Extended Design Reuse Trade-Offs in Hardware-Software Architecture Mapping

F. Vermeulen *, F. Catthoor ‡, D. Verkest, H. De Man ‡,
IMEC, Kapeldreef 75, Leuven, Belgium.
* also Ph.D. student at the Katholieke Univ. Leuven
‡ also Professor at the Katholieke Univ. Leuven

ABSTRACT

In the design of embedded systems-on-chip, the success of a product generation depends on the flexibility to accommodate future design changes. This requirement influences the hardware-software partitioning strategy. Therefore we propose a novel hardware-software architecture and mapping methodology, which provide new trade-off opportunities for cost-effective component reuse.

1. INTRODUCTION

To make the design of systems-on-chip feasible, a tremendous increase in design productivity will be necessary. One of the keys to heavily increase current design productivity is design reuse at the level of system components. The flawless integration of reused components in a new context depends on standardization and flexibility in functionality and interfaces [1].

In today's design methodology, flexibility is traded off against energy-delay performance in the hardware-software partitioning phase. Based on the required flexibility, a target technology (custom or reconfigurable technology) and a processor architecture are selected (custom or instruction set processor).

With the processor architecture we will propose, we introduce interesting new trade-off possibilities in this selection. We obtain more optimal results because the design changes needed in successive versions of a product generation are typically small and local, but the performance penalty in any conventional co-design methodology is incurred on the entire component that is implemented in software or on a reconfigurable platform for flexibility reasons.

In the architecture we propose, we take a different approach towards providing the needed flexibility. We select the optimal target technology and processor architecture independently of reuse considerations, e.g. components benefitting from a custom hardware implementation are still implemented in their optimal architecture (but with a slightly adapted controller). Flexibility is added to the system as a separate (low cost, low performance) programmable component, which can take over control in those cycles where the functionality needs to change. Usually such a processor is already available in the system for some other background tasks and it has enough spare time to also execute the few changed subtasks. Specific techniques are used in the synchronization and in the control and context switch. In conventional "coprocessor" solutions, the granularity of switching between the two platforms is too large, leading to a still large penalty. In contrast, in our approach a novel protocol allows for fine-grain control. This is needed since it is not known in advance which execution cycles of the hardware realization will have to be substituted by a new functionality on the flexible platform. The fine-grain control is realized with a control-flow inspection mechanism and an interrupt mechanism. The overhead in the control switch is minimized thanks to a specific memory organization scheme, where all necessary data is available in shared memory. The performance (in terms of power and speed) of the resulting architecture can remain of the order of the original custom solution, while flexibility can become comparable to a software or field programmable solution.

The opportunity window for the proposed solution is in components that clearly benefit from the custom hardware implementation for power and speed reasons, and where a feasibility study shows a margin on the performance specifications (either in a new design or in a reuse library component). This opportunity window exists because of the orders of magnitude performance gap between custom hardware solutions and an instruction set processor implementation. Delay–energy trade-offs are made possible in a whole new range of solutions between pure hardware and software implementations.

As an example, we present in section 4 a small custom hardware component which originally consumes 7 mW and which we would like to reuse in a new design, but which requires small modifications in the new context. Our approach enables this reuse at a power consumption of 9.5 mW, which in many cases will still satisfy the specifications (which cannot be said about the 122 mW pure software implementation). Any part of the functionality can be adapted, but the performance penalty incurred when implementing some functionality in software which was formerly implemented

in hardware, typically limits the total amount of modifications to a small amount of code change. In practice, this is adequate to cope with evolving standards, bug fixes, or changes in user requirements.

2. RELATED WORK

In hardware-software co-design methodologies [2, 3, 4], a partitioning step decides which functionality is implemented as custom or as instruction set processor and on a custom or reconfigurable technology. This decision is based on energy consumption, speed, area, design cost and reusability. The global architecture consists of components communicating via one or more system busses and controlled by a global controller [5, 6, 7].

Integrating reusable components at the stage of hardware design is possible through configurable or parameterizable components. Flexibility in the component's functionality after hardware design can be provided by reconfigurable hardware (e.g. field programmable logic) and programmable hardware (e.g. instruction set processors).

