Retargetable Timed Instruction Set Simulation of Pipelined Processor Architectures

Stefan Pees Vojin Živojnović Andreas Hoffmann Heinrich Meyr

Integrated Signal Processing Systems
Aachen University of Technology
Templergraben 55, 52056 Aachen, Germany

Abstract - This paper presents a retargetable timed instruction set simulator for programmable architectures. The simulator is automatically generated from a machine description in the language LISA and its generic processor model. Based on the LISA machine descriptions, pipeline hazards of the processor model can be specified and analyzed to fulfill the specific requirements of processor design. The development of the new retargetable simulator was necessary since existing approaches are based on processor models which are either too inaccurate or cannot deliver acceptable simulation speed. The first part presents the generic machine model of the retargetable simulator. In the second part, the description of the TI TMS320C6201 signal processor and the generated simulator is discussed.

1 Introduction

Driven by the new telecommunication applications, the amount of functions in signal processing systems is expanding enormously. To deal with rapidly growing system complexity, quickly changing system requirements, and late design changes, designers have increasingly turned to programmable architectures as a base for their DSP systems. Today, many modern telecommunication products, such as cellular phones, modems and networking devices feature one or more processor devices.

A main issue in the design of programmable architectures is the realization of the instruction pipeline [1]. Especially, since digital signal processors typically introduce a lot of parallelism and inhomogeneity, designers need to spend a lot of effort in exploring different architectures. In order to analyze architectural alternatives and tradeoffs, they need quantitative performance measurements based on real applications. Furthermore, absolute measures of the processing power are needed to evaluate changes to the instruction set and the processor architecture. These measures can be obtained by means of simulators that use bit- and cycle-true models.

Beyond the modeling capabilities, the usefulness of a simulator is very much characterized by the capability of detecting pipeline hazards which are not explicitly described. While data paths are relatively easy to verify, the same task for the control path is extremely complex and a main source for bugs [2].

There are different types of pipeline hazards [1]. Structural hazards arise from overlapping instructions competing for the same resource, e.g. registers, functional units, or busses which are typically resolved by means of pipeline stalls. Data hazards occur in case of read/write conflicts which are changing the order of register or memory accesses. Solving these hazards through the processor logic introduces forwarding logic or pipeline stalls (interlocks). Control hazards are produced by branches and exceptions which require mechanisms like flushing of the pipeline.

In highly pipelined processor architectures, these hazards have a significant impact on the overall performance and must be regarded during simulation to obtain accurate performance measurements. For this reason cycle-accurate processor models have to used. However, the development of cycle-true instruction set simulators is very lengthy, extremely error-prone and tedious process. So, tool support is essential even more for modern DSP architectures with increasingly complex pipeline control mechanisms.

2 Previous Work

Hardware description languages (HDLs) like VHDL or Verilog are widely used to model and simulate processors, but mainly with the goal of developing hardware. Using these models for instruction-level processor design has a number of disadvantages. They cover hardware implementation details which are necessary for hardware design and verification but which are not needed for performance evaluation and software verification. Moreover, the description of detailed hardware structures has a significant impact on simulation speed [3]. Another problem is that the extraction of the instruction set is a highly complex task and instruction set information, like e.g. the assembly syntax cannot be obtained from HDL descriptions.

A number of publications on machine models and retargetable simulation use instruction set description languages [4]. The approaches of nML [5,6] and ISDL [7] are based on latency specifications which cannot be used for simulation since they are not able to deliver cycle-accurate models of pipelined processor architectures.

ISPS [8] which is derivative from the language ISP [9] aims at a common processor description for code generation, simulation and synthesis. It incorporates a rich set of control mechanisms to describe parallelity and synchronization of processes, but the synchronization mechanisms are not adequate to model pipeline hazards.

The language Maril introduces resources which are allocated by operations, but latencies for particular instruction pairs have to be specified by the user [10]. Maril is part of the Marion system which uses reservation tables for code generation. However, basic pipeline operations like flushes are not supported. A similar approach based on standard reservation tables is used for VLIW compilation [11]. In both cases a standard reservation table approach [12] was used which is unable to capture all details of the pipeline, such as data/control hazards and pipeline flushes.

