C++ based System Synthesis of Real-Time Video Processing Systems targeting FPGA Implementation

Najeem Lawal, Mattias O'Nils, Benny Thörnberg

Mid Sweden University
Dept. of Information Technology & Media
SE-851 70 Sundsvall, SWEDEN
{mattias.onils, benny.thornberg, najeem.lawal}@miun.se

Abstract

Implementing real-time video processing systems put high requirements on computation and memory performance. FPGAs have proven to be effective implementation architecture for these systems. However, the hardware based design flow for FPGAs make the implementation task complex. The system synthesis tool presented in this paper reduces this design complexity. The synthesis is done from a SystemC based coarse grain data flow graph that captures the video processing system. The data flow graph is optimized and mapped onto an FPGA. The results from real-life video processing systems clearly show that the presented tool produces effective implementations.

1. Introduction

Video processing is gaining importance in such areas as process surveillance, communication and security systems. Often the video processing in these systems are done in real-time. In real-time video processing systems (RTVPS) huge amounts of information are processed in real-time. Memory accesses are the main bottle neck in these systems, which make the optimization of memory structures and memory access the key design challenge in achieving cost effective implementations for embedded applications [1].

Smart camera is a term normally used for a video camera with built-in computing power [2]. Smart cameras are, for instance, used in robotic vision and the information leaving the camera can, for example, be statistics based on various objects [3]. The most important idea behind the smart camera architecture is to collect video data and then analyze, interpret and reduce the information before it leaves the camera. To ensure real-time processing, the computation architecture used in the camera traditionally are VLIW processor and more recently it has been shown that programmable logic (FPGAs) are more efficient in implementing RTVPS systems [4][5]. Based on this fact, we have selected FPGA as the target architecture for the design environment presented in this paper.

The pre-processing parts of an RTVPS are usually neighborhood oriented. Examples of such 2-D operations are convolution, histogram, spatial and gray-level transforms, erosion, dilation and component labeling [6]. Consequently, spatio-temporal RTVPS will operate on a 3-D neighborhood, which will also increase data storage and transfer intensity. Examples of 3-D operations are optical flow calculations and scene change detection [7]. Figure 1 depicts an example of a 3-by-3-pixel spatial neighborhood. The neighborhood slides over the video operation's input frame in a progressive scan order [7]. One output pixel is calculated at every neighborhood position. Consequently, the data flow dependencies are regular, meaning that they are the same for every pixel position, except for the frame boundaries.

There are several attempts to increase the abstraction levels of the design entry with the target to shorten the design time. This has led to many propositions for implementing hardware from high level languages. These include C/C++ [8]-[13], Java [14], [15] and MATLAB [16], [17]. Familiarity with C and its variants has led to focus on synthesizing hardware from C. Since C modules can be compiled into object codes for several architectures, compiling these object codes into hardware is seen as an efficient way of hardware synthesis from system level designs.

1-4244-0910-1/07/$20.00 ©2007 IEEE.

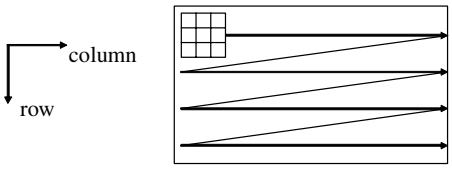

Figure 1. A 2-D pixel neighborhood slides over the input frame in a progressive scan order.

De Micheli [11] summarized the major research contribution in the use of C/C++ for hardware modeling and synthesis while Edwards [12] provided in details, challenges to hardware synthesis from C-based languages. It was observed in [12] that the studied approaches generate inefficient hardware due to difficulties in specifying or inferring concurrency, time, type and communication in C and its variants. Modeling languages like SystemC [9] and HardwareC [18] have been optimized to efficiently overcome some of these shortcomings (for example, both handle concurrencies through process-level parallelism) and are often employed to capture the system behavior in the form of executable specifications. The executable specification provides the possibility of design exploration (choosing among different algorithms and resources), system functionality partitioning (choosing between software and hardware), and memory requirements and state transitions. These specifications can be converted into RTL design manually or automatically using CAD tools like the Synopsis C2HDL [19] (creates VHDL and Verilog modules from multi-module level hierarchy in C and also provides HDL simulations).

