Automatic Interface Synthesis based on the Classification of Interface Protocols of IPs

ChangRyul Yun[1], DongSoo Kang[2], YoungHwan Bae[3], HanJin Cho[3], KyoungSon Jhang[2]

[1]Agency for Defense Development, Daejeon, Korea, ipinterface@gmail.com
[2]Dept. Computer Engineering, ChungNam National University, Daejeon, Korea, {atom, sun}@cnu.ac.kr
[3]Multimedia SoC Design, ETRI, Daejeon, Korea, {yhbae,jhcho}@etri.re.kr

***Abstract* - In a System on a Chip (SoC) design, we use an IP-based design methodology to reduce design time. An interface circuit design is one of the most essential factors in IP-based design. However, it is not easy to generate interface circuits because IPs have various characteristics. For example, one IP may send only one outstanding address in a burst but another IP may need one address for each transfer in a burst. IPs also use different clock frequencies or different data widths. It is necessary to analyze the interface protocols of each IP to consider and resolve these differences during synthesis. In this paper, we categorize the various interface protocols and use the synthesis algorithm to select the appropriate structure based on the categorizations, clock frequencies, and data width differences of the IPs. Through the experiments, we show that we could automatically generate interface circuits for IPs with different clocks, different data widths, and no address concepts. Experiments also show the pros and cons of two structures based on the comparisons of the synthesis results of several IP pairs which could be employed between two alternative structures, namely, product FSM-based structure and FSMD-like structure.**

I Introduction

In a SoC design, IP-based design methodology is used to respond to the demands of high performance, complex functionality, and short time-to-market. Protocol conversion between IPs with different protocols is one of the most important topics in an IP-based design. It is necessary to generate interface circuits automatically because these design processes are both time-consuming and error-prone.

There are two kinds of automatic interface synthesis methods based on interface FSMs. One is based on a product FSM and the other is based on a FSMD (FSM and Data Path). In the former approach, a product FSM, as shown in Figure 1, is derived from the interface FSMs of IPs. The interface circuit consists of a product FSM and buffers. This structure allows the transfer of data without any clock cycle delay. Seawright and Brewer [1] were the first to report on the interface synthesis from interface FSMs. They used a regular expression to describe interface FSMs. Passerone [2] proposed a synthesis algorithm which creates a FSM that is the subset of the product FSM. Vijay D'silva [3, 4] proposed a synthesis algorithm to build a product FSM from each FSM of an IP and the compatibility check algorithm between two IPs. With this approach, we can determine whether or not an interface circuit is necessary. Yin-Tsung Hwang [5] proposed a template-based interface logic. An interface template consists of a protocol translation FSM, a mode control/command register module, a status/signaling register module, address/data buffers. An interface template is suited to the connection between IPs with similar characteristics.

Several studies [6][7] have been conducted on interface synthesis based on the FSMD structure. FSMD consists of a FSM and internal data buffers. The structure with FSMD is shown in Figure 2. FSMDs are used to communicate with each IP; a queue is used to transfer data between FSMDs[6]. Every data transfer has at least two cycle delay in order to pass through a buffer and then a queue. For this reason, this structure is not well suited for IPs expecting a fast acknowledgement. There have been proposals for the use of the Interface Protocol Component (IPC)[7], a kind of FSMD, in interface synthesis. An IPC is a complementary FSM for the corresponding interface FSM. The difference between a FSMD and an IPC is that an IPC has no internal buffer to store data temporarily. An IPC has several signals to control assumed external buffers to store addresses/data. We decided to use IPCs in our interface synthesis because we can reduce data passing delay. IPC-based interface circuit structure has at least one clock cycle delay, but it tends to have smaller area than product FSM-based structure because the former consists of two IPCs and external buffers as shown in Figure 11.

