MPSoC'05 Tutorial

Software-Centric System-Level Design

11-July-2005

Hiroaki Takada

Graduate School of Information Science, Nagoya Univ.

Chairman, TOPPERS Project

Email: hiro@ertl.jp URL: http://www.ertl.jp/~hiro/

Self Introduction

Major Research Topics

- RTOS for Embedded Systems
- Real-time Scheduling and Analysis
- Embedded Software Development Environments

  ➡ *software researcher (basically)*
- System-Level Design (*started in 5 years ago*)
- Automotive Control Systems

TOPPERS Project

http://www.toppers.jp/

- A project to develop various open-source software for embedded systems including ITRON and OSEK-conformant RTOS.
- RTOS for function-distributed multiprocessors (FDMP) is one of the recent results.

Agenda

- HW Engineers vs. SW Engineers
- "System-Level Design"
- Practical Flow of System-Level Design
- SW-side Expectations on SLD
- SystemBuilder – A SW-Centric SLD Environment
  - Design Flow, System Description
  - HW/SW Partitioning, Implementation Synthesis
  - Design Example, Multiprocessor Extension
- HW-Centric and SW-Centric Approaches to SLD
- Interface Description with Higher Abstraction

! *This material will be put on the following URL.*

http://www.ertl.jp/~hiro/tmp/mpsoc05.pdf

HW Engineers vs. SW Engineers

! *Difficulties of communication between HW engineers and SW engineers is a serious obstacle for deciding appropriate HW/SW partitioning and interface.*

Causes of Communication Difficulties

- Terminology is different eg) “test” vs. “verification”
- What is designed is different (*by definition!*)
- Major design concern is different
  - HW engineers ... performance (cost-performance)
    + uncertainty of physical phenomena
  - SW engineers ... complexity
    → design productivity

→ *One does not understand the other's problem.*

"System-Level Design"

What is "System?"

- HW engineers and SW engineers grasp "system" differently because they design different things.
- A test to distinguish HW and SW engineers:

  "Draw a system diagram of mobile phone."

➡ A system is the whole figure one can see!?

What is "Specification?"

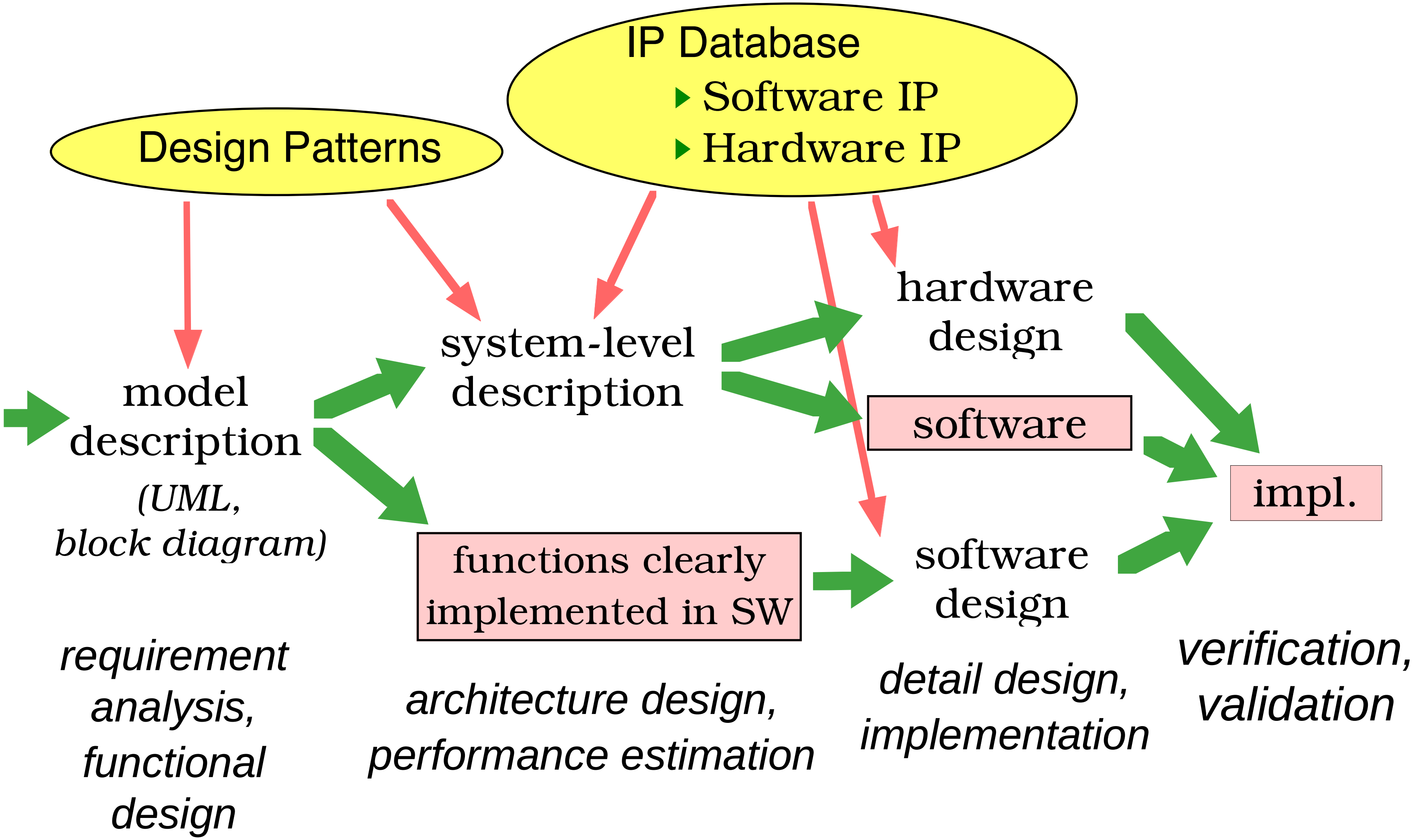
- SW engineers' naive question on SpecC:

  "Though C language is to describe an implementation, why SpecC is for specification?"

