The Virtex II Pro™ MOLEN Processor

G.Kuzmanov, G.N. Gaydadjiev, and S. Vassiliadis

Computer Engineering Lab, EEMCS, TU Delft, The Netherlands
E-mail : {G.Kuzmanov,G.N.Gaydadjiev,S.Vassiliadis}@ET.TUDelft.NL
http://ce.et.tudelft.nl/

Abstract. We use the Xilinx Virtex II Pro™ technology as prototyping platform to design a MOLEN polymorphic processor, a custom computing machine based on the co-processor architectural paradigm. The PowerPC embedded in the FPGA is operating as a general purpose (core) processor and the reconfigurable fabric is used as a reconfigurable co-processor. The paper focuses on hardware synthesis results and experimental performance evaluation, proving the viability of the MOLEN concept. More precisely, the MPEG-2 application is accelerated very closely to its theoretical limits by implementing SAD, DCT and IDCT as reconfigurable co-processors. For a set of popular test video sequences the MPEG-2 encoder overall speedup is in the range between 2.64 and 3.18. The speedup of the MPEG-2 decoder varies between 1.65 and 1.94.

1 Introduction

The MOLEN polymorphic processor [9], a Custom Computing Machine (CCM) based on the co-processor architectural paradigm, resolves some shortcomings of many recent reconfigurable processors (e.g., opcode space explosion, modularity and compatibility problems identified for [2, 3, 6]). More specifically, the MOLEN concept suggests that for a given ISA, a single architectural extension comprising between 4 and 8 additional instructions suffices to provide an almost arbitrary number of reconfigurable functions. In addition, unlike [1, 11], the concept allows implementations with large, virtually unlimited, number of input and output parameters for the reconfigurable functions. In this paper, we present a prototype design of the MOLEN CCM utilizing a Virtex II Pro™ FPGA of Xilinx [10]. The implemented minimal PowerPC ISA augmentation comprises only four instructions and the reconfigurable hardware utilization of the prototype is extremely low with only 156 FPGA slices consumed. Experimental results indicate that this prototype realization can speedup the MPEG-2 encoder between 2.64 and 3.18 when implementing SAD, DCT and IDCT as reconfigurable functions. When implementing the IDCT operation alone, the projected speedup of the MPEG-2 decoder is between 1.65 and 1.94 for a set of popular test video sequences. The reconfigurable hardware costs of the aforementioned functions is between 8% and 53% of the utilized xc2vp20 chip, depending on the particular configuration considered.

The remainder of the paper is organized as follows. Section 2 briefly introduces the MOLEN CCM background. In Section 3, Virtex II Pro™ specific design considerations and hardware evaluation of the MOLEN prototype infrastructure are presented. Considering MPEG-2, Section 4 evaluates the prototype based on experimental results.

Reconfigurable hardware utilization by the custom functional units and performance estimates are reported. Finally, concluding remarks are presented in Section 5.

2 Background on the MOLEN CCM

This section briefly describes the MOLEN $\rho\mu$-coded CCM organization, originally introduced in [9]. The two main components in the MOLEN machine organization (depicted in Figure 1) are the *Core Processor*, which is a general-purpose processor (GPP), and the *Reconfigurable Processor* (RP). The ARBITER performs a partial decoding on the instructions in order to determine where they should be issued. Instructions implemented in fixed hardware are issued to the GPP. Instructions for custom execution are redirected to the RP. Data transfers from(to) the main memory are handled by the *Data Load/Store* unit. The *Data Memory MUX/DEMUX* unit is responsible for distributing data between either the reconfigurable or the core processor. The reconfigurable processor consists of the *reconfigurable microcode* ($\rho\mu$-code) unit and the *custom computing unit* (CCU). The CCU consists of reconfigurable hardware and memory, intended to support additional and future functions that are not implemented in the core processor.

