A Methodology for Fast System Level Synthesis of Image Processing Chains

Muhammad Omer Cheema */**, Omar Hammami *, Lionel Lacassagne **, Alain Merigot **
* ENSTA, 32 Boulevard Victor, 75739 Paris, France. {cheema,hammami}@ensta.fr
** IEF, University of Paris Sud, 91405 Orsay, France. {lionel.lacassagne,alain.merigot}@ief.u-psud.fr

***Abstract*— Domain specific system synthesis methodologies can lead to better performance and more efficient solutions to system modeling and synthesis. While generalized system synthesis approaches are not optimal in terms of design productivity, domain specific approaches provide a balance between generality and optimality. In this paper, we propose a framework that facilitates system synthesis for applications from image processing domain. We advocate high level simulation techniques, a semi-automated domain specific framework and component based image processing software design to increase the design productivity while keeping the framework general enough to be utilizable for a range of image processing applications. We show the effectiveness of our approach by synthesizing a couple of image processing chains using our approach.**

***Index Terms*—System Level Synthesis, Domain specific application synthesis, Image processing systems, HW/SW Partitioning, Transaction Level modeling**

I. INTRODUCTION

With the evolution of new computation intensive applications, need to have methodologies to optimally use the available computational resources is becoming more and more important. Among the new generations of applications, real time image and video processing applications are the most computation intensive applications. There are a wide variety of image processing applications. Roughly, we can divide these applications in three sub-domains according to the size of images being processed. Image processing of small sized images (roughly 256x256 pixels) is normally used in biomedical applications like artificial retina and military applications like UAVs and guided missiles etc. Medium sized images (640x480 to 1920x1080 pixels) are used in traditional VGA, NTSC, PAL and more recently HDTV standards while satellite imagery requires large sized images typically of the size of (6000x6000) pixels or so. With this vast range of image processing applications available, there is a need to have the methodologies that dedicated to the design of embedded real time systems involving modern image processing techniques. These methodologies should allow us meet the real time performance constraints while offering shorter time to develop and market, lesser area requirements and lesser energy consumption for the designed System on Chip (SoC).

There are different factors that prolong the system design and synthesis time and hence result in increase of the time to market for a product. Firstly, traditional design methodologies [] often start the system design from the scratch. System is implemented in a higher abstraction level such as algorithmic or heterogeneous models of computation. Then a series of refinement stages is applied to the system and after each refinement stage, system is described completely in more detailed manner than its higher abstraction layer. Starting the system synthesis from scratch and step by step refinement prolongs the system synthesis process adding significant amount in time to market for the product. Secondly, there is a trend of more increase in software complexity than the increase in hardware complexity in modern embedded systems. Embedded software take a significant time in project life cycle because of another fact that SW development starts after HW models of the system are available. Thirdly, simulation at lower abstraction layers is very slow and hence design space exploration becomes impossible for realistic applications with reasonable sizes of datasets. In this paper, we propose a system synthesis methodology that deals with above three problems. As a solution to the issue of generalized design ands synthesis methodologies, we propose the approach of domain specific system design methodology. We have built a framework for fast synthesis of image processing applications and we conclude that image processing applications often have very similar properties and it becomes easier to propose a design flow that is suitable to image processing applications in general and significantly reduces the system synthesis time. Problem of software synthesis is resolved by supporting an approach of components based software designs i.e. image processing chains development. Lastly, simulation times are reduced by modeling and simulating the system at higher layers of abstraction. We use transaction level modeling and it has been shown that TLM accelerates the simulation while maintaining a high degree of accuracy. We have applied our approach on many applications and our results show the robustness of our approach.

Rest of the paper is organized as follows: Section II presents related work. Section III explains our System design methodology. Section IV and V describe the experiment environment and results. Section VI presents conclusions and future work.

