ViCE-UPSLA: A Visual High Level Language for Accurate Simulation of Interlocked Pipelined Processors

Dennis Klassen

University of Paderborn
Department of Computer Science
Fürstenallee 11, 33102 Paderborn, Germany
dennis.klassen@uni-paderborn.de

Abstract: Simulation of processors is needed in early stages of development to reduce cost and increase quality of processor designs. Suitable simulators can be generated automatically from high-level specifications of the processor architecture. For this purpose, we have developed the domain specific visual language ViCE-UPSLA. It allows to describe pipeline based register-register, register-memory processor architectures and generates efficient simulators for such processors. In this way a variety of processors can be quickly prototyped for validation and evaluation. We have successfully used ViCE-UPSLA to model and simulate a processor with an ARM [ARM00] like architecture.

1 Introduction

In several domains, domain specific applications are developed with focus primarily on throughput. In addition to software and compiler optimization, it is necessary to optimize the processors. These optimizations can have different aims such as speed, power consumption, parallelism, real-time suitability, cost reduction, etc. However, efficiency enhancement through the development of application-specific processors is mostly in our focus.

Effects of processor optimizations must be determined with the help of simulators during the early stages of development. Developers need to estimate optimization trends from the simulation results and add promising improvements manually to the hardware design. In order to shorten development cycles, the implementation effort of simulator development has to be reduced. A development environment on the abstraction level of the processor-reference-documentation [GK83] is desirable. Then, developers would be able to immediately locate targets for optimization and test and manipulate the simulators as they need. Besides the efficient simulation of processors, the systematic validation of processors, starting from a specification on a high abstraction level, is the next step in development.

In this paper we present ViCE-UPSLA, a visual language for processor design. This visual language is inspired by the textual language UPSLA (Unified Processor Specification...
L. A. (KLST04). In addition to the description of an instruction set simulator, like UPSLA, ViCE-UPSLA adds the description of microarchitecture components on a high abstraction level. The visual language is characterized by the usage of typical and established terms and symbols from the processor design and architecture domain. From the processor specification, the toolchain of ViCE-UPSLA generates simulators for interlocked or non-interlocked microarchitectures automatically.

In this paper we present the concepts of the visual language. Section 3 focuses on the early stages of processor design, where an instruction set simulator is sufficient. Later, a more detailed microarchitecture simulator can be obtained, as described in Section 4. Section 5 presents the principles of our approach to generate the simulators. By looking at related works, we classify our visual language in Section 6. Some existing applications of the language are outlined next. A prospect on the language’s advancement concludes the paper.

2 Concepts of the visual language

In the early stages of processor development, only abstract documents such as reference documentation or technical data sheets are available. The different background of the processor and software designers, who cooperate in the development, demands a visual language with domain specific description elements, at a comparable abstraction level to the available documents. Our visual domain specific language ViCE-UPSLA allows the specification of a processor at the level of processor-reference-documentation [arm87, pow94], with a focus on simulation and validation.

In this section, we describe the fundamental concepts of the visual language for the specification of instruction set and microarchitecture of the target processor. The language is modelled after typical processor documents with visual description components. Hence, the developer creates and speaks about a data flow graph to express a microarchitecture or to describe instruction behavior etc. From the specification, suitable simulators are automatically generated. These simulators advance the development for example via design space exploration. Another goal is the automatic static and dynamic validation of the developed processor design against its specification. The language allows to check the static consistency of the specification. Also the specification is detailed enough to generate test cases for dynamic validation.

We describe the concepts of the language using an example of the development approach. The first scenario is the development of an instruction set simulator and the second scenario is a microarchitecture simulator. After the overview of the language, we explain the parts of the specification needed to describe the different kinds of simulators.

The first scenario is typical for the beginning of the development process. Here, only an instruction simulator is needed to evaluate the requirements of the applications on the processor. The flexibility of ViCE-UPSLA allows to create and evaluate variants of a processor in a short time. Since only an instruction set simulator is needed for this, we need to describe in ViCE-UPSLA just the instructions set components. This amounts to
descriptions of the structural and behavioral properties of the instructions.

