Reconfigurable Operator Based Multimedia Embedded Processor

Daniel Menard[1], Emmanuel Casseau[1], Shafqat Khan[1], Olivier Sentieys[1], Stéphane Chevobbe[2], Stéphane Guyetant[2], and Raphael David[2]

[1] INRIA/IRISA, CAIRN, 22100 Lannion, France
daniel.menard@irisa.fr
[2] CEA, LIST, 91191 Gif-Sur-Yvette, France

Abstract. Image processing applications need embedded devices that can integrate evolutionary standards or various standards, that is to say devices have to be flexible to implement different algorithms at different times. In other respects these devices are constrained with stringent power requirements as well as high performance. Reconfigurable processor can address these points. However, previous reconfigurable architectures suffer from their interconnect cost and do not meet low power constraints. In this paper preliminary work about the design of a reconfigurable processor based on a coarse-grain granularity tailored for multimedia applications is presented. The architecture is flexible and scalable. Coarse-grain operators can be optimized in term of the function they implement, the data word-length and the parallelism speed-up. The processor is designed to limit interconnection overhead.

1 Introduction

In multimedia applications, image processing is one of the major challenges embedded systems have to face. Image processing at pixel level, like image filtering, edge detection, pixel correlation or at block level such as motion estimation have to be considered. Such applications are typically computationally intensive with control statements and designers have to cope with power and performance stringent requirements when embedded system integration is investigated.

Focusing on the applicative domain permits some simplifications on the architecture, for instance on the choice of operators, the sizing of memories and the interconnect patterns. Moreover, the multimedia domain allows the use of subword parallelism SWP operators and fixed point arithmetic.

For that goal, we propose to develop a reconfigurable processor able to adapt its computing structure to image processing applications. The processor is built around a pipeline of coarse-grain reconfigurable operators exhibiting efficient power and performance features. On the contrary of previous attempts to design reconfigurable processors which have focused on the definition of complex interconnection networks between operators, we propose a pipeline-based of evolved coarse-grain reconfigurable operators to avoid traditional overhead, in reconfigurable devices, related to the interconnection network.

J. Becker et al. (Eds.): ARC 2009, LNCS 5453, pp. 39–49, 2009.

The reconfigurable processor presented hereafter is associated with a software framework currently on progress [1]. Roughly, from the high-level applicative specification, frequently executed code fragments, such as kernel loops, are extracted. A dependency graph-based internal representation is used to perform this pattern extraction. Pattern code transformation is then performed to exhibit features that the reconfigurable processor can efficiently implement [2]. A compilation step then generates the control code as well as the configurations of the operators.

The paper is organized as follow. Section 2 presents related work around reconfigurable processors. The organization of our reconfigurable processor is presented in section 3. Subword-based multimedia operator design is presented in section 4 as well as first implementation results. Finally conclusions are presented in section 5.

2 Related Work

The model of an ASIP processor with instruction-set extensions dedicated to a given application domain, such as Tensilica[3] or [4] can be pushed further with reconfigurable functional units inside the processor pipeline. In this design space, different trade-offs were explored such as DREAM[5] with a multi-context fine grain extension, or the Stretch processors [6] with a coarse-grain reconfigurable extension. These two solutions try to take advantage of the flexibility of the reconfigurable area to speed-up kernel loops throughout the application. Their drawbacks are mainly due to the sequentiality of the execution model based on the Von Neumann paradigm.

To decrease the constraints of the previous execution model on the intrinsic parallelism of the multimedia applications, processors based on coarse-grain arrays have been intensively explored, with roughly two approaches based on the data access. On the one hand, each processing element (PE) has a local register file, filled from external caches or data banks, global for the whole array. CRISP (Coarse-grained Reconfigurable Instruction-Set Processor)[7] clusterizes PEs into slices; ADRES [8] has a hybrid structure with either VLIW mode or an acceleration mode on the array; ASERP[9] is a recent proposal of such array with mixed SIMD/MIMD modes. The memory access and the size of the register file inside the PEs become a limitation when the application requires a high computational load combined with a high data bandwith.