In the Rapide system [13], systems are modeled as partially ordered event sets by specifying causal and timing relationships between events. However, pipeline flushes cannot be modeled with this approach.

Our wish to capture more detailed timing information than available at the instruction set level, motivated the introduction of operation-level behavior and scheduling description. In terms of pipeline sequencing representation, LISA follows the same main idea of reservation tables and Gantt charts, as in [10] and [11]. However, in order to enable additional modeling of data/control hazards and pipeline flushes, we extended the modeling ability of Gantt charts by introducing L-charts and operation descriptors.

3 Generic Machine Model

In order to enable cycle/phase-true modeling, instructions have to be partitioned into operations as basic schedulable units. At each control step *t* the admissible operations form the transition function F_t which changes the machine state. At the next control state *t+1* the new set of admissible operations is determined. In a pipelined processor e.g., these operations may represent the pipeline stages which operate in parallel during one processor clock cycle.

The instruction sequence in the program determines the operations that wait for execution. Admissible operations are selected from precedence and resource constraints which are specified for the operations of each instruction [14]. The precedence specifies partial ordering of operations for one instruction. The ordering between instructions is determined by the program. The example in Figure 1 shows the operation precedence graph of four instructions in a processor with the pipeline stages IF, ID, EX, and WB. Horizontally, the operations of one instruction are ordered explicitly by the precedence, vertical ordering is provided through the program sequence.

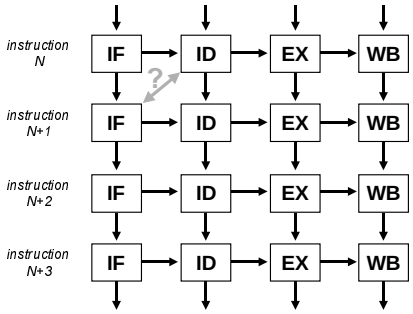

Figure 1: Partial Operation Ordering.

But the ordering is not complete, e.g. there is no ordering between *ID* of instruction *N* and *IF* of instruction *N+1*. In order to determine the execution timing of these operations a scheduler provides this ordering. In our retargetable simulator, we use an As Soon As Possible (ASAP) scheduler for this task, since it covers the class of our target processor architectures which use ASAP sequencing. However, processors with out-of-order execution have to be excluded.

The ASAP scheduler transforms the partial ordered operations of Figure 1 into a set of fully ordered operations as shown in Figure 2. Here, the operations that are executed in the same cycle are shaded in the same style.

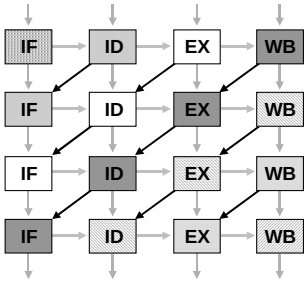

Figure 2: Complete Operation Ordering

The operations in Figure 2 show the situation of instruction execution without hazards. For the illustration of hazards, the precedence and resource constraints can be depicted in L-charts, a representation similar to Gantt-charts. L-charts are derived from Gantt-charts by converting the time axis into a precedence axis. Furthermore, L-charts introduce the annotation of *operation descriptors*. For our retargetable simulator, we used the following operation descriptors:

- *Allocate*: The operation can only be scheduled, if the resource to be allocated is available. In this case, the resource is allocated by the operation.
- *Require*: The operation can only be scheduled, if the required resource is available, but no resource allocation is performed. This is marked with ***r:*** in L-charts.

- *Kill/Replace(op)*: The operations that allocate or require the resource in the current cycle are replaced by the specified operation *op*. This is indicated with ***k(op):*** in L-charts.

Operation descriptors can be conditional to enable dynamic operation scheduling.

Modeling of Structural Hazards

Figure 3 shows the L-charts for a sequence of instructions *N*, and *N+1* with their respective operations, e.g. instruction *N* consists of the operations *O1* up to *O5* which allocate the resources *IF*, *ID*, *EX*, and *WB*. The operation sequencer performes the operation scheduling based on the operation descriptors.