Current approaches to C/C++ based system synthesis or any other synthesis environment do not efficiently make use of the FPGA architecture especially the memory sub-systems for real-time video processing systems. This is due to the manner in which memories are currently being instantiated in FPGAs. In this paper, we present a system synthesis tool for implementing RTVPS with multiple neighborhood oriented filters targeting FPGAs.

The tool takes advantage of our already developed memory modeling tool IMEM [23], memory allocation [21], boundary conditions management tool [24] and behavioral simulation platform. The synthesis process explicitly separates the modeling and implementation of memory requirements and behavior of the filter functions. In this paper we show that real-time video processing systems can be synthesized from C/C++ or SystemC codes to FPGA implementation. The approach supports verification through simulation of both the C/C++ and VHDL modules of the filter with real video signal to ensure that the behavioral specifications of the filter are satisfied.

2. Conceptual model – IMEM

The video system is captured using a coarse grained synchronous dataflow graph called IMEM, see Figure 2A. IMEM means Interface and MEmory Model and is build on top of the SystemC modeling library [23]. Each node in the IMEM dataflow graph captures both the abstract video interface and the memory model as shown in Figure 2B. The model is stated to be conceptual since it explicitly captures the data dependencies. The memory model is a description of the neighborhood of pixels that the task operates on, Figure 3A shows an example of such a neighborhood. Additionally, each node consists of a description of the task's functional behavior. The task does not include any data dependency or timing related to the dataflow, just an un-timed C++ description of the relation between input and output pixels.

The IMEM model can be verified through simulation at system level using the SystemC simulator. Source and sink nodes can easily be added to the model, which produce and consume video data, respectively. When verified the supporting tool set can extract the IMEM model from the system to the synthesis tools, which is further described in Section 4.

3. Target architecture

The target architecture is FPGA having on-chip Block RAMs. These RAMs are required as cache memory for the streaming data oriented application that we target. Resource reuse is not possible between processes but only within individual tasks (as shown in Figure 2).

The architecture in Figure 4 handles data storage and boundaries conditions for the spatial pixel neighborhood shown in Figure 3.

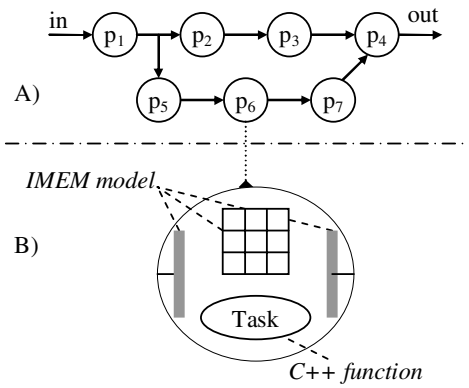

Figure 2. IMEM model of a video processing system.

In Figure 4, the video/image processing (VIP) algorithm is the neighborhood oriented filter. It is connected to the memory architecture through the port interfaces for all the pixels data required in the neighborhood. The sliding window controller *SLWC* monitors the centre pixel in a spatial neighborhood and using the position information provides valid data for all the pixels in the spatial neighborhood through the *Line buffers, Window ctrl* and *Pixel Switch*. The *Line buffers* in Figure 3B are required to buffer image data in order to create the neighborhood shown in Figure 3A. They are implemented in hardware through the line buffer modules described in details in Section 5.

Window control (*Window ctrl)* provides control signals used by the *Pixel switch* to build a spatial neighborhood around the current pixel. *Window ctrl* is implemented in hardware such that only one copy is instantiated and used to control all *Pixel Switch* modules instantiated for all the spatial neighborhoods in a VIP algorithm involving more than one frames. The *Pixel switch* replaces all pixels in a spatial neighborhood affected by boundary condition using predefined default values if the centre pixel is at the image boundary. The *output sync* is optional and is required to realign the pixels with other video signals where time synchronized data and control signal outputs are expected. This is because the neighborhood's output pixel is usually skewed with respect to the input video control signals by an amount depending of the neighborhood size and the number of pipeline stages.

