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

Yongxin ZHU
School of Computing
National University of Singapore
zhuyx@comp.nus.edu.sg

Zhenxin SUN
School of Computing
National University of Singapore
sunzhenx@comp.nus.edu.sg

Alexander MAXIAGUINE
Computer Engineering and Networks Laboratory
ETH Zurich
maxiagui@tik.ee.ethz.ch

Weng-Fai WONG
School of Computing
National University of Singapore
wongwf@comp.nus.edu.sg

Abstract

While enabling fast implementation and reconfiguration of stream applications, programmable stream processors expose issues of incompatibility and lack of adoption in existing stream modelling languages. To address them, we describe a design approach in which specifications are captured in UML 2.0, and automatically translated into SystemC models consisting of simulators and synthesizable code under proper style constraints. As an application case, we explain real time stream processor specifications using new UML 2.0 notations. Then we expound how our translator generates SystemC models and includes additional hardware details. Verifications are made during UML execution as well as assertions in SystemC. 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 [14], StreamC [12], and NVIDIA Cg [11] with 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 modelling.

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 [7]. 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, which are available in products such as Rational Rose Real Time and I-Loigx's Rhapsody [5].

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 [6] 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. 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 collec-

[*]Acknowledgement: This research was partially funded under A*STAR Project 022/106/0043.

tion 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 [7]. Two instances of UML extension and translation are [2, 15]. Some studies exploited the possibility to describe a synchronous dataflow model [4] and a model for incremental designs [3] in UML. A previous effort [13] focused on hardware generation from UML models without explicit UML specifications of stream processing. Another recent work [10] mainly studied the model driven architecture approach via SystemC using UML 1.x.

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 workflow that starts with UML 2.0, then SystemC as an intermediate language. Using this workflow, we specify stream processors and applications in UML at an abstract level with Rhapsody. From the UML specifications, a stream processor simulator in SystemC is generated. Here are the distinct features of our work: (a) to ensure the maximum reusability and compatibility, we simply use existing UML 2.0 notations without introducing any new ones; (b) in our UML specification, we explicitly describe real time stream application features; (c) due to the asynchronous nature in video and audio stream processing, data packets are processed at various rates by multiple processing elements. 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 workflow, present the translator which can produce SystemC code from restricted Rhapsody designs, and illustrate the workflow with a case study. Some concluding remarks will follow the discussions.

3 Requirements of Real Time Stream Models & UML 2.0 Notations Needed

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.

Modelling Requirements: Flows require entrances and exits at processing elements. These gateways are also to be represented in abstraction with distinct attributes.

There is also a need to hierarchically specify the computation complexity inside processing elements. Since video and audio stream protocols have become very sophisticated, the complexity of calculation grows so hard that a non-hierarchical representation would be not manageable.

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 classic logic 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 is 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.

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*.

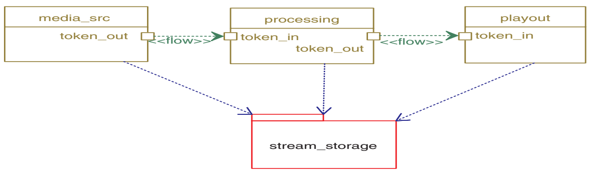

Figure 1. Functional Block Modelling of a Stream Processor

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. We expect these two notations 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 hard computation complexities, "hierarchical statecharts" are needed. Nested statecharts are allowed to represent sub-machines as part of the 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 an example in Fig. 5.

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 processing blocks. 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. To check the validity of a design in simulation of the model of the design, a constraint is an assertion rather than an executable mechanism. It does not exist in the final description for hardware synthesis.

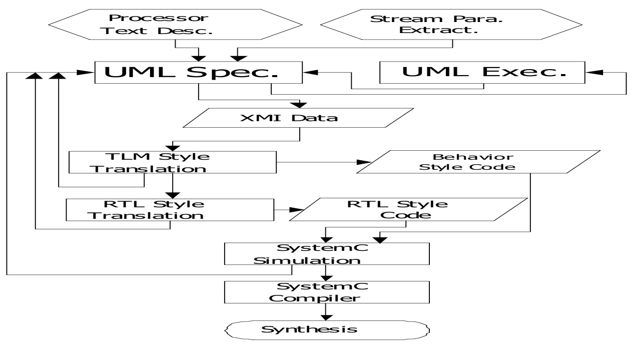

Figure 2. A UML 2.0 Based Design Workflow

4 A UML 2.0 Based Design Workflow

We propose and implement a design workflow based on UML 2.0 with SystemC as its intermediate language in Fig. 2. In this design workflow, our own efforts include the steps 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 Metedata 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 workflow, designers usually work with the UML specifications only as the rest steps of the workflow 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 as early verifications.

