Dynamic Platform Management for Configurable Platform-Based System-on-Chips

Krishna Sekar Kanishka Lahiri Sujit Dey

Dept. of Electrical and Computer Engineering,
UC San Diego, La Jolla, CA

Abstract— **General-purpose System-on-Chip platforms consisting of configurable components are emerging as an attractive alternative to traditional, customized solutions (*e.g.*, ASICs, custom SoCs), owing to their flexibility, time-to-market advantage, and low engineering costs. However, the adoption of such platforms in many high-volume markets (*e.g*, wireless handhelds) is limited by concerns about their performance and energy-efficiency. This paper addresses the problem of enabling the use of configurable platforms in domains where custom approaches have traditionally been used. We introduce *Dynamic Platform Management*, a methodology for customizing a configurable general-purpose platform at run-time, to help bridge the performance and energy efficiency gap with custom approaches. The proposed technique uses a software layer that detects time-varying processing requirements imposed by a set of applications, and dynamically optimizes architectural parameters and platform components. Dynamic platform management enables superior application performance, more efficient utilization of platform resources, and improved energy efficiency, as compared to a statically optimized platform, without requiring any modifications to the underlying hardware.**

We illustrate dynamic platform management by applying it to the design of a dual-access UMTS/WLAN security processing system, implemented on a general-purpose configurable platform. Experiments demonstrate that, compared to a statically optimized design (on the same platform), the proposed techniques enable upto 33% improvements in security processing throughput, while achieving 59% savings in energy consumption (on average).

I. Introduction

General-purpose configurable platforms refer to System-on-Chips featuring general-purpose components and/or architectural parameters that can be customized towards the requirements of a specific application (or applications). Examples of components that might be included in such a platform are configurable processors, parameterized caches, reconfigurable memory hierarchies, flexible bus architectures, programmable logic, and parameterized co-processors. Figure 1 illustrates how such platforms potentially combine the best benefits of general-purpose and custom design styles. While customized, application-specific hardware (*e.g.*, ASICs, custom SoCs) are suited to providing high performance, low power, and small size, they are usually associated with heavy engineering costs, slow time-to-market, and an inability to provision for post-deployment upgrades (hence reduced time-in-market). At the other extreme, solutions based entirely on general-purpose processors provide high degree of flexibility, enabling upgrades, and shorter development cycles, but often fall short of performance and power requirements. Domain-specific platforms attempt to bridge this gap, featuring architectures that are optimized for certain application areas (*e.g.*, network [1], security [2], multimedia processors [3]). However, the costs of developing new platform architectures for each domain remains substantial.

General-purpose configurable platforms are significantly more flexible than their domain-specific counterparts, permitting targeting of the same hardware to multiple applications. However, Figure 1 shows that a significant difference can exist between the performance and power characteristics of such platforms and more specialized solutions, when used in a naive manner. Techniques for application-specific customization of general-purpose platforms (such as the approach described in this paper) are therefore essential to help bridge this gap.

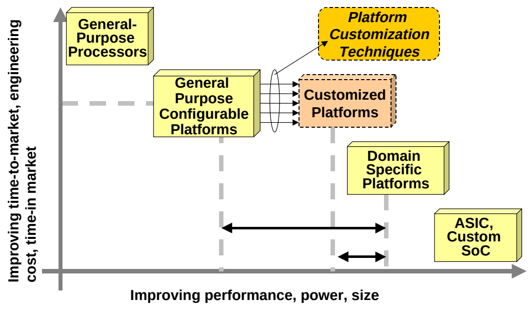

General-Purpose Processors
Platform Customization Techniques
General Purpose Configurable Platforms
Customized Platforms
Domain Specific Platforms
ASIC, Custom SoC
Improving time-to-market, engineering cost, time-in market
Improving performance, power, size

Fig. 1. Importance of platform customization for configurable platform-based System-on-Chip design

One of the areas in which such platforms are expected to play an important role is the function rich, high data rate, wireless handset market. In this application domain, severe limitations on device cost, size and power consumption, together with the need for high performance (due to more demanding wireless applications), software upgradability (due to evolving standards, emerging applications), and short product cycles make configurable platforms an attractive approach. Recently, several general-purpose configurable platforms have become available [4], [5], [6]. However, these platforms are typically configured statically (at design time), and moreover, often require changes to, or optimization of, the underlying platform hardware (*e.g.*, the number of processor functional units). Recognizing the need for greater (run-time) configurability, products featuring components and parameters that can be configured or programmed on-the-fly have started to appear [7], [8].

The need for dynamic configurability in SoC platforms stems from the increasing number of domains in which embedded systems need to execute multiple (possibly concurrent) applications (*e.g.*, wireless handhelds). Since different applications can have widely different characteristics, significant temporal variation may occur in the manner in which underlying platform resources (*e.g*, CPU, memory) are used, depending on which application is executing, or the concurrent mix of applications. In addition, individual

This work was supported by the National Science Foundation, grant 99-12414

applications, and their operating environments, can impose a wide range of processing requirements, due to variations in performance criteria, available battery capacity, and properties of the data being processed. If platform-based SoCs are to achieve semi-custom performance and power goals, it is imperative for the system to recognize, and adapt the underlying architecture to such changing requirements. While the development of dynamically configurable platforms is an important and active area of research, the focus of our work is complementary, addressing techniques for the run-time management and customization of platform components and parameters.

A. Paper Overview and Contributions

This paper introduces the concept of *dynamic platform management*, a methodology for run-time customization of a general-purpose configurable platform, with the aim of providing applications running on the platform, with performance and/or energy consumption characteristics that are associated with more customized system implementations. Our techniques aim at enabling the use of a general-purpose platform across numerous application domains, without incurring the cost and effort of re-designing, or introducing new platform hardware. Dynamic platform management is implemented as a layer of customized software that understands and exploits knowledge of the applications and their characteristics, in order to optimally manage and configure platform resources (Figure 2). In this approach, targeting a platform to a new application domain involves customizing the platform management algorithms, hence, avoiding the large development costs associated with hardware design/customization. Also, adding new applications to a deployed platform involves only upgrading the platform management layer, along with application software.