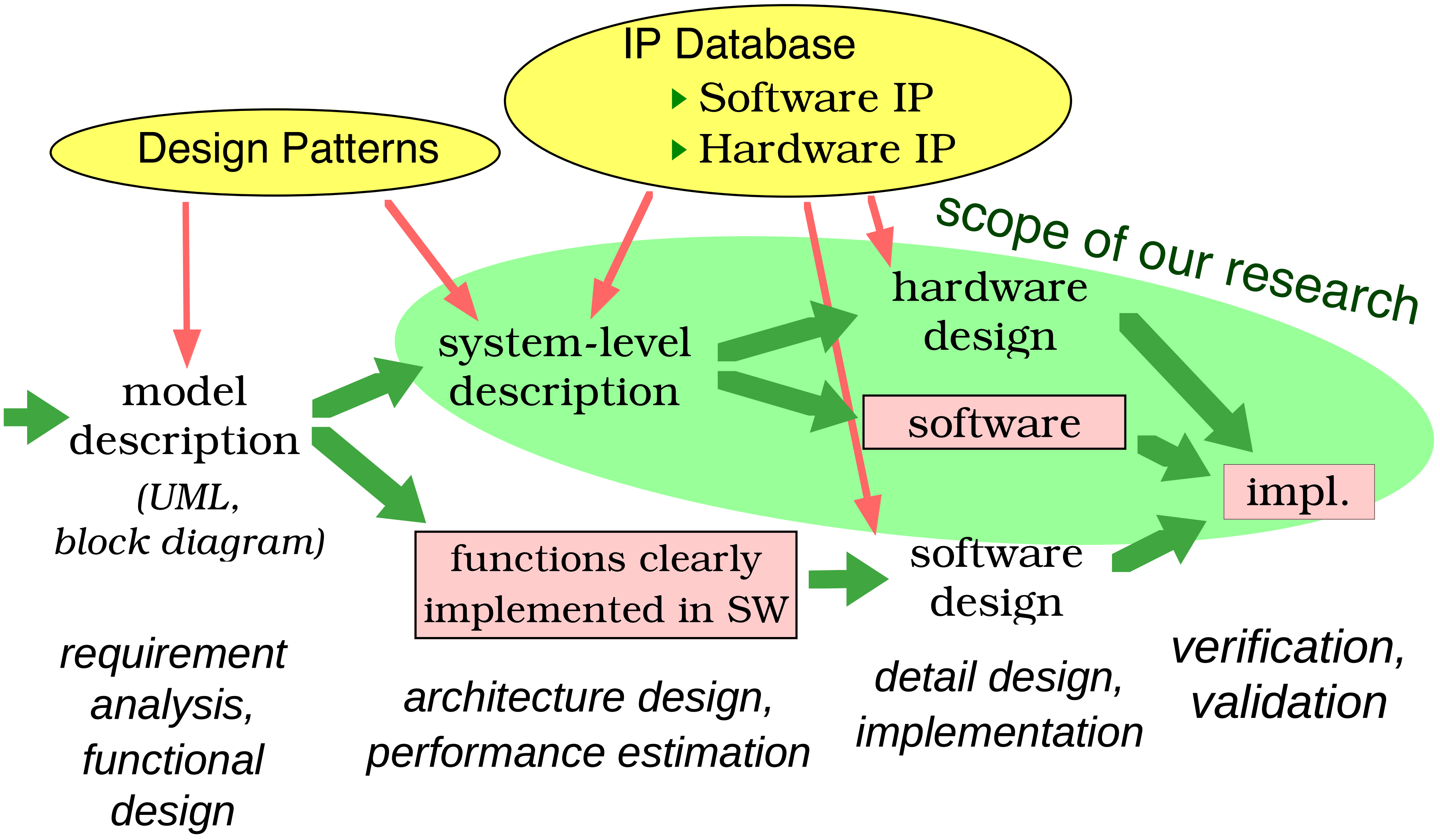
! SW engineers's impl. can be a spec. of HW engineers.

➡ A specification is the description before one starts his design work!?

Practical Flow of System-Level Design (SLD)



Practical Flow of System-Level Design (SLD)


IP Database
Software IP
Hardware IP
Design Patterns
scope of our research
model description
(UML, block diagram)
system-level description
hardware design
software
impl.
functions clearly implemented in SW
software design
requirement analysis, functional design
architecture design, performance estimation
detail design, implementation
verification, validation

SW-side Expectations on SLD

SLD for Design Productivity

- Describing hardware (*or* device) and software that directly handles it (*or* device driver) in one language can improve design productivity.
  - Automatic synthesis of implementation (HW, SW, and interface between them) from system-level description is preferable.
  - Even if the synthesis is difficult, system-level description is useful as the interface description between HW and SW.

  ! *Hard-to-understand device manual is a major source of misunderstanding between HW design and SW design.*

SW-side Expectations on SLD (cont.)

Abstraction of HW/SW Interface Design

- With automatic HW/SW interface synthesis, the abstraction level of HW/SW interface design can be raised.
    - Detail structure of device registers are not important for design and should be determined by a synthesis tool.
    - Bus interfaces should be registered as IPs and an appropriate one should be selected.

    ! *Bus is most important factor in HW/SW interface design for HW engineers, but is not appeared in "system" diagrams of SW engineers!*

SystemBuilder – A SW-Centric SLD Environment

- *SystemBuilder is a SW-centric system-level design environment developed by our laboratory.*

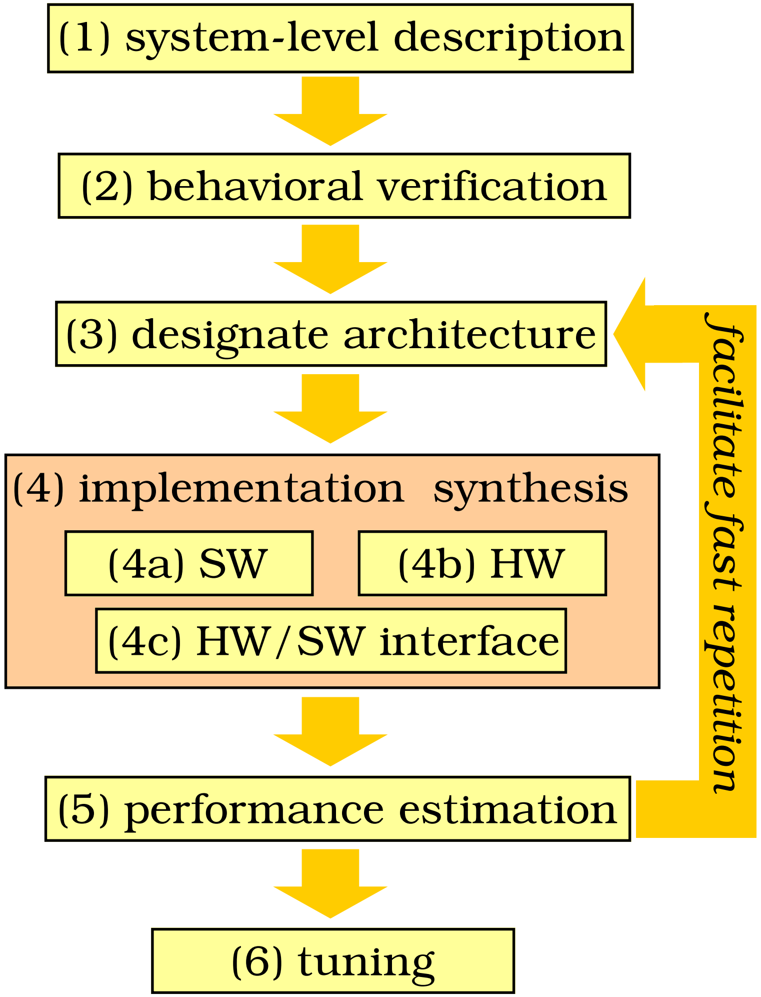
Main Features

- System-level description in C language
- SW/HW partitioning by human designers
- Generation of software running on RTOS
- Automatic behavioral synthesis with a commercial tool
- Automatic SW/HW interface synthesis
- Both uniprocessor and multiprocessor supported
- SW/RTOS/HW cosimulation at various abstraction levels
- FPGA implementation

SystemBuilder (cont.)

