Trident Multiprocessor Operating Environment

Supporting Multiprocessor Embedded Systems Development

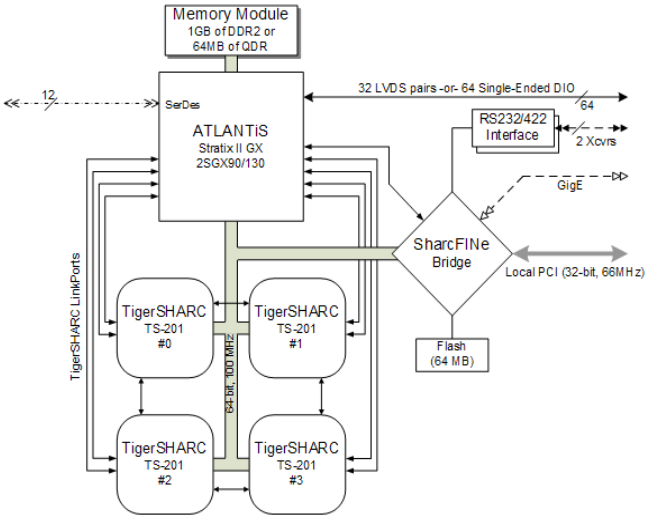
By Dr. Kenny Adamson, BittWare, Inc.

January, 2007

In general, the use of tools such as an RTOS and a multiprocessor operating environment will provide for a more reliable programming methodology, higher real-time predictability, and shorter development time. This is especially true when targeting the parallel hybrid solutions called for by today's complex applications. This article explores the software requirements of embedded applications and discusses how vendor-supplied tools assist developers in attaining optimal computing solutions with specific reference to BittWare's family of hybrid COTS boards.

The Ever-Changing Face of Embedded Applications

Modern high-performance COTS environments typically marry a modular processing core to industry-standard bus interfaces such as VME, cPCI, PMC and PC/104. Along with the core processing elements provided within most COTS architectures, there is an ever increasing proportion of the processing being located within at least one associated large FPGA. For example, the processing core of BittWare's GT family combines the raw performance of 4 Analog Devices processors with an Altera Stratix II GX FPGA (as shown to the right). Such hybrid (processor + FPGA) architectures have the potential to provide unparalleled performance thanks to numerous processing elements that can act concurrently; however, the success of any application is dependent on the developer's ability to program such an environment efficiently.



Embedded Software Challenges

DSPs began as simple devices designed to process single streams of data with fixed data-rates. They have since matured into much more powerful devices capable of processing multiple data streams with differing data-rates across multiple channels. The first devices were rudimentary and the early signal processing applications were programmed using assembler or C code to directly interface with

the hardware. The growth in processing capability has led to an increase in the sophistication of signal-processing applications as well as the overheads necessary to maintain the state of such devices. A moderately complex application must manage the underlying hardware as well as undertake the task for which it was designed. In practice this means maintaining a finite-state representation of the processor and 'time slicing' the execution of each task. This becomes even more difficult if tasks are faced with delays or have uneven durations. Add to this the problem of managing a number of separate processors concurrently to allow the seamless communication of information and control signals in an efficient and transparent manner, and you are faced with a non-trivial system before the application is actually coded. Developers are either faced with the prospect of doing it all themselves or using a tried-and-tested vendor tool.

How do Operating Systems Help?

An operating system kernel supports sophisticated applications by directly managing the state of the underlying hardware. The developer uses a more simplistic model for software development based on the concept of a thread. This is a single stream of control that undertakes a specific job; typically there are a number of them within an application. Each thread can be thought of as working independently and it is the job of the kernel to interact with the hardware and to ensure that each thread is granted sufficient execution time whilst masking any latency within the system from the developer. Threads seldom exist on their own therefore the kernel must also provide prioritization mechanisms and thread synchronization primitives. The threaded development model vastly simplifies the process of creating and maintaining new applications as well as porting them to other platforms.

There are a wide variety of commercial and open-source operating systems in existence though embedded applications favor the use of a specific type called Real-Time Operating Systems (RTOS). These provide a much more specialized environment than that of a general purpose operating system, such as Microsoft Windows or Linux. In an RTOS, the time taken to perform an action is as important as the results of the action itself. Such systems can be further categorized into soft real-time systems, where a late result can be tolerated, and hard real-time, where a late result can be catastrophic. It is imperative that such systems are predictable.

Real-time systems respond to stimuli occurring at different times, which is impractical to handle using purely sequential programming techniques. Consequently, real-time systems are typically designed as a set of concurrent, cooperating processes. It is very important to fully support the management of these processes and provision of an execution platform is a huge benefit. In any case we have a number of common problems that must be resolved. Task scheduling, which decides when and why tasks should be run; policing of the use of resources to prevent damage and corruption (mutual exclusion); and we need task communications facilities (synchronization and data transfer). The design environment therefore should remove the burden from the developer and screen the complexity of the hardware system from the programmer.

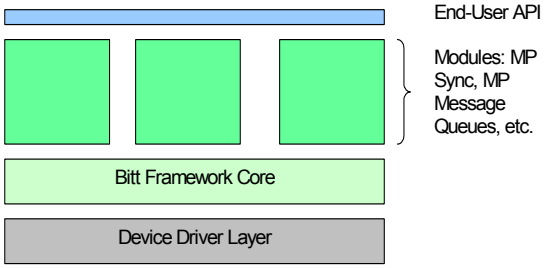
The DSP processor within BittWare's GT family is the Analog Devices ADSP-TS201. There are only two RTOS' in existence for this processor including Visual DSP++ Kernel (VDK). This is a real-time kernel provided by Analog Devices across their family of digital signal processors. Complex applications are represented as a series of concurrently operating tasks, or threads. A thread possesses its own stack(s) and can share resources or synchronize with other threads through the use of kernel signaling mechanisms such as semaphores, events, messages and a device flag. Each thread has a priority that the kernel uses to determine execution time on the processor. At run time the thread with the highest priority that is capable of running is given time on the processor until it explicitly relinquishes it's time or a higher-priority thread becomes available (as a result of a hardware event).

