SING: A Multiprocessor System-on-Chip Design and System Generation Tool

Marc Branchaud, Daniel Shapiro, Vishal Thareja, Srivatsan Vijayakumar and Miodrag Bolic
Computer Architecture Research Group
School of Information Technology and Engineering
University of Ottawa
800 King Edward Ave, P.O. Box 450, Stn A,
Ottawa, Ontario K1N 6N5

ABSTRACT

Increasingly complex embedded systems are being designed onto single chip systems that contain multiple parallel processing elements and memories. The design and implementation of these multiprocessor on-chip architectures is time consuming, delaying the time to market. Manual hardware design is also error-prone, requiring careful verification and further delaying the time to market. In order to solve the aforementioned problems, an open source VHDL generation tool called SING was created to design and implement heterogeneous multiprocessor system-on-chip designs. The requirements for the tool which are partially or fully addressed are: 1. use of existing and open-source components, 2. automated generation of VHDL, 3. Eclipse framework, and 4. simulation capabilities. The tool comes with a graphical user interface (GUI) and an application programmer interface (API), and exports VHDL with accompanying documentation in accordance with the WISHBONE B.3 standard. The exported VHDL can then be prototyped on an FPGA platform. Even though some similar tools exist, there are not many papers that present detailed internal design of the tools. We believe that our detailed description of the object oriented implementation will be helpful to designers of similar systems.

Categories and Subject Descriptors

Algorithms, Design, Documentation

General Terms

Multiprocessor, System-on-Chip, Interconnection Networks

1. INTRODUCTION

Modern embedded systems, from simple controllers to complex integrated multimedia cell phones, are designed using a wide range of processor architectures and bus standards. In recent years, the complexity of these embedded systems has increased while the time to market window has decreased. Under such competitive stresses, Electronic System-Level (ESL) design tools have emerged which promise to enable fast implementation of complex systems based on high-level system descriptions and software. Following this approach, a tool called Simulator and Interconnection Network Generator (SING) was designed and implemented. SING is an Eclipse based framework that generates synthesizable VHDL for an architecture specified through a GUI or an API. The IP blocks used in the design are precompiled parameterized hardware blocks. The bus architecture used by the tool follows the WISHBONE B.3 standard, and the processors are LEON3 based on the SPARC V8 architecture [13, 19]. Memories can be either instantiated by using Altera's Library of Parameterized Modules or Xilinx's Core Generator IP.

The main contributions of this paper can be summarized as follows: We have implemented a new integrated framework for graphical and textual representation of system-on-chip designs. Automatic generation of synthesizable HDL code was integrated into the framework. A system can be designed from a high level of abstraction (Java) using advanced Object Oriented Programming (OOP) methods. SING allows for modular design and makes it relatively easy to add different simulation levels for design space exploration.

SING is currently being used as a subsystem for a larger project called SHIRA that deals with hardware/software co-design. SHIRA is an extension of the COINS compiler [23]. The requirements for SING came out of that project's need for a synthesis tool. The requirements were to produce a synthesizable and open source embedded system and all of its subcomponents along with documentation, to interface WISHBONE and AMBA components in the design, to allow for both simulation of the parallel system as well as its implementation, and to enhance the design process by hooking into the Eclipse Integrated Development Environment (IDE). Furthermore the tool had to support iterative design space exploration by exposing an API for the model. We explore the design space of multiprocessor embedded systems from the SHIRA tool by accessing the SING API.

SING was programmed so that more processor descriptions, bus standards, and IP cores can be added to the component library by the SING's users. Because the tool is open-source to the public, researchers can delve deep into the details of the synthesis and/or simulation, and make changes to the code. Hopefully SING will be used as a subcomponent for other research projects, in much the same way as we have used it as a subcomponent of a multiprocessor co-design tool.

The idea for performing simulation of interconnection networks at various abstraction levels in SING comes from [12]. Our approach quickly exposes problems in the design using a high level behavioral model, and then detail in the model is increased to the functional simulation level once the high-level concepts are shown to be valid. In order to perform a low level simulation the user must point the tool to a VHDL compiler and simulator, while

high level model simulation is performed on Java objects which abstractly model the system hardware. The use cases for the tool were defined as: networks drawn by human actors in a user interface as in [3, 7, 8, 10], and networks specified by a program via an API for design space exploration as in [5, 6, 14, 12].

