A UML-Based Approach for Heterogeneous IP Integration

Zhenxin SUN

Supervised by Prof. Wong Weng-Fai

NUS National University of Singapore
School of Computing

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

NUS National University of Singapore School of Computing

Challenges

- **Increasingly complexity**
  - –Hundreds of IP in a design
  - –Different IP sources
- **Informal IP and bus definition**
  - –Definitions are commonly Natural Languages & waveforms
  - –Inconsistencies

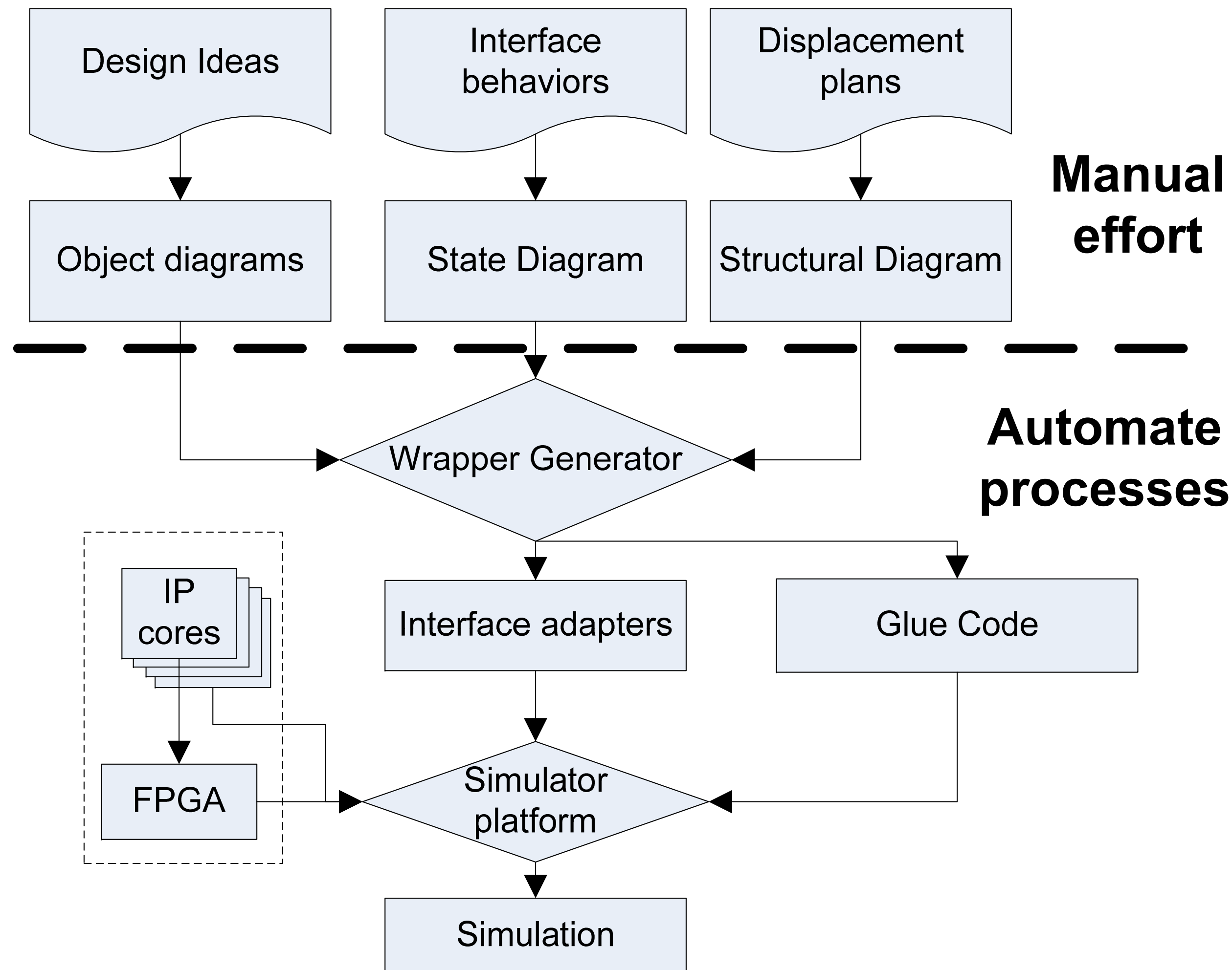
Problem Description

- **A failure to perform the connection may be due to one of two possible reasons:**
  - the interface protocol does not matched
  - 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.

