Multi-Threading for Efficient Set Top Box SoC Architectures

This paper introduces the challenges in building set-top box architectures and describes how some of these challenges are met using an underlying hardware multi-threaded processor. The benefits offered by multi-threading transcend performance and provide additional significant benefits for designing an efficient system overall, encompassing hardware as well as software aspects. This paper describes some of these advantages in detail.

Document Number: MD00545
Revision 1.00
February 2, 2007

Contents

1 Introduction

Set-top box (STB) manufacturers are facing significant challenges today in designing the hardware and software system architecture. These challenges include, for example:

- Growing performance requirements due to additional features.
- An increasing variety of standards that must be supported in the same box: MHP, OCAP, DVB-T/S, etc.
- Cost pressures leading to new price/performance points which are hard to reach with traditional architectural approaches.

These challenges apply to both entry-level products as well as full-featured products, while the latter are particularly impacted with an increasing convergence of features and services. And moreover, these designs must complete with a minimum time to market.

An effective way to address these challenges would be to use a multi-threaded processor core in the system design. A few salient features of hardware multi-threading as applicable to this discussion, and as implemented in the MIPS32® 34K® core family are:

- Duplication of user architecture state, that is, thread contexts (TCs) in hardware, reducing the context switch overhead to zero.
- Dup.ication of system architecture state in hardware allowing multiple virtual processing elements (VPEs) to share the same computing resources.
- Fine grained context switching, that is, the ability to issue instructions from different TCs in consecutive cycles, allowing hardware to fill stall cycles arising from any cause.
- User-definable and user-programmable thread scheduling policy allowing QoS for real-time tasks.

For a more complete description of the MIPS MT architecture or the 34K processor core, please refer to the appropriate documents.

For STB products, multi-threading not only offers performance benefits, but also a cost reduction in the overall system. It can also improve the power and energy efficiency of the system. The MIPS® MT (multi-threading) ASE (Application Specific Extension) offers quality of service (QoS) scheduling policies for threads. Using this QoS feature can provide real-time guarantees in hardware and improve the behavior of the product.

For software, multi-threading offers flexibility in the effective use of the available computing resources, and this type of flexibility can be used to layer applications such as audio, JVM (Java Virtual Machine), and security on top of the operating system. This reduces development time and decreases time to market for the product in question.

An additional benefit comes from the zero cycle hardware context switching, which allows fast interrupt processing and hence a reduction in the latency of response to an interactive task. For example, in a PVR, the time from when a user presses a button on a remote control to the response as seen by the user in terms of the redrawing of the screen is an essential part of the user experience for that product. An important metric in a set-top box is the time to draw an EPG (Electronic Program Guide) or render a browser page. With the overall performance benefits of multi-threading including the latency reduction from fast context switching, this drawing time can be minimized for a better end product.

This document first expands upon the general advantages offered by multi-threading for hardware and software before discussion of the specific topic of Set-Top Boxes. The document proposes new system partitioning options to reach new price/performance points. This analysis is based on the MIPS® 34K® core family and its unique multi-threading capabilities.

2 SoC Partitioning with Multi-Threaded Processor Cores

When a new SoC architecture for a STB is being developed, there are advantages to using a multi-threaded processor in one of two possible ways, listed here:

1. Two or more non-multi-threaded processors can be replaced by a single multi-threaded processor, or

2. A single non-multi-threaded processor can be replaced by a multi-threaded processor.

In either case, the use of a multi-threaded processor opens up new possibilities for system-level optimizations and software efficiencies. The two sections that follow provide details for these two possibilities. This chapter also includes a section on the performance that can be obtained with a multi-threaded architecture. The final section of this chapter deals with the quality of service feature and its benefits.

2.1 Replace Two Single-Threaded Cores with One Multi-Threaded Core

This scheme replaces two single-threaded processors with one multi-threaded processor like the MIPS 34Kc™ processor. In this situation, in the original design, one processor would typically serve as the control application processor running Linux or a similar operating system. This processor’s main task is to control the overall system functions on the SoC. In current designs this processor is also increasingly executing more and more Java-based applications.

The second processor is typically running tasks that are sensitive to real-time response, and which will require QoS (Quality of Service). QoS may be required for guaranteed throughput, for example an audio application like Dolby Digital, or for guaranteed low latency, for example a voice application. This processor may also run security tasks like Digital Rights Management (DRM).

The key benefits of replacing the two single-threaded processors by one multi-threaded processor like the 34Kc processor core are:

- **Lowered price/performance point:** This comes primarily from a cost reduction due to smaller chip area, reduced royalty, and elimination of multiple development tool chains for heterogeneous architectures. With two processors, there is non-negligible communication overhead, which has an impact on performance and area. The interconnect hardware as well as software contribute to the overhead with either a coherent or a non-coherent architecture. Having a unified tool chain can also shorten time to market since software designers have a smaller overhead with a single tool-chain which can be used to create and debug software on a single platform.