Design Flow

- HW and SW described with a system-level language
- Behavioral verification of the system-level description
- Implementation synthesized by the tool with designated HW/SW partitioning (architecture)
- Performance estimation of the implementation
- Repeat the process if the performance is insufficient


(1) system-level description
(2) behavioral verification
(3) designate architecture
(4) implementation synthesis
(4a) SW
(4b) HW
(4c) HW/SW interface
(5) performance estimation
(6) tuning
facilitate fast repetition

System Description in SystemBuilder

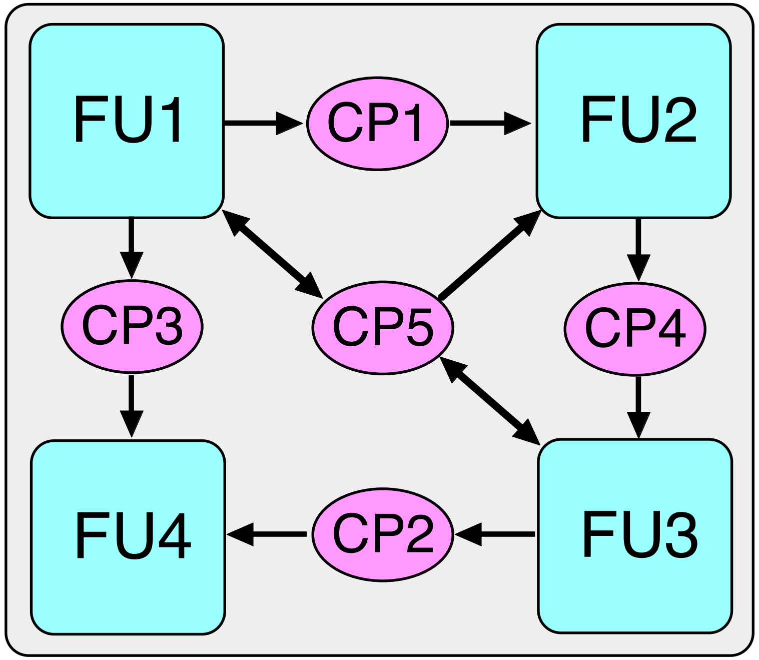
- A *system* is described as a set of function units and communication channels among them.

Function Units (FU)

- Unit of concurrent execution
- Unit of HW/SW partitioning
- SW: task *or* thread
- HW: module *or* behavior

Communication Primitives (CP)

- Non-Blocking Communication (NBC) *or* Register
- Blocking Communication (BC) *or* FIFO
- Memory (MEM)



System Description in SystemBuilder (cont.)

System Description Language

- Each FU is described in normal ANSI-C language as a function including an infinite loop.
  - C-based language is preferable for SW engineers.
  - Some extensions to C language are preferable but are not MUST.
  - We started with SpecC, but gave up because no behavioral synthesis tool is available.
  - I don't think SystemC suitable for SW engineers (I don't like it, at least).
- Specification of CPs and how FUs and CPs are connected are described with a simple script in a system definition file (SDF).

System Description in SystemBuilder (cont.)

An SDF Example

```
SYS_NAME = test
SW = FU1, FU4
HW = FU2, FU3

BCPRIM   cp1, SIZE = 32
BCPRIM   cp2, SIZE = 32
NBCPRIM  cp3, SIZE = 32
MEMPRIM  cp4, SIZE = 32
NBCPRIM  cp5, SIZE = 16

BEGIN_FU
 NAME   = FU1
 FILE   = "fu1.c"
 USE_CP = cp1(OUT),
          cp3(OUT),
          cp5(INOUT)
END
```

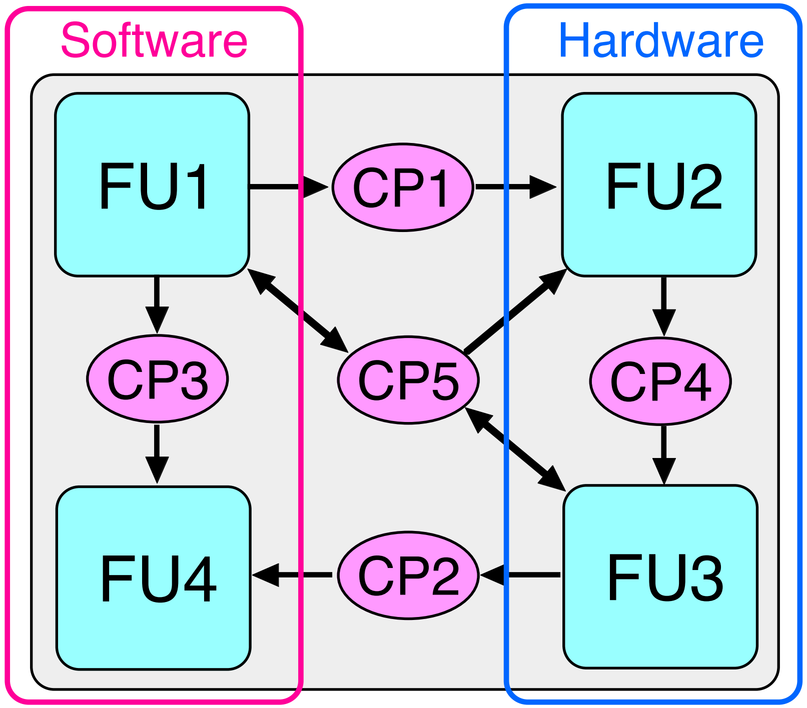
```
BEGIN_FU
 NAME   = FU2
 FILE   = "fu2.c"
 USE_CP = cp1(IN),
          cp4(OUT), cp5(IN)
END

BEGIN_FU
 NAME   = FU3
 FILE   = "fu3.c"
 USE_CP = cp2(OUT),
          cp4(IN), cp5(INOUT)
END

BEGIN_FU
 NAME   = FU4
 FILE   = "fu4.c"
 USE_CP = cp2(IN), cp3(IN)
END
```

HW/SW Partitioning in SystemBuilder

- System designer designates HW/SW partitioning in SDF as below.