NUS National University of Singapore
School of Computing

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**


Design Ideas
Interface behaviors
Displacement plans
Object diagrams
State Diagram
Structural Diagram
Manual effort
Wrapper Generator
Automate processes
IP cores
FPGA
Interface adapters
Glue Code
Simulator platform
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.**

Terminology

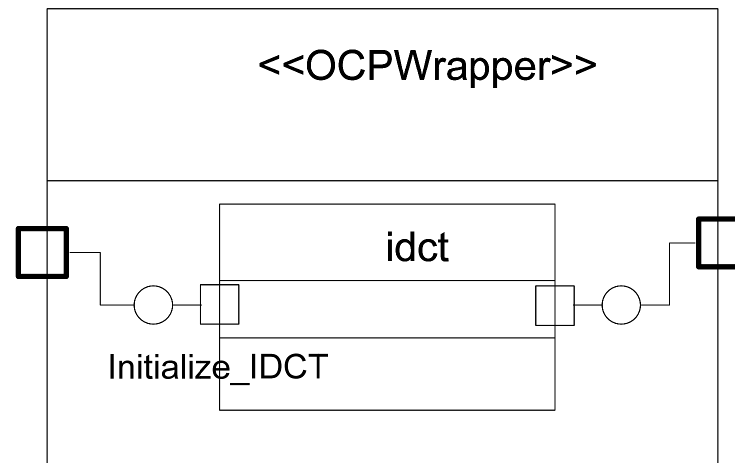
- **Interface**
- **SLport**
- **UMLport**
- **Wrapper Port**

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

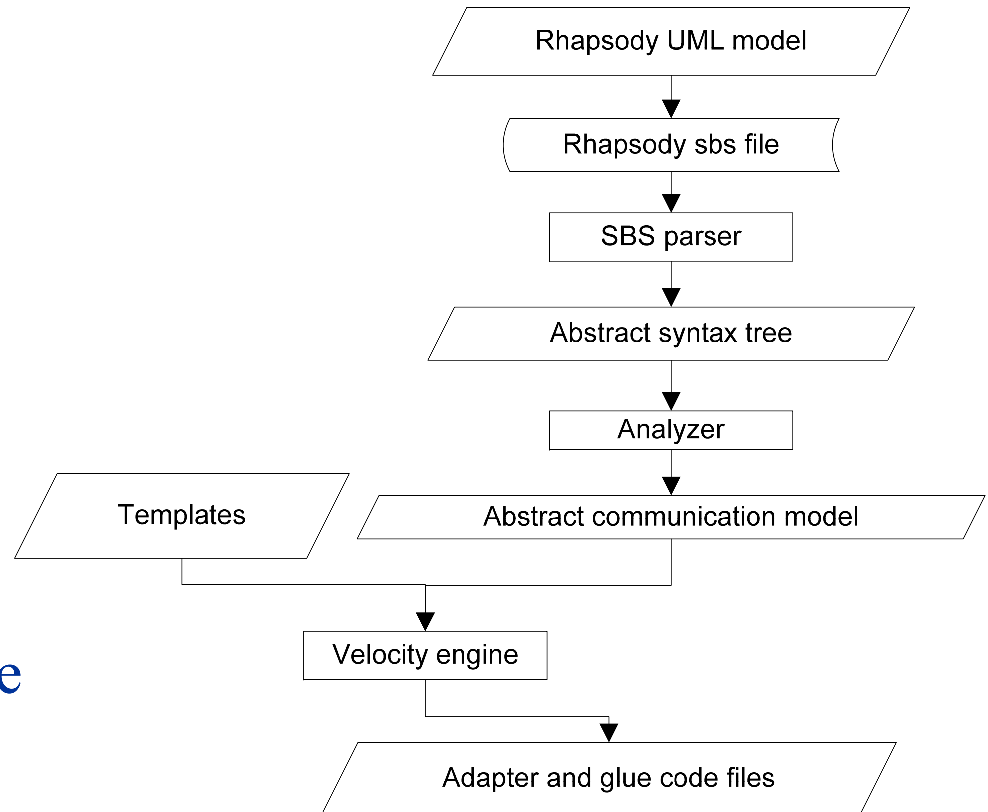
User Input

- Step1: Formalize the IP interface using UML notations
- Step2: Define the wrapper classes
- Step3: Define behaviors of incompatible interfaces



