Customized SIMD Unit Synthesis for System on Programmable Chip – A Foundation for HW/SW Partitioning with Vectorization

Muhammad Omer Cheema
UEI, ENSTA
Paris, 75739
Tel : 33(0)1 45 52 54 60
Fax : 33(0)1 45 52 83 27
e-mail : cheema@ensta.fr

Omar Hammami
UEI, ENSTA
Paris, 75739
Tel : 33(0)1 45 52 54 60
Fax : 33(0)1 45 52 83 27
e-mail :hammami@ensta.fr

Abstract— Use of Single Instruction Multiple Data (SIMD) functional units enables multimedia systems to exploit parallelism to a higher degree resulting in significant system performance improvements. While implementation of whole SIMD system functionality for an application results in wastage of area resources, we have observed that for a specific multimedia application, we only need to implement a customized SIMD unit that is a subset of whole SIMD standard implementation. Based on this study, we have proposed an extension to the traditional system design and synthesis flow by integrating a methodology of SIMD unit Synthesis. Our system synthesizes a customized SIMD unit along with an extended instruction set and generates an equivalent version of assembly code for the application using the extended instruction set. The results of area and performance obtained by experimenting over our implementation of AltiVec compatible customized SIMD units show the effectiveness of our approach.

***Index Terms*— SIMD Synthesis, HW/SW Codesign, AltiVec Architecture, Vectorization**

I. INTRODUCTION

Multimedia standards such as MPEG-1, MPEG-2, MPEG-4, MPEG-7, JPEG2000, and H.263 put challenges on both hardware architectures and software algorithms for executing different multimedia processing jobs in real-time. To meet the high computational requirements of emerging media applications, current systems use a combination of general-purpose processors accelerated with DSP (or media) processors and ASICs performing specialized computations. However, benefits offered by general-purpose processors in terms of ease of programming, higher performance growth, easier upgrade paths between generations, and cost considerations argue for increasing use of general purpose processors for media processing applications [1]. The most visible evidence of this trend has been the SIMD-style media instruction-set architecture (ISA) extensions announced for most high-performance general purpose processors (e.g., AMD's 3DNow! [2], Motorola's AltiVec [3], Intel's SSE1/SSE2 [4], Sun's VIS, HP's MAX, Compaq's MVI and MIP's MDMX).

Research work done over the study of area constraints of SIMD shows that implementation of whole SIMD units is very expensive in terms of area and energy requirements [5],[6]. On the other hand, research also indicates that multimedia applications don't use all the components of an SIMD unit and hence implementation of many parts of SIMD units can be avoided to save area and energy without affecting the speed. As a result, synthesis of customized SIMD units to optimize the system resources is suggested. Synthesis of customized SIMD units is somehow equivalent to an Application Specific Instruction Set Processor (ASIP) synthesis problem and recently, an increasing interest in this direction has been observed [7],[8],[9][10],[11],[12]. While ASIP synthesis can be considered a generalized SIMD synthesis problem, a very few methodologies to synthesize SIMD units in particular have been proposed [13],[14]. [14] uses the optimization of Control Data Flow Graphs (CDFG) of the application code for extraction of SIMD instructions. A drawback of this approach is that SIMD pattern recognition through CDFG doesn't completely exploit the possible parallelism of a program hence speedup because of SIMD usage remains very limited. [13] uses step by step SIMD instruction decomposition for a manually vectorized program to get an area efficient processor core, but it doesn't take into account the standard SIMD based systems with complex architectures hence it doesn't represent a real world scenario of a DSP application using SIMD instructions. That's why we have implemented standard AltiVec unit being used as a coprocessor with a PowerPC 405 processor to keep in mind the practical aspects of SIMD while proving the concept behind our work.

There are also some commercial tools available for synthesis of extensible processors. Commercial examples of extensible processors include HP Laboratory and STMicroelectronics' Lx [15], Altera's NIOS [16] and Tensilica's Xtensa [17]. In Altera's NIOS architecture, extensible instruction set is obtained by introducing the instructions in the already existing pipeline of the processor which increases the critical path length of the processor. Just like Xtensa, our methodology emphasizes on the use of coprocessor that extends the existing instruction set with one more advantage that we are using well known Instruction Set Architecture (ISA) based on PowerPC architecture.

