

A Generic Framework for Rapid Prototyping of System-on-Chip Designs

Dmitrij Kissler, Alexey Kupriyanov, Frank Hannig,
Dirk Koch, Jürgen Teich
Department of Computer Science 12,
Hardware-Software-Co-Design,
University of Erlangen-Nuremberg, Germany.

Abstract

The integration of different Intellectual Property (IP) cores to modern System-on-Chip (SoC) designs becomes more and more an important topic because of the benefits in the overall system performance and the design costs. In this paper we present a new generic framework consisting of a graphical user interface with an extendable highly parameterizable IP component library for convenient SoC architecture entry, as well as software tools, which provide an automatic generation of fast cycle-accurate simulators for verification purposes and synthesizable HDL code for hardware synthesis. Because the communication of the single IP cores also plays an important role, our IP core library includes an open-source bus component, which is used in a case study design.

1 Introduction

The integration of several functional IP cores into a single SoC is advantageous because of the better performance characteristics, lower power consumption, lower production costs as well as a higher degree of miniaturization than in the case of separated units. These are the key properties needed in such fast growing areas of application as mobile communication, multimedia, and portable entertainment devices. A typical SoC design usually consists of one or more integrated processor cores, embedded memory, a number of dedicated hardware components, input/output modules and finally it contains an On-Chip-Bus (OCB) for communication purposes. As target technology ASICs (Application-Specific Integrated Circuit) as well as Field-Programmable Gate Arrays (FPGA) are deployed. The importance of SoC is apparent in consideration of the fact, that leading FPGA manufacturers such as Altera and Xilinx brought their own systems on the market: from Altera the EXCALIBUR system with the

NIOS or ARM processor and from Xilinx the hard IP core of the POWERPC or alternatively the soft IP core of the MICROBLAZE processor for the FPGAs.

2 Related Work

Several software tools are available for the specification of SoC designs [1–5]. The system-on-a-programmable-chip builder (SOPC Builder) [6] from Altera is such a tool, which allows the user to define SoC designs graphically and automatically generate VHDL or Verilog source code according to this description. The SoCLib project [7] is an open modeling and simulation platform for multiprocessor SoC. The core of the project is a library of simulation models for several IP cores, written in SystemC. Two abstraction levels are available: cycle-accurate/bit-accurate and TLM (transaction level modeling). However, the synthesizable models for these IP cores are not available due to legal restrictions. All of the SoCLib components implement the VCI [8] protocol. The simulation of the generated HDL code is not integrated into these tools and has to be done with external simulation software, like Modelsim [9].

Modules for the interconnection of single IP cores to a complete System-on-Chip play a very important role. The choice of the right interconnect architecture has a strong effect on the system performance [10], development time and consequently project costs. At the beginning of SoC development proprietary interconnection schemes which were not compatible to each other were utilized. As the system complexity increased the need for standardized bus interconnection schemes grew and different commercial as well as open-source standards were developed. The best known commercial systems include the CORECONNECT bus architecture from IBM [11], the AMBA bus from Arm Ltd. [12], and the AVALON bus from Altera [13]. As a represen-

tative for an open-source bus system one can name the WISHBONE bus originally designed by Silicore Corporation [14]. One difficulty with the different On-Chip-Buses (OCBs) is, that the interface to the specific OCB is usually integrated into the functional logic of the IP core components. Therefore, they must first be adapted to some other OCB. This cannot always be done easily, especially if only hard or firm IP cores are available. The SOPC Builder and the SoCLib library, respectively, integrate the interface to the specific OCB into the functional logic of the components and these CAD tools provide only IP cores with this specific bus interfaces. To deal with this problem one could implement the core-centric protocol OCP-IP (Open Core Protocol International Partnership) [15] which introduce de facto a new bus protocol as an abstraction layer over the existing OCBs. To keep the possibility to integrate new IP cores regardless which OCB interface they implement, we use another approach, namely to separate the actual bus interface from the application specific logic of the IP core. This results in master and slave modules which are connected with the functional IP cores. With this technique it is possible to interconnect IP cores which are not provided with a special bus interface. Furthermore, we use the WISHBONE bus and its concept of tag signals, which is explained in detail in our case study in Section 5.

