[12] B. Stine, D. Boning, and J. Chung, "Analysis and decomposition of spatial variation in integrated circuit process and devices," *IEEE Trans. Semicond. Manuf.*, vol. 10, no. 1, pp. 24–41, Feb. 1997.
[13] S. R. Nassif, "Modeling and analysis of manufacturing variations," in *IEEE Custom Integrated Circuits Conf.*, San Diego, CA, May 2001, pp. 223–228.
[14] X. Lu, Z. Li, W. Qiu, D. M. H. Walker, and W. Shi, "PARADE: PARAmetric delay evaluation under process variation," in *IEEE Int. Symp. Quality Electronic Design*, San Jose, CA, Mar. 2004, pp. 276–280.
[15] A. R. Alvarez, B. L. Abdi, D. L. Young, H. D. Weed, J. Teplik, and E. R. Herald, "Applications of statistical design and response surface methods to computed aided VLSI device design," in *ACM/IEEE Design Automation Conf.*, Feb. 1988, vol. 7, no. 2, pp. 272–287.
[16] N. Megiddo, "Linear programming in linear time when the dimension is fixed," *J. ACM*, vol. 31, no. 1, pp. 114–127, Jan. 1984.
[17] W. Qiu and D. M. H. Walker, "An efficient algorithm for finding the K longest testable paths through each gate in a combinational circuit," in *IEEE Int. Test Conf.*, Charlotte, NC, Oct. 2003, pp. 592–601.
[18] W. Qiu, J. Wang, D. M. H. Walker, D. Reddy, X. Lu, Z. Li, W. Shi, and H. Balachandran, "K longest paths per gate (KLPG) test generation for scan-based sequential circuits," in *IEEE Int. Test Conf*, Charlotte, NC, Oct. 2004, pp. 223–231.

Platform-Based Design for an Embedded-Fingerprint-Authentication Device

Patrick Schaumont, David Hwang, and Ingrid Verbauwhede

***Abstract*—Fingerprint authentication, in an embedded and portable context, requires complex signal, network, and security-protocol processing in a resource-constrained implementation. We present a platform-based design approach for this application, based on a hierarchy of virtual machines (VM). The fingerprint authentication is programmed in Java, C, and VHSIC hardware description language, and mapped onto a hierarchy of three machines, consisting of an embedded Java VM, an Sparc-V8 core, and an field programmable gate array. We show how our approach is able to cope with multiple concurrent design processes and multiple application domains, including biometrics signal processing, as well as security-protocol implementation. The platform-based design approach also deals with reuse requirements for embedded software and hardware. The formulation of a platform as a VM enables design exploration and incremental design validation throughout the design traject, and results in a specialized, but still programmable, platform. The Java bytecode of our fingerprint authentication takes less than 10 kB.**

***Index Terms*—Embedded systems, hardware–software codesign, system-on-chip.**

Manuscript received May 10, 2004; revised October 18, 2004 and January 20, 2005. This work was supported in part by the National Science Foundation under Grant CCR 0310527, in part by the Semiconductor Research Cooperation under Grant SRC-2003-HJ-1116, and in part by the John and Fannie Hertz Foundation (DH). This paper was recommended by Associate Editor A. Raghunathan.

P. Schaumont was with the Electrical Engineering Department, University of California at Los Angeles, CA 90095-1594 USA. He is now with the Electrical and Computer Engineering Department at Virginia Tech, Blacksburg, VA 24061 USA (e-mail: schaum@vt.edu).

D. Hwang was with the Electrical Engineering Department, University of California at Los Angeles, CA 90095-1594 USA. He is now with KeyEye Communications, Sacramento, CA 95827 USA (e-mail: dhwang@ee.ucla.edu).

I. Verbauwhede is with the Electrical Engineering Department, University of California at Los Angeles, CA 90095-1594 USA, and also with the Electrical Engineering Department, Katholieke Universiteit Leuven, B-3001, Belgium (e-mail: ingrid@ee.ucla.edu).

Digital Object Identifier 10.1109/TCAD.2005.853709

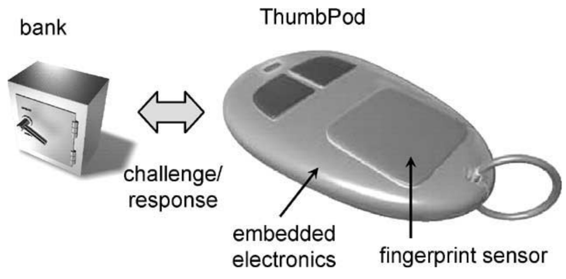

Fig. 1. ThumbPod embedded-fingerprint-authentication device.

I. Introduction

Many novel applications in the embedded context have the combined requirements of high complexity and high energy efficiency. These applications are not easy to implement, because they need a specialized architecture, as well as a software-centric design approach. They can also cover multiple application domains. We present a platform-based design strategy [1], [2] for such devices, based on our recent design experience with an embedded-fingerprint-authentication device. Multiple levels of platforms are obtained by a hierarchy of virtual machines (VMs). Platform specialization is expressed as a native interface design on these VMs.

A. ThumbPod Application

The embedded-fingerprint-authentication device is called ThumbPod [3]. It has the form factor of a keychain, as shown in Fig. 1. ThumbPod is a biometrically driven electronic key that establishes a strong and secure bond between the owner of the key and the key itself. In contrast, a classic key such as a metal car key is anonymous. ThumbPod scans a person's fingerprint, analyzes the spatial features of the fingerprint, compares them to a prestored template, and generates a positive or negative authentication. The prestored template makes the ThumbPod a personal device; the template is captured and securely stored in the device upon service enrollment.