0-7803-9451-8/06/$20.00

Based on the above discussion, this paper presents a methodology for the application specific synthesis of SIMD units for digital signal processing applications. Given an application program written in C or Assembly language and a set of application data, our methodology synthesizes an RTL description of an SIMD based coprocessor and the extended instruction set along with the modified assembly language program capable of running over synthesized system. As a case study, we experimented over PowerPC architecture based AltiVec [3] units and the results obtained indicate the effectiveness of our methodology. These results testify to the high potential of the SIMD computation paradigm in the synthesis of high performance and low-power application specific hardware architectures.

Rest of the paper is organized as follows: Section II gives an introduction to PowerPC/AltiVec and presents benchmarking results of multimedia applications outlining the motivation behind our work. Section III explains the System Synthesis methodology. Section IV and V describe the experiment environment and results. Section VI and VII present conclusions and future work.

II. Study of Utilization of AltiVec Units

AltiVec is a floating point and integer SIMD instruction set designed and owned by Apple Computer, IBM and Motorola (the AIM alliance), and implemented on versions of the PowerPC including Motorola's G4 and IBM's G5 processors. AltiVec is a trademark owned solely by Motorola, so the system is also referred to as Velocity Engine by Apple [18] and VMX by IBM. Fig. 1 explains the various components of an MPC 7400 system that consists of a G4 processor having an AltiVec extension.

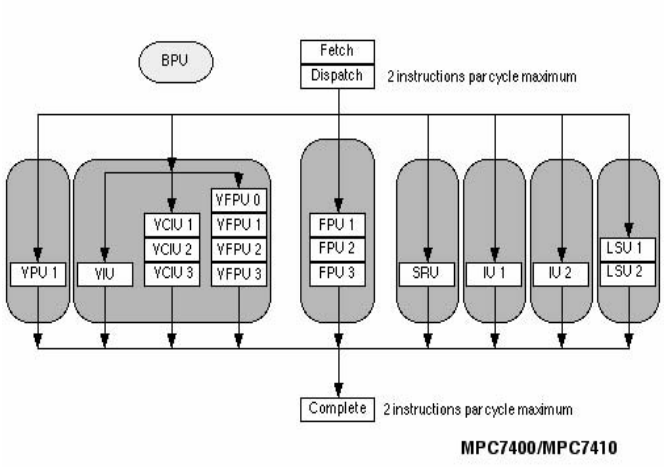

Fig.1 PowerPC G4 Architecture

Vector Permute Unit (VPU) Vector Integer Unit (VIU), Vector Complex Integer Unit (VCIU) and (Vector Floating Point Unit) VFPU are part of the AltiVec unit. Vector permute unit arranges the data to make it usable by vector instructions. VIU, VCIU and VFPU are used to execute vector integer, vector complex integer and vector floating point instructions. Other components shown in Fig.1 are part of general-purpose PowerPC and are used to execute non-SIMD instructions.

Our first experiment was to measure the utilization of these vector units for certain applications. For that we developed a few multimedia test benches for AltiVec enabled G4 system. We used profiling and simulation tools like Shark, Amber, MONster [19] and SimG4 [20] to calculate the usage of various components in the system. We used different versions of filters used in image processing keeping in mind that each of the filters had a different level of vectorization so that it can reflect realistic results on AltiVec unit usage during the application execution.

For an image of size 320x240, when applied to various versions of filter program having different vectorization level, results in Table 1 were obtained. Looking at the Table 1, we can see that even if the branch prediction and cache performances remained in reasonable limits, usage of most of the AltiVec components was very poor. As a matter of fact, for our application that dealt with integers parts only, floating point unit was never used. Looking at the statistics, we can claim that most of the SIMD resources are underutilized (or un-utilized in some cases).

TABLE 1
Statistics of G4 Components Usage for Multimedia Applications

	Filter v1	Filter v2	Filter v3	Filter v4
Instruction/Cycle	0.8615	0.8483	0.6703	0.8465
FXU1 Idle Time	53.28%	58.44%	45.46%	54.09%
FXU2 Idle Time	76.36%	70.41%	64.18%	75.89%
FPU Idle Time	100 %	100 %	100 %	100 %
VAUS IdleTime	99.27%	99.32%	100	99.25%
VAUC Idle Time	93.90%	93.23%	92.83%	93.77%
VAUF Idle Time	100 %	100 %	100 %	100 %
VPU Idle Time	100 %	91.87%	100 %	90.76%
SYS Idle Time	91.90%	92.52%	97.98%	91.74%
LSU Idle Time	56.01%	61.16%	49.73%	67.42%
DL1 Hit Rate	98.52%	98.72%	97.18%	98.36%
IL1 Hit rate	99.82%	99.84%	99.54%	99.82%
Branch Predict.	93.45%	93.45%	94.91%	93.45%

One important thing to note is that researchers in [21] also got the similar results and concluded that in the dynamic instruction stream of media workloads, 85% of the instructions are not performing computation but are load/stores, loop/branches and address generation instructions. They observed an SIMD efficiency ranging only from 1 to 12%.

