ASP-DAC 2006
26. Jan. 2006

Reusable Component IP Design using Refinement-based Design Environment

Sanggyu Park
Seoul National University

Introduction

- Standard Interface-based Hardware Components
  - Required for Reuse-centric Design Methodology
  - Easy hardware integration
  - Drawbacks
    - Too many architectural decisions in component
    - Complex design and verification
- A new component IP design methodology
  - Efficient component IP design & verification
  - More flexible communication
- Refinement-based design concept for component IP design

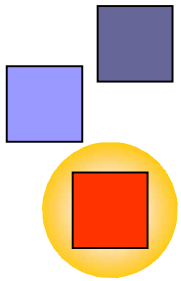
Outline

- Introduction
- Drawbacks of Standard-Interface-based Component IP
- CATtree-based Design Methodology
- Component IP Design Flow
- Case study: H.264 Decoder VLD Component Design
- Conclusion

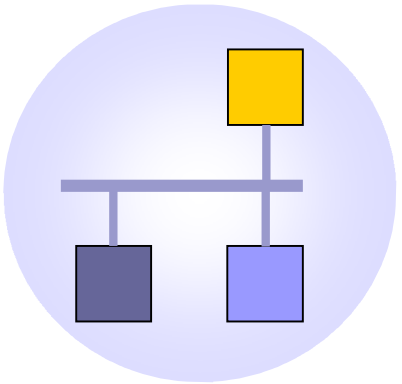
Reuse-Centric Design Approach

IP Provider

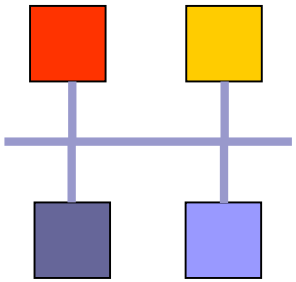
Platform Provider



Design a derivative system re-using pre-described IPs and Platforms



IP Providers, Platform Providers and System Designers are different people or groups

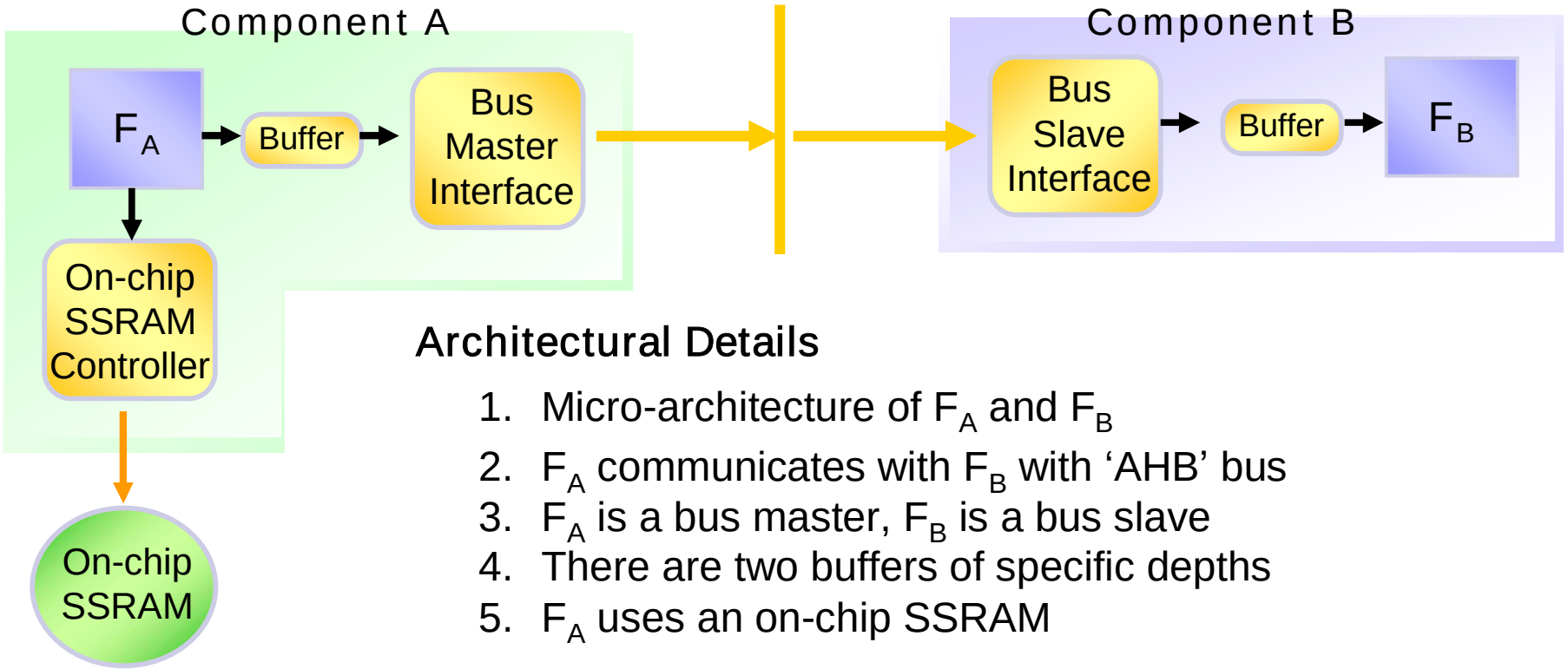


Interface standardization

System Design House

Drawback of Standard Interface [1]

- A standard interface-based HW component contains many communication architectural decisions that limits system level design space



