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

Shunitsu Kohara† Naoki Tomono†‡ Jumpei Uchida† Yuichiro Miyaoka†,* Nozomu Togawa† Masao Yanagisawa† Tatsuo Ohtsuki†

† Department of Computer Science, Waseda University
‡ Presently, the author is with the Toyota
* Presently, the author is with the Toshiba
E-mail: kohara@yanagi.comm.waseda.ac.jp

Abstract—In SoC designs, efficient communication between the hardware IPs and the on-chip processor becomes very important, however the interface is usually affected by the processor core specification. Thus in this paper, we focus on developing an efficient interface circuit architecture for the communications between the on-chip processor and embedded hardware IP cores. We also propose a method to synthesize it. Experimental results show that our method could obtain optimal interface circuits and works well through designing a MPEG-4 encode application.

I. Introduction

The growing demand for hardware/software systems, together with the ability to put the entire system on a single chip using deep sub-micron technologies, has led to the evolution of complex hardware/software system-on-chips (SoCs). While the complexity of SoCs increases, so does the demand to reduce their time-to-market. Typically, IP-based SoC design contains the following steps such as application specification, hardware/software partition and hardware/software integration. Though the design time of SoCs can be greatly reduced by efficient re-use of intellectual property (IP) cores, how to develop an efficient interface circuit between the hardware IPs and the on-chip processor becomes an important task.

One of the solutions is to generate the ones automatically [8]. Works in this approach include [10, 7, 5, 15]. In [10], an arbiter consists of protocol conversion FSM and FIFOs to regulate transfers and mismatched protocols are mapped into a standard communication scheme. In [7], regular expression is used to describe protocol and the interface is generated from protocol conversion FSM. In [5], communication protocol of IP is described as FSM, and a protocol translation algorithm is proposed, which derives an interface FSM between two IPs. In [15], for saving the complexity in design space exploration, parameterized templates are used to synthesize a hardware.

On the other hand, to integrate a processor core into SoC, the practical method is not to use a processor core, but use a configurable one so as to satisfy the performance requirements and the area constraints. When SoC application is implemented on a configurable processor core, such as [12, 13], and several hardware IPs, generating the interface between the processor core and the hardware IPs requires: (1) to communicate with the hardware IP (2) to communicate with the configurable processor core.

In the literature, most of the previous works assumes that both of connected IPs have same model for interface description. However most of the used IPs are not standard IPs, if one side of them is a configurable processor core, the interface is affected by the processor core specification, such as instruction set, pipeline stages.

In this paper, we propose an architecture of the interface circuit to communicate with a configurable processor core and a hardware IP. We also propose a method for synthesizing the one. The models of the interface in previous works are based on FSM and so on. We use architecture templates of the interface circuit. Our proposed architecture and method enable us to obtain optimal interface circuits.

This paper is organized as follows. Section II describes IP-based SoC design method and target architecture. Section III proposes an architecture of an interface circuit (IFC) and a method for synthesizing the one (IFC_Synthesizer). Section IV shows the experimental results with the proposed method through a MPEG-4 encoder application. Section V gives the concluding remarks.

II. IP-BASED SoC DESIGN METHOD

In this section we describe IP-based SoC design method and define target architecture.

A. Design Method

Figure 1 shows a design method with an interface circuit synthesizer "IFC_Synthesizer". A designer searches hardware IPs for hardware parts from a hardware IP database.
and uses a processor core synthesis system for software parts. The interface description of using hardware IP and instruction set for hardware IP generated by the processor core synthesis system are inputs for IFC_Synthesizer. The output of IFC_Synthesizer is an interface circuit (IFC), which communicates with the processor core and the hardware IP.

### B. Architecture Model

Figure 2 shows the architecture model of the target SoC. The architecture consists of a processor core, a memory and several hardware IPs which are connected with each other via a shared bus. In our approach, first, the input application is partitioned into hardware/software parts, then the hardware parts are implemented with hardware IPs, and the software parts are implemented on a processor core.