The architecture in Figure 5 eliminates the optional *output sync* and is suitable for a system with many neighborhoods and high demands for Block RAMs. A central state machine is employed to maintain the data and control signal synchronization for all the neighborhoods.

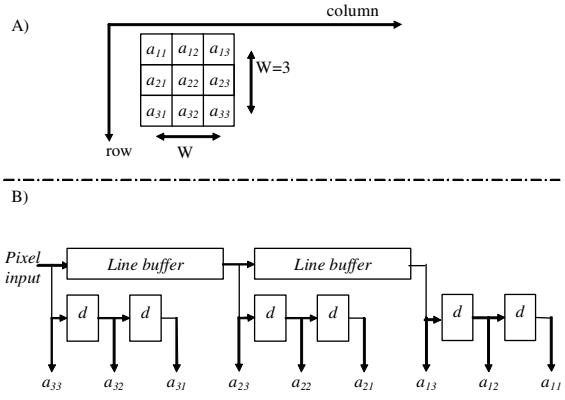

Figure 3. A: Spatio-temporal neighborhood of pixels. B: Memory architecture for a single image processing operation.

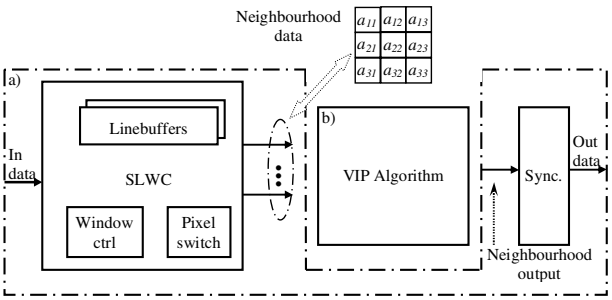

Figure 4. Boundary conditions implementation architecture.

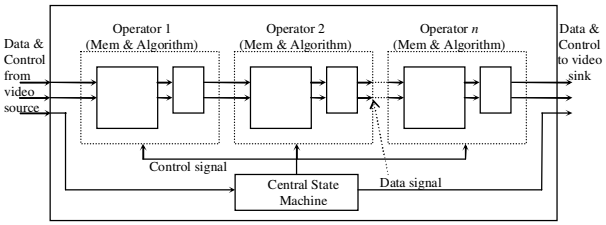

Figure 5. Neighborhood oriented system.

4. System synthesis

The IMEM synthesis workflow depicted in Figure 6 demonstrates how our research on modeling and high level synthesis fits into an implementation trajectory. This workflow is defined at six different levels along the left-hand axis. The video-processing algorithm is developed and simulated using IMEM at level 1. This executable model can then be verified through functional simulation. Data dependency information, frame sizes, composition of the 3-dimensional neighborhoods and color space models are exported into an interface and memory model at level 2. Hence at it is at this level that the memory requirements of a RTVPS are separated from the behavioral C++ description of the RTVPS filters (as shown in Figure 2B). The interface between the memory and filters of each operator is also defined at this level. The model exported in level 2 is the input to the memory synthesis process at level 3. This is where memory estimation, memory hierarchy optimization, memory allocation and address generation is performed.

At level 3, the SystemC functional description together with the interface template generated from the memory model is synthesized using a SystemC based commercial high-level synthesis tool, in this paper Agility from Celoxica. The VHDL code from both the functional part and the optimized interface and memory model is integrated at level 4 and synthesized at level 5. Hence the components separated at level 2 are integrated at level 5. Hardware simulation and compilation are also carried out.

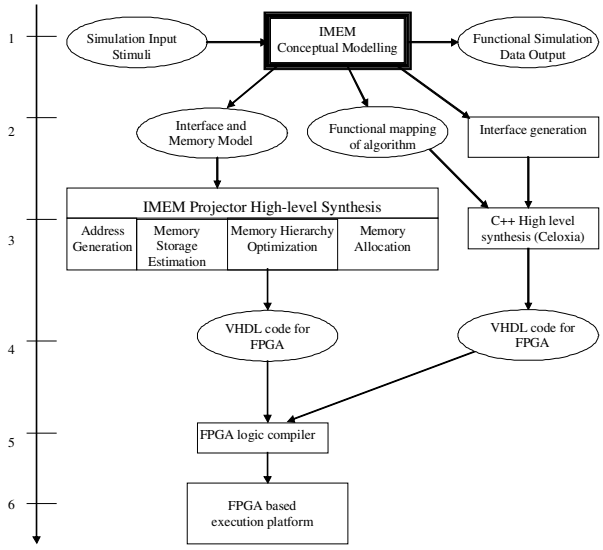

Figure 6. System synthesis workflow.