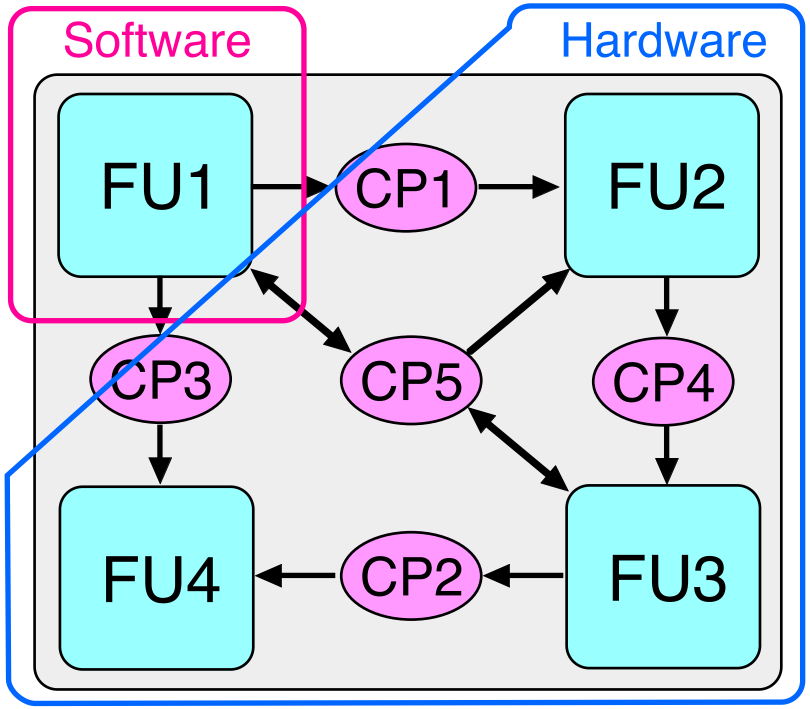
SW = FU1, FU4

HW = FU2, FU3



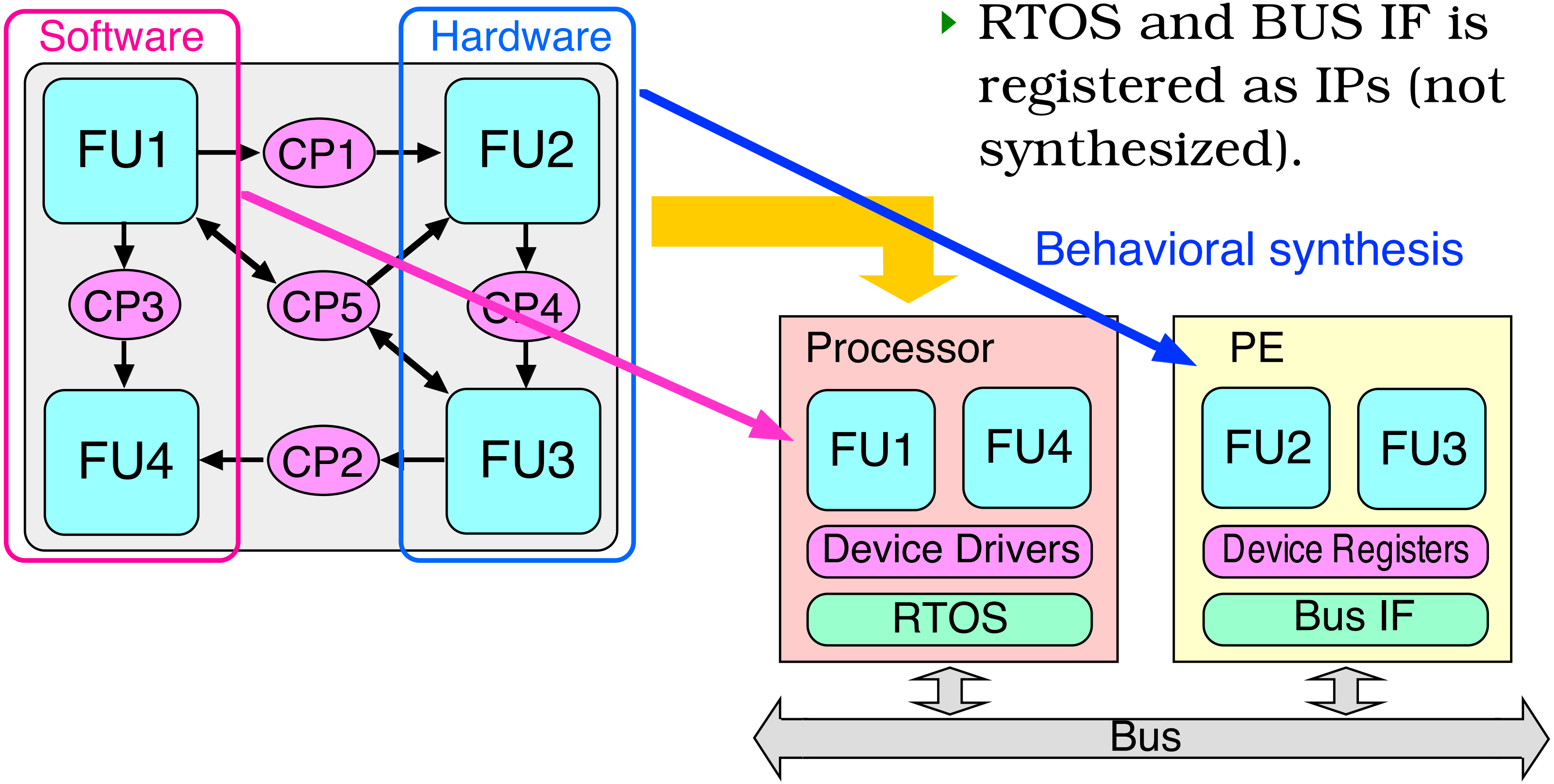
SW = FU1

HW = FU2, FU3, FU4



Implementation Synthesis in SystemBuilder

- Behavioral synthesis with a commercial tool.
- Interface synthesis between FUs with our tool.

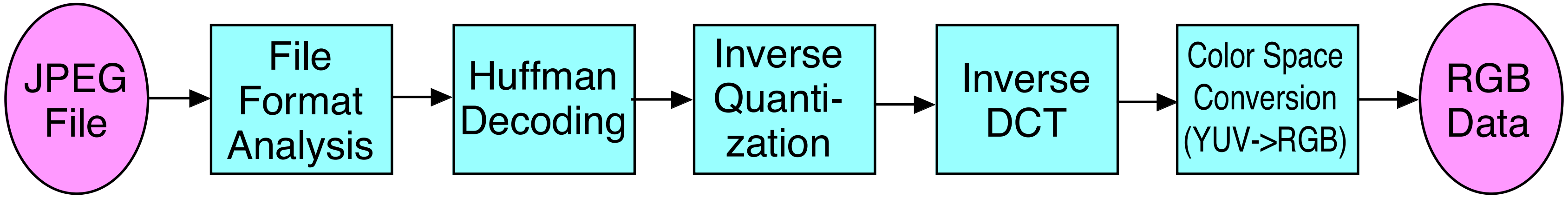


- RTOS and BUS IF is registered as IPs (not synthesized).