- **Simplified and more flexible partitioning of computing resources:** The computing resources of a single multi-threaded processor can be efficiently allocated as needed across the applications. Examining the overall instructions per cycle of typical control applications, it is clear that half or more time is spent idling, waiting for memory or some other long latency event. Using two processors then is very wasteful resulting in gross under-utilization of computing resources. Multi-threading the tasks on the same processor fills the idle slots with effective use of all resources on the processor. The QoS-sensitive tasks will still be guaranteed their needed compute resources through the core's QoS logic and thread policy manager. Extremely critical code can be loaded and locked in the level-one cache or stored in scratchpad memory, although this is not unique to multi-threading, the 34K processor core allows for this possibility.

- **Lower power consumption:** The higher pipeline efficiency due to multi-threading reduces the overall power consumption per executed task. Avoiding the use of a second processor reduces overall power consumption since the leakage current from a second processor is eliminated.

2.2 Replace a Single-Threaded Core with a Multi-Threaded Core

The key benefits from replacing one single-threaded core with one multi-threaded are:

- Higher performance from the more efficient single pipeline, and
- Better quality of service (QoS), due to real-time scheduling.

If higher performance is not the main focus, then another benefit is reduced power consumption and/or reduced cost. Because of the higher pipeline efficiency due to multi-threading, the processor operating frequency can be lowered or the clock can be switched off once the compute task are completed. There is potential area reduction when the core is synthesized at a lower clock frequency.

There are two very different scenarios to consider, based on the tasks being performed by the single-threaded processor core:

1. **An processor running both the operating system as well as real-time sensitive tasks is replaced with a multi-threaded processor:** Using a multi-threaded implementation such as the MIPS32 34Kc processor core, the operating system and the real-time sensitive tasks can be separately executed via different Virtual Processing Elements (VPEs) to imitate a multi-processor solution. Alternately, these tasks could also execute on the different hardware Thread Contexts (TCs).

2. **A processor executing either QoS sensitive tasks or executing an operating system including, for example, Java applications, is replaced by a multi-threaded processor:** Application tasks can typically be partitioned either manually or automatically into multiple threads. In the case of Java, the operating system can automatically tie applications executed on a Java virtual machine to TCs.

2.3 Defining a New SoC Architecture using a Multi-Threaded 34K Core

The definition of a new SoC architecture is always a significant challenge and undertaking, whether needed for a next generation product requiring more features at a lower cost, or for entry into a new market. The required features have to be mapped onto hardware and software functional units. Multi-threaded processors, because of their additional performance, QoS policy, and the ability to provide task isolation, can take on more features at almost negligible additional cost. Moreover, when there is an existing set of hardware modules or software libraries, the flexibility for re-partitioning these components is higher with multi-threading.

In an SoC solution that uses asymmetric multiprocessors, the performance requirement of each individual processor core must be pre-determined in a situation where the standards and feature sets are changing. This can lead to over-estimation when allowing for the potential extra headroom, leading to costlier end-products. In this type of situation, if the multi-threading processor can handle the processing needs of the asymmetric multiprocessors, then it offers a significant benefit since it eliminates the need to accurately estimate the performance needs of each individual core. With a multi-threaded solution, only the overall performance requirements need to be estimated to design the chip. A multi-threaded solution eliminates the need to move applications off an overloaded digital signal processor to the main processor or vice versa. It also eliminates the need to shift security-type applications off the secure processor to the generic processor. The flexibility of a multi-threaded processor as the single compute resource may not only eliminate a redesign and speed up the product availability, but it may also give the semiconductor product a larger market window. A broader product portfolio could be generated as well.

All the benefits of multi-threading discussed so far also apply in the context of designing a brand new SoC architecture. Briefly, these benefits include, the higher pipeline efficiency due to multithreading that reduces the overall power consumption per executed task. A second core can be often avoided which saves leakage current. The frequency and voltage of the processor can be lowered to save power. Clock and power can be switched off after task completion while the core is idle. Transistors with a higher threshold voltage can be used to reduce leakage current.

2.4 Performance Advantages of a Multi-Threaded Processor Core

Multi-threading can be used to efficiently hide memory latencies caused by cache misses. It can also be used to hide the latency of any operation, for example, peripheral access on an I/O bus, non-cached multi-cycle memory accesses, or multi-cycle latency multiply operations. Peripheral access latency are often not negligible. Non-cached multi-cycle memory accesses are sometimes used for data structures and buffers modified by another master or for sparse data access patterns while preserving the contents of the data cache. In high frequency synthesizable designs, the pipeline stages are deep enough that operations such as multiplies and divides may stall the main issue pipe long enough that efficient multi-threading schemes can fill those idle cycles with useful work from other threads. Also the bubbles caused by mispredicted branches can be filled in this way. The 34K core has the ability to do fine-grain context switching on the active threads, and hence is capable of filling any stalls cycles in the core as long as there are other active threads in the system. Hence the 34K core can fill not only the long latency stalls such as cache misses, but it can also fill stalls that happen for other reasons such as data dependency, multiplies, synchronization, etc.