II. RELATED WORK

There is a little work done over domain specific system synthesis. Most of the existing design and synthesis methodologies are targeted for general system synthesis approach. These methodologies differ from each other primarily based on the nature of target applications and their performance requirements and choice of models of computations. Vulcan [5][6], for example, uses HardwareC to model hardware and C to model software and tries to reduce

hardware costs by moving functions from hardware to software as long as the performance constraints can be satisfied. Finally the resulting partition serves as input to high-level synthesis and software compilation tools. COSYMA (CO-SYnthesis for eMbedded Architectures) [7] was developed about the same time as Vulcan. In contrast to Vulcan, co-synthesis starts from a configuration, where all functions are implemented in software. The advantage with this approach is that the system may include functions that cannot be implemented in hardware, such as dynamic data structures. The POLIS [8] system is designed for control-dominated systems, where the target architecture consists of a micro-controller and ASICs. The SpecC system-level design methodology [9] [10] follows a top-down approach and starts with the development of a *specification model* expressed in the language SpecC. The MESCAL project [11] was recently formulated in order to "develop the methodologies, tools, and appropriate algorithms to support the efficient development of fully programmable, platform-based designs for specific application domains" [12]. A goal is to develop a platform that can be used efficiently for various applications inside the same application domain. PeaCE is a hardware/software co-design methodology that uses Ptolemy as its underlying synthesis system and has been found quite useful for DSP based application. However, these generalized system design methodologies are not optimal especially in terms of design productivity.

On the other hand, domain specific languages (DSLs) are considered to be an alternative to sub-optimal design languages supported by generalized system design methodologies. A domain specific language (DSL) is a programming language tailored for a particular application domain. An effective DSL enables development of a complete program or design for a domain quickly and effectively. A fundamental requirement for an effective DSL is capturing precisely the semantics of the application domain. Common examples of DSLs include Matlab for signal processing, HTML for document markup, Click [] for Networking application and OpenGL for 3D graphics. Potentially, there are many advantages to using DSLs, the most fundamental being that programs are generally easier to write, reason about and modify compared to using general purpose languages (such as Verilog and C). Typically, DSLs will be at a higher abstraction level than general-purpose languages and used by domain experts. However, the single most inhibiting factor against using DSLs is the significant initial cost related to the infrastructure required to support a DSL. For example, transforming programs in DSLs such as Matlab onto a hardware description language such as Verilog, VHDL or system description languages such as SpecC and SystemC requires significant effort and tool support. To deal this problem, in this paper, we propose a methodology that uses languages used by generalized system design approaches to avoid the manual transformation process from Domain specific languages to System Design Languages. On the other hand, our proposed framework based on Platform based design [] approach makes it possible to avoid the drawback of sub-optimality of generalized system design flows.

Transaction level modeling based on System level design languages has proven to be a fast and efficient way of system design []. It has been shown that simulation at this level is much faster [| than Register transfer level and makes it possible for us to explore the system design space for HW/SW partitioning and parameterization. Lastly, component based software design [][][] is a fast way to model image processing application. Image processing chains modeling has been classically used for fast software design and synthesis of image processing applications. In this paper, we propose the use of component based software development and transaction level modeling to further accelerate the process of system synthesis and hence improving the design productivity resulting in shorter time to markets.

III. System Synthesis Methodology

Our proposed system synthesis methodology consists of following subtasks:

a) Image Processing Chain (IPC) development
b) Hardware Resource/Performance Estimation
c) Automatic HW/SW Partitioning
d) Parameterization

a. IPC development

Our image processing system synthesis starts from application description in the form of an image processing chain. A sample chain is shown in Fig [] describing a Harris corner detectors normally used for point of interest detection in real time systems just after data capture. Each node in the chain represents some image processing operator which is implemented using library function according to the recommended coding style as used by numerical recipes []. Starting the system synthesis this way assures rapid development of the initial software. Keeping in mind that software development takes a significant time in current system design approaches; our approach saves a lot of time by avoiding software development starting from scratch.

Fig: Harris Corner Detector Chain

b. Hardware Resource/Performance Estimation

In the next step of our system design approach, Area and Energy estimates are obtained for the operators implemented in the image processing chain. At SystemC behavioral level, the tools for estimating area and energy consumption have recently been showing their progress in the EDA industry [][][]. We use Celoxica's agility compiler for Area estimation in our case but our approach is valid for any behavioral level synthesis tool in the market. At this point, one might argue that translating C code to systemC code for viewing synthesis results might be time consuming and cumbersome. This is important to mention here that restricting ourselves to image processing domain makes the module description of various operators very similar and hence manually transforming the C code to systemC code is not that time consuming. Secondly, as we advocate the fast chain development through libraries containing image processing operators, similar libraries can also be developed for equivalent systemC image processing operators which will be reusable over a range of projects hence considerably shortening the Hardware (HW) development times as well. At the end of this step, we have speed, area and energy consumption estimates for all the components of the image processing chain to be synthesized. This information is stored in a database and is used during HW/SW partitioning done in the next step.

