The Gap between Specification and Synthesis

Stephen J Mellor
Too Much Flexibility

The gap between specification and synthesis is large.
Lots of Options to Explore

Feed the software guys enough caffeine and you'll need at least two cores...
Explore Design Space

Let them go snowboarding, and the logic designers have to do all the work...

...but at least you can use a cheap core.
## Process
What’s required before we can start implementation?

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Algorithm</td>
</tr>
<tr>
<td>Micro-architecture</td>
<td>Software architecture</td>
</tr>
<tr>
<td>Implementation: C, C++, HDL, RTL</td>
<td>Implementation: C, C++, Java, Plex</td>
</tr>
</tbody>
</table>

- Huge investment required before implementation begins
- Testing is delayed until something is running
- Design inertia resists architectural changes
An Idea

• Since both teams are doing similar things:
  – Creating abstractions,
  – Formalizing them, and then
  – Verifying behavior

• ...it seems reasonable to use the same approach
  for specifying and verifying behavior.
Towards a Solution

Jointly:

■ Build a single Application Model
■ Build an Executable Application Model
■ Don’t Model Implementation Structure
■ Map the Application Model to Implementation
Abstraction in Hardware

- Gate density is increasing exponentially
- Complexity is increasing along with gate density
- We need a way to manage this complexity
- We need to move to a higher level of abstraction

The answer is C!
Abstraction in Software

Price Performance

Assembly
Assembler
Machine Code
1980s

High Level Language
Compiler
Assembly
1990s

Model Compiler
UML Models
High Level Language
2000s

System Complexity

Increased Productivity
UML – A Big Language

UML is the industry standard

- It has notations for everything you could possibly do in software
- Can we add notations for everything we can possibly do in hardware?

Executable UML is a defined:

- Streamlined
- Tractable
- Subset of UML

Achieved by having defined execution rules
Building Models

- Models capture the behavior of the entire system
- Including an Object Action Language (OAL)
Intelligent model capture verifies models:
- Syntactically
- Semantically

For an executable model *with actions*
Translation Rules

Translation rules:

- Read the database to produce text
- Text can be a language for software or hardware
- Build a complete system from models consistently
Translation Rules Generate Text

Metamodel

Repository

Entity Shutter_VHDL { ...
Class Exposure_C { ...

Class
Number {I}
Name
KeyLetters
Description

State
Name {I}
Class {R7}
isFinal

State
Name {I}
Class {R7}
isFinal

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Number</th>
<th>isFinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
<td>101</td>
<td>973</td>
<td>No</td>
</tr>
<tr>
<td>Checking</td>
<td>101</td>
<td>974</td>
<td>No</td>
</tr>
<tr>
<td>Open</td>
<td>101</td>
<td>975</td>
<td>No</td>
</tr>
</tbody>
</table>

Class
Number {I}
Name
KeyLetters
Description

Application

Translation Rules

Shutter (S)
Shutter ID {I}
Aperture
Zoom
OpenTime
Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Number</th>
<th>isFinal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
<td>973</td>
<td>No</td>
</tr>
<tr>
<td>Checking</td>
<td>974</td>
<td>No</td>
</tr>
<tr>
<td>Open</td>
<td>975</td>
<td>No</td>
</tr>
</tbody>
</table>
Two Translation Rule Sets

.select many states related to instances of class->State where (isFinal == False)

public:

enum states_e
{
  NO_STATE = 0,
  CAMERA_CHECKING
}

.TYPET t_CameraState IS (  
  CAMERA_OPEN,
  CAMERA_CHECKING,
  CAMERA_CLOSED  
);
Marks are extended properties of the metamodel that allow different rules to be applied.

```plaintext
.select many classes where markIsHW == TRUE;
// generate VHDL

.select many classes where markIsHW == FALSE;
// generate logic for a C++ class
```
Write rules to create instances in the repositories

.select many classes where markIsHW == TRUE;
// populate the VHDL repository

.select many classes where markIsHW == FALSE;
// populate the C++ repository

Write clean rules from each repository
Interfaces...

- Tight coupling
- Easy to misunderstand
- Distributed throughout implementation

Memory-mapped I/O
Interfaces...

Changes ripple through both:
- Software and
- Hardware

Manual maintenance is
- Expensive
- Error-prone

Register I/O
Model it!

Model the Application
Mark it!

Assign classes to hardware or software
Make it!

Software
name

Hardware
name

Software
name

Hardware
name

Software
name
Profile it!

- Run Profiler
- Analyze the performance
- Not as good as you expected?
Re-Mark it!

Reassign Classes to Different Tasks
Make it Again!
xtUML is a streamlined subset of the UML industry standard that:

(X) Executes models
- Allows for early verification
- Pre-code interpretive execution
- Integration of legacy code

(T) Translates models
- Complete code generation from models
- Customizable compilation rules
- Optimized code
Questions and Discussion
Create Reusable IP Assets

• Application Model
  – Completely captures application subject-matter expertise
  – Executable and therefore can be verified
  – Implementation-independent

• Model Compiler
  – Completely captures design and implementation expertise
  – Executable and therefore can be verified
  – Application-independent
Modeling, what does it buy us?

• Abstraction => Increased Productivity
• Communication => Fewer Defects
• Early Verification => Fewer Defects
• Correct Interfaces => Fewer Defects
• Delay Technology Decisions => Lower Cost
• Skill Specialization => Increased Productivity
• Reusable IP => Increased Productivity, Lower Costs
Late-night Blues to Blue-sky

• Exploring a big design space for SoCs
  – Analyze the application models:
    • Statically
    • Dynamically
  – Determine:
    • Degree of concurrency
    • Throughput and response requirements
  – Define, allocate, and generate:
    • Custom cores and logic
    • Optimized software for the cores
Verification is Over 60% of the IC/ASIC Design Effort...

Effort Allocation of IC/ASIC Design Engineers by type of Activity

Source: 2002 IC/ASIC Functional Verification Study, Collett International Research
...Yet Flaws Still Make It To Silicon

Source: 2002 IC/ASIC Functional Verification Study, Collett International Research
...Putting Half of Embedded Designs Behind Schedule

3.9 Months Behind Schedule (average)

Behind Schedule: 54%
Canceled: 13%
Outsourced: 11%
Ahead Schedule: 1%

Source: Electronic Market Forecasters, June 2003
...With Half Missing Performance and Functionality Goals

Design Results vs. Expectations

Source: Electronic Market Forecasters, June 2003
A model compiler

- is a collection of *translation rules*
- uses *marks* to decide which rule to apply to which model element
xtUML Model Translation

A model compiler
Consistently builds complete system from models
Translates into best language for system
Uses rules and marks for optimization of resulting code

This approach…. leverages expertise of best architects, captures that expertise.