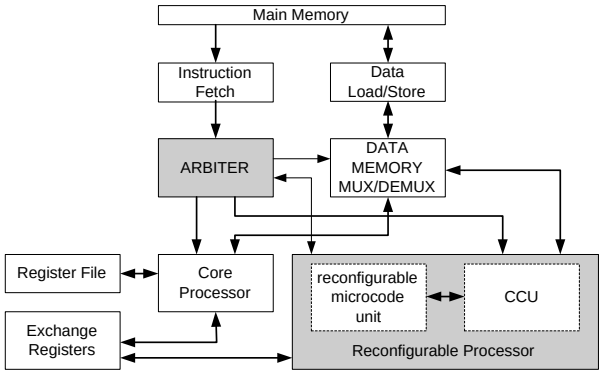

Fig. 1. The MOLEN machine organization

Pieces of application code can be implemented on the CCU in order to speed up the overall execution of the application. A clear distinction exists between code that is executed on the RP and code that is executed on the GPP. Data must be transferred across the boundaries in order for the overall application code to be meaningful. Such data includes predefined parameters (or pointers to such parameters) or results (or pointers to such results). The parameter and result passing is performed utilizing the so-called *exchange registers* (XREGs) depicted in Figure 1.

An operation, executed by the RP, is divided into two distinct phases: set and execute. The set phase is responsible for reconfiguring the CCU enabling the execution of the operation. Subsequently, in the execute phase the actual execution of the operations is performed. No specific instructions are associated with specific operations to configure and execute on the CCU as this would greatly reduce the opcode space. Instead, pointers to *reconfigurable microcode* ($\rho\mu$-code) are utilized. The $\rho\mu$-code emulates both

the configuration of the CCU and the execution of implementations configured on the CCU resulting in two types of microcode: 1.) reconfiguration microcode that controls the configuration of the CCU; and 2.) execution microcode that controls the execution of the implementation configured on the CCU.

Arbiter operation. The arbiter directs instructions to either the core processor or the reconfigurable processor. A general view of an arbiter organization, the operation of which is entirely based on decoding the instruction flow is depicted in Figure 2. Instructions

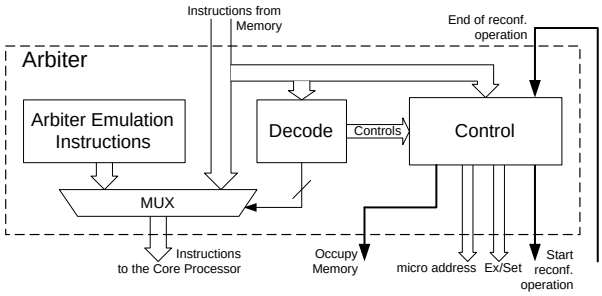

Fig. 2. General arbiter organization.

from the original GPP ISA are directed (via MUX) to the core processor. Up-on decoding of an RP instruction, "arbiter emulation instructions" are multiplexed through the core processor instruction bus to drive the GPP into a wait state. In the same time, control signals are issued (via the control block in Figure 2) to the RP (in essence the $\rho\mu$-code unit) to initiate a reconfigurable operation. The microcode location address is redirected to the $\rho\mu$-code unit and the data memory control is transferred to the RP. After an RP operation is completed, data memory control is released back to the GPP, an instruction sequence is generated to ensure that the GPP exits the wait state and the program execution continues with the instruction immediately following the executed RP instruction. More details regarding arbiter operation can be found in [5].

The $\rho\mu$-code unit comprises three main parts: the sequencer, the reconfigurable control (ρ-control) store, and the reconfigurable microcode ($\rho\mu$-code) loading unit. The $\rho\mu$-code loading unit, loads $\rho\mu$-code into the ρ-control store from the main memory. The sequencer generates the address of the next microinstruction from the ρ-control store. More details regarding $\rho\mu$-code loading and related design considerations can be found in [4]. The ρ-control store is used to store microcodes and comprises two sections - a *set* and an *execute* section. Both sections can be identical and are further divided into a *fixed* and a *pageable* part. The fixed part stores the resident reconfiguration and execution microcode of the *set* and *execute* phases, respectively. Other microcode is stored in memory and the pageable part of the ρ-control store acts like a cache to provide temporal storage. For more details regarding the considered cache mechanisms and the ρ-control store organization, the interested reader is referred to [9].

