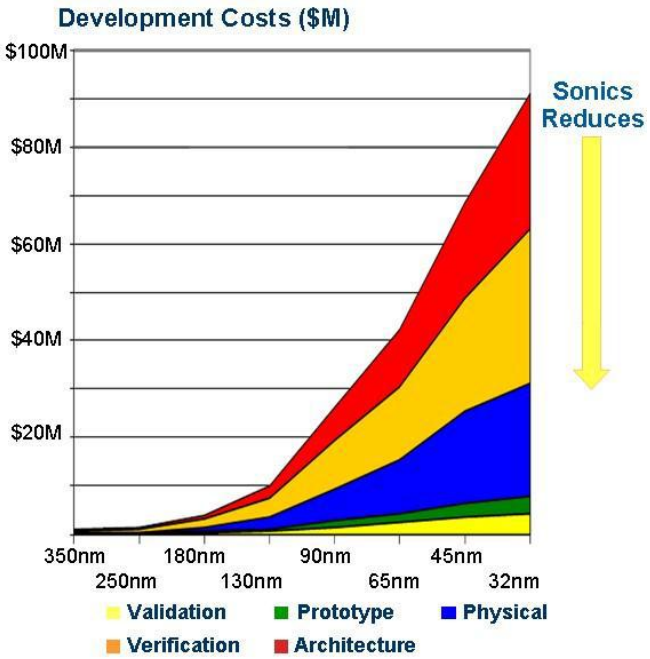
Using On-chip Networks to Minimize Software Development Costs

Today's multi-core SoCs face rapidly escalating costs driven by the increasing number of cores on chips. It is common to see code bases for chips that exceed 5 million lines of code. These lines of code are distributed among various processing elements often containing multiple operating systems, combinations of hardware-specific firmware, and high-level application code. The software integration task has become a huge compatibility and verification challenge with real-time consequences to the performance of the device. Software teams race to create finely tuned systems only to have much of their work undone by last minute core or standards changes.

There are some fundamental issues in multi-core SoC designs that drive the complexity of the software task. Among these we can list the following:

1. The lack of a centralized core that can monitor and manage all tasks on a chip.
2. Real-time constraints that vary from core to core. Three aspects with differing design constraints influence this facet: latency, bandwidth and isochronous flows.
3. Traffic contention at the memory interface that is difficult to predict. This is caused by varying data chunk sizes and access methods as well as demanding bandwidth requirements.
4. Error recovery and diagnostic capabilities competing with gate count requirements and in-band timing requirements.
5. Late- or fast-changing market requirements driven by new standards and rapid consumer obsolescence.

Sonics portfolio of on-chip networks have built-in features and services that minimize schedule impact and facilitate a software team's ability to adapt to all these issues.



1. Working Around the Lack of a Centralized Core

Heterogeneous SoCs lack a centralized processing element that tightly manages all of the tasks being performed on the chip at any given time. From one generation to another, the architecture of the chip may change to add more hardware acceleration or to incorporate a new standard. For example, the TI OMAP 2430 chip is capable of running multiple OS's and has 33 devices on board that can contend for attention. TI describes this challenge[1] as follows:

New complex multimedia and advanced connectivity applications are becoming available on mobile phones and are expected to grow exponentially over the next few years. In addition, new complex usage models are expected to emerge through the combination of existing and new applications. End-users will run multiple applications in parallel, whether they do it consciously or not. In all cases they will expect the handset to perform with little or no degradation of performance and with instantaneous response. Some of these new usage models are:

- Watching live TV while receiving an incoming phone call, which requires concurrent:
  - DVB-H radio tuner
  - H.264/WMV video decode
  - 64-voice polyphonic MIDI ring tone
  - 3G cellular network connection and live call
- Video conferencing while recording, which requires concurrent:
  - MPEG4 or H.264 video encode and decode
  - AAC+ audio encode & decode
  - MMC record
  - 3G cellular network connection
- Over the air synchronization while listening to MP3 songs, which requires concurrent:
- Synchronization protocol
- MP3/WMA audio playback
- 3G cellular network connection

Trying to create a balanced software system that contemplates all the possible use models of the on-board hardware is a daunting task. This is further aggravated if late changes are made to the protocol standards or an underlying algorithm proves to take more or less time than that allocated to it.