In this paper, we present a methodology for dynamic management of a general-purpose platform that supports run-time configuration of a flexible memory architecture, fine-grained frequency setting, and supply voltage scaling. The methodology enables optimized usage of available CPU and on-chip memory resources. It consists of two parts. In the first part, we develop detailed (off-line) characterizations of how platform resource usage (*e.g.*, CPU cycles, memory accesses) varies with the characteristics of the set of executing tasks. The second part consists of dynamic platform management algorithms that customize the platform for time-varying application requirements. We demonstrate how our techniques can achieve significant gains in performance and energy efficiency by applying it to the design of a dual-access UMTS/WLAN wireless

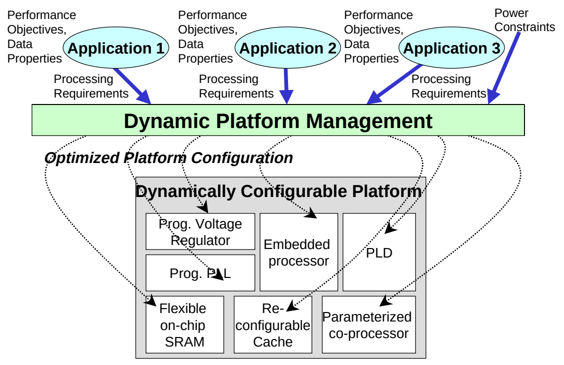

Performance Objectives, Data Properties
Application 1
Performance Objectives, Data Properties
Application 2
Performance Objectives, Data Properties
Application 3
Power Constraints
Processing Requirements
Processing Requirements
Processing Requirements
Dynamic Platform Management
Optimized Platform Configuration
Dynamically Configurable Platform
Prog. Voltage Regulator
Prog. PLL
Embedded processor
PLD
Flexible on-chip SRAM
Re-configurable Cache
Parameterized co-processor

Fig. 2. Dynamic platform management: run-time customization of a general-purpose, configurable platform

security processing system. Experiments demonstrate that, compared to a conventionally optimized design (on the same general-purpose platform), the proposed techniques enable upto 33% improvements in security processing throughput, while achieving 59% energy savings (on average).

In the next section, we describe the configurable platform architecture that we consider in this paper. In Section III, we illustrate, using the UMTS/WLAN security processing system as an example, the execution of the system based on dynamic platform management, and the advantages it provides. In Section IV, we present details of the methodology, considering the key steps, and algorithms employed. Finally, in Section V we present experimental results that evaluate the performance and energy-efficiency of the developed security processing system, and compare it to a conventionally optimized design.

B. Related Work

Increasing interest in configurable platforms has led to the development of technologies for configurable platform architectures, as well as platform customization techniques. The latter include techniques that modify or make additions to the underlying platform hardware to better suit application-specific characteristics. These “hardware centric” customization approaches can result in the creation of new instructions and functional units [4], co-processors [5], customized memory subsystems [9], *etc*. In our approach, the platform hardware is pre-designed, but is customized at run-time using software. The approach taken by [10] is similar to ours in that they also consider pre-designed, general-purpose platforms, and provide a methodology for optimizing (or “tuning”) architectural parameters. However, they focus on statically optimizing the platform for a particular application, and do not consider dynamic configurability.

Recognizing the advantages of dynamic customization, numerous technologies have emerged, or are emerging, for platforms featuring components that can be dynamically configured. These include configurable processors [7], caches [11], [12], [13], adaptive on-chip communication architectures [14], [15], and specialized hardware for logic-level (FPGAs) or micro-architectural reconfigurability [16], [17], [18]. In our work, we assume the availability of an underlying platform consisting of readily-available, general-purpose components, that can be configured dynamically, and focus on techniques for run-time management, and simultaneous configuration of these components.

It bears mentioning that policies for configuring individual architectural parameters and components, such as frequency, voltage, cache/memory organization have been proposed [9], [19]. In this paper we consider the simultaneous configuration and optimization of multiple platform resources. Policies for managing the power states of components along with frequency/voltage scaling have been proposed for saving energy [20]. However, our platform management technique focuses on optimally configuring both the memory architecture and frequency/voltage setting to improve the performance as well as the energy efficiency of the system.

II. Configurable Platform Architecture

The configurable platform architecture that we consider features fine-grained, run-time settable voltage and frequency, an embedded processor (the StrongARM1 core [7]), and a flexible memory architecture, containing a small, fast, on-chip data memory (Figure 3). Dedicated on-chip memories are often used in embedded

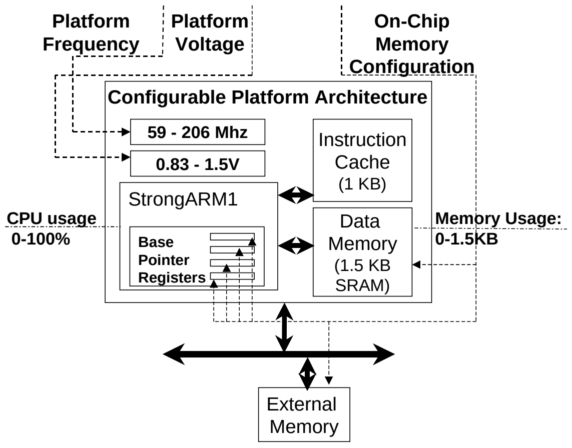


Fig. 3. A general purpose, configurable platform architecture, featuring dynamically configurable on-chip memory, platform voltage and frequency

systems in place of data caches, to reduce power consumption, exploit application characteristics, and obtain more predictable execution [21]. The platform architecture supports dynamic partitioning of data items used by the various tasks between the fast on-chip data memory (SRAM) and slower external memory. This provides the flexibility of selecting at run-time, the optimum set of data items (*e.g.*, the most frequently accessed) for storage in the on-chip memory. The space of platform configurations is defined by two dimensions. The first dimension consists of different $\langle f, v\rangle$ pairs, which define clock frequencies and their associated supply voltage values. The second dimension consists of the different ways in which the set of data items (accessed by the tasks) can be partitioned between on-chip and external memory. For a set of data items $D = \{D_1, D_2, ..., D_n\}$, a configuration of the on-chip memory is defined by a set $D_{on} \subseteq D$, such that the total size of items in D_{on} does not exceed the capacity of the on-chip memory. A candidate platform configuration is defined by $\langle f, v, D_{on}\rangle$. A configuration is successful in meeting performance requirements, if the associated CPU utilization is no more than 100%.

III. Demonstrating Platform Management for Security Processing

In this section, we illustrate the operation and advantages of our dynamic platform management technique using a dual-access (UMTS/WLAN) security processing system as an example. We first describe the security processing tasks, and their mapping to the platform architecture. Next we highlight the space of available platform configurations. We then illustrate the problems associated with configuring the platform statically, and finally, illustrate how dynamic platform management chooses optimized platform configurations at run-time, and thereby achieves desired security processing throughput, and improvements in energy-efficiency.

A. Example: UMTS/WLAN Security Processing

The system implements Layer 2 security protocols of two wireless standards: the UMTS standard for third generation cellular networks [22], and the IEEE 802.11b standard for wireless LANs [23]. Our design is motivated by the projected emergence of converged handsets, capable of simultaneous communication over multiple wireless interfaces [24]. The need to support upgrades (due to the evolving nature of security protocols), while achieving high security processing throughput and energy-efficiency, makes a general-purpose configurable platform a suitable implementation choice.

