

Blending Asymmetric and Symmetric Multiprocessing with a Single OS on ARM11 MPCore

Author:

Masaki Gondo, Director of Engineering, eSOL Co., Ltd.

Synopsis:

ARM11 MPCore is an excellent multicore processor, appropriate for both SMP and AMP. Many embedded system designs already use the AMP model, either on chip or off chips, so unforeseen design challenges are limited for software developers using AMP model with MPCore. On the other hand, the increasingly demanding applications require more and more CPU power, and the utilization of SMP is soon to become a necessity.

However, SMP brings new design issues along with its advantages. The first half of this article is dedicated to discussing the issues and pros and cons of AMP/SMP, including throughput, concurrency, real-time determinism, reuse of existing software, programming model, and debug/analysis. The later half introduces eSOL's eT-Kernel Multi-Core Edition RTOS and eBinder tools, and how the issues can be put under control with its unique blending technology.

My brother happens to run a small coffee roaster/wholesale company. So he knows coffee. One thing I learned from him about coffee is the difference between “stand alone” and “blends.” Jamaican Blue Mountain coffee is said to be the most precious of “stand-alone” coffee beans, owing to its balanced, rich, delicate, and mild taste. However, my brother, who is also a coffee appraiser, recommends blends. He says blends combine different stand-alone coffee beans to create the optimal balance, like the one Blue Mountain has. This occurs through the art of mixing beans with different characteristics. Beans used for blending are not as costly as the stand-alone ones, resulting in a well balanced coffee, coupled with a more appealing price tag.

The ARM11 MPCore is a multi-core processor, which can host up to four ARM11 cores. Thanks to its innovating cache coherency technology and advanced MMU, it is well suited for SMP and AMP. Both AMP and SMP have their own characteristics, hence different advantages and disadvantages. Now, let's say you are considering using a multi-core processor in your next designs: which software model should you employ – AMP or SMP? This article addresses this by first identifying the points to consider and then introducing a unique OS technology that can offer the “best flavor and taste”.

Multi-processing Software Model Decision Points

The software in modern complex embedded systems, including consumer electronics and auto-infotainment devices, mobile phones, medical equipment, etc., performs a wide variety of processing. Some require

the highest throughput the system can crank out, some are very real-time (deterministic), and some are soft-real-time. They may require several device drivers with a lot of I/O, and many of them are concurrently executed. This is a result of ever increasing demands for more and advanced features. The increased demand poses several challenges. This includes high reuse of existing software, including ones from previous systems and COTS software. With even stronger pressure for time-to-market, more new software engineers need to be employed from the non-real-time, non-concurrent PC programming domain. Finally, debugging and analyzing these complex software systems can be a nightmare without the proper tools. These are enough challenges to consider – and now comes multicore design, soon to be the only way to continue this never-ending evolution.

Common software models for multicore processor are SMP and AMP. Symmetric (the S in SMP) typically means the software is not concerned about if the processor is multicore or not, because they are symmetric and the OS takes care of the multiple cores and provides abstraction. Therefore, the SMP model is realized with a single OS image managing all cores. On the other hand, with the Asymmetric variant the software is aware of what core it is to be run on. Essentially, with AMP you must assign some specific system functionalities to each core. Hence, separate OS images exist for each core, and the OS may not necessarily be the same kind of OS. The software needs to employ some inter-core communication libraries, and use that to synchronize or interwork with the software on other cores.



Each model, AMP and SMP, has its advantages and disadvantages. It is very important to understand these when deciding on which software model to employ when using a multicore processor like MPCore. The following are points to consider:

- Throughput
- Concurrency
- Realtime determinism
- Reuse of existing software
- Programming model
- Debug/Analysis

Let us look at them one by one.

Throughput

Theoretically, the SMP scheduling provides the best average performance over multiple cores. The OS schedules the task/thread to a free core, never leaving a core to idle unless there are no ready tasks to execute. On the other hand, with AMP, the OS performs the regular scheduling of single processor. The OS does not know whether or not other cores are free or not, nor can it schedule tasks on different cores. So if it happens that for one core, multiple tasks are ready and waiting to be executed, while on another core there is no ready task to run, the core is idling.