The authentication result during normal service is used in the context of a secure client–server protocol. The server is at a remote location, for example at a bank institution, and relies on the client to implement secure authentication. The secure protocol avoids the transmission of raw biometric data by using a challenge/response mechanism. The server sends a random challenge to the ThumbPod that can only be answered by means of a successful fingerprint authentication. If the authentication fails, the ThumbPod's response is incorrect. Based on this incorrect response, the server can detect the type of error involved (false fingerprint, socket-connection error, false ThumbPod) and take appropriate measures. Thus, by doing fingerprint processing locally, within ThumbPod, we obtain increased protection of sensitive information, at the cost of additional embedded computation.

The ThumbPod application exists at the crossing of two application domains: biometrics signal processing and security-protocol design. The application is implemented at multiple abstraction levels: the security protocol is developed in Java, the fingerprint signal processing is written in C, and the hardware platform is custom designed in VHSIC hardware description language (VHDL). Using a hierarchy of VMs, we will formulate the ThumbPod application as a case of platform-based design.

B. Platform-Based Design With a VM

We start by making a short comparison between platforms and VMs. In the context of platform-based design, Sangiovanni-Vincentelli defines a platform as "an abstraction layer in the design flow that facilitates a number of possible refinements into a subsequent abstraction layer (platform) in the design flow" [2]. A VM then, is defined by a VM instruction set. Applications can be expressed as programs written in this instruction set. According to the definition of a platform, a VM by itself is not sufficient to classify it as a platform, because the fixed instruction set represents a fixed, nonmodifiable abstraction layer in the design flow. This changes when we also include the use of native interfaces in the VM. A native interface [4] allows access to features that are located outside of the VM. When the performance or the features of a VM instruction set are inadequate for the application, some form of specialization can be provided. This establishes the concept of platform-based design using VMs.

There are two distinct cases when the use of native interfaces is required.

1) The performance delivered by a VM might be inadequate. For example, when the Java implementation of an AES encryption routine is too slow, then a Java native method in C can provide speedup.
2) A VM might be unable to implement a certain feature. For example, ThumbPod contains a biometric fingerprint sensor, a device specific to the application. The top-level system model in Java requires a native method to abstract the interface of the fingerprint sensor.

These two distinct cases correspond to top-down and bottom-up development strategies. Both are equally important in a complete design flow. Top-down strategies are common in software development and system design. Top-down strategies analyze design requirements at a higher abstraction level, and then refine those requirements into implementations at a lower level. Bottom-up strategies are used in hardware development and in the integration of intellectual property (IP). Bottom-up strategies concentrate on the efficient and convenient integration of design components.

C. Paper Overview

Section II gives a system design perspective on the ThumbPod application, starting with the design requirements, and then elaborating on the ThumbPod platform. This platform is implemented as a hierarchy of VMs. This gives particular advantages for platform-based design. Validation and refinement can proceed incrementally, because each VM is an executable by itself. Section III discusses the refinement process of the ThumbPod security protocol and biometrics signal processing. We also provide details on the overhead resulting from native interfaces, which is an important cost factor for this form of platform-based design. Section IV collects the results of our design strategy and Section V presents related work.

II. ThumbPod Embedded Platform

The ThumbPod embedded platform contains several intermediate design layers. A review of the design requirements will motivate the usefulness of a platform-based design strategy. In particular, a hierarchy of VMs enables incremental design refinements and their validation at multiple levels of abstraction.

A. Design Requirements

Five design requirements are enumerated for ThumbPod.

1) ThumbPod is not a stand-alone entity, but rather operates as a client in a client–server security protocol. Security applications need a holistic approach that considers all the aspects of an application at the same time. It is easy to see that authentication is of little use, if it is not done in front of a third party that can trust the authentication result. This third- party uses a security protocol to verify the credibility of the authentication process, as well as the authentication result. For example, one needs to verify if the ThumbPod device has not been tampered with. So even if the ThumbPod will be a system for design purposes, it is only a component for validation purposes.
2) ThumbPod needs to accommodate very heterogeneous IP. The fingerprint sensor is a third-party module [5]. The fingerprint-matching algorithms are not developed from scratch, but based on existing reference software for workstation applications [6]. Both the sensor and the matching algorithms need to be integrated in an embedded context. Thus, the ThumbPod architecture needs to have the flexibility to integrate architectural and functional IP at very different levels of abstraction.
3) ThumbPod needs to operate in a real-time energy-efficient embedded context. Real time means executing a complete iteration through the security protocol, including the fingerprint authentication, in less then 5 s. This roughly corresponds to the user-end time for a credit-card transaction.
4) ThumbPod has a hard design schedule and a fixed design team. The design is started 8 mo before the prototype deliverable is due. Seven students and a professor join forces to come up with a single result. Obviously, the only way this size of a team can create a result within 8 mo is by consistently applying a codesign-everything philosophy: codesign the security protocol with embedded-fingerprint algorithms, and with the target architecture and custom hardware.
5) Security weaknesses are in the interfaces. Many security attacks focus on these interfaces to trigger exception behavior and other anomalies. The systematic VM with well-defined native interfaces facilitates a secure design approach.

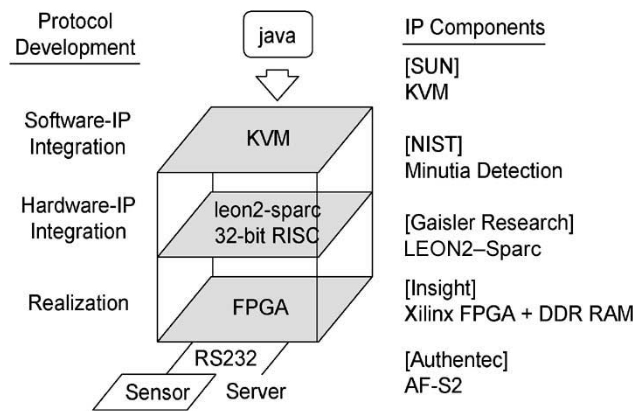

Fig. 2. VM hierarchy of ThumbPod.