The dual-access security processing system executes two tasks. The UMTS task is responsible for ciphering and integrity of UMTS frames [22]. Each UMTS frame may be ciphered and integrity checked, or only ciphered, depending on the frame type. All frames are ciphered using the *f8* algorithm. In addition, signalling frames are also integrity checked using a 32-bit Message Authentication Code, computed using the *f9* algorithm. Both *f8* and *f9* are are based on the *KASUMI* block cipher [22]. The WLAN task (defined in [23]) is responsible for encrypting Layer 2 frames (if encryption is enabled) using the Wired Equivalent Privacy (WEP) protocol, which is based on a 64 bit symmetric key stream cipher, and computing a Frame Check Sequence (a 32 bit CRC) for data integrity. While processing a UMTS (or WLAN) frame, the UMTS (or WLAN) task accesses a set of data items (*e.g.*, CRC tables, substitution tables, the program stack, frame data *etc*).

The security processing tasks (UMTS/WLAN) are mapped to the configurable platform described in the previous section. The tasks execute on the StrongARM core, and the various data items are partitioned between the on-chip and external memory (since all the items cannot be accommodated in the on-chip memory). The tasks individually exhibit significant dynamic variation in their CPU requirements, and memory access patterns, due to variable data rates, frame types and frame size distributions.

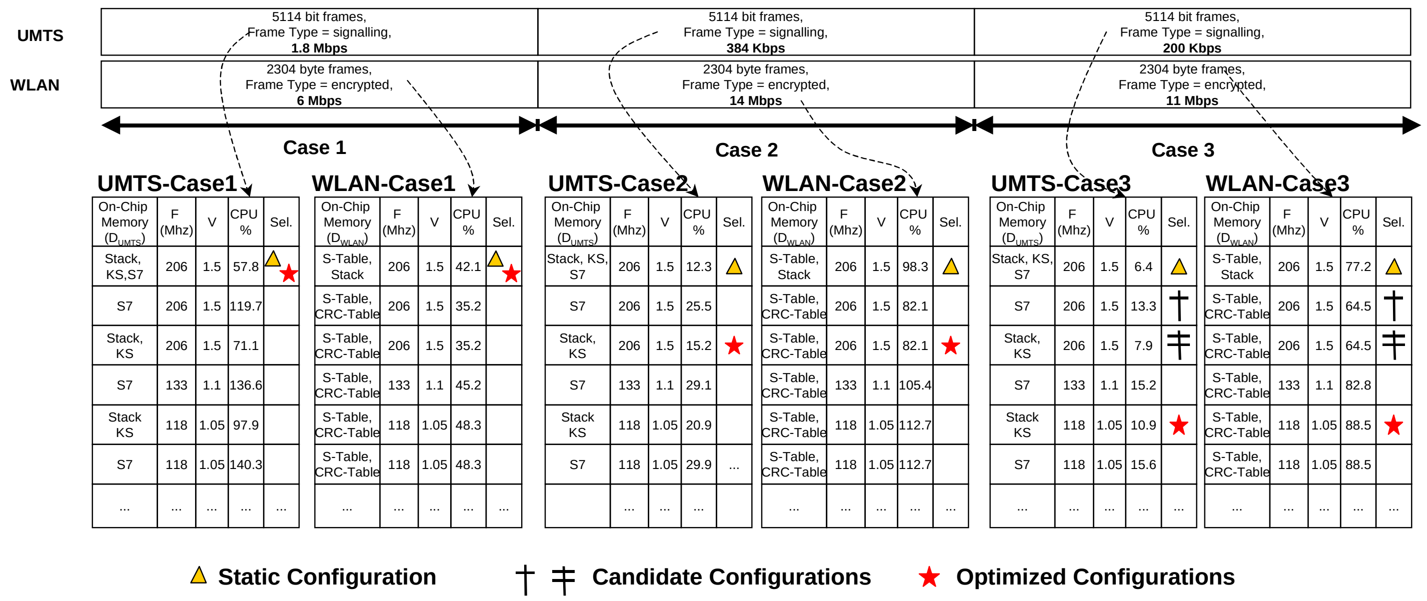
B. Platform Configuration Space

Figure 4 depicts three workload scenarios, labeled **Case 1**, **Case 2** and **Case 3**. In all the cases, both the UMTS and WLAN tasks are active, and are processing respective frames. In this example, the frame characteristics are held constant (as indicated in Figure 4), while the data rate associated with each task varies as shown. Below each case, we illustrate a pair of tables, one for each of the two tasks. The UMTS table for **Case 1** depicts a space of possible platform configurations $\langle f, V, D_{UMTS}\rangle$, with the CPU% consumed by the UMTS task under the given workload. For example, row 1 of Table UMTS-Case1 indicates that if the platform is operated at 206 Mhz, 1.5 V, and if the Stack, Key Schedule, and S7 data items are stored in the on-chip memory, then while processing 5114 bit frames (of type signalling) at 1.8 Mbps, the UMTS task consumes 57.8% of the CPU. A similar table is presented for WLAN-Case1, and for the other cases.

At any given time, a candidate platform configuration is defined by a pair of rows, one from each table, such that the following two conditions are met: (i) the frequency and voltage in the two selected rows are identical, (ii) the total size of the data items stored in the on-chip memory is less than its capacity. A platform configuration that satisfies these conditions, and achieves no more than 100% CPU, will satisfy performance (data rate) requirements.

C. Security Processing: Static Configuration

We first consider the execution of the platform for each of the three cases depicted in Figure 4, when it is statically configured. The particular static configuration that we choose is one that is optimized for a large space of requirements (details in Section V). Figure 4 illustrates the sequence of platform configurations, showing how, in this case, the pair of "selected rows" remain fixed over time (indicated by $\triangle$). The static configuration is defined by $\langle f, V, D_{on}\rangle$, where $f = 206, V = 1.5, D_{on} = \{Stack_{UMTS}, KS_{UMTS}, S7_{UMTS}, S-Table_{WLAN}, Stack_{WLAN}\}$. Inspection of the CPU% figures for this architecture indicates the following problem. While for **Case 1**, the


UMTS
WLAN
5114 bit frames, Frame Type = signalling, 1.8 Mbps
5114 bit frames, Frame Type = signalling, 384 Kbps
5114 bit frames, Frame Type = signalling, 200 Kbps
2304 byte frames, Frame Type = encrypted, 6 Mbps
2304 byte frames, Frame Type = encrypted, 14 Mbps
2304 byte frames, Frame Type = encrypted, 11 Mbps
Case 1
Case 2
Case 3
△ Static Configuration
† ‡ Candidate Configurations
★ Optimized Configurations

UMTS-Case1

On-Chip Memory (D_{UMTS})	F (Mhz)	V	CPU %	Sel.
Stack, KS,S7	206	1.5	57.8	△ ★
S7	206	1.5	119.7	
Stack, KS	206	1.5	71.1	
S7	133	1.1	136.6	
Stack KS	118	1.05	97.9	
S7	118	1.05	140.3	
...	...	...	...	...