5. Memory synthesis

The memory synthesis tool creates all necessary memory and control functionality needed for a functional spatio-temporal RTVPS. The required memory architecture specified by IMEM for both spatial and temporal neighborhood is automatically optimized and mapped against the memory resources in a manner that produce an efficient implementation in terms of used resources. The tool also generates a VHDL template for the filter function, instantiates the filter and interfaces it with a memory management VHDL.

5.1 Space-time optimization

In the clock synchronous target architecture, temporal resource sharing is only used within a task. Thus, the scheduling problem for the overall system is reduced to a space-time mapping problem. Where tasks, buffers and delays are placed in time. Since memory is a scarce resource on the FPGA, the placement is done as a minimization process where the objective function is to reduce the number of storage elements.

$$Minimize \sum_{Q \in SI} NS_Q \qquad (1)$$

$NS_Q \in Z^+$ is the memory storage requirement associated with all data flow dependencies stemming from the video stream indexed by Q. $SI = \{1,2,\cdots,NV \mid NV \in Z^+\}$ is an index of NV number of video streams.

One optimization that is explored is buffer retiming, which is depicted in Figure 7 [25]. The objective is to move buffers such that the behavior of the whole system does not change and such that the total size of all buffers is minimized. For example, in Figure 7A the delay is placed after operation *B* the width of the buffer is 12, and when moved to the signal width is reduce to 8. The behavior of the system is unchanged but the buffer size is reduced by 33% [25]. Other optimizations that are explored are sharing of buffers between tasks and buffer elimination.

5.2 Memory allocation and mapping

The Line buffers identified in Figure 4 are required to store data required in the spatial neighborhood in Figure 3. They are implemented using global memory object (GMO) architecture [20]-[23]. For each neighborhood oriented operator in the VIP algorithm, a GMO can be achieved through:

$$W_{Ri} = n_{lines} \times w_p \qquad (2)$$

where W_{Ri} is the width of the GMO, n_{lines} is the number of required line buffers for an operator and w_p is the bit width representing a pixel. The length of the GMO is equal to the length of the operator's line buffers [20]. GMOs require a minimal number of required memory entities in comparison to the direct mapping architecture. Consequently, the number of memory accesses for an RTVPS operation is minimal for a GMO.

Implementing GMOs and their allocation to Block RAMs requires an efficient algorithm so that accessing the allocated data and reconstructing the line buffers would be seamless and with as little overhead and latency as possible. An allocation algorithm has been developed and implemented [21] for this purpose. This algorithm creates the GMOs based on Equation (2). It partitions the GMOs to ensure that their widths conform to those specified by the FPGA, thus ensuring optimal usage of the Block RAMs. The algorithm takes advantage of the dual port capabilities of the Block RAMs to achieve optimal allocations and the possibility of allocating a GMO to as many Block RAMs as required.

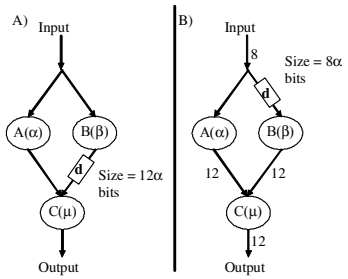

Figure 7. Example of register retiming.

Figure 8 depicts how the algorithm would allocate four memory objects according to the GMO architecture. In Figure 8A, the four line buffers were grouped together to form one GMO. Assuming the GMO is 640 pixels wide and if it were to be allocated on a Xilinx Spartan 3 FPGA, it would be partitioned into two segments, of widths 32 and 16, since it would be not possible to have a data path width of 48 on a Xilinx Spartan 3 FPGA. In addition, since each Block RAM is 16KBit (excluding parity feature), the first segment, of width 32, would require 2 Block RAMs, thus creating two partitions. The second segment would require a single partition on a Block RAM.