On the other hand, several reconfigurable processors are based on computing patterns mapped on the array, that receive data from memory banks with address generators or fifos to cross over the previous memory access limitation. Thus this kind of processing is more oriented towards data streaming. The XPP, from PACT[10], is a standalone platform, that integrates a coarse grain streaming matrix with RAMs and small VLIW processors. The matrix by itself is a coprocessor, just as other proposals, as [11] which has a hierarchical structure, or [12] with complex address generators.

A last model derives from the processing-in-memory (PIM) concept: in this case, the reconfigurable PE's are associated with small memory cuts, as in the

MORA (Multimedia Oriented Reconfigurable Array)[13]. Each PE has its own micro-code; although similar to the ADRES array mode, the processing is here applied on larger data sets and the control is distributed. This solution takes advantage of the data locality of the application but pays it by a none negligible cost on the communication network.

Besides, as technology scales, gate delays become negligible in comparison with signal propagation in wires. Thus, lot of the previously defined reconfigurable architectures face the problem of keeping under control the communication delays in reconfigurable devices designed in advance technologies. Finally, together with algorithmic and technological evolutions, reconfigurable architectures have to face another challenge related to their energy efficiency. If reconfiguration paradigm has been intensively explored to demonstrate its interest on a performance or silicon density, very few projects have considered the optimization of dynamic and even more static power.

Power saving, as well as performance enhancement, can be addressed taking special care about the operators used to implement the computations. For example, power consumption is linked to data-word-length [14]. Thus, each operation word-length has to be adapted according to its contribution in the global system accuracy. Fixed architectures can not address this kind of optimizations when various applications or accuracy constraints are concerned. On the contrary, reconfigurable architectures can be specifically configured depending on the particular piece of code fragment to be implemented which leads to interesting power consumption gains.

3 Processor Design

3.1 Processor Design Flow

The processor described hereafter is a framework to define applicative-domain specific reconfigurable processors, targeting computing efficiency (expressed in GOPS/W) and flexibility inside the domain, expressed subjectively by the coverage of the application domain.

The design flow consists first in selecting the applicative domain, and defining the kind of computations that are typically found into the loop kernels and frequently executed code fragments. At this point, creating a bench representative of the targeted application domain can greatly facilitate the design space exploration by further refinements of the architecture. Selecting operations at a too low level, such as sum of absolute differences in image and video processing, or selecting an insufficient number of functions for the bench will lead to optimize only the peak performance of the architecture, but this will prove to be not representative enough of a real applicative behavior, in particular because control statements have a great impact on the overall performance.

After the complexity analysis of the algorithms, the operators can be extracted and optimized separately (cf. section 4). The operator is constrained only by the operator interfaces, and must be left complete freedom for the internal design,

provided that the operator behavior is respected. At this level the latency of the operators is not constrained, as it is considered as a performance issue, but is not mandatory for synchronization; moreover, data dependent processing will lead to unpredictable execution time inside the operators.

The usual chaining between the operators is also identified and the union of these patterns serves to create the operator interconnect scheme, simplified from the costly full crossbar, in order to mitigate the area cost.

A parameterizable number M of memories feed the operators; this number and their size are also decided according to applicative requirements, taking into account data dependencies, for example image lines or blocks.

Setting all these parameters defines an implementation of the processor that can be synthesized in order to conduct performance evaluation. The application is ported to the structure with the help of a toolchain that compiles the control code and prepares the configuration bitstreams for the operators. For further details on the toolchain, one can refer to [2].

3.2 Processor Description

A more detailed view of the processor template is depicted in figure 1. The high-level control and the datapath are separated. To face the increasing amount of control statements multimedia processing exhibits, data dependent processing for example, they are tightly coupled, so that an intermediate result from any operator can raise an interrupt at any time to the control sequence. A configuration interface allows the controller to feed the operators with configuration data. As the design of the processor started with the goal to avoid the classical area overhead due to reconfigurable interconnects in reconfigurable processors, with the flexibility reported inside the operator design, the interconnect is separated in two parts: one is the interconnect towards the memories, and the other is a low latency interconnect used for chaining the operators together.