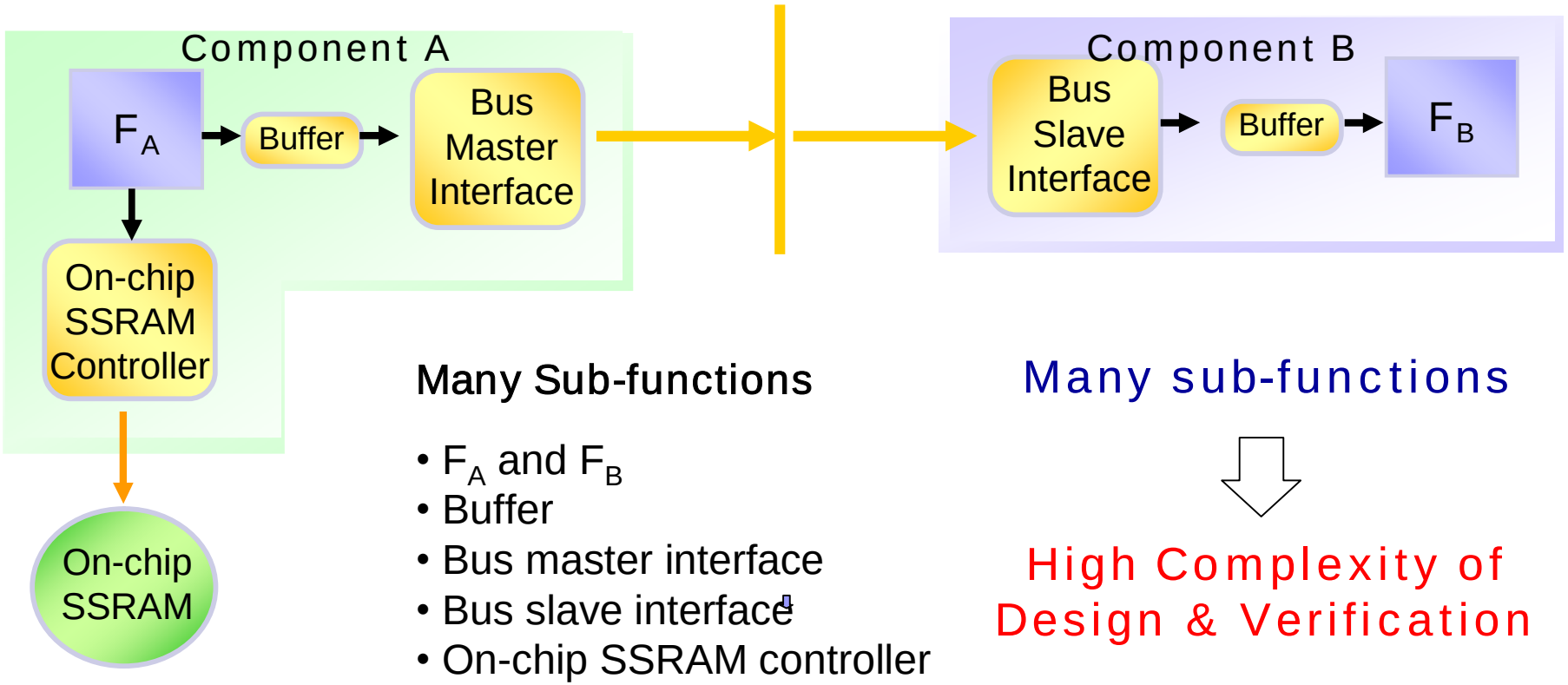
Architectural Details

1. Micro-architecture of F_A and F_B
2. F_A communicates with F_B with 'AHB' bus
3. F_A is a bus master, F_B is a bus slave
4. There are two buffers of specific depths
5. F_A uses an on-chip SSRAM

Limits reusability or system architecture

Drawback of Standard Interface [2]

- Standard interface-based HW has many sub-functions
  - A component contains many sub-functions that controls standard interfaces



Many Sub-functions

- F_A and F_B
- Buffer
- Bus master interface
- Bus slave interface
- On-chip SSRAM controller

Many sub-functions

⇩

High Complexity of Design & Verification

Drawback of Standard Interface [3]

- A standard interfaces such as on-chip buses, defines complex protocols to be a generic interface
  - AHB: Error, Split, Retry, Arbitration, etc
- A component should support all defined protocols
  - In a system, some protocols are not needed
    ➔ Unnecessary overhead and complexity
- Design and verification of fully compliant component

  ➔ very, very, very difficult

Problem Summary

Architectural Decisions

Computation

1. Micro-architecture of F_A, F_B

Communication

2. F_A communicates with F_B with 'AHB' bus
3. F_A is a bus master, F_B is a bus slave
4. Buffer depths
5. F_A uses an on-chip SSRAM

Design Complexity (# of sub-functions)

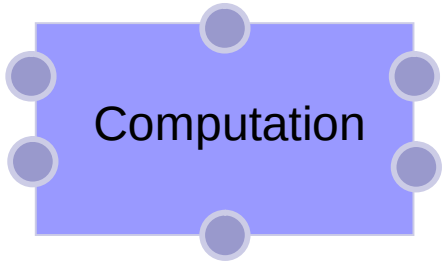
1. sub-components F_A, F_B
2. Buffer
3. Bus master interface
4. Bus slave interface
5. On-chip SSRAM controller

Verification Complexity

2. Bus master interface
3. Bus slave interface

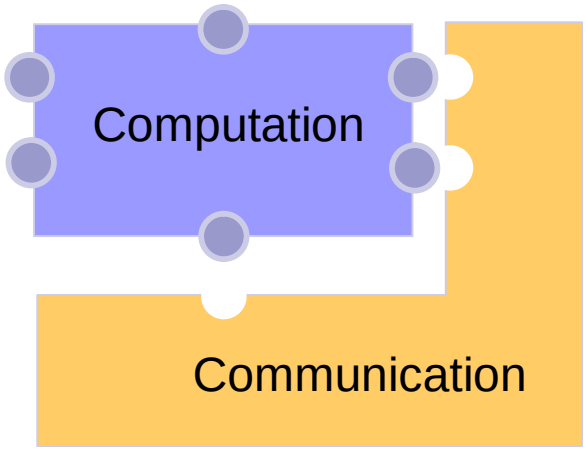
Proposed Solution

- Applying the design orthogonalisation concept to the component IP design flow



Capture a computation with abstract interfaces

Q: What are computations and what are not?



Refine communication architecture according to the system architecture

Q: What are communication architectures?

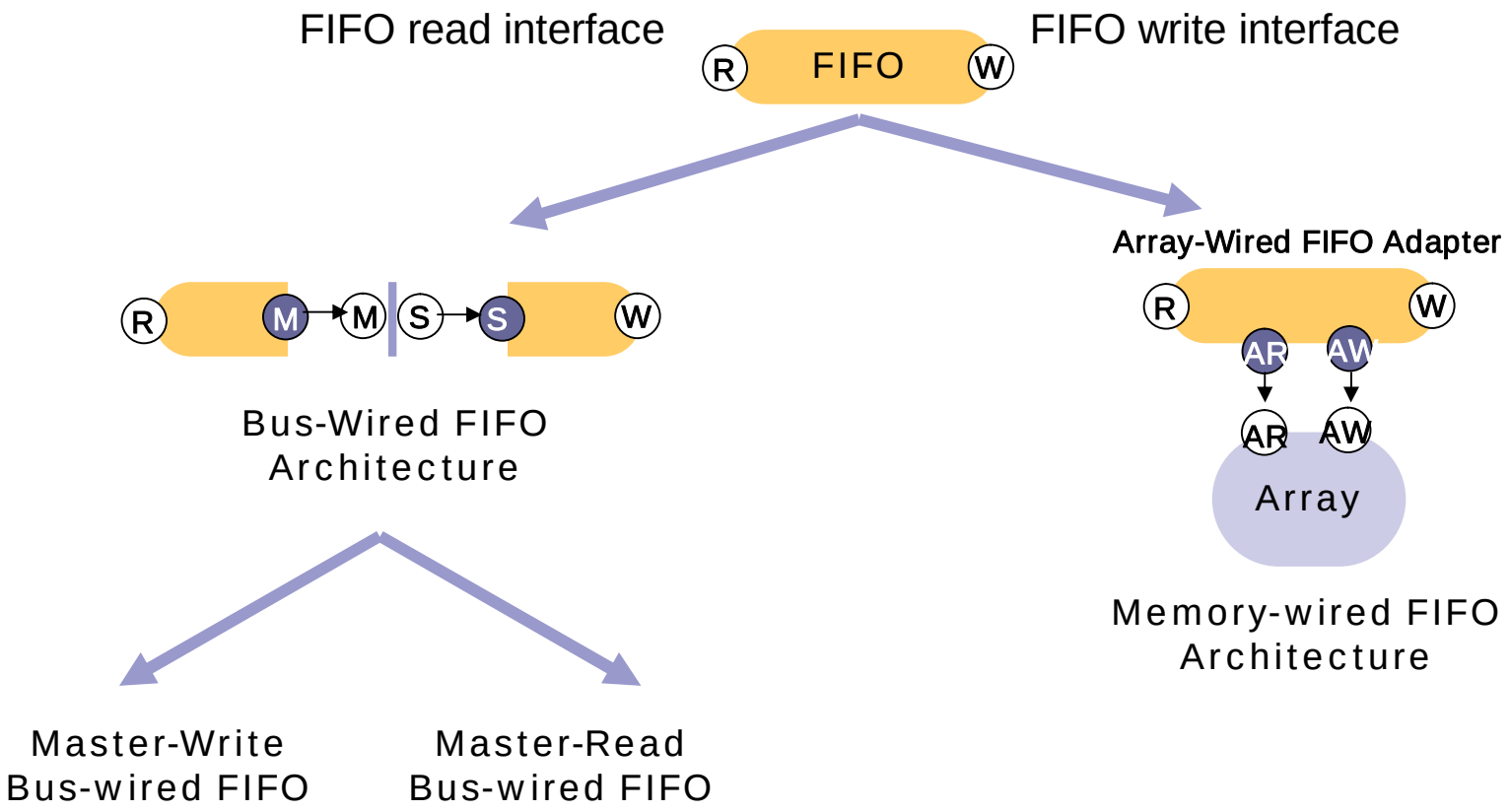
What are communications?