Interface Synthesis

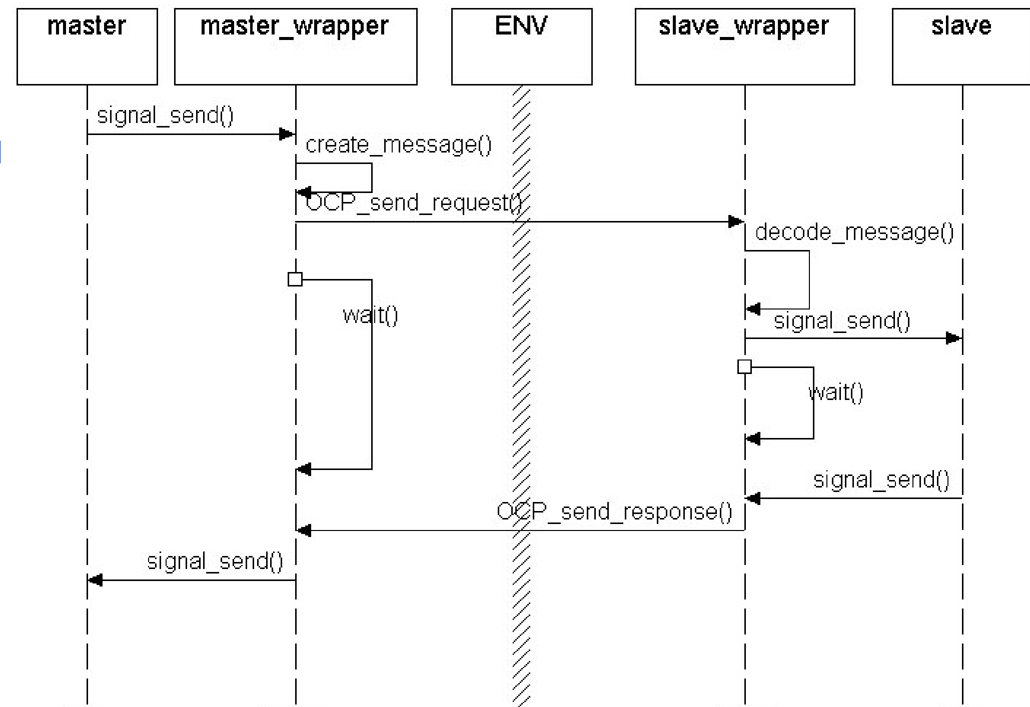
- **Rhapsody 6 as UML drawing tools**
  - Sbs file
- **Analyzer**
  - Analyze and construct the wrapper models
- **Velocity engine**
  - Merge with the template



NUS National University of Singapore
School of Computing

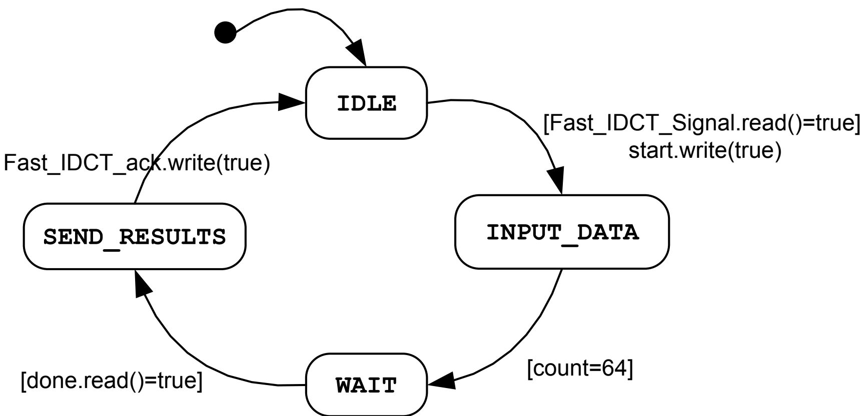
Interface Synthesis

- **Communication group**
  - One master_wrapper an
    one slave_wrapper
  - A thread is used to
    maintain the connection



NUS National University of Singapore
School of Computing

Interface Synthesis




```
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;
 }}
```