B. ThumbPod Platform

Fig. 2 illustrates the machine stack of ThumbPod. The bottom layer is an FPGA development board that integrates a fingerprint sensor and a server connection using a serial-port connection. The middle layer is a LEON2–Sparc core (http://www.gaisler.com), programmable in C. It integrates memory-mapped coprocessor units. The top layer is a Java VM (KVM, [7]). It integrates complex software IP, and at the same time, it provides an easy top-level programming model for the development of the ThumbPod security protocol. ThumbPod, thus, was created using three different layers of design abstraction. The input specification for this platform consists of Java, C, and VHDL code. The three platforms are a hierarchy of VMs: the KVM runs on top of LEON2, which runs on top an FPGA. Specialized IP is integrated using native interfaces.

TABLE I
THUMBPOD VM

Virtual Machine	Execution Model	Programming Language	Native Interface	Standard Library	ThumbPod IP
JVM or KVM	Sequential	J2SE or J2ME JAVA	JNI or KNI	J2SE or CLDC	
LEON2	Sequential	C + ASM	AMBA Peripheral Bus	stdlib	Minutiae detection
FPGA	Parallel	GEZEL + VHDL	IO-pin	Virtex2	AES HW DFT HW Sensor

Configuration and version data: Java Virtual Machine KVM CLDC 1.0.4, JVM 1.0.3-13; Embedded processor LEON2-1.0.9 with 2 kB ICache, 2 kB DCache, No FPU, 2000 on-chip boot ROM, Dual UART; C Compiler GCC 2.95.2 C-Compiler with RTEMS; FPGA Xilinx Virtex2-1000; Fingerprint Sensor Authentec FingerLoc AF-S2 [5].

Each of these layers has proper programming environments and execution semantics, as illustrated in Table I. The Java VM in ThumbPod comes in two embodiments. For initial development, we use the Java-2 Standard Edition (J2SE) environment. For embedded implementation on LEON2, we switch to the Java-2 Micro Edition (J2ME) environment, which uses the KVM. We also use a cycle-true cosimulation environment called GEZEL [8] to verify our coprocessor designs for the LEON2.

III. THUMBPOD DESIGN REFINEMENT

The hierarchical VM concept of ThumbPod integrates two different application domains: biometrics signal processing and security-protocol design. Fig. 3 illustrates the decomposition of ThumbPod over different VM levels and over the two subdomains.

The security domain implements the security protocol. Several steps in this challenge/response protocol use advanced encryption standard (AES) encryption, supported with an AES coprocessor. The use of hardware encryption is done not only for performance, but also for reasons of security. A hardware coprocessor can be implemented using secure-logic circuit styles, preventing eavesdropping and side-channel attacks [9]. The security protocol also needs a communication channel to the server, implemented with a serial connection. The use of a serial connection implies that a structured representation of secure-protocol data must be developed.

The biometrics domain implements the entire fingerprint-processing operation, from capturing the fingerprint up to matching the template. The fingerprint sensor delivers a 128×128 grayscale image of pixels representing a scanned fingerprint. Fingerprint-minutiae detection routines convert the image into a template. Fingerprint-matching routines evaluate the likeliness of two templates. A template is a structured representation of the fingerprint minutiae found in an image. Each minutia is represented by a type and a pair of (x, y) coordinates. The detection routines are computation intensive and require hardware acceleration.

A. Security-Domain Refinements

Due to the multidomain nature of ThumbPod, there is no single unique design flow leading from specification to implementation. In fact, it is easier to think of the design process as a number of concurrent "integration flows" that combine different elements of IP into the final ThumbPod. This approach allows multiple design activities to proceed in parallel.

The security protocol of ThumbPod is programmed in Java. The secure challenge/response protocol is developed as a client/server application, using socket communications. A cosimulation between the client and server can be executed at several levels of abstraction, as shown in Fig. 4. An initial model is created in the comfortable environment of a workstation (step 1). Next, an embedded Java VM (KVM) is used to create an embedded version of the ThumbPod client (step 2). The KVM can be compiled for the workstation, as well as for the instruction-set simulator of the embedded core. But there is an important difference between the two versions. Socket communications are not supported on the embedded core, due to the lack of a full operating system. For this reason there is a step 3, called "proxy integration." The client is separated into two parts. The ThumbPod client runs on top of an instruction-set simulator and uses teletypewriter (TTY) communications [standard input–output (I/O) in C]. A second program, called "proxy agent," connects to the instruction-set simulator and translates TTY communications to sockets. A fourth step moves the KVM onto the actual LEON2 implementation on FPGA, while keeping the proxy on the personal computer (PC) that connects to the FPGA board. TTY communications are then implemented using a serial RS232.

The security protocol also uses an AES coprocessor. It has a refinement process, as shown in Fig. 5. The AES coprocessor is originally available as an IP datapath core in VHDL. To integrate the coprocessor into KVM, the AES coprocessor is attached to LEON2 through a memory-mapped interface. Next, a C driver function for this coprocessor is developed and wrapped into a native method of a Java class.

When the AES is implemented at the lower levels of the VM hierarchy, a performance improvement can be expected due to the increased execution efficiency of those lower levels. Crossing multiple levels of VMs, however, introduces overhead. Table II quantifies this overhead for the case of AES encryption. The table lists cycle counts for encryption called from Java and executed in Java, in C, and in hardware. The cosimulations are done using the LEON2–Sparc instruction-set simulator [10] and the GEZEL hardware-design environment [8]. While hardware acceleration of AES offers two orders of magnitude cycle-count improvement, it also introduces up to two orders of magnitude overhead. For example, according to the column "AES in GEZEL," AES encryption executes in 11 clock cycles as a hardware model. Calling the hardware coprocessor out of C requires 797 cycles. The encapsulation of hardware AES in C thus introduces 786 cycles of overhead per invocation of the AES encryption. Likewise, the Java encapsulation takes 1780 cycles, resulting in 1769 cycles of overhead. This native interface overhead is devoted solely to moving data across the VM hierarchy.

