Method for Embedded Application Prototyping based on SoC Platform and Architecture Model

Yassine Aoudni, Guy Gogniat, Kais Loukil, Jean Luc Philippe, Mohamed Abid

Method for Embedded Application Prototyping based on SoC Platform and Architecture Model

Y. Aoudni[1, 2], G. Gogniat[1], Kais Loukil[2], J.L. Philippe[1], M. Abid[2]
[1]LESTER, Université de Bretagne Sud CNRS FRE 2734, Lorient, France,
[2]CES, ENIS engineering school, Sfax, Tunisia

Abstract

Real time embedded system design needs a contribution between architectures model, platforms and application specification in order to prototype a real time system from high level specification. In many case, one allocated solution prototype is not so good to answer the environment changes around an embedded system. For this reason, an application implementation needs to have several prototypes with different performance levels in order to address the environment evolution. This paper gives an approach for rapid embedded system prototyping using a generic high level architectural model and existing prototyping platforms. A several architecture prototype are proposed to rapid converge to a limed architecture space solutions, thus the exploration process is accelerated and an efficient solution can be selected.

1. Introduction

A common method for providing performance improvement for a real time embedded system is to create customized hardware solutions for particular tasks. For example, real time embedded system often have one or more application specific integrated circuits (ASICs) to perform computationally demanding tasks. ASICs are very effective at improving performance, typically yielding several orders of magnitude speedup along with reduced energy consumption. Unfortunately, there are also negative aspects to using ASICs. The primary problem is that ASICs only provide a dedicated hardwired architecture solution, meaning that only a limited number of applications will be able to fully explore the ASIC architecture. If an application changes, because of a fixed bug or a change in standards, the system will usually no longer be able to take advantage of the ASIC device architecture. So, the notion of reconfigurable system is introduced in real time embedded system concept as a need to solve bugs and to support the evolution of standards. Another drawback is that even when a system can utilize an ASIC, it must be specifically rewritten to do so. Rewriting system applications or few tasks of applications can be a large engineering burden. For this reason, the use of reusable components libraries is encouraged to accelerate the design process and to conserve the compatibility for the evolution in real time embedded system. In this case the modularity is proposed to evaluate the ability of reusable components libraries to develop a custom real time embedded system. Many other notions like scalability and platform adequacy are also introduced to replay the needs in new real time system design and to solve limitations of ASIC solution.

Adding custom hardware in processor core is another method for providing enhanced performance in real time embedded system. In general, the critical portions of an application's dataflow graph (DFG) can be accelerated by mapping them to specialized hardware. Usually, there are two granularity levels to add dedicated hardware to processor core system: instruction granularity level and function granularity level. The instruction granularity consists to link custom hardware with the main registers of processor core and a custom instruction opcode is added to the processor instruction set. The number of custom instruction depends on the processor core capacity for example ARM core provides 16 custom instruction extensions. The function granularity consists to add the custom hardware as a slave or a master peripheral using bus communication. In this case one instruction extension can not drive the functionality between the processor and the customized peripheral. So in many cases, specific subroutines should be coded to control the custom hardware activity and the communication with the processor core. The number of added hardware functions depends on the bus band pass and the device size in the case of FPGA circuits. In the case of instruction granularity the processor is in hold mode and it is blocked in custom instruction execution, but in function granularity the mutual execution of processor core and custom peripheral is possible.

In this paper we present a generic architecture model for real time embedded system design named PACM: Processor – Accelerator – Coprocessor – Memory. The PACM model is proposed as a solution to reply the real time embedded system design requirements. Firstly, we evoke the real time embedded system environment design for the PACM model in order to introduce several specifications requirements like reconfigurability, modularity and scalability within design process. Secondly, we proposed to combine the instruction and function granularity in the PACM model to enhance the system performance.