The UML specifications are 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 [13] can be compiled by the Synopsys CoCentric SystemC compiler into gate level netlists in Verilog, which are synthesizable into programmable architecture implementations. The coding constraints such as using static memory instead of dynamic memory, struct rather than union, and call by value instead of call by reference, are imposed by the synthesizable subset of SystemC.

In the meantime, verifications are carried out at multiple levels, i.e., UML specification, TLM style translation, RTL style translation and SystemC simulation.

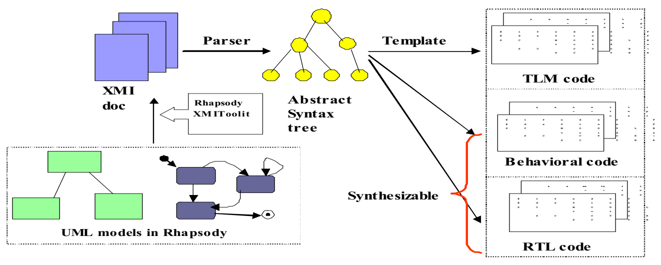

Figure 3. The Translation Process

5 The UML2Code Translator

UML 2.0 is closer to SystemC than its predecessor since some of the new concepts have direct counterparts in SystemC. For example, a meta-class corresponds to a module in SystemC. During the code generation, we apply the following mapping rules: (a) all components in the structural diagram, which are meta-classes, are translated into a "sc_module"; (b) the port type is determined by the naming convention of the port; (c) the statechart in each active object is translated into a process whose method name is "entry". In this method, we map each statechart into an unclocked "sc_thread" for TLM style translation. For RTL style translation, the mapping is a clocked process "sc_method", which will be synthesizable; (d) the definitions of message structures are encapsulated in flow whose counterpart is *sc_interface*.

Besides the above general rules, the translator also adds platform dependant hardware details such as considerations of reset, clock, and variable types. Additional constraints such as cache and memory configurations can be added to make the translator generate more hardware features.

Figure 3 shows the translation workflow of UML2Code. We start with a UML 2.0 model in Rhapsody. The design model is accepted by Rhapsody's XMITookit to generate corresponding XMI documents as inputs to our translator. XMITookit preserves all the model information, and it is a textual representation of the UML model. The generator

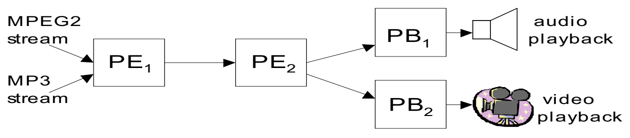

Figure 4. An Abstract Stream Processing SoC

parses the XMI documents and builds the internal representation of the model. The syntax tree is further processed to compute and store some relations. 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 at different levels: TLM, Behavioral and RTL level. Behavioral and RTL code can be further synthesized into gate-level netlists in Verilog/VHDL.

6 A Case Study

Video and audio decoders are typical streaming applications. Examples include the MPEG2 video decoder [9] and MPEG Audio Layer 3 (MP3) decoder [8]. As a case study, we shall map these applications into an abstract SoC architecture, then implement it using our design workflow.

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 processing element in the SoC, IDCT and MC to the second element.

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, (7) frequency inversion, and (8) synthesis polyphase filter. We assign the earlier 4 steps to the first processing element in the SoC, and the rest 4 steps to the second element.

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 as the abstract architecture in Fig. 4.

To obtain the requirements of execution cycles at different stages, we use one of SimpleScalar's [1] tools, "sim-profile" configured to the default values, to run reference implementations of MPEG2 decoder and MP3 decoder separately without scheduling. For each macroblock, processor cycles at different execution stages are captured and stored into trace files. For each video or audio stream, there are a set of trace files consisting of records for stream tokens. Each record contains a token index number and token attributes such as the execution cycles.

To support real time applications, I-Logix Rhapsody is preferred to Rational Rose Real Time (RT) since the latest Rhapsody provides more support for UML 2.0 notations than Rose RT such as functional block modelling, flow and concurrent state charts. Moreover, Rhapsody can produce representations in XML Metadata Interchange (XMI) format which is not supported by Rose RT.