Using the GCC 4.0.0 [22] and VAST [23] vectorizing tools, we vectorized various benchmarks and obtained even worse results in terms of AltiVec unit utilization due to the facts that vectorizing capabilities of existing tools are limited and also that even if an application is well vectorized, most of the components in SIMD remain underutilized as was observed in Table 1.

Based on above observation, we conclude that it is preferable to include only those components of SIMD in application specific embedded systems that are not underutilized or un-utilized to have a better area and energy consumption of a system: hence the basic motivation behind our work.

III. ADAPTIVE GENERATION OF SIMD UNITS

Our SIMD synthesis flow consists of following sub tasks:

a) Vectorization
b) Static and Dynamic Profiling
c) SIMD AltiVec module Generation
d) Real Time Execution of the Application
e) Repetition of above steps until a set of possible solutions is obtained. Best solution matching the system requirements is chosen.

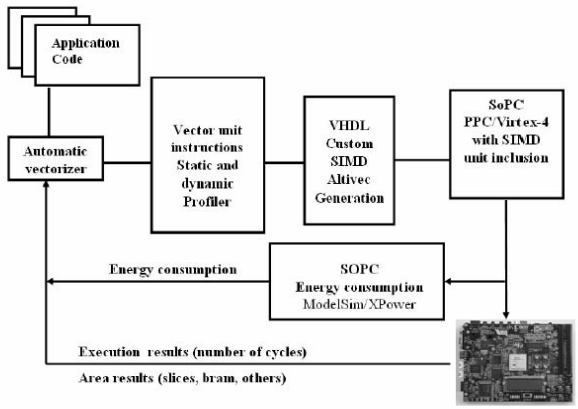

Fig. 2 System Design Flow

a. *Vectorization*

As mentioned in first section, input to the system is an application written in C or Assembly language. First step in the synthesis process is to vectorize the application. During this phase, vectorizing compiler detects the possible vectorizing options. We have defined equivalence classes of AltiVec and PowerPC instructions. Equivalence classes represent the possible replacement of an instruction with a set of instructions performing the same function. As a result, use of some instructions can be avoided which makes it possible for us to choose alternative components of AltiVec or PowerPC to test the behavior of the system for a program modified using an equivalent class. As a very simple example of the concept of equivalence classes, let's say that we have a *vadduwm* instruction that adds a vector of four elements having 32-bit size each. RTL description of the SIMD unit is implemented in a module that handles unsigned addition for byte, half word and word elements. Let's suppose that we want to generate a program version that doesn't contain *vadduwm* instruction. An obvious reason for such decision can be that the *vadduwm* is executed only a few times during the whole execution of the program while module inserted inside the system due to its inclusion adds significant amount of energy and area requirements. So we can use the concept of equivalence classes and replace this *vadduwm* instruction with four PowerPC add instructions used for addition of 32 bit unsigned elements to generate such a version. This example mentions the replacement of a vector instruction with its equivalent PowerPC instructions. There are some cases where it seems more beneficial to use another vector instruction to replace a vector instruction (i.e. multiply accumulate operation with two different operations of multiply and then accumulate for vectors). This concept of equivalence classes, when introduced in a vectorizing compiler gives a very large system design space depending on the set of vector instructions being used and their replacement methodology. Ideally, to get an optimal solution, an automatic system design exploration algorithm can be applied. Or alternatively, system can be manually tested for various vectorized versions of the program and the system configuration matching the area and speed constraints can be chosen as is done in this article. Energy based optimization has not been performed in this article although it remains an optional part of the design flow and we are in the process of developing a methodology to have good energy consumption estimates.

b. *Static Analysis and Profiling Results*

In this phase, we analyze the application and study the various aspects related to instruction set used and its usage. During this process, frequency, timing and repetition patterns of instructions are studied. This helps the system designer to capture the properties of the system and to exploit the inherent parallelism in various ways. Information obtained during this step is also helpful in automatic generation of customized AltiVec module.

c. *AltiVec Module Generation*

During this phase, the system automatically generates the VHDL description of a suitable AltiVec module that consists of only those components which are needed to execute the specific version of the program generated in step A. The modules not going to be used by program are ignored and kept out of the hardware synthesis process. System is ready to be executed in real time at the end of this step.

d. Real time execution over Virtex- 4 FPGA

In this phase, application is run over an FPGA on which customized SIMD unit is synthesized. We preferred actual execution of the application over FPGA instead of simulation to avoid the accuracy limitations of the simulation and to prove our idea in a concrete manner. Execution also speeds up the process as simulation of applications has proven to be very slow in many cases. Results of area and energy consumption and number of cycles taken by application are obtained at the end of this phase.