The paper is organised as follow. In section 1 we discuss the related work in real time embedded system design. Section 2 talks about PACM model for SoC prototyping. Section 3 presents a comparison between platforms based on PACM constraint model. In section 4, we propose the mapping process of an application under PACM model. Then, we detailed the mapping process via

case study example and experimentation. Finally, we closed with conclusion.

2. Related works

Reconfigurable architectures have been an active research issue. In [29], an adaptive reconfigurable DSP computing engine is proposed for numerically intensive audio/video communications. The approach may enjoy the flexibility of programmable processors [1], while achieve similar performance to ASIC design. More recently, a good survey [6] of media approaches observed varying processing requirements in multimedia computing and also pointed out the need for exploiting reconfigurable system for media processing.

Reconfigurable computing systems [11] that combine programmable processors and FPGAs with a reconfigurable architecture have been extensively exploited for diverse embedded system applications. Some architecture connects a reconfigurable coprocessor to a general purpose microprocessor [3], [14], [15], [22], [23], [28]. The advantage of these approaches is that the coprocessor can be reconfigured to improve the performance of particular application. Most of previously proposed reconfigurable architectures use FPGAs for the reconfigurable hardware. However, the rich programmable interconnection comes at the price of reduced operating frequency and logic density. The Garp [15] processor architecture combines an industry-standard MIPS processor with a new reconfigurable computing device that can be used to accelerate certain computations. REMARC [21] (reconfigurable multimedia array coprocessor) is a reconfigurable coprocessor that is tightly coupled to a main RISC processor.

One stream oriented architecture is the RaPiD [5], [8], [9], project that studies domain specific architecture, called reconfigurable pipelined datapaths. This architecture is optimized for highly repetitive, computationally-intensive tasks. Very deep application-specific computation pipelines can be configured in RaPiD that deliver very high performance for wide range of applications. Another stream-oriented architecture is the PipeRench [4], [12], [13], [19] project, which is focused on the concept of "virtualizing hardware" to use an interconnected network of configurable logic and storage elements to complete large amount of computations through high speed of reconfiguration hardware.

The RAW research prototype [25] uses a scalable ISA to attack the emerging wire-delay problem by providing a parallel, software interface to the gate,wire, and pin resources of the chip. Tensilica [27] enables rapid design of highly efficient processor cores by extending the processor hardware and software to fit each system's application requirements based on a lean core implementation. The Eclipse [24] provides an architecture template at subsystem level. It supports the reuse of design effort for providing a set of parameterized rules for subsystem composition.

The last presented works are based on the following idea: starting from initial processor core architecture, the goal is to extended architectural capacities in order to support the application specification and the environment constraints. But in many cases, the initial system architecture adds an over cost in term of development time, modularity, flexibility and performance. For example, the architecture system doesn't provide a possibility to add custom instruction for a specific coprocessor and only a function granularity custom hardware is premised. In other cases, the refinement tools can not accept any addition in system architecture, so for this reason the FPGA is used as a custom hardware out of the system chip.

In our approach, we proposed to start from a generic high level architecture model named PACM, and then a prototyping platform will be adapted to support the system constraints and finally implementation strategies will be analyzed for application and system adequacy.

