Evaluating Schedulers for Multimedia Processing on Buffer-Constrained SoC Platforms

Alexander Maxiaguine
Swiss Federal Institute of Technology, Zurich

Simon Künzli and Lothar Thiele
Swiss Federal Institute of Technology, Zurich

Samarjit Chakraborty
National University of Singapore

Editors' note:
Scheduling on-chip resources using analytical techniques is becoming increasingly important in multimedia processing. This article presents an analytical framework for designing and evaluating schedulers for SoC multimedia platforms. The modeling technique subsumes standard event models used in real-time scheduling and accurately captures the variability in task execution requirements.

—Radu Marculescu, Carnegie Mellon University; and Petru Eles, Linköping University

The demand for configurable SoC platforms that support multimedia applications is continually increasing. Current practice in scheduling on-chip resources on such platforms relies on techniques such as stalling a processor when a buffer overflows and dynamically monitoring and adjusting service shares allocated to different multimedia streams on the processor. Furthermore, system designers often determine a scheduler's design parameters using purely simulation-based techniques. Such techniques can lead to complicated scheduling algorithms, and are typically ad hoc without any worst-case performance guarantees.

Our analytical framework lets system designers systematically design and evaluate schedulers for SoC multimedia processing platforms, thus addressing these problems. Our framework is based on the theory of Network Calculus, which was originally developed—and is still largely used—for analyzing communication networks.[1] (Elsewhere, we present extensions of this theory to the domain of system-level design of real-time embedded systems, as well as the background for our framework.[2,3])

Our framework differs in several ways from frameworks based on traditional real-time scheduling theory. The latter involve standard event models (periodic, sporadic, and so on), whereas we based our framework on a generalized event model that subsumes such standard models at the cost of more-involved analysis techniques.[2] Furthermore, traditional schedulability analysis relies on an explicit specification of deadlines and isn't useful for naturally modeling multimedia streams in which deadlines are implicitly specified in the form of buffer and delay constraints. Finally, our framework can accurately model variable execution requirements of tasks—an important characteristic for multimedia-stream-processing applications.

We illustrate our framework's utility through a detailed case study involving the design of a scheduler for a set-top box. Although this case study doesn't reveal our framework's full modeling capability, it facilitates a better understanding of the basic ideas behind our method. In a more complex setting, our framework can also model specific task properties, such as variable consumption and production rates of streams, and end-to-end processing delay requirements.

0740-7475/04/$20.00 © 2004 IEEE Copublished by the IEEE CS and the IEEE CASS

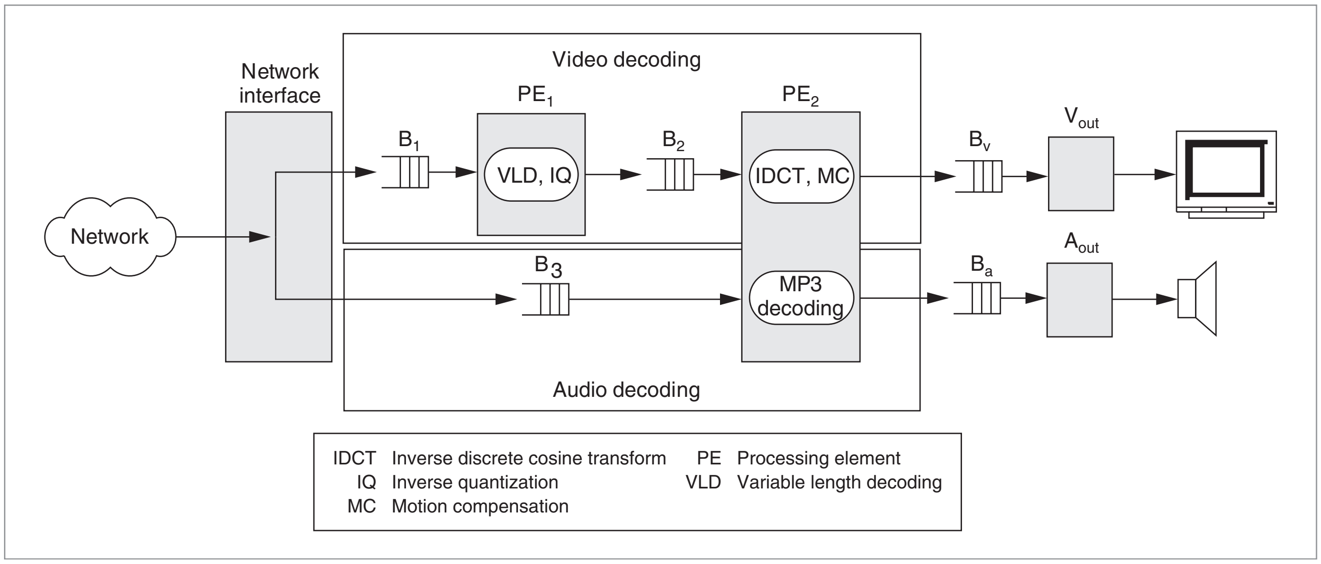

Figure 1. System model of a set-top box device processing audio and video streams.

SoC platform architectures

An example of a multimedia application is a digital set-top box that processes concurrent streams of audio and video data for broadband multimedia services. Often, it must also perform network packet processing for high-speed Internet access. The complexity of such applications, coupled with rapidly changing protocols and coding standards, high design costs, and time-to-market pressures, has led to the development of generic and scalable media processing platforms. These platforms can be tuned and deployed in a wide range of products. Examples of such platforms include OMAP from Texas Instruments, PrimeXsys from ARM, and Viper[4] and Eclipse[5] from Philips.