The second scenario, in later steps of the development, requires more precise processor simulators, to evaluate the correctness of the instructions in the envisioned processor architecture. Therefore we describe in ViCE-UPSLA, besides the instruction set, the microarchitecture of the processor. To describe and simulate an existing processor with a defined microarchitecture, as base line for design space exploration (DSE), the specification in ViCE-UPSLA can be adapted to capture relevant aspects of the target microarchitecture precisely. From the specification, interlocked or non-interlocked cycle accurate simulators with respect to the microarchitecture can be generated.

ViCE-UPSLA structures its processor specifications similar to the logical partitioning in the processors documents, like a processor-reference-document. This approach of the language allows the designer to transfer the contents of the reference-documentation into a ViCE-UPSLA specification nearly directly. The overall structure of the language includes four views of the fundamental components of the processor (Figure 1):

- Instruction set
- Instruction format and addressing modes
- Register set
- Pipeline architecture and Data path

The register set definition allows to describe various configurations of register banks for different architectures, including the distinction between physical and architectural registers [DHTK07]. The specifications of the register banks and single registers are referenced in the descriptions of the addressing modes and instructions. The addressing modes and the instruction formats describe the recurring structures of the instruction specifications. The instruction set consists of the entirety of all instructions and encompasses the structural and behavioral descriptions. The pipeline specifies the data path, resources and forwarding paths, respective to the instruction execution step sequence.

The components of the processor are drawn in different views and can be specified separately. In a completed processor specification, the components of the processor are linked to each other. This independent specification of linked components allows subsequent replacement or editing of processor constructs in a completed specification. For design space exploration, editing or expanding the already specified components is essential. For example, to expand the number of accessible registers for a group of instructions, with ViCE-UPSLA the developer has only to adapt the register bank size and the range of its addressing mode in one place. The specification of the microarchitecture or instruction set is not affected by this change. To ensure consistency of the specification and to support structured development, the specification components are related to each other. ViCE-UPSLA provides a number of static and dynamic validation methods to ensure the consistency of the design.
3 Specification of an instruction set simulator

Now we present the specification steps necessary to describe an instruction set simulator with ViCE-UPSLA. To generate a simulator of an instruction set architecture, the instruction set, memory and register access specifications are needed. For this, each instruction needs a structural and behavioral description. The structural specification is needed to describe the decoding and the usage of the instruction. The execution details are specified by the behavioral description.

**Instruction set** For the specification of the instruction set, we have introduced the concept of abstract instructions. It supports a structured specification and classifies the instructions for validation. Besides, this approach reduces the specification effort. With the abstract instructions, we describe coherent groups of instructions and build equivalence classes. According to the pursued aspects by the developer of the processor, the equivalence classes can be determined, for example by separation of the instructions via properties like arithmetical or logical instructions or the execution duration of the instructions. The specification of an abstract instruction contains a generic structural and behavioral specifications. Each concrete instruction is derived from an abstract instruction and inherits all the specified properties of this instruction.
**Instructions structure properties** Most of the needed structure properties for the specification of an instruction are specified by the chosen instruction format. An instruction format combines the coding of the order and the length of the operands with the operation code. The processor specification in ViCE-UPSLA can contain instruction words with different lengths. This concept allows to describe CISC [SG79, NMEH81, HP06] instruction sets or other architectures with different instruction lengths.

The signature of the assembler notation and the machine code of the instruction is defined by the instruction format. Figure 2 illustrates an example, of the visual expression in ViCE-UPSLA, with the specified properties for the specification of the instruction format and the generated code for the visual expression. The specification of the instruction formats [HP06] in ViCE-UPSLA uses established visualizations from the domain of processor design, as shown in Figure 2. To describe the instructions of an existing processor, the individual operation codes can be defined separately for every instruction. Through the lengths of the bit fields of the instruction format the length of an instruction word of the processor is expressed exactly.