3. PACM architecture model

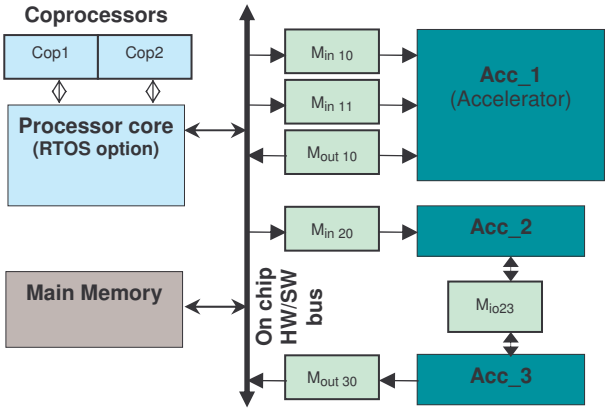

Figure 1: PACM architecture

In this section, for the architecture presented in Figure 1, we show that several CAD tools and platforms, similar in first view, do not present the same adequacy degree with the targeted model of execution. The Figure 1 presents the PACM model. Basically our architecture is built around a processor core (for example Nios, ARM, LEON…) which offers configuration opportunities for adding coprocessors reached through the main processor registers (for example floating point unit, HW divider, HW mathematic functions ...).The processor communicates with dedicated HW accelerators through a standard on chip HW/SW bus (e.g. Amba, Avalon, IBM CoreConnect…) using control logic and specific memory blocks. Coprocessors and HW accelerators usage depends

on the application complexity and on the computing constraint requirements. In order to give more flexibility and adaptability to the SoC, we have chosen the reconfigurable technology to implement our SoC.

4. Prototyping platforms replay for PACM model

If we analyze the key parameters of the PACM architecture model, the adequate platform must integrate the following features:

- The platform must integrate a FPGA device characterized by a heterogeneous architecture (logic elements, DSP blocks, RAM blocks, I/O pin…) and by a size able to integrate the HW and SW parts of the SoC.
- The platform must provide a processor core that gives opportunities to integrate some coprocessors within its ALU and reached through the processor main registers to get an ASIP model.
- The HW accelerators integration must be supported using an on chip HW/SW bus or other on chip HW/SW communication module.
- RTOS option with the corresponding port to the targeted processor core must be present.
- The HW and SW refinement tools must be robust and efficient to limit the time-to-market constraint.

All these key parameters correspond to the criteria to select a suitable SoC platform. We made a qualitative study for different representative platforms, and evaluate their adequacy with the PACM architecture model.

Platforms / Key parameters	LEON	Nios kits	Excalibur kit	PowerPC Microblaze kits
FPGAs architecture	Xilinx family >= Virtex	Altera family >=APEX	APEX family only	Xilinx family>Virtex
Coprocessors integration	+	+++	+++	----
Accelerator integration	+	+++	+++	+++
RTOS	++	+++	+++	+++
SW and HW refinement tools	+++	+++	+++	+++

Table 1 : platforms comparaison

We performed an experimental study based on the main features of SoC platforms. The results are presented in table 1. We notice that all presented SoC platforms provide a robust and efficient HW and SW refinement tools like ISE Xilinx tool and Quartus, on chip HW/SW bus as AMBA (LEON and Excalibur kits) and IBM CoreConnect (PowerPC and Microblaze kit) and also a port for many RTOS like RTEMS ported on LEON and ARM, WindRiver port on PowerPC and Microblaze, etc. However, only Nios, ARM and LEON cores can support coprocessor feature. In addition, coprocessor integration in Nios and ARM cores is more rapid and flexible using the virtualization and custom instruction generation given by SOPC Builder tool. Also, Nios SoC can be implemented in large STRATIX family which contains DSP blocks and different sizes of RAM blocks, unlike ARM development kit which is restricted to APEX device and its core is a hard IP and not a soft one like the Nios core. Thus, we notice that the SoC platform based on Nios processor core kit provided with Quartus and SOPC Builder environments by Altera is the most suitable to design a reconfigurable SoC using the PACM architecture model. Indeed, SOPC Builder tool gives the designer a virtual image of the Nios processor soft core and the accelerators can be linked to the Nios processor core through the Avalon on chip bus. Custom instructions are also provided with this platform in order to facilitate the coprocessors integration within the Nios ALU. Namely, our choice is based on this last feature in order to implement a reconfigurable ASIP core.

As a conclusion of this analysis we can see that the available CAD tools and SoC platforms can not address all the architecture models and that a study must be done in order to select the suitable platform for the appropriate architecture model. In our case the Nios processor core kit is suitable to the PACM architecture model.