However, the main challenge in designing and deploying generic platforms is to satisfy the high computational demands and real-time constraints associated with processing multimedia streams. Although custom SoCs and ASICs can provide high performance, low power, and small size, it's far more difficult to achieve similar performance using generic platform architectures. The most promising way to overcome this drawback is to use intelligent platform management and scheduling techniques.[6,7]

A typical SoC platform for multimedia applications consists of a heterogeneous collection of fully programmable processing elements (PEs)—for example, MIPS Technologies, ARM, and TriMedia processors—and coarse-grained application-specific coprocessors optimized for specific tasks. The task structure representing the application to be executed on such a platform includes a set of concurrently executing tasks that exchange information solely through unidirectional data streams.[5,6]

Figure 1 shows a simple system-level model of a set-top box device implementing an audio-video decoder. Audio and video streams enter the device, which includes two PEs: PE_1 and PE_2. The MPEG-2 video decoder's task structure contains several tasks—some mapped onto PE_1, others onto PE_2. The MP3 audio decoder includes a single task, mapped onto PE_2. On-chip buffers B_1, B_2, and B_3 store the partially processed streams. Finally, PE_2 writes the decoded streams into play-out buffers B_v and B_a, which the video and audio output devices read.

Scheduling multimedia streams

Different streams entering a PE (for example, PE_2) could be associated with different input rates. The respective output devices could also consume the outgoing processed streams at different prespecified rates. Additionally, the processing requirements associated with the streams can vary widely. Therefore, to satisfy the real-time constraints imposed by such I/O rates and the fixed buffer sizes (such as B_v and B_a in Figure 1), it's necessary to suitably schedule the multiple streams entering a PE.

The main difficulty in devising such a scheduling strategy arises from the bursty nature of the streams entering a PE. On-chip traffic from processing multimedia tasks on multiprocessor SoCs tends to be highly complex and bursty for three main reasons.[8] First, the

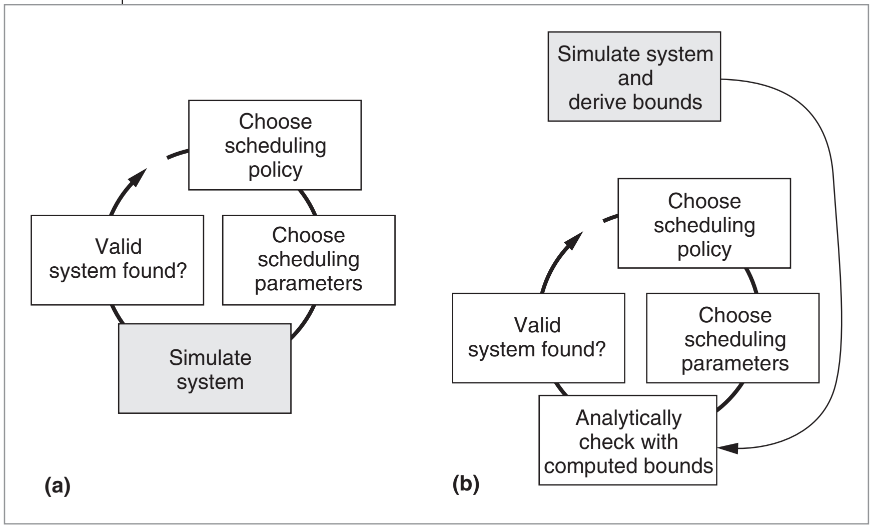
Choose scheduling policy
Valid system found?
Choose scheduling parameters
Simulate system
(a)
Simulate system and derive bounds
Choose scheduling policy
Valid system found?
Choose scheduling parameters
Analytically check with computed bounds
(b)

Figure 2. Evaluating multiple scheduling policies and their associated parameters: traditional design cycle, purely based on simulation (a); and our design cycle, which resorts to simulation only once and bases subsequent iterations on analytical methods (b).

execution time of many media processing tasks highly depends on the properties of the particular audio-video sample being processed. According to Rutten et al.,[5] the ratio of the worst-case and the average load on a PE due to such a task can easily be as high as a factor of 10. Second, the quantity of the I/O data consumed and produced by a task can also vary widely. An example of this is the variable-length-decoding task in Figure 1. Third, the input multimedia streams already tend to be highly bursty when they enter a processing device. For example, in Figure 1, the arrival pattern of the input streams entering the set-top box would depend on the network's congestion levels. In addition to these effects, the burstiness in the streams' arrival pattern could increase as they pass from one PE to the next, depending on these PEs' contention and scheduling policies.[9]

Buffer constraints

Any scheduling policy on a PE belonging to a setup similar to that in Figure 1 must typically satisfy the following constraints: None of the buffers should overflow, and the play-out buffers read by the real-time output devices should never underflow. The audio-video input devices read the play-out buffers at specified rates, depending on the required output quality. Therefore, the constraint on such a buffer's underflow is to ensure that this required output quality is guaranteed. In many cases, using blocking writes and reads to prevent buffer overflows or underflows isn't feasible. Efficiently implementing such blocking mechanisms requires either a multithreaded processor architecture or substantial runtime operating-system support for context switching.

