

Using UML 2.0 for System Level Design of Real Time SoC Platforms for Stream Processing

Yongxin Zhu[†], Zhenxin Sun[†], Alexander Maxiaguine[‡], Weng-Fai Wong[†]

[†]School of Computing, National University of Singapore
{zhuyx, sunzhenx, wongwf}@comp.nus.edu.sg

[‡]Computer Engineering and Networks Laboratory, ETH Zurich
maxiagui@tik.ee.ethz.ch

Abstract. Programmable stream processors enable quick reconfiguration and implementation of stream applications. Existing specifications in stream modelling languages lack compatibility and adoption. We describe a design approach where specifications are captured in UML 2.0, then automatically translated into SystemC models consisting of simulators and synthesizable code under proper style constraints. As an application case of the approach, we explain new UML 2.0 notations and use them for real time stream processor specifications. The trace of the execution of UML models of stream processors is given to illustrate verifications at early stage of design flow. Then we expound how our translator generates SystemC models from UML models. The generated SystemC code is acceptable to further synthesis processes. The case study demonstrates the feasibility of fast specifications, modifications and generation of real time stream processor designs.

1 Introduction

Over the last few years, there is a remarkable increase in the complexity of media-intensive products such as digital video recorder, DVD players, MPEG players and video conference devices. The complexity stems from the heavy computational workload and large data volume in the media stream.

Due to the complexity, designing streaming processors has become a complicated task. An approach to taming the complexity is to raise the level of abstraction to that of system-level designs. This approach incurs the needs to specify abstract models of systems.

Because of specific application domains, existing stream programming languages such as StreamIT [15], StreamC [13], Brook and Cg. have many differences which result in an incompatibility issue to specifications and design reuse.

More importantly, they are *programming* languages and do not meet the requirements for high level abstraction, description and modeling.

The Unified Modelling Language (UML) is gaining popularity from real time system designers. UML has been widely accepted in both software design and system architecture community since its invention. UML has also been evolving to meet the needs of real time embedded systems [8, 10]. UML 2.0 incorporates more modelling abstraction notations in the real time software domain and designers can expect to use these notations in their initial specifications, consisting of class diagrams, state diagrams and structure diagrams, that are available in products such as Rational Rose Real Time and Rhapsody [6].

However, real time UML may not suffice for stream processor designs because of their special needs. On stream processors, a large amount of on-chip resources are allocated to deliver long instruction and large data packets. Stream processor architectures also have to adapt to different stream applications. Although programmable stream processors [7] provide the hardware for quick implementation, it is still not clear for designers if UML 2.0 can be used to specify streaming features clearly and easily. This is one of the aims of this paper.

Since there is a huge gap between UML-based descriptions and their implementations, it is clear that such descriptions need to be refined to a number of more detailed ones in order to get the final implementation level specifications. Here SystemC seems to be an excellent choice to serve as the layer immediately “below” the initial top-level UML-based design. SystemC is a design language that allows designs to be expressed and verified at sufficiently high levels of abstraction while at the same time enabling the linkage to hardware implementation and verification. Furthermore, SystemC, when viewed from a programming perspective, is a collection of class libraries built on top of C++ and so is naturally compatible with object oriented design methodologies. Therefore, creating a flexible and fully automated translation mechanism from the UML-based design language at the top layer to SystemC appears to be a promising strategy.

2 Our Scope of Work and Related Work

In the domain of UML for real time systems, a large body of work has been reported. Some UML extensions from UML 1.x are proposed in [10] and [8]. A survey on the efforts to extend UML 1.x for real time applications can be found in [9]. Two instances of UML extension and translation are [2, 16]. Some studies exploited the possibility to describe a synchronous dataflow model [4] and a model for incremental designs [3] in UML. Another previous effort [14] focused on hardware generation from UML models without explicit UML specifications of stream processing.

In this paper, we explore the suitability of UML 2.0 for system-level designs of stream processors. For this purpose, we establish a design flow that starts with UML 2.0 while using SystemC as an intermediate language. Using this flow, we can specify stream processors and applications in UML at an abstract level using I-Logix's Rhapsody. From the UML specifications, we generate a stream processor simulator in SystemC. Here are the distinct features of our work:

1. To ensure the maximum reusability and compatibility, we simply use existing UML 2.0 notations without introducing any new ones.
2. In our UML specification, we explicitly describe real time stream application features.
3. Due to the asynchronous nature in video and audio stream processing, data packets are processed in different rates at different processing stages. Our SystemC simulator generated from UML designs supports the behavior of asynchronous flows. This is in contrast to a recent study [4].