#### B.1 Processor Core

The processor core is configurable. The configurable parameters include an instruction set, pipeline stages, hardware units such as ALU, multiplier, register files and so on.

The instruction set of the processor core includes hardware-IP-instructions. The hardware-IP-instructions are described in Subsect. A.2.

#### B.2 Hardware IPs

Hardware IPs distributed in the market have various architecture and interface. In the target architecture, software on the processor core controls the hardware IPs with hardware-IP-instructions. To avoid bus conflicts, hardware IPs should not have data transfer unit but data-path for processing data.

We make a premise that in our work all the target hardware-IPs in Hardware-IP Database have an interface description in CWL [4]. IFC_Synthesizer synthesizes IFC from CWL description of the target hardware IP.

#### B.3 Memory

The memory in the target architecture is a simple model like SRAM.

## III. IFC_Synthesizer

In this section we propose an IFC architecture and a method for synthesizing it. In our work, the synthesizer is called as IFC_Synthesizer. In this section, we first illustrate the interface of processor core and hardware IPs, and then we propose an IFC architecture and an algorithm of IFC_Synthesizer, where IFC_Synthesizer is the name of the synthesizer developed in our work. Details will be explained in the followings.

### A. Interface

The interface between processor core and hardware IP is based on ARM Coprocessor Interface [3]. The ARM Coprocessor Interface defines a signal interface and an instruction interface.

#### A.1 Signal Interface

Figure 3 shows a connection of the processor core and hardware IPs. The processor core can connect up to 16 hardware IPs. The processor core communicates with hardware IPs with three handshake signals as follows:

- **nCPI** not CoProcessor Instruction (Processor Core → Hardware IP): A processor core wants to execute hardware-IP-instruction.
- **CPA** CoProcessor Absent (Hardware IP→Processor Core): There are no hardware IPs which can execute the hardware-IP-instruction.
- **CPB** CoProcessor Busy (Hardware IP→Processor Core): Hardware IP can not execute the hardware-IP-instruction immediately since it is executing another hardware-IP-instruction.
A.2 Instruction Interface

Processor core sends three types of hardware-IP-instructions: (a) CDP (processing data operations), (b) LDC/STC (transfer data operations from/to a shared memory) and (c) MCR/MRC (transfer data operations from/to a register in a processor core).

The format of hardware-IP-instruction is as follows:

CDP HW#, OP#
LDC HW#, N, Rd, Rn, offset
STC HW#, N, Rd, Rn, offset
MRC HW#, Rd1, Rd2
MCR HW#, Rd1, Rd2

CDP performs processing operation with a hardware IP. Each hardware IP is numbered. HW# is the number. A processor core operates a hardware IP to use HW#. When a hardware has several functions, each function is numbered. OP# is the number. The processor core use OP# to select the function.

LDC/STC transfer data between a hardware IP and a shared memory.

MCR/MRC transfer data between a processor core register and a hardware IP register.

B. IFC

In the model of the target SoC, the processor core controls hardware IPs with the interface described in Subsect. A. Since hardware IPs distributed in the market might be provided by different vendors, they do not always have the standard interface. So it will cause many problems, such as:

1. they can not communicate on handshake communication with the signal interface.
2. hardware IP can not decode hardware-IP-instructions by the processor core.
3. therefore processor core can not control hardware IPs.

IFC synthesized by IFC_Synthesizer communicates with the processor core at the proxy of the hardware IP. Figure 4 shows the architecture of IFC. Each of units is as follows:

IFC defines the mapping of the external and internal ports of IFC.

BUS_I/O controls input/output data flow via shared bus. It (1) inputs data from shared bus to REGISTER (2) or inputs hardware-IP-instructions from shared bus to DECODER (3) or outputs data from REGISTER to a shared bus.

DECODER decodes instructions from the processor core. If the instruction is a kind of hardware-IP-instructions and HW# field and OP# field are validate for target hardware IP, DECODER decodes the instruction and queues it into INST_QUEUE.

