Conceptual Interface and Memory-Modeling for Real-Time Image Processing Systems

IMEM: A tool for Modeling, Simulation and Design Parameter Extraction

Benny Thörnberg, Håkan Norell and Mattias O'Nils
Dept. of Information Technology and Media, Electronics Design Division
Mid Sweden University
Sweden
{bentho|hakan|mattias}@itm.mh.se

***Abstract*—Most operations invoked in video processing systems are neighborhood oriented. For a video system designer, this limited spatio-temporal collection of pixels represents a natural abstraction. In this paper, we present a basic set of object-oriented design entities. Entities, which can be combined to capture an interface and memory model at a conceptual level, with the neighborhood as an abstraction. These design entities, called IMEM, are implemented as an extension to SystemC. IMEM supports conceptual modeling that excludes implementation details and has explicit data dependency built-in to the model. This makes IMEM a very efficient starting point for design-space exploration and system synthesis. We propose two workflows. The first is a system development workflow, where IMEM represents the starting point of a gradual refinement process, supported by an automated design space exploration step. The second workflow, based on direct mapping of the interface and memory model is presented as being suitable for rapid prototyping. A spatio-temporal noise-reduction filter is selected as a test-vehicle in order to demonstrate the feasibility of IMEM.**

Keywords—Interface- and memory modeling, SystemC, video systems, neighborhood, C++

I. Introduction

Typical image processing operations [5] such as convolution, histogram, spatial and gray-level transforms, erosion, dilation and component labeling are all 2-D neighborhood oriented. The *nlfilter* function in Matlab [14] is based on this fact. Consequently spatio-temporal Video Processing Systems (VPS) will operate on a 3-D neighborhood [1][10], thus increasing the system complexity. From a VPS designer's point of view, the today's specification methods lacks in abstraction. The stream oriented abstraction chosen for the PHIDEO system [4] does not reveal the neighborhood that naturally is common for most VPS operations. Nested loops, such as in a DFL-specification [8], need code pruning in order to analyze the data dependency between neighborhood pixels. This pruning will separate the spatial and temporal mapping from the functional mapping of a video processing algorithm. VPS specifications written as nested loops define how the neighborhood slides within a spatial domain, which effects the sizes on input and output buffers. This is implementation related information, which is undesirable during early design exploration.

Real-time video processing systems are data dominated. Typically the design bottleneck will be the memory data transfers maintaining a spatio-temporal neighborhood. Another closely related and also critical design parameter is the large amount of background memory and the power dissipation coming from the high-speed accesses. These critical parameters have been addressed in [11] and an implementation using a memory hierarchy to overcome the memory access bottlenecks has been presented by Oelmann et al. [12]. Wuytack et al. [6] presents a more general methodology, where data reuse exploration is done by introducing application-specific cache memory hierarchies. Applied on realistic VPS applications, the system design exploration-tool ATOMIUM has enabled power reduction of about 90% [7]. ATOMIUM is based on both loop transformations and memory organization decisions. Typically this exploration is done early in the design process. The ATOMIUM design entry is a DFL-specification [8], which needs additional profiling in order to extract inter-pixel and inter-frame data dependencies.

The evolution of object-oriented specification methods based on class libraries has made language extensions possible to implement without having to update the compilers or simulators. Our previous research [13], indicates that the object-oriented specification methods in SystemC [2] are a good candidate for modeling VPS.

Although some research has been made in the area of memory modeling and VPS, up until now, no research has shown the potential of combining a video designer friendly neighborhood abstraction, conceptual modeling and early design space exploration methods into one homogeneous C++ system design environment. This is an environment that will take a VPS all the way from specification down to implementation. The reduction of time to market and the implementation optimization serve as motivation for this and future research in this area.

This paper presents an object-oriented approach to conceptual memory- and interface modeling, called IMEM,

that targets real-time VPS. Basic modeling entities such as input and output video stream-ports, frames, frame buffers and sliding bodies, provide the VPS designer with a specification method that can easily capture stream ports, spatial and temporal mappings of video processing algorithms. The proposed conceptual-level modeling excludes implementation details and provides the designer with means for explicit data-dependency modeling. Consequently, no additional pruning is needed to extract the data dependency, as in the case of nested loops. This will of course simplify the implementation of design space exploration methods.

