An Interface-Circuit Synthesis Method with Configurable Processor Core in IP-Based SoC Designs

Shunitsu Kohara, Naoki Tomono, Junpei Uchida,
Yuichiro Miyaoka, Nozomu Togawa,
Masao Yanagisawa, Tatsuo Ohtsuki

Department of Computer Science, Waseda University

January 26th, 2006

Outline

- Introduction
- IP-Based SoC Design Method
  - Design Flow
  - Architecture Model
  - Interface
- IFC_Synthesizer
  - IFC Architecture
  - IFC Synthesis Method
- Experimental Results
- Conclusion

Introduction

Introduction (1)

- ✓ Requirements for SoC design
  - ✓ Small Area, high performance and low energy
  - ✓ Design in a short period and low cost

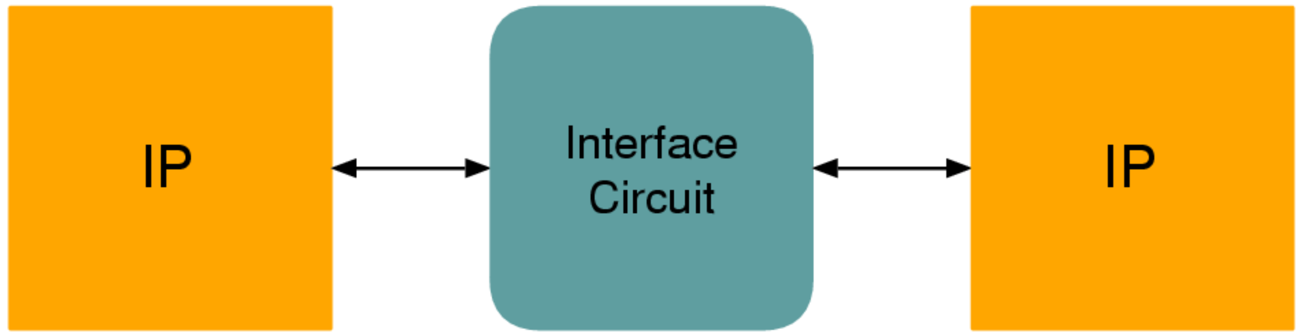
- ✓ Method for designing in a short period
  - ✓ IP-Based Design：

    Reuse of IPs (Intellectual Property)
  - ✓ Configurable Module：

    Adjustment for performance / area cost

Introduction (2)

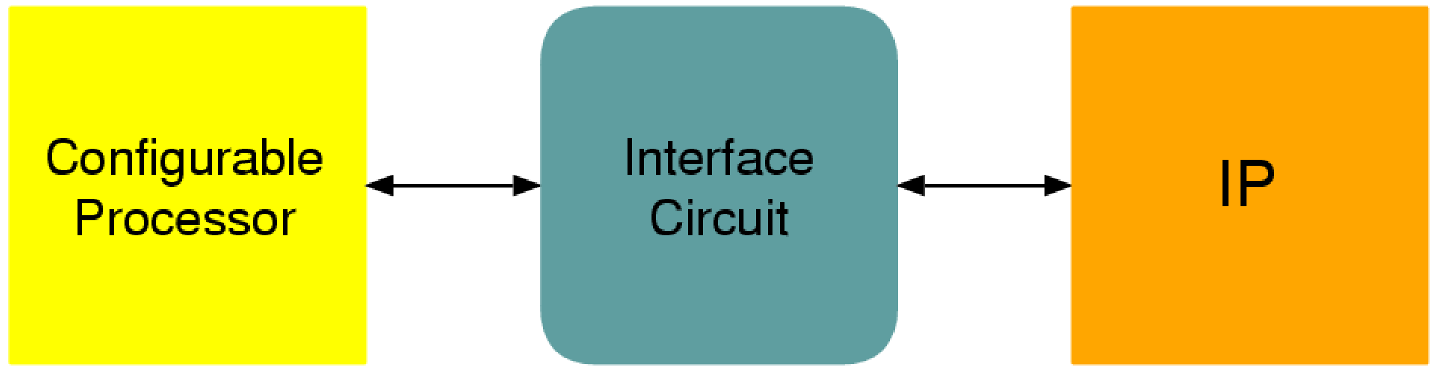
IP-Based SoC Design:



- ✓ Interface circuit should be designed automatically
- ✓ Previous works about an interface generation target such a situation

Introduction (3)

IP-Based SoC Design with Configure Processor Core:



✓ Interface circuit is affected by the configurable processor core

Introduction (4)

Our Proposal:

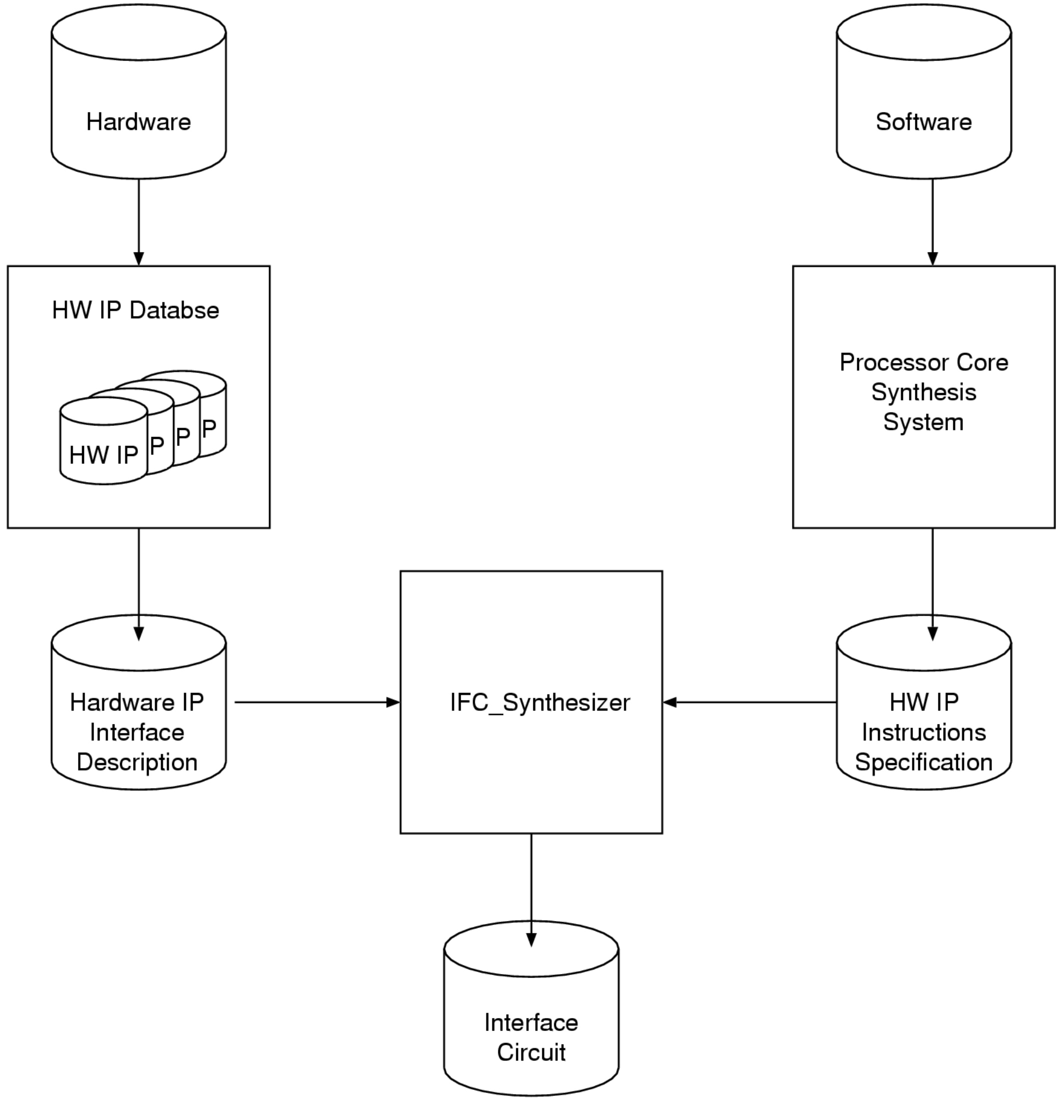
- ✓ IFC Architecture
- ✓ IFC Synthesis Method

IFC: An Interface Circuit between
a Configurable Processor Core and
a Hardware IP

IP-Based SoC Design Method

Design Flow

1. Application HW/SW partitioning
2. (HW part) Selecting HW IPs from DB
3. (SW part) Processor Core Synthesis
4. Interface Circuit Synthesis


Hardware
Software
HW IP Databse
HW IP
P
P
P
Processor Core Synthesis System
Hardware IP Interface Description
IFC_Synthesizer
HW IP Instructions Specification
Interface Circuit

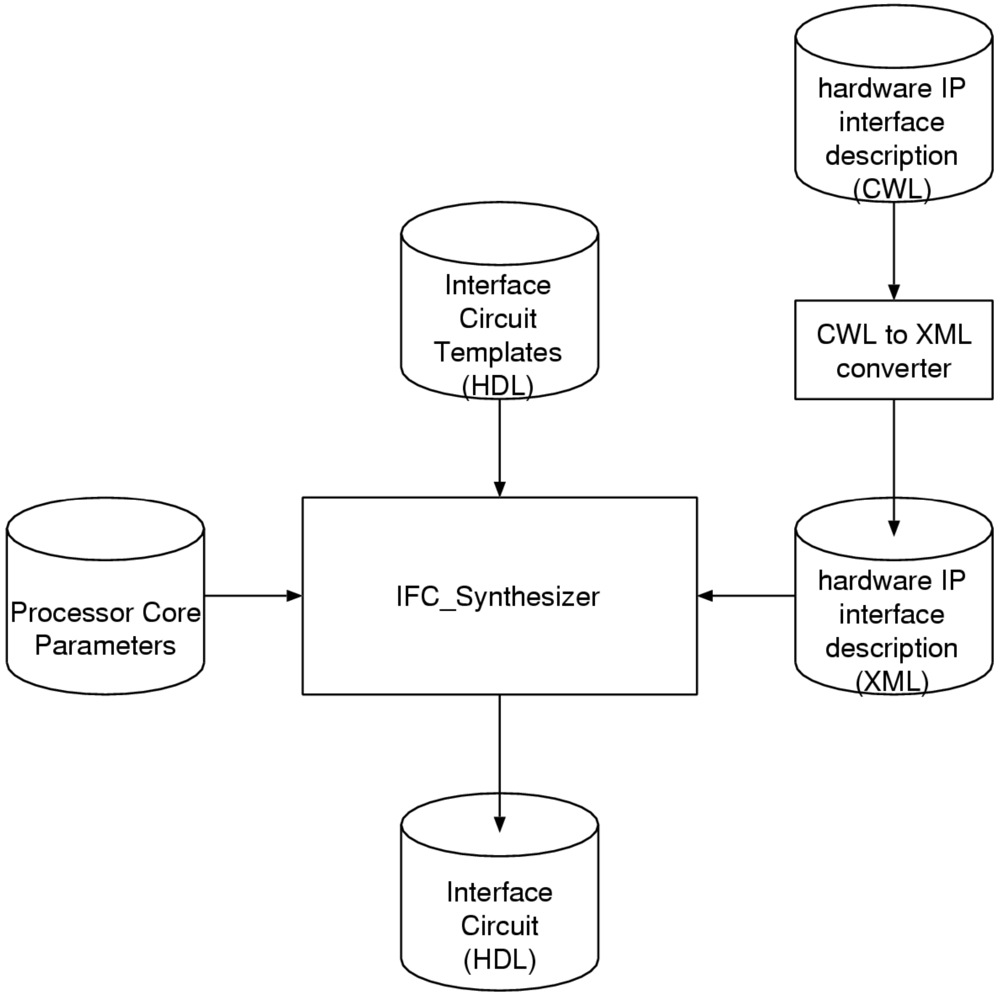
IFC_Synthesizer

Input:

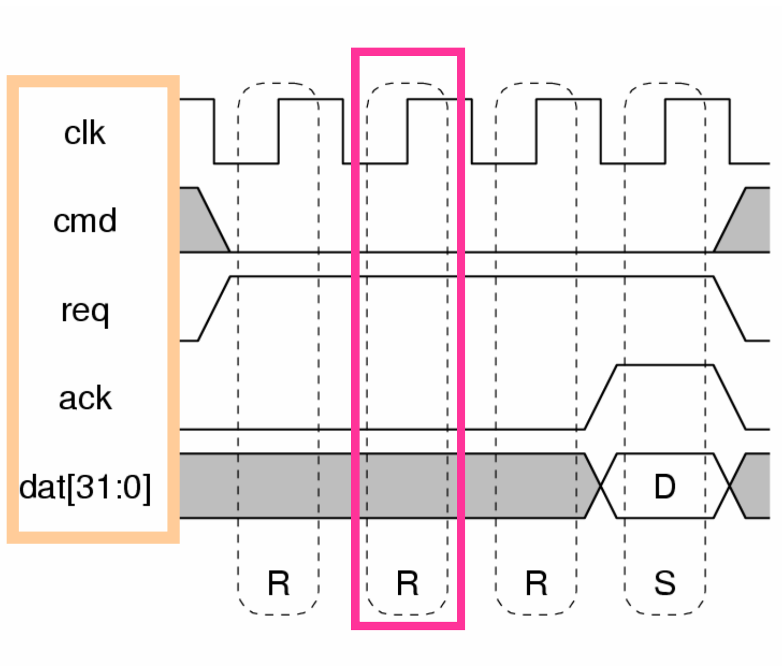
- ✓ Hardware IP Interface Description (CWL)
- ✓ Processor Core Parameters
- ✓ IFC Templates (HDL)

Output:

- ✓ Interface Circuit (HDL)

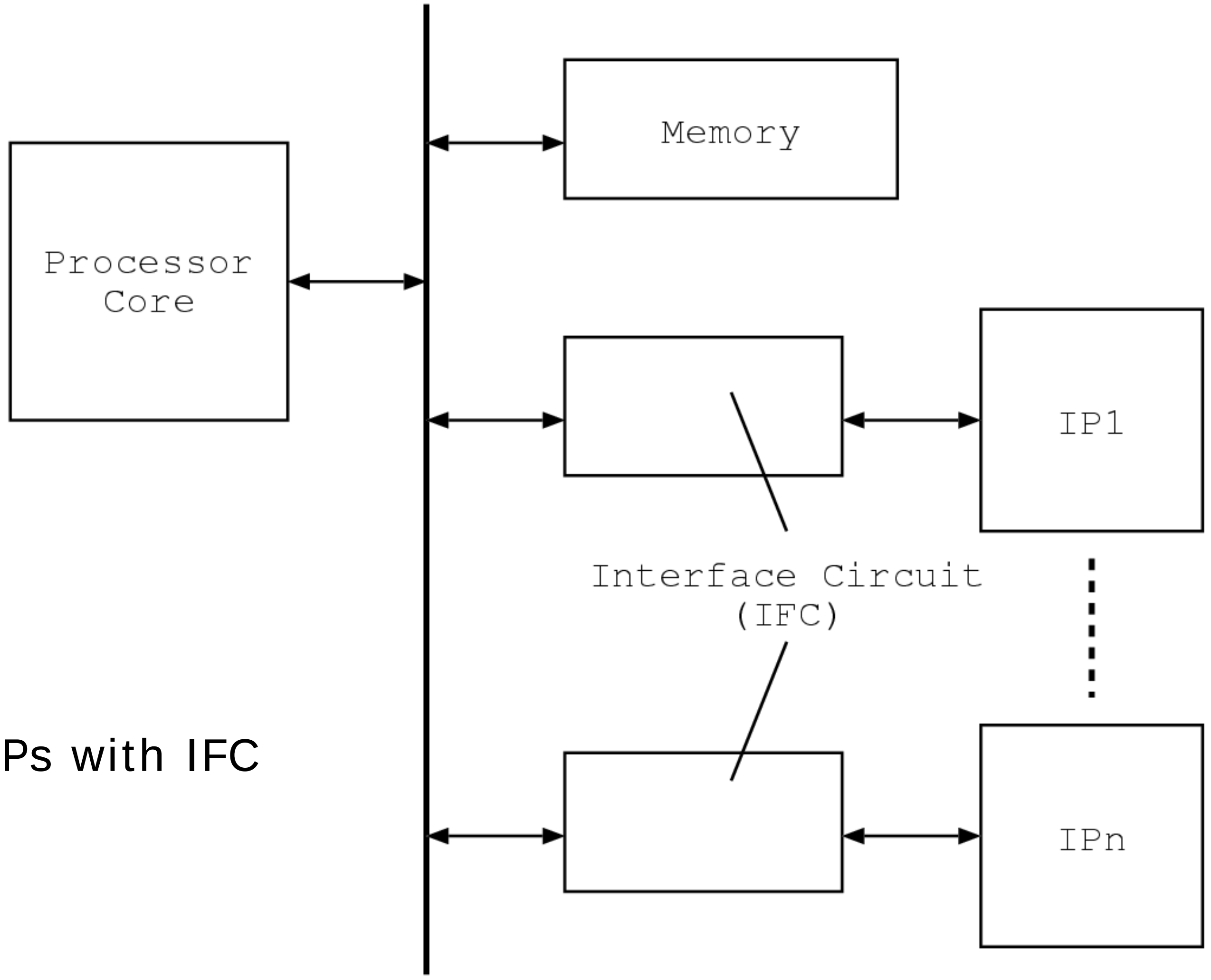

hardware IP interface description (CWL)
CWL to XML converter
Interface Circuit Templates (HDL)
Processor Core Parameters
IFC_Synthesizer
hardware IP interface description (XML)
Interface Circuit (HDL)

Interface Description Language: CWL (Compornent Wrapper Language)



```
interface ex1:
  port:
    input.clock      clk;
    input.control        cmd;
    input.control        req;
    output.control   ack;
    output.data[31:0]    dat;
  endport
  alphabet:
    signalset all = {clk, cmd, req, ack, dat};
      W:         {R, ?, 0, 0, ?};
      O(Xa):     {R, 0, 1, 0, ?};
      ERR:  {R, 0, 1, 1, ?};
    endsignalset
  endalphabet
  word:
    read(reg[9:0] Xa, reg[7:0] Xd) :
                Q(Xa) W{0,8}  [ O(Xd) | ERR ];
  endword
endinterface
```

Architecture Model



- ✓ A Processor Core
- ✓ A Memory
- ✓ Several Hardware IPs with IFC
- ✓ A Shared Bus

Interface between Processor Core and Hadware IP

- ✓ Based on ARM7TDMI Coprocessor Interface
  - ✓ Signal Interface
    for Handshake Protocol
  - ✓ Instruction Interface
    for Data Processing and Transferring

Signal Interface

name	meaning	direction
nCPI	Not CoProcessor Instruction	Processor -> CoProcessor
CPA	CoProcessor Absent	CoProcessor -> Processor
CPB	CoProcessor Busy	CoProcesosr -> Processor

* Here, CoProcessor = Hardware IP

Instruction Interface

Hardware-IP-Instructions:

- ✓ CDP (CoProcessor Data Operation)
  Operate data in the Hardware IP

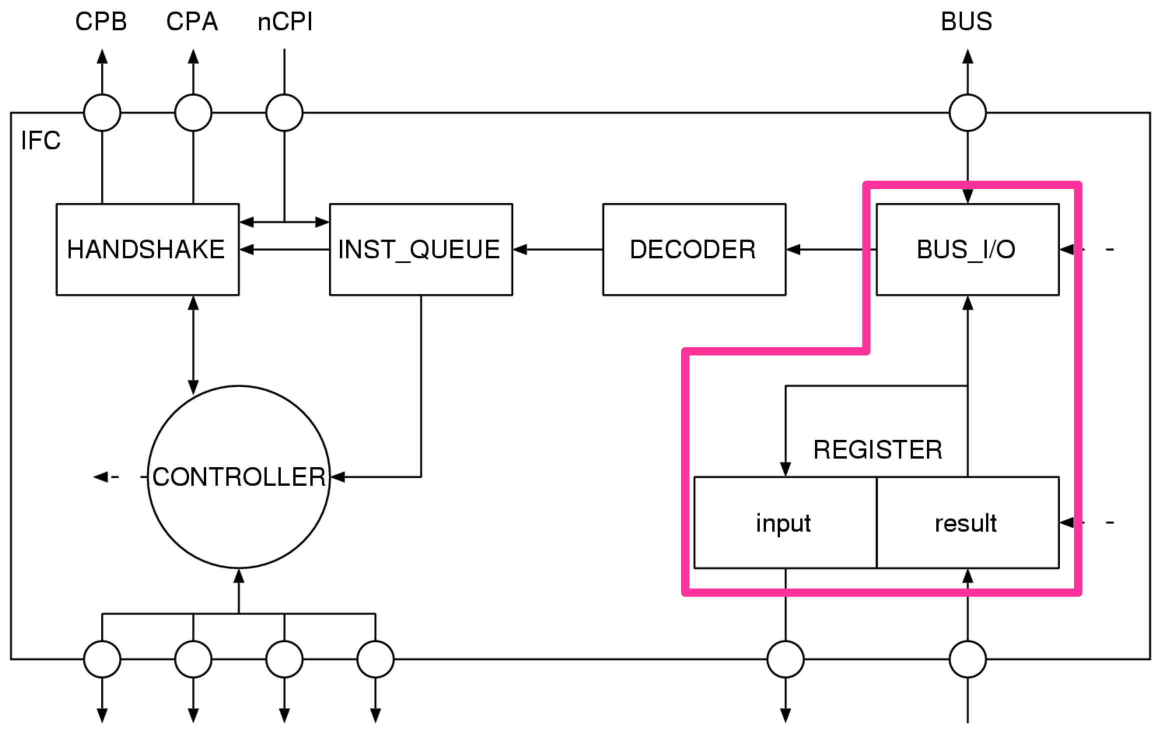
- ✓ LDC/STC (CoProcessor Load/Store Operation)
  Transfer data between IP and memory

- ✓ MRC/MCR (Register Transfer Operation)
  Transfer data between IP and processor core

IFC_Synthesizer

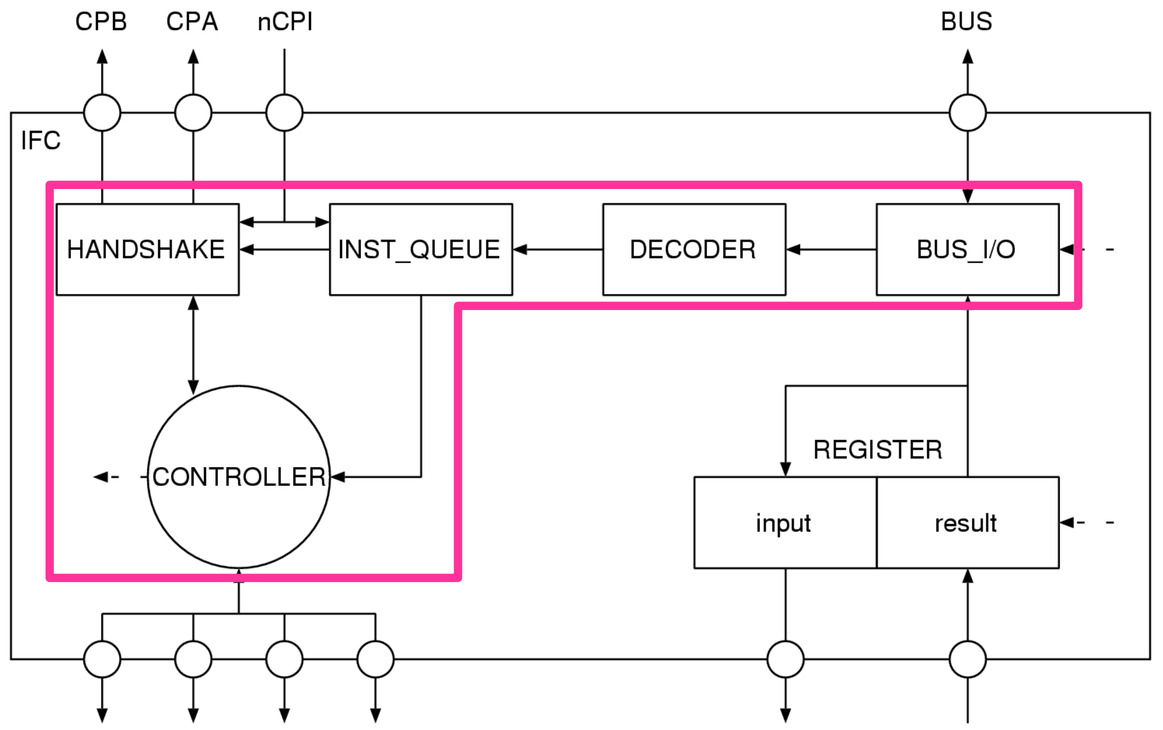
- ✓ IFC Architecture
- ✓ IFC Synthesis Method