INST_QUEUE preserves bit vectors decoded by DECODER. If the target hardware IP is not busy, it dequeue first bit vector to CONTROLLER.

HANDSHAKE deals with handshake protocol with the signal interface. It is controlled with nCPI signal from processor core and control signals from CONTROLLER, and output CPA and CPB signals for the handshake communication.

REGISTER saves data from/to a shared memory and the target hardware IP. input REGISTER saves data before hardware IP processing, and result REGISTER saves data after hardware IP processing.

CONTROLLER controls all units in IFC with control signals and controls hardware IP for processing data. It consists of counter and state machine. The input is given from INST_QUEUE and HANDSHAKE. It is described in detail in Sect. C.3.

C. IFC_Synthesizer

IFC_Synthesizer synthesizes IFC HDL from the interface description of a hardware IP (Fig. 5). The interface description is written in CWL. To communicate with both of a processor core and a hardware IP, IFC must have the interface of them. The interface to a processor core described in Sect. A has been defined. Since the interface to a hardware IP depends on its own specification, IFC is synthesized for each of the using hardware IPs.
One of the input for IFC_Synthesizer is HDL template description of IFC. Not all the units are synthesized for each of the using hardware IPs. The units which need not vary are template description as they are.

In Fig. 4, BUS_I/O, HANDSHAKE are interfaces to a processor core. Since the interface to a processor core has been defined, they are independent of using hardware IP.

DECODER and INST_QUEUE are also interfaces to a processor core. In the hardware-IP-instructions, HW# indicates which hardware IP is a target one, and OP# indicates which function are a target one. Since HW# and OP# depend on hardware IPs, the HDL description of DECODER varies for them. The queue length of INST_QUEUE is correspond with pipeline stages of a processor core.

CONTROLLER is the interface to a hardware IP. Since the interface to a hardware IP depends on its own specification, the HDL description of CONTROLLER varies for the hardware IP.

REGISTER is the interface to a hardware IP. The register size is decided by the specification of the hardware IP.

Methods of synthesizing the units which varies for using hardware IPs are described following sections.

### C.1 DECODER

IFC_Synthesizer refers hardware-IP-instructions generated by compiler to synthesize DECODER. It preserves values of the HW# and OP# correspond with a target hardware IP. DECODER decodes hardware-IP-instructions correspond with the target hardware IP and ignores the others.

### C.2 INST_QUEUE

IFC_Synthesizer refers hardware-IP-instructions generated by compiler to synthesize INST_QUEUE. INST_QUEUE depends on the decoded bits by DECODER and pipeline stages of a target processor core.
**Ports Decision**  
*IFC_Synthesizer* decides control signals to *BUS I/O* and *HANDSHAKE*. Control signals to a hardware IP are correspond with *input_control* in CWL description. output ports of a hardware IP are input ports of *CONTROLLER*. The bit width of CWL description is equal the one of HDL description.

*output.data* signals connects *REGISTER*. Control signals to *REGISTER* depend on the register size (Sect.C.4 in detail).

Figure 8 shows output signals of *CONTROLLER* in VHDL at the example in Fig. 7.

**States Decision**  
*IFC_Synthesizer* decides states correspond with hardware-IP-instructions from a processor core. For CDP instructions, the number of states are equal to the number of functions of the target hardware IP. In case of Fig. 7, if the target hardware IP is numbered as “1” and *proc* is numbered as “2”, the processor core sends CDP 1, 2. *IFC_Synthesizer* defines the state *S_CDP 2* correspond with it.

**Sub-states Decision**  
*IFC_Synthesizer* decides the values of each sub-states.

When a target hardware-IP-instruction is LDC/STC or MCR/MRC, the control signals to hardware IP are “don’t care”. In case of receiving data instructions such as LDC and MRC, BUS I/O behaves as a data receiver from the bus to the *input REGISTER*. On the contrary, in case of sending data instructions such as STC and MCR, BUS I/O behaves as a data sender to the bus from the *result REGISTER*. In case of data transferring instructions such as LDC and STC, *HANDSHAKE_TR* is set to “1”, which means transferring data.

