Specification Capture based SoC Modeling

Ashwin K. Kumaraswamy, V. A. Chouliaras, T. R. Jacobs, and J. L. Nunez-Yanez

Abstract— **We propose a new EDA tool flow which allows SoC architects to utilize an object-oriented approach in the development of streaming next generation SoC's including shared-memory, cache-coherent, single-chip multiprocessors. The tool will allow the visual definition of a complex computation kernel/SoC through instantiation of parametric IP such as processors, SDRAM controllers, DMA engines, on-chip buses, switch matrices and coherency directories, coprocessors, etc. Such IP is captured either at the specification level via UML, at the model level (SystemC, SpecC or ANSI C) or at the implementation level (RTL VHDL or Verilog). The unified environment then simulates the whole system during the architectural refinement phase and in the process, a near-optimal solution in terms of area, power and performance, is achieved. Finally, the output of the tool consists of a cycle-accurate executable model accompanied by the system RTL.**

I. Introduction

We present preliminary results of a new EDA flow which unifies the *specification capture* and *design modeling*. Current, tools in the market namely Incyte, Mageillem, Visual Elite have provided solutions for specification optimization, graphical design entry and hardware-software partitioning to help designing of high performance IPs , but clearly there is still a lacking of a complete robust flow which helps the designers to take designs from specification to silicon and there is been a concrete effort to develop such a flow.

We propose the unification of the *specification capture*, *modeling* and *optimization* of very high performing streaming system-on-chip designs through a unique combination of technologies. The overall flow can be classified into two parts namely the unified specification capture and the unified simulator.

At present the flow is partially completed with work being done on the specification capture stage so as to accommodate the existing system level design languages (SLDL) such as SystemC and SpecC.

We use UML as the front end specification capture format and convert the UML to a known SLDL like SystemC or SpecC. This translation is being performed using a unique combination of technologies and we have a prototype of translation kit from specification to SLDL(SpecC). Simultaneously work is being performed to accommodate SystemC, as it is becoming more widespread through tools such as SystemC complier.

II. Unified Specification

We propose a methodology that can transform UML models into a known System Level Design Language (SLDL) (SystemC/SpecC). In other words, UML model acts as a “wrapper” to the SLDL methodology. In UML each aspect of the SLDL’s methodology can be modelled and refined. The standardization of UML provides a base to revise the approaches to combine SLDL with object oriented analysis and design techniques (OOAD) techniques. One of the main directions for the joint application of SLDL and UML can be identified as modelling SLDL constructs in UML. This direction serves mainly the idea to make large SLDL specification better understandable and to give additional information (e.g. inheritance hierarchies, dependencies, pattern structures) for documentation purposes or as additional implementation information. UML is mapped onto the SpecC methodology. Uniqueness, to this new methodology, is that the UML representation of the system is separated from the underlying methodology. This helps in unifying the ways a system can be represented in UML without worrying about the way it will be implemented.

A Hardware/Software co-designed system can be specified through the concepts of behaviours that interact via channels through ports and interfaces. There is a clear separation between computation and communication where behaviours model computation, and communication is modelled by using shared variables and/or channels [5]. Computation will consist of behaviours and their definitions. Communication will consist of ports, channels and interfaces. (Interfaces can also be used in the modelling of computation [3, 8].)

Modeling of Computation

In UML, the behaviours are modelled as classes. The local variables and the functions are also modeled within the class in their respective positions. A composite behaviour will contain instances of other behaviours. These compositions can be modeled using associativity. When breaking down behaviour into sub-behaviours, for structural hierarchy, generalizations can be used. There can be two types of hierarchy: structural and behavioral. Structurally, behaviours can be broken down

into sub-behaviours and these into sub-behaviours, and so on. Designs are specified in a hierarchical manner using top-down functional decomposition (behavioral hierarchy). Both these hierarchies correspond to the concept of generalization and associativity in UML [5, 8].