In a single threaded system where instructions are primarily executed from the instruction cache, the overall system performance depends, among others, on the instruction cache miss rate and the number of cycles required to fetch the instruction so the CPU can continue execution. The instruction cache miss rate depends on the application, primarily on the spatial and temporal locality of the instruction accesses, and the size and configuration of the cache.

When the cache miss rate in the system is very low or negligible, then the majority of the pipeline stalls are caused by pipeline dependencies, long latency operations like multiplies and divides, stalls due to thread synchronization, or stalls from branch misprediction. In this situation, the multi-threaded processor that can fill these stall cycles with useful instructions from another thread would result in better overall performance than a single-threaded processor that is unable to fill these stall cycles. Having the ability in the hardware to fill branch misprediction stalls is important since these are impossible to schedule around unlike stalls due to say multiplies and other data dependencies.

The dark blue line in Figure 1 shows how in principle the system performance is affected by the cache miss rate. The shown behavior is valid for the instruction cache as well as for the data cache. The performance degradation also depends heavily on the memory sub-system behind the cache. A high performance fast memory system has lower degradation. The implication of performance degradation due to memory latency is the under-utilization of the CPU pipeline and other resources due to stalling.

In a multi-threaded system, the CPU stall cycles from one thread are typically filled by instructions from another thread. The red line of Figure 1 shows this scenario. The more threads that are available, the higher the CPU utilization. But this reaches a point of diminishing returns as the number of stall cycles gets smaller, saturating the pipeline with the growing number of threads. And when all the threads are using the same memory sub-system, including caches, then saturation can occur much faster, leaving the CPU still somewhat under-utilized.