- What are communication functions ?
  - A data transfer? buses? FIFO ?
  - Or functions which are not computations?
- What are communication architectures?
  - A bus system? Switched network? NoC?
  - HW or SW implemented FIFO?
  - Memories?
- Clear guides to function capture and architecture refinement are needed

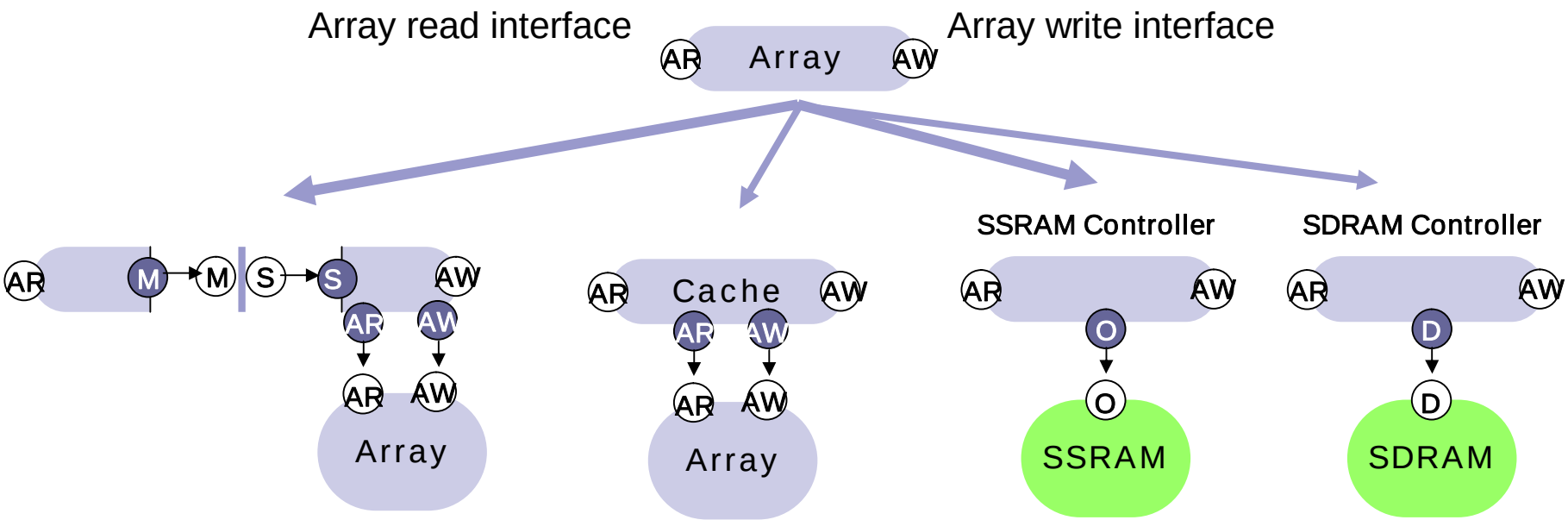
CATtree

- Communication Architecture Template Tree (CATtree)
  - A collection of information that covers a specific range of communication function and its architectures
  - Channel: a function model of a CATtree (in SystemC)
  - Interface: defines a set of functions the channel provides
    - TLM I/F in SystemC, RTL I/F in HDL, SW I/F in SystemC
  - Abstraction adapters
    - TLM-to-RTL and RTL-to-TLM abstraction adapters
  - Architecture Templates
    - parameterized implementation of architectures that are used for channel refinement
    - TLM templates in SystemC, RTL templates in HDL, SW templates in C