B. Biometrics Domain Refinements

Another integration flow is that of the biometrics. This mixed top-down, bottom-up flow is illustrated in Fig. 6. The fingerprint algorithms are created starting from a reference implementation from the National Institute of Standards and Technology (NIST) [6], and mapped into the target architecture using a top-down design flow. The fingerprint sensor, on the other hand, is a custom off-the-shelf device from Authentec, Inc. It is integrated into the target architecture with a bottom-up approach. The point of convergence between the two is the embedded Java program that combines native methods into a single application.

The fingerprint-analysis algorithms available from NIST are minutiae-detection algorithms. They support analysis of a fingerprint's composition and return the locations of a fingerprint's minutiae. They are developed in floating point and their operation is memory intensive. The initial design steps to refine these algorithms into the ThumbPod platform are therefore, fixed-point refinement and memory

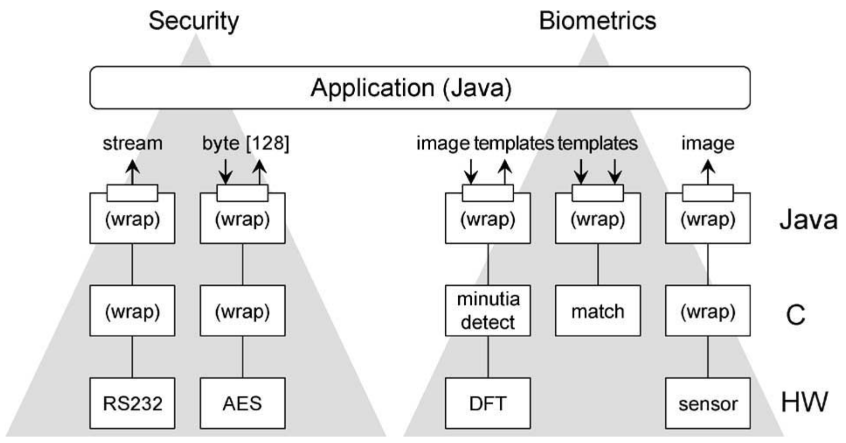

Fig. 3. ThumbPod application spans two domains.

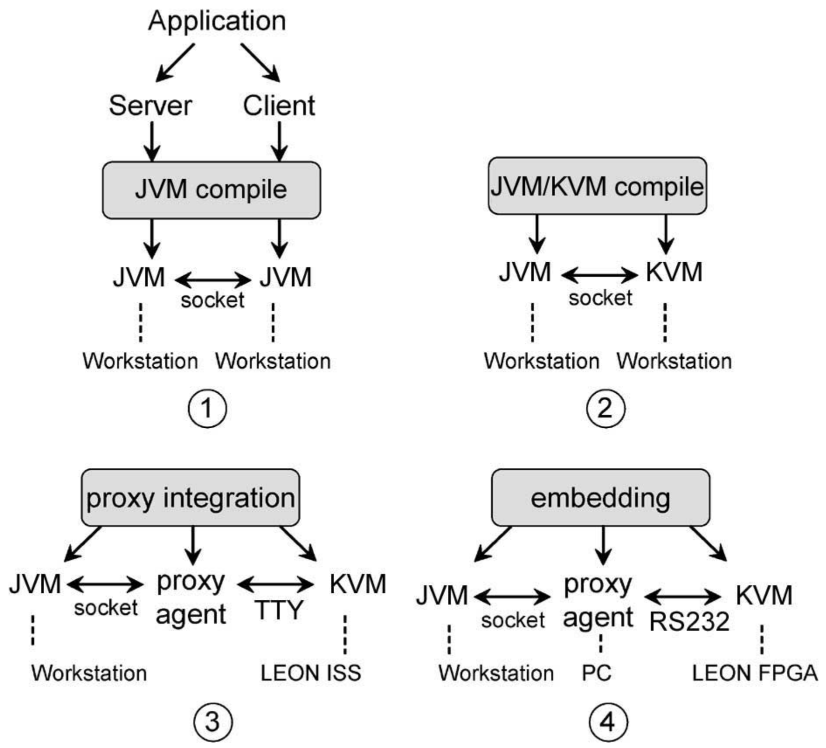

Fig. 4. Design flow for ThumbPod security protocol.

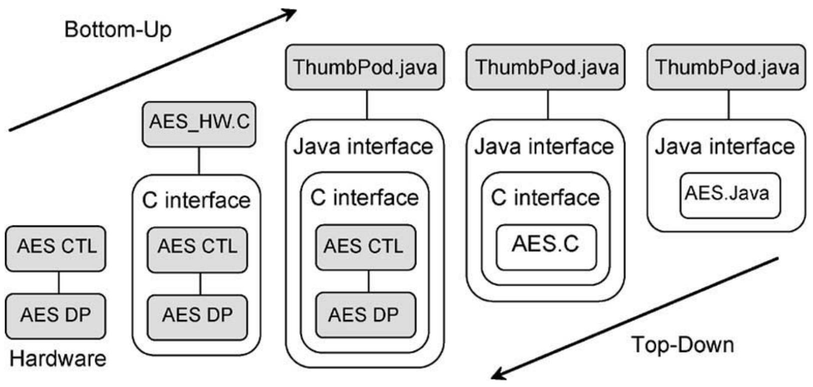

Fig. 5. AES coprocessor hardware–software integration.