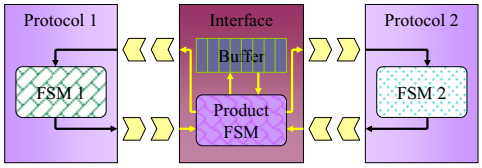


Fig. 1. The interface circuit structure based on a product FSM [3]

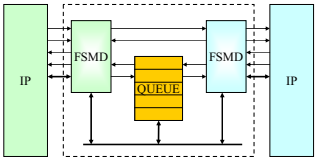


Fig. 2. The architecture of interface circuit based on FSMDs [6]

In addition, interface circuits have to deal with the differences in protocols, data widths, and clock frequencies. In product FSM-based structure, the number of states and transitions increases especially when IPs use different clock frequencies and different data sizes. On the other hand, the increase of states and transitions is not so severe in IPC-based structure. Which structure is better depends on the characteristics of the protocols of IPs as well as on the differences in clock frequencies and data widths. For example, let us consider the interface circuit for an OCP[8] master and an AHB [9] slave with different data widths, where the AHB slave uses a data bus two times wider than the master. Their product FSM has about three times more states and four times more transitions than the product FSM of the OCP-AHB pair with the same data width [3]. In addition, depending on the situations of a system, low latency on data transmission may be more important than low area or vice versa. However, most research usually has a fixed architecture for interface circuits. Our synthesis algorithm has the flexibility to select and generate an appropriate structure depending on the designer's interaction and decision. These are represented in the matching description as shown in Figure 8.

The original role of interface circuits is to transfer addresses and data correctly. Studies mentioned above

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

assume that a master and a slave sends/receives the same amount of data/address. However, one IP (e.g., PCI) may send only one outstanding address in a burst but another IP (e.g., AHB) may need one address for each transfer in a burst. The state-of-the-art protocols use different features on address transmission for efficiency, but other studies have not looked into this kind of IP pair as far as we know. We could not directly generate interface circuits between these IPs because the amount of addresses exchanged is different. In this case, the interface circuit should take the role of generating addresses for a slave IP. To deal with these kinds of differences, we need to classify IPs according to their communication characteristics. Our synthesis algorithm takes the different interface circuit structures according to the pairing of IP categories.

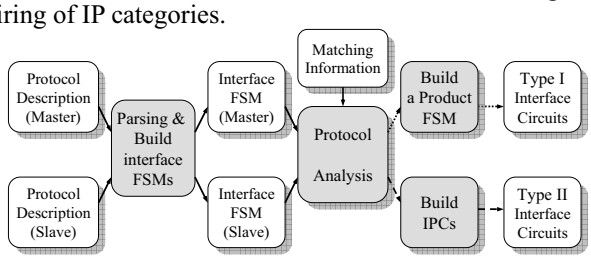


Fig. 3. Proposed interface synthesis flow

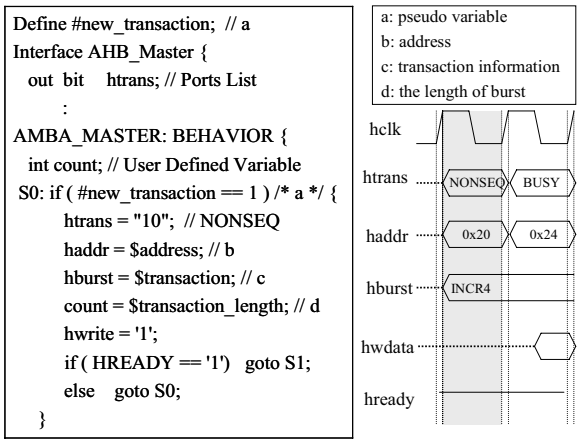
Our proposed interface synthesis flow is shown in Figure 3. Interface FSMs are built from input descriptions proposed in [10]. The Interface protocol description method is described in Section II. We describe the interface circuit structures based on matching information that corresponds to the different characteristics of IPs in Section III. Our algorithm for a product FSM and IPC generation shown in Section IV chooses an appropriate structure according to the protocol analysis. Experimental results with various protocols are described to show the efficacy of our method in Section V. Finally, we conclude this paper in Section VI.

II. Interface FSM

The waveform or timing diagram has been used to describe an interface protocol of IP, but it is informal and sometimes ambiguous for an automatic tool. An interface protocol must be described in a formal language. We use SIMPLE [10] to describe interface protocols of IPs that consist of transfers and parameters to indicate address, data, transaction information, etc. Parameters are used to match the different features of IPs. With this method, we can noticeably reduce the states and transitions of the interface FSMs of IPs with many transactions. An example for the AHB master protocol is shown in Figure 4. A pseudo variable such as '#new_transaction' in Figure 4(a) indicates an internal condition of an IP but is ignored in the synthesis. The parameter value of '$transaction', '$burst_length' is determined by matching information. We can reduce the states and transitions of an interface FSM because various transactions are described with only one parameter ('$transaction').