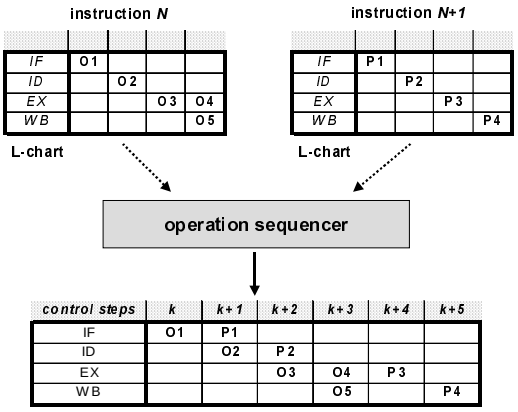

Figure 3: L-chart Scheduling

Below the operation sequencer, the scheduling result in form of the operation timing is depicted. In control step *k+3* the operations *O4* and *P3* produce a structural hazard by competing for the resource *EX*. The ASAP scheduler selects *O4* whereas *P3* is delayed to the following cycle.

Modeling of Control Hazards

Flushing of pipeline stages is a common mechanism found in programmable DSP architectures to solve control hazards. We assume the following program sequence to illustrate how control hazards can be modeled in LISA. The sequence begins with a branch to the target instruction *X*. The two instructions *C* and *D* following the branch shall not be executed and are cancelled.

```
        B target
        C
        D
        ...
target: X
        Y
```

Figure 4 depicts the L-chart of the branch instruction *B* with four kill/replace operation descriptors that replace the respective operations with NOPs. The other instructions share the same diagonal L-chart pattern. The resulting scheduling diagram shows how the operations of instruction *C* and *D* are simply replaced without affecting the timing of the other instructions.

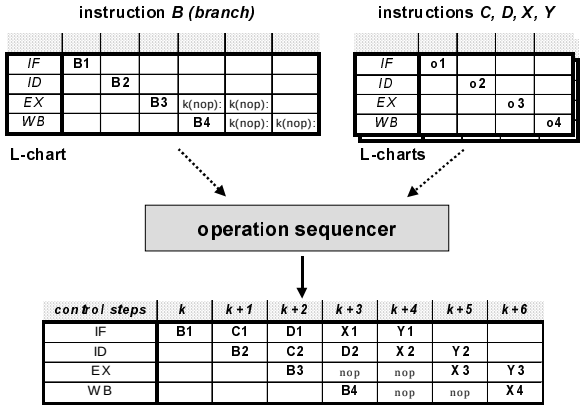

Figure 4: Kill/Replace

Besides flushing the pipeline, also interrupts can be modeled based on the kill/replace operation descriptor by injecting the instruction word of the interrupt instead of the NOP into the pipeline.

4 LISA language

LISA processor descriptions in general consist of resource declarations and operation definitions. Declarations in the RESOURCE section are global and define all storage elements that represent the processor state in the simulator. At the same time these declarations list the valid resources that may be allocated or required by operations.

Operation definitions are structured into several sections that describe different operation attributes which are necessary for simulators, assemblers, disassemblers and code synthesis:

- The CODING section defines the operation opcode and the fields for operands, parameters, and operation modes.
- The mnemonics and parameters of the assembly language are provided in the SYNTAX section.
- The BEHAVIOR section lets the designer specify the behavioral code of the operation in C or C++ for use in the simulator.

Operation properties can be shared by means of the CLASSES which are very similar to the or-rules in nML [5]. Further, operations can be structured hierarchical. So, operations can comprise sub-operations. The top of this hierarchical structure is the operation *main*, which is executed in every control step.

5 TI TMS320C6201 LISA Description

In order to evaluate the implementation of the generic machine model, the TI TMS320C6201 digital signal processor was modeled with LISA and the simulator was generated. This processor was chosen because of its complex pipeline mechanisms and the high amount of parallelity. Our description of the processor produces a bit- and cycle-accurate machine model. Resources are used to model the mechanisms of the instruction pipeline, such as partial and full pipeline stalls or pipeline flushes.

The instruction pipeline of the C6201 consists of two parts - the fetch pipeline has the task to load the 256 bit wide fetch packets with eight instruction words from the program memory and the execute pipeline that dispatches execute packets of one to eight instructions and feeds the functional units [15]. In general, the execute pipeline works continuously and is mainly stalled in case of memory waitstates. Since execute packets may be smaller than fetch packets, the fetch pipeline is frequently stalled and has to wait until all execute packets of the previous fetch packet have left the last stage of the fetch pipeline.