WLAN-Case1

On-Chip Memory (D_{WLAN})	F (Mhz)	V	CPU %	Sel.
S-Table, Stack	206	1.5	42.1	△ ★
S-Table, CRC-Table	206	1.5	35.2	
S-Table, CRC-Table	206	1.5	35.2	
S-Table, CRC-Table	133	1.1	45.2	
S-Table, CRC-Table	118	1.05	48.3	
S-Table, CRC-Table	118	1.05	48.3	
...	...	...	...	...

UMTS-Case2

On-Chip Memory (D_{UMTS})	F (Mhz)	V	CPU %	Sel.
Stack, KS, S7	206	1.5	12.3	△
S7	206	1.5	25.5	
Stack, KS	206	1.5	15.2	★
S7	133	1.1	29.1	
Stack KS	118	1.05	20.9	
S7	118	1.05	29.9	...
	...	...	...	

WLAN-Case2

On-Chip Memory (D_{WLAN})	F (Mhz)	V	CPU %	Sel.
S-Table, Stack	206	1.5	98.3	△
S-Table, CRC-Table	206	1.5	82.1	
S-Table, CRC-Table	206	1.5	82.1	★
S-Table, CRC-Table	133	1.1	105.4	
S-Table, CRC-Table	118	1.05	112.7	
S-Table, CRC-Table	118	1.05	112.7	
...	...	...	...	...

UMTS-Case3

On-Chip Memory (D_{UMTS})	F (Mhz)	V	CPU %	Sel.
Stack, KS, S7	206	1.5	6.4	△
S7	206	1.5	13.3	†
Stack, KS	206	1.5	7.9	‡
S7	133	1.1	15.2	
Stack KS	118	1.05	10.9	★
S7	118	1.05	15.6	
...	...	...	...	...

WLAN-Case3

On-Chip Memory (D_{WLAN})	F (Mhz)	V	CPU %	Sel.
S-Table, Stack	206	1.5	77.2	△
S-Table, CRC-Table	206	1.5	64.5	†
S-Table, CRC-Table	206	1.5	64.5	‡
S-Table, CRC-Table	133	1.1	82.8	
S-Table, CRC-Table	118	1.05	88.5	★
S-Table, CRC-Table	118	1.05	88.5	
...	...	...	...	...

Fig. 4. Dynamic selection of optimized platform configurations for UMTS/WLAN security processing

total CPU% is $57.76 + 42.11 = 99.87\%$, in **Case 2**, the rows corresponding to the static architecture (△) have CPU% values that add up to 110.58%, indicating that the static architecture is not capable of satisfying the processing requirements imposed by **Case 2**.

D. Security Processing: Dynamic Platform Management

We next illustrate the execution of the security processing tasks with our dynamic platform management technique, for the three cases considered in the previous example. The platform management technique considers (i) time-varying requirements imposed by the individual tasks (by examining workload parameters, such as frame sizes, types, and data rate requirements), and (ii) predetermined characteristics of each task, to choose an optimized configuration of the platform. For example, under **Case 1**, the platform management technique chooses a platform configuration that is identical to the static configuration used in the previous example. However, as the requirements imposed by the application change, the selected platform configuration may change, as illustrated next.

When the requirements change from **Case 1** to **Case 2**, the platform management techniques consider the space of possible platform configurations available to further optimize the system for **Case 2** (defined in Tables UMTS-Case2 and WLAN-Case2). Recall that a pair of rows from these uniquely defines a platform configuration. Clearly the pair of rows labeled with △ define a platform configuration that cannot meet the requirements imposed by **Case 2** (total CPU exceeds 100%). A possible configuration is the one defined by the pair of rows labeled with a ★. This configuration differs from the previous one in terms of the set of data items stored in the on-chip memory: the items $S7_{UMTS}$ and $Stack_{WLAN}$ are replaced by $CRC - Table_{WLAN}$. Under the new configuration, the CPU% consumed by the two tasks are 15.17% and 82.12%, resulting in a total of 97.29%, which is less than 100%. Hence, this is determined to be a platform configuration that satisfies **Case 2**.

However, the platform management technique does more than just select an alternate set of data items to be stored in memory. It does this in a manner so as to minimize wasted CPU cycles, and hence increase CPU availability. To illustrate this, consider **Case 3**, where in the static case, (denoted by △), the total CPU% is $6.42 + 77.20 = 83.62\%$. Even though this configuration meets performance requirements comfortably, the platform management technique considers the space of possible configurations that might result in more efficient use of the CPU. For example, to process the workload of **Case 3**, the configuration defined by the pair of rows labeled with † results in 77.82% CPU utilization, whereas the configuration defined by the pair of rows labeled with ‡ result in 72.42% CPU utilization. The platform management technique selects ‡ over †, since ‡ achieves 13% savings in CPU cycles, while † achieves only 7% savings. Reducing CPU utilization can enable (i) accommodation of other processing tasks (if they exist), or (ii) in our case, reductions in power consumption via frequency and voltage scaling. In the example, the platform management technique selects a configuration defined by the pair of rows labeled with ★. Exploiting the resultant CPU slack enables operating the platform at a lower frequency (118 Mhz) and voltage (1.05 V), leading to energy savings. Note that, if the memory configuration defined in † had been used, then the platform would have had to be operated at 133 Mhz and 1.1 V, which would have led to higher power consumption. On the other hand, if the platform was operated at 118 Mhz (the optimum speed for ‡), it would have led to a total CPU utilization of 104.1%, resulting in failure to meet performance requirements. This illustrates that the platform management technique configures the platform in a holistic manner, with simultaneous concern for memory usage, CPU utilization, and operating frequency/voltage.

From this example, we draw a few important conclusions:

- Tasks can exhibit significant dynamic variation in their workload characteristics, resulting in a wide range of processing requirements imposed on a platform. While we considered data rate as a variable parameter, the platform resource usage profiles could vary significantly due to several other factors (*e.g,*, the exact set of tasks currently active, and frame properties, such as frame types, sizes, *etc*).
- Dynamically configuring the platform while exploiting (i) an

accurate characterization of application tasks, and (ii) a detailed knowledge of the platform architecture can help improve platform resource utilization (CPU, memory), resulting in performance improvements, and large energy savings.

- Platform management techniques should be based on tightly coupled algorithms for dynamically optimizing different components and parameters for maximizing performance and energy-efficiency. The example illustrated how usage profiles of different platform components are interdependent, and demonstrated how intelligent use of an on-chip memory helps free up CPU cycles (by reducing the number of slow external memory accesses), enabling larger CPU headroom, or potential power savings.

IV. Dynamic Platform Management Methodology

In this section, we describe the details of the dynamic platform management methodology. We first define terminology, and then go on to describe (i) the off-line task characterization step, and (ii) the dynamic platform management algorithms that optimize the platform configuration at run-time, so as to meet performance requirements, while minimizing energy consumption.