TABLE II
AES Encryption in Java Using Java, C, or Coprocessor Hardware

Host	AES in JAVA[1]	AES in C[2]	AES in GEZEL
JAVA	194 100	10 111	1780 cycles
C		9234	797 cycles
GEZEL			11 cycles
Speedup[3]	1	19.2	109
Overhead[4]		1.1	161.9

[1] KVM CLDC 1.0.4 with JVM 1.0.3-13
[2] LEON2–SPARC processor with gcc 2.95.2 O2-level optimization
[3] Compared to AES implementation in Java
[4] (Cycles for Java Native Function) / (Cycles for AES Functionality)

optimization. We also developed a minutiae-matching algorithm to complement the minutiae-detection algorithm. Fixed-point refinement is done by rewriting the code to use only C integers. Memory optimizations are performed using A Toolbox for Optimizing Memory I/O Using geometrical Models (ATOMIUM) [11]. A stand-alone database of fingerprints is used, and the matching algorithms are evaluated on a workstation using file I/O. When an acceptable performance is reached, the resulting algorithms are further optimized for performance in a cosimulation environment consisting of the GEZEL and LEON2 instruction-set simulator. Profiling of the fingerprint-detection routines reveals that a major amount of computations originate from discrete Fourier-transform (DFT) calculations in the fingerprint-image preprocessing. Therefore, a memory-mapped DFT coprocessor is custom designed in GEZEL. This coprocessor is integrated into the system using the same instruction and data buses as the AES coprocessor, with a separate instruction set controlling each coprocessor. The coprocessor is called from within the fingerprint-minutiae detection routines. Yang *et al.* has shown that our fingerprint matching can be done on a 50 MHz LEON2 with a running time of 4 s, 0.5% false-reject rate, and 0.01% false-accept rate [12].

C. System Design-Flow Organization

Fig. 7 illustrates the mapping of the ThumbPod design flow to teams. The project spans two phases: refinement and integration. In the first phase of the project, each team works on one particular machine level, in order to become familiar with the platform and to create an appropriate design infrastructure. In the second phase of the project, the different platforms are integrated together using native interface mechanisms. The teams created at the start of the project are listed on the left of Fig. 7.

1) System Protocol: A single-person team develops a challenge–response protocol in Java on workstation (J2SE), using socket programming. This version is migrated onto an embedded Java VM as the project proceeds.

2) Minutiae Matching: A two-person team works exclusively on minutiae-detection and -matching algorithms, as discussed in Section III-B. This is, by far, the piece of IP that needs the most refinement work.

3) Software Platform: A two-person team ports a reference implementation of the embedded Java VM, KVM, onto the LEON2 target processor for this project. These persons also prepare a development environment that supports Java and embedded-C compilations.

4) Hardware Platform: A two-person team implements the LEON2 onto an FPGA board, creating additional interfaces to off-chip random-access memory (RAM) and making sure that the embedded processor boots correctly.

Once individual machine levels operate reliably, the second phase of the project is initiated to integrate all components together. Projectwise, this means that the "platform" teams focus on building interfaces between each other and between other project teams, and that gradually, more complicated machines are built. This second phase is far more difficult than the first phase, because it requires extensive and detailed communication between the different teams and design-abstraction levels. Detailed project technical data that illustrate this point may be consulted in [13].

IV. Results

Fig. 8 illustrates the design complexity of ThumbPod. This chart uses noncommented lines of source code (NLOC). Because these numbers represent files that have been—at some point during the ThumbPod project—manipulated individually at the source-code level, they can serve as a relative measure of the design complexity. This is the design complexity faced by the design team, rather than by the individual designers.

About 31% of the code is application specific, and contains Java, native functions in C, and coprocessor descriptions in GEZEL. The fingerprint-minutiae detection and -matching C code, obtained out of the refinement of the NIST code (Section III-B), is included within the native functions in C. The rest of the code is platform-definition code, and includes the LEON2 VHDL description and the KVM C description. The only runtime-defined part is the top-level Java applet, and it contains only a fraction of the complete design description. Thus, derived and related applications on the ThumbPod platform require only minimal modification and design time.

One of the strong points of using a hierarchy of VMs in a platform-based design method is that a verification and validation framework is available during the entire design flow. This framework is incrementally constructed towards a virtual prototype of the actual embedded system. Table III illustrates several intermediate simulation platforms. A distinction is made between intramachine platforms, which contain a single VM, and intermachine platforms, which contain a hierarchy of VMs. These intermachine evaluation platforms are needed for systematic platform-based design. In a heterogeneous embedded system, bugs can occur at multiple levels of software and hardware, and most often at their interfaces. An intermachine platform allows the designer to deal with this in an incremental way, thus eliminating problems at one machine (or between two machines) before moving up or down to the next VM layer.

Fig. 9 illustrates the prototype architecture of the final embedded implementation. The KVM executes on top of an LEON2 Sparc processor, which in turn is configured as a soft core in a Virtex XC2V1000 FPGA, which also contains the two acceleration coprocessors: the AES and the DFT. The prototyping environment is an Insight Electronics development board, which contains, besides the FPGA, also a 32-MB DDR RAM. The interface to the server is provided by RS232 communication, and fingerprints are captured by means of an Authentec AF-S2 complimentary metal–oxide–semiconductor (CMOS) sensor. We have demonstrated the system to be operational under the requirements enumerated in Section II-A.

Table IV illustrates some of the design statistics of the final implementation. It is worth noting that the design size of higher abstraction levels (the Java Applets) is considerably smaller than the design size at lower abstraction levels. The upper programming layer indeed reflects the native functions that are needed to exactly match the application, thus allowing one to write smaller programs that execute more efficiently.