The motivation for this buffer-centric design and evaluation of schedulers is that buffers are available only at a very high premium, because of their large on-chip area requirements.[8] Hence, they play a central role in the design of any scheduling or SoC platform management policy. Our analytical framework can help a system designer systematically design and evaluate schedulers for the different PEs of such a buffer-constrained SoC platform. The main novelty of this framework is that it provides a way to accurately characterize the variable execution times of multimedia processing tasks and the burstiness of on-chip traffic.

Multimedia scheduling

Designing schedulers for on-chip PEs involves significantly different constraints from those for scheduling and buffer management of multimedia applications in operating systems and communication networks.[10] In the latter domain, the scheduling overhead is often negligible in comparison to the execution times of the tasks. This allows for complicated, online scheduling algorithms. However, in our resource-constrained setup, implementing such algorithms might be infeasible: Often, on-chip PEs have only lightweight or even no operating system support. Furthermore, in the communication networks domain, buffer-use restrictions are not as acute, and it's possible to recover from data loss due to buffer overflows. However, such mechanisms are too complicated for an on-chip setup.

Our evaluation framework provides worst-case guarantees on any scheduler's properties. Most of the state of the art in this area rely on simulation-oriented techniques to evaluate any scheduling or platform management policy, and follow the design cycle in Figure 2a. Our technique follows the design cycle in Figure 2b; we resort to simulation only once, to derive certain system bounds. Subsequently, in our framework the evaluation of all scheduling policies and parameters (such as suitable weights for a weighted round-robin scheduler) relies solely on analytical methods. Hence, when the parameter space associated with designing a scheduler is relatively large, our framework is a few orders of magnitude faster than purely simulation-based approaches.

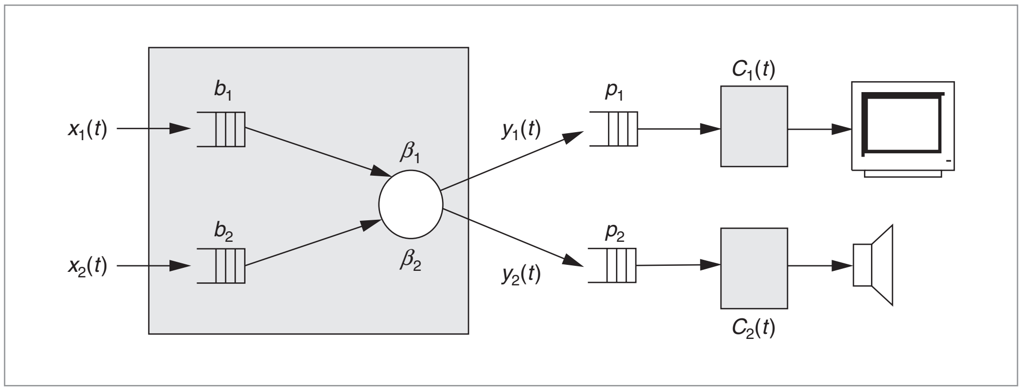

Figure 3. Abstract view of PE$_2$ for the system model of the set-top box shown in Figure 1.

Framework overview

For the sake of generality, we consider any multimedia stream to include a potentially infinite sequence of stream objects. A stream object could be a macroblock, a video frame, an audio sample, or a network packet, depending on the part of the architecture where the stream exists. For example, in Figure 1, stream objects are network packets when the relevant stream is that entering the network interface. In contrast, the stream objects are partially processed macroblocks when the relevant stream is that written into buffer B$_2$.

Figure 3 shows an abstract view of processing element PE$_2$ in the system model of the set-top box in Figure 1. Functions $x_1(t)$ and $x_2(t)$ specify the two streams entering this PE. These functions denote the total number of stream objects that arrive at the two internal buffers in Figure 3 over time interval $[0, t]$. Now assume stream i ($i = 1, 2$) receives service β_i from the PE. β_i is a tuple (β_i^l, β_i^u), where β_i^l and β_i^u are the lower and upper bounds on the service provided by the PE to stream i, such that within any time interval of length Δ, the PE can process at least $\beta_i^l(\Delta)$ and at most $\beta_i^u(\Delta)$ stream objects. Two factors thus determine β_i:

- the maximum and minimum execution times of the stream objects belonging to stream i, and
- the scheduling policy implemented on the PE to schedule the different streams and possibly other tasks processed on this PE.

In Figure 3, the two internal buffers with sizes b_1 and b_2 (the number of input stream objects these buffers can store) correspond to buffers B$_2$ and B$_3$ in Figure 1. The two play-out buffers of sizes p_1 and p_2 correspond to buffers B$_v$ and B$_a$. Function $y_i(t)$ specifies the processed output stream entering a play-out buffer. Like $x_i(t)$, this function denotes the number of stream objects exiting the PE over time interval $[0, t]$. The real-time output device associated with stream i consumes stream objects from the play-out buffer at a rate specified by function $C_i(t)$, which denotes the number of stream objects consumed during time interval $[0, t]$. It's important to note the distinction between functions x_i, y_i, C_i, and β_i. The first three functions denote cumulative values over $[0, t]$, whereas β_i takes *time interval length* as an input parameter. We will refer to $\beta_i(\Delta)$ as a service curve, with β_i^l being the lower curve and β_i^u being the upper one.[2]