Concurrency

With AMP, a single processor is used by the OS with tasks running on top. Therefore, even though the OS provides the concept of concurrent execution with allowing multiple tasks/threads, they never run in parallel in reality because there is only one processor the OS can schedule the tasks on.

With SMP, on the other hand, now the OS finally has multiple cores to run multiple tasks on in parallel. A task in OS terms is an independent unit of execution, so the tasks that had to wait before with a single core processor now work beautifully in parallel – if the tasks are correctly using OS synchronization/exclusion primitives for inter-task processing. If the computation performed by each task is completely independent, meaning they do not communicate with each other, do not share any global variables, etc., then this works just fine under SMP scheduling, because they do not require any synchronization/exclusion in the first place. However once they do such, the problem may surface, depending on how this is implemented.

One typical way is to use interrupt-locking at the CPU level. This will not work, since it

assumes if interrupts are locked the task gets the exclusive control because another task is never scheduled without a scheduling trigger such as a timer interrupt or some other peripheral interrupt handler. This interrupt-locking method is no longer appropriate, because there are multiple cores to run multiple tasks at the same time – locking interrupt does not stop tasks on the other cores. It also has to be noted the CPU level interrupt locking is much more costly with multiple cores, as it can delay the invocation of some other tasks waiting for the interrupt trigger – because with SMP there are more than one core to run ready tasks – there may be a free core waiting to run a ready task – something that was not available with a single processor. The necessary model for interrupt handler and driver task exclusion is to use interrupt locking at the level of the interrupt controller, while for inter-task synchronization/exclusion use primitives such as the semaphore provided by the OS. This is also the basic principle for single processor based system; however the above coarse CPU level interrupt locking is frequently abused.

There is another, bigger problem other than interrupt locking; the dependency on priority-based scheduling for inter-task synchronization. With a single processor, a task with higher priority is always executed first. The SMP scheduler can also honor priority in that it schedules tasks based on the priority – however, with multiple cores to schedule, tasks with different priorities may be executed. This breaks the implicit assumption that no tasks run in parallel in reality thus the higher priority task will always run before lower priority tasks. Although the OS offers the abstraction of the CPU providing multiple execution contexts over a single processor, the implicit application dependency to the OS implementation now becomes a problem. It is possible to implement the OS so that it runs tasks with only the same priority, but then it can greatly reduce the throughput advantage of the SMP scheduling mentioned above as it can put cores idle too often. More difficult is predicting when these tasks with different priorities become ready, making it difficult to estimate the performance penalty and realtime-determinism, discussed next.

Real-time Determinism

‘Real-time’ means the ability to guarantee that particular processing finishes before a

defined deadline. The variance in the level of ‘guarantee’ is normally expressed as ‘hard,’ and ‘soft’ real-time. The guarantee level is also expressed as determinism, or predictability, as they are the same in this context. A hard real-time system is often found in mechanical control applications. For example, Anti-lock Breaking System (ABS) is a hard real-time system. You hit the break, and it has to be in effect in a predefined time, or you are in big trouble. The processing has to be 100% deterministic. One example of soft real-time system can be your PC mice. It has to react to, or follow your hand movement at least around tens of milliseconds. However, your life will not be lost if once in a while it takes 200 mill-seconds. So there is a sort of deadline, but not as hard as the ABS, though the quality of mice is probably better without such unusual response. As you can imagine, many systems have soft real-time features.

Being real-time is all about knowing when some particular processing finishes. With AMP, which essentially is just like a single processor system, the same old technique and know-how used with single processor design can be applied. At least it does not make it anymore difficult than it is now. On the other hand, SMP poses a potential problem. SMP scheduling can be said to be ‘best effort’ scheduling, providing the best average throughput multiple cores can give, without software designer to worry about how many cores are there. However, it is very difficult to determine ‘how long’ a particular task will take to finish some processing. For example, let us take a dual core processor that uses SMP. The system has three tasks. When only one task becomes ready, it can use one of the two cores and run. Comparing this to a single processor case, there is no difference in performance. Next, two tasks become ready to run. With SMP, the second task can be scheduled to the other free core, resulting in two tasks running at the same time. Comparing this to the single processor, this gives twice the throughput. Now, if the third task becomes ready, it has to wait for either one of the cores become free. The task scheduling order is also governed by when the trigger (basically the interrupt) happens, task priority with possible priority inversion and exclusion/synchronization, and the subsequent chain of task processing. The interrupt timing is hard to determine except for cyclic interrupts. A real-time task can only be scheduled in a



wide timing margin in this example. With two cores, as opposed to a single core, at least twice as many timing paths can affect the determinism of your processing.

