A General Synthesis Approach for Embedded Systems Design with Applications to Multi-media and Automotive Designs

Alberto Sangiovanni Vincentelli

(Qi Zhu and Abhijit Davare)
Outline

• Design Trends and Challenges
• Semantic-Driven Synthesis Flow
  ○ Common Modeling Domain
  ○ Automatic Synthesis
• Validating the Design Flow
  ○ Automotive Stability Control
  ○ Image Processing
• Future Work
Toyota Autonomous Vehicle Technology Roadmap

Source: Toyota Web site
Electronics, Controls & Software: Shifting the Basis of Competition in Vehicles

- More functions & features
- Less hardware
- Faster

Potential inflection point. Now!

Source: Matt Tsien, GM
A Typical Car Architecture (BMW)
Design Trends

• Time to market
• Reliability
• Flexibility

Complexity of embedded systems

➢ Design reuse
➢ Correct-by-construction

Separation of concerns

Functionality
MP3, MPEG-4, Wi-Fi, BT, GPS …

Architecture
Cell, MXP series, IXP series, Nexperia …
Platform: library of resources defining an abstraction layer

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

Diagram illustrating the mapping process between function spaces and platform design spaces.
The next level of Abstraction …

- **Transistor Model**
  - Capacity Load
  - 1970’s

- **Gate Level Model**
  - Capacity Load
  - 1980’s

- ** RTL**
  - 1990’s

- **SDF**
  - Wire Load

- **IP Block Performance**
  - Inter IP Communication Performance Models

- **IP Blocks**
  - RTL Clusters
  - SW Models

- **IP Blocks**

- **Year 2000 +**
Challenge

Functionality and Architecture are modeled separately

- Different semantics (e.g., synchronous vs. asynchronous)
- Different abstraction level (e.g., instruction vs. block level)

Mapping is ad-hoc, manual, error-prone

Find a common language and decide the abstraction level

Mapping is automatic, correct, reproducible
Outline

- Design Trends and Challenges
- Semantic-Driven Synthesis Flow
  - Common Modeling Domain
  - Automatic Synthesis
- Validating the Design Flow
  - Automotive Stability Control
  - Image Processing
- Future Work
Technology Mapping

Common language: Boolean Logic
Primitives: NAND2 gate and inverter

Function $F = t_1 t_3 + fgh$

Architecture
- $nand2(2)$
- $nor(2)$
- $nand3(3)$
- $oai21(3)$
- $oai22(4)$
- $xor(5)$
- $nor3(3)$

Mapped Design
- $inv(1)$
- $nand3(3)$
- $oai21(3)$
- $inv(1)$
- $nand2(2)$
- $nor(2)$
- $nand3(3)$
- $oai21(3)$
- $oai22(4)$
- $xor(5)$
- $nor3(3)$
Synthesis Flow

1. Stage 1
   - Function Model
   - Architecture Model

2. Stage 2
   - Function Model in CMD
   - Architecture Model in CMD
   - Common language
   - Primitives

3. Stage 3
   - Covering
   - Further Synthesis
Modeling Domain – Semantic Domain

**Modeling domain is based on semantic domain and primitives**

- Q: semantic domain - language
  (trace-based agent algebra [R. Passerone, PhD thesis])
  - Q.D: domain of agents – “building blocks”
  - Q.A: master alphabet – “all signals between blocks”
  - Q.a : Q.D -> 2^Q.A, each agent has an alphabet which is a subset of Q.A – “each block has a set of signals”
  - Operators: renaming, projection and parallel composition – “rules to initialize and compose agents”

\[
s_3 = proj(\{i_1, o_2\}) \\
(\text{rename}(r)(s_1) \parallel \text{rename}(r)(s_2))
\]

![Diagram showing the relationship between states and transitions in a semantic domain model.](Image)
New agents can be constructed by applying a finite number of operators in sequence on existing agents.

Agent Closure $C_Q(S)$ contains all the agents that can be constructed from a set of agents $S$ by applying operators in semantic domain $Q$.

Primitives $P$ – abstraction level
- Primitives are agents, $P \subseteq Q.D$.
- No agent in $P$ can be constructed from other agents in $P$, i.e., $\forall p \in P$, there exists no $P' \subseteq P \setminus \{p\}$ s.t. $p \in C_Q(P')$.
- $\{s_1, s_2\}$ is a set of primitives in $PN$ semantics, $\{s_1, s_2, s_3\}$ is not.
Modeling Domain

- Modeling domain $M = C_Q(P)$ is the agent closure of a set of primitives $P$ in semantic domain $Q$.
  - Contains all the agents that can be constructed from $P$.
    - E.g., $C_{PN}(\{s_1, s_2\})$ contains all the agents that can be constructed from $\{s_1, s_2\}$ by applying operators in $PN$ semantics.
  - Semantic domain $Q$ provides a language for modeling.
  - Primitive $P$ defines a sub-space of the entire modeling space of $Q$. Abstraction level can be explored by choosing different primitives.

