A UML-Based Approach for Heterogeneous IP Integration

Zhenxin SUN

Supervised by Prof. Wong Weng-Fai
Outline

➢ Motivation
➢ Model design using UML
➢ Wrapper Synthesis
➢ Case studies
➢ Experiment results
➢ Conclusion and future works
Background

- **IP (Intellectual Properties)**
  - Virtual Components with well predefined functionalities, usage methods and tools to support the usage

- **IP Reuse**
  - 2-3x benefit
  - Design for reuse is expensive

*Source of IP: Internal teams and third party providers*
Challenges

- **Incompatible bus protocols**
  - Major bus protocols are not compatible to each other
  - Third party IP make the things works
- **High Cost of design exploration**
  - Design partitioning/displacement
    - Cumbersome
    - Error-prone
  - Bus re-configuration is time-consuming
  - Need to be done iteratively
Challenges

- **Increasingly complexity**
  - Hundreds of IP in a design
  - Different IP sources

- **Informal IP and bus definition**
  - Definitions are commonly Natural Languages & waveforms
  - Inconsistencies
Problem Description

➢ A failure to perform the connection may be due to one of two possible reasons:
  – the interface protocol does not match
  – the SLports of one interface are not sufficient to drive the other.

➢ We are going to solve two types of compatible pairs:
  – Type 1 compatible pair: There is a one-to-one correspondence between the SLports of the two interfaces.
  – Type 2 compatible pair: The output SLport of one interface can be made to drive the input SLport of the other through some runtime transformation of its data.
Proposals

- Formalizing the interfaces using UML models
- Capture wrapper configuration that are used to customize IP cores
- Auto-generator to produce the merging glue of IP to the On-Chip-Buses
- Plug-and-play to enable fast integration and testing
- Using simulation for verification
A UML-Based Approach for Heterogeneous IP Integration

Manual effort

Automate processes

Design Ideas

Interface behaviors

Displacement plans

Object diagrams

State Diagram

Structural Diagram

Wrapper Generator

Interface adapters

Glue Code

FPGA

Simulation
Features

➢ To ensure correctness and reusability, we use UML structural and state diagrams to specify and formalize system interfaces. This single model is used consistently throughout the entire design process. It not only gives a system level view of the design but also allows for reuse in future designs.

➢ Automation is applied in every level of abstractions, and between different environments. Code is generated from the same source model, minimizing ambiguity.

➢ Our framework supports both interface protocol customization and glue logic generation, thereby maximizing IP integration.

➢ All changes are applied at the higher level, and user will only need to deal with the high level design decisions.
A UML-Based Approach for Heterogeneous IP Integration

Terminology

- Interface
- SLport
- UMLport
- Wrapper Port
User Input

➢ **UML Structural diagrams**
  – Describe the component structure of a system
  – Each component is treated as a black-box and modeled as a class

➢ **UML behavioral diagrams**
  – Describe the interaction between the components
  – Behaviors are modeled as states transitions associated with the classes
User Input

- Step 1: Formalize the IP interface using UML notations
- Step 2: Define the wrapper classes
- Step 3: Define behaviors of incompatible interfaces
Interface Synthesis

➢ **Rhapsody 6 as UML drawing tools**
  - Sbs file

➢ **Analyzer**
  - Analyze and construct the wrapper models

➢ **Velocity engine**
  - Merge with the template
Interface Synthesis

- Communication group
  - One master_wrapper and one slave_wrapper
  - A thread is used to maintain the connection
Interface Synthesis

```java
while(true){
    switch (state) {
        case IDLE: //in idle state
            wait_until(Fast_IDCT_signal.read()=true);
            //state is guarded by Fast_IDCT_signal
            start.write(true); //drive driver port
            state=INPUT_DATA; //change state
        case INPUT_DATA:
            addr=Fast_IDCT_addr.read();
            for i from 1 to 8, j from 1 to 8
                din.write(addr[j*8+i];
            state=WAIT;
        case WAIT:
            wait_until(done.read()=true);
            state=SEND_RESULTS;
        case SEND_RESULTS:
            for i from 1 to 8, j from 1 to 8
                addr[j*8+i]=dout;
            Fast_IDCT_ack.write(true);
            state=IDLE;
    }
}
```
## Mapping Rules between UML notations and design properties