Design Example with SystemBuilder

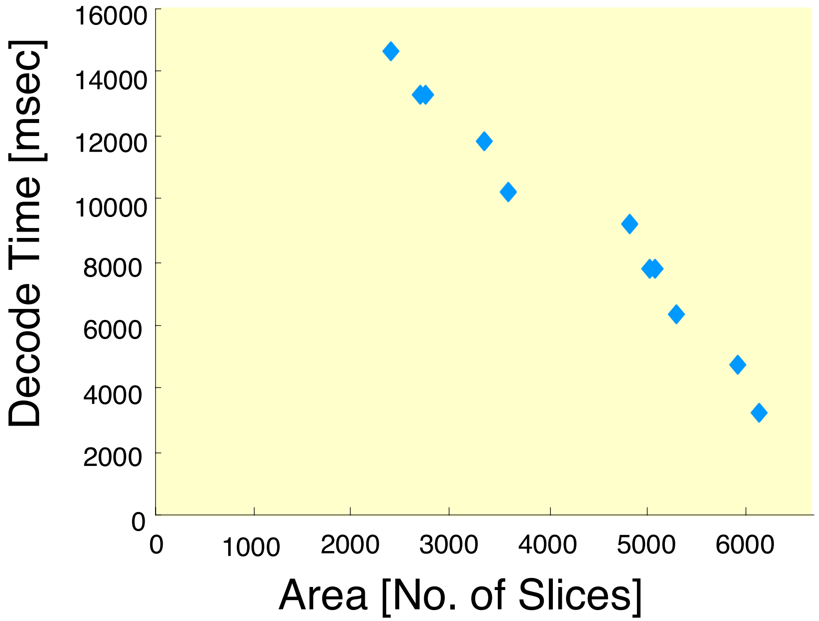
JPEG Decoder Example

- JPEG decoder is described with 5 FUs as below.



- Implemented on FPGA with soft-core processor.
- Performance estimation of about 10 HW/SW partitionings are shown on the right graph.

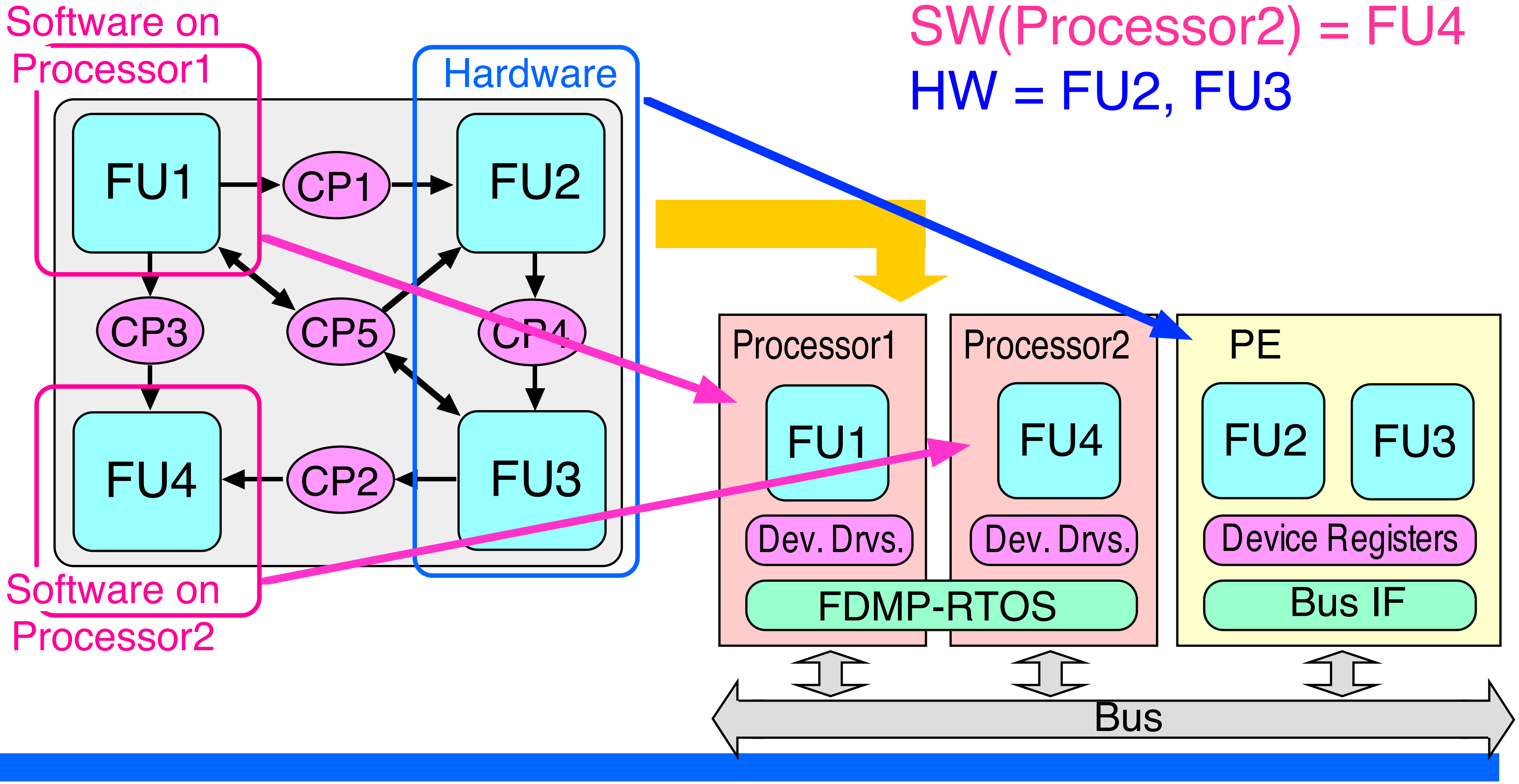
! *Estimation can be done in several hours.*



Multiprocessor Extension of SystemBuilder

- FUs implemented in SW can now be assigned to different processors.