UML Specifications: To detail the descriptions of "flow" driven activities in Fig. 2, we use a *structural diagram* in Fig. 5 which describes 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 hence inefficient in hardware to transfer complete 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.

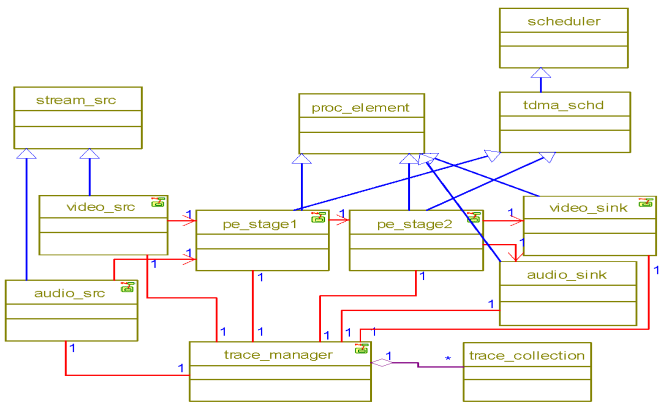

Figure 6. A Stream SoC's Class Diagram

In Fig. 6, we use a traditional class diagram to specify the inheritances and association among the components of the SoC 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 hierarchical states and concurrent calculations. For example, a hierarchical state containing concurrent submachines in the "trace_manager" is shown in Fig. 7.

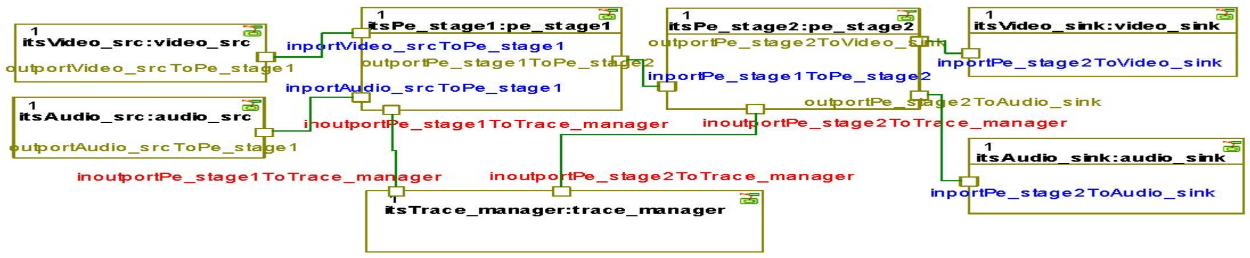
1
itsVideo_src:video_src
outportVideo_srcToPe_stage1
1
itsAudio_src:audio_src
outportAudio_srcToPe_stage1
1
itsPe_stage1:pe_stage1
inportVideo_srcToPe_stage1
outportPe_stage1ToPe_stage2
inportAudio_srcToPe_stage1
inoutportPe_stage1ToTrace_manager
1
itsPe_stage2:pe_stage2
outportPe_stage2ToVideo_sink
inportPe_stage1ToPe_stage2
outportPe_stage2ToAudio_sink
inoutportPe_stage2ToTrace_manager
1
itsVideo_sink:video_sink
inportPe_stage2ToVideo_sink
1
itsAudio_sink:audio_sink
inportPe_stage2ToAudio_sink
inoutportPe_stage1ToTrace_manager
inoutportPe_stage2ToTrace_manager
1
itsTrace_manager:trace_manager