<table>
<thead>
<tr>
<th>UML Notations</th>
<th>System Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Class</td>
<td>Wrapper module</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>UMLPort</td>
<td>Interface adapter</td>
</tr>
<tr>
<td>Subclasses</td>
<td>IP cores</td>
</tr>
<tr>
<td>UMLPort of subclass</td>
<td>Interface</td>
</tr>
<tr>
<td>Port name</td>
<td>Name</td>
</tr>
<tr>
<td>port type</td>
<td>Type</td>
</tr>
<tr>
<td>Interface type</td>
<td>Direction</td>
</tr>
<tr>
<td>Stereotype</td>
<td>Protocol</td>
</tr>
<tr>
<td>Port properties</td>
<td>Driving signals</td>
</tr>
<tr>
<td>State chart</td>
<td>Driving behaviors</td>
</tr>
<tr>
<td>Contract Attributes</td>
<td>Signals</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Stereotype</td>
<td>type</td>
</tr>
<tr>
<td>Tag</td>
<td>Width</td>
</tr>
<tr>
<td>UMLport State Chart</td>
<td>Adapter’s control code</td>
</tr>
<tr>
<td>States and transitions</td>
<td>Finite state machine</td>
</tr>
</tbody>
</table>
Case Studies: Simple Bus

- Taking from SystemC open source
  - 1 kernel, 1 arbiter, 2 memory slaves and 2 masters
  - Wrapped up the masters

- Experiment results
  - 2684 lines of original code
  - 3743 lines of wrapped code
  - 24.3% of overhead
Case Studies: Mpeg-2 Decoder

- Mpeg-2 Decoder
- IDCT versions
  - F-IDCT
  - R-IDCT
  - Verilog-IDCT
  - PCI-IDCT
A UML-Based Approach for Heterogeneous IP Integration
A UML-Based Approach for Heterogeneous IP Integration
Case Studies: Mpeg-2 Decoder

Table 1 Simulation result of decoders with F-IDCT

<table>
<thead>
<tr>
<th>Input</th>
<th>Unwrapped w/F-IDCT</th>
<th>Wrapped w/F-IDCT</th>
<th>Extra Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>short.m2v</td>
<td>0.412s</td>
<td>0.450s</td>
<td>9.22%</td>
</tr>
<tr>
<td>fball.m2v</td>
<td>154.886s</td>
<td>172.415s</td>
<td>11.32%</td>
</tr>
<tr>
<td>zoo.m2v</td>
<td>629.155s</td>
<td>730.5275s</td>
<td>16.11%</td>
</tr>
<tr>
<td>dhl.m2v</td>
<td>801.406s</td>
<td>932.31s</td>
<td>16.33%</td>
</tr>
</tbody>
</table>

Table 1 Simulation result of decoders with R-IDCT

<table>
<thead>
<tr>
<th>Input</th>
<th>Unwrapped w/R-IDCT</th>
<th>Wrapped w/R-IDCT</th>
<th>Extra Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>short.m2v</td>
<td>0.422s</td>
<td>0.444s</td>
<td>5.21%</td>
</tr>
<tr>
<td>fball.m2v</td>
<td>162.84s</td>
<td>184.004s</td>
<td>13.00%</td>
</tr>
<tr>
<td>zoo.m2v</td>
<td>699.8335s</td>
<td>789.0175s</td>
<td>12.74%</td>
</tr>
<tr>
<td>dhl.m2v</td>
<td>910.759s</td>
<td>1024.9585s</td>
<td>12.54%</td>
</tr>
</tbody>
</table>
A UML-Based Approach for Heterogeneous IP Integration

ASP-DAC 2009 Yokohama
Experiment Results

- Overhead is about 9%-16%
- Overhead of the wrapping is not proportional to number of wrappers
  - Proportional to number of transaction passing through the wrappers
  - Wrapping components with less workload has less impact on overall performance
  - Trade off between configurability and performance
Conclusions and Future works

➤ We have presented a framework for integration of heterogeneous and incompatible predefined IP-cores
➤ We have enabled a auto-generation of protocol adapters and glue logic from defined UML models
➤ Our algorithm is tested under several environments including SystemC, Verilog and FPGA modules.
➤ Future works
  – Cross platform design
  – Template generation
  – Design space exploration
A UML-Based Approach for Heterogeneous IP Integration