Our evaluation framework can solve the following two problems. First, for the PE in Figure 3, given functions $x_i(t)$ and $C_i(t)$ for streams 1 and 2, and buffer sizes b_1, b_2, p_1, and p_2, the first problem is to compute functions β_1 and β_2 such that none of the four buffers overflow and the two play-out buffers never underflow. Second, once we've obtained β_1 and β_2, the second problem is as follows. Given any scheduler, how do we compute the service this scheduler offers to the two streams, in terms of its service curves, denoted β_1' and β_2'? If, for all $\Delta \geq 0$, $\beta_1^l(\Delta) \leq \beta_1^{l\prime}(\Delta)$ and $\beta_1^u(\Delta) \geq \beta_1^{u\prime}(\Delta)$, and similar inequalities hold for β_2 and β_2', then we can conclude that this given scheduler satisfies all buffer constraints and is thus a feasible scheduler for the PE. A designer can then further evaluate this scheduler by considering other factors, such as scheduling overhead and implementation complexities.

Computing bounds on the required service

We denote the min-plus convolution of any two functions, f and g, as $(f \otimes g)(t) = \inf_{s:0 \leq s \leq t}\{f(t-s)+g(s)\}$. We denote the min-plus deconvolution of f and g as $(f \oslash g)(t) = \sup_{u:u \geq 0}\{f(t+u) - g(u)\}$. We use $f \wedge g$ to denote the infimum of f and g (or the minimum, if it exists) and $f \vee g$ to denote the supremum of f and g (or the maximum, if it exists).

Now, consider the two streams $i = 1, 2$ in Figure 3. For simplification, we drop identifier i. We express the play-out buffer underflow constraint for the stream as $y(t) \geq C(t), \forall t \geq 0$. Similarly, we express the constraint on the play-out buffer overflow as $y(t) \leq C(t) + p, \forall t \geq 0$. Finally, we express the constraint on the overflow of the internal buffer associated with the stream as $y(t) \geq x(t) - b, \forall t \geq 0$. Combining the first and third constraints, we obtain constraint $y(t) \geq C(t) \vee [x(t) - b], \forall t \geq 0$. If lower service curve β^l represents the minimum service that the PE guarantees to the stream, then $y(t) \geq (\beta^l \otimes x)(t)$.[1] Hence, the minimum value of $y(t)$ at any time t is $(x \otimes \beta^l)(t)$. Then, substituting for $y(t)$ in the constraint just given, we obtain $(x \otimes \beta^l)(t) \geq C(t) \vee [x(t) - b], \forall t \geq 0$. Using techniques described by Le Boudec and Thiran,[1] we can further reformulate this inequality as

$$\beta^l(t) \geq \{C(t) \vee [x(t) - b]\} \oslash x(t), \forall t \geq 0 \qquad (1)$$

If upper service curve β^u represents the maximum service that the stream can receive from the PE, then $y(t) \leq (\beta^u \otimes x)(t)$ holds. Therefore, using $(\beta^u \otimes x)(t)$ as the maximum value of $y(t)$, we can reformulate the constraint on the play-out buffer overflow as $(\beta^u \otimes x)(t) \leq C(t) + p, \forall t \geq 0$, or equivalently,

$$\beta^u(t) < [C(t) + p] \oslash x(t), \forall t \geq 0 \qquad (2)$$

Inequalities 1 and 2 (bounding β^l and β^u) give lower and upper bounds on the service that any feasible scheduler implemented on the PE must provide to the stream.

Characterizing scheduling disciplines

Now we come to the second problem. Given a scheduler to be implemented on a PE, does the resulting service offered to each multimedia stream processed on the PE match the service that the stream requires? The service required by a stream is what we computed as our first problem. However, we computed this requirement in terms of the number of stream objects to be processed within any given time interval. The service that a scheduler provides to a stream, on the other hand, is naturally expressed in terms of the number of processor cycles. Hence, we need a way to express this service in terms of the number of stream objects. Because of the variability in the execution requirements of the different stream objects belonging to a stream, this is difficult. We address this problem by characterizing the variability using the concept of workload curves,[3] which is very similar to the concept of service curves introduced earlier.

Workload characterization

We specify a workload curve, γ, associated with a stream by the tuple (γ^l, γ^u), such that $\gamma^l(k)$ denotes the minimum number of consecutive stream objects from this stream guaranteed to be completely processed using k processor cycles, and $\gamma^u(k)$ is the maximum number of consecutive stream objects that can be completely processed using k processor cycles. Let service curve $\sigma = (\sigma^l, \sigma^u)$, given in terms of the number of processor cycles, specify the service that a scheduler offers to a particular stream. Then, $\sigma^l(\Delta)$ and $\sigma^u(\Delta)$ denote the minimum and maximum number of processor cycles available to the stream within any time interval of length Δ. Now let $\beta^l(\Delta)$ and $\beta^u(\Delta)$ be the service curves the stream requires to meet the buffer constraints, expressed in terms of the number of stream objects. Then, for the scheduler to be feasible,

$$\gamma^l(\sigma^l(\Delta)) \geq \beta^l(\Delta), \forall \Delta \geq 0$$
$$\gamma^u(\sigma^u(\Delta)) \leq \beta^u(\Delta), \forall \Delta \geq 0$$

These inequalities should hold for all the streams processed by the PE that contains the scheduler. Various methods can serve for obtaining a stream's workload curves; these include simulations and analytical modeling of the software processing the stream.