The MOLEN programming paradigm is a sequential consistency paradigm targeting the MOLEN organization (for details see [7]). The complete list of the eight required instructions, denoted as polymorphic instruction set architecture (πISA), is as follows: 1) partial set (p-**set** $<address>$) performs common and frequently used configurations;

2) complete set (c-**set** $<address>$) *completes* the CCU's configuration to perform less frequent functions; 3) **execute** $<address>$: controls the execution of the operations on the CCU configured by the set instructions; 4) **set prefetch** $<address>$ and 5) **execute prefetch**: prefetch the needed microcodes responsible for CCU reconfigurations and executions into a local on-chip storage (the $\rho\mu$-code unit); 6) **break**: synchronizes the parallel execution of the RP and the GPP; 7) **movtx** $\text{XREG}_a \leftarrow \text{R}_b$ and 8) **movfx** $\text{R}_a \leftarrow \text{XREG}_b$: move the content of general-purpose register R_b to/from XREG_a. The $<address>$ field in the instructions introduced above denotes the location of the reconfigurable microcode for the configuration and execution processes.

We note that it is not imperative to include all eight instructions when implementing a MOLEN organization. In our prototype design, we have considered a **minimal πISA** which comprises four basic instructions, namely **set** (more specifically: c-**set**), **execute**, **movtx** and **movfx**. By implementing the first two instructions (**set**/**execute**) any CCU implementation can be loaded and executed on the reconfigurable processor. The **movtx** and **movfx** instructions are needed to provide the input/output interface between the RP targeted code and the remainder application code. The minimal πISA is essentially the smallest set of MOLEN instructions needed to provide a working scenario.

3 Prototype design considerations and hardware estimation

In this section, we present Virtex II Pro™ specific design considerations and evaluate the reconfigurable hardware utilized by the MOLEN prototype infrastructure, i.e., excluding any CCU implementations. We consider a minimal πISA comprising the instructions **set**, **execute**, **movtx**, and **movfx**. Specific software considerations and πISA instruction encoding are referred. Some implementation issues are presented as well.

The development platform. We experimented with the Alpha Data XPL Pro™ lite development board (ADM-XPL). As a reconfigurable hardware platform, we used a Xilinx xc2vp20-5 device from the Virtex II Pro™ family. The MOLEN organization has been described in VHDL and synthesized by the Xilinx XST tool of ISE 5.2, SP3.

Software considerations. Due to performance reasons, we do not use PowerPC special operating modes instructions as arbiter emulation instructions (e.g., exiting power-down modes requires an interrupt). We employed the *'branch to link register'* instruction (**blr**) to emulate a 'wait' state and *'branch to link register and link'* (**blrl**) to move the processor out of this state. Thus the arbiter emulation instructions (Figure 2) are reduced to only one instruction for 'wait' and one for 'wake-up'. Implementation details and additional performance enhancing software considerations are discussed in [5].

Instruction encoding. We mapped the **movtx** and **movfx** instructions to the existing PowerPC instructions **mtdcr** and **mfdcr**. This implementation solution has been imposed by the fact that the Virtex II Pro™ PowerPC core has a dedicated interface to the so called Device Control Registers (DCR) [10]. We implemented the XREGs as DCRs and utilized the **mtdcr** and **mfdcr** instructions to support XREG transfers. Thus only the **set** and **execute** instructions have to be considered for encoding. As a guideline, we decided to closely follow the already established PowerPC instruction format. We have chosen an opcode from the set of unused opcodes to represent both instructions. Figure 3 depicts the implemented RP instructions format, referred to as *the ρ-form*. The manner

to distinguish a **set** instruction and an **execute** instruction (using the same opcode) is via instruction modifiers. This encoding allows us to utilize a 24-bits address (embedded in the instruction word) to specify the location of the microcode. Finally, we have to note that within this address field, a modifier bit R/P (resident/pageable) specifies where the microcode is located and how to interpret the address field. That is, either a location in the memory (R/P=1) or in the on-chip $\rho\mu$-code unit (R/P=0).

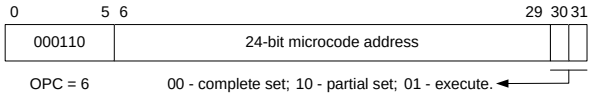

Fig. 3. The ρ-form: **set** (p-**set**, c-**set**) and **execute** instructions.

