High-Level Synthesis of Control and Memory Intensive Applications

Peeter Ellervee

Electronic System Design Laboratory
Royal Institute of Technology
ESDlab, Electrum 229, S-164 40 Kista, Sweden

Outline
- Introduction
- Prototype High-Level Synthesis
- IRSYD - Internal Representation
- Pre-packing of Data Fields in \( N \)
- Segment-Based Scheduling
- Unified Allocation and Binding
- Conclusions / Future Work

Introduction

Efforts needed to design a 20 kgate circuit (Modified from [P. Michel et al. 1992])

What is High-Level Synthesis?

Behavioral Domain
- System Specification
- Algorithm
- Register-transfer specification
- Boolean Equation
- Differential Equation
- Circuit

Physical Domain
- Logic Synthesis
- Physical Design
Sub-tasks of HLS

- **Partitioning** divides a behavioral description into sub-descriptions in order to reduce the size of the problem or to satisfy some external constraints.
- **Allocation** is the task of assigning operations onto available functional unit types available in the target technology.
- **Scheduling** is the problem of assigning operations to control steps in order to minimize the amount of used hardware. If performed before allocation (and binding), it imposes additional constraints how the operations can be allocated.
- **Binding** assigns operations to specific instances of unit types in order to minimize the interconnection cost.

Deficiencies of Traditional HLS Approaches

- HLS research in the past -- partitioning, scheduling and allocation
  - data-flow dominated applications (DSP)
- But HLS ≠ partitioning, scheduling and allocation
- The main problems are:
  - there exist a need for application specific synthesis strategies which more efficiently could cooperate with the features of a specific application domain;
  - existing internal representations can not encapsulate details of a specification without some loss of information, therefore more general representations are needed;
  - the need for efficient estimation algorithms, especially in the deep sub-micron area where wiring dominates both gate delay and chip area.
- A specialized approach is needed for Control and Memory Intensive Systems (CMIST)

Distribution of Operation Types

HLS Sub-tasks Targeted in the Thesis

- **Internal representation** which encapsulates primarily the control flow to support CMIST synthesis strategy
- **Pre-packing of data fields** of dynamically allocated data structures to optimize the memory bandwidth in data transfer intensive (DTI) applications
- **Segment-based scheduling** of control dominated applications which targets as-fast-as-possible (AFAP) schedule and takes a special care of loops
- Fast heuristics for **unified allocation and binding** of functional units and storage elements
xTractor -- Synthesis Flow

- IRSYD generator (translator)
- Memory Extractor
- State Marker (scheduler)
- Allocator/Binder
- RTL HDL Generator

xTractor -- Target Architecture

- merged FSM & DP
- separate FSM & DP

xTractor -- Component Tools & Synthesis Steps

- generate CDFG
- propagate constants
- simplify operations
- list memories
- edit memory mapping
- map / extract memories
- propagate constants
- mark states
- allocate & bind FUs
- allocate & bind registers
- generate HDL (@ RTL)

IRSYD -- Internal Representation for System Description

- Specifying large and complex systems - combination of many description paradigms
- Integration of different tools for analysis and synthesis
- Features needed to preserve the semantics of the original specification:
  - static and dynamic structural and functional modularity and hierarchy;
  - sequence, concurrency, synchronization and communication among various sub-systems;
  - representation of abstract data types;
  - mechanism for representing tool dependent information;
  - mechanism for communication among various tools;
  - reuse of design or behaviors.
IRSYD -- Concepts

• Based on Flow Chart
  - Unified graph representation for control flow and data flow
  - Concepts to model hierarchical and concurrent systems
  - Different communication mechanisms

• Two distinct approaches to preserve semantics of the source languages:
  - Union of Concepts
  - Primitive Atomic Concepts

• IRSYD -- based on a mix of the two approaches
  - A set of Primitive Concepts (not necessarily atomic or minimal) is used
  - A special attribute mechanism to avoid loss of information
  - Conventions to unify translation from various source languages

IRSYD -- Structure

• Module -
  - the basic unit of hierarchy
  - Sub-modules
  - Declarations
  - Processes

• Sequential Computation
• Parallel Computation
• Communication
• Data Representation

Pre-packing of Data Fields in Memory Synthesis

• Dynamically allocated data structures
  - flexible descriptions of complex applications
  - specialized optimization strategies needed

• Past -- applications requiring large storage implemented in software

• Present -- embedded software or hardware, customized memories

• Related work:
  – Foreground memory (register) allocation
  – Array-oriented versus scalar-oriented approaches
  – Recent non-scalar oriented memory allocation approaches:
    - MeSA -- static memory allocation (U.C.Irvine: L. Ramachandran et al. 1994)
    - Horizontal and vertical array mapping (CMU: H. Schmit et al. 1997)