IFC Architecture – transferring data



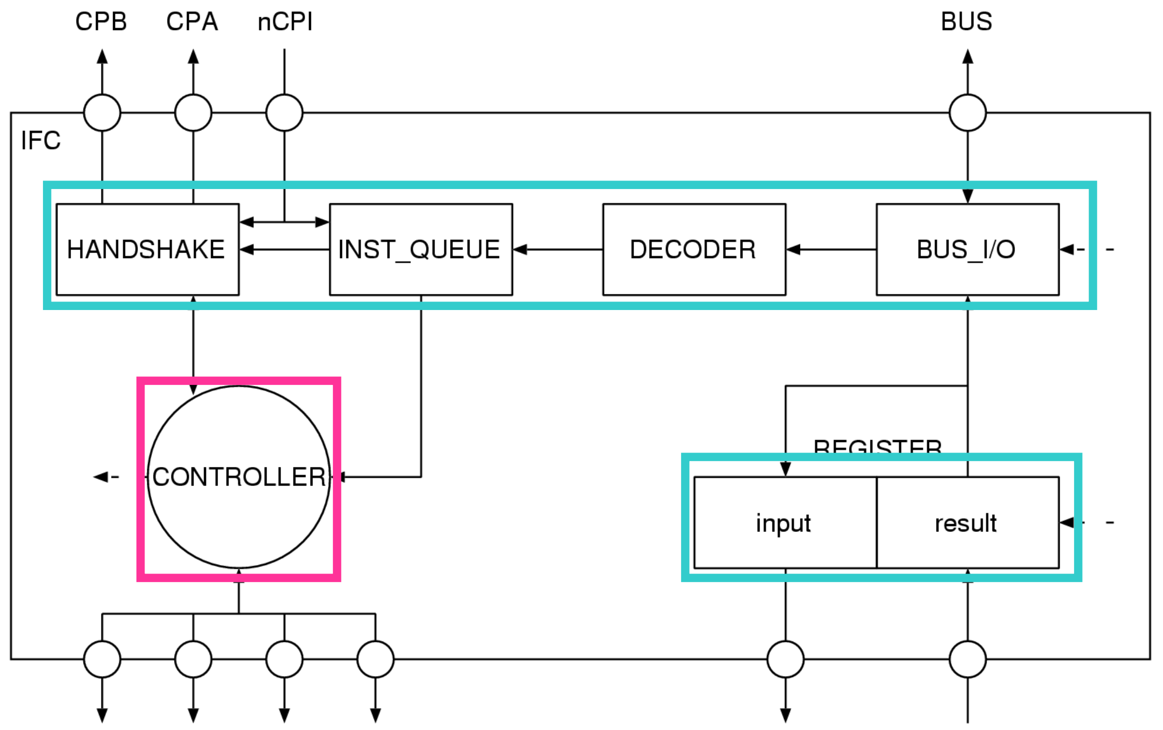
- ✓ BUS_I/O: controlling data flow via shared bus
- ✓ REGISTER: Saving data from / to a shared memory

IFC Architecture – control



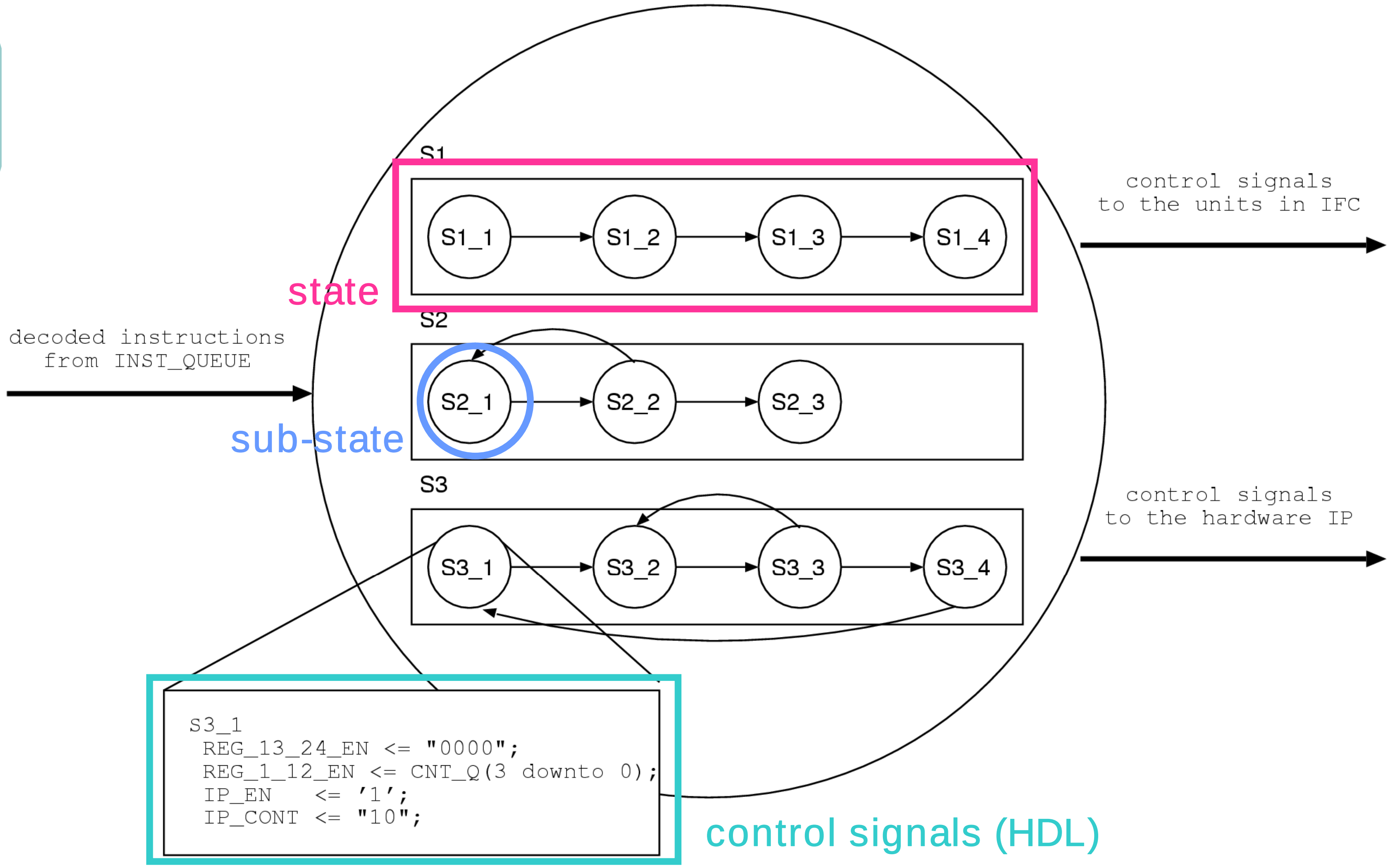
- ✓ DECODER: Decoding Hardware-IP-Instructions
- ✓ INST_QUEUE: Preserving decoded bit vectors
- ✓ HANDSHAKE: Handling handshake protocol
- ✓ CONTROLLER: Controlling all units in IFC with control signals

IFC Synthesis Method



- ✓ Synthesizing CONTROLLER is essential
- ✓ Refer to the paper about the ohters

CONTROLLER


S1
S1_1
S1_2
S1_3
S1_4
state
S2
S2_1
S2_2
S2_3
sub-state
S3
S3_1
S3_2
S3_3
S3_4
decoded instructions from INST_QUEUE
control signals to the units in IFC
control signals to the hardware IP
control signals (HDL)

```
S3_1
 REG_13_24_EN <= "0000";
 REG_1_12_EN <= CNT_Q(3 downto 0);
 IP_EN    <= '1';
 IP_CONT <= "10";
```

CONTROLLER Synthesis Algorithm

1. Ports Decision

    External and internal ports in IFC are decided

2. States Decision

    States for processing and transferring data are decided

3. Sub-states Decision

    Sub-states, which define control signals to all the units, are decided

4. Sub-state Transitions Decision

    Transitions among sub-states are decided

Step1: Ports Decision

```
port:
  input.clock       CLK;
  input.enable      EN;
  input.control[1:0] CONT;
  input.data[7:0]        ADR;
  output.data[31:0]  DATA;
endport
alphabet:
  signalset a = {CLK, EN, CONT, ADR, DATA};
            I:  {R,  1,  2'b01, x,   Z   };
            N:  {R,  1,  2'b00, x,   Z   };
        R(Xa):  {R,  1,  2'b10, Xa,  Z   };
        O(Xd):  {R,  1,  2'b11, x,   Xd  };
  endsignalset
endalphabet
word:
  proc(Xa,Xd): (R(Xa) N[2])[1,2] O(Xd)[3];
endword
```

Harware IP CWL

```
BUS_IO_S:       out std_logic;
HANDSHAKE_RUN: out std_logic;
HANDSHAKE_TR:   out std_logic;
REG_EN:         out std_logic;
IP_EN:          out std_logic;
IP_CONT:        out
   std_logic_vector(1 downto 0);
```

IFC HDL

- ✓ Focusing on "port" section in CWL
- ✓ Deciding ports in HDL

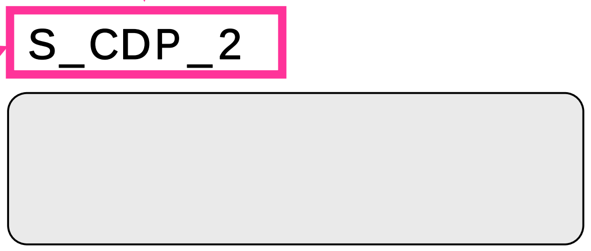
Step2: States Decision

```
port:
  input.clock       CLK;
  input.enable      EN;
  input.control[1:0] CONT;
  input.data[7:0]        ADR;
  output.data[31:0]  DATA;
endport
alphabet:
  signalset a = {CLK, EN, CONT, ADR, DATA};
            I:  {R,  1,  2'b01, x,   Z   };
            N:  {R,  1,  2'b00, x,   Z   };
        R(Xa):  {R,  1,  2'b10, Xa,  Z   };
        O(Xd):  {R,  1,  2'b11, x,   Xd  };
  endsignalset
endalphabet
word:
  proc(Xa,Xd): (R(Xa) N[2])[1,2] O(Xd)[3];
endword
```

Harware IP CWL

```
CDP 1, 1
CDP 1, 2
LDC 1, 16, …
STC 1, 16, …
…
```

Hardware-IP-Instruction set



- ✓ Deciding "state" from Hardware-IP-Instruction set
- ✓ In this case, the word "proc" in CWL correspond with the instruction "CDP 1 2"