To make custom hardware components more flexible, reconfigurable computing architectures ranging from general purpose FPGAs [8, 9, 10] to application specific architectures, especially with a specific memory architecture [11, 12, 13] are proposed. General purpose reconfigurable logic suffers from a performance penalty (area, power, clock frequency). This penalty can be reduced in dedicated reconfigurable architectures with custom designed subblocks. In order to guarantee the flexibility that will be necessary in the application, a large design-for-reuse investment is then needed.

Instruction set processors are more and more being customized with dedicated instructions (e.g. MMX [14]), data types and memory architectures [15, 16, 17]. This is supported by retargetable compilers and memory optimizing compilers [18, 19, 20, 21]. The trade-off is here between fine-grain software control and associated bus load bottleneck, and accelerators implementing complex operations. The latter solution limits flexibility. Design changes may make the (re)use of the accelerator hardware impossible. In the presented architecture, the hardware is not running as a slave of the instruction set processor, but both have synchronized master controllers. This allows to further reduce the data transfer and instruction control bottleneck, beyond the arithmetic bottleneck.

To obtain optimal performance especially in data-intensive applications, the data access mechanisms of the accelerator hardware also need careful integration with the instruction set processor memory architecture. Compared to our architecture, which also has this memory architecture integration, the accelerator hardware solution shows a higher design cost and less flexibility.

3. FLEXIBLE SYSTEM ARCHITECTURE

To allow the reuse of an optimized but inflexible IP component, even when small functional changes are necessary, we adapt its controller and add a flexible component (e.g. a small programmable processor) to the system (see Fig. 1). This flexible component, combined with our novel protocol, allows to implement any changed functionality at reuse

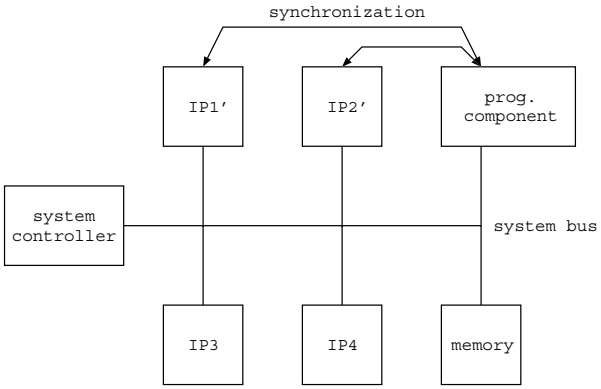

Figure 1: System architecture for a reusable platform. IP' are IP components with slightly adapted controller to support the proposed protocol.

time (providing that typically less than about 10% of the functionality is changed), creating the illusion of full programmability (see Fig. 2).

For the optimized IP component vs. flexible component combinations, multiple feasible implementations exist:

- custom processor – instruction set processor
- custom processor – reconfigurable hardware
- reconfigurable hardware – instruction set processor

In the following discussion, we will concentrate on the custom processor – instruction set processor platform.

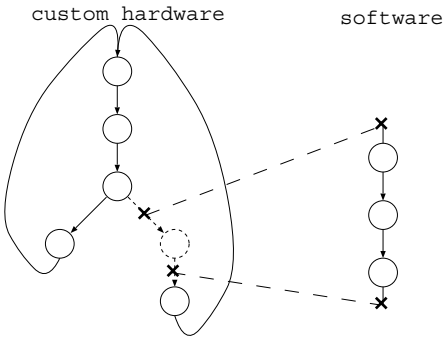

Figure 2: CDFG of original custom hardware functionality, with one part (dotted lines) replaced by a CDFG implemented in software. The crosses represent the context switching, occurring at a point which is not fully predefined when the chip is processed.

To obtain negligible power overhead when the custom hardware is performing its original functionality, both the custom and the instruction set processor run as master controllers. This minimizes the system bus load generated by the communication between hardware and software and it allows for a selective power-down of the instruction set processor, using conventional low-power techniques [22].

The novel protocol that allows the custom and the instruction set processor to run both as master, but still to be tightly synchronized, will be described in detail in a future publication. The hardware controller is modified to support this protocol. State information is communicated to the instruction set processor through a synchronization register at relevant points in the control flow. The modified controller allows for a flexible interrupt by the instruction set processor. After being interrupted, the custom hardware is powered down.