Figure 8B illustrates the partitioning of the GMO while Figure 8C shows how the GMO is allocated to two Block RAMs using a data path of 32-bit and 16-bit. The main objective of the allocation algorithm is to minimize Block RAM usage. This is achieved in Figure 8 since two Block RAMs were used as opposed to four Block RAMs required for direct mapping of the four line buffers. In the figure *op, seg, par* and *BR* represent the operator, segment, partition and Block RAM numbers respectively. In [22] two possible approaches for accessing and reconstructing the allocated memory objects were presented and compares. The implemented GMO takes the form of a circular buffer allocated to a set of memory locations corresponding to the video width and performs first-read-then-write memory access operation in one single clock cycle.

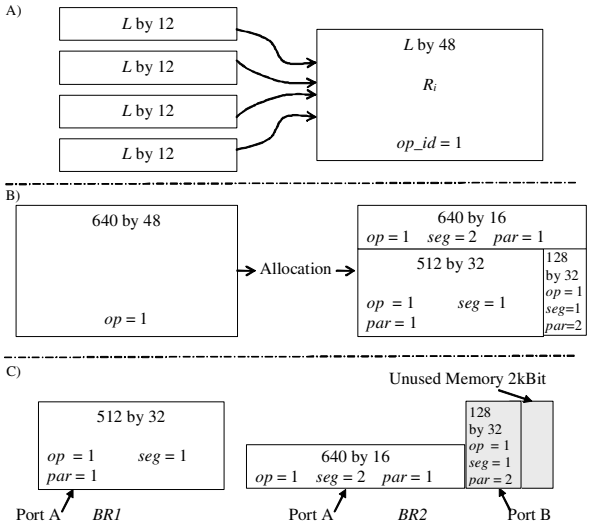

Figure 8. Implementation of memory architecture.

6. Integration and verification

Figure 9 depicts the integration of tools and steps required for system synthesis and verification. The memory requirement, determined by IMEM (example is shown in Figure 9 [A]), is used in the memory synthesis tool to generate a memory management module in VHDL and a SystemC header module (Figure 9 [B]) that contains a reference to the neighborhood oriented filter written in C/C++/SystemC (Figure 9 [C]) as clock sensitive thread. SystemC compilation refines the filter function iteratively through simulation until a synthesizable module satisfying the behavioral specifications of the RTVPS is achieved. This module is then compiled into VHDL module.

VHDL compilation instantiates the memory management module and the synthesizable filter function, implements the timing relation of the system data-flow and verifies the behavior of the system by simulation. The final VHDL module is synthesized and downloaded into FPGA. The SystemC simulator is also used to provide video signal impulse data to the VHDL simulator test-bench and to write its video response thus verifying that the VHDL module produces expected result.

From Figure 9 we can define two approaches to implementing RTVPS namely, automatic synthesis, in which C-like algorithms can be compiled into HDL while our tool is used to manage memories, and semi-automatic synthesis in which the designer writes HDL modules and relies on our tool which is used to manage memories.

7. Results

To measure the performance of the two synthesis approaches identified above, we implemented three simple RTVPS applications with 640-by-480 frame resolution and 3-by-3 spatial neighborhood and compared the results with manual synthesis. The first video operation is a 1-bit morphological erosion, the second an 8-bit mean filter and lastly an 8-bit median filter. Table 1 shows the synthesis results.