All of the above steps are repeated several times and results for area, energy and speed are obtained for corresponding configurations. Based on the results obtained and the system requirement, suitable SIMD system and corresponding extended instruction set is chosen for the application.

IV. Experimental Setup

In this section, we briefly explain the experimental setup for the hardware environment. We have used Xilinx Virtex-4 FX platform devices to execute the application in real time and get the execution results. Virtex-4 consists of a PowerPC 405 processor: a 32-bit IBM RISC processor at its core along with various peripheral component interfaces. Virtex-4 FPGA is the newest of Virtex FPGA [24] series and is the first FPGA that provides an option to connect a coprocessor with PowerPC processors with the help of an APU (Auxiliary Processor Unit). And this feature was the major reason for us to choose Virtex-4 to perform the experiments.

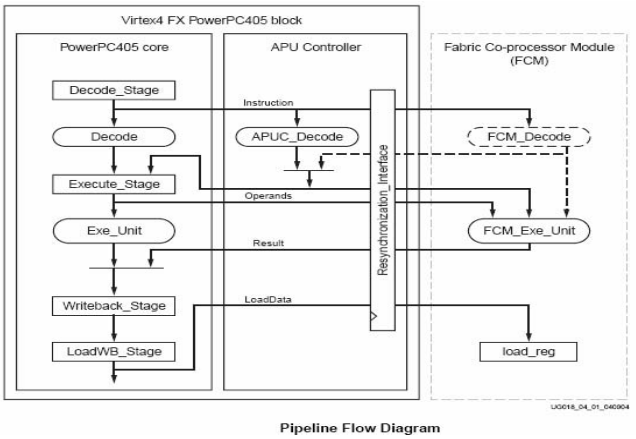

Fig.3 PowerPC with APU Interface

As shown in the Fig. 3, Virtex-4 APU allows a designer to extend the native PowerPC 405 instruction set with custom instructions for execution by an FPGA Fabric Coprocessor Module (FCM). An APU-enhanced system enables tighter integration between an application-specific function and the processor pipeline, making the APU implementation superior to, for example, a bus peripheral. When an instruction arrives, the processor and the APU decode it simultaneously. If the instruction is meant for the APU and the FCM, the APU relays it to the FCM.

The Embedded Development Kit (EDK) is a widely used tool to program Xilinx FPGAs. EDK 7.1 is the latest version and the only way to develop the Virtex-4 FPGA based APU enabled systems. EDK includes the IPs of Processor Local Bus (PLB), On-Chip Peripheral Bus (OPB), Block RAM (BRAM) controllers that were reused in our system design. (Integrated Software Environment) ISE 7.1 is used to synthesize the system and get the area requirements of the system.

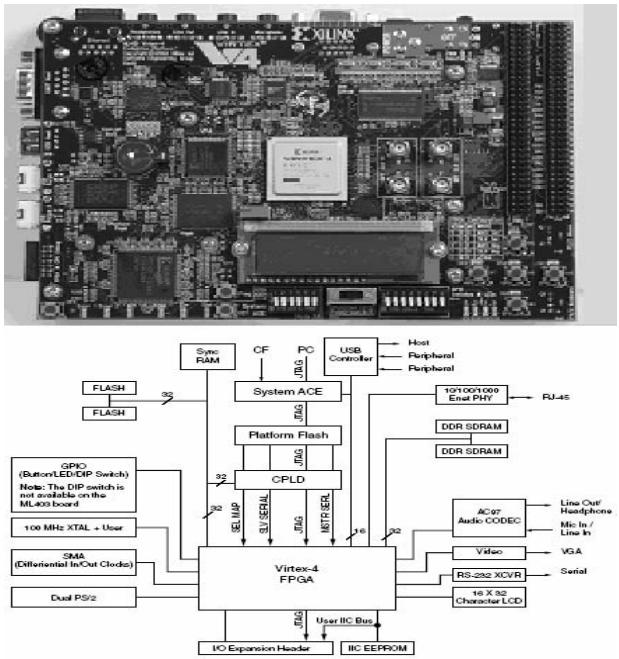

Fig. 4 Xilinx ML403 FPGA Resources

All the experiments have been performed over Xilinx ML403 [25] board that allows designers to investigate and experiment with features of the Virtex™-4 family of FPGA.