This determinism issue can become an immediate problem when re-using legacy applications, including combining two program sets previously run on two separate processors in AMP model into one SMP model using multiple cores.

Reuse of Existing Software

The re-use of existing/legacy software is a very critical issue for keeping software development cost down. This includes not only the shared development with previous device models, but also COTS software and engineering resources.

So far, the software re-use with AMP is not much of a problem because essentially it is the same old single processor model. On the other hand, there are many issues to worry about with SMP despite the attractive (or demanding) throughput benefit – the difficulty of programming concurrency, the added complexity of real-time determinism, and the re-use of precious existing software assets.

Programming Model

Inherently these systems require multi-programming/concurrent programming, aided by an RTOS – however correct multi-programming is no trivial task, and as discussed above in ‘Concurrency,’ it is often not truly exercised. For AMP the environment does not really change, but for SMP correctness is vital. The required model is not new, but requires time to actually learn and implement the real parallel execution of programs. A different way of looking at this is that now the developer can use the multicore platform as a verification tool for multiprogramming. Something that was not cleanly multi-programmed could run just fine with a single processor; now it no longer runs. However, how can you find out what is wrong when multi-programming is incorrect. This raises the next topic of debug and analysis.

Debug/Analysis

For AMP, there are many existing solutions and most tools already support parallel debugging/analysis of AMP type multi-processor configuration. Since we are talking about systems that use a multi-threading real-time OS, tools that are OS aware in terms of both debugging and analysis

should assist developers to perform correct multi-programming for the particular OS. For SMP model, as we have discussed so far, the importance of correct multi-programming is paramount compared to AMP. Also, debugging and analyzing SMP software require OS specific support appropriate for a particular OS SMP implementation. It is necessary to know how the OS scheduled tasks/threads over multiple cores, in what order they run, what OS synchronization/exclusion APIs are called against what, which interrupt occurred when, and so on. Such capability may not be as important when the system is not so concurrent nor interacting much. For example, for PC applications, you may run a virus checker and a word processor at the same time, or compile your target software. Each application is implemented as a separate process, applications are mostly not multi-threaded and they essentially do not interact. This is not really the case in the complex embedded systems providing more and more integrated features.

eT-Kernel Multi-Core Edition blended solution

eT-Kernel Multi-Core Edition, an eSOL implementation of an RTOS for symmetric multicore processors like ARM MPCore, is designed to assist developers with the issues presented above. It is an SMP type RTOS, meaning a single image/ instance of the OS governs all the cores and all the software. What is unique about eT-Kernel Multi-Core Edition is that it offers two scheduling modes, named SPM (Single Processor Mode) and TSM (True SMP Mode), which can be used in combination. For a given MPCore based processor running eT-Kernel Multi-Core Edition, there can be tasks that run in SPM and others run in TSM.

Single Processor Mode and True SMP Mode

The TSM (True SMP Mode) is essentially normal SMP scheduling, where eT-Kernel automatically schedules ready tasks over multiple cores. The developer tells the OS to run a task in TSM, and the OS decides which core to run

the program on and when. On the other hand with SPM, the developer tells the OS which core to run the task on. The task only runs on the specified single core like that of a single processor, and never run in the other cores. The scheduling mode and the core are specified upon task creation. Therefore, it is possible to run the same program in either SPM or TSM by changing a parameter when instantiating the task.

SPM and TSM, used in combination, can successfully manage the seemingly competing issues discussed already. Let us discuss them in following sections.