At the moment of the control switch between the custom and the instruction set processor, also a context switch has to happen. The context switch is usually the most expensive aspect, since copying data between custom and instruction set processor memory space (message passing) involves a large speed and power penalty. Therefore the proposed architecture exhibits a partly shared memory architecture between the custom and instruction set processors and we require the context to be in the shared memory at the moment of the potential control switch points. This limits the number of switching points in the hardware control flow. It means that in the memory hierarchy and management, a trade-off between (re)design cost and performance is involved, which can be tuned for a specific application context.

4. DEMONSTRATORS

To substantiate the relevance and feasibility of the methodology in real-life designs, drivers were taken from different application domains: video compression, wireless communication and ADSL (Asymmetric Digital Subscriber Loop) modem.

In the most simple instance of the proposed architecture (Fig. 3), we have a single IP component (custom hardware) and an external memory. The added instruction set processor chosen was an ARM7 [23]. The ARM was chosen as a power efficient small embedded processor with tool support (compiler, simulator, ...). The ARM should be regarded as one instance of a class of embedded processors (present in a reuse library), from which an optimal solution is selected. An ASIP (application specific instruction set processor) may have a more optimal performance on specific applications and can also have dedicated support for the synchronization protocol through specific instructions and registers. It requires more design effort to derive initially though, so a trade-off is involved.

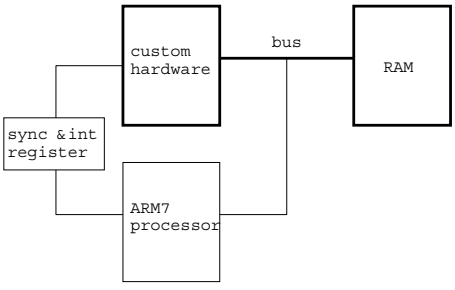

Figure 3: Simple instance of proposed flexible architecture (with original architecture in bold).

4.1 IDCT component in H.263

In the H.263 video conferencing block coding standard, a block-based encoding/decoding of image frames is performed to exploit spatial and temporal redundancy. The inverse discrete cosine transform (IDCT) operates on 8-by-8 blocks. The hardware was synthesized and the power was evaluated on the resulting netlist. The software was compiled using the ARM compiler and power was evaluated using the average power figure of 1.6 mW/MHz [23]. For a standard image size IDCT implemented in a $.35\mu$ technology, power consumption is 7 mW and area 2.8 mm^2. The real-time pure software solution needs 122 mW and 20.5 mm^2 (which is still optimistic, since the implemented algorithm matched the ARM rather well).

The finite state machine (FSM) description of the IDCT controller was adapted to support the synchronization protocol (Fig. 4). This means:

- adding a write to the external synchronization register (read by the instruction set processor) at selected points in the control flow. In this specific case, the loop counter is written at the end of the loop body.
- reading the external interrupt register (written by the instruction set processor) at synchronization points and jump to the interrupt state.
- adding an interrupt state. It polls the interrupt register. When the interrupt flag is cleared, the controller resumes execution at the state (e.g. loop counter) specified in the interrupt register.

The necessary adaptations were done at the behavioral level, as a post-processing step on an existing soft IP component [1], so the design effort is very limited.

We found that no speed overhead is incurred and limited area (1%) and power (2%) increases are due to adding the protocol control to the custom solution.

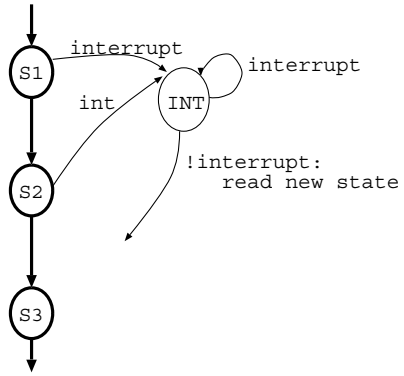

Figure 4: Custom hardware controller adapted to support the proposed protocol (original state transition diagram in bold).

Starting point for the software development at reuse time, is that the hardware behavior is fixed and hence fully known.

Synchronizing to the exact point of interruption is done through cycle counting and reading of the synchronization register. The low power goal is realized by powering down most of the instruction set processor when the hardware is active.

The final architecture has an area of 23 mm^2 which is approximately the combined area of the custom and instruction set processor. Power consumption is now 7.1 mW when nothing of the functionality is moved to the instruction set processor. When active, the processor adds a proportional share of 116 mW to this. The extra instructions to move control between custom and instruction set processor form a fixed cost, which is most of the time negligible with respect to the functional instructions.