Memory organization. For the memory design, we considered the on-chip memory blocks of the utilized FPGA. The available BRAM blocks in xc2vp20 allow the implementation of 128 KBytes memory for both data and instructions. The PowerPC has a Harvard architecture with separated instruction and data addressing spaces. Therefore, for better performance, we separated the main memory into two equal segments - 64 KBytes for instructions and other 64 KBytes for application data. In this case, we note that the amount of memory is limited only by the available on-chip memory. By utilizing external memories, it is possible to extend the memory volume up to the entire memory space addressable by PowerPC (i.e., 32-bit addresses). The later option, however, has not been considered in our prototype and in the experiments to follow.
Clock domains. Due to the polymorphic nature of the MOLEN processor and for performance efficiency, three clock domains have been implemented in our prototype:

- **PPC_clk**- clock signal to the core processor. The frequency of this signal has been set to 250 MHz, the maximum recommended for the PowerPCs in xc2vp20-5;
- **mem_clk**- clock signal to the main memory. This signal has been set to be three times lower than the PPC_clk, i.e., 83 MHz;
- **CCU_clk**- clock signal to the CCU driven by an external pin. It may be utilized by any CCU, which requires frequencies, different from the PPC_clk and mem_clk.

Additional design parameters. For the prototype implementation, we have considered a microcode word length of 64 bits. A 32MByte memory segment has been considered for storing microprograms into a 64-bit organized main memory. The ρ-control store has been designed to handle 8KBytes 64-bit microcode words. As primary microcode storage units for the ρ-control store, we have used the BRAM blocks of the FPGA fabric, configured as a dual port memory. Each port is unidirectional - a read-only port is used to feed the microinstruction register, while a write-only port loads microcodes from the external memory into the pageable section of the ρ-control store. The XREGs have been implemented in a single BRAM organized as 512×32-bit storage.
Synthesis results. Hardware costs reported by the synthesis tools are presented in Table 1. The first column displays the FPGA resources considered. Column two reports the actual values of these resources, consumed by the reconfigurable processor, without considering any CCU implementation, i.e., the $\rho\mu$-code unit and the associated infrastructure. This includes the $\rho\mu$-code loading unit, the sequencer and the ρ-control

store. Column three presents resource utilization of the arbiter. In column four, the resources consumed by the entire MOLEN organization are displayed, including the reconfigurable processor infrastructure, the arbiter and the XREGs. Finally, columns five and six respectively present the available FPGA resources in the xc2vp20 chip and the utilized part of these resources by the MOLEN organization (in %). Synthesis results strongly suggest that the MOLEN infrastructure consumes trivial hardware resources, thus leaving virtually all FPGA resources available for CCU implementations.

Table 1. MOLEN Organization Synthesis Results (* without any CCU implemented)

Device xc2vp20 Speed Grade -5	Reconfig. Processor*	Arbiter	Total incl. XREGs	Available Resources	%
Number of Slices	71	84	156	10304	1
Number of Slice Flip Flops	78	69	147	20608	1
Number of 4 input LUTs	171	150	322	20608	1
Number of BRAMs:	4	N.A.	5	112	3
Maximum Frequency [MHz]	130	143	130	N.A.	N.A.

4 Prototype evaluation

In this section, we describe the experiments that have been carried out to evaluate the MOLEN prototype performance and report the obtained results. We target and profile the MPEG-2 application. The profiling data are used to identify and design performance critical kernels as CCU implementations. Due to memory limitations, we run only the extracted kernels on the prototype MOLEN processor and directly measure the performance gains. Using these measurements, the profiling data, and Amdahl's law, we estimate the projected overall speedup, rather than directly run the entire MPEG-2 application on MOLEN. Hardware estimations for the considered CCUs are presented.

Software profiling results. The first step of the experimentation involves identifying the functions that are suitable for hardware implementations. The objective is to identify the most time-consuming kernels from the application. To this purpose, we performed the measurements on a PowerPC 970 running at 1600 MHz. The considered application is the Berkeley implementation of the MPEG-2 encoder and decoder included in libmpeg2. As input data, we used a representative set of four popular video sequences, namely *carphone*, *claire*, *container* and *tennis*. Profiling results for each considered function and its descendants (obtained with the GNU profiler **gprof**) are presented in Table 2. For the MPEG2 encoder, the total execution time spent in SAD, DCT and IDCT operations (Table 2, column 6) emphasizes that these functions require around 2/3 of the total application time. Although the IDCT function in MPEG2 encoder takes only around 1% of the application time (Table 2, column 5), in the MPEG2 decoder it requires on average around 42%. Consequently, all considered functions are good candidates for hardware implementations although their individual contribution to the performance improvement may differ per sequence and application.