Additionally, a system design workflow and a rapid prototyping workflow are proposed. Both workflows use IMEM as the common design entry for both the hardware and the software parts of a system. The system design workflow involves automated design space exploration and a rapid prototyping workflow involves a direct mapping of the interface and memory model. Design space exploration and direct mapping are areas of research that we would like to address in future. IMEM will serve as the basis for this research.

II. SPATIO-TEMPORAL AND FUNCTIONAL MAPPING

A body is a 3-dimensional collection of pixels that serves as an excellent abstraction for neighborhood oriented video processing operations. This body is the mapping of an algorithm onto the spatial and temporal domains. The functional mapping defines how an output pixel is determined from a body as input. Figure 1a) shows such a geometric graph of a collection of pixels and Figure 1b) an UML class diagram of the same body modeled with IMEM design entities. The geometric graph also shows two examples of how individual pixels are addressed.

Design entities are implemented in IMEM as C++ classes which can be instantiated and connected together into a structure.

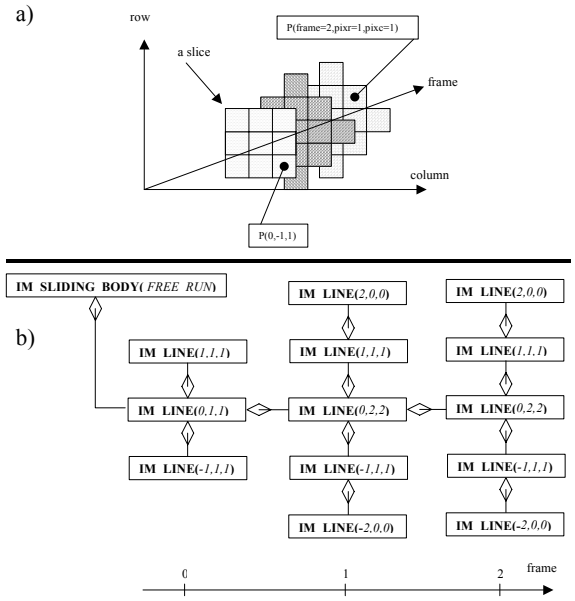

Figure 1. How to specify a body in IMEM.

A structure of IM_LINE entities is used to model a body. IM_LINE has three parameters, *row, left* and *right*. *Row* is the relative row-axis position. *Left* and *right* corresponds to the number of pixels to the left and to the right of the body centeroid. This body consists out of three slices, where one slice is a spatial collection of pixels. The first slice, the oldest in the temporal dimension, is modeled with three instances of IM_LINE, owned by IM_SLIDING_BODY. This slice is referenced as having the relative frame number 0. The latest slice in this example has the relative frame number 2 and this part of the IM_LINE structure is depicted rightmost in the UML class graph.

III. INTERFACE MODELING

Figure 2 depicts an example of how to use design entities for interface modeling. Figure 2a) shows a structure that captures an input video stream. The first two parameters of IM_FRAME indicates that the frame size is 576x720 pixels. The last two parameters defines the synchronization position of the output frame at the spatial coordinates (50,50). The smaller output frame has its origin positioned within the input frame at the synchronization position. This cropping mechanism is illustrated in Figure 2c). The first parameter of IM_LAYER defines the layer number and the second its semantic. This entity is in this example used to capture the RGB color space model. The order of the IM_LAYER entities defines the order they appear on the stream port. The topmost entity appears first. The IM_BUFFER has one parameter that defines the size in frames. The sequence of the input port is set to INTERLACED_ODD by the IM_IVPORT entity. Figure 2b) shows a structure that defines the output stream. The buffer in this case is set to GENERIC, which means that the minimum size will be determined by IMEM. For example a change in the pixel sequence would require the insertion of a buffer. But the selection of sequence will also have a most crucial effect on the systems memory bandwidth, [12].