FIFO CATtree



Array CATtree



Bus-Wired Array Architecture

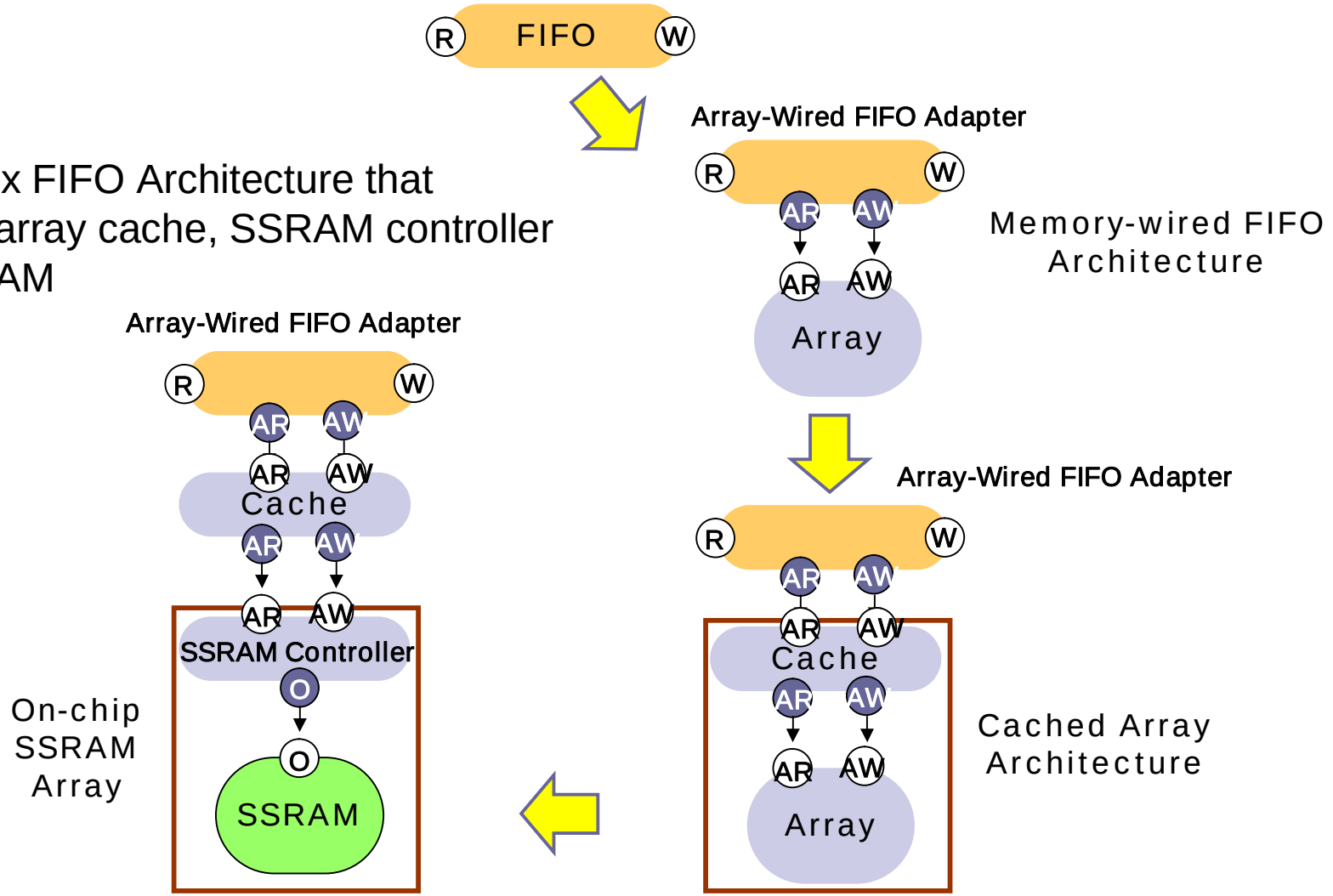
Cached Array Architecture

On-chip SSRAM Array Architecture

External SDRAM Array Architecture

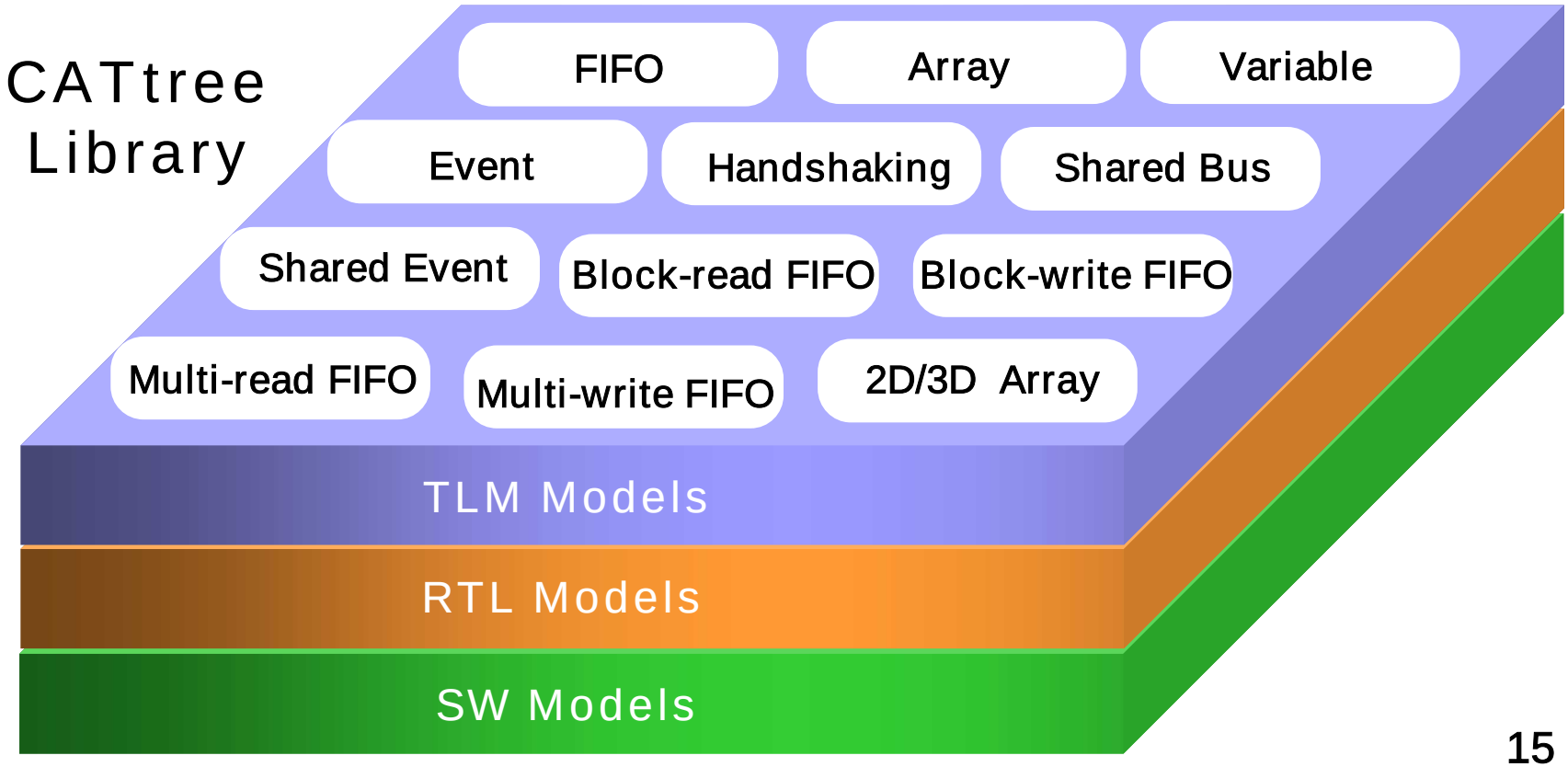
Complex Architecture Refinement

A complex FIFO Architecture that contains array cache, SSRAM controller and SSRAM

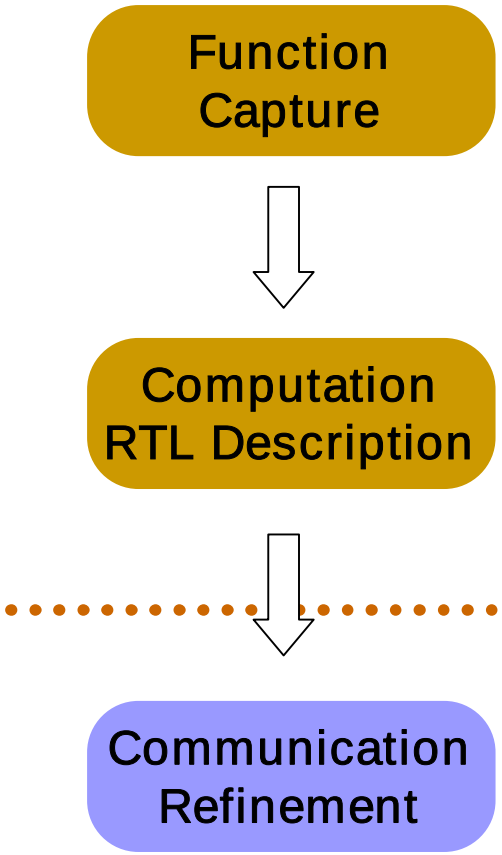


CATtree Library

- CATtree Library
  - A set of CATtrees
  - Clearly shows what are communication functions & architectures
  - Currently, we developed 14 CATtrees



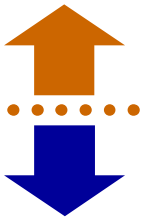
CATtree-based HW Component Design



- Describe computation TLM (SystemC)
- Describe testbench computation TLM (SystemC)
- Integrate computations with CATtree channels
- Verify & Validate the computation function

- Describe computation RTL in HDL
- Insert abstraction adapters
- Verify the computation RTL

Role of Component Designer

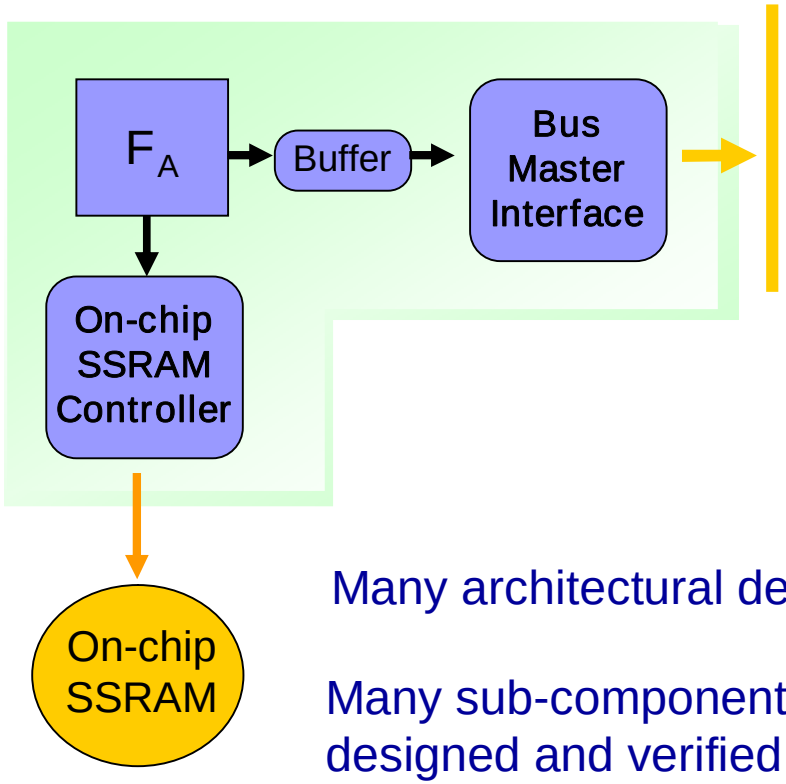


- Refine the communication part of the computation to be best-fit to the target system architecture
- Using CATtree architecture templates