NUS National University of Singapore
School of Computing

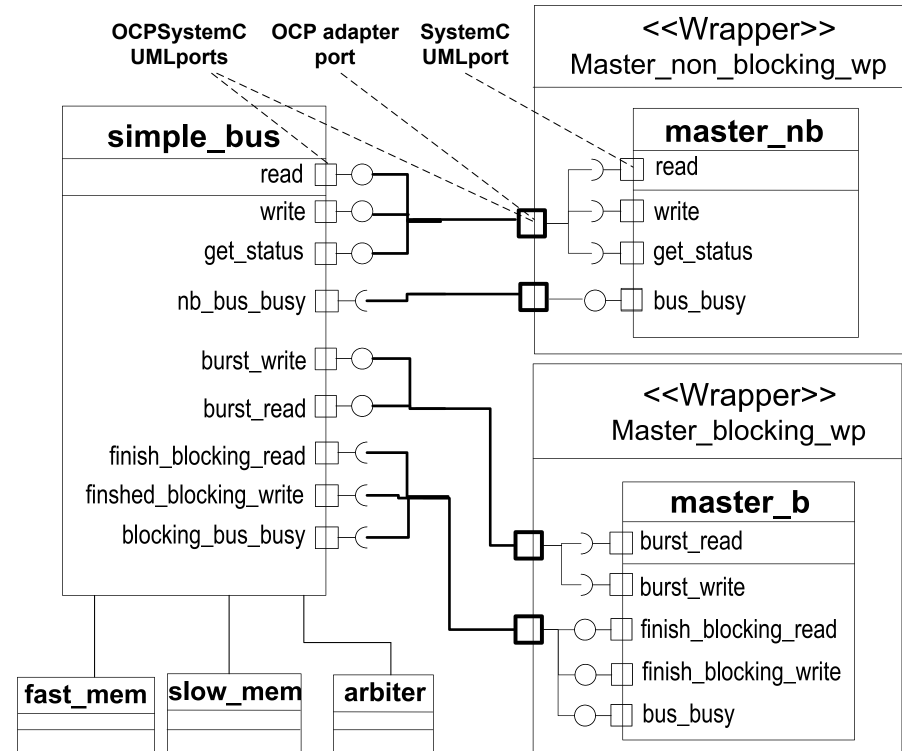
Mapping Rules between UML notations and design properties

UML Notations		System Properties	
Parent Class		Wrapper module	
	Name		Name
	UMLport		Interface adapter
	Subclasses		IP cores
UMLPort of subclass		Interface	
	Port name		Name
	port type		Type
	Interface type		Direction
	Stereotype		Protocol
	Port properties		Driving signals
	State chart		Driving behaviors
Contract Attributes		Signals	
	Name		Name
	Stereotype		type
	Tag		Width
UMLport State Chart		Adapter's control code	
	States and transitions		Finite state machine