The operator interconnect maps the high-level patterns that chain the operators; possibly all patterns are implementable at design time: some examples are SIMD patterns, where all operators do the same processing on independent data flows, up to the pipeline that chains all operators and that processes only one data stream, and all possible variants, with various Y-shaped patterns, with multiple inputs and one output.

The operators are configured by a bus interface: the configuration interface is memory mapped, but broadcast modes are provided to accelerate the loading of the operator configuration. In order to avoid reconfiguration overhead at run time, the size of the reconfiguration data is kept low, typically under a hundred bytes. It can consist of operator configuration and/or selection, internal simple routing data, constant parameters, functional look-up-tables, adaptive filter coefficients, or even a short microcode for a programmable finite state machine.

Address generators are used to create the streams from all memories, in order to relieve the control processor with the task of fetching data; the generators should provide at least the following access modes: base address and horizontal or vertical scan with a step, in both directions.

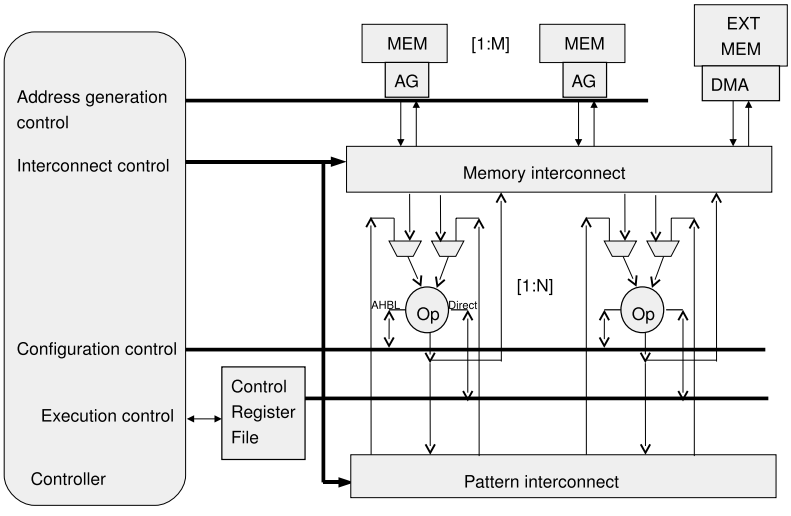

Fig. 1. Reconfigurable processor template architecture

3.3 Execution Model

The control code contains explicit operator reconfiguration commands, that are dynamically loaded before a function can start executing. Every function mapped on the processor follows the same steps: first the context has to be loaded; it is comprised of operator's configuration data, interconnect patterns, selection of involved memories, and program for address generators.

The execution of the function can then start with a control signal on the execution control interface with data that come from the external DMA or are already stored into memories from previous computations. To save time, unused operators or data memories can possibly be loaded during this step in preparation for the next computation. The processing step is finished as soon as the write-back address generator has completed its program. The status of the different operators can be checked, else the processor can step forward.

4 Reconfigurable Operators

The processor is made of reconfigurable operators operating concurrently. Computing efficiency and flexibility inside the multimedia domain drove the operator design. Both parallelism rate and efficiency can be increased if data-level parallelism is also implemented. Subword parallelism SWP [15] particularly suits image processing. With SWP each operand is partitioned into multiple lower precision operands, called subwords. Such as with SIMD, a single SWP instruction performs the same operation on multiple sets of subwords in parallel using SWP enabled operators. As a result, same datapath and operator can be used to

perform more than one computation on a composite word in parallel. In image and video applications, input data are mainly pixels thus are either 8 or 10 or 12-bits or sometimes 16-bits.