In the rest of the paper, we will discuss the UML model of system level stream processing, describe the design flow, present the translator which can produce SystemC code from restricted Rhapsody designs, and illustrate the flow with a case study. Some concluding remarks will follow the discussions.

3 Requirements of Real Time Stream Modelling

We view stream applications as operations on flows. Flows of stream packets undergo processing at different stages of calculation. Representations of flow are mandatory in stream modelling.

Flows require entrances and exits at processing stages. These gateways are also to be represented abstractly with distinct attributes.

There is also a need to hierarchically specify the computation complexity inside processing stages. Since video and audio stream protocols have become very sophisticated, the complexity of calculation at each stage grows so much that a non-hierarchical representation will be too complex to manage.

We further consider real time constraints in stream processing. In classic real time systems, real time constraints are usually applied to system response time. Other than the classic constraints, constraints on stream processing are often in the form of quality of service, which ensures a processing rate of the stream. For example, an MPEG-2/4 video clip requires a minimum playback rate at 25 frames per second (fps). The number of pixels per frame depends on the resolution settings during sampling. Similarly, for MP3 audio clips, the choices of playback rates usually range from 64 Kilo bits per second (Kbps) to 256 Kbps depending on the requirements of playback quality.

The implication of the guarantee of the playback rate is there are enough data stored in the buffer in front of the playback device which is a real time client. Let B_0 denote the buffer fill level of the playout buffer, $x(t)$ to denote the input stream of this buffer, D_0 represent the initial decoding delay, and C denote the consumption rate of the real time client assuming the initial buffer fill level is 0. Then the constraint on the playout buffer underflowing can be stated as:

$$x(t) \geq C(t) - B_0, \quad \forall t \geq D_0 \tag{1}$$

Similarly to the above limit on the minimum amount of the stored data, there is constraint on the maximum amount to avoid overflow.

4 UML 2.0 Notations Used

To address the needs of abstract representations of stream processors in UML, we resort to a few new notations introduced in UML 2.0, namely *flow*, *composite classes*, *port*, *structural diagram* and *hierarchical statechart*.

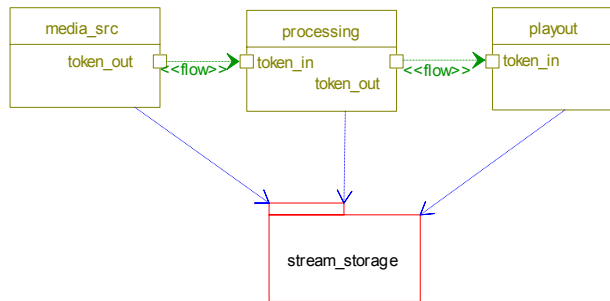


Fig. 1. Functional Block Modelling of a Stream Processor

In Fig. 1, a functional block model of a stream processor is shown. The model illustrates the exchange of information “flow”s at the highest abstract level. Streams are represented in tokens which flow into and out of processing elements.

The blocks in the functional block model are “composite classes” which decompose hierarchically into standard classes. The concept of composite class is similar to the definition of “Metaclass” proposed by the UML for SoC forum [5]. We expect the notations of composite class and metaclass will be merged in the final version of UML 2.0.

The notation of “port” is new in UML 2.0. It is used in defining complicated incoming and outgoing interfaces. In Rhapsody’s terminology, the interface for incoming messages is called a “provided interface”, similarly the interface for outgoing messages is termed a “required interface”.

To decompose descriptions of large computation complexities, “hierarchical statecharts” are needed. Nested statecharts are allowed to represent sub-machines as a part of parent chart.

The “structural diagram” is usually a detailed description of functional block models. It emphasizes the structure of the objects in a system, including types, relationships, attributes and operations. We will show a structural diagram in Fig. 5.