The main contributions of this paper can be summarized as follows:

- A new integrated framework was built for graphical, system-level modeling of reconfigurable SoC designs,
- Automatic generation of fast cycle-accurate simulators and generation of synthesizable HDL code was integrated into the framework,
- A separation is made between the functional logic of an IP core and its bus interface and no additional (abstract) bus protocols are needed.

The rest of the paper is structured as follows. In Section 3, we present our integrated SoC framework. Subsequently, in Section 4, our generic interconnect methodology is discussed. Our approach was tested in a case study, Section 5, with a MIPS processor and peripheral devices on an Altera Cyclone FPGA. Finally, we conclude our paper in Section 6 with a summary and outlook.

3 System Integration Framework

ARCHITECTURECOMPOSER is a framework for graphical and interactive architecture entry and composition. Its origin is the BUILDABONG project [16], [17], a

project for architecture and compiler co-design of embedded application specific instruction set processors (ASIPs). In ARCHITECTURECOMPOSER the designer chooses from a library of customizable building blocks to compose graphically his desired architecture. This is done by connecting instantiated components with each other. The library contains a lot of necessary modules for the processor design such as register files, memories, multiplexers, arithmetic and logical units, simple busses, and others. All components are parametric, e.g. the word width of the modules can be chosen arbitrarily. Furthermore, ARCHITECTURECOMPOSER offers a hierarchical architecture entry to handle complex systems and to reuse already designed subsystems. Realistic microprocessors such as a MIPS processor or a DSP from Texas Instruments TMS320C6 series have been successfully modeled within hours. From the graphical architecture description a mixed structural/behavioral model of the architecture is generated automatically. This model is based on the formalism of Abstract State Machines (ASM) [18]. Additionally, a debugging and simulation environment which allows to verify the functionality of the given architecture on pipeline-cycle and instruction-cycle levels is generated. At any time, the graphically entered hardware architecture can be automatically transformed into a synthesizable hardware description language program. Three HDL-generators, namely, for VHDL, Verilog, and SystemC are available. The aim to bring the ARCHITECTURECOMPOSER framework to the higher level of System-on-Chip and multi-core interactive design required two things. First, faster simulations as the above described ASM approach were needed thus we recently developed the so-called RasimSimulator [19], [20], which can automatically generate a stand-alone cycle-accurate and bit-true C++-compiled fast simulator which executes the simulation approximately up to two orders of magnitude faster, compared to the Modelsim [9] VHDL-RTL simulation. Second, the integration of Wishbone OCB and several peripheral IP as presented in this paper are necessary to bring the framework to the higher level of SoC design.

Before we present the case study in detail, we would like to give an overview over our interconnect methodology of IP cores in the next section.

4 Interconnection Methodology

In the embedded system industry bus architectures for core interconnection became widely accepted. Since several bus standards exist, each manufacturer of an IP core tailors its product to the bus architecture of his choice. Abstractly speaking, the IP core manufacturer

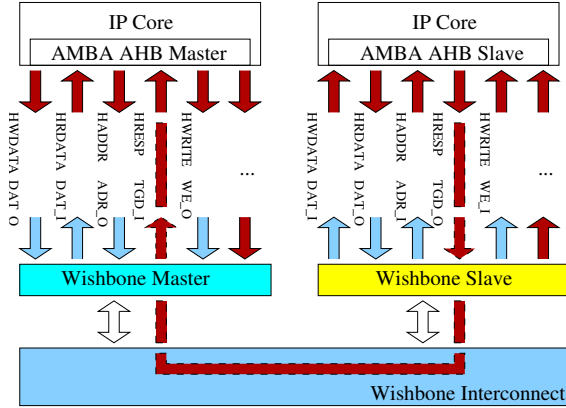


Figure 1: Tag signals treatment in Wishbone.

