

# Hierarchical Microprocessor Design Using XASM

M. Anlauff<sup>1</sup>, D. Fischer<sup>2</sup>, P. W. Kutter<sup>3</sup>, J. Teich<sup>2</sup>, R. Weper<sup>2</sup>

1: GMD FIRST, Berlin, Germany, [ma@first.gmd.de](mailto:ma@first.gmd.de)

2: Institute of Electrical Engineering and Information Technology, University of Paderborn, Germany, [{fischer,teich,weper}@date.upb.de](mailto:{fischer,teich,weper}@date.upb.de)

3: Department of Electrical Engineering, Swiss Federal Institute of Technology, Zurich, Switzerland, [kutter@acm.org](mailto:kutter@acm.org)

## 1 Abstract

In this paper, we introduce a tool-supported approach for mixed behavioral and structural simulator generation of complex state-of-the-art microprocessors using Abstract State Machines (ASMs) [5,3]. First, we describe our methodology starting from graphical architecture entry to automatic translation into XASM [1] language code. A major weakness of existing ASM language definitions is the lack of support in modeling hierarchy which is of utmost importance for modeling complex architectures, i.e. for architectures with highly regular and repeatedly instantiated components. Therefore, we discuss problems of extensions of ASM languages to support hierarchy and show how XASM's consequent realization of hierarchical ASMs (as introduced in [6]) is able to support hierarchical modeling in the discussed field.

## 2 Motivation

Special application domains such as digital signal processing require the design of special instruction set processors (ASIPs) in order to fulfil efficiency, cost, and other design criteria. All in all, the final architecture of an ASIP is not a priori fixed but is the result of an iterative process of prototyping. Thus, a precise architecture model and simulation tool is of utmost importance. The project BUILDABONG at the University of Paderborn aims at architecture and compiler co-generation for ASIPs. Therein, the processor architectures are modeled by the formalism of ASMs, and architecture composition is modeled with hierarchical ASMs.

Based on the ASM-specification of the ARM microprocessor by Huggins and van Campenhout [4], we presented a methodology for the automatic generation of a cycle-accurate simulator for a realistic microprocessor [7]. Thereby, the simulator was automatically generated by the Gem-Mex environment [2] from a given XASM specification of the processor. In [8], we modeled a highly complex VLIW Digital Signal Processor of the Texas Instruments TMS3200 C6200 family using ASMs and

demonstrated that they allow the description of as well the control flow as inherent parallelism of a specification in a single model. A major point for improvement of existing modeling approaches, however, is the fact that a processor designer is confronted with the notational burden of the ASM language used. A solution in the area of design automation is to use graphical tools for design entry.

### 3 Architecture Composition and Automatical ASM Generation

In order not to burden a processor designer with ASM syntax, we therefore developed a graphical editor tool (*ArchitectureComposer*). From the graphical input of a processor's control- and datapath, *ArchitectureComposer* automatically generates an XASM specification. *ArchitectureComposer* provides a library of customizable building blocks such as register files, memories, arithmetic and logical units, buses, etc. The basic elements are modeled as generic as possible being parametrized (bit width, number of inputs/outputs, etc.). The parameters of each element can be manipulated at any time via a dialog. While composing basic architecture blocks, *ArchitectureComposer* automatically checks if an interconnection is valid. For example, it is not allowed to connect one input port to another, the bit widths of connected ports have to be equal, the output of a bus driver has to be connected to a bus, the control element of a bus driver must not be connected to another bus driver sharing the same bus. Finally, the editor allows for clustering nets to define new architecture components hierarchically. New components may be easily added. Figure 1 shows a screenshot of the design process for a simple architecture using *ArchitectureComposer*.