Modeling of Communication

Affiliations
Ashwin K.Kumaraswamy, V. A. Chouliaras and Tom R. Jacobs are with Dept of Electronic and Electrical Engineering, University of Loughborough,Loughborough,UK,email:ashwink.ctes2004b@cselondon.com
J. L. Nunez-Yanez is with the Dept of Electronic Engineering, University of Bristol, Bristol, UK

To model interfaces, UML's interface notation is used. An interface is like an abstract class that consists of a set of method declarations. Interfaces can also be placed in a hierarchical fashion. Behaviours can, optionally, "realize" single or multiple interfaces. The channel or the behaviour that realize the interfaces should supply the definitions for the method declarations.

The stereotype, *<<channel>>*, is used to represent a class as a channel. Channels are also modelled in the same manner as behaviour. Ports can be modelled in two ways. A port can either be a simple variable or another Interface or Class. In order to identify an object as a port, the <<port>> stereotype is used. If the port is declared as a simple variable of type type1, the variable declaration in UML will be as

```
name: type1 <<port>>
```

The <<port>> stereotype helps in identifying certain variables and also associations as ports, rather than local variables or instances respectively.

Modeling of Execution

The main problem in designing a system is the modeling of execution or show parallelism i.e., to represent behaviors that will be executing in sequence, parallel or pipelined. There are two different ways of specifying this. It is well known that in UML different views are meant for different activities of modeling. Thus, these considerations have to be mentioned in more than one of the views. In the static view (class diagram) we annotate these using stereotypes. This is very helpful, because the class diagram shows the static structure of the system. The problem of showing parallelism in the execution model can be solved through composition. Leaf behaviour, by itself will only perform its operations sequentially. If a component has to be modeled to execute in parallel or pipelined mode, then its behaviour can be further reduced into separate classes and its objects will be composed into the main component. These sub-behaviours can then be modeled to run in parallel or pipelined mode by specifying the mode of execution to the composite behaviour (main component). This can be done in the static view of the model. The actual execution of the composite behaviour can be modeled in detail, using State chart diagrams and/or Sequence diagrams.

It was concluded that the State Machine view and the Activity view of the UML had enough notations specified to describe the internal behaviours of any component. Clocks can also be modeled as behaviours and can be made to generate events. These events can be used in other views to specify the timing characteristics of the system.

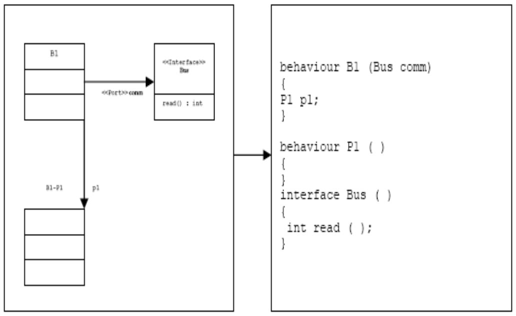
Transformation of Static View

Since SpecC is not an Object oriented language, there is no way of representing object hierarchies. Thus generalization is used to model behavioral hierarchy. In other words, behavioral hierarchy is modeled as a composition of multiple behaviours, according to the SpecC methodology. Therefore generalizations are transformed in the same manner as associations. A static view is shown in figure (1).

Transformation of State Machine View

The state machine view describes the dynamic behaviour of objects. Each object is treated as an isolated entity that communicates with the environment by detecting events and responding to them [7]. A state machine is a graph of states and transitions. Usually a state machine is attached to a class and describes the response of an instance of the class to events that it receives.

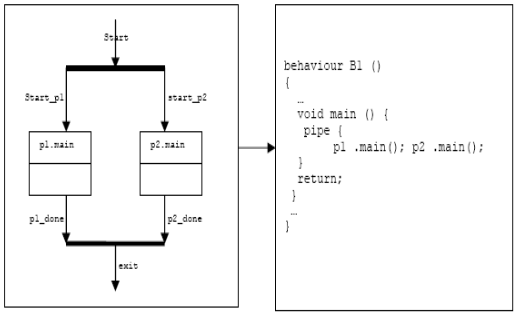
A State Machine view is used to model the internal behaviour of an object of a class. A state machine contains states that are connected by transitions. Each state is defined as some unit of time in which the object stays and performs certain operations, whereas transitions are instantaneous, i.e. they occur at zero time. When an event occurs, it may cause the firing of a transition that takes the object to a new state. When a transition fires, an action attached to the transition may be executed. Theoretically, this execution period is zero. State machines are shown as a state chart diagram (Figure 2).



```
behaviour B1 (Bus comm)
{
P1 p1;
}

behaviour P1 ( )
{
}
interface Bus ( )
{
 int read ( );
}
```