The hardware and software trade-offs that are made during the architecture phase are key to the success of the project. Understanding how the software can make use of hardware acceleration while still maintaining coherent alignment with the operating system and the application software is vital in order to partition the system correctly. Code re-use can only be facilitated if the hardware partition complements the application software architecture. Sonics on-chip connectivity solutions provide multiple services that allow the software development team to divide and conquer the task. The modular nature of the software partitioning allows for much simpler system integration, while minimizing the risk of introducing significant performance degradation.

Core decoupling has various facets that allow software to maintain data flows under extreme data contention conditions. Core de-coupling can allow the following:

a) burst types to be intermixed without problems with automatic packing and un-packing
b) dedicated or shared threads with non-blocking flow control for concurrent access
c) data width conversion
d) protocol conversion (AMBA to OCP or vice-versa)

These advanced interconnect hardware features allow each software driver to directly access the on-chip resource it needs to accomplish its task without being affected by other software tasks trying to access the same or other on-board resources. With the mixed burst support provided by SonicsSX®, each software task can use the most efficient burst type to process data in its native and most efficient modes (see Table 1).

2. Managing Real-Time Constraints

The firmware task is intimidating due to two types of processing requirements: asynchronous (latency sensitive) and synchronous (both isochronous and quasi-isochronous). The first is the typical low latency

requirement that results from such accesses as those required by a processor cache miss. The second requirement has a different set of constraints. The value of the data to be accessed has a limited lifespan as time progresses. For example, if the data relates to the processing of a frame buffer in a high-definition video application, the value of the data goes to zero once the processing is not accomplished in the right timeframe. Synchronous data can be quasi-isochronous when the task is regular in time, but not periodic (interrupt request processing) or has a variable length (media compression engine).
A good example of synchronous data requirements is an H.264 decoder data flow as shown in Figure 1 and Figure 2.

Figure 1: H.264 Data Flows

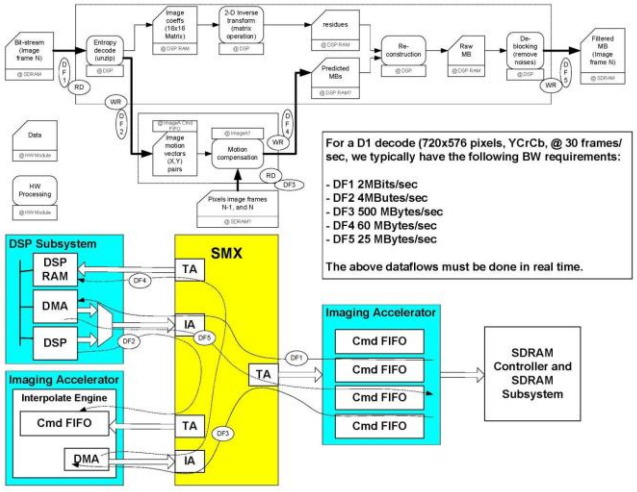
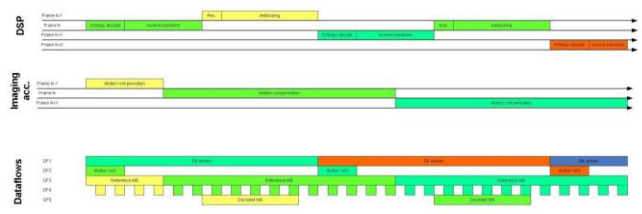


Figure 2: Decoding Window



Sonics on-chip networks inherently integrate precise quality of service (QoS) arbitration at various points in the system to ensure that requests are all processed according to the programmed levels on the device. Run-time programmability allows for dynamic allocation of various bandwidths if an aggressive dataflow behavior is required. The hardware control of the accesses to critical resources allows the software to bypass all requirements and ensure that the appropriate bandwidth is being allocated to each task. Starvation avoidance is built into all Sonics interconnect solutions, preventing bandwidth- intensive threads from monopolizing a resource. A wealth of hardware options allows the SoC architect to finely tune performance independently from the software algorithm development to achieve the result expected in both successful operating conditions, as well as times in which a temporary resource shortcoming is present in the system.