The SING tool contains the following components: a graphical user interface that allows the user to define an interconnection network between IP cores, a system generator that translates user requirements into VHDL code, an API and simulator for exploring the design space, and a document generator that exports interface information for each component synthesized by the tool.

A Java-based GUI is implemented to allow the user to define, save and open interconnection schemes. Another way to use SING is through an API which allows a program to connect SING with external tools in order to perform automated design space exploration. These aspects of SING will be explained later on in more detail.

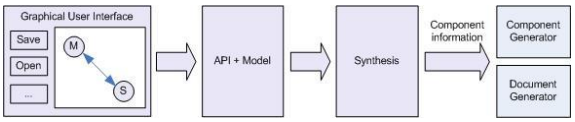

Figure 1. Relationship between SING modules.

The interconnection types supported by SING are the crossbar, shared bus, and point-to-point. Crossbar switches provide blocking communication between all processors (masters) and each memory (slaves), while shared busses enable blocking communication between one master and one slave at a time, using less chip area than a crossbar. Point-to-point connections are direct connections between masters and slaves. Dataflow connections can be implemented as a set of point-to-point connections. Dataflow is usually used to pipeline a set of processors, each performing a task in a chain and handing off the results to the next processor. Both shared busses and crossbars can be implemented with either round robin or fixed priority arbitration schemes. Priority is used to resolve the condition when two slave ports request the bus on the same clock cycle. Priorities of components connected to a bus are set by the port number that they are connected to. Lower port numbers have higher priorities. Using round robin arbitration prevents process starvation by shifting the priority of all ports by one address each time a transaction starts on the bus.

As show in Figure 1, the *System Generator* processes the user specified interconnection network into the formats required by the *Component Generator* and *Document Generator* blocks. From these blocks VHDL and WISHBONE documentation are generated. The system is implemented in a way that automatically supports connection of busses and IP-cores that have different width communication ports. Individual components are implemented in hardware based on parameterized templates. Some of the parameters of these templates include the size of the data bus, the number of master ports and slave ports on the component, the size of the address bus, an array of the address ranges that each master is responsible for, arrays of point-to-point connections for master and slave ports, the clock frequency, and the arbitration type. Of course not every component uses all of the available fields.

SING is designed to implement wrappers around processors and memories to facilitate implementation on either Altera of Xilinx FPGAs. In our case, the software for programming the multiprocessor system comes from the COINS compiler [17].

2. BACKGROUND

We begin by explaining our goals, and then proceed to show the need for a tool like SING by presenting the capabilities of off-the-shelf tools. Finally, we discuss other approaches to this problem. Several tools and academic solutions for the design and synthesis of multiprocessor systems are available. A small number of these tools are described in brief in this section.

The emphasis of our work on SING is on facilitating system synthesis, strong modularity, standards compliance, scalability, behavioral and functional simulation, and documentation generation.

Altera and Xilinx provide similar tools for generating a system with both IP cores and busses. Systems can be programmed and synthesized using these tools, and the output can be Verilog, VHDL, and other formats. For example, Altera's SOPC Builder follows the Avalon bus standard [3, 21]. A user must know ahead of time what topology is being designed in order to use the tool because peripherals are connected to the bus through a fixed array of connections. Systems designed in SOPC Builder can be simulated in the NIOS II Eclipse-based IDE [2]. SOPC Builder does not support behavioral Design Space Exploration (DSE) and it does not have an API, but it does support the use of TCL scripts for compilation and simulation. The equivalent Xilinx tool is called Embedded Design Kit (EDK) and uses IBM's CoreConnect Bus technology to connect Xilinx's MicroBlaze soft IP processors and hard IP PowerPC cores to an interconnection network along with various peripherals [8]. The Xilinx Platform Studio (XPS) interconnection network is defined via a grid in the user interface. However, in EDK the network can also be specified in a text file called an MHS file. XPS is a part of EDK and it is an Eclipse-based IDE used to design and debug software for a hardware platform created in EDK [8]. Simulations of multiprocessor systems within XPS reveal the real-time behavior of the software being designed.