Another important thing to be noted is that HW synthesis is also a multi-objective optimization problem. Previously, [][] have worked over efficient HW synthesis from systemC and shown that for a given SystemC description, various HW configurations can be generated varying in area, energy and clock speeds. Then the most suitable configuration out of the set of pareto optimal configurations can be used in the rest of the synthesis methodology. Right now, we don't consider this HW design space exploration for optimal area/energy ands speed constraints but in our future work, we plan to introduce this multi-objective optimization problem in our synthesis flow as well.

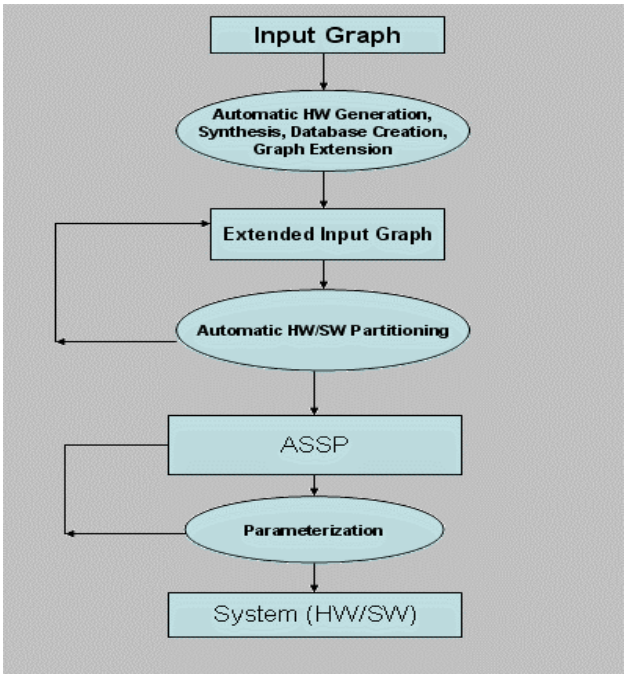

Fig. **System Design Flow**

c. Automatic HW/SW Partitioning

During the third phase, automatic partitioning of image processing application is done. An image processing chain consisting of n nodes has n^2 possibilities for system's partitioning into Hardware and Software. Our tool based on this framework proposes a methodology to automatically check these n^2 possibilities and give us the performance results for each of the configuration.

For automatic generation of a specific configuration, our tool inspects the functionality to be sent to hardware, and based on the information converts a SW computation function to a data transferring function between general purpose processor and hardware accelerator. This is done simply by reading the function definition and inspecting the input, output image parameters along with their heights and widths. After reading the parameters, body of the software function is removed and replaced with data transfer operations to send and receive images to hardware module. (See Fig []). This way, we make sure that software for a specific configuration is updated with minimum possible changes and there remains no need to debug and verify its functionality because of cleanliness and simplicity of our approach.

On the other hand, hardware accelerator is embedded into a generic module that communicates with the software running over the general purpose processor (GPP) to receive/send images from processor to the hardware accelerator. On the hardware side, as mentioned above, the communication interface (HW/SW interface) receives the data along with the information about number of images, their height and widths and the information about output images. The C code of the operator being implemented in HW is copied from SW to a function inside the HW accelerator module and estimated computation times

calculated in step 2 where an operator was actually synthesized in behavioral systemC are passed as approximates delays in the top level module combining all the components in the system. Lastly, scheduling of various operators in the chain is done to make sure that data communication overhead is reduced to achieve maximum speedup for a configuration. Hence, using our approach we can automatically shift an operator implemented in software to a hardware giving us realistic performance estimates without a need to change the HW/SW interface for each configuration.

