**Processor Design and Tool Chain Automation Using ADL**

*Ming-Hwa Wang, Ph.D.*

**COEN 388 Principles of Computer-Aided Engineering Design**

**Department of Computer Engineering**

**Santa Clara University**

**Processor Design Process**
- architecture exploration
  - instruction-set design, micro-architecture, hardware/software partitioning
- hardware implementation
  - realization of HDL model, synthesis
- software application design
  - macro processor, assembler, linker, loader, instruction set simulator (ISS), debugger, compiler, application
- system integration and verification
  - system integration, simulation, co-simulation, verification, validation

**Design Methodology**
- traditional design methodology: design manually
  - results in a long, labor-intensive process requiring highly skilled engineers with specialized know-how
- need to hire architectures, hardware engineers, software engineers, application engineers, system engineers, verification engineers, and validation engineers
- communication is difficult between these groups of engineers
- hardware simulation too slow, turn-around time too long
- sequential design process
- ADL-based, automated design methodology: a better way
  - parallel/concurrent design process:

**ADL and Automatic Tool Chain Generation**
- automatically generate tool chain (macro processor, assembler, linker, loader, instruction set simulator, and even compiler or HDL) from a processor’s architecture description using ADL (Architecture Description Language)
- commercial EDA tools
  - CoWare’s LisaTek
  - ARM’s MaxCore
  - Target’s Chess/Checkers

**Architecture Description Language (ADL)**
- instruction-set centric languages
  - for retargeting high level language (HLL) compilers: nML, ISDL, Valen-C, CSDL, etc.; e.g., Chess/Checkers uses nML
- architecture-centric languages
  - for synthesizing HW implementation: Mimola, COACH, AIDL, etc.
  - instruction-set and architecture oriented languages
  - for both: Lisa, MDES, PEAS, RADL, EXPRESSION, MetaCore, ASIA, etc.; e.g., both LisaTek and MaxCore use Lisa
- others
  - Jazz for VLIW processor, Xtensa for reconfigurable core

**ADL Processor Design Methodology**
- an automated environment supporting the complete design flow of application-specific instruction-set processors (ASIP)
- with architecture specification described in ADL, tool chain can be automatically generated for architecture exploration, software application design, hardware implementation, system integration, and verification/validation

**LisaTek by CoWare: The LISA Language**
- models
  - resource and memory model, instruction-set model, behavior model, timing model, micro-architecture model
- abstraction levels
  - abstraction of architecture
    - data-flow model, instruction-set model, micro-architecture model
  - abstraction of time
    - high level language statement, instruction, cycle, phase
- a Lisa model contains a resource and a operation tree starts with main and decode functions; sections in each operation are:
  - declare section, coding section, syntax section, behavior/expression section, activation section

**Resource and Memory Model**
- resource and memory models are defined in the resource section
- simple resource:
  - internal: single registers, flags, vectors (register file, memory)
  - external: buses, pins
- pipeline structures for instructions and data-paths
  - pipeline registers or latches between each pipeline stage
  - memory maps that locate resources in the address space (for object code linking)
  - storage elements represent the state of the processor

**Instruction-Set Model**
- the instruction-set model identifies valid combinations of hardware operations and admissible operands
  - compiler and assembler can identify instructions based on this model; the same information is used at the reverse process of disassembling and decoding
  - the coding section describes the binary image of the instruction word
- The syntax section describes the assembly syntax of instruction, operands, and execution mode.
- The declare section contains local declarations of identifiers and admissible operands.
- The Instruction-Set model can be automatically generated by spreadsheet-like tools.

**Behavior Model**
The behavior model abstracts the activities of hardware structures to operations changing the state of the processor for simulation purposes.
- The behavior section contains pure C/C++ code that is executed during simulation.
- The expression section defines the operands and execution modes used in the context of operations.

**Timing Model**
The timing model specifies the activation sequence of hardware operations and units.
- The activation section is used to activate other operations in the context of the current instruction; the activated operations are launched as soon as the instruction enters the pipeline stage the activated operation is assigned.
- The activation of operations can be seen as the setting of control signals by decoders in hardware.
- Control hazards, data hazards or structural hazards can cause multiple pipeline stalls or even cancellation of the instruction execution; predefined functions stall, insert, shift, flush, and execute can be used.

**Micro-Architecture Model**
The micro-architecture model contains additional architectural information that is required to enable HDL code generation from a more abstract specification.
- Hardware operations can be grouped to function units (entities or modules); HDL code generator generates empty frames and the according interconnections to processor resources.
- The behavior code is discarded for HDL code generation.

**Architectural Exploration**
Architecture exploration by iterative refinement of the architecture.
- First draft of the architecture (“educated guess”).
- Get quantitative data on the performance of the architecture and application by the generated tool chain to evaluate the suitability of the chosen instruction set, the resource utilization, the executed speed.
- The architecture is changed according to the benchmarking and profiling results iteratively until the design goal is met and a best-fit between application and architecture is obtained.
- Every time the architecture is changed, all tools need to be adapted as well.

**Profiling-Based Architecture Refinements**
- High level language (HLL) algorithmic kernel model.
- Parallelized HLL algorithmic kernel model.
- Data-path model.
- Instruction-based model.
- Cycle-based model.
- Register transfer level (RTL) micro-architecture model.

**Hardware Implementation**
- Hardware implementation in synthesizable HDL code is generated either automatically by the tool (for control path) or manually inserting hand-optimized code (for data path) into the generated frames of the HDL model.
- The HDL code processed through synthesis tool to get gate-level model, runplace and route to get chip layout for tape-out.
- Only one ADL model needs to be maintained.

**Application Tool Development**
- Automatically generated tool chain: macro processor, assembler, linker, loader, simulator, and graphic debugger.
- Options for simulation performance: interpretive, compiled or just-in-time (JIT) compiled code (requires no external memory and no self-modifying program).
- Options for accuracy: instruction-accurate, cycle-accurate.
- GUI debugger displays resources like pipeline stages, registers, I/O ports, memory, and optionally flags.

**System Integration and Verification**
- A system-on-chip (SoC) contains micro-controller, DSP, shared memory, dedicated logic (ASICs) and on-chip communication.
- System integration of heterogeneous components (e.g., 3rd party IP) using various languages, formalisms, and tools into a single framework.
- Enable seamless integration by using application programming interface (API) to control and access the processor resources.
- Provide user-friendly heterogeneous multiprocessor debugging environment.