Tensilica describes the design flows required to correctly identify bottlenecks or hot spots within an algorithm using Tensilica's Xtensa toolchain in [24]. The performance-demanding tasks are found by modeling the algorithm, a baseline is set for performance and area metrics, and major I/O bottlenecks and performance critical tasks are identified. Once the major tasks have been identified, the system model can be refined by utilizing multiprocessors to execute the same tasks. With the task broken up into smaller tasks on multiple processors, the smaller tasks can be profiled on each processor, further identifying any hot spots within the smaller tasks. Finally, the user can perform a System Analysis by integrating all the processor models (generated by Tensilica automatically) and confirm that the partitioning of the model was appropriate. The system can be simulated with an Instruction Set Simulator (ISS) to measure performance, check usage on shared resources, interconnect bandwidth, total silicon area and power budget.

Tensilica's Xtensa Modeling Protocol (XTMP) is an API and run-time environment to model multiprocessor systems. The API can describe communication networks, organization, and software to get an accurate simulation of the algorithm. XTMP has its own simulation engine and generates SystemC-compatible models.

XTMP can model/simulate homogenous and heterogeneous multiprocessor systems as well as complex single processor systems. The XTMP API allows designers to write custom multi-threaded simulators to model complex systems.

Tensilica's Xtensa does not utilize a traditional bus interconnect between processors and other peripherals. Xtensa follows an RTL-like direct communication with processors with the use of unique ports and queues. Tensilica argues that direct communication between processors allows for faster communication when compared against a slow, shared 32-bit bus. The ports used for direct communication can be arbitrarily wide. The maximum amount of wires is one million signals (1024 ports, each 1024 bits wide). Queues are utilized in the interconnect as an element to stream data throughout the system. Queues operate like processor registers, but without the need for load or store.

CMPware offers a tool called Configurable Multiprocessor Development Kit (CMP-DK) [7]. It is an Eclipse-based IDE used to simulate multiprocessor systems. Designs can include the Cell Broadband Engine, MIPS, SPARC, and several other processors. The tool offers very detailed processor simulation, with profiling and performance monitoring, but synthesis is not supported. As with SOPC Builder and EDK, the desired architecture must be known to the designer because DSE is not readily available.

Another tool called WISHBONE Builder creates WISHBONE compliant interconnection networks from a text file specification or a user interface [22]. This tool does not support high-level simulation, but low-level simulations can be performed using a tool such as Altera's Quartus II or Xilinx's ISE. Except for WISHBONE Builder, all of the other tools have restrictive licenses.

The features of the aforementioned tools are summarized in Table 1. Some other approaches to heterogeneous multiprocessor system-on-chip design are top-down topology generation, and fixed specification of the topology. Many approaches have been described in the literature, some using network interfaces to control and abstract communication, and some using busses with no abstraction. Some approaches to top-down design are multi-standard interconnection topology generation as in [9], automated generation of network interfaces for processors in an on-chip network as in [6], and layered approaches to topology generation using iterative DSE at various abstraction levels in order to find iteratively better topologies as in [6] and [14]. We have followed the top-down approach in a way similar to [12] by allowing the user or an external tool to come up with an algorithm for exploring the design space. Tools such as EDK and SOPC builder describe systems using a design-time user-specified topology, which assumes that the user of the tool will be better at designing interconnection networks than an automated tool [8, 3]. Although a tool can be written to iteratively modify EDK or SOPC builder specifications in search of a good topology, this would still not solve the problems of closed-source tools and restrictive licenses. Finally, an approach to HDL code generation is described in [15].

3. SING INTERNALS

SING was designed to synthesize, document and simulate embedded systems. It was implemented in Java and partitioned into three packages as shown in Figure 1. One package contains code for the GUI, another package contains the specification and design space exploration functionality, and the last package contains the documentation and VHDL generators.

The synthesis package creates the top level files of the design, builds a library of the components that will be used, and initializes and interconnects the components of the design. The generated code is relatively clean and human-readable.