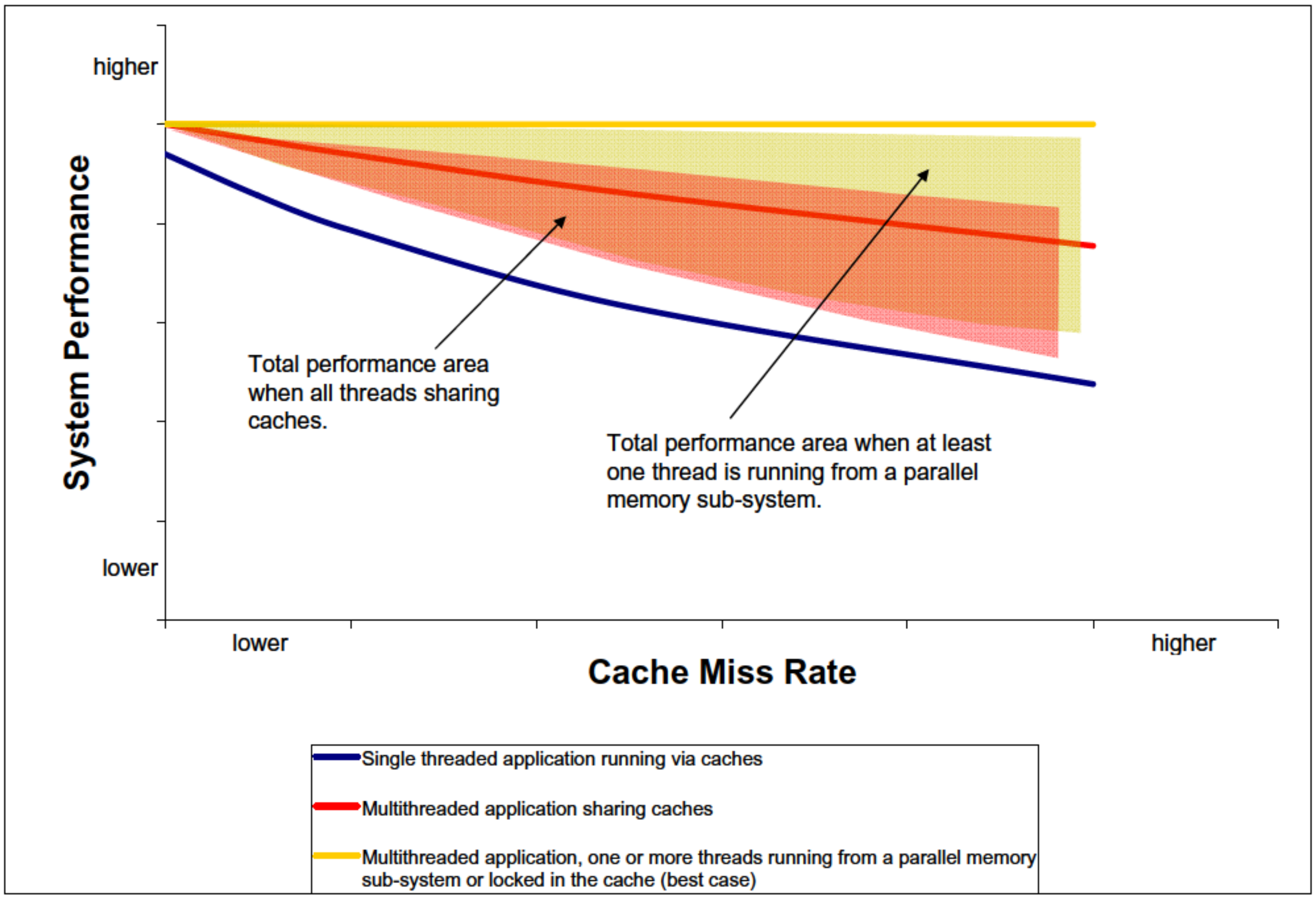

Figure 1 Effective System Performance Depending on Cache Miss Rates

The red line shows the principle relationship between the overall system performance and the cache miss rate. The overall performance is dependent on several factors like the cache miss rate of individual threads, cache thrashing and interference from multiple simultaneously executing threads, as well as cache line sharing among multiple threads. The typical performance area is shown in red. The expected run-time jitter for each thread depends on the application. In most common scenarios, the measured jitter is within acceptable limits, and results in reasonably stable and predictable overall behavior.

If the cache miss rate of even two of the threads increases further, the dark blue line and the red line will meet each other. In worst case conditions, these lines can even cross, if cache thrashing effects become dominant. This undesirable condition is not shown in Figure 1, because the system should not be architected in this manner. But this can happen if the system is not used properly. Such cache effects can also be encountered in single threaded systems, if attention is not paid to caching effects.

A further increase in CPU utilization can be reached when at least one of the threads is executed from a parallel memory sub-system. This could be another cached or uncached memory sub-system, for example, ISPRAM (Instruction ScratchPad RAM) or DSPRAM (Data ScratchPAD RAM). The instructions and data could also be locked in the cache. Here, under theoretical ideal conditions, the whole CPU performance can be used up, shown by the yellow line. It can be reached if, for example, a background thread located in the second memory sub-system uses up all remaining performance. The more realistic performance area for most applications is the yellow area. The expected run-time jitter of each thread is lower if a different memory-subsystem is used. This approach could be beneficial for very QoS sensitive routines.

The best application profile for this approach occurs when one task of the application has compact code and/or data size. This could be for example, a computation-intensive signal processing task that spends most of its time in an FFT loop which fits in the SPRAM. If the computation-intensive task requires a significant portion of the overall available performance (e.g. 35 percent or more), then the memory sub-system for the other tasks can be simplified, for example, the cache sizes may be reduced, without reducing the overall performance.

Note that performance gain by overcoming stalls due to pipeline dependencies is not discussed in detail in this section. When task behavior has been characterized, for example, as computation-intensive, or branch-intensive, then this information can be used to guide the software partitioning decisions. Alternately, if the IPC (Instructions Per Cycle) for each task is measured along with its cache workload characteristics, then this information can also be used in the software partitioning decisions.

2.5 Quality of Service

Multi-threading can increase the quality of service. There are three possibilities:

1. Using a background thread to monitor throughput-oriented threads and to give the thread more cycles if needed, using whatever mechanism that exists for controlling thread scheduling. In the case of the 34K core, this could be the weighted round-robin policy manager implemented in hardware as described below.

2. Assigning critical tasks to individual threads with individual TCs for immediate response. Also, two critical tasks can be executed interleaved and can respond significantly faster than if they are executed one after another.

3. Using the hardware thread scheduling policy supported by the MIPS MT architecture. This allows the user to define the policy in hardware which is software controllable using programmable registers. The 34K core implements a hardware policy manager for both a round robin as well as a weighted round robin scheduling mechanism. The core also allows the addition of a new hardware policy manager like earliest deadline first or guaranteed percentage scheduling, if the two examples provided are not adequate to meet the specific needs of an application.

QoS improvements in general:

There are some ways to guarantee that hard real-time threads meet their deadlines. Note that most of these techniques apply to both single-threaded as well as multi-threaded processors.

1. Give these threads sufficient performance headroom such that the worst case runtime jitter for any thread is covered without causing a glitch in the real-time response. This headroom is often less than the accumulated headroom needed for traditional multi-core solutions. This is because the worst-case performance peaks do not appear simultaneously in a statistically probable environment and hence do not need to accumulated when computing the worst-case margins on a single processor core.

2. The needed headroom can be reduced by dedicating cycles to a critical thread via pure hardware or mixed hardware/software policy managers. MIPS MT provides the flexibility of optimizing these policy managers on a per system basis. By using a mechanism such as the policy manager, the need to utilize the DMT (Disable Multi-Threading) instruction to switch into single threaded mode can be reserved only for absolutely critical situations.