V. Related Work

In the introduction, we already mentioned platform-based design [1], [2] as a framework to define system-level design, in the context of

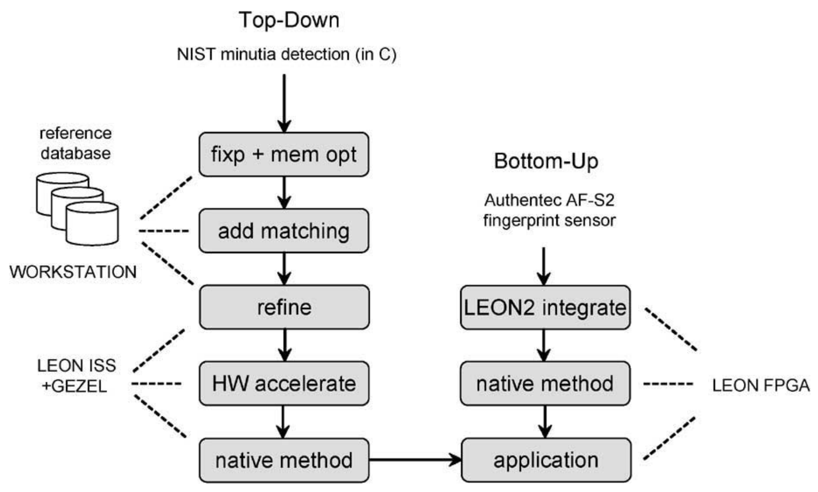

Fig. 6. Design flow for ThumbPod biometrics.

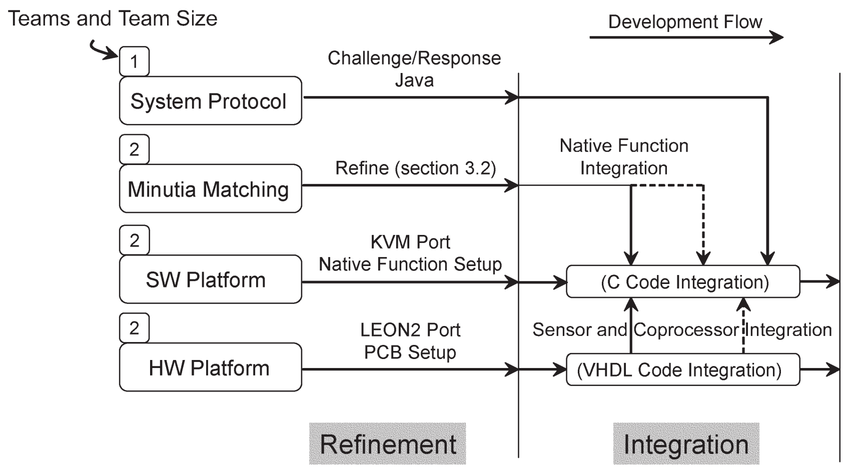

Fig. 7. Teams and project-development flow.

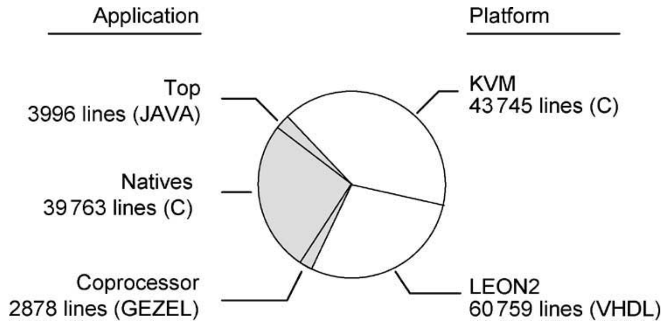

Fig. 8. Code complexity in ThumbPod.

TABLE III
INTERMACHINE AND INTRAMACHINE PLATFORMS

	Language	Platform	Evaluation Parameter	Cost Factor
Intra-machine Platforms	JAVA	JVM	Performance	JAVA Ticks
		KVM	Performance	JAVA Ticks
	C	GCC	Performance	Seconds
		ATOMIUM	Memory	Bytes
		TSIM	Performance	Cycle count
	VHDL	Modelsim	Performance	Critical Path
		Synplicity	Area	
				CLB
		LEON2-FPGA	Performance	Seconds
Inter-machine Platforms	JAVA + C	KVM on TSIM	Performance	Cycle count
	C + GEZEL	TSIM + GEZEL	Performance	Cycle count
	JAVA + C + GEZEL	KVM on TSIM + GEZEL	Performance	Cycle count
	C + VHDL	LEON2-FPGA	Performance	Seconds
	JAVA + C + VHDL	KVM on LEON2-FPGA	Performance	Seconds

integrated and embedded systems. In this paper, we have applied the VMs as an embodiment of a platform. Platform variants are supported by customizing the VM with native interfaces.

In recent years, VMs have permeated many aspects of computing and design. For the application domain of wireless handheld applications, the KVM [7] executes Java bytecode programs in a memory footprint of 160 to 512 kB. For even more constrained sensor node networks, the Mate VM [14] executes in only 16 kB of read-only memory and 1 kB of RAM. These VMs are used for a different reason than platform virtualization. They are used because they offer a compact and efficient representation of the application. Indeed, in the sensor networks targeted by Mate, or the mobile-phone networks targeted by KVM, communication bandwidth and energy cost are primary issues. Wireless reprogramming of an application should be done with a minimum amount of code. For this purpose, the KVM and Mate are instrumented with native interfaces and tailored towards the target application and platform. In our method, we propose to