V. Evaluation Results

We tested our methodology over two sets of applications. Smaller application using lesser number of vector instructions was a matrix transpose application. It consisted of only five vector instructions being used including *lvx* and *stvx*. For a larger application, we developed a set of image processing filters which used several vector instructions.

Fig. 5 represents the area taken by various components of AltiVec on a Virtex-4 FPGA. Some components like Vector Permute Units and the modules related to shift instructions take as much as one thousand slices, which is more than 20 % of total ML403 area, while most of the modules are less expensive in terms of area requirements. An obvious reason for this fact is that the shift capabilities in AltiVec instructions are more than that of a “barrel shifter” since every block of the vector can be shifted by a different value. For standard implementation of the VPU, whole “cross bar” functionality has to be implemented to keep it compatible to standards resulting in adding a lot of RTL logic. Area might have been smaller for shift instructions if same shift value

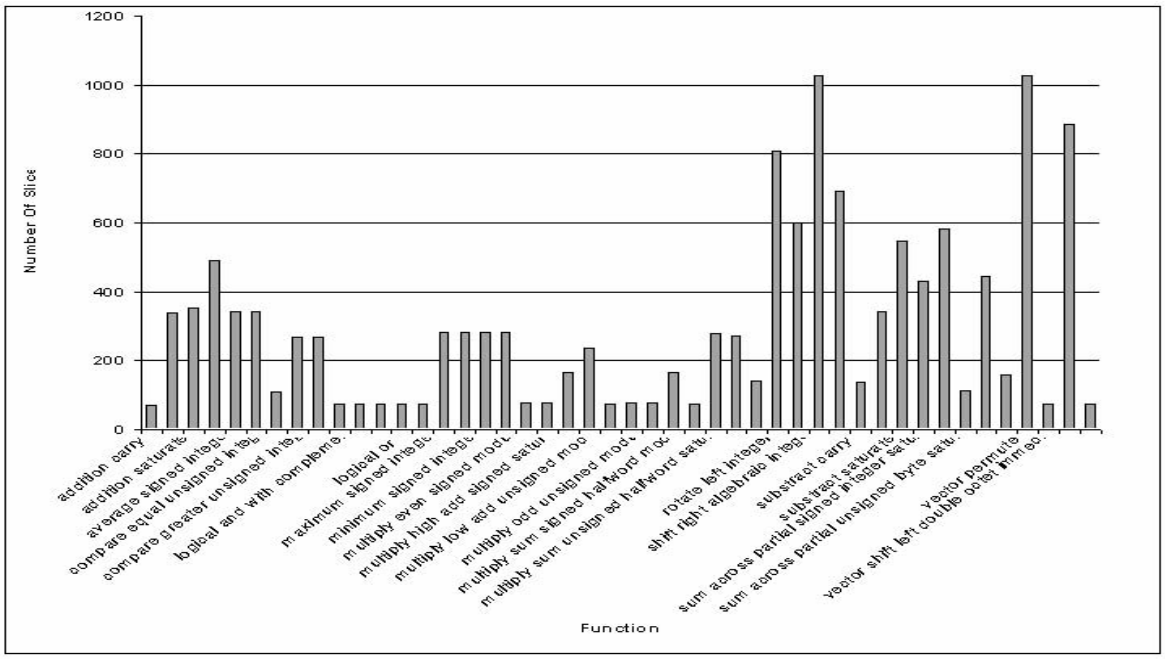

Fig. 5 Area of AltiVec Modules in Virtex-4

was used for every data component in the instruction. Similarly, the instruction with saturation takes up more area because of additional logic for implementation of saturation functionality.

Repeating the methodology mentioned in previous sections, results of area and energy consumption obtained by system synthesis and real time execution of matrix transpose application are summarized in Table 2. Results show that a speed up of up to 5.2 can be obtained with an area cost of 89% of FPGA total area. Configuration 5 is using scalar only code while other configurations use one or more vector instructions. Configuration 1 is using all possible vector instructions in the program resulting in maximum area and maximum speed up.