Fig: Transforming SW Computation into Communication Interface

There are generally two cases of system speedup by introduction of hardware accelerators. In first case, parallelism inherent in the system is exploited by transferring parallel operators on other execution elements in the system. This way, we can execute all possible parallel operators in the system in parallel by adding additional Hardware accelerator for each parallel operation. Second case of system speedup comes from sequential execution. Hardware implementation of a sequential operation makes the system run faster because of availability of dedicated hardware. As a rule of thumb, in both the cases, computation time on the hardware and communication time taken by data transfers should be lesser the computation time on the software side to achieve any speedup and to justify the use of dedicated hardware accelerators consuming more design effort and area and energy costs, the sum of computation and communication time should be significantly larger than pure software computation. That means that best candidates for hardware implementation are those functions which are very computation intensive and don't require too much communication between software and hardware parts. Not to mention at this point that operations exploiting parallelism are often more suitable than the operations which are implemented in HW for sequential speedup.

d. Parameterization

In the last step of image processing chain synthesis flow, we perform the parameterization of the system. At this stage, our problem becomes equivalent to (Application Specific Standard Products) ASSP parameterization. In ASSP, hardware component of the system is fixed; hence only tuning of some soft parameters is performed for these platforms to improve the application performance and resource usage. Examples of such soft parameters include interrupt and arbitration priorities. Further parameters associated with more detailed aspects of the behavior of individual system IPs may also be available. Although, a lot of work has been done on automatic parameterization of the system [][]. Some researchers use genetic algorithms to deal with the problem of parameterization. For the time being, we deal with the problem manually instead of relying on a design space exploration algorithm and our approach is to start tuning the system with the maximum resources available and keep on cutting down the resource availability until the system performance remains well within the limits and bringing down the value of a parameter doesn't dramatically effect system performance. However, in future we plan to tackle this parameterization problem using automatic multi-objective optimization techniques as mentioned above.

IV. EXPERIMENTAL SETUP

We have tested over approach using IBM's PowerPC 405 Evaluation Kit (PEK) [] that allows designers to evaluate, build, and verify SoC designs using Transaction level modeling. General architecture of our target system is shown in Fig [2]. Our target is to synthesize a system based on a general purpose processor (in our case, IBM PowerPC 405) and extended with the help of suitable hardware accelerators to improve the system performance significantly. A gcc based cross compiler for PowerPC405 was used to compile the software while systemC compiler was used to compiler hardware modules. We used Agility compiler v 1.1 [] to synthesize behavioral description of the HW modules to get area estimations.

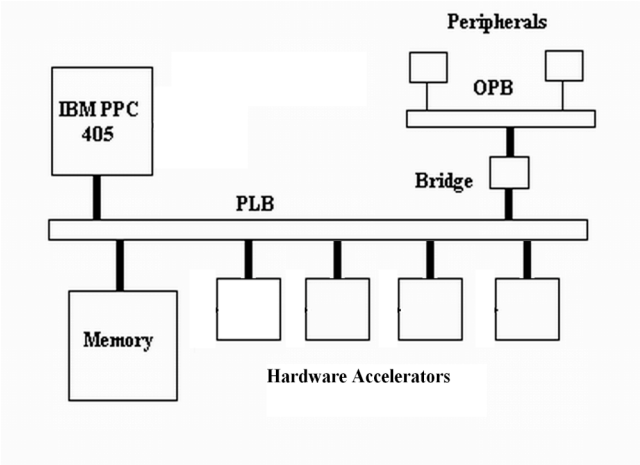

Fig. Diagram for System Built using IBM TLM

V. EVALUATION RESULTS

We tested our methodology over two sets of applications: A Chain of filters and an application for point of interest detection normally used in drones, automatic robots and guided missiles for image processing after data capture.

For the first application, we had three components in our chain: median, conservative and average filters. We have implemented these filters in hardware and synthesized them to get the results of their speed and area requirements for Virtex 4 FPGA []. The synthesis results are shown in Table 1. It is important to mention here that there is a large design space for hardware synthesis as well depending on the way modules are written in SystemC and depending on the optimizations applied for synthesis results. Optimizing the hardware implementation for area results in larger critical path and more number of cycles required to process an image and vice verse. The values given in the table only present one instance from that large design space. It is understandable that exploration of hardware design space might result in more optimized system.