Activity dependencies have to be represented as well. A stream processor needs stream traces to drive its execution. The relationship is shown as a *dependency* from processing blocks to the stream storage package in Fig. 1. In the stream storage package, a trace manager manages stored trace files and delivers necessary stream attributes to the execution pipelines. The description of the stream storage package is an abstraction of memory management mechanisms. The memory can be either physically centralized RAM blocks or logically assembled cache lines associated with each processing element.

A constraint is a condition or restriction expressed in text. It is generally a Boolean expression that must be true for an associated model element for the model to be considered well-formed. It indicates a restriction that must be enforced by the correct design of a system.

A constraint is an assertion rather than an executable mechanism. It is useful to check the validity of a design in simulation of the model of the design. It does not necessarily exist in the final description for hardware synthesis.

5 A UML 2.0 Based Design Flow

We propose and implement a design flow based on UML 2.0 with SystemC as its intermediate language in Fig. 2. In this design flow, our own efforts include the stages of UML specification, transaction level modelling (TLM) style translation, register transfer level (RTL) style translation and SystemC simulation.

The translator is based on the XML Metadata Interchange (XMI) format of UML representations made with Rhapsody. From XMI representations, the translator generates SystemC code in both TLM and RTL style.

To use the flow, designers usually work with the UML specifications only as the rest steps of the flow are automated. The major work of the designers is to visually specify the processor architecture parameters and stream application parameters in UML 2.0. The UML models can be executed for early verifications.

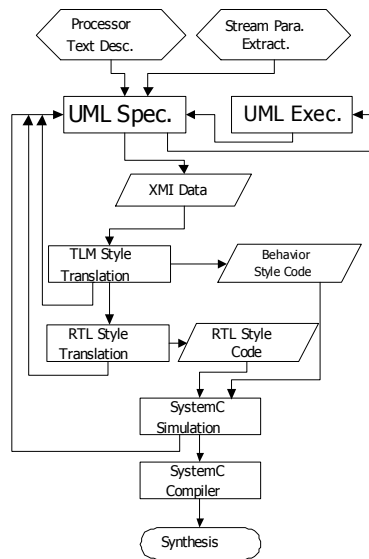


Fig. 2. A Design Flow Based on UML 2.0

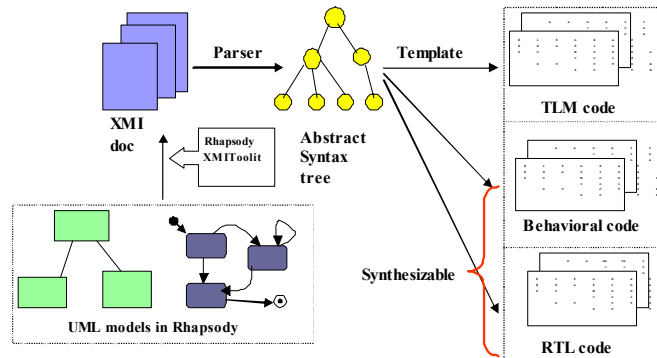


Fig. 3. The Translation Process

The UML specifications are easily exported to XMI format files which are translated by our translator to SystemC simulator models in TLM and RTL styles. The simulator code under proper coding constraints [14] can be compiled by the Synopsys CoCentric SystemC compiler into gate level netlists in Verilog. The netlist files are synthesizable into programmable architecture implementations.

In the meantime, verifications are carried out at multiple levels, i.e., UML specification, TLM style translation, RTL style translation and SystemC simulation. Early verification enables the early detection of possible design errors, and reduces the costs of debugging.

6 The UML2Code Translator

UML 2.0 is closer to SystemC than its predecessor. A meta-class corresponds to a module in SystemC. Some of the new concepts in UML 2.0 such as *port* and *flow* have direct counterparts in SystemC. Like *sc_port* in SystemC, the port in UML serves as the gateway for communications between modules. Flow encapsulates the definitions of the message structures. Its counterpart in SystemC would be *sc_interface*.

Figure 3 shows the translation flow of UML2Code. We start with a UML 2.0 model in Rhapsody 5.01. Corresponding XMI documents are generated for the completed design model using Rhapsody's XMIToolkit. These are the inputs of our translator.

