SOFTWARE SYNTHESIS FOR MULTI-CORES

Alberto Sangiovanni-Vincentelli

The Edgar L. and Harold H. Buttner Chair of EECS
University of California at Berkeley

Co-Founder, CTA and Member of the Board
Cadence Design Systems
Intel MXP5800 Architecture

- Highly concurrent heterogeneous architecture
- Eight Image Signal Processors (ISPs) with mesh interconnect
- Low overhead blocking read, blocking write operations
Inside Each ISP

- Heterogeneous programming elements (PEs)
- Two hardware accelerators
- Memory command handler is a programmable DMA
The JPEG Encoder Application

- Algorithm operates on 8x8 blocks from image
  - RGB to YCbCr color space conversion
  - Forward DCT
  - Quantization
  - Run-length encoding
  - Huffman compression

- Representative multimedia application
  - Sub-blocks appear in other image/audio/video codecs
  - Features some data dependent communication
Separation of Concerns (ca. 1990)

Development Process

- Specification
- Analysis
- Implementation

Behavior Components
- C-Code
- Matlab
- Dymola

Virtual Architectural Components
- CPUs
- Buses
- Operating Systems

Platform
- ECU-1
- ECU-2
- ECU-3

Behavior
- f1
- f2
- f3

Mapping

Performance Analysis

Refinement

Evaluation of Architectural and Partitioning Alternatives
Platform-Based Design

Platform: library of resources defining an abstraction layer with interfaces that allow legal connections

- Resources do contain virtual components i.e., placeholders that will be customized in the implementation phase to meet constraints
- Very important resources are interconnections and communication protocols

© Alberto Sangiovanni-Vincentelli. All rights reserved.
Learning from logic synthesis

High level function model

- Separation of func and arch
- Common language for func and arch netlists (Boolean logic, NAND2 gate)
- Automatic mapping

Gate library (platform)

Function model in netlist

Technology Mapping (covering)

Mapped design

Gate library in netlist

- Restructuring
Proposed software synthesis flow

Stage 1: Common modeling domain (CMD) selection
- Common semantics for func and arch
- Primitives to decide abstraction level

Stage 2: Automatic mapping

Stage 3: Code generation
Challenges in the flow

• Stage 1: Common modeling domain selection
  ▪ Various models of computation exist in system level.
  ▪ Trade-off between expressiveness and ease of manipulation when selecting the common semantics.
  ▪ Trade-off between granularity and optimality when selecting the primitives.

• Stage 2: Automatic mapping
  ▪ Various constraints and objectives.
  ▪ Domain-specific algorithms may be used albeit not necessary.

• Stage 3: Code generation
  ▪ Communication interface synthesis maybe needed to guarantee correct semantics.
Choosing the common MoC

- Tradeoff between expressiveness and strength of analysis
- Undecidable dataflow models easier to specify, impossible to analyze
- Restrictive models may not capture system details
- Finding a “sweet spot” may be domain or system dependent
Covering problem after CMD selection

- Symbols:
  - Function primitive instances:
  - Architecture primitive instances:
  - Mapping decision variables:
  - Architecture selection variables:
  - Quantities (power, area, bandwidth...):

- General covering formulation:

Function covering constraints

\[ \forall f_i \in \mathcal{F}, \sum_{a_j \in A_{f_i}} d_{f_i,a_j} = 1 \]

Architecture selection constraints

\[ \forall f_i \in \mathcal{F}, a_j \notin A_{f_i}, d_{f_i,a_j} = 0 \]
\[ \forall a_j \in A, \sum_{f_i \in \mathcal{F}} d_{f_i,a_j} \geq s_{a_j} \]
\[ \forall f_i \in \mathcal{F}, a_j \in A, d_{f_i,a_j} \leq s_{a_j} \]

Domain specific. Determines complexity!

Objective functions

\[ H_{t,l}(\{d_{f_i,a_j}\}, \{Q_{f_i,a_j,t}\}, \{s_{a_j}\}, \{Q_{a_j,t}\}) \leq 0 \]
\[ \min G_t(\{d_{f_i,a_j}\}, \{Q_{f_i,a_j,t}\}, \{s_{a_j}\}, \{Q_{a_j,t}\}) \]
Stage 1: CMD selection – common semantics

1. Process Networks (PN): expressive but high modeling complexity. Need transformation of both func and arch models.

2. Loosely time triggered architecture (LTTA): transformation of func model to support asynchronous communication.
   - Chosen in this case study

3. Synchronous reactive (SR): transformation of the arch to support synchronous communication, by applying following protocols.
   - Clock synchronization.
   - Constraints on task periods.

D = C_{PN} (P_D)

F = C_{SR} (P_F)

C_1 = C_{LTTA} (P_1=P_F \cup P_A)

C_2 = C_{SR} (P_2=P_F \cup P_A)

A = C_{LTTA} (P_A)
Stage 2: covering problem

Functional Model

- IR Sensor
- Wheel Sensor
- Nav. Task
- Fusion Task
- Object ID Task
- Brake Act.

Signals 150 ms

Architectural Model

- ECU1
- BUS1
- ECU2
- BUS2
- ECU3
- ECU4

Primitives: tasks, signals

Primitives: ECUs, messages on buses

Quantity constraints and objective functions
- End-to-end latency
- Utilization
- Extensibility
- ......

Covering variables
- Task to ECU
- Signal to message
- Message selection
- Priority
- Period

Variety of algorithms
- mathematical programming
- heuristics
- meta-heuristics
- machine learning
- ......
Allocation & priority synthesis results

Function Model
- 41 Tasks
- 83 Signals
- 171 paths

After mapping
- Meet all requirements
- Total latency from 36486ms in manual design to 12900ms

Architecture platform
- 9 ECUs
- single bus
Case studies in other domains

• Building automation domain [1]
  ▪ Similar semantics as in automotive – synchronous function model and LTTA architecture platform.
  ▪ Also choose SR as the common semantics, however additional timing constraints are added to the architecture for preserving synchronism, as we consider the physical interaction with environment.
  ▪ Mapping leverages COSI for communication network synthesis.

• Multimedia domain [2]
  ▪ JPEG encoder application. Intel MXP architecture platform.
  ▪ Semantics for both function and architecture are dataflow.
  ▪ Challenge is to choose the proper abstraction level. Different levels are explored and compared through choices of primitives.

Concluding remarks

- **Software (and hardware) synthesis based on a formal mapping procedure**
  - Formally determines the semantics and abstraction level of the design by choosing a common modeling domain.
  - Automatic and optimal mapping algorithms.
  - **Generality** – applied to various domains with different models of computation as well as different implementation platforms. Domain-specific mapping algorithms may be leveraged in the framework.
  - **Optimality** – trade-off between complexity and mapping space through the selection of CMD.
  - **Reusability** – common semantic selection requires designers’ expertise. However proper selection is typically general for particular domains.