SW(Processor1) = FU1
SW(Processor2) = FU4
HW = FU2, FU3

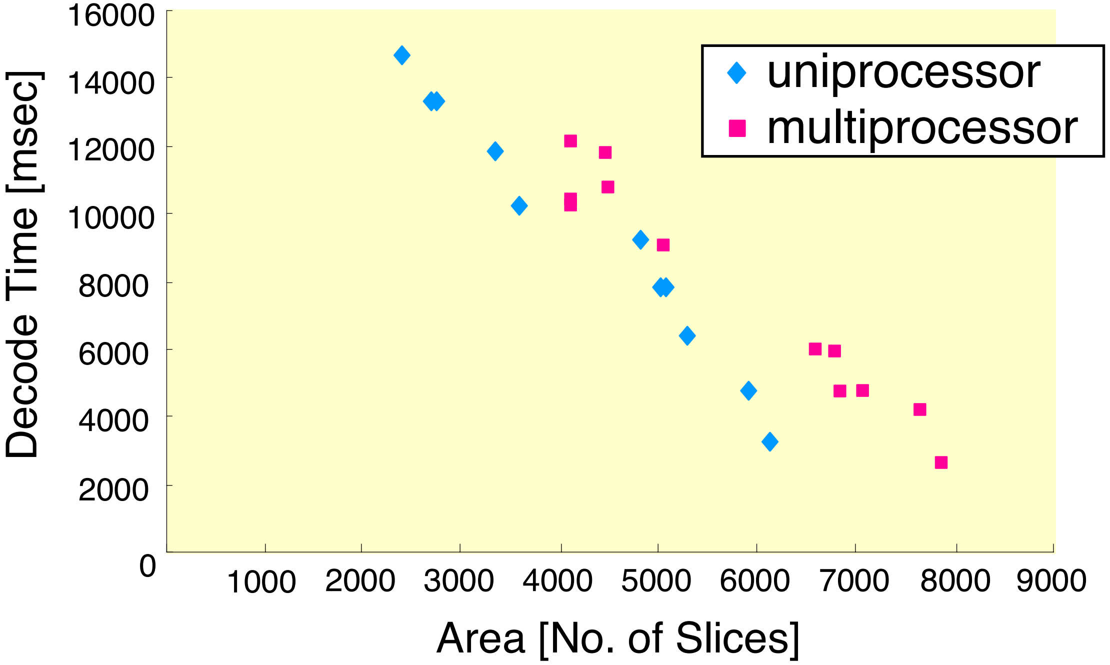


Multiprocessor Extension of SystemBuilder (cont.)

Performance Estimation of JPEG Decoder Example

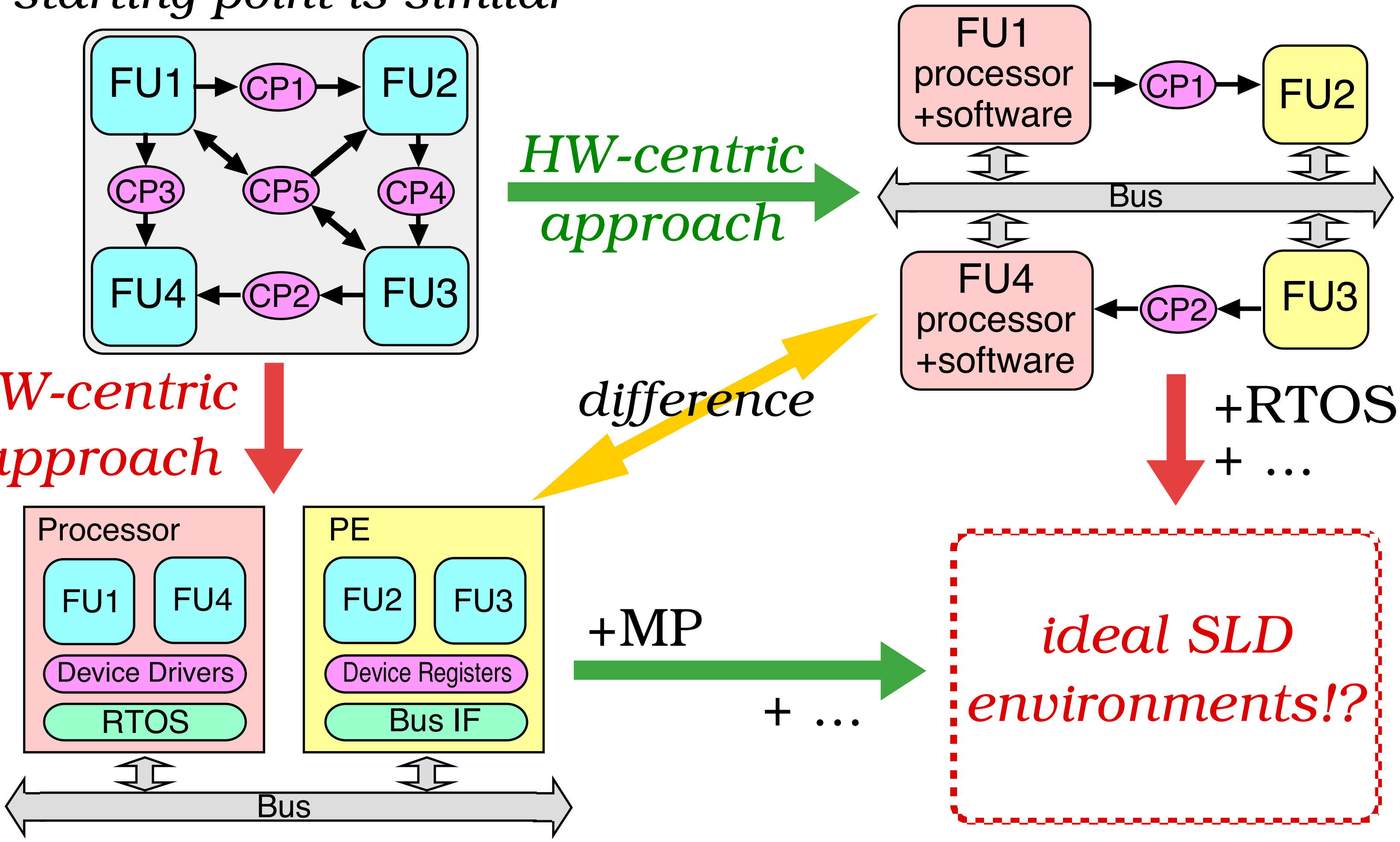
- Implemented on FPGA with 2 soft-core processors.
- Performance estimation of about 10 configurations.

! *Again, estimation can be done in several hours.*



HW-Centric and SW-Centric Approaches to SLD

starting point is similar



Current and Future Work on SystemBuilder

Evaluations

- The effectiveness of SystemBuilder is now under evaluations by some companies.

Further Extensions

- Wider space for architecture explorations.
- (HW/SW) interface description with higher abstraction and interface synthesis from it.
- HW/SW cosimulation environment supporting multiprocessors.
- Profiling of system-level description.

Interface Description with Higher Abstraction

Goal

- *Interface description in the system level (behavioral level) should be independent of architecture.*

⬆

- System-level description should not be modified during architecture exploration phase to facilitate explorations.

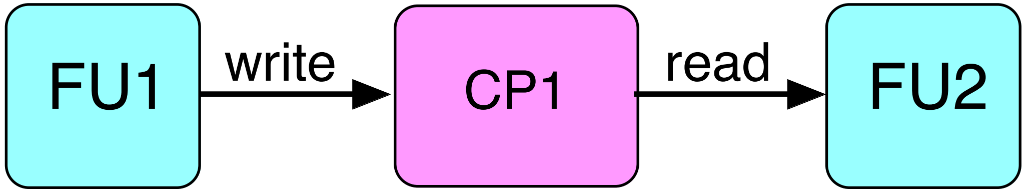
Problem of the Current Method

- Current method does not achieve this goal, because abstraction level of interface description is still low.

Interface Description with Higher Abstraction (cont.)

Example:

- FU1 produces and writes large data on CP1.
- FU2 reads and consumes it.