TDMA schedulers

We now characterize a time division, multiple access (TDMA) scheduler in terms of the service it provides to any particular stream when that scheduler is scheduling multiple streams. There are two main reasons for using TDMA as an example. First, it's simple enough to implement in a SoC setup, and it has low scheduling overhead, so it's widely used for scheduling on-chip PEs and communication resources. Second, TDMA is also relatively easy to characterize in terms of service curves; hence, it provides a simple illustration of the theory just presented. Of course, our evaluation framework is not restricted to analyzing only TDMA schedulers. System designers can use it to analyze any static or dynamic priority-scheduling algorithm, including preemptive and nonpreemptive versions. These include scheduling policies such as fixed-priority, weighted round-robin, and earliest-deadline first. In fact, our framework can evaluate any scheduling policy that is characterizable using service curves. Moreover, it can evaluate a platform in which different scheduling policies are used on the different PEs.[2]

Now, consider two streams scheduled on a PE by a TDMA scheduler with period P. The weights associated with streams 1 and 2 are w_1 and w_2, where $w_1 + w_2 \leq 1$. The scheduler divides time into periods of length P. Within any period, the scheduler allocates $w_1 \times P$ consecutive units of the PE's time to stream 1, and $w_2 \times P$ consecutive units to stream 2. If a stream doesn't have a sufficient number of stream objects to exhaust the processor share allocated to it, the unused processor cycles are wasted.

Assuming P is infinitesimally small, we can neglect the effects of a finite sampling of the processor cycles. Then, we can calculate the service offered to the two streams in terms of processor cycles as follows. If c is the number of processor cycles available from the PE per unit time (that is, c is the PE's clock rate), service curve $\sigma^l(\Delta) = \sigma^u(\Delta) = c\Delta$ constitutes the total service offered by the processor. The service curve for stream 1 is $\sigma_1^l(\Delta) = \sigma_1^u(\Delta) = w_1c\Delta$; the service curve for stream 2 is $\sigma_2^l(\Delta) = \sigma_2^u(\Delta) = w_2c\Delta$. Therefore, the lower and upper service curves for both streams coincide and are straight lines with slopes w_1c and w_2c, respectively.

When P has a finite value, the service curves take the form of a staircase function, and the lower and upper curves no longer coincide. Figure 4 gives an example of such a service curve. Here, P is set to a time interval over which the PE offers a total of 2.5×10^6 cycles. The TDMA scheduler schedules one video and one audio stream on the PE. The weights associated with the video and audio streams are 0.109 and 0.891. Figure 4 shows the lower and upper service curves offered to the video stream.

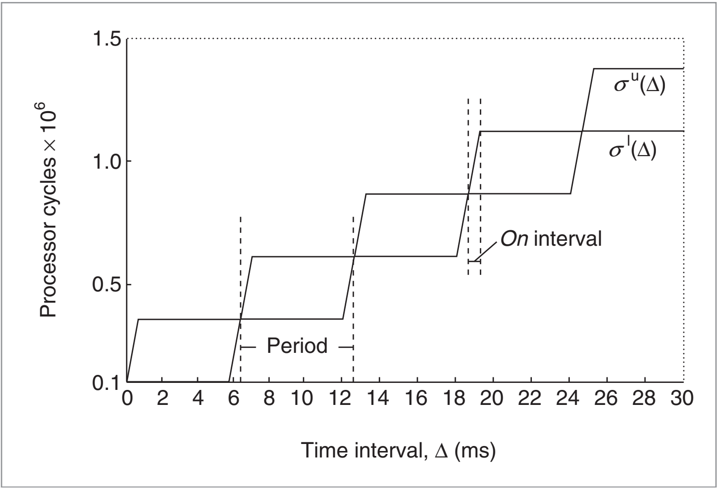

Figure 4. Example service curves. Here, σ^l and σ^u are the lower and the upper service curves corresponding to the service received by one of the two streams scheduled on a PE using a TDMA scheduler. The scheduler's period, *P*, is set to the equivalent of 2.5×10^6 processor cycles. The weight associated with the stream whose service curve is shown is 0.109, and the weight associated with the second stream is 0.891. The *on* interval indicates the time over which the PE processes the first stream within any period. The value of σ^l is initially 0, corresponding to the maximum time the processor is unavailable to the stream.

Set-top box application scenario

We used our framework to evaluate different schedulers on the PEs of the set-top box in Figure 1. Again, for simplicity, we restrict ourselves to TDMA-based schedulers. After the network interface processes the MPEG-2 video stream, the coded bitstream enters PE_1, which executes the variable-length decoding (VLD) and inverse quantization (IQ) tasks. PE_2 then processes the resulting partially decoded stream of macroblocks. PE_2 executes the inverse discrete cosine transform (IDCT) and motion compensation (MC) tasks for each macroblock and writes the resulting decoded macroblocks into video play-out buffer B_v, which video output device V_{out} periodically reads; V_{out} finalizes the processing and displays the video information.

The audio stream appears at the network interface's output as a sequence of frames. PE_2 performs MP3 decoding on each frame in the sequence and writes decoded audio samples into audio play-out buffer B_a, which audio output device A_{out} periodically reads one frame at a time; A_{out} performs final processing and play-out.

PE_2, therefore, executes tasks for both streams and represents a shared resource in the platform architecture. Identifying an appropriate scheduler for PE_2 is thus an issue that the system designer must address. To avoid degradation of the sound and picture quality, such a scheduler must ensure that no audio or video samples are lost due to an overflow of any of the buffers and that play-out buffers never underflow. Such a scheduler might be difficult to identify because of the high variability in the execution time of the different tasks running on PE_2 and the burstiness of the two streams that it processes.