Data Mapping Opportunities

Intuitive approach:
  - same base address
  - candidate fields in single basic block (or very “close”)

```plaintext
// ...a = p1 -> field_0;
// ...b = p1 -> field_1;
// ...p2 -> field_0 = 3;
// ...p2 -> field_1 = 254;`
**Dependency Cases: Classification**

- **read-after-read**
- **write-write**
- **read-read**
- **single-write**

**Compatibility Graph: Definition**

- Sum of node weights == number of accesses
- Differences in edge weights --> gains / losses in number of accesses

**Compatibility Graph: Clustering**

- **CDFG #2**

**Results of Experiments**

- Four subsets from real-life ATM cell processing applications
- Significant reduce in the number of memory accesses -- from 40% to 85%
- Reduce in the size of memories -- to 20%
- Significant reduce in the power consumption -- from 40% to 60%
Segment-Based Scheduling

- Requirements of control dominated applications:
  - control constructs
  - operator chaining

- Related work:
  - path based scheduling – Camposano 1990
  - tree-based scheduling – Huang et al. 1993
  - loop-directed scheduling – Bhattacharya et al. 1994
  - dynamic loop scheduling – Rahmouni et al. 1994
  - etc.

- Segment-Based Scheduling
  - control graph is divided into segments during the graph traversal
  - segmentation drastically reduces the number of possible paths

Traversing -- State Oriented vs. Transition Oriented

- prob(p1) > prob(p2) traverse p1 before p2
- prob(p1) < prob(p2) traverse p2 before p1

Scheduling Loops

- do {
  "loop body"
} while (condition):
  type "loop-while"

- while (condition) {
  "loop body"
}
  type "while-loop"

State Marking Rules

Rule 1: between two WAIT marks
Rule 2: between two port accesses separated by a WAIT mark
Rule 3: between two synchronization signal accesses
Rule 4: memory read
Rule 5: after memory write
Rule 6: before loop body

Legal range for the state mark
Results of Experiments

- Bubble-sort example -- loop + memory accesses
- Reduced number of states
- Reduced number of registers
- Speed-up of the algorithm
  - number of clock-cycles reduced by 35%

Unified Allocation and Binding

- Three tasks traditionally (most of the tools implement separately):
  - Mapping Arithmetic Operations onto Functional Units
  - Mapping Variables onto Storage Units
  - Mapping Interconnections onto Multiplexers
- Mapping onto graph problems:
  - Conflict graph coloring
  - Compatibility graph clique partitioning
- Unified allocation and binding
  - No separate task for Interconnections:
    - Allocation and binding tasks of units are merged into single step
  - Weighted conflict graph coloring

Binding Tasks

Operations

- 8-bit adder & 9-bit subtracter
- 2 9-bit multiplexers & 9-bit adder/subtracter

Variables

- 8-bit & 9-bit registers
- 9-bit multiplexer & 9-bit register

Coloring Examples

H1
- The most expensive nodes first:
  V0, V3, V8, V1, V2, V5, V9, V6, V7, V4
- cost = 275 (65%)

H2
- The cheapest nodes first:
  V0, V8, V7, V5, V1, V2, V9, V3, V6, V4
- cost = 264 (63%)

H3
- Dynamic selection:
  V0, V3, V8, V1, V2, V5, V9, V6, V7
- cost = 275 (65%)

H4
- Random selection:
  V0, V3, V8, V1, V2, V5, V9, V6, V7
- cost = 264 (63%)
Results of Experiments

- More than 200 real-life and 150 random coloring tasks

- Real-life examples
  - Average penalty: 1.2% to 4.8%
  - Maximum penalty: 4% to 15%

- Random examples
  - Average penalty: 1% to 4%
  - Maximum penalty: 3% to 12%

Synthesis Results

- F4/F5 -- pre-packing of data fields
  - 4% smaller area
  - 40% lower power consumption in memories

Future Work

- IRSYD & xTractor
  - compilers for high level front-end languages
  - data and control flow transformations
  - estimation tools

- Pre-packing of data fields
  - improved compatibility graph clustering (hyper-edges, long dependency chains, etc.)
  - better area and power estimation algorithms

- Segment-based scheduling
  - incorporation of various data-flow based scheduling techniques
  - control-flow to data-flow transformations (and vice versa)

- Unified allocation and binding
  - detection and removal of false loops
  - enhancements of greedy heuristics

Summary

- Internal representation which encapsulates primarily the control flow to support CMIST synthesis strategy

- Pre-packing of data fields of dynamically allocated data structures to optimize the memory bandwidth in data transfer intensive (DTI) applications

- Segment-based scheduling of control dominated applications which targets as-fast-as-possible (AFAP) schedule and takes a special care of loops

- Fast heuristics for unified allocation and binding of functional units and storage elements

- Prototype HLS tool -- xTractor

- Synthesis results of industrially relevant applications