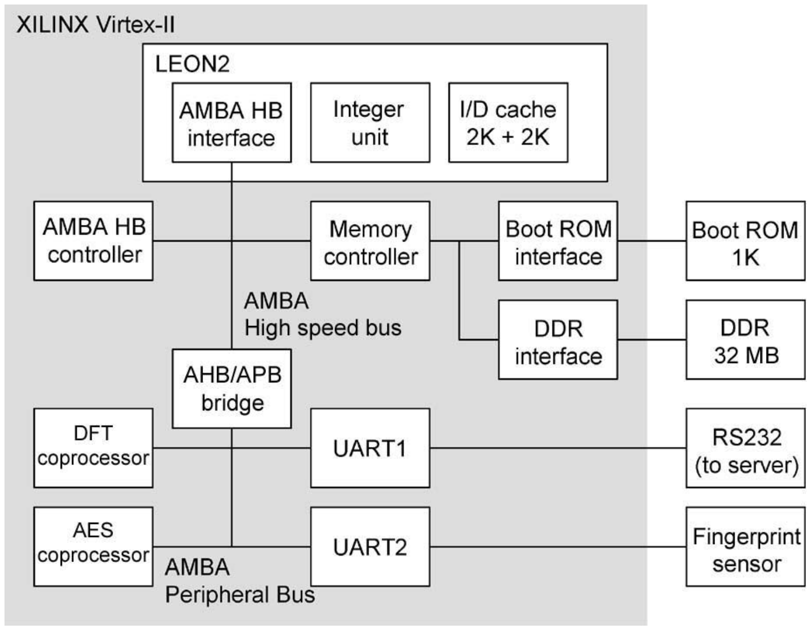

Fig. 9. ThumbPod prototype architecture.

TABLE IV
OVERALL DESIGN STATISTICS

Java Application	Java Bytecode	Cost
ThumbPod Match Applet	9536	bytes
ThumbPod Server Applet	8768	bytes
KVM Implementation	Static Code + Data Segment	
KVM + J2ME Classlib on LEON2	295 348	bytes
C-Library Support	54 273	bytes
Native Functions	171 339	bytes
LEON2 Implementation		
LEON2 Core	4856	LUT
	13	BlockRAM
AES Coprocessor	3474	LUT
	16	BlockRAM
DFT Coprocessor	2844	LUT
DDR RAM Interface	169	LUT
KVM Implementation		
System Clock	50	MHz

LEON-2 implementation parameters are No FPU, 8 Reg Windows, 2K DCache, 2K ICache, Integer Multiplier, No Divider, 512-word Bootprom, Virtex-II target technology.

introduce these native interfaces systematically, as part of the design refinement. Our considerations are similar, although not exactly the same as with KVM and Mate. We are interested in using native interfaces to reduce the variability in the application model. Less variability implies a more specialized platform, which in turn offers better energy efficiency.

A VM model can also be used for exploration purposes, as demonstrated in the Cadence virtual component co-design (VCC) [15] environment, as well as in the modeling environment for software and hardware (MESH) design methodology [16]. In these models, behavior is expressed independently from the resources required to execute that behavior. Resource consumption is expressed in terms of virtual execution time, memory usage, and so on. With this model, system exploration can be formalized as finding an optimal schedule of functionality onto resources.

To summarize, we use the VM as a means to capture the design activities at a particular abstraction level. This model offers the same benefits as platform-based design, as it uncouples the design of applications from those of the architecture. Using native interfaces, we also have a systematic way of expressing platform specialization.

VI. CONCLUSION

We have presented a platform-based design approach using a hierarchy of VMs. For the design of a fingerprint-authentication system, we illustrated that this is a viable way to cope with multiple application domains, heterogeneous platforms, and integration of heterogeneous intellectual property (IP). The concept can deal with software-design requirements that cannot be met by top-down compiler technology. Indeed, by using native interfaces, a VM can cope with bottom-up and top-down design styles at the same time. This enables a tight integration of the hardware- and software-development process. In fact, throughout the development of the fingerprint-authentication system, we applied a rigorous codesign process that lined up hardware-, software-, and application-development teams around VM-based cosimulations.

REFERENCES

[1] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1523–1543, Dec. 2000.
[2] A. Sangiovanni-Vincentelli. (2002, Feb. 5). Defining platform-based design. *EETimes* [Online]. Available: http://www.eedesign.com/showArticle.jhtml?articleID=16504380
[3] D. Hwang, P. Schaumont, Y. Fan, A. Hodjat, B. C. Lai, K. Sakiyama, S. Yang, and I. Verbauwhede, "Design flow for HW/SW acceleration transparency in the ThumbPod secure embedded system," in *Proc. 40th IEEE/ACM Design Automation Conf. (DAC)*, Anaheim, CA, 2003, pp. 60–65.
[4] T. Lindholm and F. Yellin, *The Java Virtual Machine Specification*. Reading, MA: Addison-Wesley, 1996.
[5] Authentec Inc. Product Specification for the AF-S2 Fingerprint Sensor. [Online]. Available: http://www.authentec.com

[6] M. Garris, C. Watson, R. McCabe, and C. Wilson, "User's Guide to NIST Fingerprint Image Software (NFIS)," Nat. Inst. Standards Technol., Gaithersburg, MD, Interagency Rep. 6813, 2001.

[7] Sun Microsystems. CLDC and the K Virtual Machine. [Online]. Available: http://Java.sun.com/products/cldc

[8] The GEZEL Design Environment, UCLA. [Online]. Available: http://www.ee.ucla.edu/~schaum/gezel

[9] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Proc. Design Automation and Test Europe (DATE)*, Paris, France, 2004, pp. 246–251.

[10] J. Gaisler, *TSIM Simulator User's Manual*, Gaisler Research, Göteborg, Sweden, 2005.

[11] ATOMIUM Tool Suite, IMEC, Belgium. [Online]. Available: http://www.imec.be/design/atomium/

[12] S. Yang, K. Sakiyama, and I. Verbauwhede, "A compact and efficient fingerprint verification system for secure embedded devices," in *Proc. 37th IEEE Asilomar Conf. Signals, Systems, and Computers*, Pacific Grove, CA, 2003, pp. 2058–2062.