Each processor, bus, custom logic, and memory in an interconnection network is referred to as a component. Master and slave ports are added to components when they are instantiated, and communication between these components occurs on channels between master ports and slave ports. Similar to the WISHBONE B.3 standard, our model forces communication channels to be explicit at the top level of the design. Each port can be connected to only one other port, and so interconnection components such as crossbars and shared busses are needed to facilitate switching of the connections between components. To connect one processor component to two bus components would require that the processor has at least two master ports. Also, a memory being accessed by two busses would need to have at least two slave ports. Bit width matching was designed into the tool so that whenever two busses of differing bit widths are connected, the least significant bits of each bus are by default connected to each other.

Table 1. Tool chain comparison summary

Tool	Eclipse based GUI	API	Document Generation	Simulation	Synthesis
Altera SOPC Builder	✓			✓	✓
Xilinx Platform Studio	✓			✓	✓
Tensilica Xtensa	✓			✓	✓
CMPware CMP-DK	✓			✓	
Wishbone Builder					
SING	✓	✓	✓	✓	

3.1 VHDL Generation

The tool facilitates easy integration of processors, busses, and memories into a model, and provides the option to compile the design at any time during the modeling process. At compile time, connector components from the object oriented model of the system are translated into VHDL modules with wrappers surrounding them. Peripherals and processors alike must be surrounded by these wrappers which standardize port names and connections. User-defined hardware blocks can be added to a design by filling out the empty connector VHDL. Alternatively, the user may extend SING by adding a model of the hardware to the SING model, along with parameterized VHDL in the compile backend. In that case the component's VHDL will be compiled by SING.

Individual components in the model are implemented in VHDL based on templates. The templates are filled in by a program after

the OOP model of the connectors is read into an array of component descriptions. The top level entity that connects the components together is synthesized by looking at the connection information provided in the components of the model and adding VHDL accordingly. The sections of the top level file are: the VHDL library import, the top level port map with signals for testing the synthesized design, signal declarations ordered by component type and name, instantiation of components and association of components and signals, and finally, assignment of signals to form the connections in the design. This code is inherently clean because the signal names are very descriptive and the details of the design are all present in the final section of the top level entity. Although our approach makes the code a bit longer, the connection information is not cluttered by component declarations or instantiations.

During compile, SING must iterate through all of the components and build a two dimensional array of numbers, with each row representing one component, and each cell representing a parameter for a component. When the component array is constructed, enumerations from the object oriented model are replaced with integers in the array. The possible component parameters for each row of the array are: the unique ID of the component, the component type, the size of the data bus, the number of master ports and slave ports on the component, the size of the address bus, the selector bus width, an array of the address ranges that each master is responsible for, arrays of point-to-point connections for master and slave ports, the clock frequency, the component type and the arbitration type. Of course not every component uses all of the available fields.

Each component in the system model is associated with one port manager which keeps track of the component's master and slave ports, and abstracts communication details. The port manager can quickly allocate or free a port on a component. Also, it tracks the sets of connected and free ports, and the slave to master connections on an interconnection component. When a component is added to the interconnection network model, the address ranges of all affected ports in the system are updated by port managers. During compile time, the port managers are removed and the data is added to the component VHDL.

Point-to-point connections between components in the interconnection network are described using four integers. Two integers are needed to point to the two components being connected together, and two more integers describe the ports on each device which will be used. The package generates each component and then the top level entity. A confirmation message is displayed in the prompt, and then all of the exported files can be located in the working directory.

It should be noted that using memories in a model and then synthesizing the design does not result in VHDL being synthesized for a memory module in the design. Instead a wrapper for the memory is added, and a person or tool must place the Altera or Xilinx compatible memory file of the specified name into the directory of the design. The memory file in our case comes from a compiler, linker, and loader.

3.2 Component Design

The following two paragraphs describe how the VHDL templates for the busses were designed. Next we describe how the documentation is generated. Finally, we explain how AMBA compliant cores such as the LEON3 processor can be used in SING, and how clock domains are synthesized.