SPM for Reuse of Existing Software/Concurrency/Real-time Deterministic

With SPM, the tasks run only on the specified processor core. In fact, at the OS boot time, each core is configured to be either a SPM core or a TSM core. The SPM core only runs SPM tasks, while the TSM cores only runs TSM tasks (see Figure 1). This essentially means the task-set (collection of tasks/process) on a SPM core has the same environment as in a single processor system – no two tasks runs in parallel in reality. Interrupts handled by the task-set can be sourced only to the specific core, also creating the same timing/concurrency conditions present in a single processor system. Naturally, it offers very practical re-use of existing single processor based software. This greatly eases the issue of software reuse and concurrency. Real-time

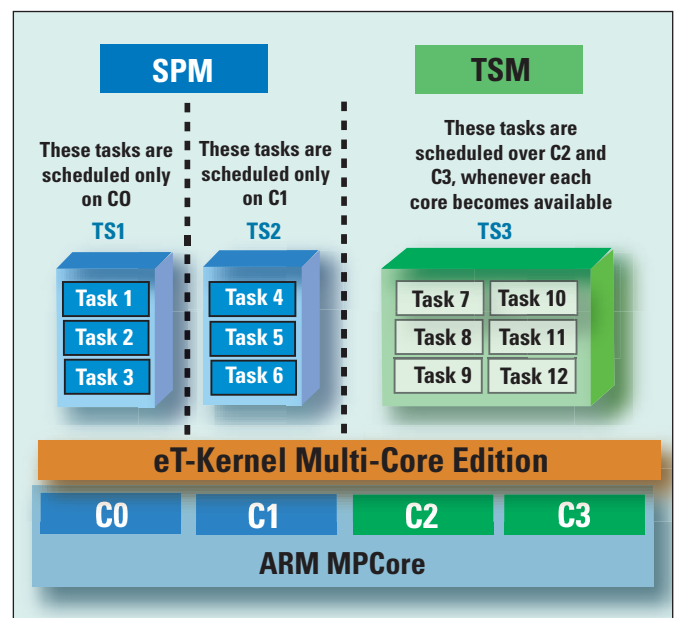


Figure 1

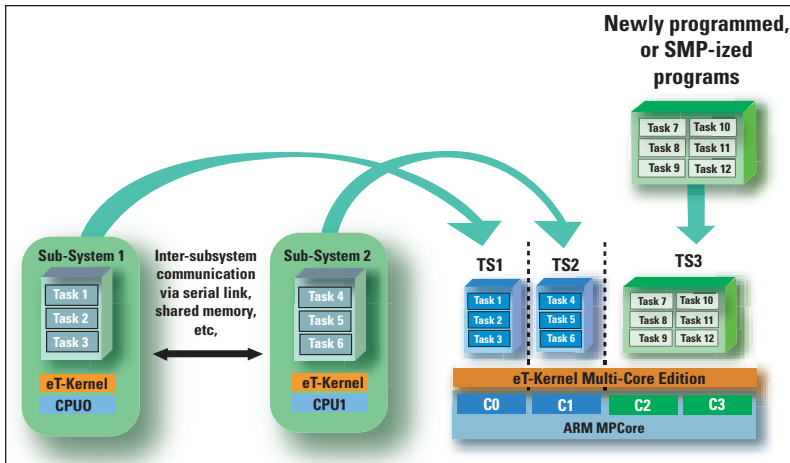


Figure 2

deterministicity is also improved back to the level of the single processor, due to the nature of SPM.

There is indeed added benefit that was not available with actually having only single processor core. Since eT-Kernel Multi-Core Edition is a SMP type OS, meaning all kernel objects (tasks, semaphores, etc) are managed by a single OS. Therefore, a task on one SPM core can easily use regular OS primitives to interact with another task in either yet another SPM core or the TSM cores. It is indeed now necessary for these tasks to be aware of concurrency, however this is only limited to the tasks that wish to communicate. One application of our RTOS in an existing system is shown in Figure 2. The previous AMP type system, consisting of two sub-systems each hosting a processor and OS, is communicating each other via some physical link (like PCI, etc.) wrapped by some device driver, further wrapped by some communication protocol layer. Now the device driver can be ported to a shared-memory based implementation, easily synchronizing through regular OS primitives like a message box, semaphore, etc. The implementation is straight forward as the two driver tasks each hosted on a different SPM core, run on the same OS.