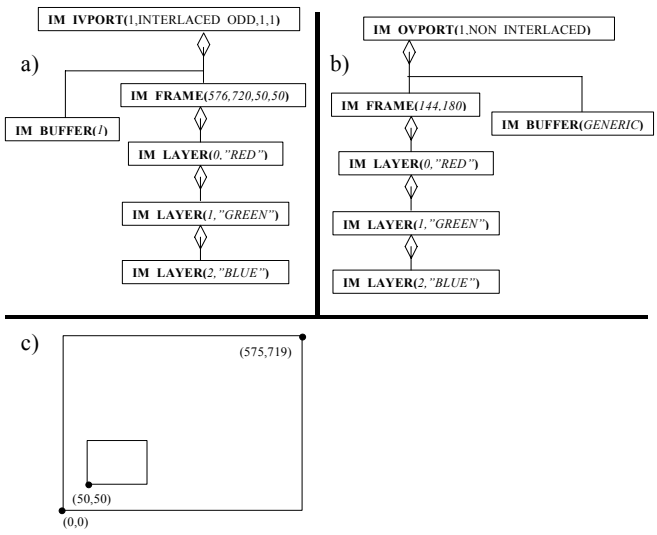

Figure 2. Interface modeling with IMEM.

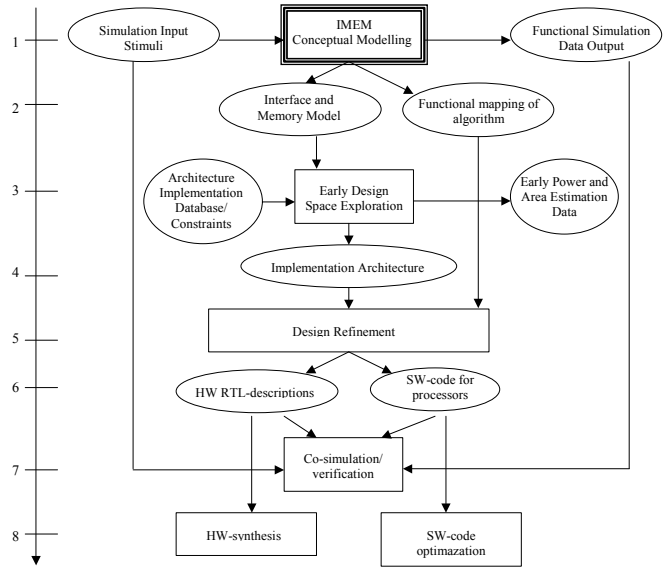

Figure 3. System design workflow.

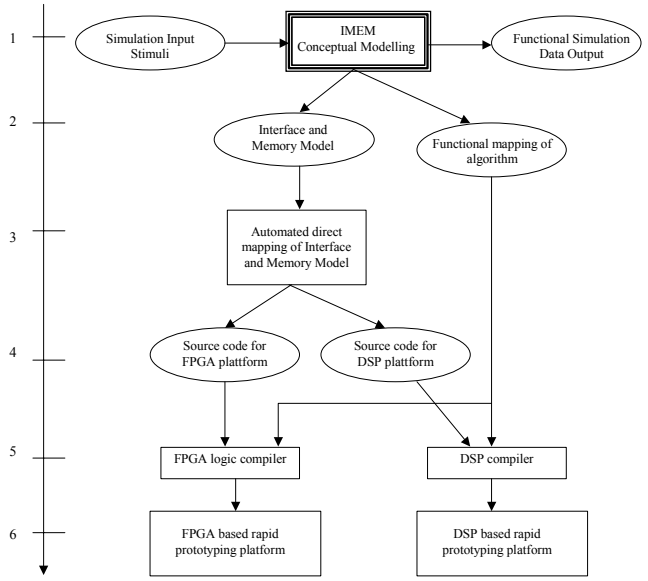

Figure 4. Rapid prototyping workflow.

IV. System-design work flow

Figure 3 depicts the workflow we propose to be used together with IMEM. The workflow runs along eight levels defined at the left-side axis. The video processing algorithm is developed by using IMEM at level 1 for conceptual modeling. This means that the developed algorithm is captured without adding any implementation related information. The model can be verified through functional simulation. Data dependency information such as frame size, composition of the 3-dimensional neighborhood and color space model mapping is exported into an interface and memory model at level 2. A database of devices belonging to the selected design space to be searched is added at level 3. Power/area trade of constraints are processed together with the database in order to derive a near optimal implementation architecture. The output from this design space exploration is cash levels and sizes, background memory configuration and organization, processing sequences, buffers and bus coding. An estimation of power and area is also generated. The implementation architecture is the input to a manual design refinement step at level 5. The output from this refinement can be hardware RTL-descriptions and/or software source code at level 6. The complete system can be co-simulated at level 7. Hardware descriptions are synthesized and software modules are optimized at level 8 to reach the final implementation.