Crossing the Processor Threshold - the Multiprocessor Operating Environment

VDK provides an efficient and effective means of writing threaded applications on a single processor; however, communication between processors is not

explicitly supported. Multi-processor software development adds another level of complexity to the problem, requiring application developers to address issues such as how the processors are interconnected, how threads on different processors should be addressed and how to route information between them efficiently. BittWare's hybrid architecture COTS boards with their ATLANTiS technology provide unparalleled multiprocessing performance and I/O capabilities. VDK provides a real-time operating system for applications development on a single processor, but this picture lacks a software technology for 'gluing' multiple processors together, either on the same board or across multiple boards.

Trident is BittWare's proprietary Multiprocessor Operating Environment designed to support all the processing elements found within their hybrid COTS technology. It incorporates VDK's preemptive scheduler and has been designed to integrate fully with the operating system. Trident consists of a number of embedded libraries, plus a plug-in for the Analog Devices VisualDSP++ development environment.



Trident's embedded structure is illustrated in the figure . It has a layered, modular design consisting of a number of components whose sizes are configurable to ensure that they maintain a minimal footprint within TigerSHARC memory. The lowest layer consists of the device drivers that govern the processor hardware. The upper layers of Trident interact with the device-drivers for inter-processor communication. Direct access of the hardware by the developer is unnecessary. The next layer consists of the Trident framework. This is the central component that provides the core messaging and mapping services across all processors in the network. It is also responsible for synchronizing the system when Trident first boots up, and managing all other services. The underlying inter-process communication supporting each Trident module occurs across high-performance link-ports, either directly connected or configured via a technology such as ATLANTiS. Messages are based on small quad-word transfers with optimal DMA-based transfers for memory blocks, which results in a fast, robust, and extremely adaptable message-passing framework capable of supporting a potential network of 65,000 processors.

The Trident framework builds an abstraction of the distributed network and calculates optimal routing paths across it that the other modules communicate with. All of this work is hidden from the developer and the modules and applications that use it can be reconfigured for different network topologies without having to be rewritten. This feature ensures that applications based on Trident are transparently scalable. Trident provides a multiprocessor operating environment within which boards and processors are interconnected and threads positioned to provide optimal data-flow solutions, thereby allowing the developer to concentrate on partitioning the application at a functional level.

On top of the framework are a number of services, or modules, that user-applications are built upon. Three modules are initially supported: Multiprocessor Synchronization (MPSync), Multiprocessor Message Queues (MPMQ), and Continuous Data Flows (CDF). Each of these APIs can be run within the processor network without explicit knowledge of the underlying network configuration, which is built using the host-side tools at design time.

The final result from Trident is a build which consists of all of the elements required to configure a project to work within a multiprocessor environment. This build will address topological configuration, which provides a description of the embedded network of processors and how they are interconnected; Trident Object creation, declaring the objects to be used in the application; and the creation of Trident projects; which will generate the VDK projects containing Trident-enabled threads, the processors they are located on, and the objects they will use. Synchronization is an essential activ-

ity within a multiprocessor environment; therefore, Trident's MPSync module is mandatory, along with the framework and device drivers. The other modules are optional and can be omitted to reduce Trident's memory requirements. The plug-in determines which modules are required based on the objects within the build and adjusts the VDK project to add the appropriate libraries and header files.

Putting it All Together

The tight integration between the VisualDSP++ environment, the VDK, and Trident, facilitates rapid application development (RAD). This offers huge commercial time-saving and code re-use benefits avoiding the need for the hand creation of all of the control code. The use of automatic code generation allows the developer to concentrate on the algorithms and the desired control flow rather than on the implementation details. The environment supports the use of C, C++, and assembly language, and BittWare also provide an extensive range of hand optimized libraries (TS-Libs) which integrate within this development environment. This encourages and supports the development of highly readable and maintainable code. BittWare's Trident allows a thread to control and interact with a device in a portable and hardware abstracted manner through a standard set of APIs. Trident provides all the communication mechanisms using a single API which means that the application developer no longer has any need to develop these low-level facilities themselves. No ISRs, no reading/routing threads - just Trident.

Conclusion

It is clear that the embedded marketplace is characterized by an irrevocable trend towards greater software complexity. These complexities not only complicate the world of embedded software developers, but also create financial pressures to complete designs on time, meet design expectations, and meet product windows of opportunity. Software development as a process for embedded system design has changed very little over recent years. The advent of object-oriented languages, tools, and related design techniques have enabled greater reuse of software. However, their use in embedded design has been limited due to real-time requirements and the need to program as close to the hardware as possible. The most significant improvements in embedded software have been associated with the software tools evolution. The selection of supporting tools has broad implications in determining the total product cost and time-to-market. In general, the use of tools such as an RTOS and an MP Operating Environment will provide for a more reliable programming methodology, higher real-time predictability, and shorter development time. This is especially true when targeting the parallel hybrid solutions called for by today's complex applications. Trident, created to ease the development of MP applications, handles all of this complexity while also providing a common development platform essential for team development. This unique transparent multi-processing operating environment, coupled with vendor support, software design tools, and documentation, allows for the most beneficial use of strategic engineering resources.