XMIToolkit preserves all the model information, and it is a textual representation of the UML model. The generator parses the XMI documents and builds the internal representation of the model. The syntax tree is further processed and some relations are computed and stored. The syntax tree holds all the data required to build different levels of code. By using different templates, the same model can be used to generate code in different levels: TLM level, Behavioral level and RTL code. Behavioral and RTL code can be further synthesized into gate-level netlists.

7 A Case Study

Video and audio decoders are typical streaming applications. Examples include the MPEG2 video decoder [12] and MPEG Audio Layer 3 (MP3) decoder [11]. As a case study, we shall map these applications into an abstract processor architecture, then implement the abstract architecture using our design flow.

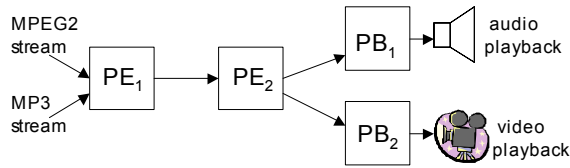


Fig. 4. Abstract Architecture of a Stream Processor

7.1 Experimental Setup

The process to decode MPEG2 video streams consists of 4 major steps, namely variable length decoding (VLD), inverse quantization (IQ), inverse discrete cosine transition (IDCT) and motion compensation (MC). We map VLD and IQ to the first stage of a stream processor, IDCT and MC to the second stage.

To play back MP3 audio streams, we usually need 8 steps, namely (1) synchronous and error checking, (2) Huffman decoding, (3) re-quantization, (4) reordering, (5) alias reduction, (6) Inverse Modified Discrete Cosine Transform (IMDCT), (7) frequency inversion, and (8) synthesis polyphase filter. We assign the earlier 4 steps to the first stage of the stream processor, and the rest 4 steps to the second stage.

In addition, another important design factor, *scheduling*, is also considered in our model. In our case study, we implemented the TDMA scheduling policy as the scheduler.

We summarize the above settings in the abstract architecture of the stream processor in Fig. 4.

To obtain the requirements of execution cycles at different pipeline stages, we use one of SimpleScalar’s [1] tools, “sim-profile”, to run reference implementations of MPEG2 decoder and MP3 decoder separately without scheduling. The execution cycles are captured and stored into trace files. For each video or audio stream, there are a set of trace files for every pipeline stage to indicate the execution cycles. The trace files are records of stream tokens. Each record contains a token index number and token attributes such as the execution cycles at different pipeline stages during stream processing.

To support real time applications, we face two choices of UML tools from the industry, namely, Rational Rose Real Time (RT) and I-Logix Rhapsody. Since

the latest I-Logix Rhapsody provides more support for UML 2.0 notations than Rose RT such as functional block modelling, flow and concurrent state charts, we choose I-Logix Rhapsody as our front end. Another important feature of I-Logix Rhapsody is that it can produce representations in XML Metadata Interchange (XMI) format which is not supported in Rose RT.

7.2 UML Specifications

To detail the descriptions of “flow” driven activities in Fig. 2, we use a *structural diagram* in Fig. 5. In this structural diagram, we describe the activities of instances of objects whose realistic roles are shown in Fig. 4.

In Fig. 5, the ports belonging to object instances form the communication channels for streams. The video and audio streams are tokenized into token flows. Token IDs are initiated from video source or audio source, then pass through processing elements to reach the video sink or audio sink. Each processing element queries the trace manager for detailed attributes of tokens. The processing elements take the proper actions in accordance to the attributes of tokens.

In UML 2.0, there is no need to inherit explicitly the interface definitions, as was the case in UML 1.x, for the ports where token IDs go through. The interface definitions are built into the ports in UML 2.0.

The abstract operations in Fig. 5 represent a typical design in stream processing. Some streams consist of large packets of data. Each processing element does not need all the data in a packet. It is not efficient in hardware to transfer all the data in the packets through the processing elements. In our example, only token IDs are passed through processing elements, and each processing element only fetches necessary data for its processing from the trace manager. The hardware costs will be reduced by this means.

In Fig. 6, we use a traditional class diagram to specify the inheritances and association among the components of the processor and management of stream traces. These entities are classes which inherit or aggregate attributes from other classes or interfaces.

