplier with a complete portfolio of operating environments and tools that provide the flexibility and choice required by developers including those developing heterogeneous systems on heterogeneous multicore architectures.

For details on specific solutions and/or products, please visit the Mentor Embedded Systems website.

Author's biography

Warren Kurisu is the Director of Product Management in the Mentor Graphics Embedded Systems Division, overseeing the embedded runtime platform business for Nucleus, Mentor Embedded Linux, Automotive Technology Platform, AUTOSAR, and virtualization technologies. Warren has spent nearly 30 years in the embedded industry, both as an embedded developer and as a business executive, working broadly in industries including Aerospace, Networking, Industrial, Medical, Automotive and Consumer. Warren holds a master's degree in Electrical Engineering from the University of Southern California and a Master of Business Administration from the University of California at Berkeley.