Because PE_2 processes only two streams, any TDMA-based scheduler is completely specified by weights w_1 and w_2 and period P. Determining w_1 and w_2 is relatively straightforward. The long-term average rate at which PE_2 processes either of the two streams must exactly equal the corresponding output device's long-term average consumption rate for that stream. Either a buffer overflow or underflow is bound to occur at some point if these long-term rates do not match. Designers should therefore choose weights w_1 and w_2 to match these rates. However, there can be short-term mismatches in the processing and consumption rates because of the burstiness

Table 1. Buffer sizes for the platform architecture in Figure 1.

Buffer	Size
B_2	4,000 macroblocks
B_v	3,200 macroblocks
B_3	4 frames
B_a	4 frames

Table 2. Specification of the two multimedia streams processed by the platform architecture in Figure 1.

Multimedia stream	Parameter	Specification
MPEG-2 video*	Constant bit rate	8 Mbps
	Frame rate	25 fps
	Picture resolution	704 × 576 pixels
	Clip duration	15 s
MP3 audio	Constant bit rate	256 Kbps
	Sampling frequency	44.1 kHz
	Clip duration	15 s

* susi080.m2v, available at ftp.tek.com/tv/test/streams/Element/MPEG-Video/625/

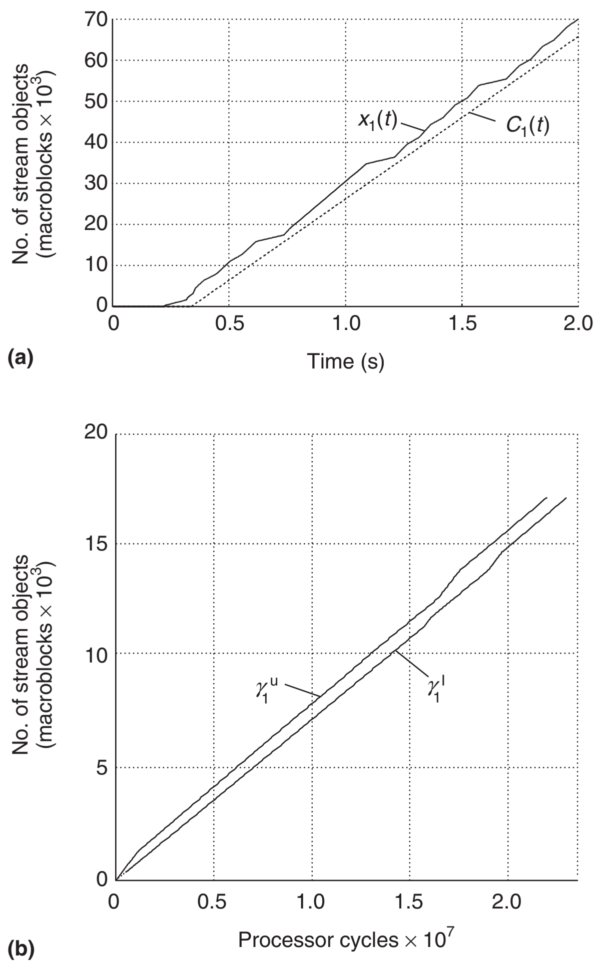

Figure 5. Specification of the video stream: functions $x_1(t)$ and $C_1(t)$ from Figure 3 (a); and workload curves (γ_1^l, γ_1^u), capturing the total execution requirements for tasks IDCT and MC running on PE_2 (b).

of the streams and the variability in their execution requirements from PE_2. The tolerable amount of mismatch depends on the sizes of the internal and play-out buffers associated with each stream, and period *P*.

Finding an appropriate value of *P* is not straightforward. There is generally a set of such values satisfying all buffer constraints. However, given any value of *P*, our framework can determine whether the resulting scheduler is feasible. After determining a set of feasible values of *P*, the system designer can use other evaluation criteria, such as incurred scheduling overhead or power consumption, to narrow that set (see Figure 2).

Arrival pattern and workload specifications of streams

The system configuration for the platform architecture in Figure 1 is as follows. Each PE is a reduced-instruction-set computing (RISC) core. PE_1 has application-specific extensions for MPEG-2 processing and runs at a clock rate of 200 MHz. PE_2 has application-specific extensions for video-processing functions and runs at a clock rate of 390 MHz. Table 1 gives the buffer sizes for this architecture. Table 2 gives the parameters related to the two streams given in Figure 1. The streams correspond to an MPEG-2 video and an MP3 audio clip. These represent typical clips that the set-top box in Figure 1 must process.

We obtained the $x_1(t)$ function in Figure 3 by measuring the execution times of the VLD and IQ tasks for each macroblock in the video sequence and by accounting for

- the constant arrival rate of the compressed bitstream at the input of PE_1 shown in Figure 1; and
- the number of bits per macroblock, which is variable because of the VLD task.

Figure 5a shows the resulting function, $x_1(t)$. Similarly, function $C_1(t)$, shown in Figure 5a, specifies the consumption of the video stream by V_{out}. The value of this function is 0 for the first 0.34 seconds, corresponding to an initial buffering delay. After this delay, the function increases with a constant slope, representing a periodic consumption pattern of 39,600 macroblocks per second. (One macroblock corresponds to a 16×16 pixel block in a frame; thus, one frame with resolution 704×576 pixels contains 1,584 macroblocks. Therefore, 25 frames per second result in 39,600 macroblocks per second.)