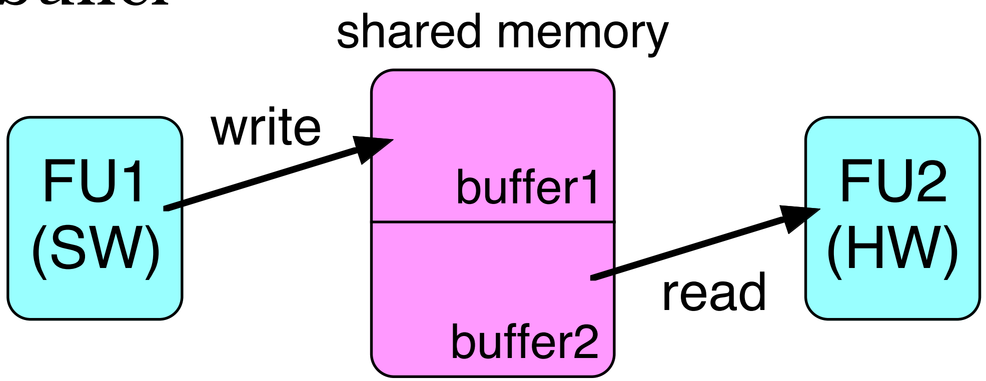


- CP1 is basically a FIFO.
- The simple UNIX-like write/read APIs are not solution.
  - Those APIs write/read the whole data at once.
  - They require local buffers both in FU1 and FU2 in addition to a buffer in CP1 resulting in large overhead and large memory consumption.

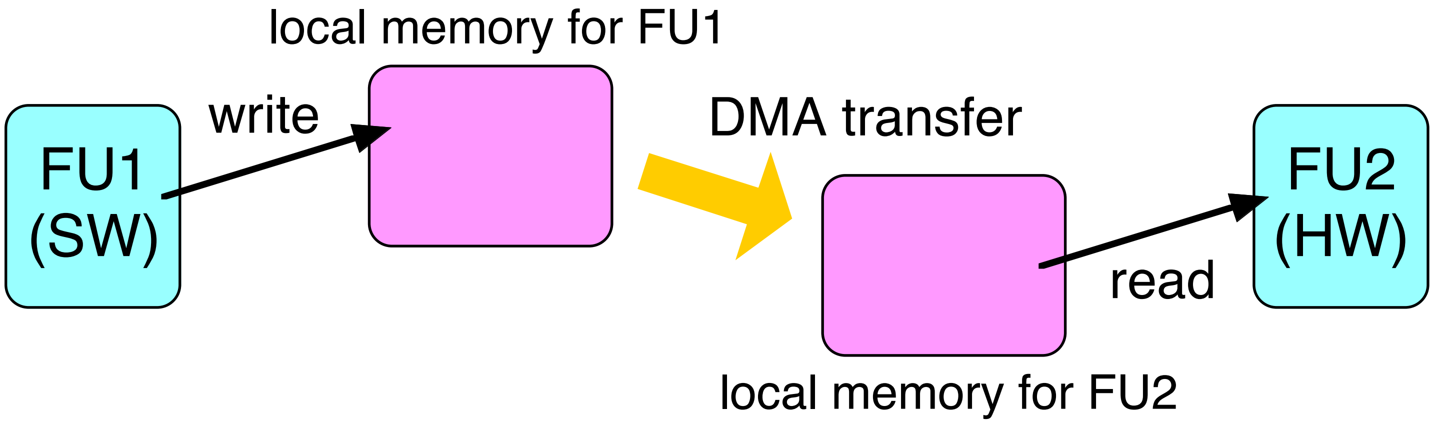
Interface Description with Higher Abstraction (cont.)

Two Typical Implementations

- Double buffer



- Local buffers + DMA transfer



Interface Description with Higher Abstraction (cont.)

Goal (in this Example)

- Because which implementation is appropriate depends on various factors, the decision should be done through architecture explorations.

↓

- *In system-level description, both implementations should be synthesized from the same interface description.*

Problem of the Current Method (in this Example)

- Memories, registers, and FIFOs must be explicitly described with the current method.
- Therefore, the interface description determines the implementation.

Interface Description with Higher Abstraction (cont.)

Approach

- Interface description with higher abstraction.
- Buffer handling should be explicitly described.

FU1

get an empty buffer;
write data to the buffer;
(in arbitrary order)
send the buffer;