designs the set of the bus interface signals of the given IP core to be a subset of the signals provided by the given bus standard. In this case we can identify only one set of signals concerning the interconnection of the IP cores, namely that of the given bus standard. Furthermore, the bus interface is integrated into the IP core and its functional logic, as shown in Fig. 1 for two IP cores with AMBA bus interfaces. For various reasons for the SoC engineer it is not always possible to use IP cores with the same bus interface in his design. Therefore, we propose to separate the bus interface of an IP core from its functional logic. After this step additional sets of signals can be identified as shown in Fig. 2. In this figure three types of bus modules are introduced: the master and the slave module which represent the master and the slave interface to the specific OCB and the interconnection logic module for the appropriate OCB. If an IP core plays both master and slave roles in bus transfers like it is the case in IP cores with DMA functionality this core is simultaneously connected with one master and one slave module. Similarly, the IP core can simultaneously be connected to multiple master or slave modules. Therefore, our approach supports any combination of master and slave modules connected to the OCB. In master and slave components a distinction is made between the set of original OCB defined signals I and the set of external signals E generated by the IP core. The set $B = \{we_i, we_o, adr_i, adr_o, dat_i, dat_o, cs\}$ consists of the base signals needed for basic bidirectional communication over the OCB. The index next to the signal name denotes the direction of the signal relative to the given module: i stands for input and o for output signals. Write enable, address, data, and chip select signals are abbreviated with we , adr , dat , and cs , respectively. In the case that $B \subset (E \cap I)$ the IP core at least partially implements the current OCB interface. If $(E \cap I) \equiv B$ only the minimal functionality possible is

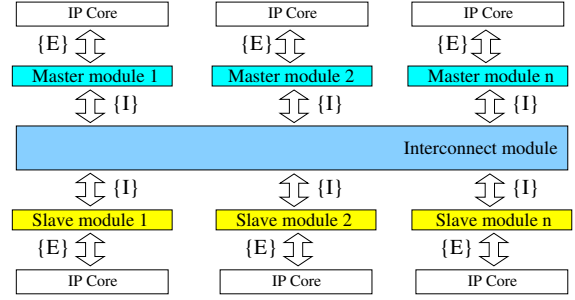


Figure 2: The logical structure of an OCB.

supported with the help of the basis signals from the set B .

$$\forall e, i \in (E \cap I) : i := e \quad (1)$$

The external signals e which are within the set $E \cap I$ can simply be assigned to the respective internal signal i , observing the right input/output relation. The signals from the sets $E \setminus I$ and $I \setminus E$ have to be treated separately.

As an exemplification we now consider the question, how two IP cores with integrated AMBA interfaces can communicate over a Wishbone bus, see Fig. 1. The signals from the set $E \setminus I$ must be treated as the so-called *tag* signals to keep as much as possible of the original performance of the IP Core with its native AMBA bus interface. According to the Wishbone standard these signals are passed through the Wishbone interconnect to other modules like shown in Fig. 1. Control signals like interrupts, errors, control status information etc. can also be integrated into the Wishbone interconnect with the help of the *tag* concept. According to the Wishbone specification [14] *tags* constitute a special class of signals which is intended for user enhancements to the Wishbone specification. In addition three tag types are defined, to indicate the precise timing to which the signal must conform. The tag types include the address tags, the data tags and the cycle tags. Parity information about the address or data transfer would be an example for address tag and data tag signals, respectively. Signals for a privileged access to the bus would be treated as a cycle tag signal. Thereby additional information can be attached to an address transfer, a data transfer or a bus cycle. Because the HRESP signal from AMBA AHB slave interface provides additional information on the status of a transfer like OK, ERROR, RETRY, and SPLIT the signal belongs to the set $E \setminus I$ and the data tag type TGD.O was chosen in Fig. 1. The signals from the set $I \setminus E$ must be assigned a default value in form of a reasonable constant. For example, if no handshaking between a master and a slave module is needed, and the IP core connected to the slave module has no acknowledgement output signal, the corresponding input signal in the slave module is assigned a constant value of 1.

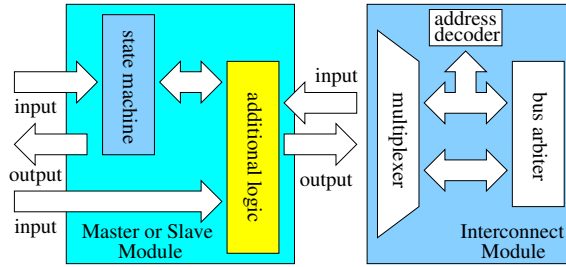


Figure 3: Internal bus modules' structure.