Figure 5 shows the evolution of power consumption as function of the fraction of changed functionality (counted in ARM cycles compared to the cycle count of a pure software implementation). For example, if we change the specification for the corner block calculation (2% of the calculations), the power increases from 7 mW to 9.5 mW, which is still acceptable (which is not the case for the 122 mW pure software implementation) and would not legitimate the major design cost that would be incurred if the hardware were redesigned.

Remark that in these experiments we have normalized towards execution speed (all implementations execute the algorithm real-time). In practice one will trade-off execution time and power increase when adapting functionality at the time of reuse.

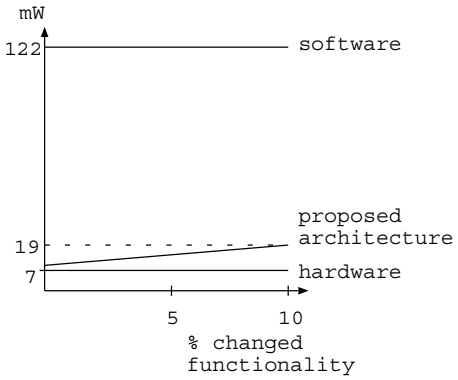

Figure 5: IDCT energy consumption as function of fraction changed functionality, evaluated for conventional and proposed architecture.

Apart from the off-the-shelf instruction set processor, two other alternatives to provide a flexible implementation platform were evaluated: an implementation on reconfigurable hardware and an ASIP implementation. The design was compiled on a Xilinx Virtex XCV600 FPGA and power has been estimated with Xilinx Virtex Power Estimator 1.5.

Between ASICs and general purpose instruction set processors, a large range of ASIP instruction set architectures can be designed, where flexibility and design time are spent to gain performance. W. Dougherty [24] et al. have found a factor 3 power increase going from an ASIC to a minimal ASIP for a FIR filter. We made an estimation of a best-case power consumption in an ASIP based on the power attribution in a StrongARM [25] low power processor and assuming the best case of same datapath and internal memory energy and only considering the instruction and memory management overhead. Energy consumption figures are summarized in Table 1. An indication of the (re)design cost is given to stress the design reuse trade-offs involved.

Implementation	Power (mW)	Area (mm^2)	(re)design cost
custom hardware	7	2.8	very high
proposed arch.	9.5	23.3	moderate
ASIP	16.1	17.8	high
FPGA	67	–	moderate
software	122	20.5	moderate

Table 1: Energy consumption and (re)design cost for different architectures.

4.2 Other applications

Our flexible reusable component architecture was also proven on a Fast Fourier Transform component used in Orthogonal Frequency Division Multiplexing (OFDM) based wireless Local Area Networks (LANs). The FFT is a major contributor in cost and can gain significantly from an optimized data storage organization. This explains the factor 41 in power difference found with respect to the instruction set processor implementation. The energy saving compared to a full software solution is again significant. If for example a simple windowing operation has to be added to the FFT, an (extrapolated) instruction set implementation would require 7.8 W, against 0.3 W for the solution with our hybrid architecture.

Another reusable component on which we have evaluated the proposed architecture is a Reed-Solomon forward error correcting code as found in ADSL modems. This application benefits even more from a specific memory architecture and from bit-level operations. That explains the 305 times larger power and 6 times larger area in a pure ARM implementation.

While relative power gains vary over application domains, the driver applications show that for applications benefitting from a custom hardware implementation, our solution provides a significant power gain compared to components implemented in software to provide the necessary flexibility.

5. CONCLUSION

We discussed the impact of design reuse goals on hardware-software architecture mapping, where flexibility has to be traded off against energy consumption and performance. We have proposed a novel hybrid processor architecture that allows to switch functionality from the original custom hardware to an added instruction set processor with minimal overhead. This architecture provides an interesting alternative to existing platforms for hardware-software mapping, as was substantiated by several realistic design experiments.

6. REFERENCES

[1] VSI Alliance, "VSI Alliance Architecture Document," *VSI Alliance*, 1.0 edition, 1997.

[2] M.Chiodo, P.Giusto, A.Jurecska, H.Hsieh, A.Sangiovanni-Vincentelli, L.Lavagno, "Hardware-Software Co-Design of Embedded Systems," *IEEE Micro,* pp.26-36, Aug. 1994.