Figure 5 presents the definition of an interface FSM. We assume that any transition in a state must have a distinguishable set of action on the output port with other transitions in a state. A transition is written as $q \xrightarrow{t} q'$, where q and q' are the source and destination states, respectively, and 't' is a set of action.

An interface FSM shown in Figure 6(a) is extracted from the protocol description of the AHB master in Figure 4(a). Figure 4(a) shows only a transition from "S0" to "S1" in Figure 6(a). We could not describe all the ports of IPs as the space is limited in Figure 6. The input action that checks or saves the value of a signal is represented as "signal_name?value". Output action is denoted as "signal_name!value" and 'τ' is empty action. '$address', '$wdata', and '$rdata' mean address, write data, and read data, respectively. Such variables appear frequently in the matching description in Section III.C.

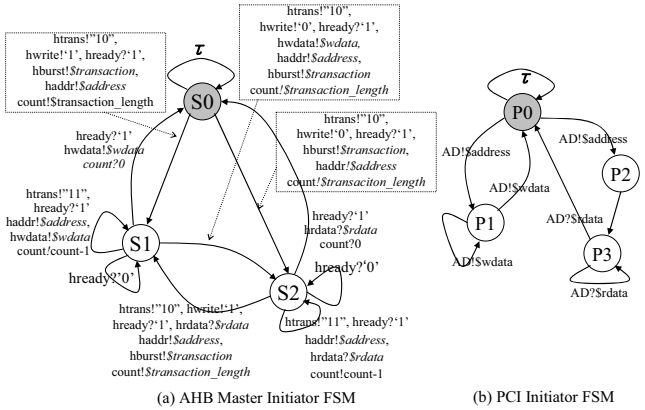


(a) Protocol Description (b) Equivalent Timing Diagram

Fig. 4. A part of the protocol description of AMBA AHB Master

Interface FSM $P = (Q, I, O, D, T, V, q_0)$
$t\ (\in T) = (A_i, A_o, A_d, A_v, q')\ (q, q' \in Q)$
Q: the set of states ($q \in Q$)
I, O: the sets input, output control signals ($i \in I, o \in O$)
D: the sets of address and data signals ($d \in D$)
V: the set of internal variables ($v \in V$)
T: the set of transitions on states ($t \in T$)
$q0$: initial state of FSM
q': next state
A_i, A_o, A_d, A_v, A_p : the sets of actions on input, output, address/data, internal variables and parameters

Fig. 5. The definition of an interface FSM



(a) AHB Master Initiator FSM (b) PCI Initiator FSM

Fig. 6. An interface FSM of the AHB Master and the PCI Initiator

III. The Categorization of interface protocols and the structure of interface circuits

A. IP Categorization

Interface protocols are classified into three categories according to the transmission styles of address and data. This is because the fundamental role of an interface circuit is the correct transmission of addresses and data and such styles affect the synthesis process.

We illustrate the examples of two interface FSMs, AHB master and PCI Initiator, in Figure 6. An interface FSM in Figure 6(a) belongs to category 'A' because the address, write data, and read data are sent by different ports. The FSM in Figure 6(b) belongs to category 'B' because the address, write data, and read data are sent through a port 'AD'. We have classified interface protocols into three categories as shown in Table 1. Most bus protocols belong to category A. PCI belongs to category 'B'. The category of an IP is automatically determined by analyzing the interface FSM.

Table 1. The categories of interface protocols of IPs or bus protocols

Category	Features	examples
A	Separated ports for addresses, write data, and read data	AHB, OCN, VCI, OCP, PLB, etc.
B	Shared one port for addresses, write data, and read data	PCI, PCI-X, etc.
C	No address	DES, AES, Color converter[11], etc.

B. *Matching Information*

The matching information includes port pairing, transaction mapping, address assignment on ports, etc. We will describe matching information together with the interface structures in the next section in detail because our algorithm decides the structure of interface circuit depending on the matching information. Designers use a simple Grapical User Interface (GUI) or a script file to construct the matching description.