3. The system designer can optionally separate the memory subsystems and interrupts of very QoS sensitive components from the others to get lowest runtime-jitter:

- **Handling instruction access:** The best solution is the use of a separate ISPRAM. A less efficient solution that does not require an ISPRAM would be to lock down critical cache lines or to reserve a cache way for a specific VPE. This technique can provide in general a major performance gain and improvement in the QoS.
- **Handling data access:** Similar to the instruction side, a dedicated DSPRAM provides the best solution. This can work well since sometimes data may be streaming, for example audio and/or video, and sometimes specific data may not be cached anyway, for example frame buffers.
- **Handling interrupts:** The easiest way to handle this would be to dedicate a VPE to a QoS sensitive thread. The interrupt signals to this VPE and hence the thread, are separated from the interrupts to the other VPE. But most significantly, interrupts to the other VPE do not block the QoS sensitive VPE. It is possible, however, that an operating system may block the other VPU in some circumstances such as when it disables multi-threading in interrupt service routines. This situation needs to be avoided or minimized.

3 SoC Partitioning for Set Top Boxes

For next generation Set Top Box platforms, there are many options for partitioning the hardware and software components. This partitioning decision is influenced by several factors; whether it is an entry-level or a full-featured product, whether it covers satellite, cable, or ethernet only, and whether there are existing hardware and software components.

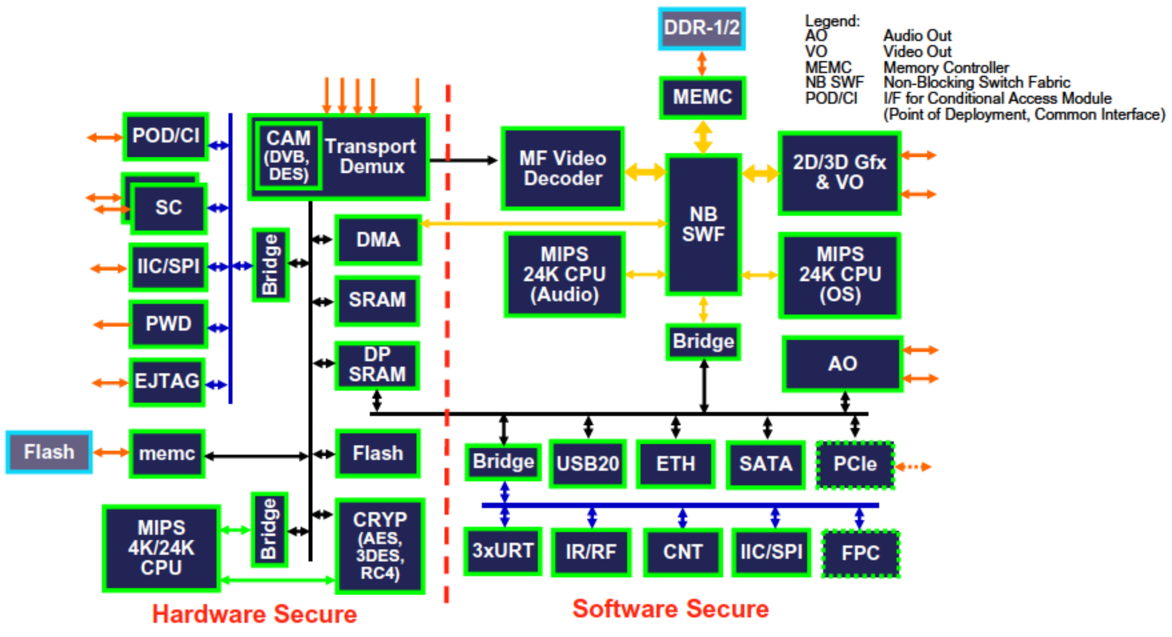

Figure 2 STB Example - Using Single Threaded MIPS Cores

Figure 2 shows a modern set top box SoC architecture without a multi-threaded CPU. The vertical dashed line separates the security sensitive front end, shown on the left side from the main processing components shown on the right. The system CPU running the OS is on the right side. A non-blocking switching fabric connects the system CPU, video decoder, graphics engine, audio, memory, front end, and the peripherals. High-end devices may provide two memory controllers for two banks of external DDR memory to meet data bandwidth needs. The number and type of peripherals vary depending on the end product definition, for example, not every product needs PCIe and a hard drive.

The right side has limited access to the security sensitive front end on the left. The system CPU cannot access the peripherals and memories on the right side, except the dual ported communication RAM. It cannot configure the DMA unit.

3.1 Module View

There exist a three options where a multi-threaded 34K core can give major advantages:

1) Executing audio, the operating system and control functions on a single 34K core.

2) Executing only audio functions on a 34K core.

3) Executing only the operating system and control functions on a 34K core.