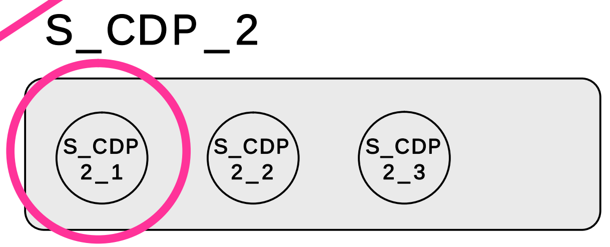
Step3: Sub-States Decision

```
port:
  input.clock        CLK;
  input.enable       EN;
  input.control[1:0] CONT;
  input.data[7:0]        ADR;
  output.data[31:0]  DATA;
endport
alphabet:
  signalset a = {CLK, EN, CONT, ADR, DATA};
            I:  {R,  1,  2'b01, x,   Z   };
            N:  {R,  1,  2'b00, x,   Z   };
        R(Xa)   {R,  1,  2'b10, Xa,  Z   };
        O(Xd):  {R,  1,  2'b11, x,   Xd  };
  endsignalset
endalphabet
word:
  proc(Xa,Xd): (R(Xa) N[2])[1,2] O(Xd)[3];
endword
```

Harware IP CWL

```
if CURRENT_STATE = S_CDP_2_1 then
  BUS_IO_S <= '0';
  HANDSHAKE_RUN <= '1';
  HANDSHAKE_TR <= '0';
  REG_EN <= CNT_Q(3 downto 0);
  IP_EN <= '1';
  IP_CONT <= "10";
```

IFC HDL



- ✓ Deciding "sub-states" from "word" section in CWL
- ✓ Deciding control signals every sub-state from "alphabet" section in CWL

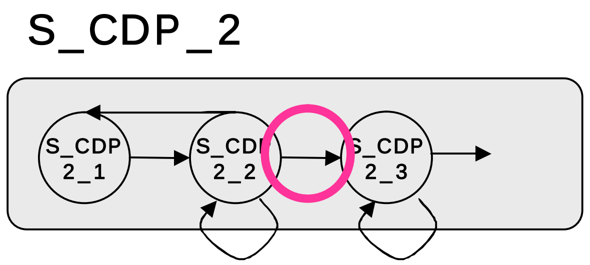
Step4:
Sub-States Transitions Decision

```
port:
  input.clock        CLK;
  input.enable       EN;
  input.control[1:0] CONT;
  input.data[7:0]        ADR;
  output.data[31:0]  DATA;
endport
alphabet:
  signalset a = {CLK, EN, CONT, ADR, DATA};
            I:  {R,  1,  2'b01, x,   Z   };
            N:  {R,  1,  2'b00, x,   Z   };
        R(Xa):  {R,  1,  2'b10, Xa,  Z   };
        O(Xd):  {R,  1,  2'b11, x,   Xd  };
  endsignalset
endalphabet
word:
  proc(Xa,Xd): (R(Xa) N[2])[1,2] O(Xd)[3];
endword
```

R, N, N, R, N, N, O, O, O

Harware IP CWL

```
elsif CURRENT_STATE = S_CDP_2_2 then
  if CNT_Q = 3 then
    NEXT_STATE <= S_CDP_2_1;
  elsif CNT_Q = 6 then
    NEXT_STATE <= S_CDP_2_3;
  else
    NEXT_STATE <= S_CDP_2_2;
  end if;
```

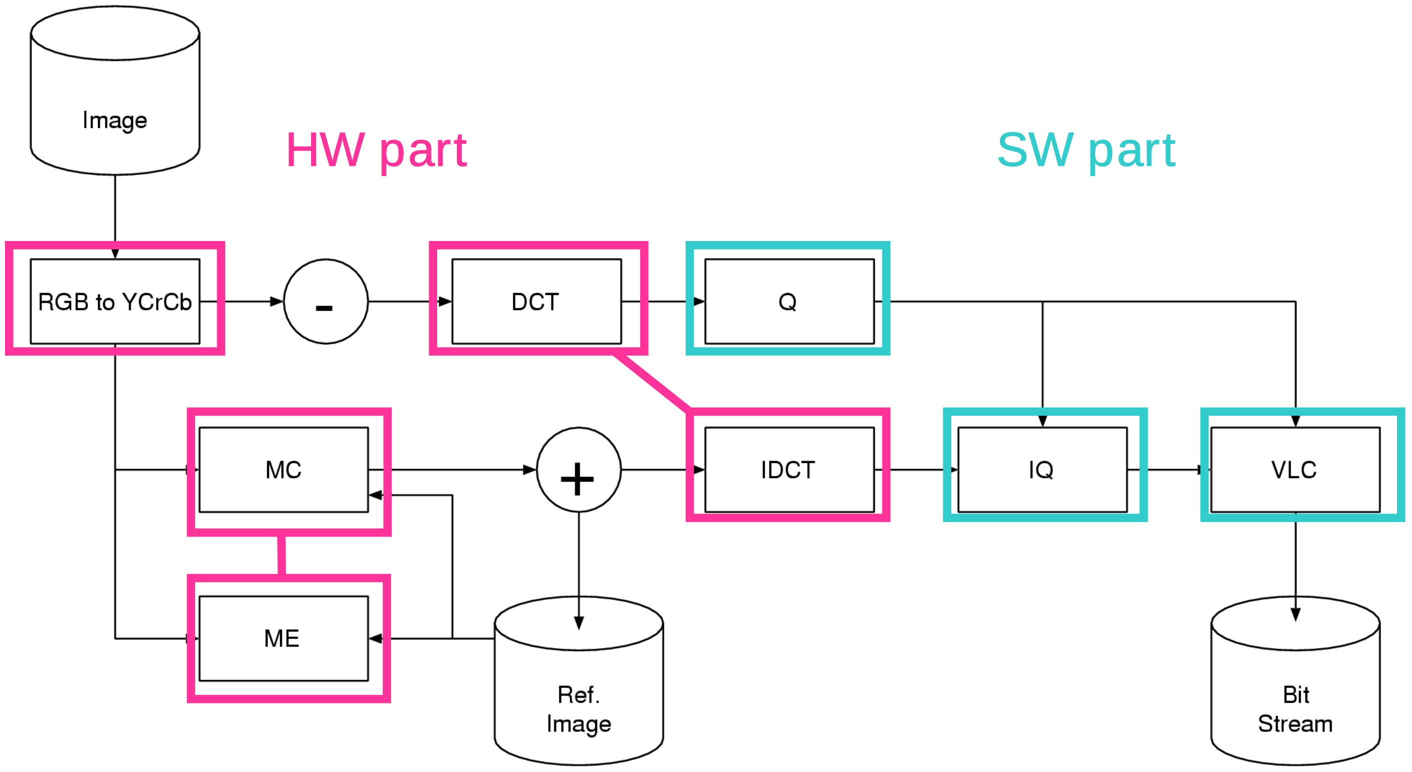


- ✓ Deciding sequences of sub-states from “word” section in CWL

Experimental Results

Target Application

- ✓ MPEG-4 Encoder



Using Hardware IPs

Function	Area [mm^2]
RGB to YCrCb	0.3904
DCT / IDCT	2.4480
ME / MC	3.6000

Configuration of Synthesized Processor Cores

Name	Area [mm^2]	Frequency [MHz]	Configurations			
			Kernel	Issue	#ALUs	#Regs
A	5.9723	81.300	RISC	4	ALU x 2 MUL x 2	47
B	1.7554	70.225	DSP	2	ALU x 1 MUL x 1	8

Results (1)

function	Processor Core	IFC area [mm^2]
RGB to YCrCb	A	0.1080
RGB to YCrCb	B	0.1148
DCT/IDCT	A	0.1028
DCT/IDCT	B	0.1108
ME/MC	A	0.1547
ME/MC	B	0.1638

Results (2)

- ✓ IFC_Synthesizer
  - ✓ Implemented in Ruby Language
  - ✓ Executed on Linux 2.4, Pentium III 500MHz, RAM 192MB
- ✓ Execution time of IFC_Synthesizer
  - ✓ Max: 9.4 [sec]
  - ✓ Min: 4.3 [sec]

↔ Manual design: about 3 days

IFC_Synthesizer reduces the cost of designing an interface circuit

Conclusion

Conclusion

- ✓ Our proposal:
  - ✓ IFC Architecture
  - ✓ IFC Synthesis method
- ✓ IFC_Synthesizer reduce IFC development cost
  - ✓ Execution time ... less than 10 [sec]
  - ✓ Manual design ... about 3 [days]
- ✓ Future Work
  - ✓ Clock Gating for Low Energy Consumption

Thank you

Why IFCs are need?
IFCs should be in a synthesized processor.

- ✓ Result in a same thing (if IFCs are in a synthesized processor core).
- ✓ Since hardware IPs act parallely, IFCs are required independently every hardware IP.

Why CWL is adopted?

- ✓ Interface Language is required for Hardware IP database.
- ✓ CWL is based on a regular expression, so we can describe wave form simply.
- ✓ CWL parser (XML converter) has been prepared.

Why XML converter is need?

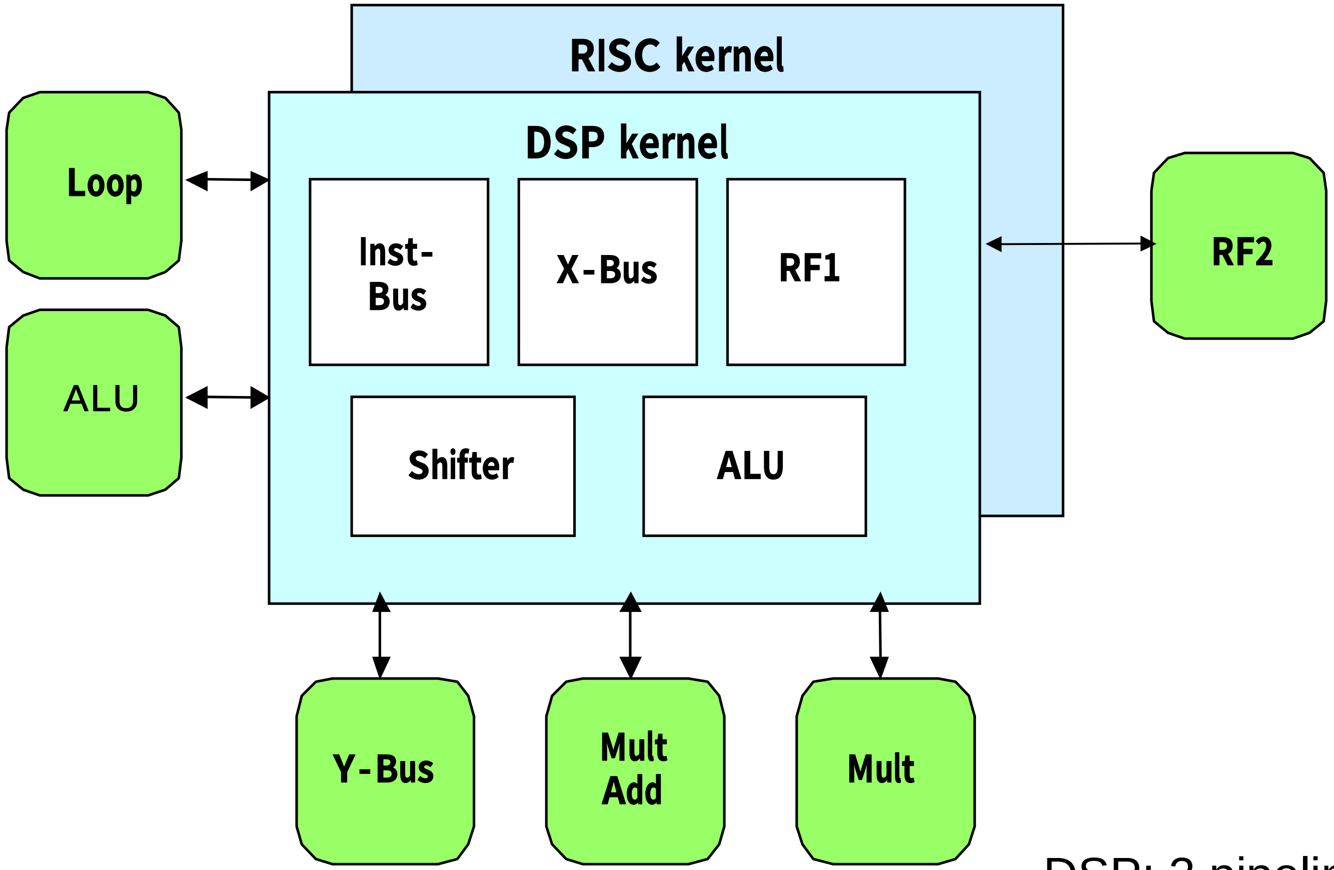
- ✓ For parsing.