TABLE 2
Area vs. Speedup for Matrix Transpose Program

Config. No.	FPGA Area	Time (cycle)	Speedup over Non-SIMD Code
Config. 1	89	3 171 944	5.2
Config. 2	83	5 275 383	3.1
Config. 3	75	6 357 824	2.6
Config. 4	70	7 188 232	2.3
Config. 5	28	16534 108	0

Fig. 6 graphically represents the above table. As expected, in all the configurations, area and execution time tradeoff is clearly visible.

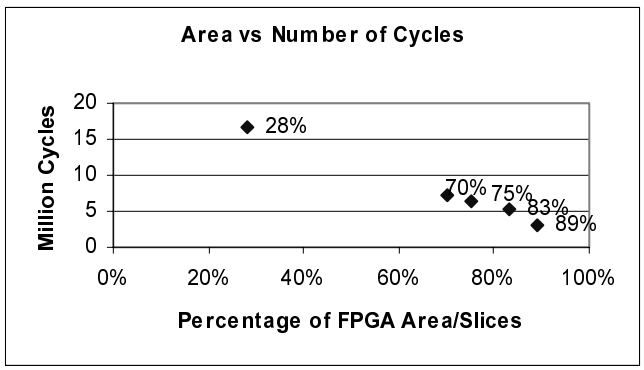

Fig. 6 Area vs. Speedup Tradeoff for Matrix Transpose Program

Similarly, various AltiVec configurations of a filter automatically generated by our customized AltiVec generation tool depending on extended instruction set being used and corresponding area and speedup results are shown in Table 3. An image of size 500x500 was used as data input for the results in Table 3. It is important to note at this point that implementation of vector register bank and vector permute unit took more than 40 percent of the area available over ML 403 board because of the reasons mentioned in the beginning of this section. Rest of the area utilization was dependent on the vectorizing compiler's decision to select/reject certain instructions in a specific SIMD configuration.

TABLE 3
Area vs. Speedup for Average Filters

Config. No.	FPGA Area	Time (cycle)	Speedup over Scalar Code
Config. 1	84%	29 812 080	2
Config. 2	86%	33 087 428	1.8
Config. 3	89%	23 853 953	2.8
Config. 4	89%	34 237 811	1.8
Config. 5	92%	12 388 586	4.9
Config. 6	78%	35 770 207	1.7
Config. 7	28%	60 810 431	0

Fig. 7 shows a tradeoff between area and speedup for the given filter application. We observe that various solutions based on the system requirements are possible. For example, if focus is on the execution speed, configuration 5 is the required solution. If area is to be minimized, among the SIMD based solutions, configuration 6 is the best option. Other configurations represent the tradeoff between these two extremes.

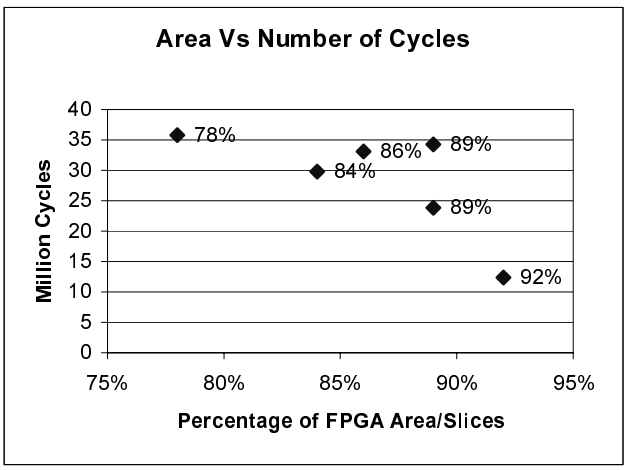

Fig. 7 Area vs. Speedup Tradeoff for Average Filters

To test the impact of system performance for different image sizes, one configuration was chosen and images of various sizes were applied to the application. Results obtained are summarized in the Fig. 8. The results show that optimal solution obtained for one image size might not be optimal for other image sizes and speed up can be lesser if images of smaller sizes are used. In ideal case, for each data size/type, system should be synthesized again to get an optimized solution.