UML 2.0 supports hierarchical and concurrent statecharts. We use traditional or single-layer statecharts to specify simply calculation logic flows inside each object. To describe complicated tasks, we use these new features to describe

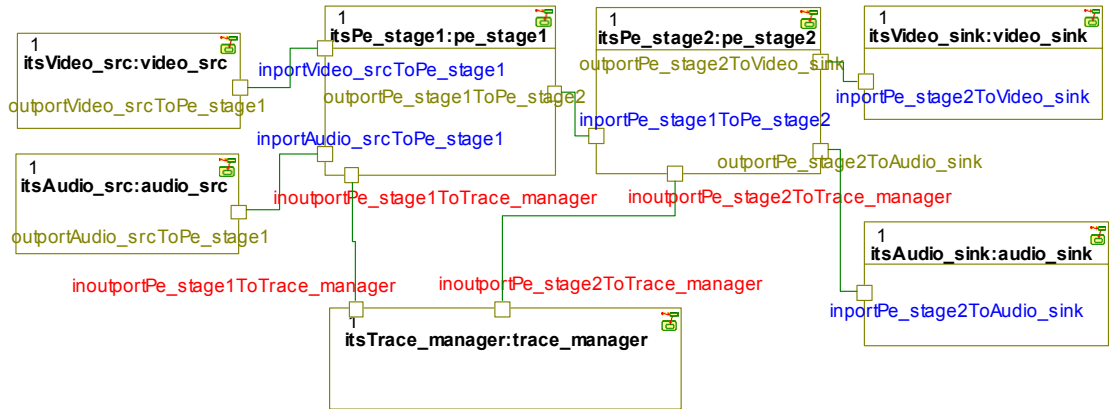


Fig. 5. A Structural Diagram of a Stream Processor

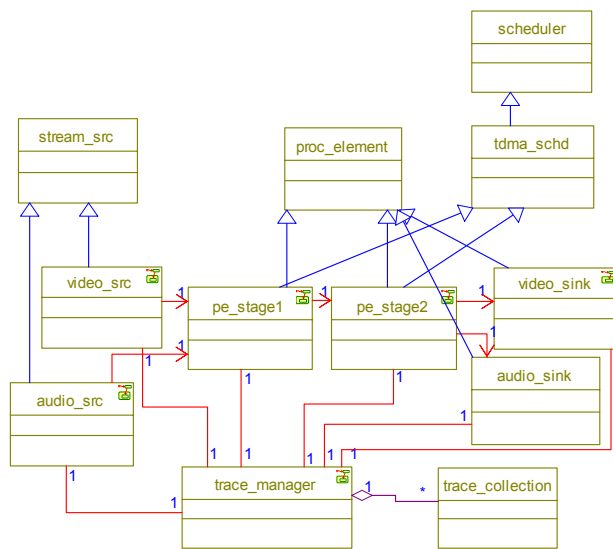


Fig. 6. A Class Diagram for a Stream Processor

hierarchical states and concurrent calculations. For example, we use such a hierarchical state containing concurrent sub-machines in the “trace_manager” as shown in Fig. 7.

Similar to assertions, constraints have to be checked constantly during processing of the streams. An example of constraints is shown in Fig. 8 for illustra-

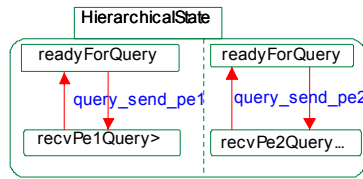


Fig. 7. A Hierarchical State in An Object

tion. The constraint has a dependency relationship with the object. A violation of the constraint incurs a pause in the playback client, consequently, users observe breaks in the playback video or audio clips. The number of constraint violations is recorded as a metric of quality of service.

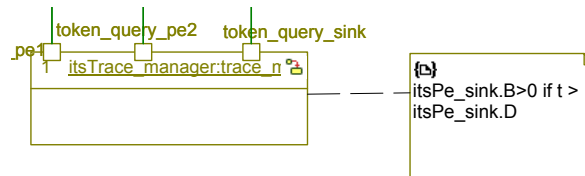


Fig. 8. A Constraint Anchored to an Object

As shown in the example, designers only need to deal with structural diagrams, class diagrams and statecharts for specifications in UML. It is straight forward to reuse the specifications and quickly modify designs in UML 2.0.