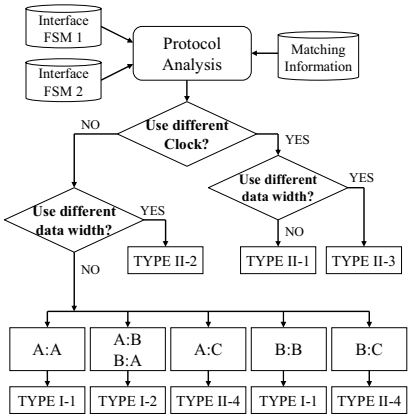


Fig. 7. The flow to decide the interface circuit structure

C. *Selection of Interface Circuit Structures based on IP Categorization*

First, IP categorization is performed and then the interface circuit structure is automatically determined as "Type I" or "Type II", as shown in Figure 7. "Type I" is the structure based on a product FSM and "Type II" is the structure based on IPCs. However, a designer can select an appropriate structure and buffer sizes through several GUI steps to trade-off between area and throughput. So far, we have not considered the case where category 'C' IP is a master.

1. The structure of "Type I"

"Type I" is the structure based on a product FSM. A buffer can be employed for each port pair with '$address', '$wdata', and '$rdata' in the interface FSMs as shown in Figure 8 "[Port Pairing]". A product FSM communicates with two IPs and controls the buffers so that data can be stored or bypassed to reduce transmission delay. Figure 8 shows a part of the matching information between the OCN [12] and the AHB slave. The "[TM]" part in Figure 8 is for the transaction matching, that is, the matching of '$transaction' in the interface FSM as shown in Figure 6(a). The underlined sentence in Figure 8 indicates that in order to connect two transactions with the length 4 of each IP, the value "011" of the "HBURST(2:0)" of the AHB slave should be mapped to the value of "01" of the "FS(3:2)" of the OCN. The transactions not specified in the matching information match the single transaction of each IP.

```
[Clock Ratio] 1 : 1 //// Master IP, Slave IP
//// Buffer Size, Signal (master), MSB, LSB,
//// Signal (slave), MSB, LSB, $address or $wdata or $rdata
[Port Pair] 1 , FA , 31 , 0 , haddr , 31 , 0 , $address;
[Port Pair] 1 , FD , 31 , 0 , hwdata , 31 , 0 , $wdata;
// Burst Transaction Length, Signal(master), MSB, LSB,
// Values, Signal(slave), MSB, LSB, Values
[TM] 1 , FS , 3 , 2 , "00" , HBURST , 2 , 0 , "000";
[TM] 4 , FS , 3 , 2 , "01" , HBURST , 2 , 0 , "011";
[TM] 16 , FS , 3 , 2 , "11" , HBURST , 2 , 0 , "111";
```

Fig. 8. A part of the matching information for OCN[12](Master) and AHB(Slave)

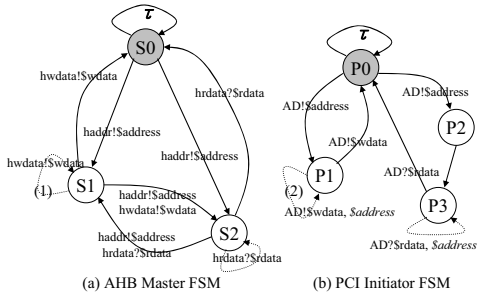


Fig. 9. Modified Interface FSM

```
//// Buffer Size, Signal (master), MSB, LSB, Signal (slave), MSB, LSB,
////    $address or $wdata or $rdata
[Port Pairing] 1 , haddr , 31 , 0 , ad , 31 , 0 , $address;
[Port Pairing] 1 , hwdata , 31 , 0 , ad , 31 , 0 , $wdata;
[Port Pairing] 0 , hrdata , 31 , 0 , ad , 31 , 0 , $rdata;
```

Fig. 10. A part of matching information for the AHB(Master) and PCI(Slave)

In "Type I – 2", the category 'B' IP only sends the first address in a burst transaction but category 'A' IP needs addresses in every transfer. Therefore, an interface circuit has to generate/ignore addresses when category 'B' IP is a master/slave. In order to generate a product FSM for "Type I – 2," the interface FSMs in Figure 6 are modified as shown in Figure 9, where the dotted lines indicate modified transitions. If category 'A' IP is a master, the actions on the address are eliminated except for the first address of a burst as shown in Figure 9(a) because a slave of category 'B' does not need other addresses except for the first one. In cases where category 'B' IP is a master IP, the actions on the address are added to the FSM as shown in Figure 9(b) because a slave of category 'A' needs addresses in each transfer in a burst. An address control part is added during synthesis. The port "ad" of PCI is paired with "haddr", "hwdata", and "hrdata", since the PCI protocol shares a port("ad") for address and data by matching information shown in Figure 10.