- What is the relation between different modeling domains?
- What is common modeling domain?
Behaviors
- $\Gamma: 2^{Q.A} \rightarrow 2^T$, each alphabet is associated with a set of traces over trace domain $T$.
- Each agent $s$ has a set of traces $\Gamma(Q.\alpha(s))$, which are called behaviors, denoted as $B(s)$.

$$B(s_1) = \Gamma(Q.\alpha(s_1)) = (i_1 \rightarrow V^\infty) \times (o_1 \rightarrow V^\infty)$$

Ancestor-Child relation between modeling domains
- $\Phi(M) = \{B(s) \mid s \in C_Q(P)\}$.
- $M_1 = C_{Q_1}(P_1)$ is the ancestor of $M_2 = C_{Q_2}(P_2)$ if $\Phi(M_2) \subseteq \Phi(M_1)$, denoted as $M_2 \leq M_1$.
- Example: $C_{PN}({s_3}) \leq C_{PN}({s_1, s_2})$
- Ancestor – more expressive, higher modeling complexity
  Child – more specific, lower modeling complexity
A model is an agent in the modeling domain.

A modeling domain $M$ is called common modeling domain between function model $f$ and architecture model $a$ if there exists $f' \in M$ and $a' \in M$ such that $B(f') \subseteq B(f)$ and $B(a') \subseteq B(a)$. 

$$C \subseteq O$$
CMD Selection

Search for CMDs on
Modeling Domain Relation Graph

Common Ancestor Modeling Domain of F and A

Original Function Modeling Domain

Original Architecture Modeling Domain

Larger mapping space

Simpler model
CMD Selection

- Two design aspects when selecting CMD $C = C_Q(P)$
  - Semantics – decided by semantic domain $Q$
    - Expressiveness vs. Analyzability.
      - e.g. Dataflow vs. Static dataflow
    - First choose semantic domain for common ancestor domain $D$, then refine it in $C$.
  - Abstraction level – decided by primitives $P$
    - Explore different abstraction level by choosing different primitives.
    - Carried out when select $C$ as child domain of $D$.
    - Utilize ancestor-child relation to explore candidate child domains formally.

Trade-off: complexity and size of mapping space
Outline

- Design Trends and Challenges
- Semantic-Driven Synthesis Flow
  - Common Modeling Domain
  - Automatic Synthesis
- Validating the Design Flow
  - Automotive Stability Control
  - Image Processing
- Future Work
Covering Problem Formulation

- Symbols:
  - Function primitive instances: \( F = (f_1, f_2, ..., f_n) \)
  - Architecture primitive instances: \( A = (a_1, a_2, ..., a_m) \)
  - Mapping decision variables: \( d_{ij} \)
  - Quantities (power, area, bandwidth…): \( Q^l_{ijk} \)
  - Costs (special quantities): \( C^l_{ijk} \)

- Constraints:
  - Decision constraints: \( \sum_{j \in S_i} d_{ij} = 1 \quad \forall i, \ 1 \leq i \leq n \)
    - Each function primitive instance needs to be covered by one and only one architecture primitive instance.
  - Quantity constraints: \( H^l_t(d_{ij}, Q^l_{ijk}) \leq Q^l C^l_t \)
    - Constraints from architecture platform or design constraints, such as power constraints, bandwidth constraints, etc.

- Objective function:
  - Cost function: \( G^l_t(d_{ij}, C^l_{ijk}) \)
Solve Covering Problem

General purpose solvers
- MILP: CPLEX, MOSEK, GLPK, …
- GP: GGPLAB, YALMIP, …

Other design concerns
- Scheduling
- Buffer sizing
- Comm. bandwidth
- …

Covering Problem

\[
\sum_{j \in S_i} d_{ij} = 1
\]

\[
H^l_t (d_{ij}, Q^l_{ijk}) \leq QC^l_t
\]

\[
G_l (d_{ij}, C^l_{ijk})
\]

Customizable
Branch-based
framework

Domain-specific Algorithms
Outline

- Design Trends and Challenges
- Semantic-Driven Synthesis Flow
  - Common Modeling Domain
  - Automatic Synthesis
- Validating the Design Flow
  - Automotive Stability Control
  - Image Processing
- Future Work
Find common language between functionality and architecture by choosing semantic domain $Q$ in CMD $C_Q(P)$.