Table 1 Synthesis Results

	Erosion			Mean			Median		
	Auto	Semi	Man	Auto	Semi	Man	Auto	Semi	Man
Area									
Slices	125	124	69	256	265	250	517	481	448
FF	88	86	55	332	358	255	509	494	389
LUT	222	220	124	415	421	405	921	840	808
BRAM	1	1	2	1	1	2	1	1	2
Speed (MHz)	92	92	100	75	92	83	42	92	85

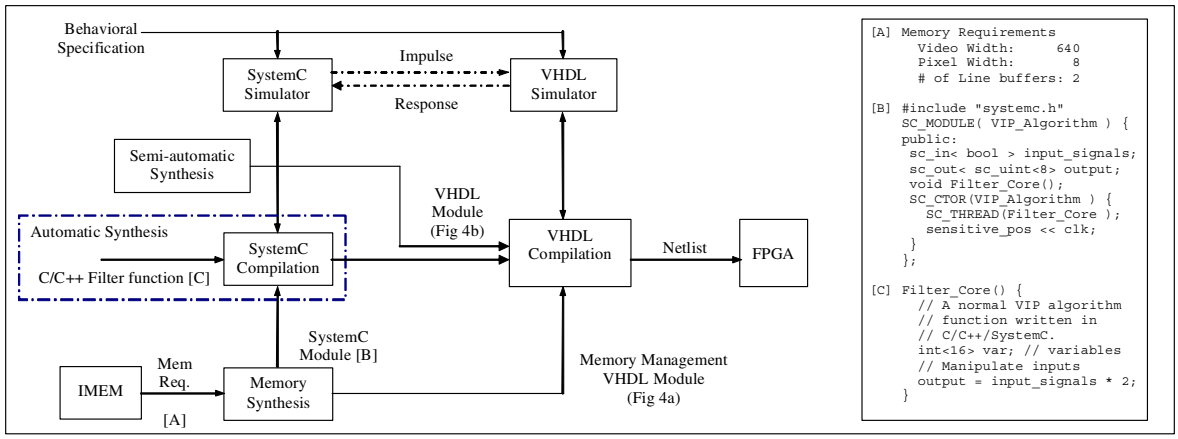

Figure 9. System integration and verification.

The table shows that in all cases, the manual approach has lower area cost except for the number Block RAMs where 50% savings are achieved. If the three filters in Table 1 had been combined according to Figure 5, we will save four out of six Block RAMs. This is due to resource sharing since true dual port allocation is used to allocate to Block RAMs rather than ordinary dual port allocation used in implementing line buffers. The table also shows that the speed of the automatic synthesis is highly affected by the algorithm complexity decreasing from erosion to median filter while remaining slightly constant for the other approaches. From the table the semi-automatic approach combines the advantages of cost and speed. The table does not compare development time since this depends on the designer's skill and algorithm complexity. Our approaches however generally implements algorithms (especially the memory management parts) within a few minute rather than days in the manual approach.

8. Conclusion

This paper has presented a synthesis tool for C++ based synthesis of real-time video processing systems targeting FPGAs. It has been shown that this tool produces cost effective implementations capable of running at high clock speeds. The number of used block RAMs is lower than it would be for a manual design and the speed of the memory architecture is close to the speed of the FPGA resources. The algorithm implementation that is synthesized using this high-level synthesis tool can be written manually or using by third party SystemC to HDL compilers. Thus, the tool presented in this paper is a big step towards accomplishing a compiler that effectively synthesizes real-time video processing systems on to an FPGA. This can lead to new video processing applications, where the combination of high performance, cost effective FPGA and a fully automated design flow would fulfill requirements that otherwise would be hard to meet by most commercial tools but are possible by the tool in this paper due to resource reuse through true dual port allocation to Block RAMs.

References

[1] F. Catthoor, et al. *Custom Memory Management Methodology*. Kluwer Academic Publishers (1998).

[2] W. Wolf, B. Ozer and T. Lv. Smart cameras as embedded systems. *Computer*, Vol. 35 No.9 (2002) 48-53.

[3] D. S. Shrestha, B. L. Steward and S. J. Birrell. Video Processing for Early Stage Maize Plant Detection. *Biosystems Engineering*, Vol. 89 No. 2 (2004) 119-129.

[4] F. Yang and M. Paindavoine. Implementation of an RBF Neural Network on Embedded Systems: Real-Time Face Tracking and Identity Verification. *IEEE Trans. on Neural Networks,* Sept. 2003, pp. 1162 - 1175.