2. The structure of "Type II"

"Type II" is the structure based on IPCs as shown in Figure 11. An IPC is generated from the corresponding

interface FSM. An IPC communicates with the corresponding IP and external buffers. An IPC sends/receives data to/from buffers from/to an IP depending on the status of an IP and the buffers. Buffers, as shown in Figure 12, manage data transmission orders when there is an endianism difference. They also merge or slice data when two IPs have different data widths.

a. Type II -1

"Type II-1" is the structure for two IPs with different clock frequencies but the same data width. We assume that the faster clock frequency is n times that of the other (n: natural number). The interface circuit should be synchronized by the faster clock. Therefore, the IPC that communicates with the IP using a slower clock must be modified with dummy states according to the ratio of the faster clock speed over its clock speed. We could determine the clock ratio between two IPs from its matching information ([clock ratio]) as shown in Figure 8.

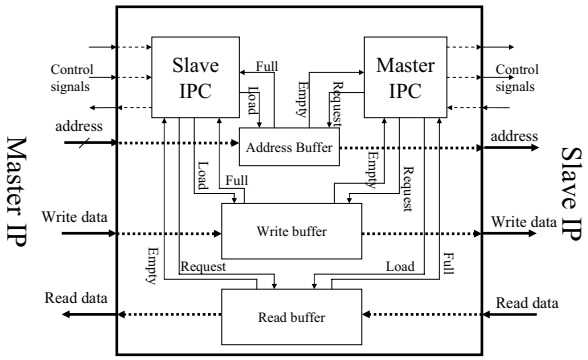


Fig 11. Interface circuit structure based on IPCs

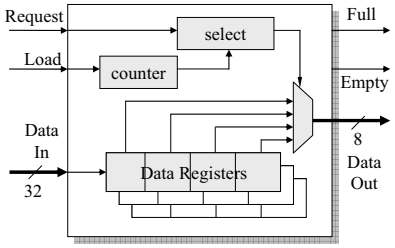


Fig 12. Data Buffer Structure for a different data width

```
//// port (Master), MSB, LSB, port (Slave), MSB, LSB,
//// address assignment, $data or $control
[Assignment] hwdata, 31, 0, key, 63, 32, haddr, 7, 0, 0x00, $data;
[Assignment] hwdata, 31, 0, key, 31, 0, haddr, 7, 0, 0x04, $data;
[Assignment] hwdata, 0, 0, start, 0, 0, haddr, 7, 0, 0x14, $control;
```

Fig. 13. A part of matching information for AHB(Master) and DES(Slave) – Address Assignment

b. Type II -2

"Type II-2" is the structure for two IPs with different data widths and the same clock frequency. Data should be merged or sliced depending on the ratio of data widths. In this paper, we divided the roles of an interface circuit as communication and data managing. An IPC only communicates with an IP and considers the status of buffers. An address buffer generates additional addresses for sliced data. and data buffers merge or slice data if it is necessary. Moreover, the order of transmission of sliced data can be changed when the endianism of two IPs is different. We could know the difference of data width between two IPs from its matching information ([Port Paring]) as shown in Figure 8.

c. Type II -3

"Type II-3" is the structure for two IPs with different data widths and different clock frequencies. Both methods described in Section a and b are applied.