Synthesis results for the considered CCU implementations. We implemented the functions, suggested by the profiling results, into reconfigurable hardware. Synthesis results for the xc2vp50 chip are reported in Table 3. For the SAD function, we implemented the organization proposed in [8]. The super-pipelined 16-byte version of this

Table 2. MPEG2 profiling results for each of the considered functions and its descendants

		MPEG2 encoder				MPEG2 decoder
sequence	# frames@Resolution	SAD(16x16)	DCT(8x8)	IDCT(8x8)	Total	IDCT(8 x 8)
carphone	96@176x144	51.1 %	12.5 %	1.3 %	64.9 %	50.4 %
claire	168@360x288	53.8 %	11.8 %	1.0 %	66.6 %	37.6 %
container	300@352x288	56.2 %	10.7 %	1.0 %	67.9 %	40.4 %
tennis	112@352x240	60.0 %	9.5 %	0.8 %	70.3 %	40.5 %

SAD organization (SAD16) is capable of processing one 16-pixel line (1 pixel is 1 byte) of a macroblock in 17 cycles at over 300 MHz. The 128-byte version (SAD128) processes eight macroblock lines in 23 cycles, and the 256-byte version (SAD256), processes an entire 16x16-pixel macroblock in 25. SAD256 requires more resources than available in the xc2vp20 chip, therefore we consider it for future implementation on a larger FPGA (e.g., xc2vp50). To support the DCT and IDCT kernels, we synthesized the 2-D DCT and 2D-IDCT v.2.0 cores available as IPs in the Xilinx Core Generator Tool. Considering the implemented clock domains and synthesis results (from Table 3) in our experiments, we have run the DCT and IDCT functions at mem_clk frequency (83MHz). The SAD designs were clocked by PPC_clk (250MHz).

Table 3. Synthesis Results per CCU implementation

Device xc2vp20 Speed Grade -5	SAD16	SAD128	SAD256 (xc2vp50)	DCT	IDCT	Available Resources
Number of Slices	831	6807	13613*	4314	5436	10304
Number of Slice Flip Flops	1448	11862	23724*	7964	9876	20608
Number of 4 input LUTs	1390	11379	22757*	6832	8624	20608
Number of BRAMs:	N.A.	N.A.	N.A. *	2	2	112
Maximum Frequency [MHz]	310	310	310*	96	96	N.A.

Experimental results. We have embedded the considered CCU implementations within the MOLEN organization and carried out experiments in two stages:
Stage 1. Compile the software kernels for the original PowerPC ISA and run them on one of the PowerPC405 processors, embedded in the xc2vp20 device. The kernels have been extracted from the original application source code (the ANSI C code used for the profiling) without any further code modifications. For our experiments, we considered the same data sequences as used in the profiling phase. The PowerPC timers are initialized before a kernel is executed and are read immediately after the kernel execution has completed. Thus, the exact number of PowerPC cycles, required for the entire kernel execution can be obtained. After we derived the cycle counts for the PowerPC ISA software runs, we initiated the next stage of the experiment.
Stage 2. The kernel software code is substituted with a new piece of code to support πISA. The corresponding kernel CCU configuration is present in the reconfigurable processor. Identically to the preceding experimentation stage, we obtain the exact number of PowerPC cycles required to complete the entire kernel operation on MOLEN.

Figure 4 depicts the measured cycles obtained in the two experimentation phases. The first four chart groups present cycle counts for the original PowerPC ISA. The last chart group presents the cycle numbers, consumed by MOLEN while processing