NUS National University of Singapore
School of Computing

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

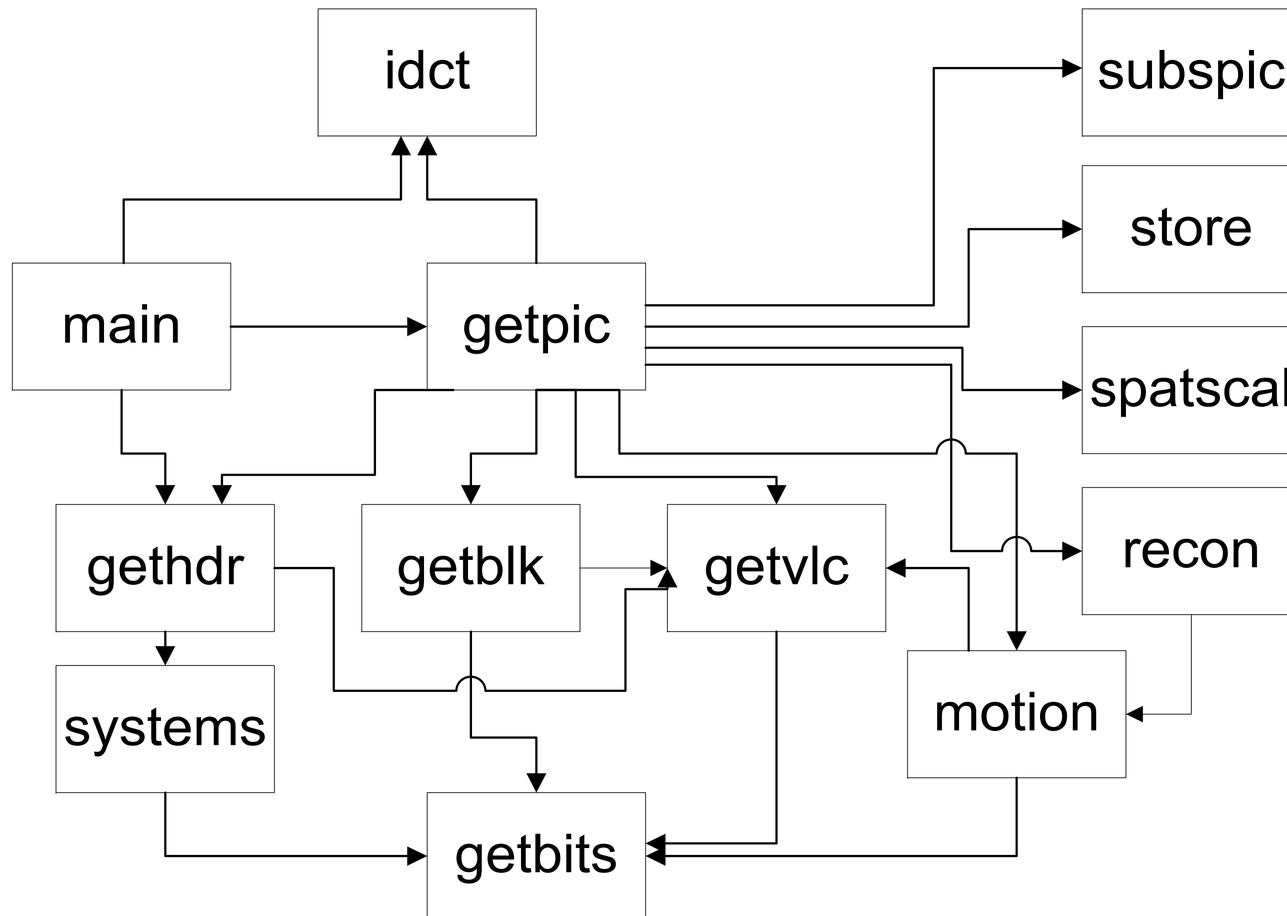


NUS National University of Singapore
School of Computing

Case Studies: Mpeg-2 Decoder

- **Mpeg-2 Decoder**
- **IDCT versions**
  - F-IDCT
  - R-IDCT
  - Verilog-IDCT
  - PCI-IDCT

NUS National University of Singapore
School of Computing



A UML-Based Approach for Heterogeneous IP Integration

NUS National University of Singapore
School of Computing

①

<<OCPWrapper>>
main_wp
mepg2dec
Get_Hdr
Init_IDCT
Show_Bits
next_start_code
Initialize_buffer
Decode_picture

<<OCPWrapper>>
idct_wp
idct
Initialize_IDCT
Fast_IDCT

<<OCPWrapper>>
utility_wp
store
Write_Frame
subspic
Substitute_Frame_buffer
spatscal
Spatial_prediction

<<OCPWrapper>>
processors_wp
getbits
Initialize_buffer
Show_Bits
gethdr
next_start_code
Get_Hdr
motion
recon
systems
getvlc
getblk

<<OCPWrapper>>
getpic_wp
getpic
Fast_IDCT
Decode_picture
Output_last_Frame_of_Sequence
Write_Frame
Substitute_Frame_buffer
Spatial_prediction