5. Steps of mapping application on PACM model

In real time system design reality, we start from two main entry models: one for application and the second for architecture. The needed result is a mapping structure of application under architecture model. In our work we proposed to use the following steps:

a. Modelling application using DFG graph;
b. Identify pattern branches locations in application DFG.
c. Verify the possibility to implement the pattern as a function or/and instruction granularity using PACM model.
d. Proposed a combination between the pattern group in application DFG and the granularity level in PACM model
e. Determinates the features of each proposed solution in term of execution time, power supply and memory code size

The proposed strategy offers the opportunity for an example of application to define concrete architectural solutions of space exploration for a real time embedded system using the PACM model

that combine general processor and custom hardware in two granularity levels.
In the next section we present an application case study in order to explain the mapping process of application on PACM model using the last five steps.

6. Experimentation on 3D graphic application

6.1. DFG description

We choose as case study the 3D graphic pipeline. The main function of the pipeline is to render, a two-dimensional image given a virtual camera, 3D objects, light sources, lighting models, and textures [30]. The typical 3D graphics system can be divided in three stages in pipelined format as in Fig.2. In our case, we are interested to study the Geometric engine.

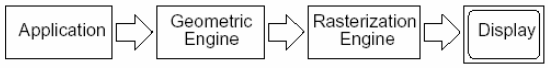

Figure 2 : 3D graphic pipeline

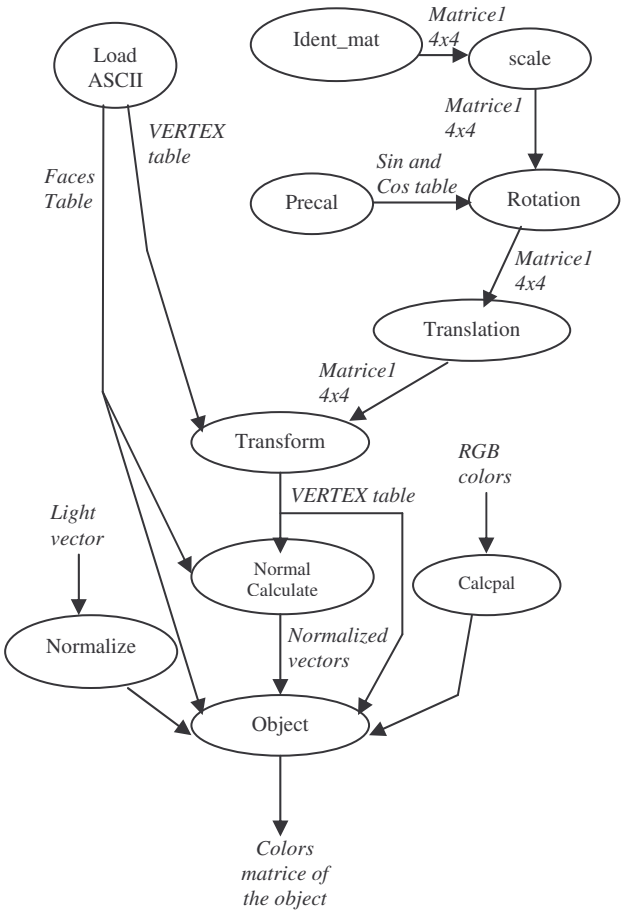

Figure 3 : 3D Data Flow graph

We proposed to implement the 3D application using a C language description. The C tasks are described by the fig3. So 11 tasks are identified in the global application code but there are 4 important tasks: translation, transformation, normal calculate and object draw. These four principal task are identified as patterns and we choose to implement them as a custom hardware. The decision of instruction or function granularity implementation of the pattern depends essentially on the size of the input and data.

6.2. Pattern identification

In our case study application, we analyze the C code of the following functions on 3D application in order to identify the arithmetic patterns. In our analyze process we look for the C function that contain loop structure. We focus on the loop core to determine the arithmetic sequences in line code. Figure 4 gives an example of pattern identification. Indeed, 6 patterns are identified. Table 2 presents the name and the execution frequency of each pattern for different objects. Values presented by the execution frequency show that a custom hardware implementation for the identified patterns is benefit for application speed up. This deduction is true in condition that software implementation is slower than hardware one. On the other hand the impact of power consumption and surface occupation should be verified.