The inner structure of a master or slave module from Fig. 3 consists of a state machine which implements the actual bus transfer protocol and some additional logic especially for mapping of external signals to internal signals according to the Equation (1). The interconnection component in Fig. 3 consists of a bus arbiter, a multiplexer which connects the master modules with slave modules according to the current arbitration scheme and the central address decoder which selects the appropriate slave module.

4.1 Integration into the Framework

The modules implementing the ideas mentioned above were designed and integrated into the overall framework. For each module several generic parameters can be specified graphically:

Interconnect Module. The width of the address and the data line for the interconnect module can be entered in 1-bit steps. The minimum width is 8 bit and the maximum width was arbitrarily chosen to be 32 bit. Any change in the width of the interconnect module is automatically propagated to the master and slave modules connected. These changes are mapped to the simulation models as well as to the generated VHDL files via the generic variable concept in VHDL.

Master Module. Master modules implement both the asynchronous and the synchronous data transfer [14]. This either can be specified at run-time by the IP core connected to the master module or it must be specified before the synthesis if the IP core does not implement the Wishbone interface.

Slave Module. The registers of the IP cores connected to the slave modules are used for memory mapped I/O. This means that the same bus is used to address both memory and I/O devices during input/output operations. The CPU instructions used to read and write to memory are also used in accessing I/O devices. The width of the address line for the slave module can also be specified graphically.

In the next Section we proceed with the implementation details and synthesis results of the case study.

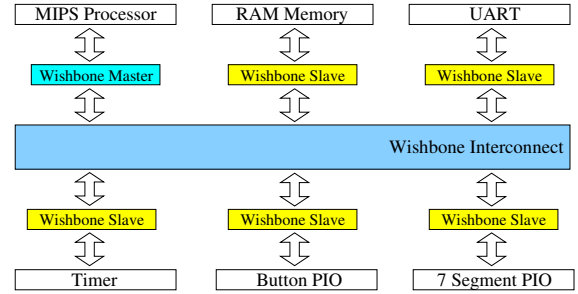


Figure 4: The implemented Wishbone OCB case study.

5 Case Study

In order to test our concept of a reconfigurable System-On-Chip, a design containing one processor which communicates with several dynamically configurable peripheral devices like GPIO (General Purpose Input/Output) and UART (Universal Asynchronous Receiver/Transmitter) over the Wishbone-compliant OCB was entered using the GUI of the ARCHITECTURECOMPOSER. As a processor core a MIPS processor was used which was also specified using ARCHITECTURECOMPOSER. The initial parameters of the modules used in the design are specified graphically and the peripheral devices as well as OCB modules are dynamically reconfigurable at run-time. Finally, the automatically generated VHDL code for this design was synthesized on an Altera Cyclone FPGA.

The design consists of a MIPS processor in form of a soft IP core connected as a master to the Wishbone bus, an integrated 1 KByte RAM memory, UART module for external communication, a timer module as well as three general purpose I/O modules (see Fig. 4). None of the IP cores in the system has a built-in Wishbone bus interface. Only the signals from the base set B are available: we , adr , dat , and cs . The integrated memory has an asynchronous interface, thus the asynchronous communication mode was chosen. The whole design builds an embedded control system with external communication tasks accomplished by the UART module, the computational tasks run on the MIPS processor, and the user interaction performed over the GPIO modules.