The shared bus functions by allowing one master at a time to access the bus via a slave port on the shared bus. To implement the shared bus in the system, the component was separated into four parts: the arbiter, the slave multiplexers, the master multiplexers and the address decoder. The address decoder checks for valid addresses from an exterior master when it requests the bus. If the request is valid, then the request is sent to the arbiter section. The arbiter chooses which connection has priority and sends a signal to the master and slave multiplexers to make the connection.

The crossbar functions by allowing multiple connections at once. Like the shared bus, it is implemented using four parts: the arbiter, slave multiplexers, master multiplexers and address decoders. Each slave port on the crossbar can be connected directly to any master port on the crossbar and has its own address decoder. When a slave port receives a request from a master, the slave port's address decoder verifies the legality of the address and then tells the arbiter of the request, and which master port to output. The arbiter in this case verifies if the crossbar's master port can be used. If so, then the arbiter tells the slave and master multiplexers to make the connection, and the requesting master component is virtually connected to the corresponding slave component.

SING offers automatic documentation generation which adheres to the WISHBONE B.3 standards. The documentation is created in PDF format using free, Java-based library called iText. Each component including the top level entity is described in a separate PDF file.

The LEON3 processor comes with an AMBA compliant interface [19, 20]. It is used in SING's WISHBONE B.3 compliant interconnection networks as a processor by using an AMBA to WISHBONE bridge. The bridge maps AMBA signals to WISHBONE signals. As well, clock domains are preserved in the synthesized design by including clock dividers implemented with flip-flop based counters.

3.3 API

The API implemented in SING allows the user to test an interconnection network behaviorally before attempting to compile a RTL design. The creation and simulation of an interconnection network is controlled from the Java class *Runme.java* in the API package. The API internals are object oriented, and each component in the network executes in parallel. This component parallelism is natural because it models the parallelism found in actual multiprocessor system-on-chip implementations.

In order to iteratively improve the topology of a system, a user will want to perform simulations based on how many peripherals and processors are added to the design, and how they are connected. Simulation of the behavior of an interconnection network requires behavioral descriptions of the peripherals and processors in the system. Therefore, a blank extensible model of a hardware block is needed with children that specialize into processors, memories, and custom logic. In SING, a component called a connector is used to model these memories, processors, and custom logic. For example, a processor would be implemented as an extension of the connector component, which may have master ports and slave ports. During simulation, messages are routed to and from connectors through interconnection components based upon the availability of the bus, and the particular destination address.

The *Component* class is at the top of the component hierarchy and contains several enumerations which facilitate human-readable comparisons between component parameters. Parameter typing is enforced using enumerations. Our use of enumerations in the API promotes code readability and fewer errors in the order of parameters will occur if the arguments are typed. Each component in the network is given a unique ID when initialized. Each master or slave port on a component also has an ID, but these IDs are local to the component. Finally, a component can be deleted from an interconnection network based on its ID, and any connection to other components will be removed automatically when this deletion method is called.

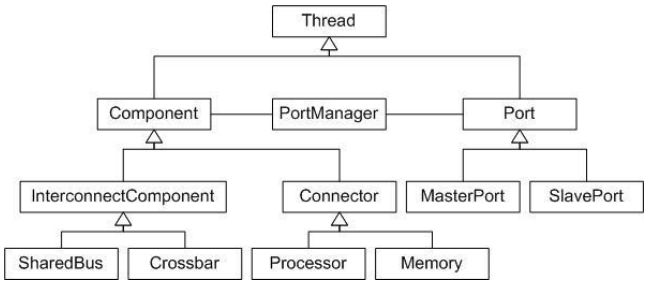

Figure 2. UML Diagram of the object oriented model.

A connector can be initialized by the API as either a master connector or a slave connector, and then attached to the network. Connectors are instantiated with a variable number of master and slave ports. There are currently several methods that permit the user to create a connector including: *addMemory*, *addSlaveConnector*, *addMasterConnector*, and *addProcessor*. The *addMemory* method creates a slave connector with one or more slave ports which in simulation can act as a memory unit, and there is an optional field that can be used to specify the size of the memory unit. Similarly, the *addMasterConnector* method creates a connector with one or more master ports and no slave ports, and adds it to the system model.