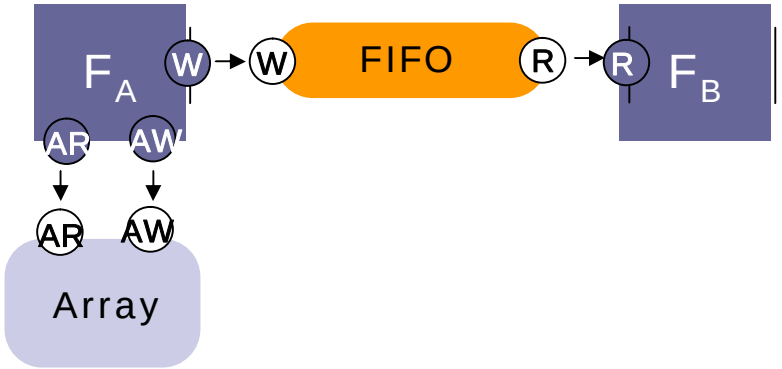
Role of System Designer

Standard I/F-based v.s. CATtree based

Standard I/F-based Component



CATtree-based Component



Many architectural decisions.

Many sub-components to be designed and verified

Complex protocols and difficult compliance test

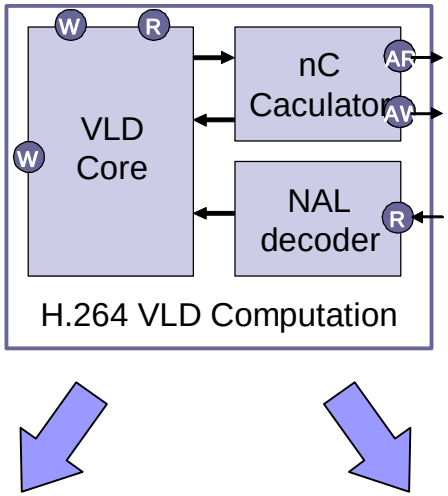
Hints for comm. refinement

Only computation should be designed and verified.

Simple protocols only.Complex protocols are handled by channels

Case Study: H.264 Design

We designed one
H.264 VLD Computation



Configured
Communication parts
for two different
system architecture

Low-End Application

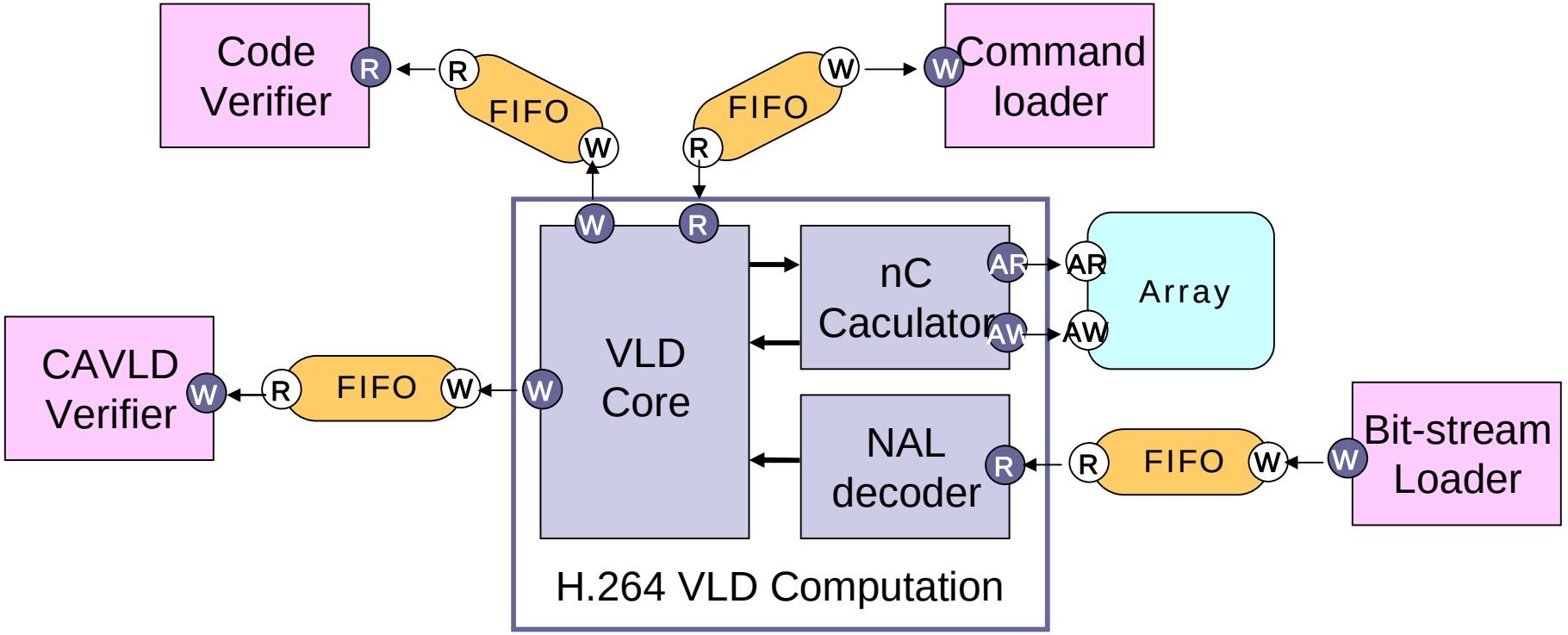
- QCIF 15 frame per second
- No Flexible Macroblock Ordering
- Communication backbone: AHB Bus

High-End Application

- HDV 30 frame per second
- Flexible Macroblock Ordering
- Communication backbone: AHB Bus