The additional bits to control the instruction execution or the selection of the addressing modes are interpreted in ViCE-UPSLA as operation code extensions. To specify those constructs in ViCE-UPSLA, the developer creates separate instruction formats for every configuration of the control bits. Figure 3 shows an ARM processor instruction format with two control bits I and S. Therefore with ViCE-UPSLA, the developer uses four instruction formats. This allows to distinguish the instructions more clearly from each other, which is desirable for simulation and validation. For example, an existing ARM processor instruction `ADD` with and without carry is expressed in ViCE-UPSLA as two different instructions `ADD` and `ADD_C`.

**Instruction Behavior** The behavior of the instruction is expressed on a high abstraction level with the visualization of a data flow graph, as shown in Figure 4. The specification view combines the cycle accurate behavior and the required resources of the instruction in one diagram. The data flow graph describes the value transfer between the elements.
During the specification process, the developer can use the operators and the interpreted operands as graph nodes, which are connected with directed edges. The operands are given by the instruction format. In addition to the interpreted operands, the implicit operands, like a carry bit, can be also added into the data flow graph. The data flow graph describes also the structural properties of the instruction, for example the operands’ read/write direction or execution cycles of the instruction etc.

The operational description of an instruction can be specified through the cascading of operators, as shown in Figure 4. The operators specify the actions, which are applied to the input values. Determined by the developer, the complexity of a data flow graph is dependent on the granularity of the operators used. ViCE-UPSLA provides a manageable number of predefined operators. To increase the design quality, a custom set of operators can be added. For design space exploration, as well, new operators can be defined by a processor developer.

It is also possible to describe parallel execution of combined operations on different ALU’s with ViCE-UPSLA. Therefore, ViCE-UPSLA supports multiple branches in the behav-
ioral description of an instruction. This allows to specify instruction sets for MIMD and SIMD architectures.

Figure 4 shows two implementations of the addition with carry (ADD C) instruction, which calculates the result for DR0 from the input operands SR0, SR1 and the implicit operand carry.

![Instruction dataflow graph and concatenation of operators](image1)

**Addressing mode** In ViCE-UPSLA, the specification of the instruction set is completed with the declaration of the instructions. To generate a simulator, the description of the access to the memory modules is needed, such as main memory or registers. In various architectures, like Motorola 68k [SG79, Mot92], the addressing modes describe non trivial rules for memory or registers access. The concept of the addressing modes [HP06] represents in ViCE-UPSLA the rules for the access to registers as well as memories. Figure 5 shows the concept for registers access. In the behavioral descriptions of the instructions, the interpreted operands are used. With the computation of an addressing mode, the value for the interpreted operand is provided from the register set, for the read direction, analogous for the write direction. The addressing modes are parametrised through the instruction operands from the instruction word.

An example from the ARM data sheet shows the use of interpreted and instruction operands in Figure 3. The operands Rn, Rd and Operand 2 represent the interpreted operands. Also, Figure 3 shows two complex addressing formats shift applied to imm and shift applied to Rm. The instruction operands for the addressing modes of interpreted operand Operand 2 are Shift+Rm or Rotate+Imm. The visualization of these specification in ViCE-UPSLA are shown in Figure 2. The addressing modes’ specification contains the assignment of the bit fields and the interpreted operands. In ViCE-UPSLA, the register addresses or immediate operands can be used as instruction operands.

![Instruction dataflow graph and concatenation of operators](image2)
in the addressing modes. The register addresses refer to a register bank and the immediate operands select a value from a defined value range. With this approach, the processor developer is able to describe the access to the register set and to specify complex addressing modes with optional pre and post calculations such a Motorola 68k [Mot92] architecture addressing modes.