The *Memory* class was implemented both as an example of a slave connector, and as a starting point for simulating multiprocessor systems. Inside the *Memory* class, a map with sparse representation is used to describe the state of the memory. More specifically, any address containing the number 0 is removed from the memory map, and any address not in the map returns data with the value 0 when read. Writing to the memory with non-zero data updates the map at the specified address. The *Memory* class is one example of how the *Connector* class can be extended to simulate application specific hardware. Using sparse representation speeds up simulation time and reduces the memory requirements for simulation.

When master and slave ports are connected to one another in the model, a bidirectional link is set up between them. A port is aware of its state as connected or not connected, the address range to which it must listen, the parent component id to which its port manager is associated, the port manager to which it is associated, and its own priority.

The slave port number on an interconnection component is used as the priority number as well. For this reason, the user will be interested in connecting components to specific slave ports when the arbitration type is set to fixed priority, and she may not care as much when the arbitration type is set to round-robin. For this reason, SING provides two ways to connect ports to each other: by telling one component to connect to another and allowing the tool to select an open port to connect to, or by connecting a component to a specific port. These two options for connecting ports together are available when connecting a slave port to a master port and vice-versa.

The *InterconnectComponent* class is used to encapsulate shared behavior between all interconnection components. An interconnection component must implement a specified arbitration type (either round robin or fixed priority), and a map between master ports and address ranges. With fixed priority arbitration, the lower the port number is the higher the priority of the connected master will be. It is a counterintuitive fact that within a bus, slave ports request access to master ports because master component (processors) access the slave ports of busses and the busses then carry out the master request.

Consider the following example: A master port of a processor sends a write request with an address and some data to a slave port on a bus. The slave port on the bus in turn requests access to a master on the bus that is responsible for that address. Therefore the interconnection component's slave ports request access to a master port on the bus via a semaphore. In a crossbar, there is one semaphore per master, while in a shared bus there is one semaphore for the entire component. The methods for getting and releasing semaphores are implemented in the lower classes (Crossbar, and SharedBus) but the methods for communication in interconnection networks are so uniform across these two types that they are maintained in the *InterconnectionComponent* parent class.

In the SING API, crossbars and shared busses are instantiated using similar methods. All of the variables used to declare these two components are the same. Table 2 shows the variables required to declare an interconnection component.

Point-to-Point connections are automatically generated by connecting a master and a slave port. Dataflow connections are not yet implemented, but they will be realized by declaring multiple processors with master and slave ports, and then connecting them in series.

Table 2. Crossbar and Shared Bus constructor paramters

Variable type	Possible Values	Description
ArbiterType	ROUND_ROBIN, FIXED	Specifies which arbiter to use in the component
BusWidth	1, 8, 16, 32, 64	Data bus width in bits
ClkDivideType	1, 8, 16, 32, 64	Clock divider for the component
int	0 or more	Number of master ports
int	0 or more	Number of slave ports
int	1 or more	Address bus width
int	1 or more	Selector bus width (This bus provides support for granularity by indicating which bytes on the data bus are valid)

Iterative decisions about the costs and benefits of adding processors or peripherals can be considered several times throughout the design process. Once a network has been specified, simulated messages can be sent between components inside the network, even if some ports are not yet connected. Control bits must be set in the simulation in order to simulate a single cycle. For example, the write enable control bit must be set when performing a write cycle. Control signals were implemented using the *BitSet* class.

3.4 The Graphical User Interface

The GUI is shown in Figure 3. It can be used to set up a network and to initiate hardware generation. The GUI uses the API to manipulate the interconnection network information.

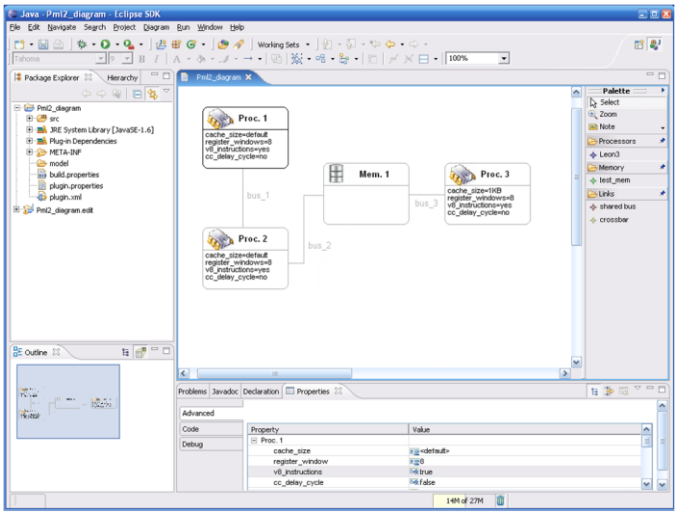

Figure 3. A screenshot of the SING graphical user interface.