V. Rapid-prototyping workflow

The time from algorithm to implementation is the most important constraint for a rapid prototyping environment. Less attention is paid to optimization. We propose the workflow depicted in Figure 4 as suitable for rapid prototyping. This workflow is defined at six different levels along the left-side axis. The video-processing algorithm is developed and simulated using IMEM at level 1. This initial step is equal as for the system design workflow. The memory and interface model at level 2 is the input to a direct mapping process at level 3. The output from this direct mapping is either source code targeted for a field programmable gate array, FPGA, or a digital signal processor, DSP. The source code can be generated by mapping the IMEM interface and memory model with a parameterized and generic target specific model. This is what we call direct mapping. The functional mapping of an algorithm can then interface directly with the generated interface and memory model and be compiled at level 5. Level 6 corresponds to either a FPGA- or DSP-based prototyping platform.

VI. Selected filter algorithm

We have selected an adaptive median noise reduction filter as test vehicle. This algorithm can be divided into two main sub-tasks:

1) To detect a part of the image and determine whether this is a part of a moving image, that is called local scene-change detection.
2) To filter out noise with local scene-change taken into account.

A block diagram of the filter is depicted in Figure 5. The notation and definition used in the algorithm description are: A frame *F(n),*

$$F(n) = \{R(n), G(n), B(n)\}$$

is a matrix of RGB-values (color components) in the n:th frame. A pixel *P(i, j, n),*

$$P(i, j, n) = \{R(i, j, n), G(i, j, n), B(i, j, n)\} \in F(n)$$

is an element in *F(n)* with the spatial position *(i, j)*. A slice,

$$S(i_0, j_0, n_0) \subset F(n_0)$$

positioned at (i_0, j_0) in the n_0:th frame includes the pixel $p(i_0,j_0,n_0)$ and a portion of the n_0:th frame that surrounds the pixel p_0. A tube,

$$T(i_0, j_0, n_0) = \{S(i_0, j_0, n) \mid (n_0 - d) \le n \le (n_0 + d)\}$$

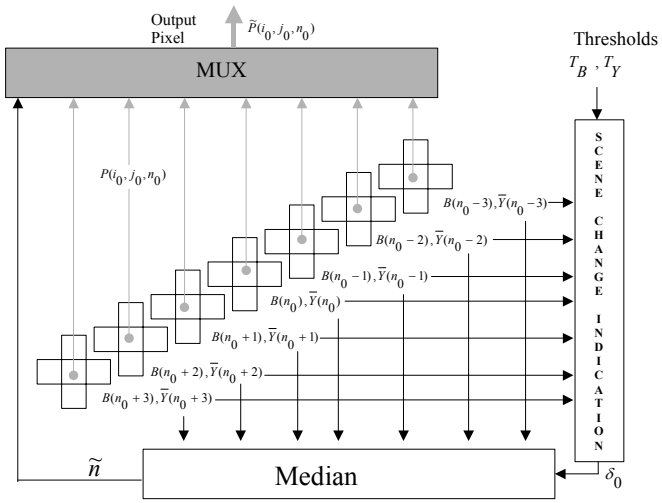

Figure 5. Block-diagram of selected filter.

is a set of slices with same *(i, j)* but located in consecutive frames, where the number of frames is defined as $2 \cdot d + 1$. d frames surround the center frame in time. For our case study d is set to 3. The first step of the algorithm is to calculate the average luminance for each slice in a tube $\bar{Y}(n)$.

The average luminance for two adjacent slices and the blue color component of two pixels belonging to the same slices are used to calculate the luminance and blue color differences. If either of the differences is higher than a certain threshold level (T_y and T_b), a scene change is indicated in a vector, $I(n, n-1)$.

From the scene change vector I, the length, δ_0, from a scene change to the center pixel determine the length used by the median filter. The luminance from the center pixel in a tube and the median filter width, δ_0, are the inputs to the median filter. The filter output is selected from the center pixel's original RGB-values in frame number ñ.