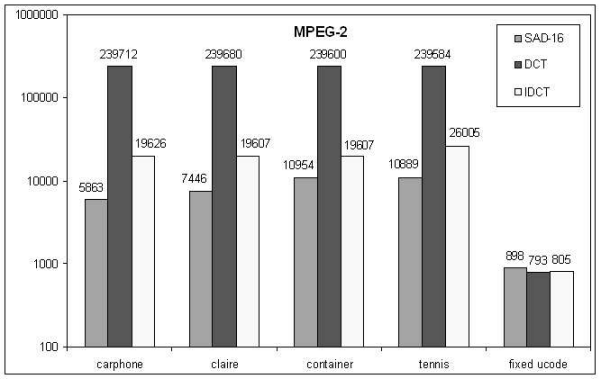

Fig. 4. Cycle numbers for kernels execution in original PowerPC ISA and fixed μ-code in πISA

the same data. In this figure, only fixed microcode implementations are depicted. In addition, we have considered both fixed and pageable microcode implementations for SAD16 and SAD128. Table 4 reports cycle numbers for executing these SAD implementations over a single macroblock. After obtaining the execution cycle numbers for each kernel both on PowerPC and MOLEN, the kernel speedup is calculated for all data sequences with respect to each CCU implementation. Table 5 presents the calculated kernel speedups.

Table 4. Cycle numbers per macroblock for different SAD implementations

	SAD16	SAD128	SAD256
fixed microcode	898	311	264
peageable microcode	914	331	284

Table 5. Speedup for Different MPEG-2 Kernels

	SAD16		SAD128		SAD256		DCT	IDCT
	fixed	pag.	fixed	pag.	fixed	pag.	fixed	fixed
carphone	6.5	6.4	18.9	17.7	22.2	20.6	302.3	24.4
claire	8.3	8.1	23.9	22.5	28.2	26.2	302.2	24.4
container	12.2	12.0	35.2	33.1	41.5	38.6	302.1	24.4
tennis	12.1	11.9	35.0	32.9	41.2	38.3	302.1	32.3

Projected application speedup. To calculate the projected speedup of the entire application, we employed the well known Amdahl's law, utilizing the following notations. Let us assume T to be the execution time of the original program (say measured in cycles) and T_{SEi} - time to execute kernel *i* in software, which we would like to speed-up in reconfigurable hardware. Assume $T_{\rho i}$ is the execution time (in πISA) for the reconfigurable implementation of kernel *i*. Assuming $a_i = \frac{T_{SEi}}{T}$ and $s_i = \frac{T_{SEi}}{T_{\rho i}}$, the speed-up of the program with respect to the reconfigurable implementation of kernel *i* is:

$$S_i = \frac{T}{T - T_{SEi} + T_{\rho i}} = \frac{1}{1 - (a_i - \frac{a_i}{s_i})} \quad (1)$$

Identically, assuming $a = \sum_i a_i$, all the kernels potential candidates for reconfigurable implementation would speed-up the program with:

$$S = \frac{T}{T - \sum_i T_{SEi} + \sum_i T_{\rho i}} = \frac{1}{1 - (a - \sum_i \frac{a_i}{s_i})}, \tag{2}$$

$$S_{max} = \lim_{\forall s_i \to \infty} S = \frac{1}{1 - a} \tag{3}$$

Where S_{max} is the theoretical maximum speed-up. Parameters a_i are the profiling results from Table 2 and parameters s_i are the results from Table 5.
The projected speedup figures for the entire MPEG-2 encoder and MPEG-2 decoder applications are reported in Table 6. It can be observed that SAD128 and SAD256 CCU

Table 6. Speedup per kernel for the Entire MPEG2

	encoder								decoder
	SAD16		SAD128		SAD256		DCT	IDCT	IDCT
	fixed	pag.	fixed	pag.	fixed	pag.	fixed	fixed	fixed
carphone	1.76	1.76	1.94	1.93	1.95	1.95	1.14	1.01	1.94
claire	1.90	1.89	2.06	2.06	2.08	2.07	1.13	1.01	1.56
container	2.07	2.06	2.20	2.20	2.21	2.21	1.12	1.01	1.63
tennis	2.22	2.22	2.40	2.39	2.41	2.41	1.10	1.01	1.65

implementations clearly outperform SAD16 due to their parallel processing organization. That is, depending on the CCU implementation, the entire application speedup can be severely affected. The projected overall speed up figures for the entire MPEG-2 encoder and MPEG-2 decoder applications are reported in Table 7. For the MPEG-2

Table 7. Overall MPEG2 Speedup

	MPEG2 encoder*		MPEG2 decoder	
	theory	impl.	theory	impl.
carphone	2.85	2.64	2.02	1.94
claire	2.99	2.80	1.60	1.56
container	3.12	2.96	1.68	1.63
tennis	3.37	3.18	1.68	1.65