The workload curves shown in Figure 5b capture the total execution requirements of the IDCT and MC tasks running on PE_2. Rather than using a constant number, we use the workload curves to capture the variation in the total execution requirements of these two tasks. To obtain these curves, we first collected a trace of execution times for the pair of tasks. We then searched this trace using time windows of different lengths (from 0 to the length of the trace), and we identified the maximum and minimum execution requirements occurring within each time window.

Functions $x_1(t)$ and $C_1(t)$, combined with the workload curves, completely specify the video stream. We obtain the specification of the audio stream similarly. Once such a specification of the streams is available, designers can use our framework to evaluate any scheduler using the process in Figure 2 and without resorting to further simulations.

Evaluating TDMA schedulers with different periods

Given functions x_1, x_2, C_1, and C_2 for the representative video and audio clips, Figure 6 shows the required service curves, β_1 and β_2, for the video and audio streams. These service curves represent the number of stream objects that PE_2 must process within any time interval of a given length. The figure also shows the offered service curves $\gamma_1^l(\sigma_1^l)$ and $\gamma_1^u(\sigma_1^u)$, and $\gamma_2^l(\sigma_2^l)$ and $\gamma_2^u(\sigma_2^u)$. These service curves represent the number of stream objects that PE_2 will process using a TDMA scheduler with a period of 2.5×10^6 processor cycles. TDMA weights w_1 and w_2, associated with the two streams, are 0.109 and 0.891. (Figure 4 shows service curve σ_1 in terms of the number of processor cycles.)

To validate our framework, we evaluated several different TDMA-based schedulers, having different values of w_1, w_2, and P. Table 3 summarizes the results. For each scheduler configuration, the table shows

- whether our framework evaluated the scheduler as feasible or infeasible, and
- the corresponding simulation results that measure the maximum and minimum buffer fill levels (from which we can identify buffer overflows and underflows).

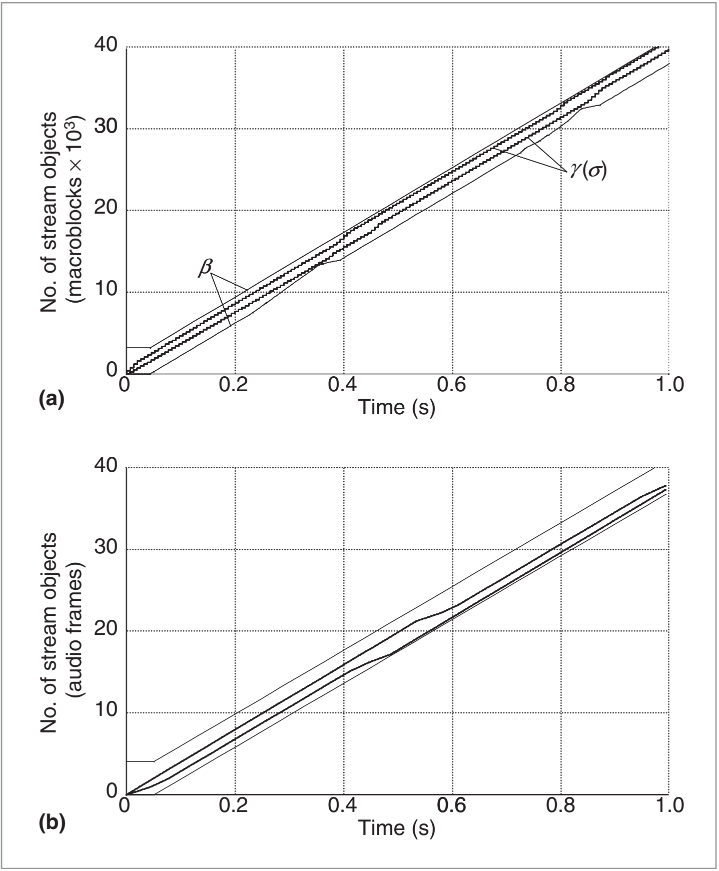

Figure 6. Service requirements specified by service curve β and the provided service specified by curve $\gamma(\sigma)$ for the video (a) and audio (b) streams. The upper and lower curves corresponding to $\gamma(\sigma)$ lie completely between the upper and lower service curves, β. This implies that the service provided to a stream matches its requirements; hence, the scheduler satisfies all the buffer constraints. For an infeasible scheduler, the resulting upper and lower curves of $\gamma(\sigma)$ would not completely lie between the upper and lower curves of β.

To obtain these simulation results, we used a transaction-level model of the architecture written in SystemC (http://www.systemc.org). The models of processors PE_1 and PE_2 are from a customized version of the SimpleScalar instruction set simulator (http://www.simplescalar.com); we used the simulator's sim-profile configuration. The PEs use the portable instruction set architecture (PISA) with application-specific extensions for MPEG-2 decoding and video processing. In the table, the buffer backlogs are in number of macroblocks for

Table 3. Results obtained with our framework compared to simulation results, for different configurations of a TDMA scheduler implemented on PE_2 of the architecture shown in Figure 1.