In order to handle this set of data sizes with a good efficiency/complexity trade-off, a 40-bit reconfigurable operator has been designed. When selected subword size is 8-bits then each input is considered as five 8-bit subwords packed in a 40-bit word. Hence on each 8-bit configuration, the SWP operator performs five same 8-bit basic operations in parallel. Basic operations are addition, subtraction, absolute value and multiplication (see section 4.1). For subword size of 10-bits, the SWP operator performs four 10-bit operations in parallel, and three 12-bit operations or two 16-bit operations for subword sizes of 12-bits or 16-bits respectively. In other words, the operator can be configured for both the computation it executes and the size of data.

4.1 Reconfigurable Operator Design

The architecture of the SWP reconfigurable operator is shown in figure 2. Mainly the SWP operator consists of SWP basic arithmetic units, SWP subwords adders units and accumulator unit. These units are connected in such a way that a variety of multimedia oriented basic and complex SWP operations can be performed, such as sum of absolute differences SAD for motion estimation algorithm, sum of products for discrete cosine transform DCT algorithm... Multiplexers are used to provide appropriate data to arithmetic units. Control signals for the multiplexers and enable signals for registers (not shown in figure 2) are provided externally by the controller based upon the selected operation. The subword size is selected by control bits which are communicated to all SWP units (not shown in figure 2). To reduce the switching activity, all operators are guarded, so that unused units can be disabled.

Basic SWP arithmetic units are used to perform basic arithmetic operations on the selected subword size data. These operations include SWP ($a \pm b$) signed, SWP (abs) signed, SWP ($a \times b$) signed/unsigned, SWP $|a-b|$ unsigned, SWP (a + b) unsigned. SWP ($a \pm b$) operator is used to perform addition or subtraction of signed subwords data. Its architecture is based upon the breaking of carry chain at subwords boundaries. SWP (abs) signed unit is used to perform the absolute operation on signed subwords data. Operator abs takes the two's complement of subwords depending upon the value of most significant bit.

SWP ($a \times b$) signed/unsigned unit is used to perform SWP multiplication of signed as well as unsigned data. Because multiplication hardware implementation is area and delay costly, particular care was taken for the design of the multiplier. Its implementation is based upon an extension of the SWP multiplier proposed in [16] which supports only classical subword sizes (8, 16 and 32-bit). Here the multimedia oriented subword sizes of 8, 10, 12 and 16-bits are considered which do not have any uniform arithmetic relation with the word size (40-bit) of the SWP operator. Partial product bits are generated in such a way that they remain valid for all subwords size multiplications. Hence no suppression and detection of carries is required at subword boundaries.