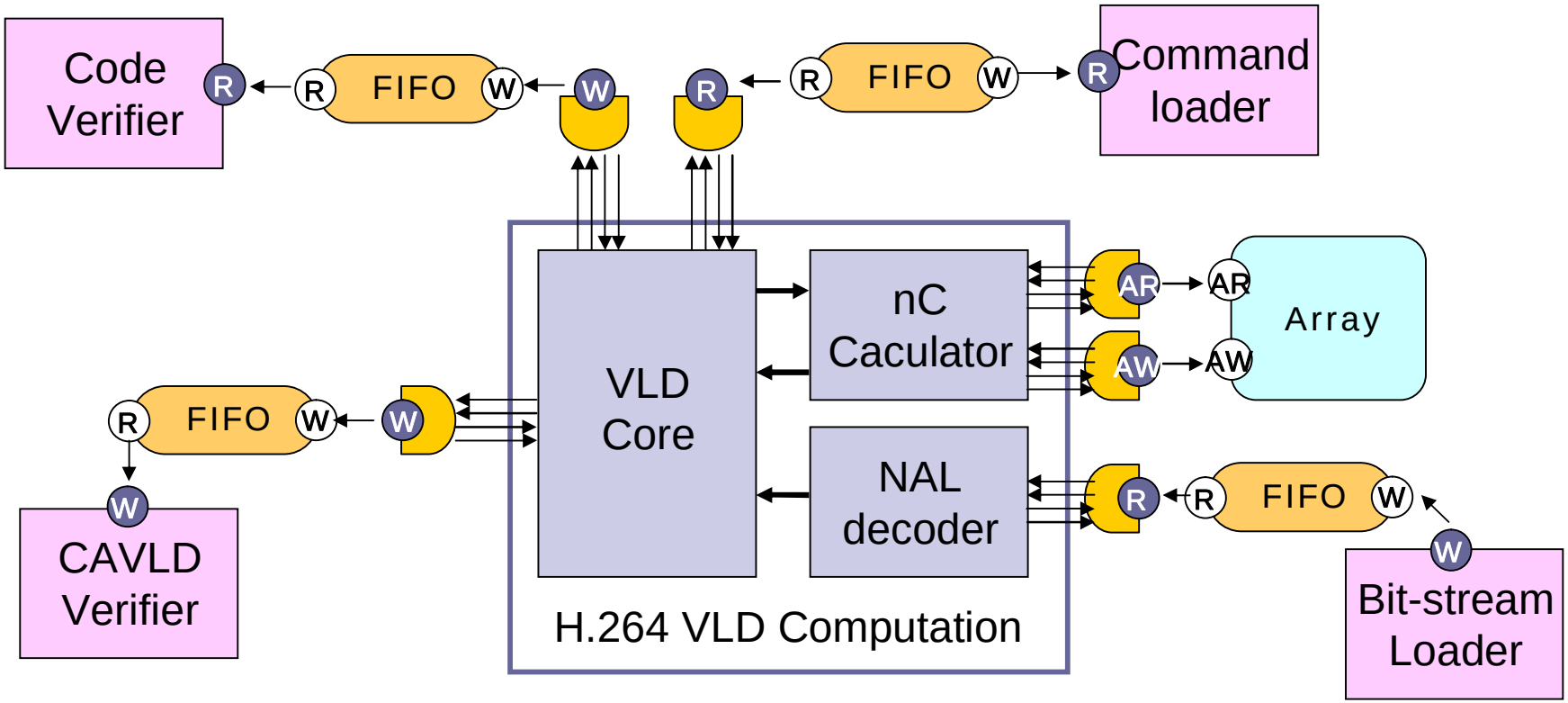
H.264 VLD TLM Design

- A VLD and 4 Testbench computations in TLM
  - VLD contains sub-functions: VLD core, nC calculation, NAL decoder
- 1 Week by 1 Ph.D student
  - Developing the complete testbench consumed the most time



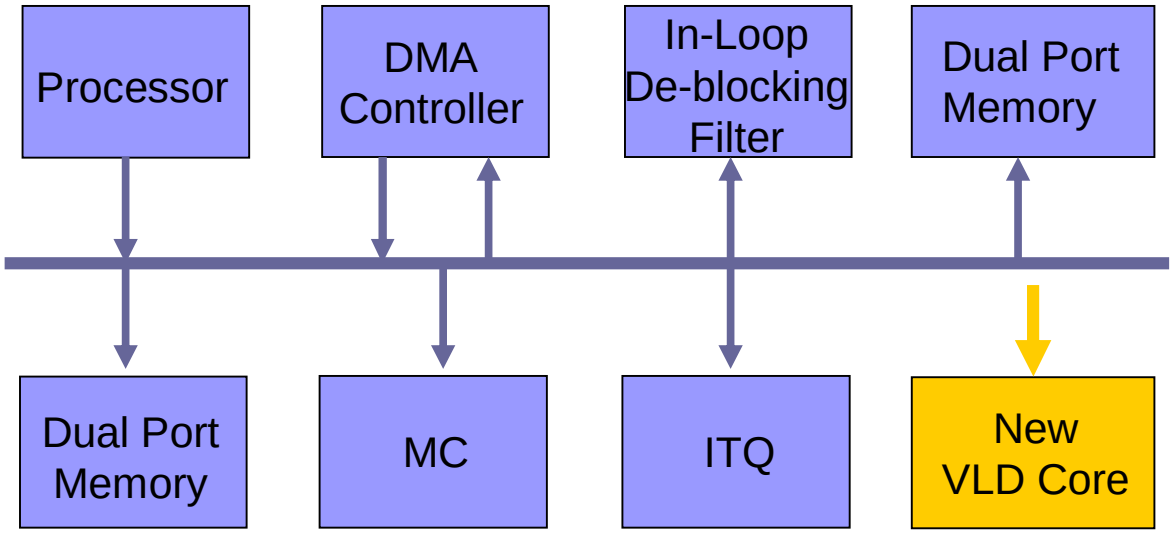
H.264 VLD RTL Design

- Abstraction adapters are inserted between channel TLMs and the VLD computation RTL
- Two days by a Ph.D student