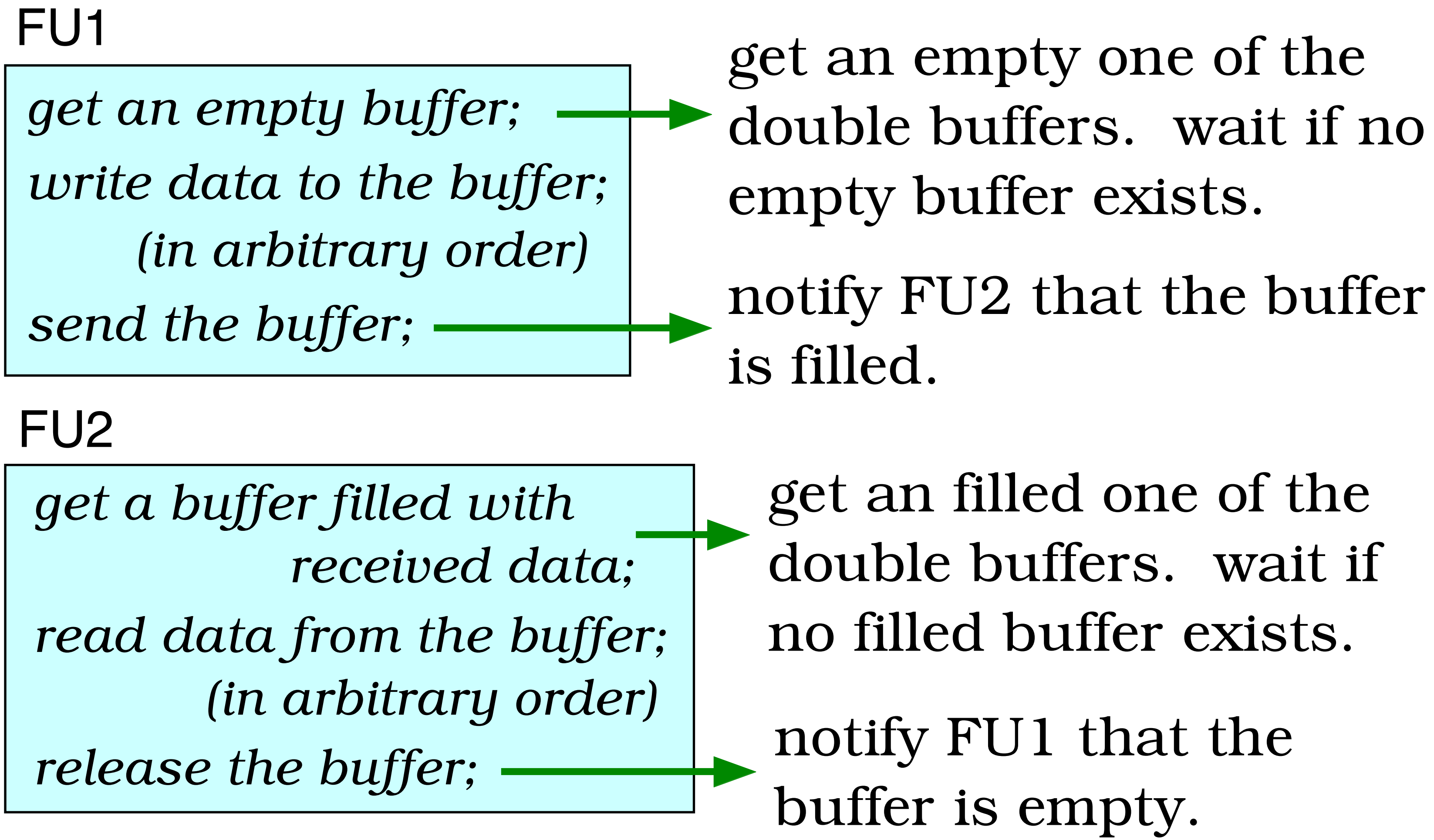
FU2

get a buffer filled with
received data;
read data from the buffer;
(in arbitrary order)
release the buffer;

- The following proposal is a good starting point.
  - Pieter van der Wolf, et. al: Design and Programming of Embedded Multiprocessors: An Interface-Centric Approach, CODES/ISSS2004.

Interface Description with Higher Abstraction (cont.)

Synthesis of Double Buffer



FU1

get an empty buffer;

write data to the buffer;

(in arbitrary order)

send the buffer;

get an empty one of the double buffers. wait if no empty buffer exists.

notify FU2 that the buffer is filled.

FU2

get a buffer filled with received data;

read data from the buffer;

(in arbitrary order)

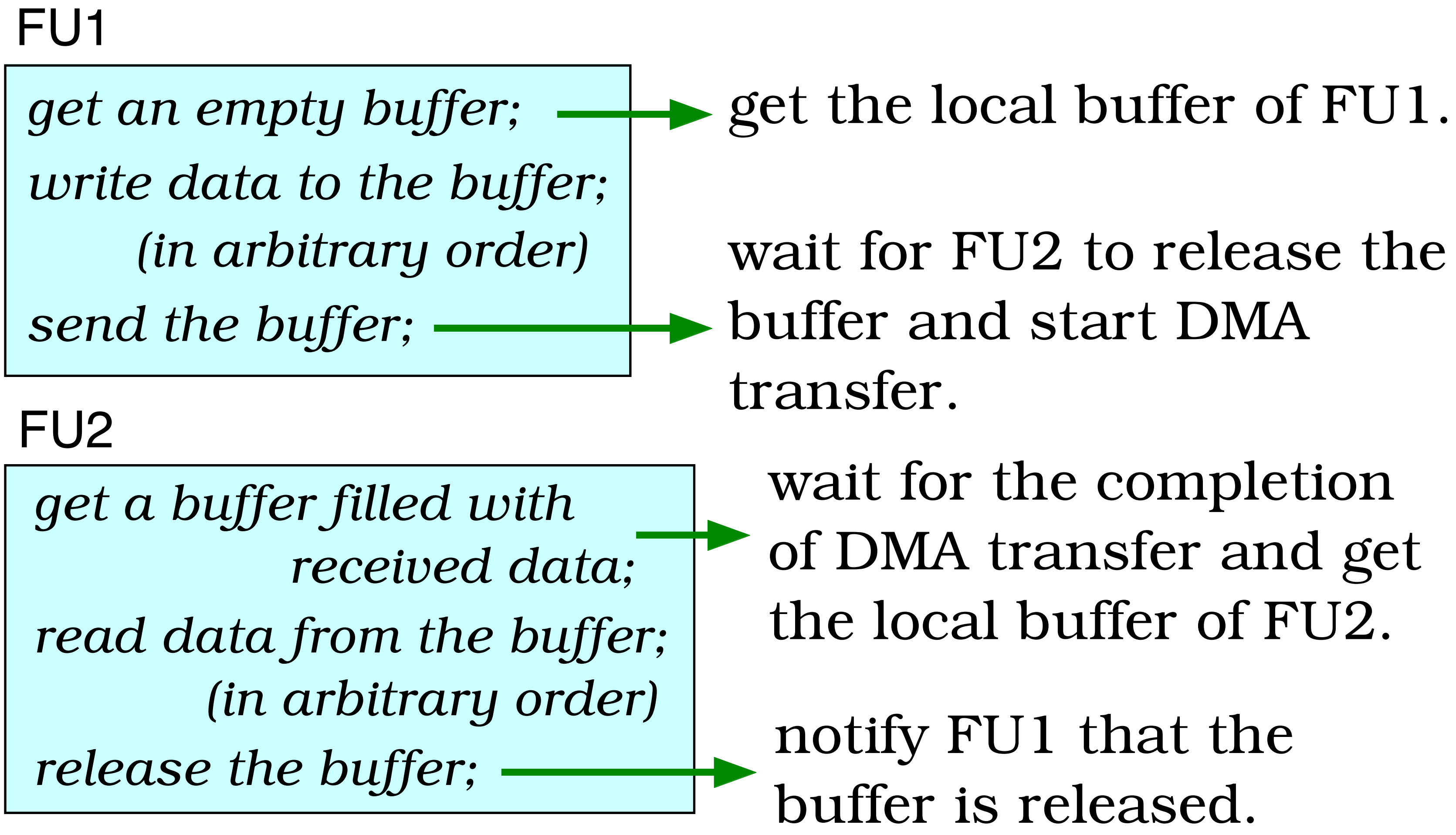
release the buffer;

get an filled one of the double buffers. wait if no filled buffer exists.

notify FU1 that the buffer is empty.

Interface Description with Higher Abstraction (cont.)

Synthesis of Local Buffers + DMA Transfer



FU1

get an empty buffer; → get the local buffer of FU1.

write data to the buffer;
(in arbitrary order)

send the buffer; → wait for FU2 to release the buffer and start DMA transfer.

FU2

get a buffer filled with received data; → wait for the completion of DMA transfer and get the local buffer of FU2.

read data from the buffer;
(in arbitrary order)

release the buffer; → notify FU1 that the buffer is released.

Interface Description with Higher Abstraction (cont.)

Our Current Work

- We are defining communication primitives and APIs with higher abstraction.
- We are investigating how to synthesize interface implementation from them and developing a synthesis tool.

Difficulty Encountered

- Existing behavioral synthesis tool is not sufficient for elegantly synthesizing such interfaces.

Future Direction: Standardization?

- Standardization of system-level interface primitives and APIs is effective to make FUs reusable to other SLD environments.

Concluding Remarks

- HW engineers' and SW engineers' expectations on SLD are far apart.
- Most SLD environments currently available are based on HW-centric approach. They do not satisfy the SW-side expectations on SLD.
- We are developing a SW-centric SLD environment, named SystemBuilder.
- I hope adopting the results of both HW-centric and HW-centric approaches leads to a more practical SLD environment.
- Necessity of interface description with higher abstraction is discussed.