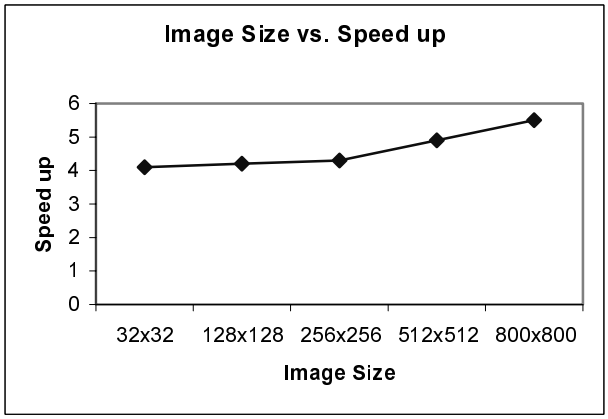

Fig. 8 Image Size vs. Speedup Tradeoff for Average Filters

Needless to say that, generally more speedup has been observed for large data sizes showing the suitability for SIMD for large data applications.

VI. Discussion and Extensions

In a broader view, this paper lays out the foundation for a HW/SW partitioning scheme, which includes vectorization. The target platform of such a scheme is described in Fig. 9. This single processor platform is composed of: (1) an IBM PPC 405 processor connected to an IBM CoreConnect infrastructure (2) peripherals (3) a custom Altivec compatible SIMD unit (4) hardware accelerators: this whole platform being the result of a HW/SW partitioning scheme. In such a platform HW/SW partitioning scheme needs to add a new dimension in the design space exploration with the inclusion of vectorization resulting in SIMD units in the system.

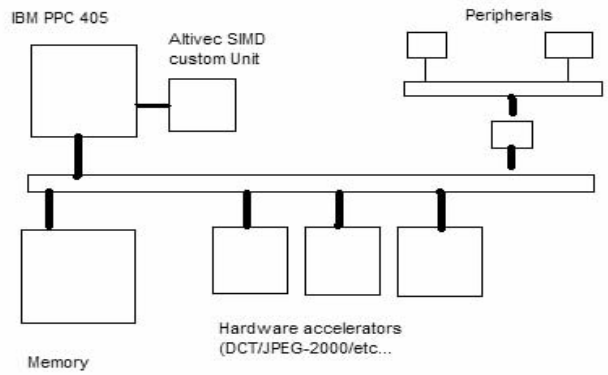

Fig. 9 Target platform with vectorization

This flow accepts as input a C application from which a call graph is extracted and profiled. Compute intensive functions having SIMD like patterns are vectorized. A design space exploration algorithm similar to [26] is applied which partitions each function in either: (1) software without vectorization (2) vectorized software with the custom associated SIMD unit (3) or pure hardware as a hardware accelerator. The resulting configuration is generated, synthesized on an FPGA platform and executed. The actual

number of cycles of the execution as well as area values resulting from the synthesis/place and route step are fed back to the DSE engine for a new exploration until the constraints are met.

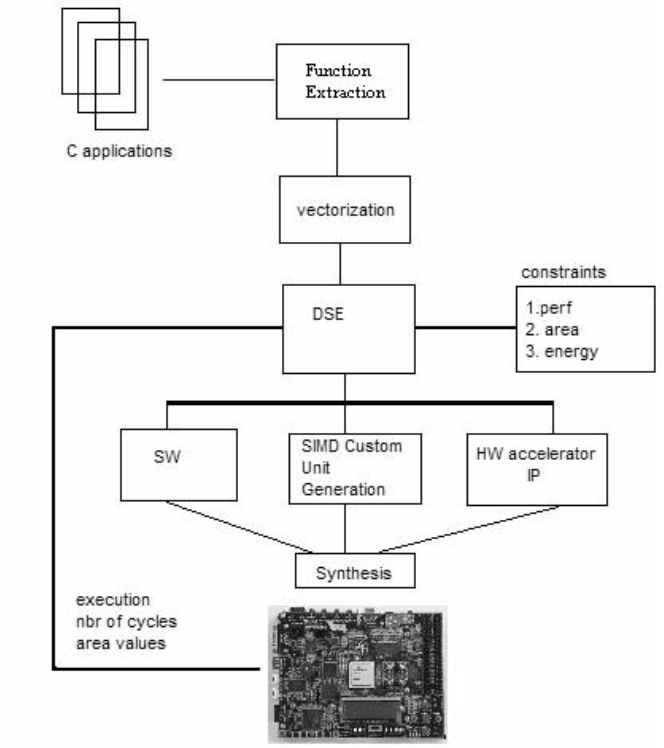

Fig. 10 HW/SW/SIMD Partitioning Flow

VII. CONCLUSIONS

In this paper, we have proposed a methodology for the synthesis of customized SIMD units. Concept of equivalence class between/among SIMD and general-purpose processor instructions has been introduced to create a system design space for synthesis of customized SIMD units. As a case study, we have used AltiVec based SIMD unit along with PowerPC405 and generated a suitable architecture for a specific image processing application along with the extended instruction set and a modified application that can execute itself over the synthesized hardware. Results of area and application execution time show that significant efficiency improvement is achieved through the use of our SIMD based synthesis methodology.