3.2 Executing Audio and the OS on a Single MIPS32® 34K™ Core

Especially for entry-level systems, it is highly desirable to merge the audio decoding functionality into the system CPU as shown in Figure 3.

The audio algorithms can be executed on a separate TC or even better a separate VPE using its own low-overhead real-time OS or on bare iron (no operating system). The benefit of doing this is the cost savings from eliminating the dedicated audio decoding engine, saving chip area and potentially lower royalty cost. The overall computing resources can be better utilized if it is a shared core, rather than having two cores with unutilized remaining resources. This has a beneficial lowering effect on the energy efficiency of the system.

This is an ideal solution for highly integrated low-cost set top boxes, especially for customers new in this field, who do not have an audio decoder or customers who want to eliminate the existing audio decoder.

The QoS requirements for the audio processing can be guaranteed by an appropriate system configuration, as mentioned in the QoS section earlier. It is usually sufficient to provide a small amount of performance headroom and regulate the execution of the audio thread via a software managed weighted round-robin policy manager. Depending on the buffer depth for the audio and depending on the performance headroom available, it may also be useful to analyze whether a separate ISPRAM is needed. A separate VPE instead of a TC also has the major benefit that VPEs have dedicated interrupts and do not block each other.

3.3 Executing only Audio Functions on a 34K™ Core

When using the multi-threaded 34Kc core for audio only, the audio processing tasks are split into multiple threads. Audio encode and decode streams should be separated from each other for maximum efficiency. Also, different audio processing steps, like Dolby Digital decoding, SRS TruSurround XT®, sample rate conversion, bass management, etc. can be separated into different threads for optimal performance. These audio processing algorithms typically work in a cascaded fashion, with the data generated by the back-end of one algorithm being used by the front-end of the next algorithm in the sequence. When correctly configured to execute with pipelined parallelism, the resulting data sharing in the cache between different threads leads to additional performance improvement. The obtained performance gain is dependent on the exact algorithm mix and their interaction, but a performance gain of over 30% can be easily achieved, depending on the system configuration. Further details can be obtained from the MIPS document MD00535 titled “Increasing Application Throughput on the MIPS32® 34K™ Core Family with Multithreading”.

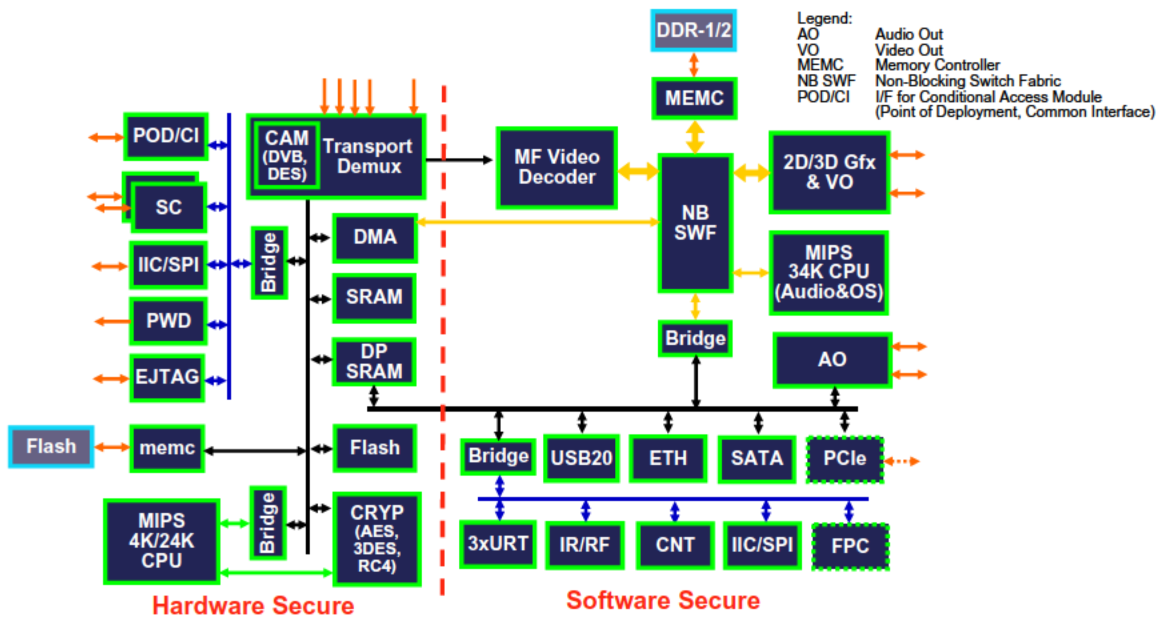

Figure 3 STB Example - Audio and System/OS merged using a MIPS 34K Core

3.4 Executing only the OS and Control Functions on a 34K™ Core

There are two benefits provided by the 34K core. First, the JVM (Java Virtual Machine) can utilize hardware multi-threading. Typically, the JVM relies on the software threading library, for example, Pthreads for thread services. Pthreads (POSIX threads) is a C and C++ based library that provides standard APIs for multi-threading under Linux. Any application that is written using Pthreads can have the software threads automatically tied to hardware TCs by an operating system such as SMTC Linux that supports hardware thread contexts on the 34K core.