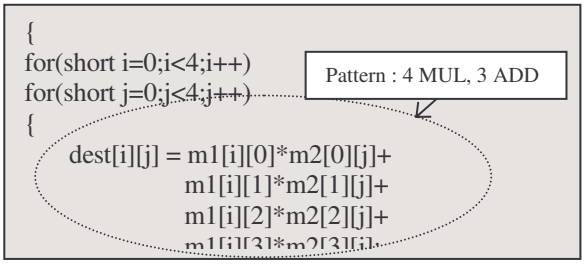

Figure 4 : pattern identification in loop structure

Pattern name	Execution frequency		
	Object1	Object2	Object3
Scalaire	1120	2260	3120
Vectoriel	2380	5280	7280
Mult_matrice	50	50	50
Projection	1210	2260	3660
Transformation	1210	2260	3660
Znormal	2380	5280	7280

Table 2 : Patterns execution frequency

6.3. Pattern hardware implementation

The hardware implementation needs a refinement environment to accelerate the realization process from high level description. In our work we adopt Quartus environment as platform for hardware implementation on Altera technology. The adopted platform design gives the opportunity to specify a custom hardware using a generic IP library. We are interest in arithmetic components present in the IP library in order to implement patterns. The SOPC

Builder provides a NiosII processor core and Avalon on chip for system on chip implementation. The use of SOPC Builder offers the possibility to implement patterns on:

- instruction granularity level using custom instruction generation and two main register in NiosII processor core
- function granularity level using Avalon on bus interface
- the Nios processor core is used to implement the software code of the application

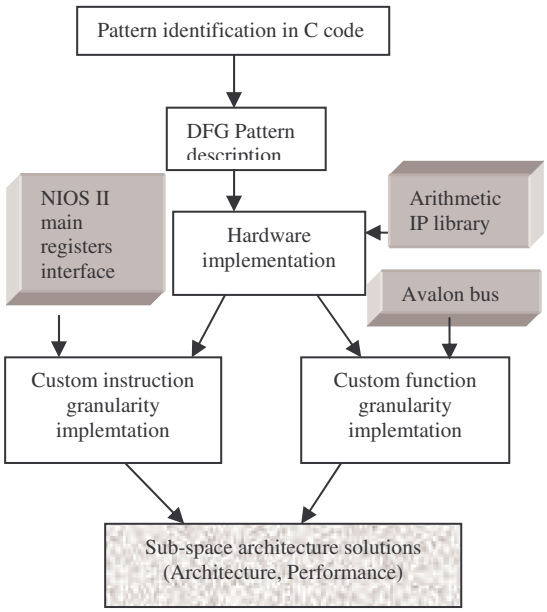

Figure 5 :sub-space architecture solutions prototyping step

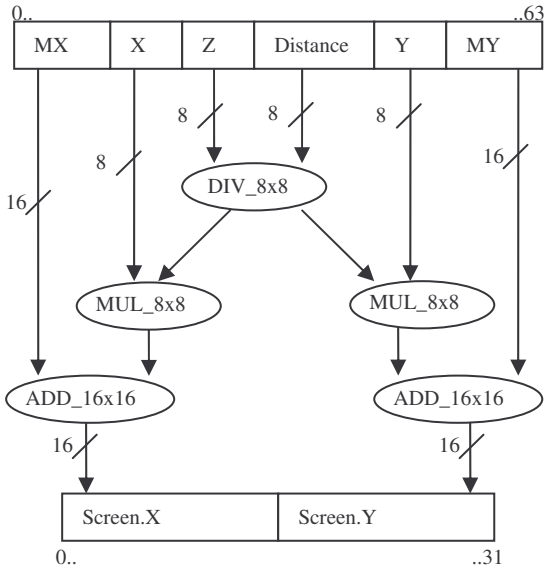

Figure 6 : Data Flow Graph of Projection pattern.