Fig 1. Static View



```
behaviour B1 ()
{
  …
  void main () {
   pipe {
        p1 .main(); p2 .main();
  }
  return;
 }
 …
}
```

Fig 2: State Machine View

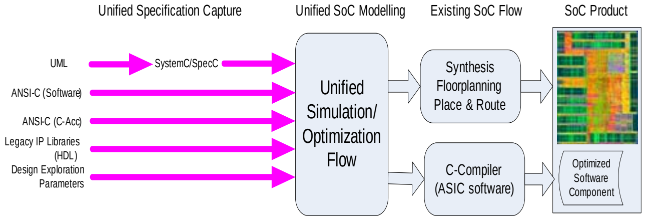

Unified Specification Capture
Unified SoC Modelling
Existing SoC Flow
SoC Product
UML
SystemC/SpecC
ANSI-C (Software)
ANSI-C (C-Acc)
Legacy IP Libraries (HDL)
Design Exploration Parameters
Unified Simulation/ Optimization Flow
Synthesis Floorplanning Place & Route
C-Compiler (ASIC software)
Optimized Software Component

FIG 3. OVERALL FLOW

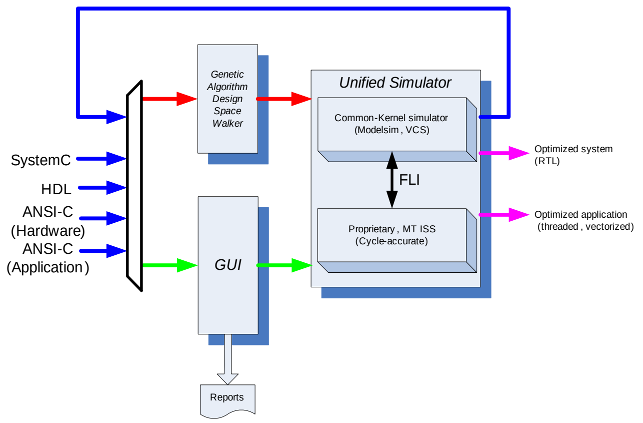


Fig 4: Internal flow path

III. THE SIMULATOR FLOW

Fig.3,4 depicts a high-level view of the proposed SoC Design Framework tool that we are currently developing. It consists of the input interface which can accept silicon and software IP in a number of forms including SystemC, standard HDL (VHDL and Verilog), cycle-accurate C conforming to the internal API and finally, standard C for the application. These elements are slotted in system-defined and used-defined 'Stencils' from which on they are available for manual or automatic instantiation and design space exploration.

A. *Core Simulation Engines*

The primary simulation engine is based around a parameterized, multi-context, Instruction Set Simulator (MT-ISS) derived from the Simple scalar computer architecture research tools. The default ISS has been re-architectured to allow the instantiation of a number of processor contexts and additional programmer-visible state for multi-processor (MP) synchronization. The simulator can be considered as an Exclusive-Read, Exclusive-Write (EREW) Parallel RAM (PRAM) machine. Architectural hooks are in place to allow interfacing to a cycle-accurate (CA) back-end. In this way, the ISS is dynamically producing (short) instruction traces which are (dynamically) consumed by the CA back-end. In the process, various parameters are evaluated such as the Clocks-Per-Instruction (CPI) ratio per CPU, bus utilization, ICache and Dcache misses, pipeline stalls due to dependencies amongst others.

The MT-ISS is one of the core simulator of the flow and drives both the programmable and non-programmable C-based simulation models along with being used for software development.

The second simulation engine is an industry-standard tool such as Mentor Graphics Modelsim. It interfaces to the cycle-accurate infrastructure via the Foreign Language Interface (FLI) and allows for the modeling of legacy IP (VHDL, Verilog) and the primary output of the specification-capture front-end which is described in System-C.