[13] P. Schaumont, K. Sakiyama, Y. Fan, D. Hwang, B. Lai, A. Hodjat, S. Yang, and I. Verbauwhede, "Testing ThumbPod: Softcore bugs are hard to find," in *Proc. IEEE Int. High Level Design Validation and Test Workshop (HLDVT)*, San Francisco, CA, Nov. 2003, pp. 77–82.

[14] P. Levis and D. Culler, "Mate: A tiny virtual machine for sensor networks," in *Proc. Int. Conf. Architectural Support Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA, 2002, pp. 85–95.

[15] C. Passerone, L. Lavagno, M. Chiodo, and A. Sangiovanni-Vincentelli, "Fast hardware/software co-simulation for virtual prototyping and trade-off analysis," in *Proc. 34th Design Automation Conf.*, Los Angeles, CA, 2000, pp. 389–394.

[16] J. Paul and D. Thomas, "Codesign virtual machine approach to modeling computer systems," in *Proc. Design Automation and Test Europe Conf.*, Paris, France, 2002, pp. 522–528.

An Efficient Coefficient-Partitioning Algorithm for Realizing Low-Complexity Digital Filters

A. P. Vinod and Edmund M.-K. Lai

Abstract—**Algorithms that minimize the complexity of multiplication in digital filters focus on reducing the number of adders needed to implement the coefficient multipliers. Previous works have not analyzed the complexity of each adder, which is significant in low-complexity implementation. A multiplication algorithm for low-complexity implementation of digital filters with a minimum number of full adders (NFAs) and improved speed is proposed here. The authors exploit the fact that when multiplication is implemented using shifts and adds, the adder width can be minimized by limiting the shifts of the operands to shorter lengths. The coefficient-partitioning (CP) algorithm proposed here minimizes the shifts of the operands of the adders by partitioning each coefficient into two subcomponents. The authors show that by combining three methods, the CP algorithm, an efficient coefficient coding scheme known as pseudo floating-point (PFP) representation, and the well-known common subexpression elimination (CSE), the NFAs required in each adder of the multiplier can be reduced considerably. Design examples show that the method offers an average FA reduction of 30% for finite-impulse response (FIR) filters and 20% for infinite-impulse response (IIR) filters over CSE methods.**

Index Terms—**Adder complexity, coefficient partitioning, common subexpression elimination, finite-impulse response (FIR) filters, infinite-impulse response (IIR) filters, pseudofloating-point representation.**

Manuscript received March 1, 2004; revised August 5, 2004. This paper was recommended by Associate Editor A. Raghunathan.

The authors are with the School of Computer Engineering, Nanyang Technological University, Singapore 639798 (e-mail: asvinod@ntu.edu.sg; asmklai@ntu.edu.sg).

Digital Object Identifier 10.1109/TCAD.2005.852659

I. Introduction

Digital filtering finds extensive application in mobile communication systems to perform various functions such as channelization, channel equalization, matched filtering, and pulse shaping. Low-complexity and high-speed digital filtering for mobile computing and communication applications generally require dedicated hardwired implementations of the filters. Although programmable filters based on digital signal processor cores offer the advantage of flexibility and high sampling rates, they are not suitable for mobile applications that demand high throughput and low-power consumption. Since the flexibility of a multiplier is not necessary in such applications, application-specific digital filters are frequently adopted to meet the constraints of performance and power consumption. However, these filters employ a large number of multipliers that lead to excessive area and power consumption. Therefore, the problem of designing digital filters with small area and low-power consumption has received a great deal of attention in the last decade.

A. Related Work

The number of additions (subtractions) used to implement the coefficient multiplications determines the complexity of digital filters. Many approaches have been proposed in the literature for reducing the number of adders (subtractors) in the multipliers of digital filters. These approaches include coefficient coding using efficient arithmetic schemes [1], [2], coefficient optimization techniques [3]–[5], distributed arithmetic techniques [6], [7], read-only memory (ROM)-based designs [8], [9], and common subexpression elimination (CSE) techniques [10]–[18]. Among these approaches, the CSE techniques in [10]–[18] produced the best hardware reduction since it deals with multiple common multiplications (MCMs), i.e., multiplication of one variable (input signal) with multiple constants (coefficients). The CSE techniques focus on eliminating redundant computations in multiplier blocks by employing the most common subexpressions (CSs) consisting of two nonzero bits. In general, techniques [1]–[18] discuss the complexity of multipliers in terms of the reduction of the number of adders and the critical path. The methods in [1]–[18] have not addressed the issue of minimizing the complexity of each adder of the multiplier, which is significant in low-complexity and high-speed implementations. The differential coefficient method (DCM) [19] uses differential coefficients to multiply with inputs and compensates the effect of differential coefficients by adding the stored partial product of the previous computation. Since differential coefficients have shorter word lengths, the resulting design and output can also use shorter word lengths and thus reduce power consumption. However, this method results in a lot of overheads, which is proportional to the product of the order of difference and filter tap number. The main idea in [20] is reordering computations and identifying common computations that maximize computation sharing between different multipliers. However, the method in [20] offers only a slight improvement in reduction of adders (11%) over the CSE method [11]. Moreover, this method results in an increase in delay, corresponding to the delay of one adder step on average.

In a recent work [21], the authors have analyzed the complexity of implementation in terms of full adders (FAs) required for each multiplier of the filter. Two techniques for optimizing the CSE methods to implement low-complexity finite-impulse response (FIR) filters have been proposed in [21]. These techniques are based on the extension of conventional two nonzero bit (2 bit) CSs in [10]–[18] to form three nonzero bit and four nonzero bit super subexpressions (SSs; called 3-bit and 4-bit SSs, respectively) by exploiting identical shifts