REFERENCES

[1] K. Diefendorff and P. K. Dubey. How multimedia workloads will change processor Ddesign. In IEEE Micro, pages 43–45, Sep 1997.
[2] S. Oberman et al, “AMD 3DNow! Technology and the K6-2 microprocessor”, In HOTCHIPS10, 1998.
[3] M. Phillip et al,”AltiVec technology: Accelerating media processing across the spectrum”, In HOTCHIPS10, Aug 1998.
[4] S. K. Raman, V. Pentkovski, J. Keshava,“ Implementing streaming SIMD extensions on the Pentium III processor”. In IEEE Micro, volume 20(4), pages 47–57, July-August 2000
[5] Schmookler M, Putrino M, Roth C, Sharma M, Mather A, Tyler J, Nguyen H.V, Pham M.N, Lent J “A low-power, high-speed implementation of a PowerPC microprocessor vector extension”,14th IEEE Symposium on Computer Arithmetic, p.12, 1999
[6] Linlay Gwennap, “AltiVec vectorizes PowerPC forthcoming multimedia extensions improve on MMX ” Microprocessor report, Volume 12, No. 6, May 11, 1998
[7] T. M. Conte et al., “Challenges to combining general-purpose and multimedia processors.”, In IEEE Computer, pages 33–37, Dec 1997.
[8] D. Goodwin and D. Petkov, “Automatic generation of application specific processors”, In Proc. of the 2003 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, pages 137-147, 2003.
[9] Slingerland N, Smith A.J, “Measuring the performance of multimedia instruction sets”, IEEE Transactions on Computers, Vol. 51, No. 11, November 2002 page 1317-1332
[10] P. Brisk, A. Kaplan, M. Sarrafzadeh, “Area-efficient instruction set synthesis for reconfigurable system-on-chip designs” Proceedings of Design Automation Conference, 41st Conference on (DAC'04) pp. 395-400,2004
[11] Atasu, K., Pozzi, L., and Ienne, “Automatic application-specific instruction set extensions under microarchitectural constraints.”, Design Automation Conf. (DAC), 2003.
[12] P. Ranganatha, S. Adve,N. P. Jouppi , “Performance of image and video processing with general-purpose processors and media ISA extensions”, Proceeding of 26th International Symposium on Computer Architecture, pp-124-135, 1999
[13] N. Togawa, M. Yanagisawa, T. Ohtsuki, “A Hardware/Software cosynthesis system for digital signal processor cores” IEICE Trans. Fundamentals, Vol E83-A, NO.11, November 1999
[14] V. Raghunathan, A. Raghunathan, M. B. Srivastave, M. D. Ercegovac, “High level synthesis with SIMD units”, Proceedings of the 15th International Conference on VLSI Design (VLSID.02), 2002
[15] P. Faraboschi et al, ”Lx: a technology platform for customizable VLIW embedded processing”, In ISCA, 2000.
[16] Altera. Nios embedded processor system development. http:// www.altera.com/products/ip/processors/nios/nio-index.html.
[17] R. E. Gonzalez, “Xtensa: A configurable and extensible processor.” IEEE Micro, 20(2), 2000.
[18] Velocity Engine: http://developer.apple.com/hardware/ve/index.html
[19] CHUD Tools(Amber, Shark, MONster): http://developer.apple.com/tools/performance/overview.html
[20] SIMG4:http://developer.apple.com/Developer/Documentation/CHUD/SimG4_Users_Guide.pdf
[21] Talla D, John L, Burger D “Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancement” IEEE Transactions on Computers, VOL. 52, NO. 8, AUGUST 2003 page 1015 – 1031
[22] Dorit Nicholas, “Autovectorization in GCC”, GCC Developers’ Summit, pp 105-117, 2004
[23] VAST Code Optimizer, Available: http://www.crescentbaysoftware.com/vast_altivec.html
[24] Virtex-4 FPGA Handbook August 2004
[25] ML 40x Evaluation Platform User Guide, UG080 (v2.0) P/N 0402337 February 28, 2005, 2004-2005 Xilinx, Inc.
[26] K. Ghali, O Hammami, ”Multiobjective design of embedded processors on FPGA platform”, ICDCS 2004