B. *Manual and Automatic Flows*

There are two major flows (feedback loops) in the tool. The first is based around a GUI solution which is used to instantiate silicon IP blocks and application software components from the *IP stencils* on to the SoC *canvas*. The contents of those stencils can be 'dragged' onto the SoC area thus, incrementally building up and simulating the SoC model. We make no distinction as to whether the stencils contain synthesizable Silicon IP or CA models as the core simulators permit their arbitrary mix. This is of paramount importance in the modeling of highly-complex, future SoC architectures.

Experimentation takes place after the SoC has been 'drawn' and it's memory map established and populated. The feedback loop of Fig. 4 illustrates the manual or automatic refinement process, from SoC specification to performance closure and clearly illustrates the synergy between the MT-ISS, CA back-end and industrial simulators in providing a unified framework for SoC modeling.

A further route exists where the process is fully automated. In this case, a genetic-algorithm (GA) design space walker takes over the refinement process of the SoC once the initial allocation of programmable and non-programmable resources has happened.

C. *Embedded CPU Stencils*

The primary programmable engine used is based on an open-source, 32-bit RISC CPU with an extended Instruction Set to allow for hardware barrier synchronization. In addition, the programmer's model was extended to include a unique, non-programmable, processor ID field which is used to identify the executing CPU to a software thread.

As we are targeting primarily Data-Level-Parallelism (DLP), we have augmented the microarchitecture of the Leon-2 CPU to include a custom coprocessor channel in order to communicate to very high performance, tightly-coupled vector coprocessors. Typical transactions along this new interface are depicted in the Fig 5 shows a coprocessor data operation on cycle 1 followed by a host-to-coprocessor register transfer on cycle 2. In cycle 3, a coprocessor register is requested by the RISC processor but due to internal stall conditions, data are made available one cycle later than the expected time (cycle 5 instead of cycle 4). During that time, the main processor is held with the *holdn* signal. Finally, a second read operation, this time directed to Coprocessor 1, is initiated in cycle 6. Results are made available to the main pipeline in cycle 7.