Table 1 QoS Modes

Table 1 QoS Modes

Mode	Level	Thread Characteristics	Thread Priority
Weighted	Best effort	Weighted fairness (where enabled) and least recently serviced within and across thread.	NA
Priority	Best effort	Weighted fairness (where enabled) and least recently serviced within and across thread.	lower
	Priority	Weighted fairness (where enabled) and least recently serviced within and across all priority threads. Priority threads always have the highest priority.	higher
Controlled	Best effort	Weighted fairness (where enabled) and least recently serviced within thread.	lower
	Bandwidth allocation	Rate-based control on each thread Weighted fairness (where enabled) and least recently serviced within thread. If demotion occurs, weighted across threads	medium
	Priority	Rate-based control on each thread. Weighted fairness (where enabled) and least recently serviced within thread. If demotion occurs, weighted across threads	higher

Changing out a core or the introduction of a new standard can be addressed efficiently by simply re-establishing the correct resource allocation and understanding the QoS requirement. The software can be written as though it were completely independent from other on-board code.

3. Optimizing Memory Usage

The contention at the memory choke point is caused by multiple factors. There are cores that have long

bursts at infrequent intervals (for example, a cache line fill for a processor) with extremely tight latency requirements. There are cores (such as the H.264 decoder example above) that will need a large two dimensional array of data in order to manipulate pixel data, as the requirements for these cores tend to favor high locality in the memory.

Some additional fundamental data burst requirements are shown in Table 2. Allowing the software the flexibility to use the most efficient burst type provides a natural efficiency gain.

Table 2 Burst Efficiency

Table 2 Burst Efficiency

Burst Type	Use Model
XOR	for critical word first cache line refill
INCR	for sequential memory accesses, good for scan line writes on video streams
UNKN	for peripheral register grouping
DFLT1/DFLT2	for user defined sequence, good for DECR address
BCLK	for 2-D burst in video processing
WRAP	WRAP is good for cache line fill

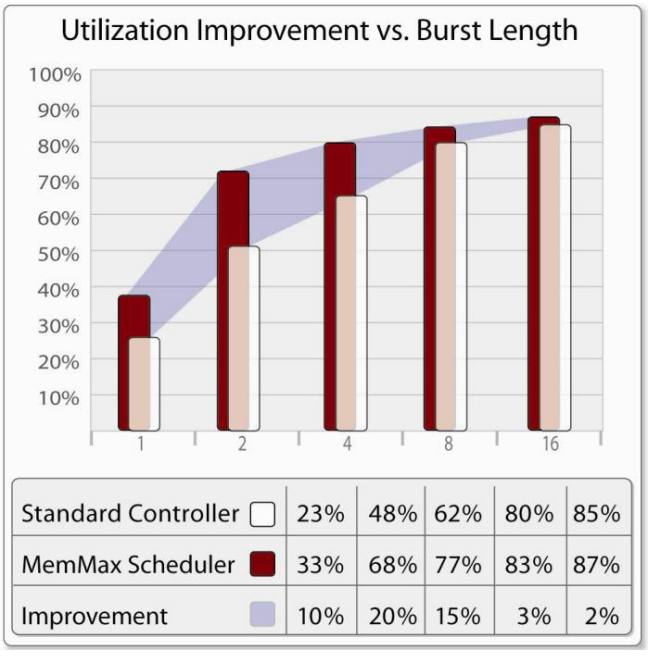
Issues such as varying priority accesses and bandwidth requirements, memory locality and write vs. read requirements all affect the performance of the system. This creates an intrinsically challenging memory contention equation. Traditionally, software solutions to these problems have resorted to complex memory maps or memory access algorithms that try to group the accesses in specific patterns. They have had, however, limited success as data interleaving tends to destroy any firmware control of the addressing modes.