4. RESULTS

A system-on-chip for matrix multiplication was designed in SING and implemented on an FPGA to test the functionality of SING and to demonstrate the performance of the tool. The design was written into the SING API and compiled. Our design at the model level was one 32-bit processor and one 32-bit memory connected via a WISHBONE bus. The software was compiled using the COINS compiler, a customized version of the GNU assembler, and a fully custom loader to create memory files for Altera FPGAs. SING generated the LEON3 processor's top level design file, an AMBA bus controller and the AMBA to Wishbone wrapper. The LEON3 was configured to use the instruction cache, local data RAM and register file. The local instruction RAM was disabled by default so that the processor would obtain instructions from the external Altera RAM through the AMBA interface and WISHBONE bus.

The generated memory connector contained a memory wrapper and memory module. The function of the wrapper is to take WISHBONE signals originating from the processor and convert them to signals compatible with Altera memory. The two connectors mentioned above were connected together in a top level file. The generated system and libraries were exported by SING into a uniquely named output directory.

The outputs of the compiled system were plugged into a performance counter project designed by hand and the system was compiled onto the Altera Cyclone II FPGA. The performance counter project contained a comparator that was connected to a counter that stops when the expected result has been observed from the SING-generated subsystem.

The SING part of the experiment was performed on an Intel(R) Core(TM)2 CPU running at 1.87 GHz, with 2046MB RAM. The operating system was Windows Vista(TM) Business, and the test was performed using the Eclipse IDE, Altera Quartus-II, and an Altera DE2 board.

The compile time for SING to generate VHDL for the embedded system was approximately 1 second. The size of the compiled system was 2489 logic elements, 65024 memory bits and 2 DSP block 9-bit elements.

5. CONCLUSION

In summary, SING is an open-source tool implemented in Java. The tool includes an API with simulation capabilities which can be used to explore the design space of a multiprocessor embedded system, an Eclipse GUI, and a wrapper for AMBA compliant IP cores. A common matrix multiplication program was designed, simulated, and implemented in VHDL using SING.

One area of the design flow which we will address soon is the importing of IP cores. Following in the footsteps of EDK, a wizard will be created to guide the user through the process of adding a peripheral or processing element to the network. For example, the WISHBONE DMA/Bridge IP Core written in Verilog could be very useful if imported [13]. The process for importing IP will be to connect an imported component's top level VHDL port map and expose the ports to the GUI and the API. Once the imported component is visible, the tool will be able to simulate the network with transactions at the gate level or the model level. Of course, the user will be able to further tune the trade-off between simulation time and attention to low level details as additional simulation models are added.

In future versions of SING the low-level hardware simulation and verification capabilities will be improved. Specifically, TCL scripts will be used to call a simulator such as ModelSim [18]. In terms of validation, value range checking for addresses must be improved to include detection of invalid connections between components. A suite of test cases must be designed to ensure that the exported VHDL can be tested quickly by the user. Finally, SING will be added to a configurable MP-SoC design tool in order to explore the trade-off between adding one more processor and adding another custom instruction.

6. REFERENCES

[1] Asokan, V. Designing multiprocessor systems in platform studio. (White Paper) http://www.xilinx.com/support/documentation/_white_papers/ wp262.pdf, November 2007.

[2] Altera Corporation. Creating multiprocessor nios ii systems tutorial. http://www.altera.com/literature/tt/_tt_nios2_multiprocessor_tutorial.pdf, December 2007.

[3] Altera Corporation. Quartus ii version 7.2 handbook - volume 4: Sopc builder. http://www.altera.com/literature/hb/qts/qts qii5v4.pdf, October 2007.