**Register Set** For simulation and validation of the processor, the register set defines the system state. During the simulation, we are able to trace the system state. To design only a instruction set simulator, a simple register specification is sufficient. ViCE-UPSLA allows to describe register set configurations for complex structures with several usage modes of the registers.

In ViCE-UPSLA, the register set of a processor includes all programmer accessible registers of the architecture [pow94, HP06]. We distinguish between two kinds of registers, the physical registers and the architectural registers. Architectural registers describe the aliasing of the physical registers. With it, different views on physical registers can be defined.

For example, the configuration of a UserMode and a SystemMode of physical registers can be specified, as shown in Figure 6. If the physical registers are reg0-23 of register bank reg, registers of the UserMode are usr0-15 as a slice of the registers reg0-15. The SystemMode needs two slices reg0-7 as registers shared with the SystemMode and reg16-23 as special registers. The concatenation of these define a SystemMode.

In ViCE-UPSLA, the architectural registers have distinct names and are characterized by the underlying sequence of physical registers. There are different constructors, to create views of physical registers to model common configurations of architectural registers. The concatenation of register banks (called SequenceRegister) or a slice of a register bank (IndirectRegister) have been used for the SystemMode as shown in Figure 6.
4 Specification of an interlocked microarchitecture simulator

In the previous subsection we have described the components to create an instruction set simulator. In ViCE-UPSLA, we can expand the specification from a simple instruction set simulator to a cycle and resource accurate simulator with interlocks. This approach allows to specify, simulate and validate a processors with a given microarchitecture. Like the other constructs, the specification of the microarchitecture is on a high abstraction level. The interaction between the different constructs in ViCE-UPSLA is shown in Figure 7. The microarchitecture specifies the restrictions and the coordination for the execution steps of the instructions.

The description in ViCE-UPSLA uses the visualization of the components of the processor design domain. The basic elements of the microarchitecture are pipeline stages, which describe the sequential execution steps of the processor. Each pipeline stage groups the
function blocks, which can be utilized in the same cycle. ViCE-UPSLA supports any number of pipeline stages to describe a variety of microarchitectures.

The connected function blocks in the data flow graph of the microarchitecture express the data path of the instructions through the pipeline stages. Placed in a pipeline stage, a function block represents also a resource of this stage.

Respective to the resources in the pipeline, the behavioral instruction descriptions contain operands and operations which are linked to the pipeline resources. These are used to generate an interlocked pipelined processor simulator from specification.

To express microarchitectures like VLIW or MIMD with ViCE-UPSLA, we need to specify the corresponding behavioral description of the instructions and the microarchitecture. The function blocks for the specification of the data flow graph in ViCE-UPSLA are:

- arithmetical logical unit (ALU)
- read- and write-ports (R/W-Ports)
- multiplexer (MUX)

The ALU describes a fixed set of instruction groups which it is able to execute. The multiplexers describe building blocks to combine data from multiple sources into a new value by calculation rules. Through multiplexers, the execution of complex addressing modes in the pipeline can be expressed. The read and write ports specify the access ports to the registers of the processor. Other dependencies between the resources are considered from the pipeline’s paths.

In ViCE-UPSLA, the specification of the microarchitecture may also contain bypasses or forwardings [HP06]. To specify a bypass, the developer has only to place a bypass link between two elements of the data path. Like data paths, the bypasses can be used as directed edges in the data flow graph between any function blocks of the data flow graph. With a specified microarchitecture and instruction set, automated validation between these components is possible as well.

5 Simulator generator

With ViCE-UPSLA we generate simulators in the language C for a given processor architecture and a specific program, as shown in Figure 8. The CSim simulator generator emits the behavior of the processor while executing this specific program.

From a processor specification in ViCE-UPSLA, a code generator generates a C-framework for the constructs of the specified processor. The framework includes all steps of the instructions of the processor, split into cycle accurate actions, pipeline behavior, resource mappings etc. Also, the framework includes the computation functions for addressing modes, to prepare the values as a part of the execution of the instructions. As well as the data structure for the register set to define the system state.
For the generation of the simulator, the generated C-framework and the program are used. During the generation step, the assembler commands of the program are replaced by calls of C-functions from the library, which represent the processor’s instructions. For simulating the pipelined execution, the CSim generator interleaves the split cycle actions of the instruction computations according to the specified microarchitecture.