Although the C6201 is described by TI as a VLIW derivative, we realized our model based on the 32 bit wide instruction words because branches can be misaligned to fetch packet borders and target any instruction word in the fetch packet. The C6201 fetch pipeline consists of the stages *PG*, *PS*, *PW*, *PR*, *PG*, and *DP*. The LISA model of the C6201 fetch pipeline is depicted in Figure 5. All instructions share the same resource allocation in the fetch pipeline as shown in the L-chart. We defined an array of eight resources for each pipeline stage. A program word index (*pwi*) provides the address in the range from 0 to 7 within the fetch packet. Additionally, the operations require the resource element with index 7. So, advancement in the pipeline is only possible if the last instruction of the previous fetch packet has left this respective stage.

PG[pwi]	O1					
PG[7]	r: O1					
PS[pwi]		O2				
PS[7]		r: O2				
PG[pwi]			O3			
PG[7]			r: O3			
PW[pwi]				O4		
PW[7]				r: O4		
PR[pwi]					O5	
PR[7]					r: O5	
DP[pwi]						O6
DP[7]						r: O6

Figure 5: L-chart of the C6201 Fetch Pipeline

In the dispatch stage (*DP*) the execute packets are determined and forwarded to the execute pipeline. Execute packets are composed by setting the parallel-bit of consecutive instruction words. Figure 6 shows the L-charts of the *ADD* instruction for both settings of the parallel bit.

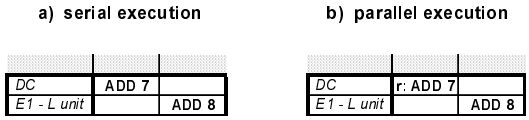

Figure 6: L-charts of the ADD instruction

In case of serial execution, the instruction allocates the resource of the decode stage (*DC*) and prevents further instructions from entering this stage. In the case of parallel execution, the instruction only requires *DC* and allows successors to join the current execute packet.

6 Retargetable Simulator

The generic machine model was implemented in a retargetable simulator that is generated based on LISA descriptions as shown in Figure 7. The retargeting process is based on a simulator generator that parses the LISA description and implements the principles of the generic machine model.

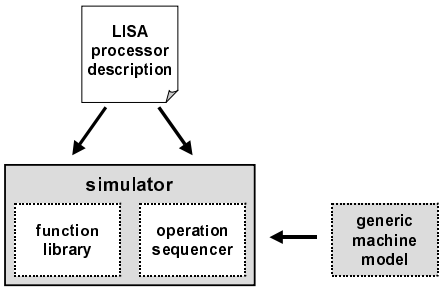

Figure 7: Retargetable Simulator

The simulator consists of the simulation function library and the operation sequencer. The operation sequencer is composed of two parts - the mechanisms of the generic machine model and the LISA processor description that parameterizes the machine-independent operation scheduler. After translating the generated C or C++ code of the function library, the simulator can load and execute application programs. The debugger interface displays the state of the registers and memories which are represented by resources and allows the user to change these values. The operation sequencer outputs consecutively the scheduling tables and reports all pipeline hazards during simulation. So, the designer can observe the flow of instructions in the pipeline and how pipeline hazards are resolved.

Our LISA model was successfully verified on some applications by comparing the register set values after each clock cycle with the simulator `sim6x`[1] from TI. This method is applicable because of the processors load-store architecture. The preliminary measurements turned out that the simulation speed of our gen-

[1] Texas Instruments TMS320C6x C Source Debugger/Simulator, Rel. 1.00 for workstation, 1997.

erated simulator was more than 50% higher than the speed of `sim6x` without using techniques of compiled simulation.