A. The Model

We consider tasks with periodic arrivals, having soft real-time requirements. At a given time, let the set of currently executing tasks be denoted by $T = \{T_1, T_2, ..., T_N\}$, where $N \leq MAX$, the total number of tasks in the system. Associated with each task T_i, where $1 \leq i \leq MAX$, is a set of data items, $D_i = \{d_{i,1}, \ldots, d_{i,m_i}\}$. A data item refers to a logical data structure, or data block, that can be addressed contiguously by the task. For example, the CRC table used for computing the 32 bit checksum in the WLAN task is a data item. Each data item, $d_{i,j}$, has an associated maximum size $s_{i,j}$. Each instance of a task T_i, has a time-interval P_i, within which it has to finish executing. Let $N_i = \{n_{i,1}, n_{i,2}, ..., n_{i,m_i}\}$ denote a set consisting of the number of accesses T_i makes to each data item in D_i in time-interval P_i (*i.e.*, there is a one-one mapping between D_i and N_i). Let C_i denote the number of processing cycles (excluding data memory access cycles) required by each instance of the task T_i in the time-interval P_i. Given D_{on}, the set of data items in on-chip memory, the execution time ET_i for each instance of a task T_i is estimated using the following:

$$ET_i = (C_i + nC_{on-chip} + (N-n)(\lceil \frac{T_{ext}}{1/f} \rceil)) * \frac{1}{f} \qquad (1)$$

where C_i is the number of processing cycles, $C_{on-chip}$ is the number of cycles to access on-chip data memory, T_{ext} is the time to access external memory, $N = \sum N_i$ is the total number of data accesses, $n = \sum_{d_{i,j} \in D_{on}} n_{i,j}$ is the number of data accesses to on-chip data memory, and f is the operating frequency. The non-linear effects due to external memory are accounted for by the $\lceil \frac{T_{ext}}{1/f} \rceil$ term. The values of $C_{on-chip}$ and T_{ext} depend on the platform.

In order to dynamically determine the optimized platform configuration, the platform management algorithms are provided with certain characteristics of the tasks which execute on the platform. This task specific information is obtained by characterizing each task off-line, using a procedure we describe next.

B. Off-line Task Characterization

The steps performed in this off-line phase are illustrated by the first box in Figure 5. For each task, $T_i, 1 \leq i \leq MAX$, which potentially executes on the platform, all the data addressed by the task is first divided into a set of logical data items, D_i. Each data item $d_{i,j} \in D_i$ should be contiguously addressable by the task. Next, the maximum sizes of these data items ($s_{i,j}$ values) are determined through a combination of analysis and simulation. Each data item $d_{i,j}$ is then characterized by the number of times, $n_{i,j}$, it is accessed by an instance of its associated task T_i. In general, the value of $n_{i,j}$ varies, depending on dynamically variable parameters of the task, and properties of the data being processed. For example, in a security processing system, the parameters may include the encryption key lengths, frame lengths and frame types. Hence, the result of this step is a set of formulae/models for estimating the number of accesses to each data item at run-time. In addition, similar models for estimating the number of processing cycles, C_i, for each task instance are developed. Note that, we base our estimation models on parameters whose values can be obtained at task arrival. In certain cases, the tasks characteristics may depend on the actual data values, in which case, accurate off-line estimation may prove difficult. While considering such tasks is beyond the scope of this paper, we believe that our methodology can be extended to incorporate predictive strategies for estimating task characteristics at run-time (*e.g.*, [25]).

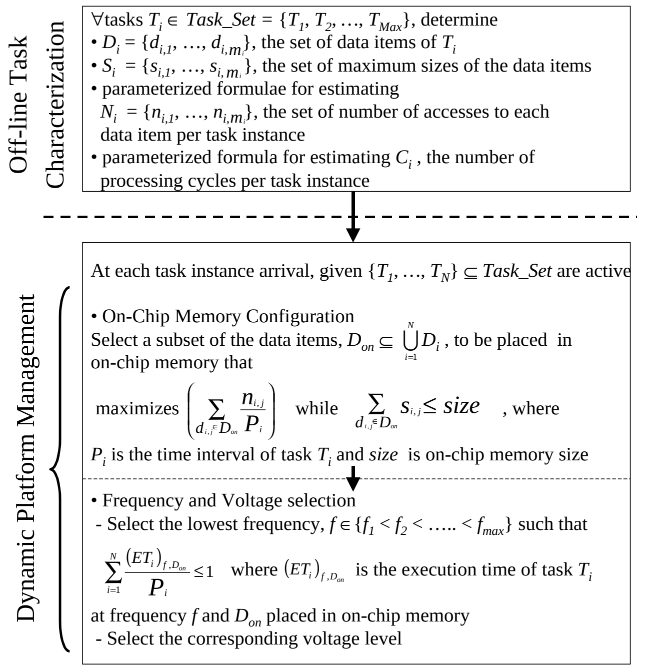

Off-line Task Characterization
∀tasks $T_i \in$ Task_Set = $\{T_1, T_2, \ldots, T_{Max}\}$, determine
• $D_i = \{d_{i,1}, \ldots, d_{i,m_i}\}$, the set of data items of T_i
• $S_i = \{s_{i,1}, \ldots, s_{i,m_i}\}$, the set of maximum sizes of the data items
• parameterized formulae for estimating $N_i = \{n_{i,1}, \ldots, n_{i,m_i}\}$, the set of number of accesses to each data item per task instance
• parameterized formula for estimating C_i, the number of processing cycles per task instance
Dynamic Platform Management
At each task instance arrival, given $\{T_1, \ldots, T_N\} \subseteq$ Task_Set are active
• On-Chip Memory Configuration
Select a subset of the data items, $D_{on} \subseteq \bigcup_{i=1}^{N} D_i$, to be placed in on-chip memory that
maximizes $\left(\sum_{d_{i,j} \in D_{on}} \frac{n_{i,j}}{P_i}\right)$ while $\sum_{d_{i,j} \in D_{on}} s_{i,j} \leq size$, where
P_i is the time interval of task T_i and $size$ is on-chip memory size
• Frequency and Voltage selection
- Select the lowest frequency, $f \in \{f_1 < f_2 < \ldots < f_{max}\}$ such that
$\sum_{i=1}^{N} \frac{(ET_i)_{f,D_{on}}}{P_i} \leq 1$ where $(ET_i)_{f,D_{on}}$ is the execution time of task T_i
at frequency f and D_{on} placed in on-chip memory
- Select the corresponding voltage level

Fig. 5. Dynamic platform management methodology

Table I shows the results of performing this characterization step for the security processing tasks: *UMTS* ciphering and integrity, and *WLAN* encryption and checksum. The data items accessed by the *UMTS* task are its Stack ($Stack$), the Key Schedule (KS), the $S7$ and $S9$ lookup tables, and the *UMTS* frame. The *WLAN* data items include its Stack ($Stack$), the State Table ($S-Table$), the CRC Table ($CRC-Table$), and the $WLAN$ frame. The number of accesses per data item and the processing cycles requirement for *UMTS* depend on both the frame size (l) and the type of frame (for user frames, only the first parenthesized term is used, while for signalling frames, both the terms are used). The corresponding estimation formulae for *WLAN* depend only on the frame size.