[5] B. A. Draper, J. R. Beveridge, A. P. W. Bohm, C. Ross and M. Chawathe. Accelerated image processing on FPGAs. *IEEE Trans. on Image Processing,* 2003, pp. 1543 - 1551.

[6] R. C. Gonzales and R. E. Woods. *Digital Image Processing*. Addison Wesley (1993).

[7] A. Bovik. *Handbook of Image & Video Processing*. Academic Press (2000).

[8] D. D. Gajski and L. Ramachandran. Introduction to High-Level Synthesis. *IEEE Design & Test of Computers,* 1994, pp 44 – 54.

[9] Open SystemC Initiative, "SystemC User's Guide", version 2.0.1, www.systemc.org

[10] J. Sanguinetti and D. Pursley. High-Level Modeling and Hardware Implementation with General-Purpose Languages and High-level Synthesis. *Ninth IEEE/DATC Electronic Design Processes Workshop,* April 2002.

[11] G. De Micheli. Hardware synthesis from C/C++ models. *In Proc. of IEEE Design, Auto & Test in Europe Conf & Exhibition*, pp 382-383, Mar 1999.

[12] S. A. Edwards. The challenges of hardware synthesis from C-like languages. *In Proc. of IEEE Design, Auto & Test in Europe Conf & Exhibition*, pp 66-67, 2005.

[13] A. Ghosh, J. Kunkel and S. Liao. Hardware Synthesis from C/C++. *In Proc. of IEEE Design, Auto & Test in Europe Conf & Exhibition*, pp 387-389, Mar 1999.

[14] T. Kuhn and W. Rosenstiel. Java based object oriented hardware specification and synthesis. *Proceedings of ASP-DAC* pp 579-581, Jan 2000.

[15] R. Helaihel and K. Olukotun. Java as a Specification Language for Hardware-Software Systems. *IEEE/ACM Inter. Conf. on CAD,* pp 690-697, 1997.

[16] M. Haldar, A. Nayak, A. Choudhary and P. Banerjee. A system for synthesizing optimized FPGA hardware from MATLAB. *IEEE/ACM Inter. Conf. on CAD*, pp 314-319.

[17] P. Banerjee, et al. Overview of a compiler for synthesizing MATLAB programs onto FPGAs. *IEEE Trans. on VLSI Systems*, pp 312-324, Mar 2004

[18] D. Ku and G. De Micheli. HardwareC – A language for hardware design. *Stanford Technical Report*, CSL-TR-88-362, August 1988, and CSLTR-90, April 1990.

[19] Synopsys, C2HDL Compiler, www.synopsys.com

[20] M. O'Nils, B. Thörnberg and H. Norell. A Comparison between Local and Global Memory Allocation for FPGA Implementation of Real-Time Video Processing Systems. *In Proc. of IEEE Int. Conf. on Signals and Electronics Systems,* Sept 2004.

[21] N. Lawal, B. Thörnberg, M. O'Nils and H. Norell. Global Block RAM Allocation Algorithm For FPGA Implementation Of Real-Time Video Processing Systems. *Journal on Circuits Systems and Computers*, Vol. 15, No. 5 Oct. 2006.

[22] N. Lawal, B. Thörnberg and M. O'Nils. Address Generation for FPGA RAMs for Efficient Implementation of Real-Time Video Processing Systems. *Proc. of the IEEE Int'l Conf. on FPLA*, Aug 2005, pp: 136-141.

[23] B. Thörnberg, H. Norell and M. O'Nils. IMEM: an object-oriented memory- and interface modeling approach for real-time video processing systems. *In Proc. of the Forum on Specification & Design Languages,* Sept. 2002.

[24] H. Norell, N. Lawal and M. O'Nils. Automatic Generation of Spatial and Temporal Memory Architectures for Embedded Video Processing Systems. *European Association for Signal and Image Processing (EURASIP) Journal on Embedded Systems,* 2006.

[25] B. Thörnberg, M. Palkovic, Q. Hu, L. Olsson, P.G. Kjeldsberg, M. O'Nils and F. Catthoor. Bit-Width Constrained Memory Hierarchy Optimization for Real-Time Video Systems. *IEEE Transactions on CAD of Integrated Circuits And Systems*.