7.3 UML Execution

UML models are executable in the Rhapsody runtime environment. This feature allows designers to verify their designs at very early stage of design flow.

UML 2.0 visualizes the communication with a new feature *message sequence chart*(MSC). Fig. 9 is the screen shot of a MSC generated during the execution of UML models. In this MSC, tokens trigger events which are represented as messages passed among processor modules. We can trace the execution of video tokens 3, 5 and audio tokens 12, 14. The example shows another clear benefit of UML 2.0 for system level design of stream processors.

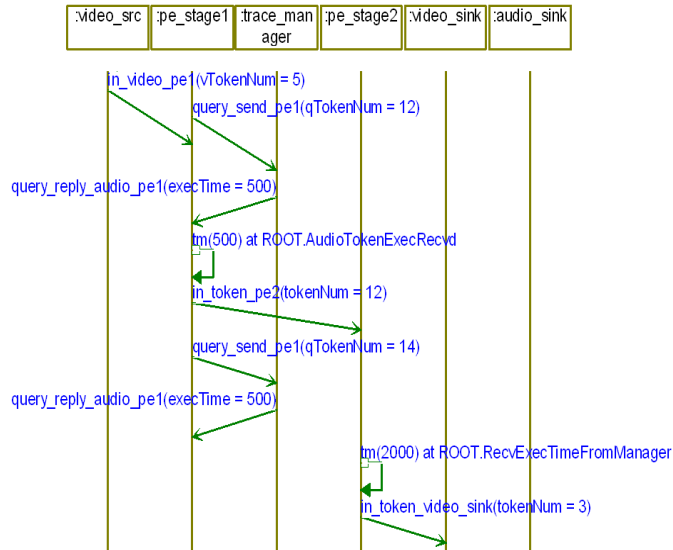


Fig. 9. Communications among the Processor Modules

To achieve a faster execution, we can also disable the MSC generation during execution. The option is observed to save the execution time by about 30%.

7.4 SystemC Code Generation

To facilitate the translation synthesis, we apply an intuitive naming conventions to the port names. The names of outgoing ports start with “outport”; incoming ports have names beginning with “inport”; bi-directional ports are identified as “inoutport*”.

During the code generation, we apply the following mapping rules. All components in the structural diagram Fig. 5, are translated into a “sc_module”. The port type is determined by the naming convention of the port. The statechart in each active object is translated into a process whose method name is “entry”. In SystemC, there are 3 types of processes, i.e, *sc_thread*, *sc_cthread* and *sc_method*. For TLM style translation, we map each statechart into an unlocked light weighted process “sc_thread” since the TLM model focuses on the functionality of transactions. For RTL style translation, the mapping is a clocked process “sc_method”, which will be synthesizable.

For the settings in the example, the size of the generated SystemC code is about 3000 lines. The code generation time for both RTL style and TLM style is less than 10 seconds. It is also interesting to note that the RTL style code may have a smaller size than TLM style code, but always a longer execution time.

To illustrate the mapping results, we take the RTL style declaration part of the SystemC of the first stage of processing “pe_stage1” as the example.

```
SC_MODULE( pe_stage1) {
//define default ports
    sc_in_clk CLK;
    sc_in<bool> reset;

//define the outgoing ports
    sc_out<bool>
        trace_manager_query_send_pe1;
    sc_out<int >
        trace_manager_query_send_pe1_qTokenNum;
    sc_out<bool>
        pe_stage2_in_token_pe2;
    sc_out<int >
        pe_stage2_in_token_pe2_tokenNum;

//define incoming ports
    sc_inout<bool> in_video_pe1;
    sc_in<int > in_video_pe1_vTokenNum;
    sc_inout<bool> query_reply_pe1;
    sc_in<int > query_reply_pe1_execTime;
    ....
//token attributes
public:
    int vTokenNum;
public:
    int vExecTime;
    ....
//processing status
public:
    int bufLevel;
private:
    int state;
    ....
//clock and process definitions
public: SC_CTOR(pe_stage1) {
//define the clocked thread
    SC_METHOD(entry);
    sensitive_pos<<CLK;
} ...
}
```

These descriptions in SystemC are synthesizable with the Synopsys CoCentric SystemC compiler in a behavior description style. After the compilation process, descriptions in Verilog at the gate level are generated, which are acceptable to further FPGA design flows.