Once these characterization tables are generated, they are incor-

TABLE I

CHARACTERIZATION OF THE SECURITY PROCESSING TASKS

Task	Data Items ($d_{i,j}$)	Max Size ($s_{i,j}$)	Estimated Number of Accesses ($n_{i,j}$)	Proc. Cycles (C_i)
UMTS	Stack	120	$(19.5\lceil l/8 \rceil+520.5)+(18.5\lceil l/8 \rceil+726.5)$	$(146.15\lceil l/8\rceil +3176.71)+(131.95\lceil l/8\rceil +4796.72)$
	KS	128	$(64\lfloor l\text{-}1/64 \rfloor+256)+(64\lfloor l+1/64 \rfloor+320)$	
	S7	256	$(48\lfloor l\text{-}1/64 \rfloor+96)+(48\lfloor l+1/64 \rfloor+144)$	
	S9	1024	$(48\lfloor l\text{-}1/64 \rfloor+96)+(48\lfloor l+1/64 \rfloor+144)$	
	Frame	1340	$(2\lceil l/8 \rceil+35)+(\lceil l/8 \rceil+40)$	
WLAN	Stack	96	272	$36l+5807$
	S-Table	256	$10l+1322$	
	CRC-Table	1024	$2l+40$	
	Frame	4654	$4l+54$	

porated into the dynamic platform management algorithms (Figure 5), which use them to optimize the platform configuration at run-time. Note that, in order to incorporate a new task into the platform, or to target the platform to a new set of application tasks, the platform management implementation remains the same. Only this off-line characterization step needs to be performed to generate the required tables, and provided to the platform management algorithms. In the next subsection, we describe how these characterization tables are used to select optimized platform configurations at run-time.

C. Dynamic Platform Management Algorithms

At the arrival of each task instance, the dynamic platform management techniques choose an optimized configuration by (i) deciding on the on-chip memory configuration, and (ii) calculating a "memory-aware" frequency and voltage setting (Figure 5). We next describe how the memory, and frequency/voltage decisions are taken. A pre-emptive EDF scheduler [26] is assumed for scheduling the arriving tasks.

C.1 On-Chip Memory Configuration

The task of configuring the on-chip memory consists of dynamically selecting the optimal set of data items, D_{on}, to be placed in on-chip memory given a set of executing tasks. Given two different on-chip memory configurations, D_{on1} and D_{on2}, the one which reduces the CPU stall cycles as it waits for external memory is better, since it results in more efficient CPU utilization. It can be shown that D_{on1} is preferable to D_{on2}, if :

$$\sum_{i=1}^{N} \frac{ET_{i,D_{on1}}}{P_i} < \sum_{i=1}^{N} \frac{ET_{i,D_{on2}}}{P_i} \tag{2}$$

where N is the number of currently executing tasks. Using Equation 1 and solving the above inequality, we get:

$$\sum_{d_{i,j}\in D_{on1}} \frac{n_{i,j}}{P_i} < \sum_{d_{i,j}\in D_{on2}} \frac{n_{i,j}}{P_i} \tag{3}$$

Equation 3 holds, provided on-chip memory access time is less than external memory access time (which is true). Hence, that on-chip memory configuration, which maximizes the rate of on-chip memory accesses, is optimal. This is subject to the constraint that the set of selected data items fit within the limited on-chip memory, *i.e.*, $\sum_{d_{i,j}\in D_{on}} s_{i,j}$ should not exceed the size of the on-chip memory (Figure 5). The problem of optimizing the on-chip memory can be formulated in terms of the *Knapsack* problem (which is *NP*-complete [27]). Hence, we use a greedy strategy to dynamically choose the set D_{on} by using $\frac{n_{i,j}/P_i}{s_{i,j}}$, the ratio of the rate of memory accesses to the size of the data item, as the cost function.

Since optimizing the on-chip memory configuration frees up CPU cycles, this enables the possibility of operating the CPU at a lower frequency and voltage setting.

C.2 Frequency and Voltage Setting

After optimizing the on-chip memory configuration, the dynamic platform management layer selects the frequency and voltage at which to operate the platform (Figure 5). Accurate off-line characterizations of the tasks, and knowledge of the currently selected memory configuration, enables the platform management layer to aggressive scale the operating frequency and voltage. We use the schedulability test for EDF [26] to determine the lowest frequency f (among a set of discrete frequencies of the platform, $f_1 < f_2 < ... < f_{max}$) at which the set of active tasks can still meet their performance requirements. The frequency is determined from the following:

$$\sum_{i=1}^{N} \frac{(ET_i)_{f,D_{on}}}{P_i} = 1; \tag{4}$$

where $(ET_i)_{f,D_{on}}$ is the execution time of the currently active instance of task T_i, under frequency f and with the set of data items D_{on} in on-chip memory (from Equation 1), and P_i is the execution time-interval of T_i. The voltage level is selected corresponding to the selected frequency setting.

C.3 Platform Management Overhead

The overhead associated with a platform configuration decision involves (i) the time taken to re-program the platform frequency and voltage, and (ii) the time taken to reconfigure the on-chip memory. For our platform, the time for (i) was assumed to be the time taken to change the frequency of the StrongARM processor (approximately 150 μs [20]). The worst-case time for (ii) was estimated to be approximately 200 μs, which occurs when the entire contents of the on-chip memory are re-organized.

Platform management decisions are potentially taken at the arrival of each task instance or frame. If the times between successive frame arrivals are large, (> 10 *ms*), then platform configuration decisions can be made at each frame arrival with insignificant overhead. However, if the task time-periods are small (say hundreds of μs) then the overhead of platform configuration may out-weigh its benefits. In such cases, platform configuration decisions need to be taken at coarser time-scales. An example policy in such cases is one where the platform management decisions are taken at the arrival of the first frame following expiration of a fixed time interval. Since workload characteristics of future frames are unknown, frame sizes and types from the previous interval are used to estimate future frame characteristics. In our experiments, we used a 10 *ms* interval.

V. EXPERIMENTAL RESULTS

In this section, we present experimental results that evaluate the effectiveness of applying the proposed platform management techniques to the dual-access security processing system described in Section III.