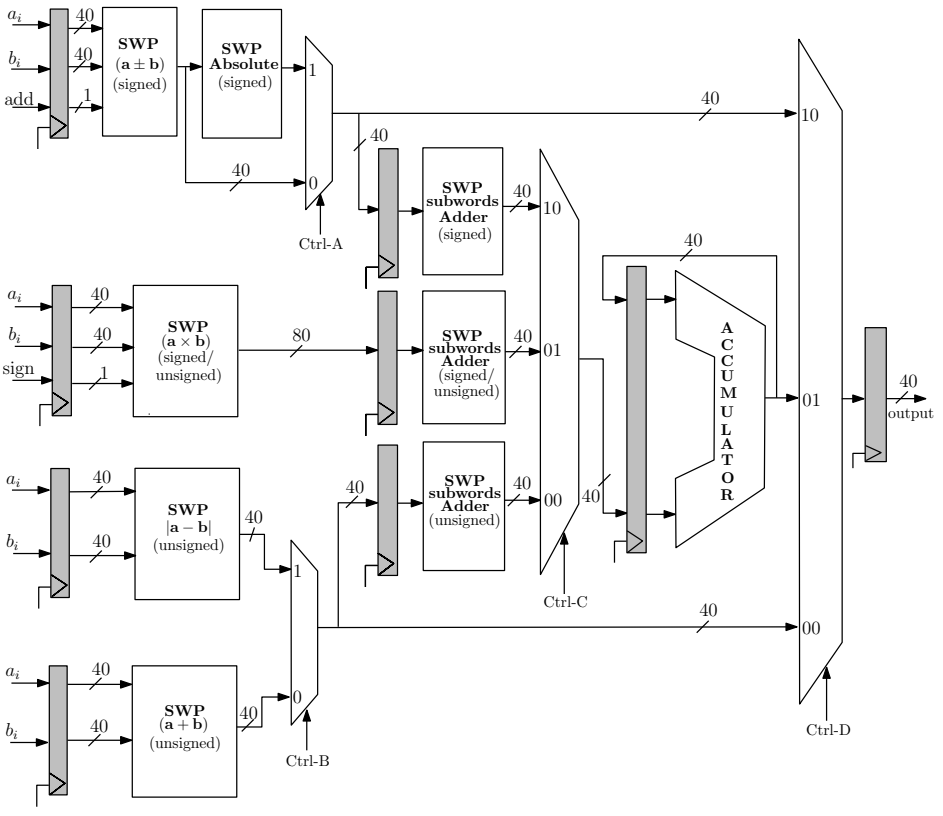

Fig. 2. Reconfigurable Multimedia SWP Operator

SWP $|a-b|$ unsigned unit is used to perform absolute difference of unsigned subwords. This unit is required because usually the pixels are stored as unsigned data. To avoid the absolute operation, this unit either calculates $a-b$ (when $a > b$) or $b-a$ (when $b > a$).

Like the inputs, the output data of all the basic SWP units consist of 40-bits except for the SWP multiplier whose output consists of 80-bits. The basic arithmetic units produce outputs in the form of resultant subwords packed in a register. As per the requirement, these packed subwords can be obtained at the output of the reconfigurable operator through the use of appropriate control bits. As a single multiplication is not usually required in multimedia applications, and also because the output of the SWP operator is 40-bits wide, the multiplier unit output is not directly routed to the output of the reconfigurable operator. However the accumulation of products can be obtained at the output of the operator through the *SWP subword adder*.

For certain multimedia applications more complex operations are required. For instance ($\sum |a-b|$) operation is required in the calculation of sum of absolute

differences (SAD). Similarly $\sum a \times b$ signed/unsigned operation is required for the multiplication-accumulation operation used in the DCT algorithm. Rather than subwords which provide loss of bit, these operations produce single 40-bit accumulated values at the output. To perform these complex operations *SWP subword adder* units and accumulator unit are used in addition to basic SWP arithmetic units. The inputs to *SWP subword adder* unit are resultant subwords from different basic SWP arithmetic units. Based upon the selected subword size, the *SWP subword adder* unit separates the N subwords x_i packed in the input register and then performs the addition of these subwords to generate a single 40-bit sum value. The expression of the *SWP subword adder* output z_{sa} is equal to

$$z_{sa} = \sum_{i=0}^{N-1} x_i \tag{1}$$

Before the addition, *SWP subword adder* unit performs either sign extension (for signed subwords) or zero padding (for unsigned subwords) depending upon the selected data format. The output of *SWP subword adder* unit can then be accumulated recursively using the accumulator to obtained the required operation output. As an example, in order to perform the signed operation $\sum a \times b$ with subword size of 8-bit, SWP $(a \times b)$ unit produces a 80-bit product value. This product value is used as input to a SWP subword adder. This adder considers this 80-bit input as five 16-bit subword products (without loss of bit) and adds them to generate a 40-bit value. Then at each clock cycle the accumulator is used to accumulate this 40-bit value with the previous values to generate $\sum a \times b$ term at the output of the reconfigurable operator. As the inputs to accumulator are single values instead of packed subwords, therefore SWP capability is not required for the accumulator unit implementation. Likewise the accumulator generates the single output without loss of bit.