The interlocked or non-interlocked pipeline execution is part of the global specification options for a processor. With this option the developer can set, which configuration of the CSim generator is used to create the simulator. To create an interlocked processor simulator, a resource map for the instruction cycles is used. The resource map is generated from the ViCE-UPSLA specification of a processor. This map manages the instructions states for each cycle, for example currently used resources, required resources for the next cycle, released resources after the current cycle etc. The CSim generator uses the resource map to plan the execution of the next cycle for each instruction.

6 Related work

The existing languages and tools have different basic approaches to describe a processor’s architecture. The basic approaches can be classified by development goals, abstraction level, expression and description possibilities. In this section we will look at different languages like the nML formalism, ViDL or Lisa to classify the ViCE-UPSLA approach.

We use the classification of Prabhat and Dutt [PD08]. Figure 9 shows the relation between structural, mixed and behavioral ADLs. In this Figure, the extremely specialized languages are shown, i.e. MIMOLA [Zim97], as a strictly structural ADL and suitable only for synthesis or validation.

On the other extreme is the language ISDL [HHD97] as a behavioral ADL suitable for simulation and compilation. However, this taxonomy is an abstract view to classify the languages. More detailed is the Y-diagram [GK83] based on the abstraction level of the
description. Dieter Wecker [Wec08] describes the processors design process and explains the development steps in the Y-diagram, as shown in Figure 10. His exploration is based on the Gajski and Kuhn approach, which considers that the specification of the processor passes different abstraction levels during development and uses axes for the design of functional, structural and geometrical properties. At present, most processor or architecture description languages are mixed languages with two goals: specific construction and representation.

![Figure 9: Taxonomy of ADLs [PD08].](image)

![Figure 10: Y-digram [GK83, Wec08].](image)

ViCE-UPSLA is inspired by UPSLA, introduced in Section 4 of Kastens, Le, Slowik and Thies [KLST04]. UPSLA is a textual instruction set description language, suitable for compiler backend development and simulation of the processor architectures. A ViCE-UPSLA specification describes the processor specification part of UPSLA in visual form.
and expands this with the description of the processor’s microarchitecture. This allows to
generate a simulator as well as significant parts of the UPSLA specification and enables
the static and dynamic validation of the processor specification. Thus, ViCE-UPSLA is a
mixed ADL in terms of Figure 9.

The mapping from the type of the language to the development goals, shown in Figure 9,
is not stringent. For example the language ViDL [Dre12] is a behavioral description lan-
guage. The primary goal of the development is to generate the VHDL code, the secondary
goal is simulation. Starting with an instruction set specification and the desired target
clock frequency, the ViDL generator creates pipelined processor specifications in VHDL.
The developer has no direct control about the number of pipeline stages or other structural
processor constructs of the microarchitecture. The description in ViDL is made in textual
form. In contrast to ViDL, ViCE-UPSLA combines the specification for the instruction
set with the structural description of the microarchitecture. This allows to validate the
specification of the instruction set against the microarchitecture.

The architecture description language xADL [BPK12] contains components, similar to
ViCE-UPSLA, for the processor’s description, consisting of register file, storage elements
and functional units. The specification with the adlgen tool supports visual specification
of the data path with forwardings or pipelined execution, which can be used to generate
cycle accurate simulators. Distinct from ViCE-UPSLA, the instruction set of the processor
is extracted from the specification of the data path in the microarchitecture. This language
follows an approach in the opposite direction of ViDL, which uses the instruction set
description to generate a suitable microarchitecture. Thus, xADL is a structural ADL in
terms of Figure 9.