TSM for Throughput

TSM is named True SMP Mode, to differentiate from SPM more clearly, since even with SPM, the OS itself is SMP type OS. After all, TSM is just a normal SMP scheduler, except that TSM tasks can talk to the SPM tasks without knowing they are run in SPM. In a traditional SMP OS, if a task wishes to interact with another task that runs on a single processor, it needed to talk

throughput nature of SMP scheduling is fully provided. It indeed needs new software, correctly multi-programmed. However, this can be introduced in phases, instead of the all or nothing choice of going all AMP or all SMP at once. Figure 3 shows the possible eT-Kernel Multi-Core Edition SPM/TSM core configuration. Please note even with all cores set to SPM, multi-programming can be verified if the tasks on different core interact with each other. The tasks can be easily transferred to TSM cores when the system has enough tasks that can utilize at least two TSM cores.

Process Scheduling Mode for Ease of Programming Model

As explained, SPM can also be used as a vehicle to attain and accumulate SMP (TSM) ready software, with control of how much software to shift or create for SMP. This helps developers to get used to the true multiprogramming model, while allowing introduction of a SMP ready multicore hardware design utilizing cores like MPCore before the software and the developers are ready.

In addition to this transition support, eT-Kernel Multi-Core Edition also offers process sub-task scheduling control. For many user mode processes, it is likely the software is to be

over some type of inter-processor/system communication protocol/library.

Back to the topic of throughput, since TSM tasks fully feature both the bright and the dark side of SMP, the high average computation

developed by ones with some good experience with desktop PC app development, but with much less multi-programming experience compared to native RTOS application developers. Processes can be assigned SPM or TSM just like the kernel tasks. In addition, when the subtasks (or threads) within a single process exist, the tasks can be scheduled on only one single core even though at the process scope each process scheduled in TSM is not necessary running on the same core all the time. It can be said that SPM is applied only for the set of tasks a single process holds. This creates, for a developer of process application, a virtual single processor environment with respect to concurrency issue, which is less demanding with regard to the correctness of multi-programming. At the same time, multiple processes are scheduled over multiple TSM cores automatically by the OS, still reaping the benefits of SMP scheduling at the allowed level.

Tool Support for Debug/Analysis

Finally but importantly, eT-Kernel Multi-Core Edition provides its own integrated development suite. eBinder is eSOL's original development tool suite for its RTOS, and eT-Kernel, including this Multi-Core Edition, is tightly integrated by the development suite. In the previous discussions on "Debug/Analysis", two points are made.

One is that the tool should support multi-programming to foster developers to practice correct multi-programming. The second is the tool should provide features to analyze software in relation to the RTOS specific SMP implementation model. The first is rather straight forward for eBinder to support, as the support for multi-program-