Though the generated design is synthesizable, we would like to stress that the design generated by far is not a fully fledged processor; instead, it is an implementation of the abstract processor model in UML 2.0. To generate a complete processor design, we should specify additional hardware constraints in the UML model, e.g. bit widths of attribute variables, and existing IP cores as architectural choices for processor components. Fortunately, these additional specifications are doable within our framework.

8 Concluding Remarks

In this paper, we outlined an approach to stream processors modelling at a system level. A translation mechanism that produces SystemC code from UML 2.0 designs was also presented. With these completions, specifications of real time stream processors in UML 2.0 are intuitive and generating simulator descriptions via SystemC is feasible. The system level simulator in SystemC we produced from the UML model explicitly reflects the spirit of stream applications by emulating the processing of tokenized streams. This SystemC implementation is also synthesizable with Synopsys tools. As the future work, we will extend our work by modelling additional hardware features of stream applications in UML 2.0 to enable generation of fully fledged processor designs using our framework.

References

1. T. Austin, D. Burger, G. Sohi, M. Franklin, S. Breach, and K. Skadron. The simplescalar architectural research tool set. In *Available online*, <http://www.cs.wisc.edu/mscalar/simplescalar.html>, 1998.
2. F. Bruschi. A systemc based design flow starting from uml models. In *The 6th European SystemC users Group Meeting*, 2002.
3. P. Green and M. Edwards. The modelling of embedded systems using hasoc. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2002*, pages 752–759, March 2002.

4. P. Green and S. Essa. Integrating the synchronous dataflow model with uml. In *Proceedings of DATE-2004.*, volume 1, pages 736–737. IEEE Press, February 2004.
5. T. Hasegawa. An introduction to the uml for soc forum in japan. In *UML for Soc Design, DAC 2004 Workshop*, Available online: <http://www.c-lab.de/uml-soc/uml-soc04/hasegawa.pdf>, June 2004.
6. I-LOGIX. Rhapsody in c++. In Available online, http://www.ilogix.com/rhapsody/rhapsody_c_plus.cfm. I-Logix Incorporation, 2004.
7. U. J. Kapasi, S. Rixner, W. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens. Programmable stream processors, *ieee computer*, August 2003.
8. L. Lavagno, G. Martin, and B. Selic. *Uml for Real: Design of Embedded Real-Time Systems*. Kluwer Academic Publishers, 2003.
9. G. Martin. Systemc and the future of design languages: Opportunities for users and research. In *The 16th Symposium on Integrated Circuits and Systems Design*. IEEE Press, 2003.
10. G. Martin, L. Lavagno, and J. Louis-Guerin. Embedded uml: A merger of real-time uml and co-design. In *The 9th Int'l Symp. on Hardware/software Codesign*, pages 23–28, March 2001.
11. MPEG. Mpeg audio resources and software. In Available online, <http://www.mpeg.org/MPEG/audio.html>. Moving Picture Experts Group, 2004.
12. MPEG. Mpeg video resources and software. In Available online, <http://www.mpeg.org/MPEG/video.html>. Moving Picture Experts Group, 2004.
13. B. Serebrin, J. D. Owens, C. H. Chen, S. P. Crago, U. J. Kapasi, P. Mattson, J. Namkoong, S. Rixner, and W. J. Dally. A stream processor development platform. In *Proceedings of ICCD-2002*. IEEE, September 2002.
14. W. H. Tan, P. S. Thiagarajan, W. F. Wong, Y. Zhu, and S. K. Pilakkat. Synthesizable systemc code from uml models. In *UML for Soc Design, DAC 2004 Workshop*, Available online: <http://www.comp.nus.edu.sg/~zhuyx/usoc04.pdf>, June 2004.
15. W. Thies, M. Karczmarek, and S. Amarasinghe. Streamit: A language for streaming applications. In *Proceedings of the 2002 International Conference on Compiler Construction*. Springer-Verlag LNCS, April 2002.
16. Q. Zhu, A. Matsuda, and M. Shoji. An object-oriented design process for system-on-chip using uml. In *The 15th Int'l Symp. on System Synthesis*, pages 249–254. ACM Press, 2002.