For our simulation, our general purpose processor (PowerPC 405) was running at 333 MHz and we can see that the frequencies (1/critical path) obtained of synthesized hardware accelerators are quite low. To make the results comparable, we convert all the times in terms of number of cycles elapsed at PowePC 405 during the execution of an operation at hardware side and the last column represents the value which is obtained by the formula:

PowerPC cycles elapsed during computation on hardware accelerator = Number of HW accelerator cycles required for function execution*PowerPC Frequency/ HW accelerator frequency

Table 1: Synthesis Results for Filters

Module Name	Area			Critical Path (ns)	Synthesis Freq. (MHz)	Avg. Cycles Taken	PPC Cycles Elapsed
	LUT	FF	Other				
Average	22669	4144	665	65.05	15.37	8196	177571
Median	26172	4904	882	74.24	13.47	16390	405187
Conser.	58924	4188	296	66.06	15.14	8006	176089

Table 2: Various configurations and Speed ups for Filters

Config. No.	HW Parts	Time (cycle)	Area Increase	Speedup Over Software Version
1	Median	3897000		
2	Average	7202000		
3	Conservative	6630000		
4	Median + Average	3493000		
5	Median+Conservative	2921000		
6	Average+ Conservative	6028000		
7	Software Version	7248000	0	0

.

Table 3 Synthesis Results for PoI Operators

Module Name	Area			Critical Path (ns)	Synthesis Freq. (MHz)		CPU Cycles Taken
	LUT	FF	Other				
Matrix_mul	28953	16471	806	51.35 ns			
Sobel	55270	12375	466	53.28 ns			
Gaussian	40367	8247	334	59.84			
Recombination							

Table 4 Various configurations and Speed ups for Point of Interest Detection

Config. No.	HW Parts	Time (cycle)	Area Increase	Speedup Over Software Version
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

.

Fig: Area Vs Speedup

VI. CONCLUSIONS

In this paper, we have proposed a methodology to synthesize an image processing chain within very short times. This methodology emphasizes on components based software design and high level (TLM) modeling and simulation. Our proposed framework/toolset automates the process of system

design by offering generic HW/SW interfaces and a methodology to automatically shift SW functionality into hardware hence automatically generating a desired configuration. This enables us to automatically explorate the HW/SW codesign space. With the help of two image processing chains, we have shown the effectiveness of our system level synthesis approach.

Future Work Here…

REFERENCES

[1] http://www-128.ibm.com/developerworks/power/library/pa-pek/

[2] R. K. Gupta. Co-Synthesis of Hardware and Software for Digital Embedded Systems. Kluwer Academic Publishers, 1995.

[3] R. K. Gupta and G. D. Micheli. Hardware-software co synthesis for digital systems. IEEE Design & Test of Computers, 10(3):29–41, September 1993.

[4] R. Ernst, J. Henkel, and T. Brenner. Hardware-software co synthesis from microcontrollers. IEEE Design & Test of Computers, 10(4):64–75, December 1993.

[5] F. Balarin, M. Chiodo, P. Giusti, H. Hsieh, A. Jurescka, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara. Hardware-Software Co-Design of Embedded Systems: The Polis Approach. Kluwer Academic Publishers, 1997.

[6] D. D. Gajski, J. Zhu, R. D¨omer, A. Gerstlauer, and S. Zhao. Spec C: Specification Language and Methodology. Kluwer Academic Publishers, 2000.

[7] R. D¨omer, D. Gajski, and A. Gerstlauer. SpecC methodology for high-level modeling. In 9th IEEE/DATC Electronic Design Processes Workshop, Monterey, California, April 2002.

[8] A. Mihal, C. Kulkarni, M. Moskewicz, M. tsai, N. Shah, S. Weber, Y. Jin, K. Keutzer, C. Sauer, K. Vissers, and S. Malik. Developing architectural platforms: A disciplined approach. IEEE Design & Test of Computers, 19(6):6–16, November-December 2002.

[9]

1. M L Vallejo, J C Lopez, "On <he hardware-software panitioning prohlcm:
System Modeling and pmitioning lechniques", ACM TODAES. V-8.2003

K Ben Chehida. M Auguin, "HWISW **partitioning** approach **for** rcconfigurvhle **System** design", CASES 2002

A Kdavade. E Lee. "A global **cnticalityLoca1** Phase **Dnven** rlgonthm **for** the **Constrained** HardwardSoftwvre panilioning problem". CODES 1994

K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, “System Level Design: Orthogonalization of Concerns and Platform-Based Design”, In *IEEE trans on Computer-Aided Design*, Vol. 19, No. 12, December 2000.

C. Kulkarni, G. Brebner, G. Schelle: Mapping a Domain Specific Language to a Platform FPGA, *DAC*, 2004

E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, “The Click modular router,” ACM Transactions on Computer Systems, 18(3):{263-297}, Aug 2000.