**Functionality**
- synchronous Simulink model
- no message loss or duplication

**Architecture**
- clock drift between distributed ECUs, asynchronous comm.
- data loss and duplication
CMD Selection (Stage 1)

- Find CMD from modeling domain relation graph

\[ D = C_{PN}(P_D) \]

Common ancestor domain: 
**Process Networks** semantics

\[ F = C_{SR}(P_F) \]

Original Function modeling domain: 
**Synchronous/Reactive** semantics

\[ C_2 = C_{SR}(P_2) \]

\[ C_1 = C_{LTTA}(P_1) \]

\[ P_1 = P_{F'} \cup P_A \]

\[ P_2 = P_F \cup P_A' \]

\[ A = C_{LTTA}(P_A) \]

Original Architecture modeling domain: 
Semantics of **LTTA** 
*(loosely time-triggered architecture)*

Transformation of architecture model

- CMD uses synchronous/reactive semantics
  - Function model does not need to be changed
  - Architecture primitives $P_A$ should be transformed to $P_{A'}$ in CMD to support synchronous/reactive semantics
    - Protocol to avoid data loss [1]
      - Constraints on process periods
      - Requires clock drift to be within a certain range
    - Clock synchronization to restrict the clock drift [2]
    - Alternating bit to avoid data duplication
    - $B(a') \subseteq B(a)$, but correct behaviors can be assured when function model $f'$ is mapped to architecture model $a'$


Automatic Synthesis (Stage 2&3)

- Covering problem
  - Function primitives instance $f_i$: tasks and messages
  - Architecture primitives instance $a_j$: ECUs and buses
  - Quantity constraints: maximum workload on ECU
  - Objective function: minimize inter-ECU communication and balance computation load across ECUs
  - Utilize the Scotch [1] package
    - Function model graph and architecture graph
    - Partition-based algorithm solves the covering problem

- Further synthesis
  - Task priorities – based on pre-assigned message priorities
  - Task periods
    - Adjusted to satisfy end-to-end latency requirements
    - Task periods need to satisfy the protocol to ensure the correctness of semantics

Experimental Results

- 14 tasks, 48 messages
- 6 ECUs, 1 bus
- Automatic synthesized design vs. 6 manual designs
- Simulated in Metropolis framework

<table>
<thead>
<tr>
<th>Design</th>
<th>Total delay (ms)</th>
<th>Bus utilization</th>
<th>Max ECU utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual 1</td>
<td>787.10</td>
<td>57%</td>
<td>60%</td>
</tr>
<tr>
<td>Manual 2</td>
<td>761.84</td>
<td>50%</td>
<td>72%</td>
</tr>
<tr>
<td>Automatic</td>
<td>559.51</td>
<td>34%</td>
<td>48%</td>
</tr>
</tbody>
</table>
Explore abstraction level of the design by choosing primitives $P$ in $CMD\ CQ(P)$.

**Functionality**

- Data-driven application
- Can be used in MPEG-2, Motion-JPEG, MPEG-4, etc.
- Highly parallel heterogeneous platform
- Designed for image application
- 8 image signal processors (ISPs)
- 5 processing elements (PEs) in each ISP
**CMD Selection (Stage 1)**

- Dataflow semantics
- Choose proper abstraction level (granularity)
  - Too coarse – inefficient usage of highly parallel platform
  - Too fine – scheduling difficulties and model complexity
- Explored four CMDs
  - Block level
  - Coarse sub-block level
  - Fine sub-block level
  - Instruction level
Function Model at Different CMDs

Block level CMD

CSC → Shift → DCT → Quantization → Huffman

Coarse sub-block level CMD

1D-DCT → Transpose → 1D-DCT → Transpose → ZigZag → Multi

Fine sub-block level CMD

Add4 → Add2 → Multi1
Sub4 → Sub2 → Multi2
Architecture Model

- Architecture model has the same abstraction level in all CMDs for this case study.
  - PEs, global registers, memories are architecture primitives in all CMDs.
  - Abstraction levels of mapped designs in this case study are dictated by the functional abstraction levels.

- No single PE can support DCT block so block level CMD is not suitable.

- Complexity is too high so instruction level CMD is not suitable.

*Important to find a CMD at proper abstraction level*
Automatic Synthesis (Stage 2&3)

- Mapping at coarse and fine sub-block level CMDs.

Choosing a CMD at proper abstraction level can greatly affect the performance of the mapping.

---

Outline

- Design Trends and Challenges
- Semantic-Driven Synthesis Flow
  - Common Modeling Domain
  - Automatic Synthesis
- Validating the Design Flow
  - Automotive Stability Control
  - Image Processing

- Future Work
Future Work

● Ongoing projects:
  ○ Task allocation and priority assignment for automotive systems – stage 2
  ○ Period synthesis for automotive systems – stage 3

● Future work:
  ○ Complete synthesis flow for automotive domain
  ○ Multimedia domain case study – H.264
  ○ Synthesis from synchronous models to general multi-processor platform
Contributions

- A synthesis flow in which semantics and abstraction level are formally determined, and automatic algorithms can be applied.

- The method was validated by industrial case studies with different focus.
  - Automotive stability control: choose semantics
  - Image processing: explore abstraction level
  - Regardless of the focus, applications from different domains can be solved in the same way by our flow.