Secondly, services which require a higher QoS (like the transport layer demultiplexing), can be separated from services with lower QoS requirements.

3.5 Functional View

3.5.1 Analyzing the Components and the Requirements for Performance and QoS

The individual STB software components with differing performance and quality of service requirements can be broken down into the following categories:

- Throughput-oriented compute tasks
- Low-latency real-time events (in the microsecond range)

- Real-time events in the millisecond range
- Soft real-time events
- Background tasks

Software partitioning on the system is normally done based on these categories, the functional components, and the interaction between these components.

The different functional components include:

- **Hardware video decoder control:** This occurs mostly on a field-by-field basis. It must react within tens of milliseconds or less to keep the video decode active in real-time. The CPU controlled video processing can also be done on a finer granularity, for example, using multi-threading. This can allow more flexibility for error-recovery as well as support the processing of non-standard bit streams.
- **Hardware audio decoder control:** It is a requirement that the audio stream run continuously, but due to the lower data density of audio stream, compared to video, the control activity is less demanding then for the video control process.
- **Software audio decoder:** The audio decoder, if implemented on a MIPS core either dedicated for audio or shared with other functionality, is a throughput-oriented compute task with specific performance requirements. The performance needs are very dependent on the implemented algorithm.
- **Transport demux:** The data stream coming from the front-end gets parsed and the individual audio, video, and information streams get stripped out and routed to the specific accelerators or memory. This is done primarily in hardware. Electronic Program Guide (EPG) information is usually interpreted in software. In this task, the MIPS32 bit extract instructions can be used to efficiently parse the data stream. For more information on these bit extract instructions see the specifications for the MIPS32®, MIPS64®, and the MIPS® DSP ASE architectures.
- **Hardware descrambling control:** The descrambling method could be Digital Encryption Standard (DES), Digital Video Broadcast (DVB) standard, or the National Renewable Security System (NRSS).
- **MHP/OCAP:** The new Multimedia Home Platform (MHP) and similar standards like OCAP are open middleware system standards. The execution environment is based on the use of a Java virtual machine and generic APIs running on the system CPU that can control the overall system.

4 Software Architecture of a Set Top Box

MHP and OCAP define an extensive application execution environment for digital interactive TV, independent of the underlying, vendor-specific, hardware and software. This execution environment is based on the use of a Java virtual machine and the definition of generic functions that provide access to the interactive digital TV terminal's typical resources and facilities. The interoperable MHP applications are run on top of these APIs. MHP and OCAP consist of a mix of soft real-time events, background tasks, and events in the millisecond range. One of the operating systems for set top box products supporting the MIPS32® 34K core’s multi-threading architecture is SMTC Linux. SMTC stands for Symmetric Multiprocessing on TCs and understands the concept of lightweight TCs.

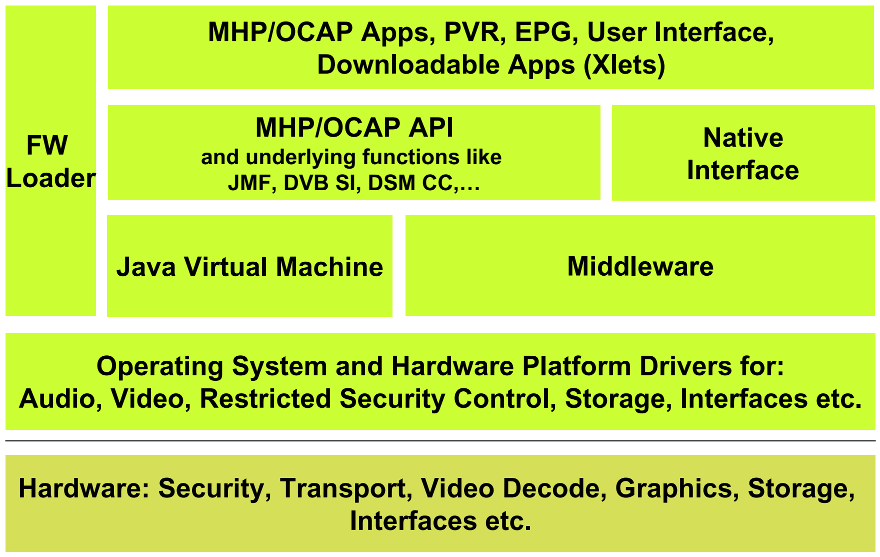
FW Loader
MHP/OCAP Apps, PVR, EPG, User Interface, Downloadable Apps (Xlets)
MHP/OCAP API
and underlying functions like JMF, DVB SI, DSM CC,...
Native Interface
Java Virtual Machine
Middleware
Operating System and Hardware Platform Drivers for: Audio, Video, Restricted Security Control, Storage, Interfaces etc.
Hardware: Security, Transport, Video Decode, Graphics, Storage, Interfaces etc.