5.1 Fast Simulation

In order to verify the functional behavior of the specified case study SoC design before synthesis, a stand-alone C++-compiled fast simulator of the whole design was automatically generated in order to test different scenarios. The structure of the generated simulator is similar to the structure of a conventional *levelized compiled code* simulator. The simulator is based

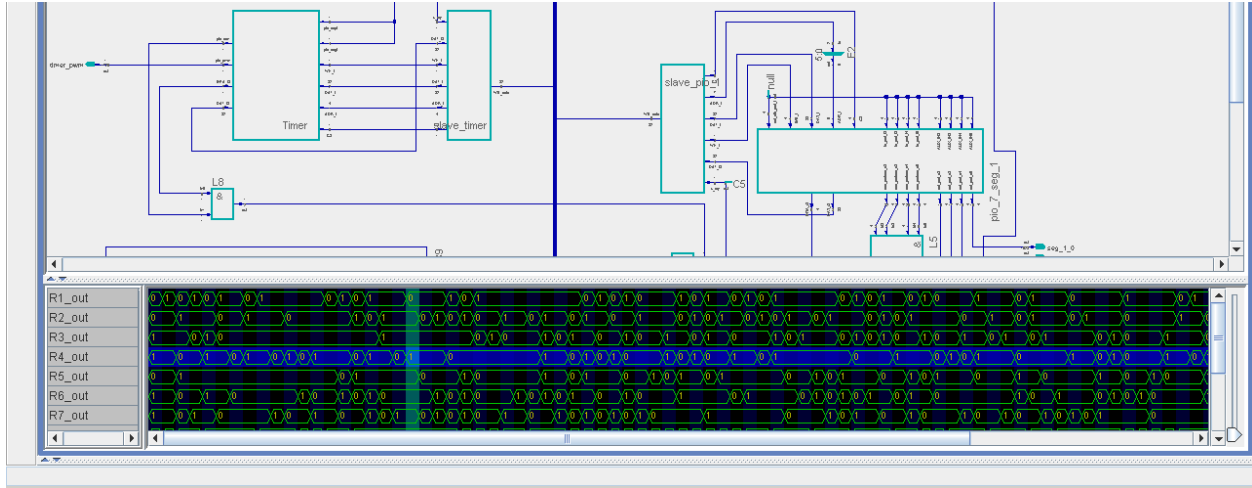


Figure 5: Waveform fragment of the simulation.

Table 1: Simulation speed results for the case study SoC design.

Simulator name	Simulation speed
Modelsim	14 285 cycles/s
RASIM	83 333 cycles/s

on an event-driven model which is especially advantageous when simulating reconfigurable architectures where some signals may change only during the configuration phase. The main principle of the simulator includes the main simulation loop in which the stimuli vectors are read from an external stimuli file, the simulation routine *sensitivity-update* is called to perform the current simulation cycle, and the selected tracing values are stored. The simulation results presented in the integrated graphical waveform viewer are shown in Fig. 5. Furthermore, the integrated design environment allows for setting watch-points, i.e. register values can be directly displayed within the registers of the schematic entry. This is very convenient during step-by-step architecture debugging. With the help of several optimization techniques based on graph theoretic approaches [19], a speedup of 6 was achieved compared to Modelsim, see Table 1. Because of the asynchronous communication mode the high potential of these techniques for synchronous designs [19], [20] could not be fully exploited in this case.

5.2 Design flow of the case study

The single steps, beginning with 1) the graphical architecture entry, 2) generation of the simulation model, 3) compiling of the generated simulator, 4) fast simulation,

5) validation of the simulation results, 6) generation of the synthesizable VHDL code, 7) design entry into synthesis tool, and finally 8) the synthesis of the whole design, as well as the corresponding time expenditure are

Table 2: Design steps and their processing times.

Design Step	Tool Name	Time (mm:ss)
STEP 1	ARCHCOMP	20:00
STEP 2	ARCHCOMP	00:01
STEP 3	ARCHCOMP	00:05
STEP 4	RASIM	00:06
STEP 5	ARCHCOMP	15:00
STEP 6	ARCHCOMP	00:01
Design time		35:13
STEP 7	ALTERA QUARTUS	60:00
STEP 8	ALTERA QUARTUS	10:15
Synthesis time		70:15
Overall time		105:28

summarized in Table 2. In the first step the design was graphically entered into our tool. Then the simulation model of the design was automatically generated. In the third step the simulation model was compiled with the *GCC 3.3*-compiler on a Pentium 4, 2 GHz machine under Linux operating system. After this, the stand-alone simulator was run on the same machine and the simulation results were analyzed in the fifth step. Synthesizable VHDL-code was automatically generated in the next step. So far, the design time took us 35 minutes. The synthesis of the design was performed on the same

machine under Windows XP operating system with the help of the Altera Quartus II v.4 software for the Cyclone EPIC20F400C7 FPGA. The design entry, project creation, pin assignments, and the definition of several synthesis options took us 60 minutes. The synthesis of the design took another 10 minutes. The overall time for the creation of the hardware part of our SoC design was only 105 minutes.