[4] Jaber. et al. Overview on system-on-chip busses, noc, crossbar. http://soc.eurecom.fr/SoC/l3/presentations/jaber-fischer-forciniti.pdf, November 2006.

[5] Gasteier, M., and Glesner, M. Bus-based communication synthesis on system-level. In 1996. Proceedings., 9th International Symposium on System Synthesis, (November 1996), pp. 65-70.

[6] Grasset, A., Rousseau, F., and Jerraya, A. Network interface generation for mpsoc: from communication service requirements to rtl implementation. In Proceedings of the 15th IEEE International Workshop on Rapid System Prototyping (RSP '04) (June 2004), pp. 66-69.

[7] CMPware Inc. Cmpware product brief. http://www.cmpware.com/ Docs/CmpwareProductBrief.pdf, January 2006.

[8] Xilinx Inc. Embedded system tools reference manual - embedded development kit, edk 9.2i. http://www.xilinx.com/support/documentation/ sw manuals/edk92i est rm.pdf, September 2007.

[9] Kyeong, K. R., and Mooney, V. Automated bus generation for multiprocessor soc design. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (November 2004), vol. 23 of 11, pp. 1531-1549.

[10] Le. et al. Workflow model of equipment maintenance support based on petri-net. In IEEE International Conference - ICCA 2007 (May 2007), pp. 2819-2822.

[11] Lteif, T., Muheto, Y., and Ridhawi, I. Modular design of multiprocessor systems. Tech. rep., School of Information Technology and Engineering, University of Ottawa, 2007.

[12] Niu. et al. An Efficient Cooperative Design Framework for SOC On-Chip Communication Architecture System-Level Design. Springer Berlin / Heidelberg, 2007, pp. 118-127. DOI = 10.1007/978-3-540-72863-4.

[13] OpenCores Organization. Wishbone system-on-chip (soc) interconnection architecture for portable ip cores. http://www.opencores.com/projects.cgi/ web/wishbone/wbspec b3.pdf, September 2002.

[14] Wieferink et al. System level processor/communication co-exploration methodology for multiprocessor system-on-chip platforms. In IEEE Proceeding - Computers and Digital Techniques (January 2005), vol. 152, pp. 3-11.

[15] Winter, M., and Fettweis, G. Interconnection generation for system-on-chip design. In International Symposium on System-on-Chip, 2006 (November 2006), pp. 1-4.

[16] Yawen, N., Jinian, B., and Haili, W. Cgem: A communication graph extraction method based on hcdfg for channel mapping in system level design. In Proceedings of International Conference on Computer Aided Industrial Design and Conceptual Design (2005), pp. 696-701.

[17] Suzuki, M., Fujinami, N., Fukuoka, T., Watanabe, T., and Nakata, I. 2005. SIMD Optimization in COINS Compiler Infrastructure. In Proceedings of the innovative Architecture on Future Generation High-Performance Processors and Systems (January 17 - 19, 2005). IWIA. IEEE Computer Society, Washington, DC, 131-140. DOI= http://dx.doi.org/10.1109/IWIA.2005.40

[18] Mentor Graphics Corp. ModelSim - a comprehensive simulation and debug environment for complex ASIC and FPGA designs. http://www.model.com/, 2008.

[19] Gaisler Research. Leon3 – Gaisler Research. http://www.gaisler.com/cms/index.php?option=com_content &task=view&id=13&Itemid=53, 2008.

[20] ARM Ltd. AMBA overview. http://www.arm.com/products/solutions/AMBAHomePage.html, 2008.

[21] Altera Corporation.Avalon interface specifications. http://www.altera.com/literature/manual/mnl_avalon_spec.pdf, 2008.

[22] Unneback M. - OpenCores Organization. WISHBONE builder: Overview. http://www.opencores.org/projects.cgi/web/wb_builder/overview, 2008.

[23] COINS-project. a COmpiler INfraStructure project. http://www.coins-project.org/international/index.html, 2008.

[24] Tensilica Inc. Multiple processor design. http://www.tensilica.com/methodology/multi_processor_design.htm, 2008.