The other complex SWP operations which can be performed using this SWP operator are $\sum a \pm b$ signed, $\sum |a \pm b|$ signed, $\sum |a - b|$ unsigned, $\sum a + b$ unsigned. Based upon the requirements, any combination of these operations can also be obtained such as $\sum a \times b + \sum |a \pm b|$ signed etc. For the complex operations which involve the accumulation of results generated by basic units, the output word-length depends upon the number of values need to be accumulated. In the worst case when performing $\sum a \times b$ operation on 16-bit subwords, the output of the SWP $(a \times b)$ unit consists of two 32-bit subwords. As the accumulator is 40-bits wide, the extra eight bits are used as guard bits to avoid any overflow. Therefore this operator can perform at least 256 (2^8) accumulations of worst data length product terms. For other smaller subword data sizes, the numbers of guard bits are greater and thus the number of accumulations which can be performed increases further without any overflow.

4.2 Synthesis Results

To analyze the area, speed and power consumption, overall reconfigurable operator design is synthesized to ASIC standard cell HCMOS9GP 130nm

Table 1. Synthesis results on ASIC technologies

Technology	NAND Gates	Critical Path (ns)	Gates × CP	Dynamic Power (mW)	Leakage Power (μW)
90nm CMOS	24063	7.7	185285	2.5	534
130nm CMOS	25379	7.6	192880	4.1	48

(CORE9GPLL 4.1 low leakage standard cell library from ST Microelectronics) and 90nm (fsd0t_a standard performance low voltage threshold cell library from UMC) technology using Synopsys Design Vision and to FPGA (Xilinx Virtex II) using Mentor Graphics Precision RTL tool. The area, critical path (CP) and power consumption have been measured. Table 1 shows the results obtained for the two ASIC technologies. To analyze the overall efficiency, product of gates and critical path (CP) is also computed. Smaller value of this product term indicates higher efficiency. On ASIC technologies, implementations are made for a clock period of 8ns. When clock frequency is decreased, the area reduces accordingly but the overall efficiency reduces because the throughput of SWP operator reduces with the decrease in clock frequency. On the FPGA Virtex II platform, 2793 CLBs are required and the critical path is equal to 17.2 ns. Actually area and CP overheads for implementing SWP capability are less on ASIC technology compared to FPGA technology. The reason is that in FPGA implementation resources are CLBs rather than gates as in ASIC. Therefore ASIC resources better suits the SWP designs.

4.3 Results Analysis

The unit which consumes maximum design resources is the SWP multiplier. Although the multiplier architecture is based on [16] which is known to be for far more efficient for SWP implementation than conventional multiplier architectures, it consumes almost 60% of total SWP operator area. The other blocks like signed arithmetic units, unsigned arithmetic units, subwords adder units and register units consumes respectively 8%, 9%, 14% and 6% of total area. Similarly power consumption of the SWP multiplier is also more compared to other units. SWP multiplier unit consumes almost 50% of total power. The other blocks like signed arithmetic units, unsigned arithmetic units and subwords adder units consumes respectively 9%, 19% and 15% of total power.

At present time, reconfigurable operators have been synthesized, and first performance assessment can be given at the operator level. The sum of absolute difference kernel used in motion estimation algorithms is a good candidate at this granularity level. For comparison, state-of-the-art Texas Instruments (TI) TMS320C64x DSP architecture is used. The processing unit of the TI DSP is made-up of two clusters. Each cluster consists of four functional units among with one multiplier and two arithmetic and logic units. This architecture provides SWP capabilities based on 8, 16 and 32 bit data word-lengths. For a fair comparison, one reconfigurable operator is considered for our processor and one cluster is considered for the TI DSP. For 16-bit pixels, the number of cycles

N_{cycles} required to compute the SAD applied to 16 by 16 image blocks is 128 for both implementations. N_{cycles} is 60 and 64 for our operator and TI DSP based solution respectively when 8-bit pixels are considered. For 10 and 12 bit pixels, the granularity in term of data size of our operator allows the number of cycles to be reduced. N_{cycles} is reduced by 50% and 25% for 10 and 12 bit pixels respectively. In practice, processing is spread on two clusters with TI DSP so N_{cycles} is divided by two.