What is Ruby Language?
Why Ruby is adopted?

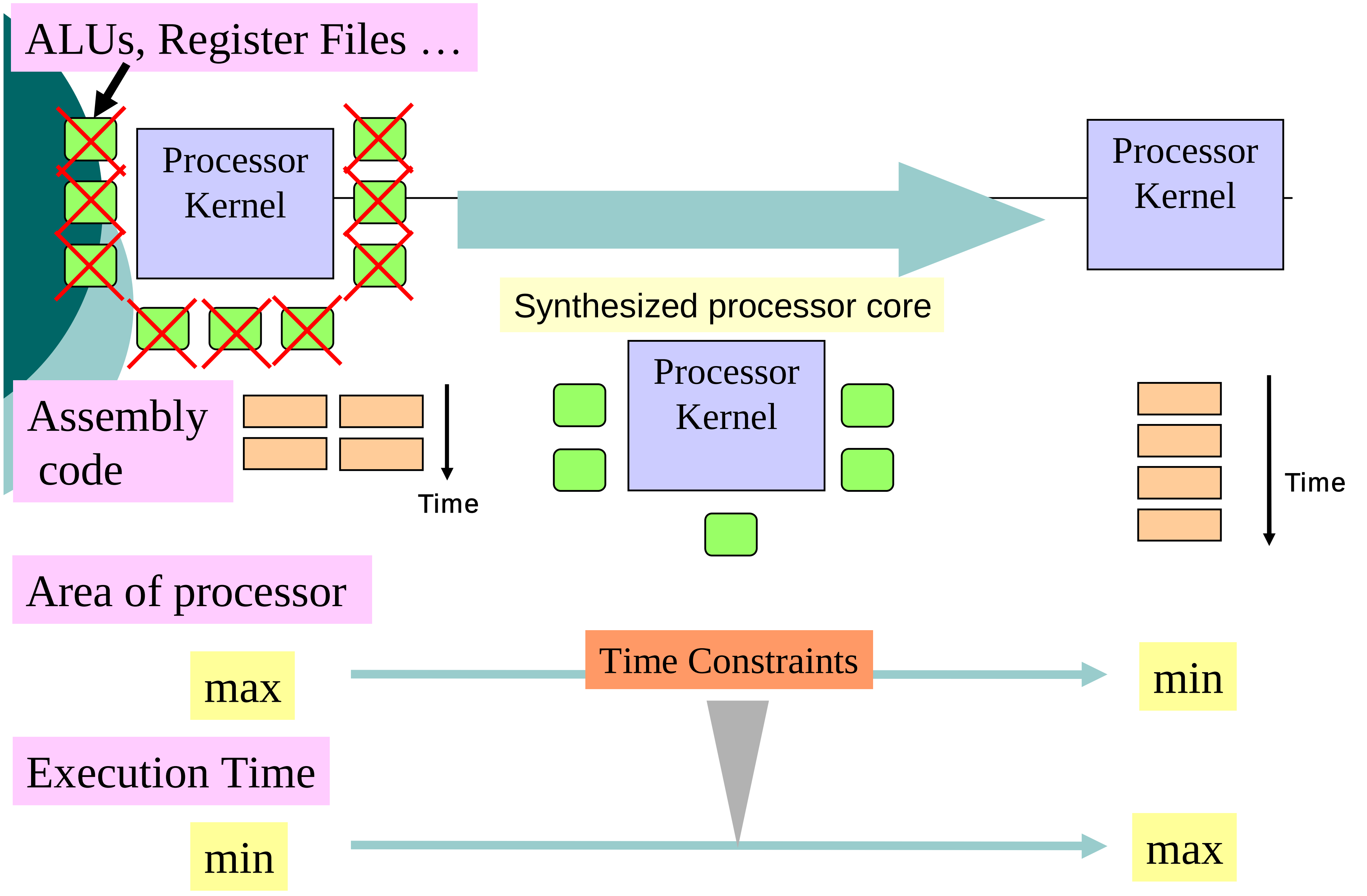
- ✓ Ruby is Object Oriented Script Language.
- ✓ http://www.ruby-lang.org

- ✓ IFC_Synthesizer need not have high performance.
- ✓ Of cource, other language can be adopted.

Architecture of Processor Core

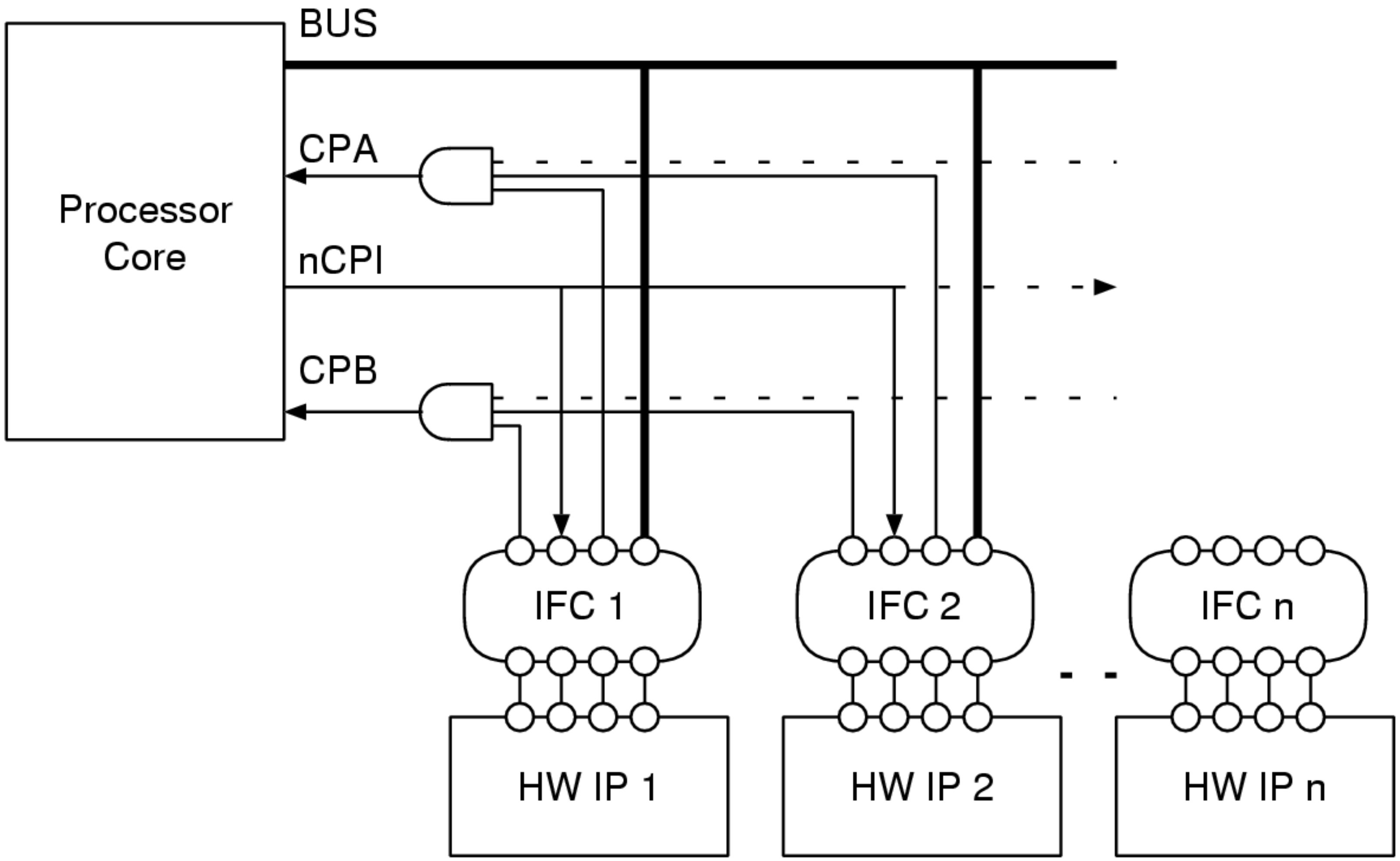


DSP: 3 pipeline stages
RISC: 5 pipeline stages

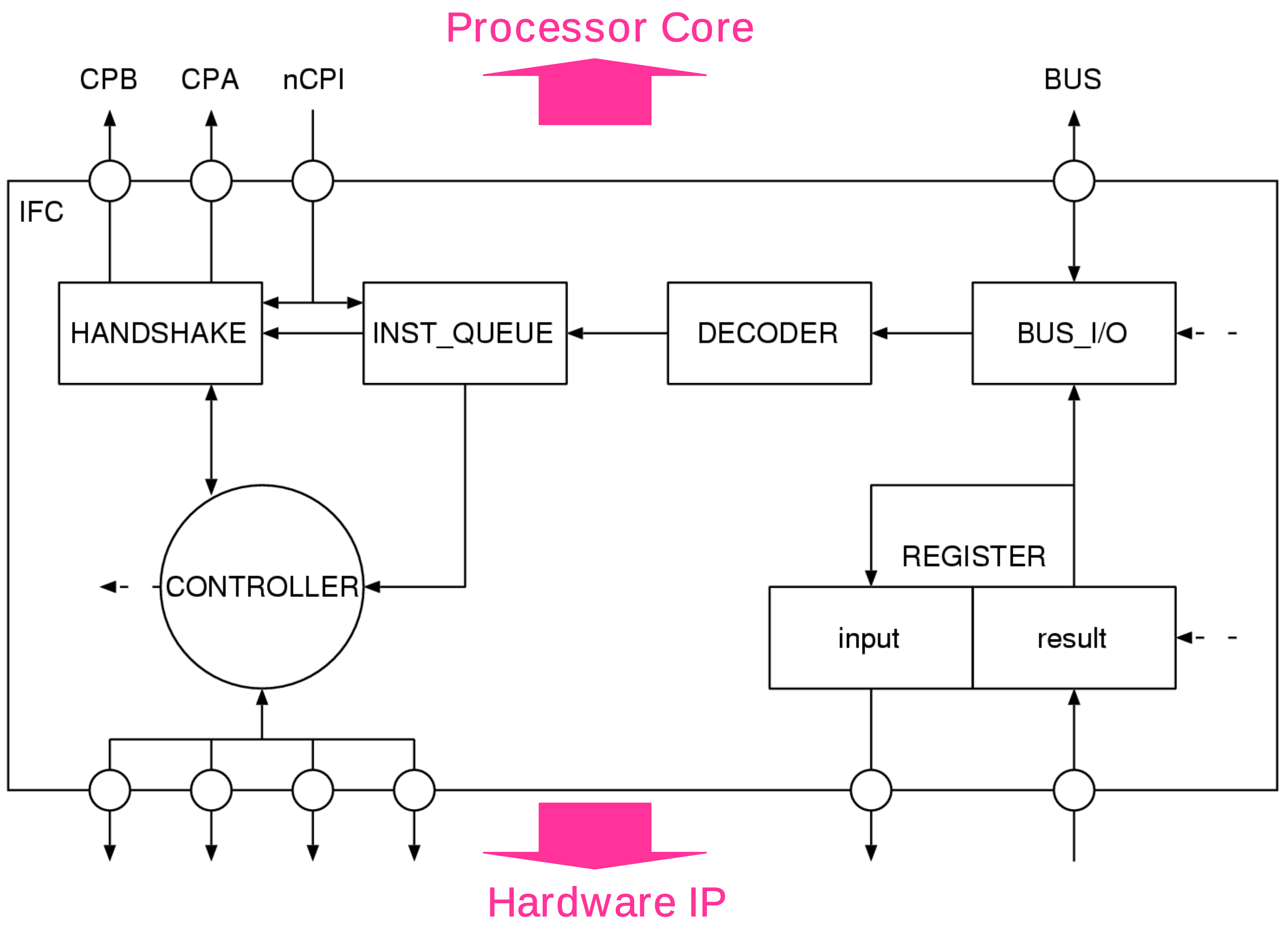


[1] N. Togawa, M. Yanagisawa and T. Ohtsuki, ``A hardware/software co-synthesis system for digital signal processor core,''IEICE Trans. on Fundamentals, vol. E82-A, no.11, 1999.