Figure 5. A Structural Diagram of a Stream Processing SoC

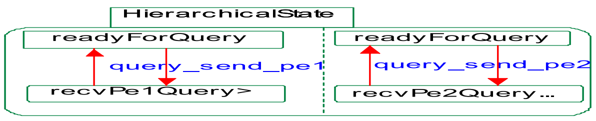

Figure 7. A Hierarchical State in an Object

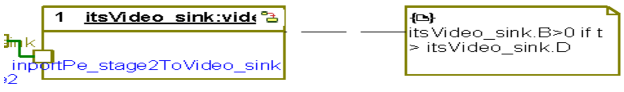

Figure 8. A Constraint Anchored to an Object

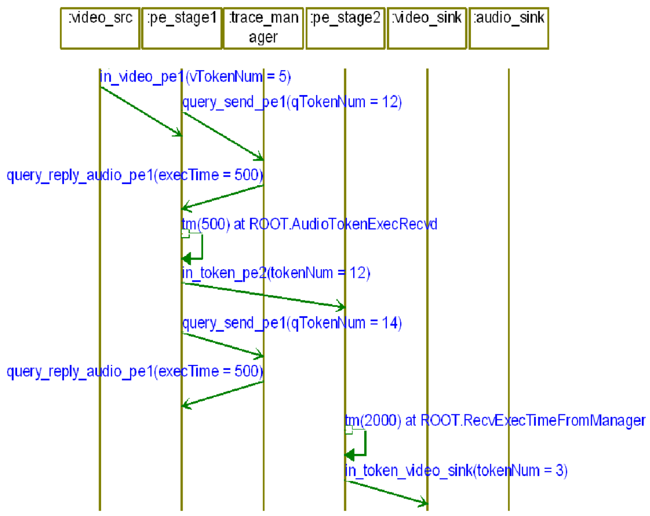
:video_src
:pe_stage1
:trace_manager
:pe_stage2
:video_sink
:audio_sink
in_video_pe1(vTokenNum = 5)
query_send_pe1(qTokenNum = 12)
query_reply_audio_pe1(execTime = 500)
tm(500) at ROOT.AudioTokenExecRecvd
in_token_pe2(tokenNum = 12)
query_send_pe1(qTokenNum = 14)
query_reply_audio_pe1(execTime = 500)
tm(2000) at ROOT.RecvExecTimeFromManager
in_token_video_sink(tokenNum = 3)

Figure 9. Communications among Modules

Similar to assertions, constraints have to be checked constantly during processing of the streams. A constraint example is shown in Fig. 8 for illustration. 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.

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.

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 workflow.

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 SoC 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 processing SoC.

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%.

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*" in Fig. 5.

Regarding the code efficiency, for the settings in the example, the size of the generated SystemC code is around 3100 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 processing element "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;
} ...
}
```

To represent the semantics of assertions, we propose constructs *AFTER, ALWAYS, EVENTUALLY* and *NEVER*, which are new to SystemC standards. An intuitive implementation of an assertion in SystemC is as follows:

```
//AFTER(t>itsVideo_sink.D)
//      ALWAYS(itsVideo_sink.B>0)
if((t>itsVideo_sink.D)&&
  !(itsVideo_sink.B>0)){
  cout<<"Assertion 1 violated!"<<endl;
  cout<<"Tot. # of violation:"
      <<vcount<<endl; vcount++;
}
```

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 workflows.

Though the generated design is synthesizable, we would like to stress that the design generated by far is not a full-fledged processor; instead, it is an implementation of the abstract SoC model in UML 2.0. To generate a more complete 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. These additional specifications can be processed by our translator to generate more details in SystemC models, and consequently further hardware particulars.

7 Concluding Remarks

In this paper, we outlined an approach to modelling stream processing SoC 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 processing SoC platforms in UML 2.0 become intuitive and generating simulator descriptions via SystemC is feasible. The simulator in SystemC we produced from the UML model explicitly reflects the spirit of stream applications by emulating the processing of tokenized streams. A case study shows 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 full-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 *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*, pages 736–737. IEEE Press, Feb. 2004.

[5] I-LOGIX. Rhapsody in c++. In *Available online, http://www.ilogix.com/rhapsody/rhapsody_c_plus.cfm*. I-Logix Incorporation, 2004.

[6] 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.

[7] L. Lavagno, G. Martin, and B. Selic. *Uml for Real: Design of Embedded Real-Time Systems*. Kluwer Academic Publishers, 2003.

[8] MPEG. Mpeg audio resources and software. In *Available online, http://www.mpeg.org/MPEG/audio.html*. Moving Picture Experts Group, 2004.

[9] MPEG. Mpeg video resources and software. In *Available online, http://www.mpeg.org/MPEG/video.html*. Moving Picture Experts Group, 2004.

[10] K. Nguyen, Z. Sun, P. Thiagarajan, and W. Wong. Model-driven soc design via executable uml to systemc. In *The 25th IEEE Int'l Real-Time Systems Symp.*, pages 459–468, 2004.

[11] NVIDIA. Nvidia cg toolkit. In *http://www.nvidia.com/cg*. NVIDIA Corp., 2004.

[12] 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 *Proc. of ICCD-2002*. IEEE, Sep. 2002.

[13] 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, www.comp.nus.edu.sg/∼zhuyx/usoc04.pdf*, June 2004.

[14] W. Thies, M. Karczmarek, and S. Amarasinghe. Streamit: A language for streaming applications. In *Proc. of the 2002 Int'l Conf. on Compiler Construction*. Springer-Verlag LNCS, April 2002.

[15] 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, 2002.