5 Conclusion

In this paper the design of a reconfigurable processor tailored for multimedia processing is introduced. The architecture is designed to provide a good trade-off between performance and power consumption targeting embedded devices. The processor is based on pipelined coarse-grain reconfigurable operators that have flexibility and scalability properties.

Future work will consist in porting real applicative cases with the help of the software framework on a FPGA-based demonstrator for validation purpose. The scalability of the processor template allows to explore the design space in order to extract performance and power consumption metrics. For instance, each operator can be configured to complete a SAD on a particular image block, and performance assessment of the motion estimation part of a video codec will be performed with various number of operators.

Acknowledgment

This work is supported by the french *Architectures du Futur* ANR program ANR-06-ARFU-004.

References

1. `http://roma.irisa.fr`
2. Wolinski, C., Kuchcinski, K.: Automatic Selection of Application-Specific Reconfigurable Processor Extensions. In: DATE 2008: Proc. of the Conf. on Design, automation and test in Europe, pp. 1214–1219 (2008)
3. `http://www.tensilica.com`
4. Mingche, L., Jianjun, G., Zhuxi, Z., Zhiying, W.: Using an Automated Approach to Explore and Design a High-Efficiency Processor Element for the Multimedia Domain. In: CISIS 2008: International Conference on Complex, Intelligent and Software Intensive Systems, pp. 613–618 (2008)
5. Campi, F., Deledda, A., Pizzotti, M., Ciccarelli, L., Rolandi, P., Mucci, C., Lodi, A., Vitkovski, A., Vanzolini, L.: A dynamically adaptive DSP for heterogeneous reconfigurable platforms. In: DATE 2007: Proc. of the Conf. on Design, automation and test in Europe, pp. 9–14 (2007)
6. `http://www.stretchinc.com`

7. Barat, F., Jayapala, M., Aa, T.V., Deconinck, G., Lauwereins, R., Corporaal, H.: Low power coarse-grained reconfigurable instruction set processor. In: Y. K. Cheung, P., Constantinides, G.A. (eds.) FPL 2003. LNCS, vol. 2778, pp. 230–239. Springer, Heidelberg (2003)
8. Veredas, F.J., Scheppler, M., Moffat, W., Mei, B.: Custom implementation of the coarse-grained reconfigurable ADRES architecture for multimedia purposes. In: FPL 2005, Int. Conf. on Field Programmable Logic and Applications, pp. 106–111 (2005)
9. Yeo, S., Lyuh, C., Roh, T., Kim, J.: High Energy Efficient Reconfigurable Processor for Mobile Multimedia. In: ISSCC 2008: Int. Conf. on Circuits and Systems for Communications, pp. 618–622 (2008)
10. http://www.pactxpp.com
11. Yazdani, S., Cambonie, J., Pottier, B.: Reconfigurable Multimedia Accelerator for Mobile Systems. In: SOCC 2008: 21st Annual IEEE Int. SOC Conference (2008)
12. Carta, S.M., Pani, D., Raffo, L.: Reconfigurable Coprocessor for Multimedia Application Domain. J. of VLSI Signal Processing 44, 135–152 (2006)
13. Lanuzza, M., Perri, S., Corsonello, P., Margala, M.: A New Reconfigurable Coarse-Grain Architecture for Multimedia Applications. In: AHS 2007: Second NASA/ESA Conf. on Adaptive Hardware and Systems (2007)
14. Yasuura, H., Tomiyama, H.: Power optimization by datapath width adjustment. In: Pedram, M., Rabaey, J.M. (eds.) Power aware design methodologies. Kluwer, Dordrecht (2002)
15. Fridman, J.: Sub-Word Parallelism in Digital Signal Processing. IEEE Signal Processing Magazine 17(2), 27–35 (2000)
16. Krithivasan, S., Schulte, M.J.: Multiplier Architectures for Media Processing. In: Thirty seventh Asilomar Conf. on Signals, Systems and Computers, vol. 2, pp. 2193–2197 (2003)