Abstract

System-level design is characterized by a behavioral specification and heterogeneous hardware/software implementations. Exploring the design space is essential for good design. Specifying and managing complex design flows, tracking dependencies and tool invocations, and maintaining consistency of design data and flows are key issues that enable efficient design space exploration.

We present a framework that manages these complex issues in the design process, transparent to the user. The framework, called DesignMaker, is implemented within the Ptolemy environment. The features of DesignMaker are illustrated with reference to an example design flow for multiprocessor synthesis. The end-objective is to embed the DesignMaker under a system-level codesign assistant.

**A Typical Design Flow for Hardware/Software Codesign**

- **Design Specification**
- **Design Constraints**
- **Design Assistant**
- **Area/Time Estimation**
- **HW/SW Partitioning**
  - Hardware Synthesis
  - Interface Synthesis
  - Software Synthesis
- **Netlist Generation**
- **System Simulation**

**Synthesized System** (Layout + software)

---

**Design Assistant: version 1**

- **Algorithm Partitioning**
- **Bitwidth Selection**
- **Interface Selection**
- **Algorithm Retargeting and Scheduling**
- **Software Synthesis**
- **Hardware Synthesis**
- **Synthesized System**

---

**Limitations of the Design Assistant-v1**

- **Sequential Design Flow**: cannot re-run isolated design steps
- **Inflexible Design Flow**: hardwired design methodology and tools

**A possible solution**: Menu Driven Approach
- Solves the sequential flow problem; but hardwired
- No automatic dependency analysis and tool invocation

**Desired**: Menu-Driven System

- **Simulate**
- **Partition**
- **Synthesize HW**
- **Simulate SW**
- **Synthesize IFC**
- **Simulate**
- **Analyze**

**Need a mechanism to manage the design Methodology, Flow, and Data.**

---

**Design Space Exploration**

- Managing the complexity of the design process is non-trivial.

**Requirements:**

- Modular and configurable flow specification mechanisms
- Mechanisms to systematically track tool dependencies and automatically determine the sequence of tool invocations
- Managing consistency of design data, tools, and flows
**Outline**

- **Motivation for systematic management of the design methodology**
- **Design Methodology Management (DMM) infrastructure**
- **Implementation**
- **An example**
Representation

Distributed datastructure

Flow Management

Tool is enabled: when all "required" input ports have data
Tool is invoked: when at least one of its dependencies is alive
Tool is executed: generates data on all required outputs, conditionally on all optional ports.

Flow invocation mechanisms

data driven          demand driven

Outline

• Motivation for systematic management of the design methodology
• Design Methodology Management (DMM) infrastructure
• Implementation
• An example
DMM Domain in Ptolemy

Design flow: specified as a graphical netlist

Tools: encapsulated within basic blocks (Star)
  — have “required” and “optional” ports, parameters

Flow definition: supports conditionals, iterations, hierarchy (Galaxies)

Tool encapsulation: involves writing scripts to call “programs”
  — Ptolemy functions
  — stand-alone programs (with their own GUIs)
  — programs on remote filesystems

DMM attributes: stored in Oct database

Flow Manager: Target (called DesignMaker)
  — supports data-driven and demand-driven flow execution
  — resolves tool dependencies, automatically invokes tools

Scheduler: Combination of dataflow and event-driven semantics
  — detects deadlocks

DMMCodeGenerator.pl

```
defstar {
    name {CodeGenerator}
    domain {DMM}
    input { name {graph} type {message} }
    input { name {numProcs} type {message} }
    output { name {codeFileNames} type {message} }
    go {
        graphName = graph.getFileName();
        name = getName(graphName);
        domain = getDomain(graphName);
        handle = getHandle(graphName);

        procFileNm = numProcs.getFileName();
        fp = fopen(procFileNm,"r");
        fscanf(fp,"%d",&numberProcs);

        // run tcl command for code gen.
        Tcl_VarEval(ptkInterp,"ptkGenCode", name,domain,handle,numberProcs);

        // generate output file names
        codeFileNames.putFileName(fout);
    }
}
```

Examples

**Multiprocessor synthesis**

Application (graph) | Throughput | IPC architecture

**Hardware/software codesign**

An Example Design Flow: Multiprocessor Synthesis

The goal is to synthesize the system with the minimum number of processors for a specified throughput requirement. The design flow and the control panel are shown.
An Example Design Flow: Components for Codesign

When the partitioning tool is invoked, it brings up the graph to be partitioned (1), and a control panel (2) that allows for manual partitioning. The area/time estimator returns the execution time and implementation area values by invoking specific tools on remote filesystems. Automated partitioning and synthesis tools have also been developed.

Conclusions

— Design space exploration is a key to system-level design
— Critical Issues:
  • Specifying and managing complex design flows
  • Tracking tool dependencies
  • Automated flow invocations
  • Maintaining consistency of design data and flows
— DMM: an infrastructure for design methodology management
— Current work:
  • Implement the codesign system in this framework