An example for a mixed ADL is the nML [FPF95, PD08] formalism with its behavioral
and structural specification, which includes execution behavior of the instructions and their
encoding. The specification abstracts from a detailed description of the data path and
other micro sequencing logic. This approach demonstrates very well the possibilities to
implement an instruction set simulator purely with concepts of functional programming
languages. However, the cognitive distance between domain specific terms of processor
development and nML is rather high. The processor specification is useful to generate
instruction simulators as software developer tools for programmers. Because of the miss-
ing constructs from the processor development in the description, the specification is not
suitable for validation of the architecture.

Tensilica’s domain-specific language TIE [Inc06] aims for comfortable design space ex-
ploration by means of processor instruction set extensions. TIE is a behavioral language
with a solid structural kernel. To achieve high acceptance and reduce specification effort,
Tensilica uses a fully implemented processor as a base processor architecture in its devel-
opment framework. On this basis, the developer has only to concentrate on the needed
extensions. This scenario is also supported by ViCE-UPSLA, with the option to let the
processor developer select the desired base processor. For its visualization, Tensilica
uses graphical components comparable to ViCE-UPSLA, like data flow graphs to show
compiler-suggested compositions for new instructions. An additional visualization is the
display of instruction formats, similar to a processor’s data sheet.
The Synopsys documentation gives an overview about the language Lisa [CoW08] and its supporting toolchain. Lisa is a very flexible language, which allows to describe a diverse spectrum of processors. The visualization of the language constructs is mostly table or text based. The hierarchically structured specification requires exact knowledge about the target processor’s microarchitecture right from the early stages of development. Similar to ViCE-UPSLA, behavioral specification of the instructions uses processor components expressed in the C programming language [CoW08, HL10, PD08]. In ViCE-UPSLA, visual components are used to describe the behavior, only operational descriptions given as C functions. In Lisa, the properties of the pipeline structure and the instruction’s resources have to be specified separately for every instruction. This raises the development effort and makes fundamental changes to the microarchitecture more costly. In ViCE-UPSLA, we introduce different views and abstract instructions to avoid these problems.

7 Applications of ViCE-UPSLA

With the visual language ViCE-UPSLA, we have designed a processor specification language on a high abstraction level. The language is based on domain specific terms and visualizations, which are intuitive for experts in the domain of processor design. The automatically generated instruction or microarchitecture simulators can be used for instruction set exploration or validation of processors.

ViCE-UPSLA has been successfully used as the graphical user interface to replace textual UPSLA processor specifications. The language has been employed to design software defined radio (SDR) and cryptography hardware extensions for a resource efficient VLIW processor [JPD+10]. The use of ViCE-UPSLA has helped to extend the scope of design space exploration [Jun11, JSGR10, JDP+10].

With enhancements implemented in recent versions of ViCE-UPSLA, we can now generate processor simulators automatically. To evaluate the generator, we have implemented a representative subset of a user mode version of an ARM processor [arm87, ARM00]. The resulting specification utilizes all language constructs of ViCE-UPSLA. Like the original ARM architecture, our subset is a load/store architecture with 16 32-bit registers. Forwarding in the pipeline is realized through the specified bypass structure. The processor’s instruction set includes arithmetical, logical, load/store and branch instructions. The execution of 32-Bit instructions with up to 4 operands is accurately done in the pipeline structure with one ALU. We use two load and one write port for the register file.

With this exemplary processor, we have demonstrated the expressiveness of our approach which allows to implement realistic processors with a multi-stage interlocked pipeline.

Future work includes more advanced case studies with different processors and matching applications programs to evaluate the simulator performance and further enhance the ViCE-UPSLA implementation.

Our next major goal is the integration of systematic methods for static and dynamic validation of the processor specification.
8 Acknowledgement

We would like to thank the project partners for the great collaboration and the German Federal Ministry of Education and Research (BMBF) for the funding of the national research project EASY-C (Enablers of Ambient Services and Systems Part C - Wide Area Coverage).

References