[3] I.Bolsens, H.De Man, B.Lin, K.Van Rompaey, S.Vercauteren, D.Verkest, "Hardware-Software Co-Design of Digital Telecommunication Systems," *Proc. of IEEE,* spec. issue on HW-SW Co-Design, pp.391-418, Mar. 1997.

[4] A.Kalavade, E.Lee, "Manifestations of Heterogeneity in Hardware-Software Co-Design," *Proc. of Design Automation Conf.*, San Diego, CA, Jun. 1994.

[5] VSI Alliance, On-Chip Bus Development Working Group, "On-Chip Bus Attributes 1.0," *VSI Alliance,* Aug. 1998.

[6] D.Flynn, "AMBA: Enabling Reusable On-Chip Designs," *IEEE Micro,* Vol.17, No.4, Jul. 1997.

[7] H.Chang, L.Cooke, M.Hunt, G.Martin, A. McNelly, L.Todd, "Surviving the SOC Revolution," *Kluwer Ac. Publ,* Boston, MA, 1999.

[8] A.DeHon, J.Wawrzynek, "Reconfigurable Computing: What, Why, and Implications for Design Automation," *Proc. of Design Automation Conf.*, New Orleans, LA, pp.610-615, Jun. 1999.

[9] G.Borriello, C.Ebeling, S.Hauck, S.Burns, "The Triptych FPGA Architecture," *IEEE Trans. on VLSI Systems,* Vol.3, No.4, pp.491-501, Dec. 1995.

[10] F.Vahid, T.Givargis, "The Case for a Configure-and-Execute Paradigm," *7th Int. Workshop on HW/SW Co-Design,* Rome, Italy, pp. 59-63, May 1999.

[11] H.Schmitt, D.Thomas, "Synthesis of Application-Specific Memory Designs," *IEEE Trans. on VLSI Systems,* Vol.5, No.1, pp.101-111, Mar. 1997.

[12] P.Lippens, J.van Meerbergen, W.Verhaegh, "Allocation of Multiport Memories for Hierarchical Data Streams," *Proc. of IEEE/ACM Int. Conf. on Comp.-Aided Design,* Santa Clara, CA, pp. 728-735, Nov. 1993.

[13] S.Bakshi, D.Gajski, "A Memory Selection Algorithm for High Performance Pipelines," *Proc. of the European Design Automation Conf,* Brighton, Great Britain, Sep. 1995.

[14] A.Peleg, U.Weiser, "MMX Technology Extension to the Intel Architecture," *IEEE Micro,* vol.16, no.4, pp.42-50, Aug., 1996.

[15] *http://www.tensilica.com/.*

[16] *http://www.carmeldsp.com/.*

[17] P.Panda, N.Dutt, A.Nicolau, "Data Cache Sizing for Embedded Processor Applications," *Design Automation and Test in Europe Conf,* Paris, France, pp.925-926, Feb. 1998.

[18] S.Adve et al, "Changing Interaction of Compiler and Architecture," *IEEE Computer,* Vol.30, No.12, pp. 51-58, Dec. 1997.

[19] E.Torrie, M.Martonosi, C.Tseng, M.Hall, "Characterizing the Memory Behavior of Compiler-Parallelized Applications," *IEEE Trans. on Parallel and Distributed Systems,* Vol.7, No.12, pp.1224-1236, Dec. 1996.

[20] D.Truong, F.Bodin, A.Seznec, "Improving Cache Behavior of Dynamically Allocated Data Structures," *Int. Conf. on Parallel Architectures and Compilation Techniques,* Los Alamitos, CA, pp.322-329, 1998.

[21] Target Compiler Technologies NV, *http://www.retarget.com/.*

[22] "Low power CMOS design," (eds. A.Chandrakasan, R.Brodersen), *IEEE Press*, 1998.

[23] Advanced RISC Machines, Ltd, *http://www.arm.com/Pro+Peripherals/Cores/ARM7TDMI/*

[24] W.Dougherty, D.Pursley, D.Thomas, "Instruction Subsetting: Trading Power for Programmability," *IEEE Wshop on VLSI 1998,* Los Alamitos, CA, pp.42-47, 1998.

[25] J.Montanaro et al, "A 160-MHz, 32-b, 0.5W CMOS RISC Microprocessor," *IEEE Journal of Solid-State Circuits,* vol.31, no.11, pp.1703-14, Nov. 1996.