Case 1: Low-End Decoder System

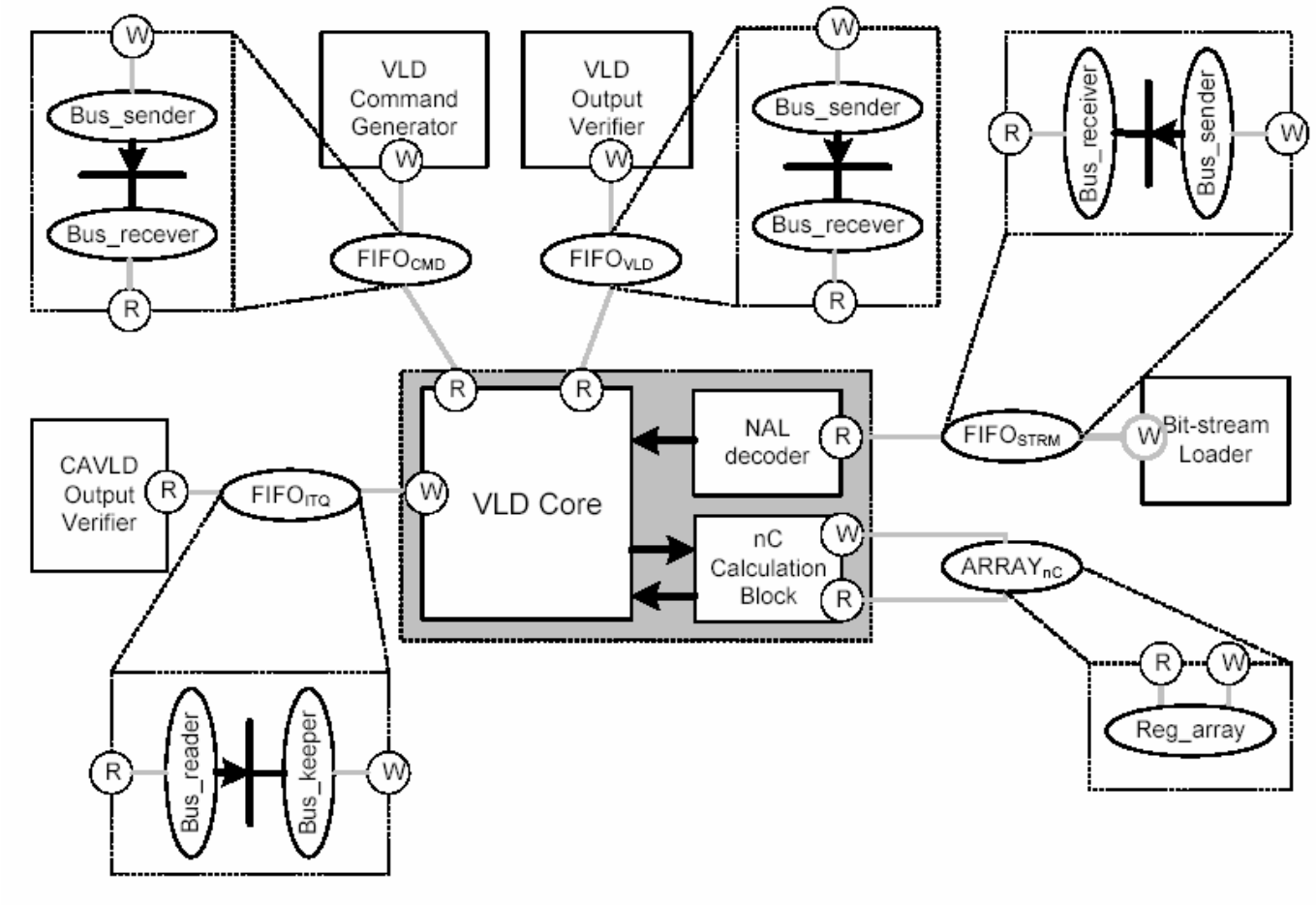
- QCIF 15 fps
- No FMO
- AHB Bus



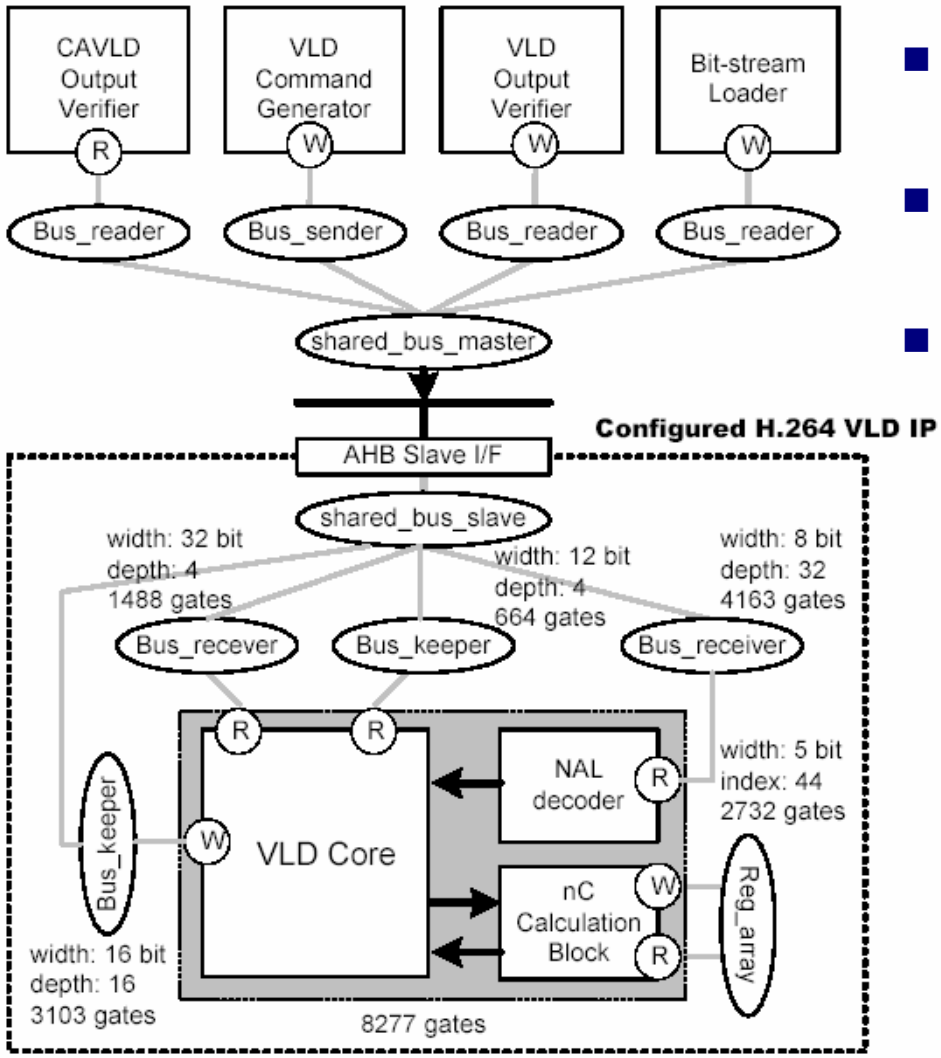
- Component Architecture Decisions (By system designer)
  - Single AHB bus slave only
  - The size of array for nC calculation is 220 bits
  - Implement the array with registers (RegArray architecture)

Communication Refinement

- Refine FIFO channels into Bus-wired FIFO architecture
- Refine Array channel into Register Array FIFO architecture



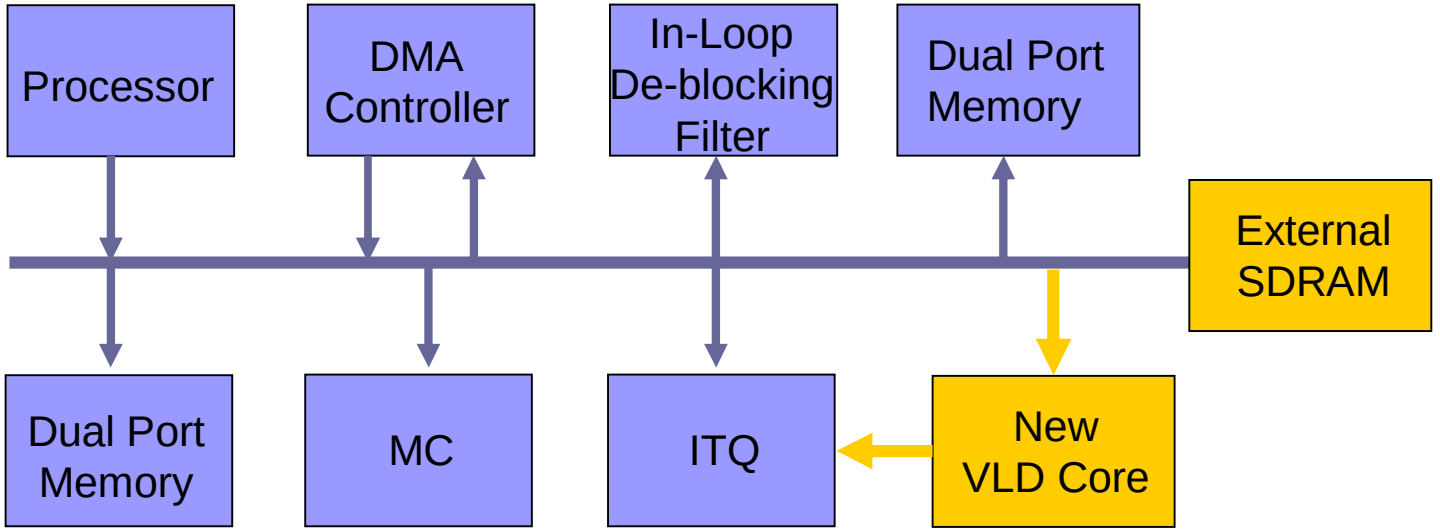
Refined Component Architecture



- Group bus channels and mapped into the shared bus CATtree
- The mapped shared bus is refined into AHB bus system
- Total Gate Count (0.18 um)
  - Computation: 8277 gates@5 ns
  - Communication: 12,150 gates @ 3 ns