* fixed μ-code SAD128 + DCT + IDCT

encoder, the simultaneous configuration of the SAD128, DCT, and IDCT operations employing fixed microcode implementations has been considered. For the MPEG-2 decoder, only the IDCT has been implemented. Columns, indicated by label ”theory” contain the theoretically achievable maximum speedup calculated with respect to Equation (3). Columns labelled by ”impl.” have been calculated employing Equation (2) and contain data for the projected speedups with respect to the considered MOLEN implementation. Results in Table 7 strongly suggest that the actual speedup of the MPEG-2 encoder and decoder obtained during our practical experimentation very closely approach the theoretically estimated maximum possible speedups.

5 Conclusions

In this paper, we presented a prototype design of the MOLEN polymorphic processor, a custom computing machine based on the co-processor architectural paradigm. The prototype was implemented on a Xilinx Virtex II Pro™ FPGA. One of the PowerPC cores embedded in the Virtex II Pro™ FPGA was used as a general purpose processor and the reconfigurable fabric operated as a reconfigurable co-processor. The paper presented hardware synthesis results for the MOLEN infrastructure and for three custom computing units implemented as reconfigurable processors, namely SAD, DCT and IDCT. The reconfigurable hardware costs of the aforementioned functions were between 8% and 53% of the utilized xc2vp20 chip, depending on the particular configuration considered. The performance of the design was evaluated by experiments. More precisely, the MPEG-2 application was accelerated very closely to its theoretical limits by implementing SAD, DCT and IDCT as reconfigurable co-processors. The MPEG-2 encoder overall speedup was in the range between 2.64 and 3.18 while the speedup of the MPEG-2 decoder varies between 1.65 and 1.94. These results proved the viability of the MOLEN concept and showed its potentials for accelerating complex real-life applications at trivial hardware costs.

References

1. F. Campi, M. Toma, A. Lodi, A. Cappelli, R. Canegallo, and R. Guerrieri. A VLIW Processor with Reconfigurable Instruction Set for Embedded Applications. In *ISSCC Digest of Technical Papers*, pp. 250–251, Feb 2003.
2. M. Gokhale and J. Stone. Napa C: Compiling for a Hybrid RISC/FPGA Architecture. In *Proc. IEEE Symp. on FCCM*, pp. 126–135, 1998.
3. S. Hauck, T. Fry, M. Hosler, and J. Kao. The Chimaera Reconfigurable Functional Unit. In *Proc. IEEE Symp. on FCCM*, pp. 87–96, 1997.
4. G. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis. Loading rm-code: Design considerations. In *Proc. Third Intl. Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS'03)*, pp 8–11, 2003.
5. G. Kuzmanov and S. Vassiliadis. Arbitrating Instructions in an $\rho\mu$-coded CCM. In *Proc. 13th Intl. Conf. FPL'03*, Springer-Verlag LNCS, vol. 2778, pp. 81–90, 2003.
6. A. L. Rosa, L. Lavagno, and C. Passerone. Hardware/Software Design Space Exploration for a Reconfigurable Processor. In *Proc. DATE 2003*, pp. 570–575, 2003.
7. S. Vassiliadis, G. N. Gaydadjiev, K. Bertels, and E. M. Panainte. The molen programming paradigm. In *Proc. Third Intl. Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS'03)*, pp. 1–7,2003.
8. S. Vassiliadis, E. Hakkennes, S. Wong, and G. Pechanek. The Sum-of-Absolute-Difference Motion Estimation Accelerator. In *Proc. 24th Euromicro Conf.*, pp. 559–566, 1998.
9. S. Vassiliadis, S. Wong, and S. Cotofana. The MOLEN $\rho\mu$-Coded Processor. In *11th Intl. Conf. FPL'01*, Springer-Verlag LNCS, vol. 2147, pp. 275–285, 2001.
10. Xilinx Corporation. *Virtex-II Pro Platform FPGA Handbook*, v.1.0, 2002.
11. A. Ye, N. Shenoy, and P. Banerjee. A C Compiler for a Processor with a Reconfigurable Functional Unit. In *ACM/SIGDA Symp. on FPGAs*, pp. 95–100, 2000.