Thus, as presented in Figure 5 starting from the equation description of each pattern, we proposed to specify the pattern using DFG description based on arithmetic operation and data size.

In Figure 6 we proposed an example of PROJECTION pattern with DFG description. Then, we implement the pattern DFG using the generic arithmetic components in Quartus environment. In this step we can compare the hardware and software execution time. Power comsuption and resssource usuage can be deducted from the synthesis step in Quartus environment. The next step consist to decide the possibility to add the pattern to the NiosII based system as a custom instruction using the processor main register (instruction granularity) or as a custom hardware function using Avalon bus communication (function granularity). Finaly, we obtained a several architecture prototypes for one application specification. The architectures solutions represent a sub-group solution for a space exploration based on a initial architecture model (PACM model). Each solution proposes real values of execution time, power consumption, and resource usage. Thus, a real time adaptation method can be injected to guide the solution choice via extern environment constrains.

6.4. Results with architecture solutions prototype

The pattern identification on 3D application gives 6 patterns. The PACM model offers the possibility to implement each pattern in one of 3 cases: software, hardware custom instruction or hardware custom function. So the number of architecture solution can be deduced: 3^6=729 solution. Thus, in this case the design has a large space solution to implement the application. Indeed, in table 3 we are limited to give a sub-group of space solution.

	Solution 1	Solution2	Solution3	Solution4
Pattern1	Software	Custom instruction	Custom instruction	Custom function
Pattern2	Software	Custom instruction	Software	Custom function
Pattern3	Software	Custom instruction	Custom function	Custom function
Pattern4	Software	Custom instruction	Software	Custom function
Pattern5	Software	Custom instruction	Custom function	Custom function
Pattern6	Software	Custom instruction	Custom function	Custom function
Resource usage	2885 LUT	4165 LUT	3420 LUT	4254 LUT
Power consumption	628,58mw	750,34mW	701,34mW	924,36mW
Execution time	78059245 cycle	143316176 cycle	323316176 Cycle	153316176 cycle

Table 3 : performance results for sub-group of space solution

From the four examples presented in table 3, we can note that each combination of patterns implementation gives a triple of resource usage, power consumption and execution time. So, the system can have several functional modes for one application and each mode represents an answer for environment constraints.

7. Conclusion

In this work, we start from the idea: to consider an initial generic architecture model as an entry for a SoC design; we select an adequate platform for rapid prototyping. Then we analyse the application specification in order to locate the most cost patterns in functions execution. Then, we propose to map the application using customs hardware pattern implementation as an instruction granularity level and function granularity level. A combination of patterns implementation can defines a SoC solution prototype for the application based on PACM model. Finally, we proposed a sub-group of solution prototypes that offers the possibility to explore the SoC architecture space solution based on a proposed architecture model.

Reference

[1] Tien-Fu Chen, Chia-Ming Hsu, and Sun-Rise Wu, "Flexible Heterogeneous Multicore Architectures for Versatile Media Processing Via Customized Long Instruction Words", IEEE Transactions on Circuits and Systems For Video vol 15 May 2005.

[2] Victor P. Nelson, Mitchell D. Theys, and Alan Clements, "Computer Architecture and Organization in the model computer Engineering Curriculum", 33rd ASEE/IEEE Frontiers in Education Conference November 5 -8, Boulder 2003,

[3] P. M. Athanas and H. F. Silverman, "Processor reconfiguration through instruction-set metamorphosis," IEEE Computer, vol. 26, no. 3, pp.11–18, Mar. 1994.

[4] S. Cadambi, J. Weener, S. C. Goldstein, H. Schmit, and D. E. Thomas,"Managing pipeline-reconfigurable FPGAs," in Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays, Feb. 1998, pp. 55–64.

[5] D. C. Cronquist, P. Franklin, S. G. Berg, and C. Ebeling, "Specifying and compiling applications for RaPiD," in Proc. IEEE Symp. FPGAs for Custom Computing Machines, Apr. 1998, pp. 116–125.