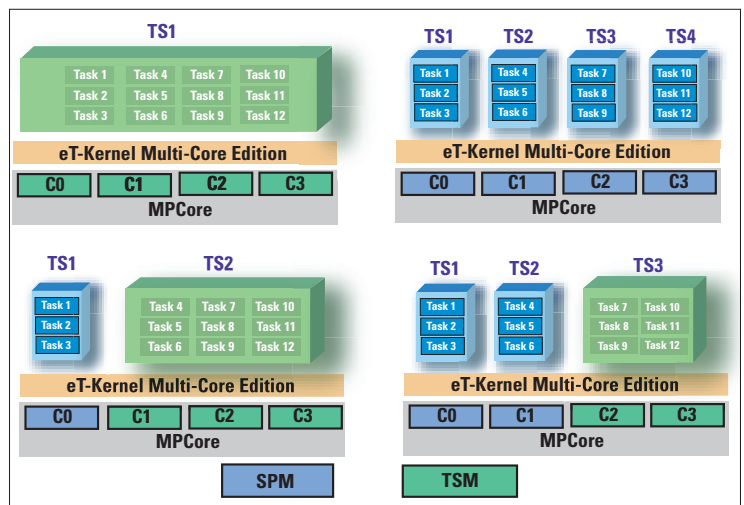


Figure 3



ming is one of the initial design goals of the tool when it first developed. There were not much multi-core processors back then, but as discussed already multi-programming is coupled with multi-tasking RTOS, not really with whether or not there are multiple cores to run SMP. I am not going to discuss in detail all the features of the tools, but one very basic and important feature is the Multi-context debugger, or task level debugger. When debugging with the Multi-context debugger, a developer first attaches to a task to be debugged, then halts or sets a breakpoint to debug. Alternatively create a task (or process) and debug from the beginning. The important part is that the multi-context debugger, when it is attached to a task context, only controls the one task, not others. Other tasks and interrupts keep running. This is true in reality anyway because a task can always (well almost) be preempted by an interrupt handler or another higher priority task invoked by the handler. This can happen unless all the interrupts are locked when the task runs, which is usually not the way to go as it can cause unwanted interrupt response latency, in addition to the already described problems with SMP. The same situation happens when a developer sets breakpoints and steps through his task with the task level debugger. This task level debugger works just the same way with the eT-Kernel Multi-Core Edition as well. The principle model of debugging one task at a time does not change when running on a single processor or multicore processor in SPM or TSM. If the core that the task gets scheduled to changes with TSM, the debugger automatically attaches to the task, so it does not matter to the developer – it is abstracted by the OS, and the tool is integrated with the OS. By the way, the debugger is called ‘Multi-context,’ because up to eight tasks can be attached in parallel and debugged at the same time for inter-task processing debug. Another special context exists called the ‘System context’ to which various JTAG ICE emulators can operate just like a JTAG ICE debugger. For some system software debugging, the capability to stop everything including the OS helps, and also for hardware bring-ups. The System context can be used for such. However, the task level debugger will have far more importance with multicore design for better multi-programming.

The second necessary feature is the SMP analysis tool. eBinder includes a tool called EvenTrek, which uses RTOS instrumenta-

tion (and also middleware or user instrumentation in addition), to trace entry and exit events for exceptions, interrupts, task dispatches, RTOS API calls, including the parameters passed and return values, with high resolution timestamps. The tool shows the instrumented data and display in graphical presentation for a system level view of all the task/process activities over time. This tool is quite useful with a regular single processor system, but with multicore based system, it is almost a must-have. With eT-Kernel Multi-Core Edition, the EvenTrek supports all the scheduling modes, and displays all the activities on all cores, mapped on the same timeframe. Therefore, for four cores, the display will look like Figure 4. One may be surprised by the complex look, but this is what is happening in an AMP-based system with four single core processors. The difference is that you usually do not have this level of inter-related, inter-mapped visibility of all the activities on all the processors in one view. Indeed, with multicore, to ensure your design is maximizing the powers of multicore, you do need to look at this and see if everything is working as designed, with regard to the processor utilization of each task. For SPM cores, for most of the time, it is not necessary to view multiple cores at the same time because, in principle, SPM tasks mostly work with tasks on the same core. EvenTrek has a filtering capability to selectively display only the activity of selected core(s), as in Figure 5. This is the same picture a developer sees with a single processor. eBinder also has another sampling based profiler called RealtimeProfiler, which show how much time each function in each context is consuming.



Figure 4

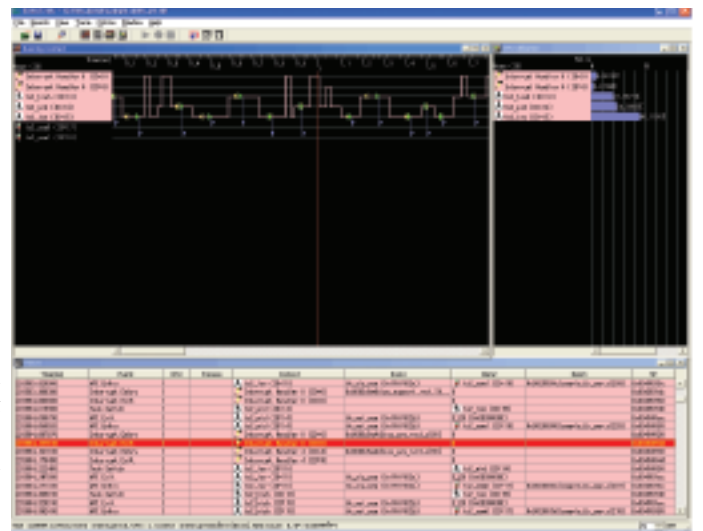


Figure 5

Looking Ahead

SMP is not something to be afraid of

As you have seen, SMP is not something to be afraid of, provided with the right knowledge and equipped with the right tools.

Blends of blends

Single MPCore processor and another heterogeneous processor

Multiple MPCore processors

Various Blends – multicore processors with separate single core processors, n by n ,

The blends have been the way for embedded systems for years, and it will continue to be so. It is not easy, but it will not make it any more difficult, either. With the right operating system and tools, you will manage to realize good blends.