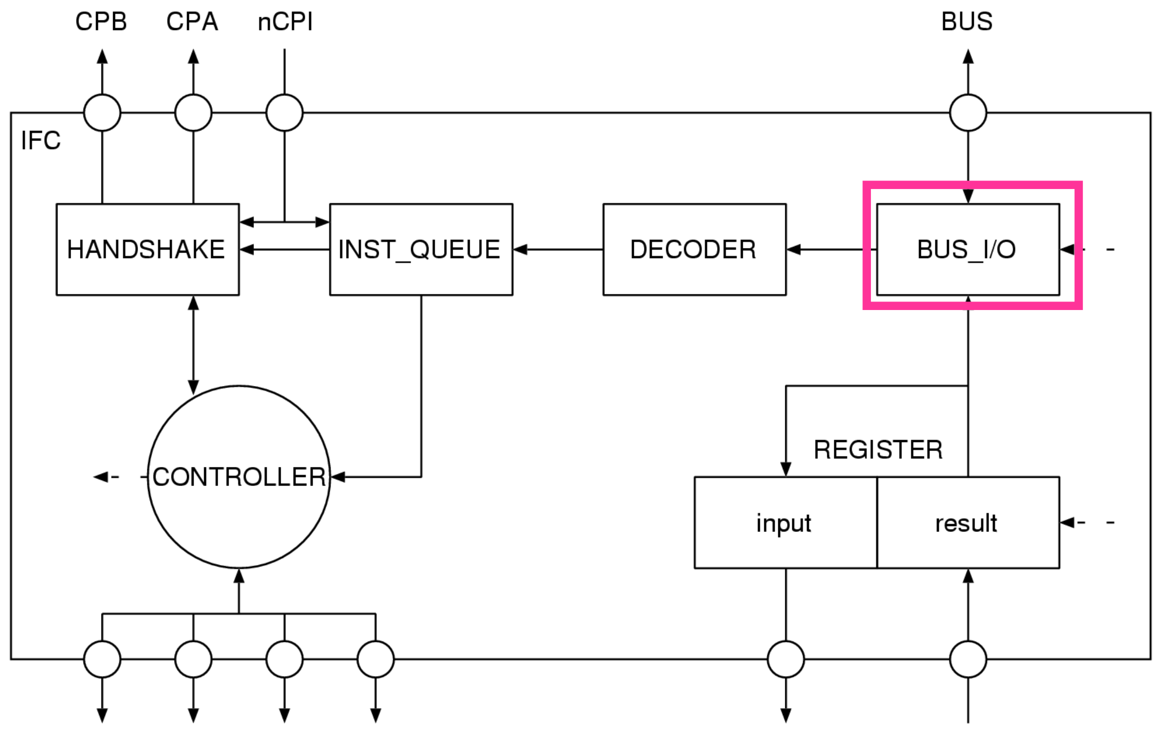
Connections



IFC Architecture

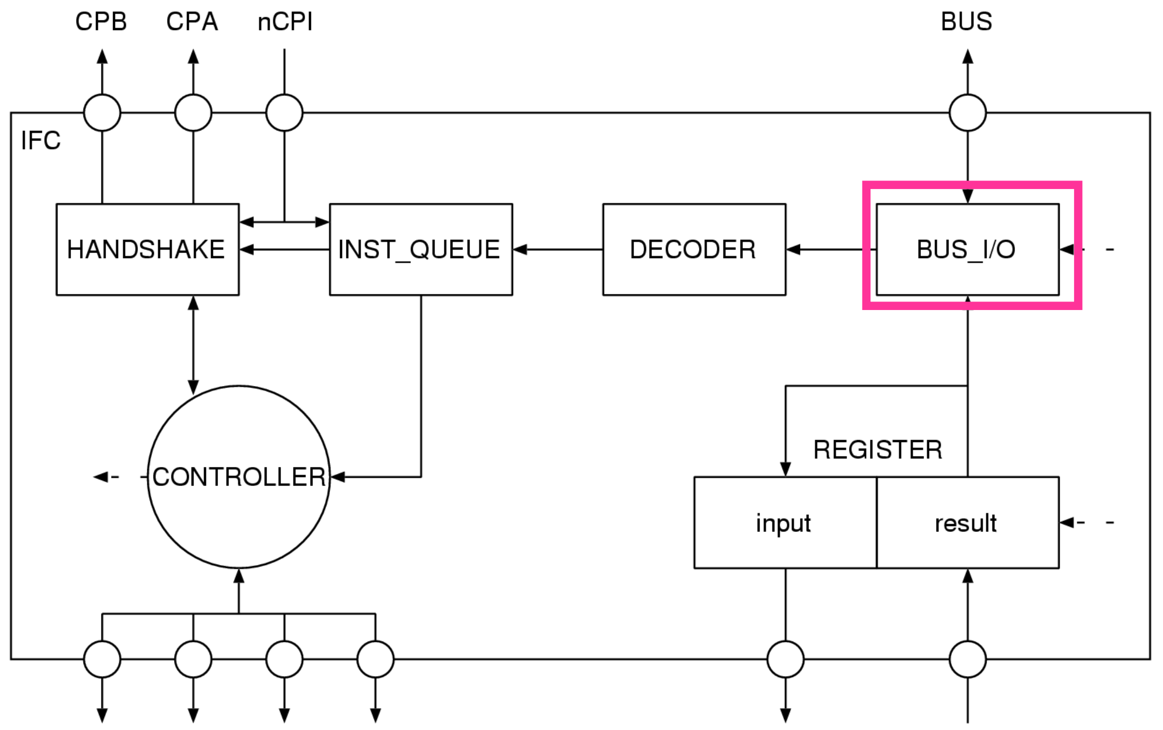


IFC Synthesis Method – BUS_I/O



- ✓ Depending on a bit length of a shared bus
- ✓ Independent of an using hardware IP

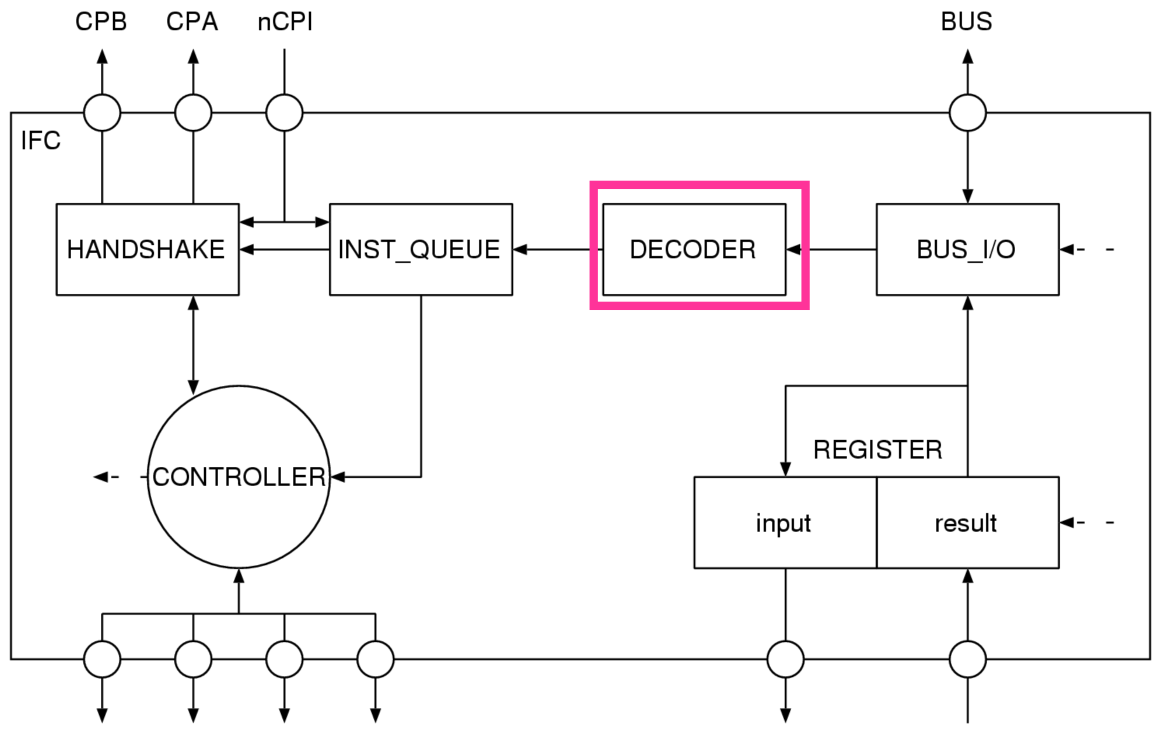
IFC Architecture – BUS_I/O



✓ controlling data flow via shared bus:

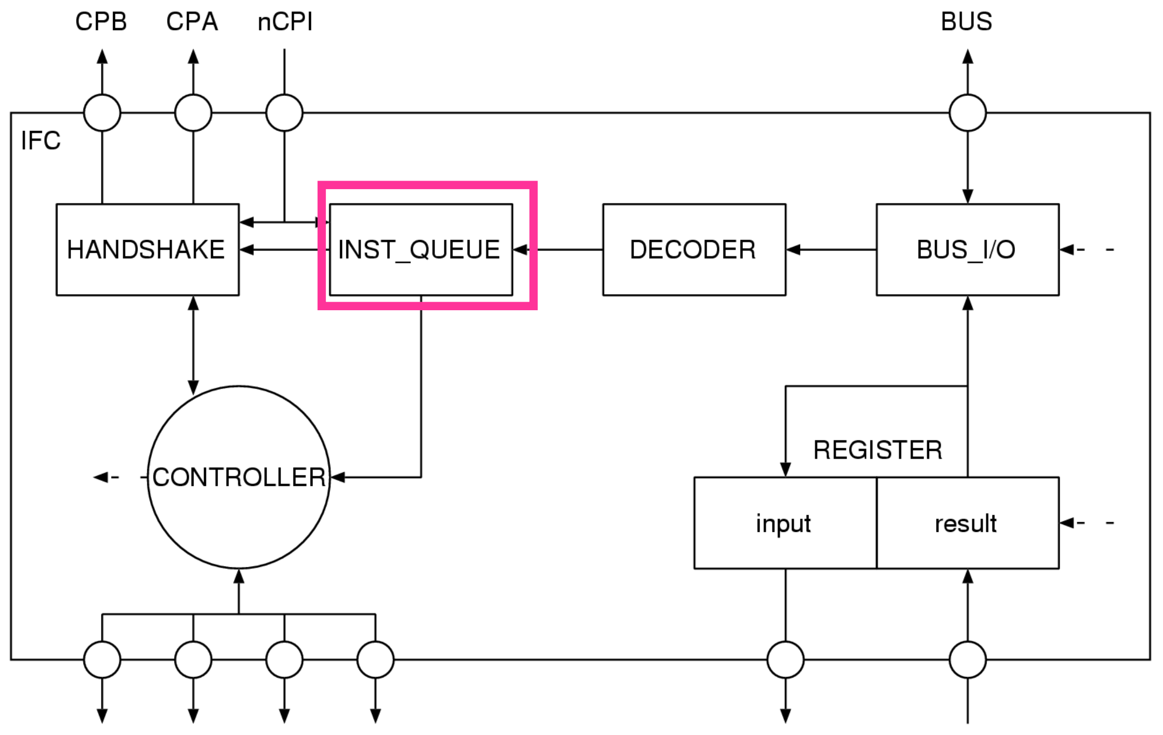
1. Hardware-IP-Instructions to DECODER
2. Input Data from BUS to REGISTER
3. Output Data from REGISTER to BUS