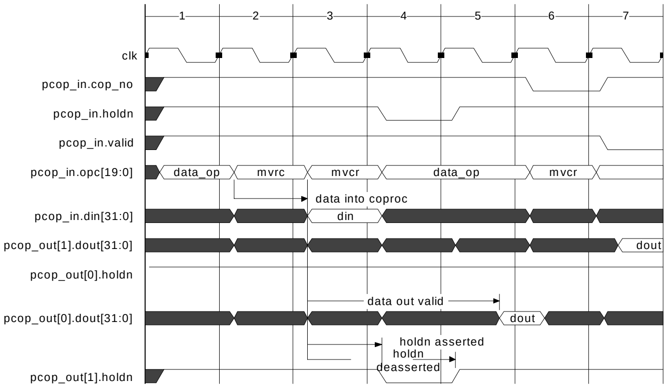


Figure 5: Typical Coprocessor Channel Transactions

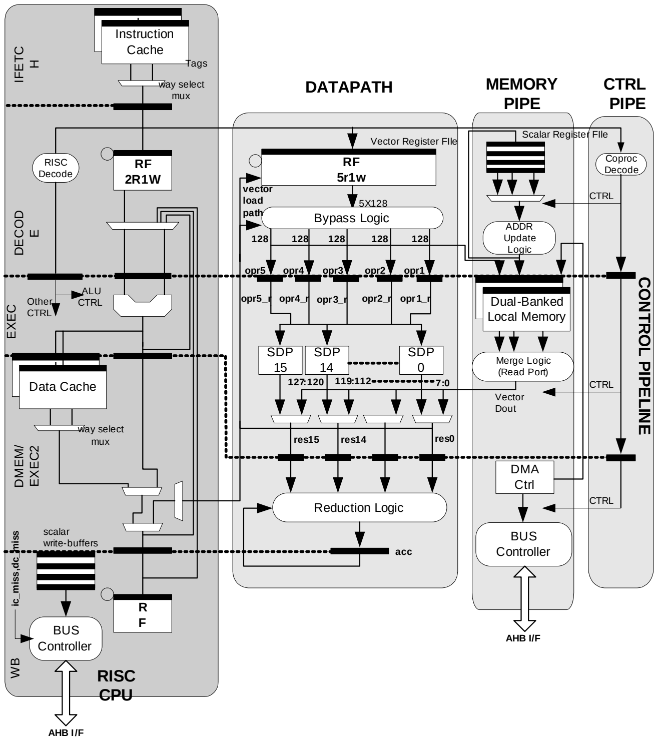


Figure 6: Scalar CPU and Vector Accelerator

Fig. 6 shows the combined processor-coprocessor microarchitecture which includes an instance of a parametric vector accelerator implementing three custom instructions for the data-parallel sections of the MPEG-2 encoder attached to the scalar CPU which is a standard 5-stage design. From the diagram, instructions are fetched from the multi-way, set-associative instruction cache and clocked into the instruction register. Decoding takes place in the DECODE stage with the RISC register file accessed at the falling edge of the clock. The bypassing logic in DECODE determines whether register file data or internally pipelined results are clocked in the ALU input registers. During EXEC, the ALU operation is performed and a virtual address is computed. Scalar data cache access takes place during DMEM/EXEC2 and scalar results return to the RISC pipeline during this cycle. Finally, results are clocked into an intermediate register prior to committing to the processor register file. The processor incorporates configurable data and instruction caches the former in a write-through configuration with no-write-allocate policy. Both caches are refilled over the on-chip bus via the bus controller.

D. *Interconnect Stencils*

We are targeting primarily the SoC modeling and implementation domain. We therefore have included support for multi-layer AMBA (AHB) based on the infrastructure provided by the Opensource CPU, augmented with the hardware synchronization primitives. A typical scenario of a paremetric, cache-coherent, SoC MP is depicted in Fig. 7.

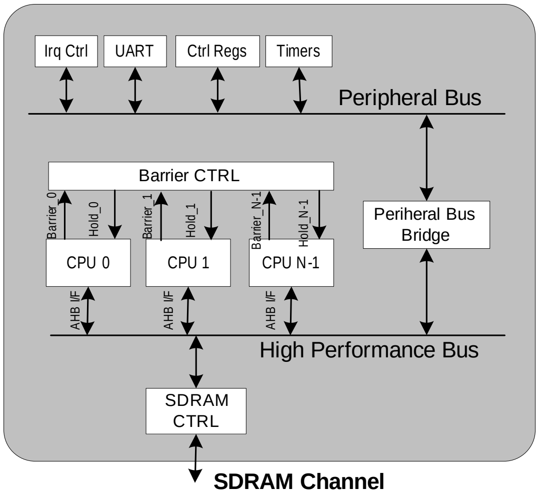


Figure **7:** Typical bus based parametric SoC multiprocessor

Further development is underway to allow for very high bandwidth interconnects (other than a hierarchy of buses) to be utilized. In this case, distributed coherency directories are utilized to ensure that the CPU caches remain consistent.

E. *Streaming unit stencils*

Prior research into statically-configurable (off-line configurable) vector/SIMD accelerators has successfully concluded that such units are of paramount importance in achieving performance closure in a consumer/media SoC. The complexity-metric reduction due to the vector instructiosn implemented via such tightly-coupled coprocessors for the case of MPEG-2 TMS reference software is as shown in Fig 8.

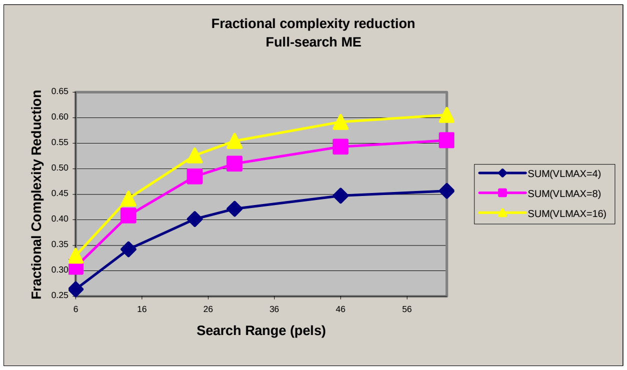


Figure8: MPEG-2 DLP benefit

IV. Results

To establish the proposed methodology, we studied a multi-threaded implementation of the MPEG-2, TM5 reference video standard [mpeg.org]. The encoder was initially profiled in order to identify the most compute-intensive parts at function-level granularity. The complexity metric used was the dynamic instruction count of the application when compiled for a MIPS II-like CPU and executing on a single-context simulator.