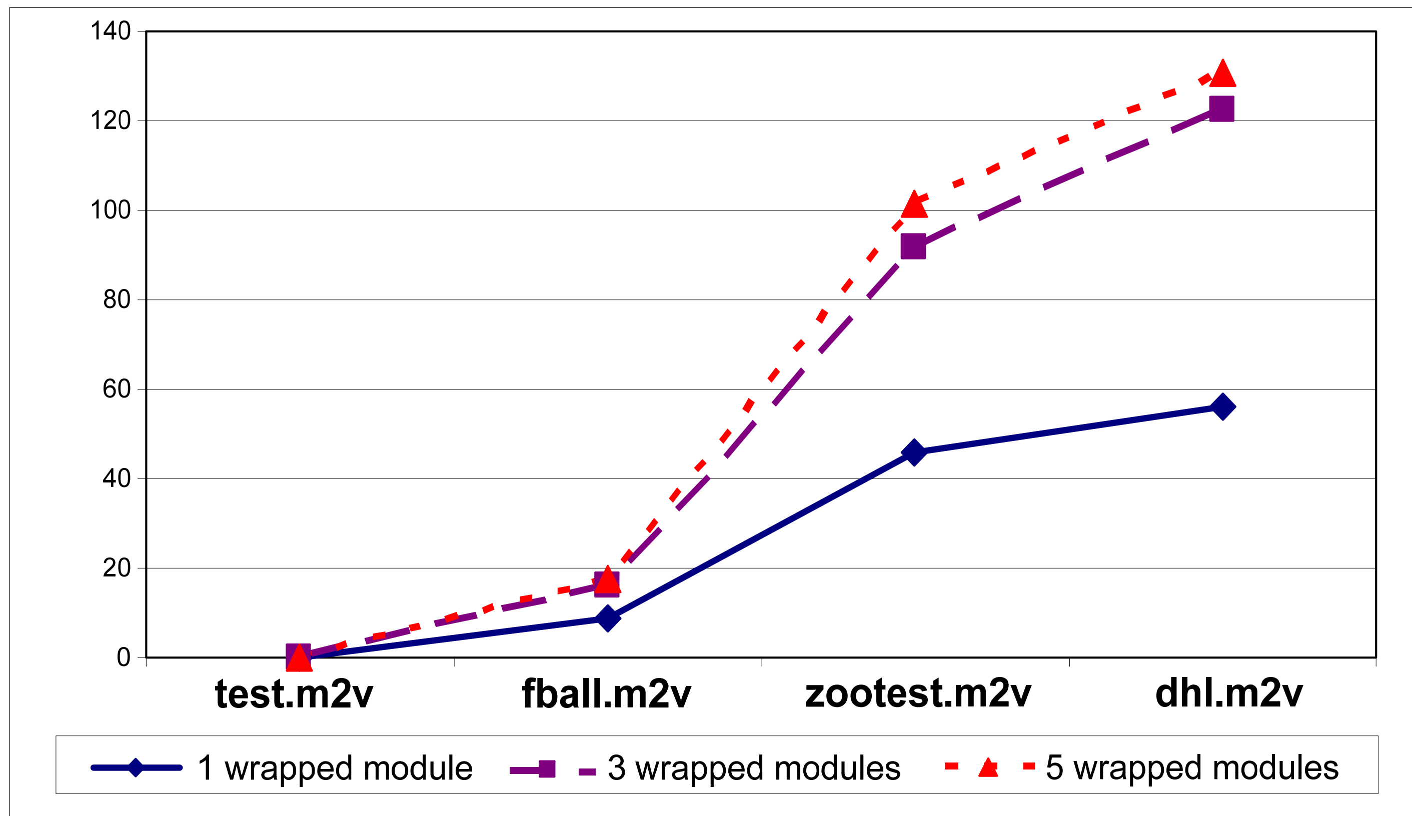
Case Studies: Mpeg-2 Decoder

Table 1 Simulation result of decoders with F-IDCT

Input	Unwrapped w/F-IDCT	Wrapped w/F-IDCT	Extra Overhead
short.m2v	0.412s	0.450s	9.22%
fball.m2v	154.886s	172.415s	11.32%
zoo.m2v	629.155s	730.5275s	16.11%
dhl.m2v	801.406s	932.31s	16.33%

Table 1 Simulation result of decoders with R-IDCT

Input	Unwrapped w/R-IDCT	Wrapped w/R-IDCT	Extra Overhead
short.m2v	0.422s	0.444s	5.21%
fball.m2v	162.84s	184.004s	13.00%
zoo.m2v	699.8335s	789.0175s	12.74%
dhl.m2v	910.759s	1024.9585s	12.54%



140
120
100
80
60
40
20
0
test.m2v
fball.m2v
zootest.m2v
dhl.m2v
1 wrapped module
3 wrapped modules
5 wrapped modules

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

NUS National University of Singapore
School of Computing