[6] A. Dasu and S. Panchanathan, "A survey of media processing ap- proaches," IEEE Trans. Circuits Syst. Video Technol., vol. 12, no. 8, pp.633–645, Aug. 2002.

[8] C. Ebeling, D. C. Cronquist, and P. Franklin, "RaPiD—Reconfigurable pipelined datapath," in Proc. 6th Int. Workshop Field-Programmable Logic and Applications, Aug. 1996, pp. 126–135.

[9] C. Ebeling, D. C. Cronquist, P. Franklin, J. Secosky, and S. G. Berg, "Mapping applications to the RaPiD configurable architecture," in Proc. IEEE Symp. FPGAs for Custom Computing Machines, Apr. 1997, pp.106–115.

[11] K.M. GajjalaPurna and D. Bhatia, "Temporal partitioning and sched- uling for reconfigurable computing," in Proc. 6th IEEE Symp. FPGAs for Custom Computing Machines, Apr. 1998, pp. 329–330.

[12] S. C. Goldstein et al., "PipeRench: A coprocessor for streaming multi- media acceleration," in Proc. 26th Annu. Int. Symp. Computer Architec- ture, May 1999, pp. 28–39.

[13] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. Taylor, "PipeRench: A reconfigurable architecture and compiler," IEEE Computer, vol. 33, no. 4, pp. 70–77, Apr. 2000.

[14] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao, "The chimaera reconfigurable functional unit," in IEEE Sympt. FPGAs for Custom Computing Machines, 1997, pp. 87–96.

[15] J.R. Hauser and J. Wawrzynek, "Garp: A MIPS processor with a reconfigurable coprocessor," in IEEE Symp. FPGAs for Custom Computing Machines, 1997, pp. 12–21.

[19] R. Laufer, R. R. Taylor, and H. Schmit, "PCI-PipeRench and the Sword- API: A system for stream-based reconfigurable computing," in Proc. 7th Annu. IEEE Symp. Field-Programmable Custom Computing Machines, Apr. 1999, pp. 200–208.

[21] T. Miyamori and K. Olukotun, "REMARC: Reconfigurable multi- media array coprocessor," in Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays, Feb. 1998, pp. 12–21.

[22] K. A. Olukotun, R. Helaihel, J. Levitt, and R. Ramirez, "A soft- ware-hardware cosynthesis approach to digital system simulation," IEEE Micro, vol. 14, pp. 48–58, Nov. 1994.

[23] R. Razdan, K. Brace, and M. D. Smith, "PRISC software acceleration techniques," in IEEE Int. Conf Computer Design, 1994, pp. 145–149.

[24] M. Rutten et al., "Eclipse: Heterogeneous multiprocessor architecture for flexible media processing," IEEE Des. Test Comput., vol. 19, no. 4, pp. 39–50, Jul.–Aug. 2002.

[25] M. B. Taylor et al., "The raw microprocessor: A computational fabric for software circuits and general purpose programs," IEEE Micro, vol.22, no. 2, pp. 25–35, Mar.–Apr. 2002.

[27] A. Wang, E. Killian, D. Maydan, and C. Rowen, "Hardware/software instruction set configurability for system-on-chip processors," in Proc. Design Automation Conf. (DAC2001), 2001, pp. 184–188.

[28] R. D. Witting and P. Chow, "OneChip: An FPGA processor with re- configurable logic," in IEEE Symp. FPGAs for Custom Computing Ma- chines, 1996, pp. 126–135.

[29] A.-Y. Wu, K. Liu, and A. Raghupathy, "System architecture of an adap- tive reconfigurable DSP computing engine," IEEE Trans. Circuits Sysy. Video Technol., vol. 8, no. 1, pp. 54–73, Feb. 1988.

[30] T. A. MOLLER, E. HAINES, "the real time rendering", second edition 2002