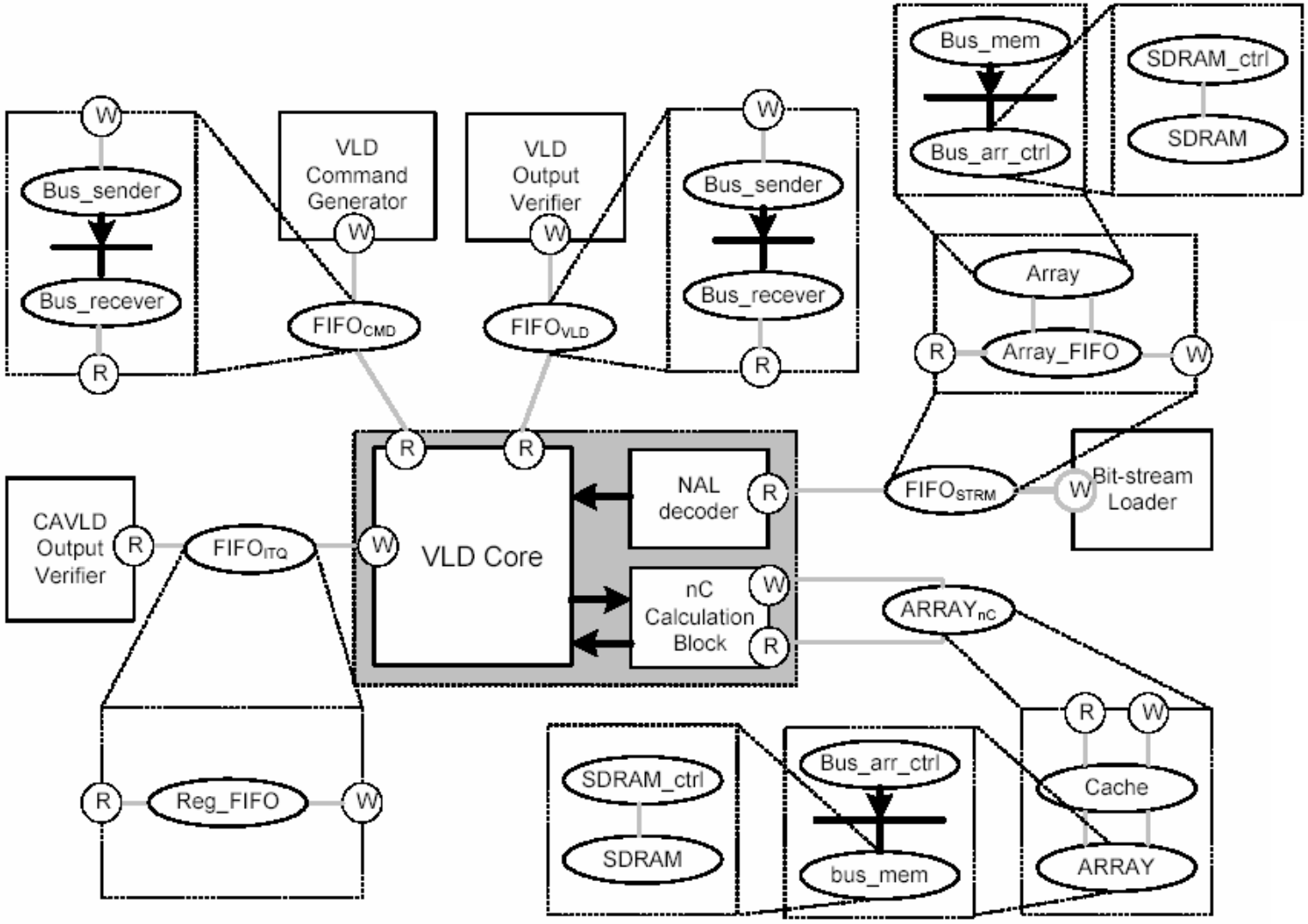
Case 2: High-End Decoder System

- HDV 30 fps
- FMO
- AHB Bus

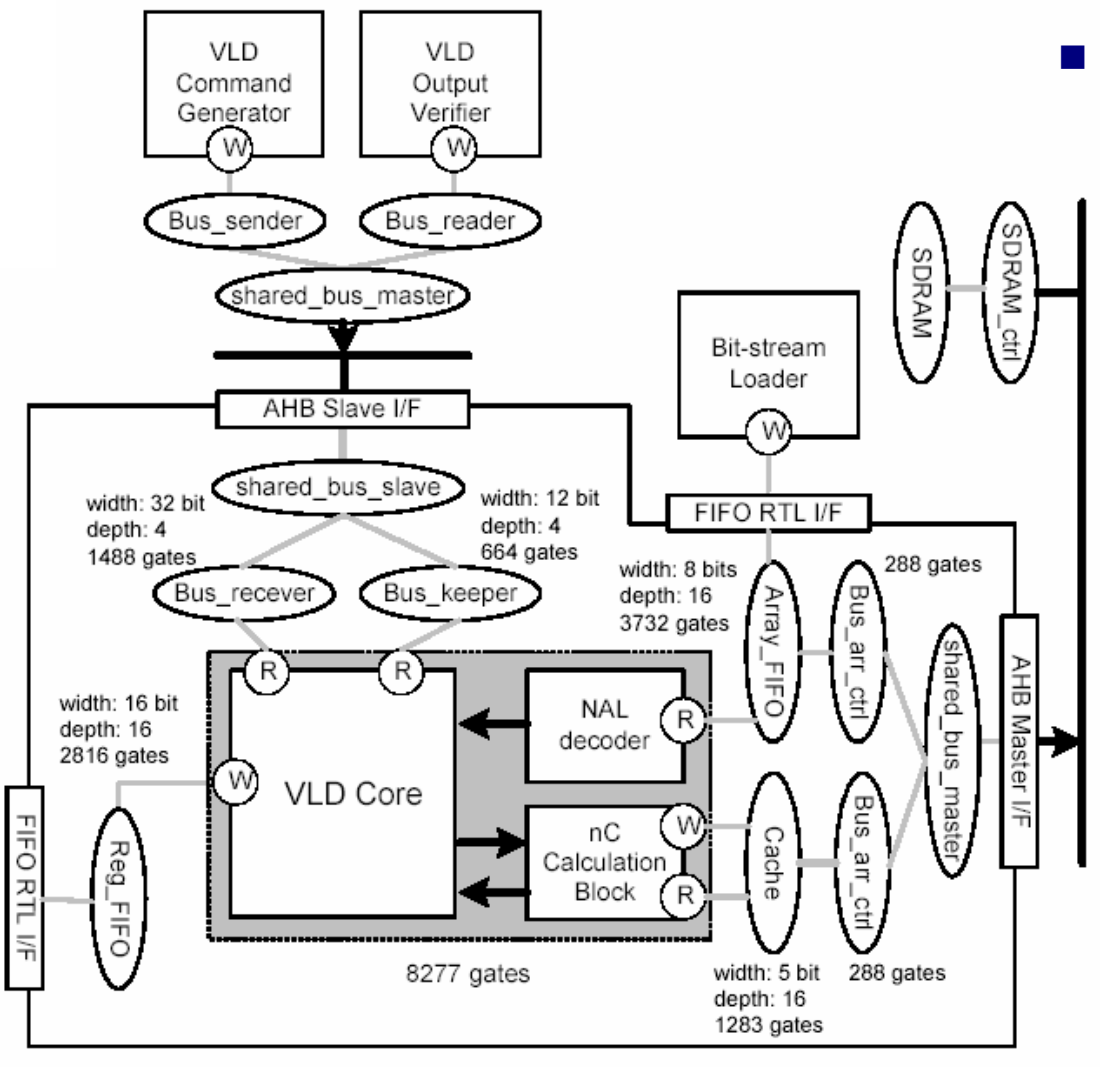


- Component Architecture Decisions (By system designer)
  - Bit-stream FIFO is refined into the FIFO with array architecture
  - The size of array for nC calculation is 92160 bytes
    ➔ Refine the array into the Cached SDRAM architecture
  - Refine CAVLD output FIFO into RegFIFO

Communication Refinement



Refined Component Architecture



- Total Gate Count (0.18 um)
  - Computation:
    8277 gates@5 ns
  - Communication:
    10,559 gates @ 3 ns

Conclusion

- A HW component IP design method that enhances the reusability is explained.
- We changed the role of designers
  - HW Component designer: capture a computation function and describe RTL
  - System designer: configure the communication part of the computation to be best-fit to the system architecture
- The proposed method has merits because:
  - The reusability of a component is greatly enhanced
  - A system designer can explore larger design space to find more better system.
  - Component design productivity is high because we provide completely verified communication templates