Scheduler parameters				Buffer backlog (no. of macroblocks, for video; no. of frames, for audio)			
Period (cycles)	Video weight	Audio weight	Schedulability test	B_2	B_v	B_3	B_a
1.0×10^6	0.109	0.891	Passed	3,610	2,437	2	2
	0.115	0.885	Failed	2,844	3,299*	2	2
	0.106	0.894	Failed	4,812*	1,979	2	3
2.5×10^6	0.109	0.891	Passed	3,736	2,559	2	2
	0.115	0.885	Failed	2,966	3,402*	2	2
	0.106	0.894	Failed	4,899*	2,110**	2	3
7.0×10^6	0.109	0.891	Failed	4,040*	2,540	2	2
	0.115	0.885	Failed	3,292	3,300*	2	2
	0.106	0.894	Failed	5,144*	2,023**	2	3

* Buffer overflow
** Buffer underflow

the video stream, and number of frames for the audio stream.

From Table 3, it's apparent that designing an appropriate scheduler can greatly influence a platform architecture's on-chip buffer requirements. If the design space is relatively large, especially for scheduling multiple PEs, resorting to purely simulation-based techniques is no longer feasible. Our framework can provide systematic guidance in such cases.

WE BELIEVE OUR FRAMEWORK can also have applications in problems other than conventional processor scheduling. One of the directions we are currently exploring is extending this framework to analyze processor voltage and frequency-scheduling algorithms for power-aware multimedia processing. ■

Acknowledgments

The Swiss Innovation Promotion Agency supported the work of Alexander Maxiaguine and Simon Künzli through projects KTI 5845.1 and KTI 5500.2. The work of Samarjit Chakraborty was partly supported by National University of Singapore grant R-252-000-190-112 through the "ASTRA: System-Level Design and Analysis of Architectures for Streaming Applications" project.

■ References

1. J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, LNCS 2050, Springer-Verlag, 2001.
2. S. Chakraborty, S. Künzli, and L. Thiele, "A General Framework for Analyzing System Properties in Platform-Based Embedded System Designs," *Proc. 6th Design, Automation and Test in Europe* (DATE 03), IEEE CS Press, 2003, pp. 190-195.
3. A. Maxiaguine, S. Künzli, and L. Thiele, "Workload Characterization Model for Tasks with Variable Execution Demand," *Proc. 7th Design, Automation and Test in Europe* (DATE 04), vol. 2, IEEE CS Press, 2004, pp. 1040-1045.
4. S. Dutta, R. Jensen, and A. Rieckmann, "Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems," *IEEE Design & Test*, vol. 18, no. 5, Sept.-Oct. 2001, pp. 21-31.
5. M.J. Rutten et al., "A Heterogeneous Multiprocessor Architecture for Flexible Media Processing," *IEEE Design & Test*, vol. 19, no. 4, July 2002, pp. 39-50.
6. M.J. Rutten, J.T.J. van Eijndhoven, and E.-J.D. Pol, "Robust Media Processing in a Flexible and Cost-Effective Network of Multi-tasking Coprocessors," *Proc. 14th Euromicro Conf. Real-Time Systems* (ECRTS 02), IEEE CS Press, 2002, pp. 223-230.
7. K. Sekar, K. Lahiri, and S. Dey, "Dynamic Platform Management for Configurable Platform-Based System-on-Chips," *Proc. Int'l Conf. Computer Aided Design* (ICCAD 03), ACM Press, 2003, pp. 641-648.
8. G. Varatkar and R. Marculescu, "On-Chip Traffic Modeling and Synthesis for MPEG-2 Video Applications," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 1, Jan. 2004, pp. 108-119.
9. K. Richter, M. Jersak, and R. Ernst, "A Formal Approach

to MpSoC Performance Verification," *Computer*, vol. 36, no. 4, Apr. 2003, pp. 60-67.

10. S.H. Lee et al., "Dynamic Buffer Allocation in Video-on-Demand Systems," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, 2001, pp. 343-354.

Alexander Maxiaguine is a PhD student at the Computer Engineering and Networks Laboratory of the Swiss Federal Institute of Technology, Zurich. His research interests include models and methods for system-level performance analysis and scheduling of embedded multiprocessor architectures, especially for real-time multimedia applications. Maxiaguine has an MS in electrical engineering from the Moscow Technical University of Communications and Informatics. He is a member of the IEEE and the ACM.

Samarjit Chakraborty is an assistant professor of computer science at the National University of Singapore. His research interests include system-level design and analysis of real-time and embedded systems, with a focus on architectures for streaming applications. Chakraborty has a BE in computer science from Jadavpur University, India; an MTech in computer science from the Indian Institute of Technology, Kanpur; and a PhD in electrical engineering from the Swiss Federal Institute of Technology, Zurich. He is a member of the IEEE and the ACM.

Simon Künzli is a PhD student at the Computer Engineering and Networks Laboratory of the Swiss Federal Institute of Technology, Zurich. His research interests include system-level modeling languages and performance models for embedded systems, especially for network processors. Künzli has a Dipl-Ing in electrical engineering from the Swiss Federal Institute of Technology, Zurich. He is a member of the IEEE.

Lothar Thiele is a full professor of computer engineering at the Swiss Federal Institute of Technology, Zurich. His research interests include models, methods, and software tools for embedded-systems design. Thiele has a Dipl-Ing and Dr-Ing in electrical engineering from the Technical University of Munich. He is a member of the IEEE and the ACM.

■ Direct questions or comments about this article to Alexander Maxiaguine, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Gloriastrasse 35, 8092 Zurich, Switzerland; maxiagui@tik.ee.ethz.ch.