From Fig. 9, the most compute-intensive function was identified as the inner loop of ME (*DIST1*). This function computes the error of the current macro block over all macro blocks in the search window of the reference frame and its complexity ranges from 52% to 73% of total dynamic instruction count for a search window of 7 to 63 pels respectively. The second most complex function was the forward-DCT computation (*FDCT*) with a complexity metric ranging between 2.1% and 21% of the total dynamic instruction count. *FullSearch* is the wrapper function around the low-level *DIST1* and implements the default ME algorithm. Its complexity ranged from 3.5% to 23.2% of the total complexity. This is the level at which we applied our threading technique as this allows the utilization of less complex, algorithmic ME methods such as three-step search [10] and four-step-search [11] in the parallelized encoder.

The performance of the threaded MPEG-2 encoder was evaluated in a relatively slow, vertical-moving sequence (Snowfall) and a very fast, circular-moving sequence (Rotating City). We used Full-Search ME which is the default algorithm in the reference MPEG-2 code.

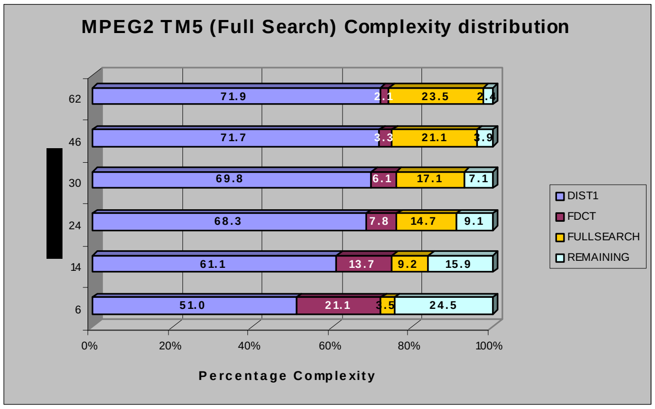


Figure 9: MPEG-2 TM5 Algorithm Profiling

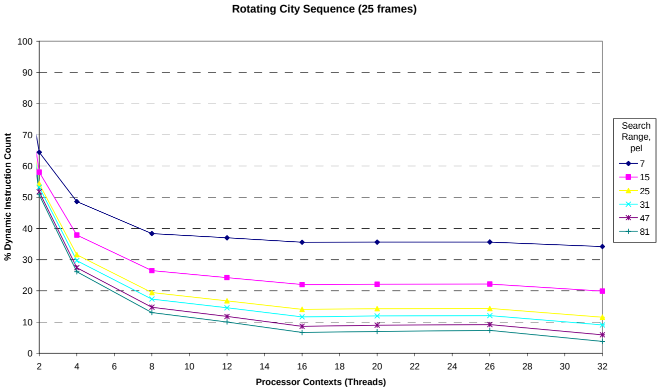


Figure 10: Theoretical Performance (Circular Motion)

Fig 10 demonstrate a significant reduction in the complexity metric both when the number of processor contexts is increased and when the search window range is increased. The complexity improves no further once the number of processor contexts exceeds 32. Performance saturation at 95% complexity-metric reduction occurs when a threaded loop is executed only once. Results compare favorably with prior studies which achieved a 65%-70% complexity-metric reduction at a search range of 62 pels, for a 128-bit DLP architecture

V. Comparison of Like Minded EDA Tools

Tools	What is does	What it does not
Incyte	1.Specification Optimization System 2.Designer specification oriented 3.Tool shows the potential problems in the design thus helping in fast design time	1.Design Modelling in not done 2.No Hardware-software partitioning 3.Not really a integrator platform based on any Bus standard
Magilleium	1.Graphical Design Entry based tool 2.Integration Platform based on AMBA 3.Transactional RTL Builder, full support for SystemC 4.Good Verification system in place	1.No option for design modelling and design exploration at the highest level. 2.Basically choosing blocks from the existing library
Visual Elite	1.Graphical Design Entry based 2.Performs Hardware-Software partitioning 3.Helps in design using specific Microprocessors	1.Not a design specification capture and design modelling based tool
New flow, specification capture based design modeling	1.Graphical Design Entry 2.Design Specification capture and architecture exploration along with direct choice from legacy IP library. 3.Integration platform with AMBA 4.Hardware-Software partitioning. 5.Supports SystemC,C RTL, SpecC	