A. Experimental Methodology

Optimized C implementations of the UMTS and WLAN security algorithms were compiled using the ARM C compiler *armcc* [28] (with maximum optimization) and were targeted to the StrongARM1 [7] based platform described in Section II. The experiments consider two variants of the example system. The first variant consists of a statically optimized configuration of the platform, where the platform is operated at 206 Mhz and 1.5V, with the on-chip memory configured as shown in row 1 of the tables in Figure 4. The second variant incorporates the dynamic platform management techniques described in Section IV, where the on-chip memory, frequency and voltage are determined at run-time. The overhead of frequency and voltage setting was assumed constant at 150 μs, while that of repartitioning data items was determined dynamically, based on the size of data, and memory access times. Accesses to internal memory are single cycle, and external memory access time is 50 *ns*. Performance under a provided workload was measured using cycle-accurate instruction-level simulation of the platform architecture in Figure 3 (using *ARMulator* [28]). Power was measured using a cycle-accurate, software energy profiling tool, *JouleTrack* [29].

B. Impact of Dynamic Platform Management on Performance

This experiment evaluates improvements in security processing throughput made possible by the proposed techniques. For this experiment, we considered the "space" of possible data rates of the UMTS and WLAN security processing tasks. We keep other characteristics, such as frame sizes and types constant. For each architecture, we measured the maximum pairwise achievable data rates (those achieved when the platform is maximally utilized). Figure 6 presents these results for the considered data rate space. The region on the left of each 100% CPU contour indicates data rate pairs that the corresponding architecture can satisfy.

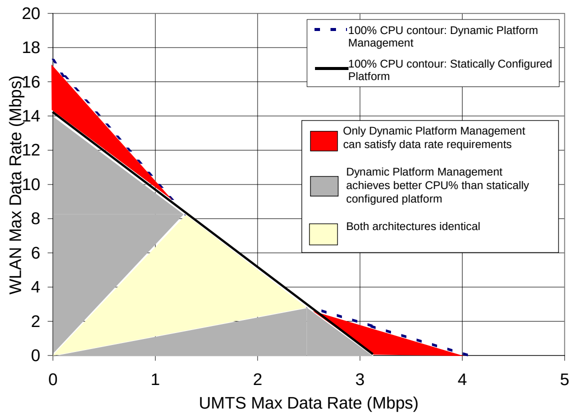


Fig. 6. Dynamic platform management: (i) space of maximum achievable data rates; (ii) space of CPU efficiency advantage

From the figure, we observe that dynamic platform management enables the security processing system to satisfy a larger space of data rate combinations, compared to the statically configured system, facilitating large improvements in performance. For example, (in the absence of WLAN) while the static configuration can achieve 3.1 Mbps UMTS throughput, the dynamic case can sustain a maximum data rate of 4.1 Mbps, a 33% improvement. Similarly, upto 21% improvements can be achieved for WLAN data rates. We exhaustively examined all possible static configurations of the platform, and found this configuration to be the one that meets the largest space of data rate requirements. By outperforming this static configuration, the dynamic platform management technique demonstrates that it can achieve significant performance gains over any static configuration.

C. Impact of Dynamic Platform Management on CPU Load

In practice, situations may often arise where data rate requirements are far lower than the maximum achievable values. We next demonstrate that even in cases where performance requirements can be met by the static configuration, it is still advantageous to use the dynamic platform management techniques. Figure 6 indicates portions of the data rate space, where both architectures can meet performance requirements (area to the left of the 100% contour for the static configuration). However, for a large fraction of this space (shaded grey), dynamic platform management results in fewer cycles being expended, hence improving CPU availability.

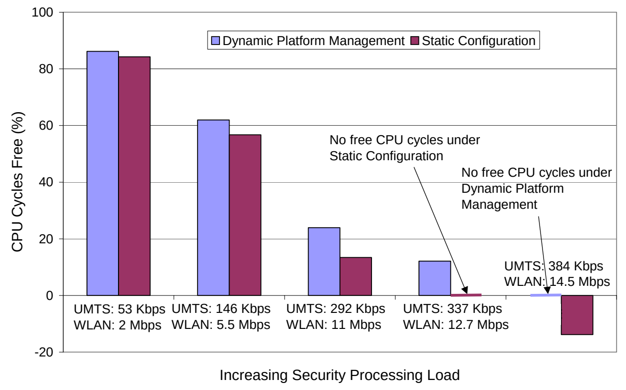


Fig. 7. CPU fraction left-over from security processing, under (i) dynamic platform management, and (ii) static configuration, for increasing security processing load

To quantify this advantage, we performed an experiment where a set of discrete data rate pairs were considered. For each pair, we measured the fraction of CPU cycles left over from security processing, under the static configuration and with dynamic platform management. The results of these experiments, along with the data rate pairs used (in increasing order of imposed load), are presented in Figure 7. The figure shows that for all cases, the dynamic architecture makes more CPU cycles available than the static architecture. The effect is more significant at higher CPU loads. For example, in the case $\langle 337 Kbps, 12.7 Mbps\rangle$, while the static case "just" meets the requirements (CPU availability is 0%), in the dynamic case, the same requirements are met, with 12.14% of the CPU left over. The increased availability of the CPU (free cycles) can be exploited to process other tasks, or reduce frequency/supply voltage, and hence reduce power consumption.

The case $\langle 384 Kbps, 14.5 Mbps\rangle$ is of special interest. The results of Figure 7 show that the static architecture needs 14% more of the CPU than is available, hence cannot meet the requirements imposed by the data rates. However, for this case, dynamic platform management chooses an optimized platform configuration, which enables satisfying the imposed requirements.

In summary, Figures 6 and 7 demonstrate that the proposed dynamic platform management techniques (i) increase the space of maximum achievable performance of a configurable platform, (ii) result in more efficient use of the CPU, and (iii) enable satisfying requirements than cannot be met by the statically optimized design.

D. Impact of Dynamic Platform Management on Energy

In our final experiment, we evaluated the power savings made possible with the proposed dynamic platform management techniques. For this experiment, we considered a dynamically varying workload consisting of varying tasks, data rate requirements, randomly varying frame size and types (Figure 8(a)). We compared the total energy consumed by the static configuration with that consumed with dynamic platform management while processing this workload. Figure 8(b) illustrates how two of the platform parameters (frequency and on-chip memory) vary with time in the dynamic case, and Figure 8(c) plots a time profile of the total energy consumption. From Figure 8(c), we observe that the dynamically configured architecture achieves 59% energy savings compared to the static case. In both cases, all performance requirements were met.