When a target hardware-IP-instruction is CDP, the control signals to hardware IP are required. In case of Fig. 7, *proc* operation is defined at *word* section. Since *proc* consists of *alphabets* R(Xa), N, O(Xd), sub-states *S_CDP 2 1*, *S_CDP 2 2*, *S_CDP 2 3*, correspond with *alphabets* R(Xa), N, O(Xd), are defined. The values of the control signals every sub-state are defined as the value at *alphabets* section in CWL description. *HANDSHAKE_RUN* is set to “1”, which means busy for processing data.

Figure 9 shows control signal decision at sub-state *S_CDP 2 1* correspond with *alphabet* R(Xa).

**Sub-state Transitions Decision**  
*IFC_Synthesizer* decides a sequence of sub-states to execute operations. In CWL description, a sequence of *word* is expressed as regular expression of *alphabet*. In case of *proc* operation in Fig. 7, the sequence of *alphabet* is R, N, N, R, N, N, O, O, O, O. Figure 10 shows sub-state transitions of *S_CDP 2 2* correspond with N.

**C.4 REGISTER**

When *IFC_Synthesizer* decides *REGISTER*, we must know the register size required. The size is given by the processor core synthesis system. Hardware-IP-instructions used in the application, include the length of transferring data, therefore we can decide the size of required registers.

**C.5 IFC**

IFC is the top layer of all the units in IFC. *IFC_Synthesizer* decides the mapping of all the units and external ports in IFC. Mapping is independent of the target hardware IP. Though external ports to the bus and the processor core is also independent, external ports to the hardware IP vary for a target hardware IP. We can decide the number and bit width of them from *port* section in CWL description of the target hardware.
IV. EXPERIMENTAL RESULTS

We implement IFC_Synthesizer in Ruby Language [9]. We design MPEG-4 encoder as a SoC application in SystemC under the design framework[14]. The design environment is as follows: OS: Linux 2.4, CPU: Intel Pentium III 500MHz, RAM: 192MB.

Figure 11 shows MPEG-4 encode algorithms. We partition them into hardware parts and software parts to estimate the performance as a representative. Color space convert (RGB to YCrCb), motion estimation (ME), motion compensation (MC) and discrete cosine transform / inverse discrete cosine transform (DCT/IDCT) are implemented by hardware IPs, quantization / inverse quantization (DCT/IDCT), variable length coding (VLC) are implemented by software.

Table I shows using hardware IPs information: the area of hardware IP, cycles during processing data.

Table II shows the results of synthesized processor cores[13]. The processor kernel is (1) RISC-type or (2) DSP-type. RISC-type kernel has five pipeline stages composed of IF, ID, EXE, MEM and WB stages. DSP-type kernel has three pipeline stages composed of IF, ID and EXE stages. The optional hardware units are functional units (ALU, multiplier), register files, and addressing units. They can be added to the processor kernel.

Table III shows the results of synthesizing IFC with synthesized processor cores in Tab. II by IFC_Synthesizer. IFC_Synthesizer synthesizes optimal IFCs correspond with a target hardware IP and the target processor core. The main reason the area of processor core B is larger than A is that the number of pipeline stage of B is more than the that of A.

The maximum of the execution time of IFC_Synthesizer is 9.4436 [sec], the minimum is 4.3475 [sec], the average is 6.5534 [sec]. However, in case of designing manually, the design of IFC requires about three days. IFC_Synthesizer reduces the cost of designing IFC.

V. Conclusion

In this paper, we presented an architecture of interface circuit to communicate with a processor core and a hardware IP, and a method for synthesizing it. Using the synthesis system “IFC_Synthesizer”, we can reduce the interface circuit development cost to less than 10 [min], while it would cost about three days by manual design. IFC_Synthesizer generates a HDL description of the interface circuit to communicate with the processor core and the hardware IP.

REFERENCES