7 Modeling Experience

The L-charts of LISA enable compact description of pipelined processor architectures and are able to deliver cycle-true machine models. To meet the specific requirements of processor design, the retargetable simulator allows the analysis of pipeline mechanisms and the detection of hazards. The description of programmable architectures based on L-charts is very useful to model structural and control hazards, because the behavior of individual instructions is described separately rather than the complex pipeline controller in a whole. Interference between overlapping instructions can be detected instead of specifying them. However while modeling the C6201, we found the case where the description based on the L-charts was not obvious to find and an explicit description of the operation control was more appropriate. The behavior of the multi-cycle *NOP* instruction is different from all other instructions. If the delay slots of a branch complete while a multi-cycle *NOP* is still executing, the *NOP* must be cancelled. However, the *NOP* may execute in any of the eight execution units (resources) which makes it hard to describe this situation by means of the kill/replace operation descriptor.

In our experience, modeling of programmable DSP architectures based on L-charts is an efficient approach especially for the exploration of processor architectures when the processor architecture is not fully specified. If however an existing processor shall be modeled, the designer may want to explicitly specify parts of the pipeline controller. The description of the operation behavior section must be used in this case.

The modeling of data hazards as discussed in [14] could not be analyzed since the C6201 architecture implies extensive use of software pipelining which introduces lots of data hazards. For processors with this programming model, automatic data hazard detection is not possible.

8 Conclusion

In the design of digital signal processors, only cycle-accurate simulators are able to deliver reliable quantitative performance measurements. In this paper, a retargetable simulator has been presented which is automatically generated from LISA descriptions. This simulator implements the mechanisms of L-chart modeling and ASAP operation scheduling which allows the description and detection of pipeline hazards for a large class of processor architectures. Our approach was successfully approved by modeling the complex C6201 processor architecture with LISA and verifying the simulator with the executable model from TI.

In our future work, we will explore the issues of retargetable compiled simulators to take advantage of the highest possible simulation speed.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., Second Edition, 1996.

[2] J. R. Burch and D. L. Dill. "Automatic Verification of Pipelined Microprocessor Control", *International Conference on Computer-Aided Verification*, LNCS 818, Springer-Verlag, June 1994.

[3] J. Rowson, "Hardware/Software co-simulation", in *Proc. 31st Design Automation Conference*, 1994.

[4] M. Hartoog, J. Rowson, et al., "Generation of Software Tools from Processor Descriptions for Hardware/Software Codesign.

[5] A. Fauth, M. Freericks, and A. Knoll, "Generation of Hardware Machine Models from Instruction Set Descriptions", in *Proc. of the IEEE Workshop on VLSI Signal Processing*, 1993.

[6] A. Fauth, J. van Praet, and M. Freericks, "Describing instruction set processors using nML", in *Proc. European Design and Test Conf.*, Paris, Mar. 1995.

[7] G. Hadjiyiannis, S. Hanono, S. Devadas, "ISDL: An Instruction Set Description Language for Retargetability", in *Proc. of the 34th Design Automation Conference*, pp. 299-302, 1997.

[8] M. Barbacci, "Instruction Set Processor Specifications (ISPS): The notation and its application", *IEEE Transactions on Computers*, Vol. C-30, No. 1, pp. 24-40, Jan. 1981.

[9] C.G. Bell, A. Newell, *Computer Structures: Readings and Examples*, McGraw Hill, New York, NY, 1971.

[10] D. Bradlee, R. Henry, and S. Eggers, "The Marion system for retargetable instruction scheduling", in *Proc. ACM SIGPLAN'91 Conference on Programming Language Design and Implementation, Toronto, Canada*, pp. 229-240, 1991.

[11] B. Rau, "Machine-Description Driven Compilers for VLIW processors", in *3rd Int. Workshop on Code Generation for Embedded Processors*, Witten, Germany, Mar. 1998.

[12] P. Kogge, *The Architecture of Pipelined Computers*, McGraw Hill, New York, NY, 1981.

[13] D. Luckham, "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial David C. Luckham, DIMACS Partial Order Methods Workshop IV, Princeton University, July 1996.

[14] V. Živojnović, S. Pees, H. Meyr, "LISA – Machine Description Language and Generic Machine Model for HW/SW Co-Design", in *Proc. of the IEEE Workshop on VLSI Signal Processing*, San Francisco, pp. 127-136, Oct. 1996.

[15] Texas Instruments, Inc., *TMS320C62xx CPU and Instruction Set*, Jan. 1997.