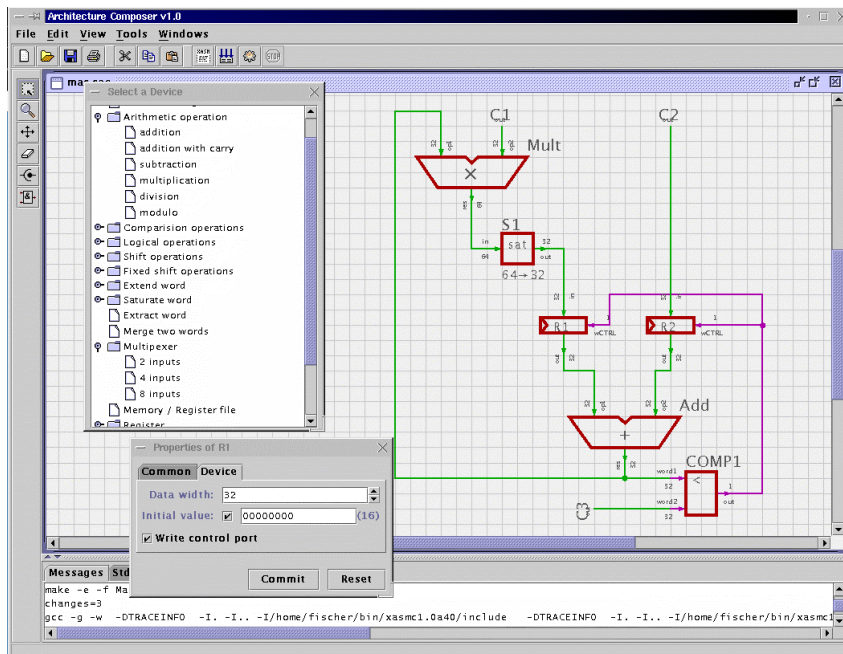


Figure 1: Designing a simple architecture using *ArchitectureComposer*

A behavioral ASM-model is generated in three steps: First, output signal declarations are generated. For each output signal of each hardware component, a function declaration is generated for computing its output signals. Then, functions corresponding to sequential elements (e.g. registers, memory) are initialized in a separate block. Finally, memory and register assignments are encoded as guarded update rules. The emitted XASM-code for the simple architecture of Figure 1 is listed below. Note that the architecture's behaviour is described by C-functions (here: `c_mult()`, `c_add()`, `c_extract`, and `c_gteq()`) which are supported by an interface of the XASM-language.

```
asm MAIN is
  use cpucore
  derived function Mult_res ==
    c_mult (Add_res, const1_out, 8, COMPLEMENT_2)
  derived function Add_res ==
    c_add (A_out, B_out, 8, COMPLEMENT_2)
  derived function E1_out ==
    c_extract (Mult_res, 16, 0, 8)
  derived function COMP1_out ==
    c_gteq (const3_out, Add_res, 8, COMPLEMENT_2)
  function A_out
  function B_out
  function const1_out
  function const2_out
  function const3_out
  init
    A_out := "00000000"
    B_out := "00000000"
    const1_out := "00000010"
    const2_out := "00000001"
    const3_out := "00001000"
  endinit
  if COMP1_out = "1" then A_out := E1_out
  endif
  if COMP1_out = "1" then B_out := const2_out
  endif
endasm
```

A major advantage of *ArchitectureComposer* is the visual representation of architecture building blocks and their composition, combined with a mathematical model of their semantics by means of ASMs. The domain specific, visual representation allows for easy manipulation, documentation, and reuse of the architectures, while the unambiguous behavioral model allows to investigate the correctness of transformations, implementations, and simulations. In addition, we are able to generate different ASM models on different abstraction levels for the same visual representation.

## 4 Approaches to Hierarchical ASMs

Processor architecture descriptions are typically hierarchically structured for complexity reasons. In our current implementation of code generation, however, we flatten the complete hierarchical design and generate a flat ASM description. Since this description is conform to the Lipari standard [5], this model can serve as input for existing ASM tools for Model Checking and Theorem Proving. But since existing ASM languages have no support for hierarchical structuring, this approach leads to lengthy and often unreadable descriptions in case of complex processor architectures

being modeled. In order to keep the efficiency of a hierarchical description, the main focus of our paper will concern definitions and use of hierarchical ASM concepts as supported by the XASM language.

We propose a method for mapping each building block of the component library of *ArchitectureComposer* into one XASM component and to compose building blocks by existing component composition mechanisms. The high degree of flexibility in interconnecting such XASM components should allow to generate XASM code component-wise. In other words, existing building blocks should be translated into XASM components and the composition of such building blocks should be done without altering the XASM models of the basic building blocks. With such a modularity it should be possible to reuse XASM components in complex designs.

## 5 References

- [1] M. Anlauff XASM - An Extensible, Component-Based Abstract State Machines Language. International Workshop on Abstract State Machines. Lecture Notes in Computer Science (LNCS) 1912, Springer, pp. 69-90, 2000.
- [2] M. Anlauff, S. Chakraborty, P. W. Kutter, A. Pierantonio, L. Thiele: Generating an Action Notation Environment from Montages Descriptions. Journal of Software Tools and Technology Transfer, Springer, to appear.
- [3] E. Boerger, J. Huggins. Abstract State Machines 1988- 1998: Commented ASM bibliography. EATCS Bulletin, number 64, pp. 105-127, February 1998
- [4] J. Huggins and D. Van Campenhout. Specification and Verification of Pipelining in the ARM2 RISC microprocessor. ACM Transactions on Design Augomation of Electronic Systems, 3(4):563-580, 1998.
- [5] Y. Gurevich. Evolving Algebras 1993: Lipari guide. Specification and Validation Methods, pp. 9-35, Oxford University Press, 1995.
- [6] W. May. Specifying complex and structured systems with evolving algebras. TAPSOFT'97, Theory and Practice of Software Development. Lecture Notes in Computer Science (LNCS) 1214, Springer, pp. 535 - 549, 1997.
- [7] J. Teich, P. W. Kutter, R. Weper: Description and Simulation of Microprocessor Instruction Sets Using ASMs. International Workshop on Abstract State Machines. Lecture Notes in Computer Science (LNCS) 1912, Springer, pp. 266-286, 2000.
- [8] J. Teich, R. Weper, D. Fischer, and S. Trinkert. A Joined Architecture/Compiler Environment for ASIPs. ACM SIG Proc. International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES 2000). San Jose, CA., U.S.A., pp. 26-33, November 2000.