VII. Results

The selected filter algorithm was captured using the IMEM design entities. A sequence of 576*720 sized frames where used as input stimuli. 1000 frames where processed on a Pentium III (800MHz) computer running Windows 2000 which resulted in an elapsed simulation time of 3h, 29 min and 8 s. Hence it took about 12.5 seconds to process one frame. The purpose of this simulation run is to demonstrate that IMEM can be used in practice for modeling and simulation.

Design parameters extracted from the IMEM-model of the filter are the selected frame size, the composition of the 3-dimensional neighborhood, see Figure 5, the constraints for an input buffer, the input and output stream pixel sequences and the RGB color space model mapping.

VIII. Conclusion

In this paper, we have shown that a 3-dimensional collection of pixels is a natural and modeling efficient abstraction for video processing operations. To support this abstraction, an object-oriented specification method, IMEM was presented. What is even more important, is that the structure of design entities, being a conceptual memory and interface model, explicitly reveals data dependency information and buffer constraints. This information can then be exported into a design exploration tool or an automatic synthesis tool. We have proposed how this conceptual modeling and design exploration can be used in a system design flow. Additionally we have presented a path to rapid prototyping from the IMEM model. These are areas of interesting and challenging research that we will address in future. The higher abstraction in IMEM will serve as an excellent extension to the already well-known SystemC workflow. We believe that the IMEM concept will be most applicable for prototyping systems or any real-time video processing system that needs to be optimized for power and production cost and were the time to market is important. Battery powered consumer electronic devices, such as camcorders or video telephones are examples of envisioned applications.

References

[1] S.J. Hill, D. Crookes and A. Bouridane, "The use of high level tools for developing volume graphic and video sequence processing applications", *Proceedings of 7th international congress on image processing and its applications, IEE 1999, (Conf. Publ. No.465).*

[2] *SystemC User's Guide*, Version 2.0, http://www.systemc.org

[3] *Master-slave communication library user's guide,* version 2.0 beta-3, http://www.systemc.org

[4] W.F.J. Verhaegh, P.E.R. Lippens, E.H.L. Aarts, J.L. van Meerbergen, and A. van der Werf, "Modelling Periodicity by PHIDEO Streams", *Proceedings of 6th International Workshop on High-Level Synthesis, pp. 256-266, 1992.*

[5] *Digital image processing*, R.C. Gonzales and R.E. Woods, Addison Wesley 1993.

[6] S. Wuytack, J.Ph. Diguet, F.Catthoor and H. De Man, "Formalized methodology for data reuse exploration for low-power hierarchical memory mappings", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.6, no.4, 1998.*

[7] L. Nachtergaele, F. Catthoor, F.Balasa, F. Franssen, E. De Greef, H. Samsom and H. De Man, "Optimization of memory organization and hierarchy for decreased size and power in video and image processing systems", *Records of the 1995 IEEE international workshop on memory technology, design and testing, 1995.*

[8] C.A Dace. "An applicative high-level language for dsp system design", *IEE Colloquium on General-Purpose Signal-Processing Devices (Digest No.085) 1993*

[9] K. Wiatr, "Dedicated hardware processors for real-time image data pre-processing implemented in FPGA structure", *Proceedings of ICIAP 97. 9th International Conference on Image Analysis and Processing, vol.2, pp 69-76.*

[10] L. L. Nozal, G. Aranguren, J. L. Martín and J. Ezquerra, "Moving images time gradient implementation using RAM-based FPGA", *Proceedings of the SPIE - The International Society for Optical Engineering 1997, vol.3028, pp.108-116.*

[11] B. Taylor, "DSP filters in FPGAs for image processing applications", *Proceedings of the SPIE - The International Society for Optical Engineering 1996, vol.2914, pp.100-109.*

[12] B. Oelmann, H. Norell, R. Andersson, Y. Xu, "Design of real-time signal processing ASIC for noise reduction in moving video images", *Proceeding of IEEE Norchip Conference 1999, pp.228-33.*

[13] B. Thörnberg and M. O´Nils, "Analysis of modeling and simulation capabilities in SystemC and Ocapi using a video filter design", *Proceeding of ECSI forum on design languages 2001*

[14] *Matlab user's manual*, http://www.mathworks.com