IFC Architecture – DECODER



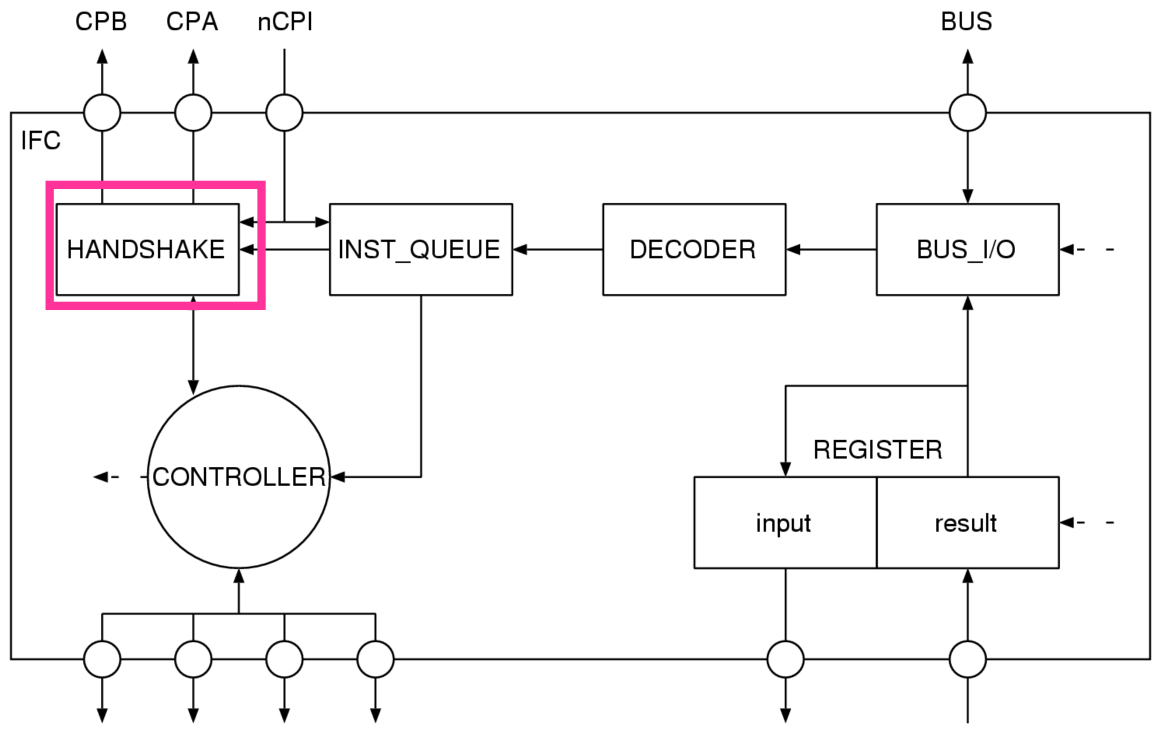
- ✓ Decoding Hardware-IP-Instructions
- ✓ Queuing decoded bit vector into INST_QUEUE

IFC Architecture – INST_QUEUE



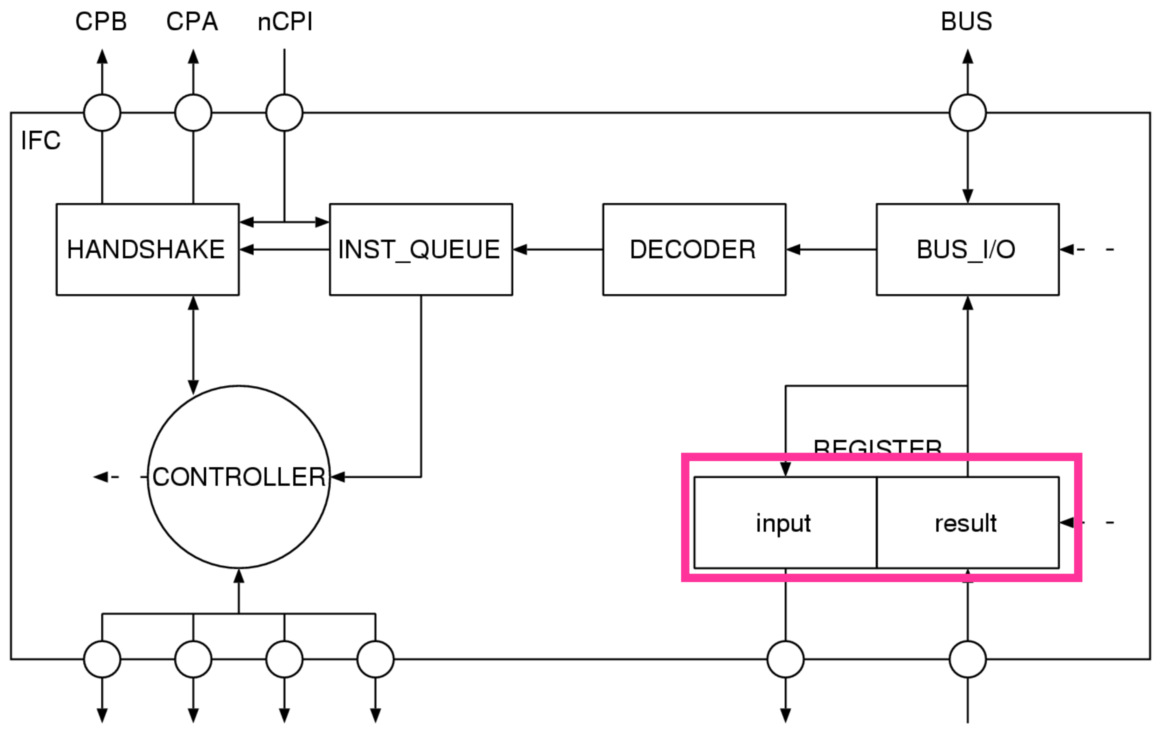
- ✓ Preserving decoded bit vectors
- ✓ Dequeuing them into CONTROLLER

IFC Architecture – HANDSHAKE



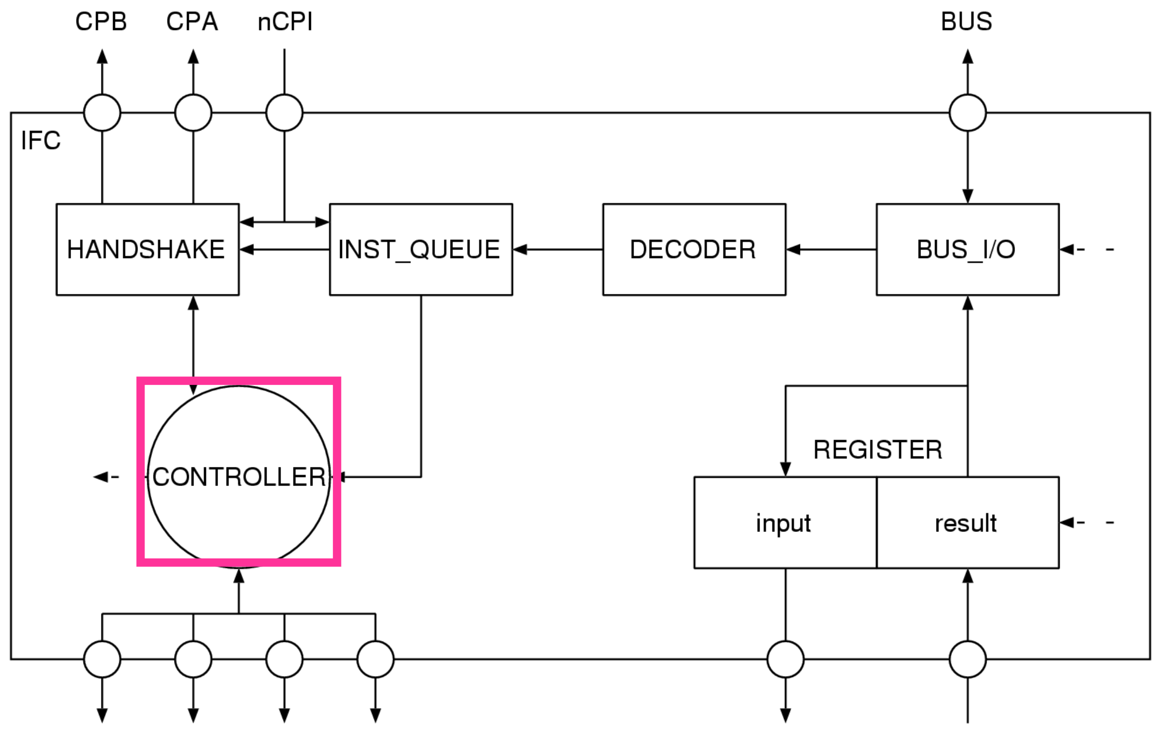
- ✓ Interface for handshake signals (nCPI, CPA, CPB)
- ✓ Handling handshake protocol
- ✓ Communication with CONTROLLER

IFC Architecture – REGISTER



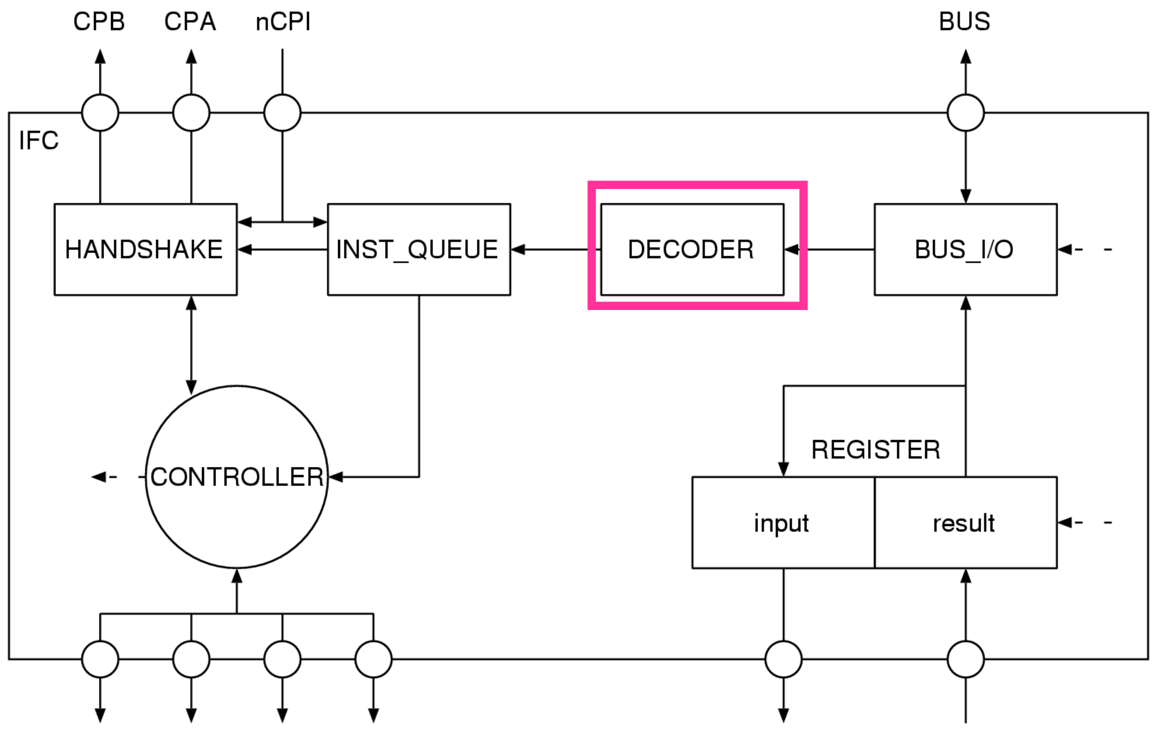
- ✓ (input REGISTER) Saving data from a shared memory
- ✓ (result REGISTER) Saving data from the hardware IP