Figure 4 Set Top Box Software Architecture

A typical MHP terminal software architecture is shown in Figure 4. The middleware, or (multi)media framework, is one of the most complex elements. Although there is no specific standard for this, the Khronos Group has published a generic API which captures the design philosophy found across most middleware implementations. Specifically, in [4] it is mentioned that: *The resulting design uses highly asynchronous communications, which allows processing to take place in another thread, on multiple processing elements, or on specialized hardware.*

Asynchronous communication is used not only by the middleware, but also in the MHP layer provided by the hardware/drivers vendor. At the OS level, asynchronous communication translates to running multiple execution threads, managing buffers, and handling thread and buffer-related events and interrupts.

The important point here is that media stack architectures such as MHP are inherently multi-threaded. Each layer has a few execution threads passing buffers around, responding to hundreds of interrupts per second, with sufficient cycles leftover for downloadable applications.

From the software architect's perspective, what would be the benefits of running a media stack on a MIPS 34K core with SMTC Linux? The next few sections describe the main benefits and show how these benefits impact the whole product life cycle.

4.1 Design and Implementation

By definition, each layer is designed, implemented, and tested as a component by different vendors in different environments. Particular vendors assume a certain level of system performance and for good reason. Once all pieces of the puzzle are in place, the reality may be that some software layer in some hard-to-reproduce context doesn't get the expected performance. This sort of typical scaling problem can be prevented well in advance by the system software architect.

The key observation here is that VPEs and TCs scale much better than software threads running on a single-threaded processor. A hardware context switch is more deterministic and is orders of magnitude faster than a software context switch. The beauty of multithreading is in its scalability: each thread runs as though the whole CPU is dedicated to itself.

Partitioning the system load properly on VPEs and TCs brings this ideal closer to life. Obviously, the other benefit here is in the low overhead and scalability of the thread management and synchronization primitives (mutexes, condition variables, and semaphores) due to the hardware based implementation. Determinism and speed in thread context switches and synchronization help the vendors to shorten the development time and deliver better-tested components.

4.2 End User Experience

Once the basic MHP terminal is up and running, the next challenge is to accommodate downloadable MHP applications. Since the terminal is better balanced by design using the hardware multi-threaded MIPS32 34K core, there is more CPU performance available for the applications and the overall user experience improves. Most likely none of these applications will get dedicated hardware TCs, but the MHP Implementation component could run in its own VPE and as such be a lot more accommodating.

In Figure 4, the Firmware (FW) Loader is the essential element for field upgradeability. Often this involves enabling even more functionality. Internet access with bidirectional references between broadcast video and web content, smart-card access for e-commerce, or caching broadcast applications for faster start-up are just few examples. More functionality invariably translates in more threads and thus requires even better scalability.

4.3 Platform Life Span

New features are added constantly to any multimedia platform and MHP is no exception. One document [5] calls specifically for better scalability in MHP 1.1.2 in the context of multiple video decoders and multiple tuners. From the Middleware Implementation's perspective, multiple video streams processing requires very little change; just instantiate few more video control objects/threads.

The hardware platform has to be able to scale properly to this extra load and the hardware multi-threading has the potential to achieve that. Consider features like simultaneous display of I-frames and scaled video or mosaic style EPGs with audio from other services. Then add supporting access to files in pluggable file systems (memory cards, USB-2 storage class devices) and multi-threading pressure becomes even greater.

The challenge for the software architect is to leverage the existing infrastructure while adding new components. A hardware multi-threaded platform is by far the best choice to meet not only the present scalability requirements but also the one coming with the new standards.

5 Conclusion

The MIPS32 34K processor core, with its implementation of the MIPS MT architecture, provides performance and QoS benefits to set top box designs. This document has shown several approaches to leverage this technology. Multi-threading helps the SoC and software architects to optimize the system effectively, reducing unused performance overhead. Multi-threading also improves the price/performance point of an end product and shortens the development cycle for the SoC and system designers.

6 References

[1] The MHP Knowledge Project, “The MHP Guide“, Institut für Rundfunktechnik GmbH, Mar. 2006.

[2] “Increasing Application Throughput on the MIPS32® 34K™ Core Family with Multithreading”, MD00535, MIPS Technologies, Oct. 2006.

[3] “MIPS® MT Principles of Operation”, MD00452, MIPS Technologies, Sept. 2005.

[4] The Khronos Group Inc., “OpenMAX™ Integration Layer Application Programming Interface Specification Version 1.0”, http://www.khronos.org/openmax

[5] Jon Piesing, “Introduction to MHP 1.1.2”, http://www.mhp.org