VI. Conclusion

We have demonstrated the new EDA flow proposal and submitted preliminary results of the flow that we have developed on two fronts.

1. UML diagrams being converted to the desirable SLDL.
2. The simulation performance of the unified simulation flow.

VII. Future Work

To integrate the tool flow so as to make the automation, we have been able to demonstrate that such a tool flow is conceivable.

- Future work will quantify on cycle effects of the bus-based configuration as well as the benefit of local scratchpad memories over the parametric Data Cache.
- Complete the UML to SystemC in the same lines as SpecC
- Develop the GUI for graphical design entry.
- Develop the automation as envisioned.

References

1. M. Keating and P. Bricaud, “Reuse Methodology Manual for System-on-Chip designs, 2nd Edition, Kluwer Academic Publishers, Norwell 1999.

2. F. Balarin *et. al*, “Hardware-Software Co-Design of Embedded systems, The POLIS approach,” Kluwer Academic Publishers, 1997.
3. D. Gajski, J.Zhu et al. “SpecC: Specification Language and Design Methodology”, *Kluwer Academic Publishers*, 2000.
4. Rainer Dömer, Daniel D. Gajski, Andreas Gerstlauer, “SpecC Methodology for High-Level Modeling,” 9th EDP IEEE/DATC Electronic Design Processes Workshop 2002.
5. Object Management Group, Omg unified modeling language specification version 1.3, June 1999.
6. D. E. Lackey, “Applying Placement-based Synthesis for On-time System-on-a-Chip Design”,*IEEE Custom Integrated Circuits Conference*, 2000, pp. 121-124.
7. Object Management Group, Omg-xml metadata interchange version 1.2, January 2002.
8. J. L. Diaz-Herrera, An isomorphic mapping for SpecC in UML, Internet: http://ist.unibwmuenchen. de/GROOM/OMER-2/papers/OMER2-
9. DiazHerrera.pdf, 2000, SPSU-CS TR 2000.
10. Sikora T, "MPEG Digital Video--Coding Standards," IEEE Signal Processing Magazine, Vol. 14, No. 5, September 1997, pp. 82—100.
11. Motion Picture Experts Group http://www.mpeg.org
12. V. A. Chouliaras, J. L. Nunez, Fabrizio. S. Rovati, Daniele Alfonso ‘A multi-standard video coding accelerator based on a vector architecture’, Proceedings of the IEEE International Conference in Consumer Electronics (ICCE 2005), Las Vegas, Nevada, USA
13. Shen K, Delp E J, "A parallel implementation of an MPEG encoder: faster than real-time!", In Proceedings of the SPIE Conference on Digital Video Compression: Algorithms and Technologies, pp. 407-418, San Jose, California, 5-10 February. 1995.
14. “The Leon-2 processor User’s manual, XST edition, ver. 1.0.14”, http://www.gaisler.com
15. Theo Ungerer, Borut Robič, Jurij Šilc, “A suervey of processorw with explicit multithreading”, ACM Computing Surveys (CSUR), Volume 35 Issue 1, March 2003
16. SimpleScalar LLC http://www.simplescalar.com/
17. Martinez J. F., Torrellas J “Speculative synchronization: applying thread-level speculation to explicitly parallel application” ACM SIGARCH Computer Architecture News, 30, 5, pp.18-29
18. Zeng and Liu “A new 3 step search Algorithm for Block Motion Estimation”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 4, No 4, Aug. 1994.
19. Lai-man Po and Wing-Chung Ma, “A novel four step-search algorithm for fast block motion estimation”, IEEE Transactions on Circuits and Systems for Video Technology, vol. 6, pp. 313-317, 1996.