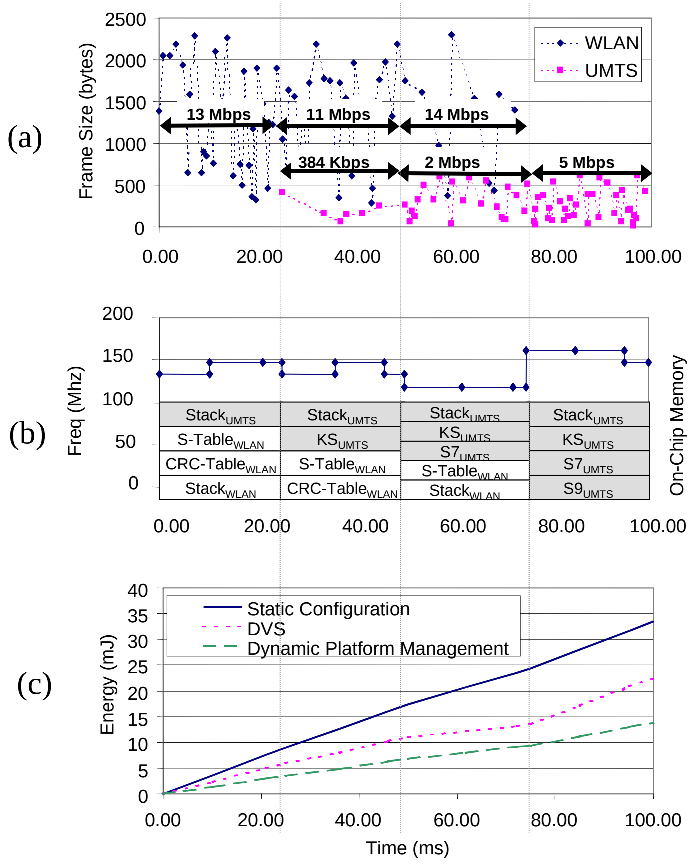


Fig. 8. Energy savings using dynamic platform management: (a) varying UMTS/WLAN workload; (b) platform configuration sequence; and (iii) cumulative energy profile

It should be noted that the savings in energy consumption are due, in large part, to careful exploitation of the interdependence between on-chip memory configuration, CPU slack, and voltage scaling. To evaluate the benefit of our integrated approach, we measured the energy savings via traditional dynamic voltage scaling (DVS) [19], while keeping the on-chip memory configuration constant. From the cumulative energy plot corresponding to DVS in Figure 8(c) we observe that dynamic platform management achieves 39% energy savings over DVS. These results demonstrate that the described platform management approach can be used to enhance system energy-efficiency over and above conventional techniques.

VI. Conclusions

This paper proposed dynamic platform management techniques for run-time customization of general-purpose configurable platforms. We demonstrated the benefit of the proposed techniques on a dual-access security processing system. In future work, we will develop an efficient implementation of the platform management techniques incorporating additional configurable components, and use it to re-target a general-purpose platform to different applications, comparing their performance with corresponding custom implementations.

VII. Acknowledgements

We would like to thank our colleague Abhishek Agarwal for his valuable contribution and comments on this paper.

References

[1] "Intel IXA Network Processors." http://www.intel.com/design/network/products/npfamily.
[2] N. R. Potlapally, S. Ravi, and A. Raghunathan, "System design methodologies for a wireless security processing platform," in *Proc. Design Automation Conf.*, pp. 777–782, June 2002.
[3] "Trimedia Technologies." http://www.trimedia.com.
[4] "Xtensa Architecture and Performance." http://www.tensilica.com/Xtensa_white_paper.pdf.
[5] V. Kathail, S. Aditya, R. Schreiber, B. R. Rau, D. C. Cronquist, and M. Sivaraman, "PICO: Automatically Designing Custom Computers," *IEEE Computer*, vol. 35, pp. 39–47, Sept. 2002.
[6] "Excalibur System on Programmable Chip Data Sheets." http://www.altera.com/literature/ds/ds_arm.pdf.
[7] "Intel SA-1110 Processor." http://www.intel.com/design/pca/applicationsprocessors/1110_brf.htm.
[8] A. Malik, B. Moyer, and D. Cermak, "The MCORE M340 unified cache architecture," in *Proc. Intl. Conf. on Computer Design*, pp. 577–580, Sept. 2000.
[9] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg, "Data and Memory Optimization Techniques for Embedded Systems," *ACM Trans. Design Automation Electronic Systems*, vol. 6, pp. 149–206, Apr. 2001.
[10] T. D. Givargis and F. Vahid, "Platune: A Tuning Framework for System-on-a-Chip Platform," *IEEE Trans. Computer-Aided Design*, vol. 21, Nov. 2002.
[11] R. Balasubramonian, D. H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures," in *International Symposium on Microarchitecture*, pp. 245–257, 2000.
[12] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji, "Adapting Cache Line Size to Application Behavior," in *Proc. Int. Conf. on Supercomputing*, June 1999.
[13] C. Zhang, F. Vahid, and W. Najjar, "A Highly-Configurable Cache Architecture for Embedded Systems," in *Proc. Int. Symp. Computer Architecture*, June 2003.
[14] "The Avalon Bus Specification." http://www.altera.com/literature/manual/mnl_avalon_bus.pdf.
[15] K. Lahiri, A. Raghunathan, G. Lakshminarayana, and S. Dey, "Communication Architecture Tuners: A Methodology for the Design of High Performance Communication Architectures for System-on-Chips," in *Proc. Design Automation Conf.*, pp. 513–518, June 2000.
[16] "Quicksilver Technologies." http://www.qstech.com.
[17] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. Taylor, "PipeRench: A Reconfigurable Architecture and Compiler," *IEEE Computer*, vol. 33, Apr. 2000.
[18] "PACT XPP Technologies." http://www.pactcorp.com.
[19] T. Pering and R. Broderson, "Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System," in *Proc. Int. Symp. Computer Architecture*, June 1998.
[20] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. D. Micheli, "Dynamic Voltage Scaling and Power Management for Portable Systems," in *Proc. Design Automation Conf.*, June 2001.
[21] R. Banakar, S. Steinke, B. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad Memory: A Design Alternative for Cache On-Chip Memory in Embedded Systems ," in *Proc. International Symposium on Hardware/Software Codesign*, pp. 73–78, May 2003.
[22] "Specifications of the 3GPP Confidentiality and Integrity Algorithms (Documents 1 and 2)." http://www.3gpp.org/TB/Other/algorithms.htm.
[23] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications." IEEE Computer Society LAN/MAN Standards Committee, IEEE Std 802.11-1999 Edition.
[24] K. Balachandran, "Convergence of 3G and WLAN." IEEE Intl. Conf. on Communications, May 2002, http://www.icc2002.com/notes.html.
[25] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram, "Frame-Based Dynamic Voltage and Frequency Scaling for an MPEG Decoder," in *Proc. Int. Conf. Computer-Aided Design*, pp. 732–737, Nov. 2002.
[26] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, 1973.
[27] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
[28] "ARM Developer Suite (ADS) version 1.2." http://www.arm.com/devtools/ADS.
[29] A. Sinha and A. P. Chandrakasan, "JouleTrack - A Web Based Tool for Software Energy Profiling," in *Proc. Design Automation Conf.*, pp. 220–225, June 2001.