Table 3: Synthesis results for the case study SoC design of Wishbone bus interfaced modules.

Bus Module	Logic Cells	Registers
1 master and 6 slaves	75	0
multiplexer and arbiter	79	2
address decoder	14	0
whole SoC design	4097	1536

5.3 Synthesis Results

For the synthesized SoC with 16-bit address and 24-bit data line width the results from Table 3 were obtained. Altera Quartus II v. 4 software was used for the synthesis. The synthesis results show only a small logic cell demand for the introduced bus modules, see Table 3. As only an asynchronous mode for communication was chosen no registers and no state machines in the master or slave components were generated and consequently no registers were synthesized for these modules. The design runs with the frequency of 50 MHz. The peak throughput of the bus module is therefore 150 MByte per second. It can be increased if a synchronous burst mode for data transfer is chosen. By the use of the synchronous transmission scheme the overall frequency can further be increased and the use of burst mode also gives a performance gain by reducing the overall cycle number, needed to transfer the given amount of data. To explore the impact of the data path width of the OCB bus on the area usage and the frequency of the design, the width of the data path was varied in the first step in Table 2 and then steps 6 to 8 were performed for 16, 24 and 32 bit widths. The results are summarized in the Fig. 6. According to the results, the width of 24 bits should be chosen for maximum throughput.

6 Conclusions and Future Work

The need for an integrated environment for architecture specification, simulation and synthesizable HDL code generation is existent and urgent. In this paper such a

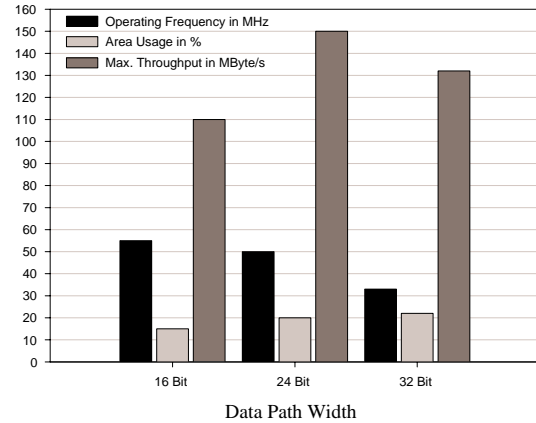


Figure 6: Operating frequency, area usage and maximum throughput for different data path widths.

software framework, ARCHITECTURECOMPOSER, was presented. It allows a graphical specification, cycle accurate simulation for convenient system validation and debugging respectively, and finally the automatic HDL code generation. In connection with the integrated OCB and peripheral components, complex SoC designs can be implemented as well. In the case study a Wishbone-compliant OCB was implemented and tested in a standard SoC system with a MIPS processor and peripheral devices on an Altera Cyclone FPGA. With the integration of the Wishbone OCB and peripheral components like UART, PIO, and timer into the component library of ARCHITECTURECOMPOSER a complete solution for the rapid prototyping of systems on a chip is now available. In the future, we would like to automate the synthesis of bus protocols using an approach similar to the one presented in [21] and [22]. Furthermore, we plan to use ARCHITECTURECOMPOSER in practical courses to inspire students with interest in hardware architecture and reconfigurable computing.

References

- [1] Ryu, K., Mooney, V.: Bus Generation for Multiprocessor SoC Design. In: Proceedings of Design, Automation and Test in Europe (DATE), Munich, Germany (2003)
- [2] Martin, G., Schirrmeyer, F.: A Design Chain for Embedded Systems. Computer 35(3) (2002) 100–103
- [3] Shin, C., Kim, Y., Chung, E., Choi, K., Kong, J., Eo, S.: Fast Exploration of Parameterized Bus