While various cores use software structures that match the algorithms being developed, these do not always result in the most efficient memory access due to physical characteristics of the DRAM. For instance, a core may like to process data as a two-dimensional array. The natural memory request would be a 2D request with a given width and height. However, this request may not be aligned to the physical memory (banks and pages) connected to the chip.

The software team is then tasked with creating a memory allocation routine that is hardware-aware or that must understand the physical portioning of the memory chips in order to issue a series of bursts that map efficiently to the memory. Any change in the memory subsystem has a direct and obvious impact on the performance of the device. For example, in one project a network security processor chip was challenged to meet its throughput requirements. After four weeks of debug and almost cancelling the project, the correct memory allocation mechanism was finally found: it doubled the performance of the encrypted stream being processed.

All of these considerations can be abstracted from the software task using Sonics MemMax® Memory Scheduler. The chart below shows how MemMax can improve the utilization of the available memory bandwidth independently from the software task.

Figure 3 Memory Utilization



The improved utilization at the memory interface coupled with fine grained QoS control allows the system to perform at peak efficiency while isolating the software implementation from physical attributes present in the memory subsystem.

4. Leveraging Error Recovery and Diagnostics

Trying to bring up a multi-million gate chip always presents a design challenge. Error recovery and diagnostic capabilities provide useful data but they add gates to the design. Cost pressures on the project make it expensive to insert specialized hardware diagnostic capabilities and a mechanism to pipe the data off-chip. Furthermore, error detection must occur in real-time to pinpoint any firmware bugs.

Software bring-up and debug can be simplified with the error detection, logging and recovery incorporated in Sonics' on-chip networks. SonicsSX provides facilities that allow for: agent status reporting, including primary and secondary error reporting; and burst request and response timeouts with single or multiple condition reporting. With these built-in hardware features, debugging software programming problems is accelerated as the cores that experience communication problems are explicitly reported and logged.

5. Addressing Rapidly Changing Requirements

The 802.11 standard is one example of a specification that has evolved significantly to provide additional benefits for the user population. Market leaders in this space have capitalized on their ability to release devices simultaneously with the standards release. Broadcom has achieved a 70% market share by being first to market, commanding a premium for its devices while the competition struggled to catch up. Software teams must be capable of absorbing changes like these within the hardware infrastructure on very short schedules to capitalize on a true time-to-market edge.

The hardware services present in SonicsSX, Sonics3220™ and MemMax allow software developers to create software drivers that are architecturally consistent. Sonics' on-chip connectivity solutions have register-based control points that allow software drivers to use discovery mechanisms to run the correct code suite for the device. This software can then safely bring up the rest of the system facilitating field firmware upgrades. The introduction of this register-based programming inside the SoC interconnect, therefore, creates a new "conduit" by which software developers can perform system management while dramatically reducing the complexity of accomplishing these tasks.

6. Conclusion

Lack of scalability in typical SoC architectures often requires re-design from chip to chip, creating technical discontinuities in the SoC architecture. This trend provides no foresight for software developers, and as a result, forces major structural changes to software code as product lines are re-engineered. As a result, software portability costs skyrocket.

Sonics' on-chip connectivity solutions enable SoC architects to rely on proven interconnect technology that contains all the data flow management services required. The ability to configure, verify, and model the interconnect and the predicted data flows for the SoC early in the architecture research enables developers to understand what performance they should expect and how they can manage the system resources during these activities. This provides a reliable "look ahead" capability flexible enough to change during the chip development cycle, so that software developers can more readily plan and execute system management schemes upfront, while significantly minimizing re-engineering efforts as the chip is developed.