IFC Architecture – CONTROLLER



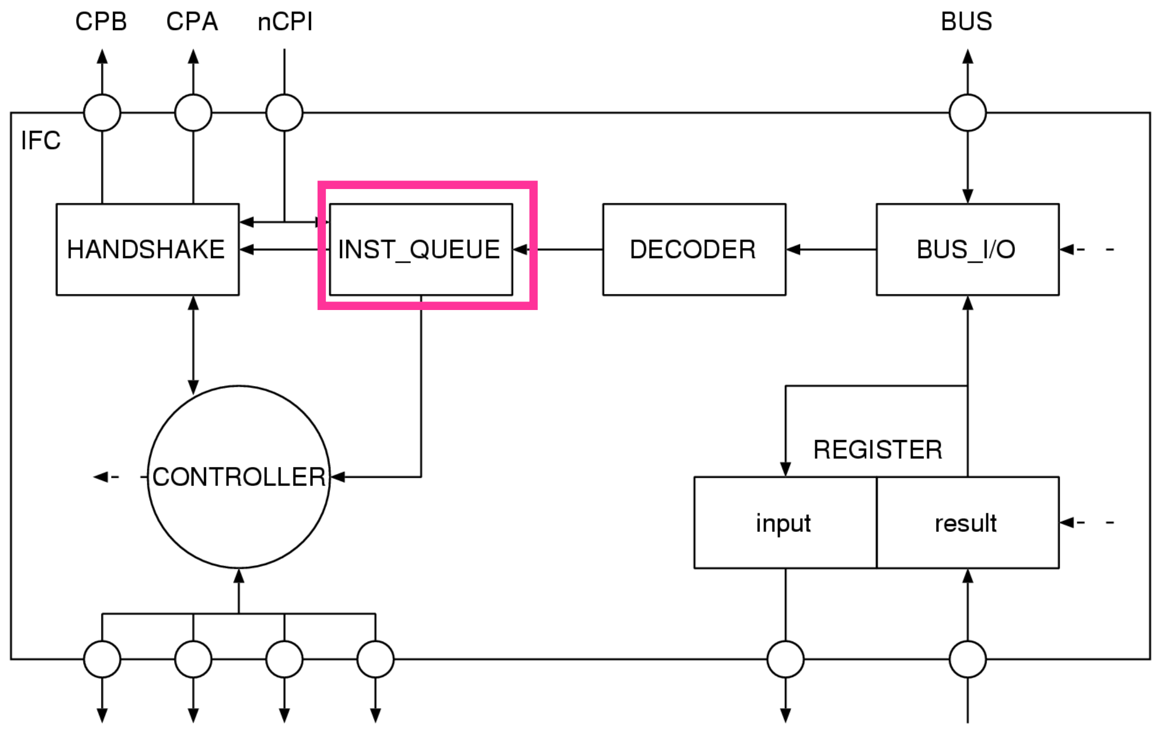
- ✓ Controlling all units in IFC with control signals
- ✓ Controlling the hardware IP for processing data
- ✓ See below for further details

IFC Synthesis Method – DECODER



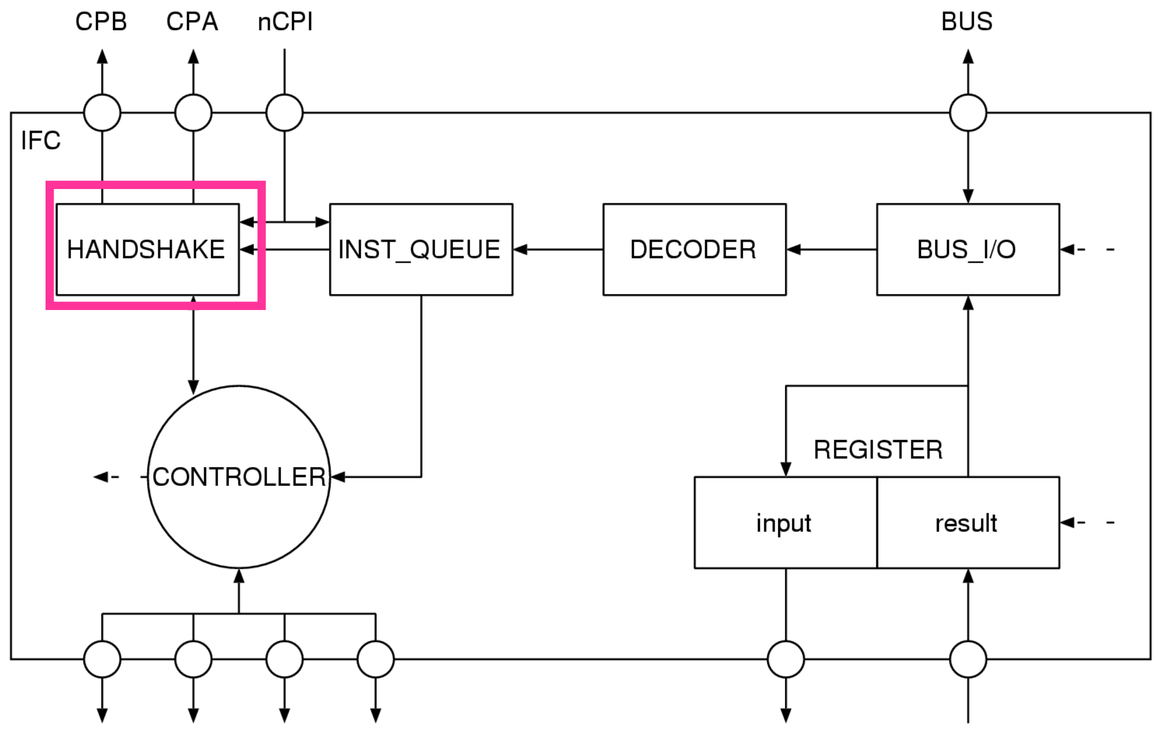
- ✓ Depending on a synthesized processor core.
  … Instruction encoding specification
- ✓ Independent of an using hardware IP

IFC Synthesis Method – INST_QUEUE



- ✓ Depending on a synthesized processor core.
  ... Pipeline stages
- ✓ Independent of an using hardware IP

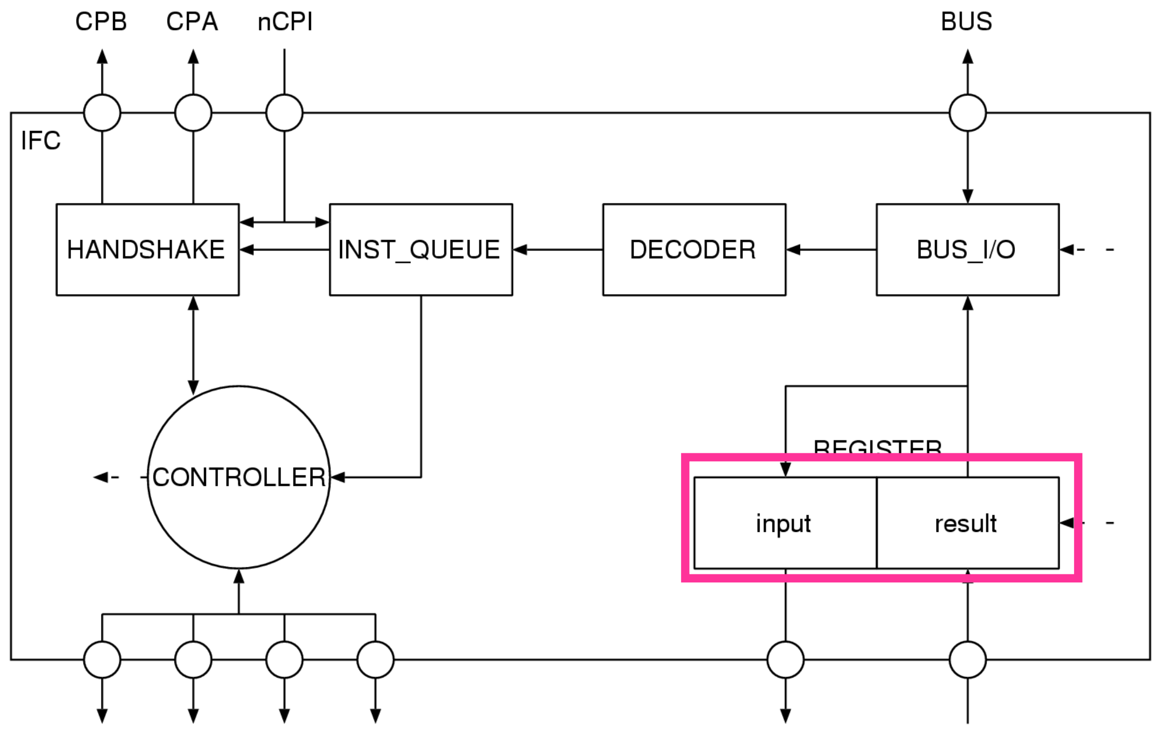
IFC Synthesis Method – HANDSHAKE



- ✓ Fixed

  ... Handshake protocol has been defined.

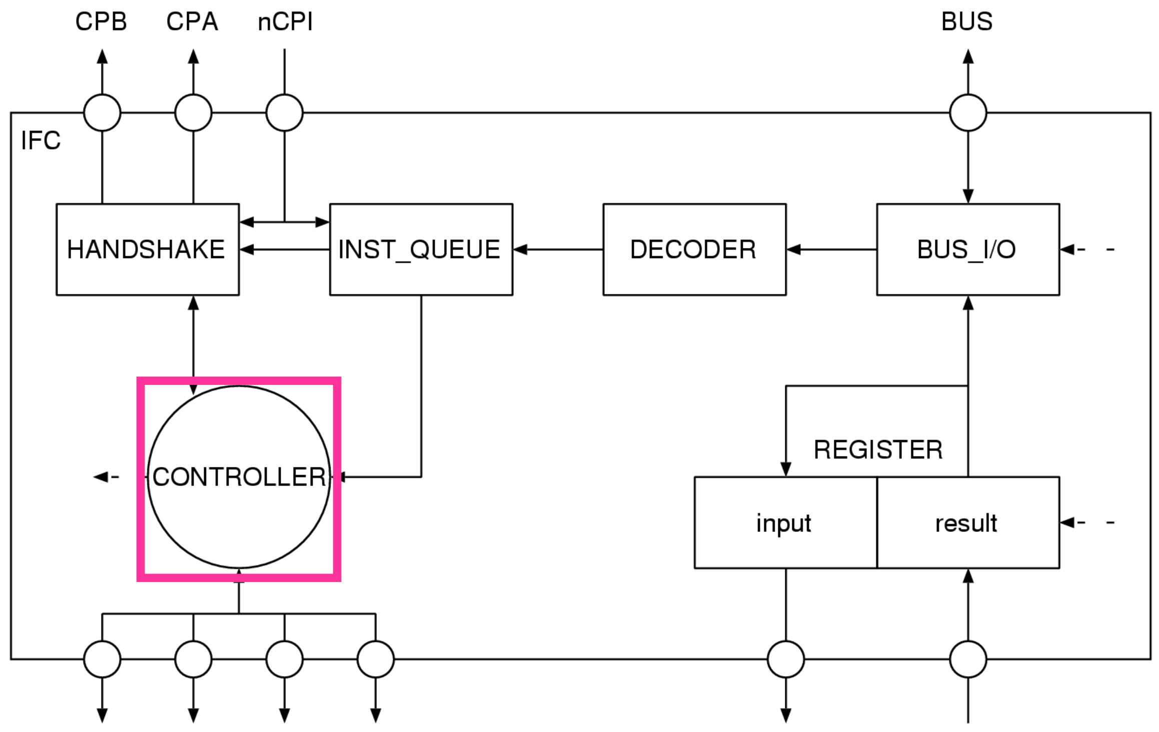
IFC Synthesis Method – REGISTER



✓ The size of registers is given by processor core synthesis system

… Hardware-IP-instructions used in Software include the length of transferring data

IFC Synthesis Method – CONTROLLER



✓ Next slides ...

Using Hadware IPs

function	area [mm^2]
RGB to YCrCb	0.3904
DCT / IDCT	2.4480
ME / MC	3.6000

* Hitachi 0.35 um CMOS

Configuration of Synthesized Processor Cores

Name	Area [mm^2]	Frequency [MHz]	Configurations			
			Kernel	Issue	#ALUs	#Regs
A	5.9723	81.300	RISC	4	ALU x 2 MUL x 2	47
B	1.7554	70.225	DSP	2	ALU x 1 MUL x 1	8

[13] N. Togawa et al, IEICE Trans. Fundamentals, vol. E82-A, no.11 1999