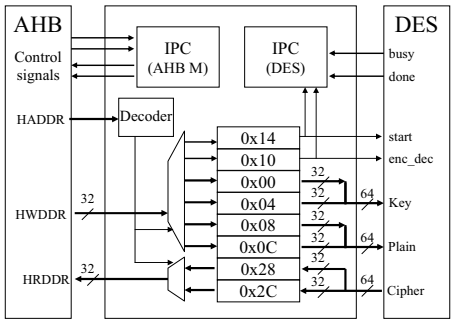


Fig. 14. An example to connect a IP(DES) with no address

d. Type II -4

"Type II – 4" is the structure for IPs with no address (category 'C'). When category 'C' IP is connected to a system bus or other IPs, we have to consider address assignment to control an IP. Figure 14 is an example for category 'C' IP, which is an interface circuit between an AHB master and DES. DES needs a 64-bit plain text and key but no address. The addresses of the master IP are assigned to correspond with each port of DES to control its operation. Because we cannot assign the addresses based only on an interface protocol, designers need to assign addresses through GUI steps, as shown in Figure 13. According to this information, we assign the addresses not only on data ports but also on control ports as shown in Figure 14. In this way, a master IP (AHB) can control a slave IP (DES) directly.

IV. The Generation of a Product FSM and IPCs

A. *Building a Product FSM*

An algorithm to build a product FSM is shown in Figure 15. Two FSMs (P_A, P_B) and matching information are inputs of an interface synthesis algorithm. The synthesis algorithm constructs a product FSM $P_I = (Q_P, I_P, O_P, D_P, V_P, T_P \in q_P, q_{A0B0})$. The output (input) ports of PI correspond to input (output) ports of interface FSMs and D_P is determined by port pairings in matching information. q_{A0B0} is an initial state of PI. Q_P is a subset of $\{< q_A, q_B, s(a), s(w), s(r)> \mid q_A \in Q_A, q_B Q_B, a \in D_P, w \in D_P, r \in D_P\}$, where s(a/w/r) indicates the existence (0 or 1) of data in the buffer for address/write data/read data port pairs.

We modified the synthesis algorithm presented in the paper [3] in order to consider matching information and to prune redundant states and transitions during the interface synthesis. The meaning of *valid()* function is as follows. *valid*($t_1' \cup t_2'$, s(a), s(w), s(r)) at (a) in Figure 15 at some state is defined true if the sender side has requested data stored in a buffer or can directly send the requested data from the sender side port to the receiver side port. The function *ModifyCounter()* function at (b) in Figure 15 changes the number of data in each buffer as data comes in and out of the buffer. The addresses sent from a master to the address buffer are useless except for the first start address when a slave needs the first address. Therefore, we changed *ModifyCounter()* so that in that case, the counter of the address buffer does not increase except for the first address by ignoring the following addresses depending on modified interface FSMs and matching information.

```
Q := null; // Q is a set of product-FSM.
PS := { [ P_A0, P_B0, s(a), s(w), s(r)] } // PS : a set of temporary states
Insert Initial State to PS;
while PS != null do
    Assign a state of PS to CurrentState
    Determine T_A ={t1:q_A}, TB ={t2:q_B}
  // TA ={t1:q_A} : the set of transitions originating from q_A
    for all t1 ∈ T_A, t2 ∈ T_B do
        t1':= ComputeComplement(t1)
// an action with all its responses complementary in transition (t1)
        t2':= ComputeComplement(t2)
        if valid(t1' ∪ t2', s(a), s(w), s(r) ) then   /* a */
            ModifyCounter(s(a), s(w), s(r), t1' ∪ t2') /* b */
    NTransition:=NewTransition(t1'∪t2',[q'A,q'B,s(a)',s(w)', s(r)'] )
     // ntransition's action (t1' ∪ t2'),
     // ntransition's next state ( [ q'A, q'B, s(a)', s(w)', s(r)' ] )
          AddTransition(NTransition, CurrentState);
          // Add new transition in CurrentState
          if [ q'A, q'B, s(a)', s(w)', s(r)' ]  (Q ∪ PS) then
              Add [q'A, q'B, s(a)', s(w)', s(r)' ] to PS
          end if
        end if
    end for
Prune_Transition (CurrentState)
Add  CurrentState  to Q and remove from PS
end while
```

Fig. 15. Modified Algorithm for a product FSM Generation

```
for all q ∈P_A, P_B do   // all states
  for all t1 ∈q do   // all transitions
     t1 := ComputeComplement(t1)  // a
    AddBufferControlSignals(t1);  // b
  end for;
CheckDeadStates();  // c
end for;
```

Fig. 16. IPC Generation Flow

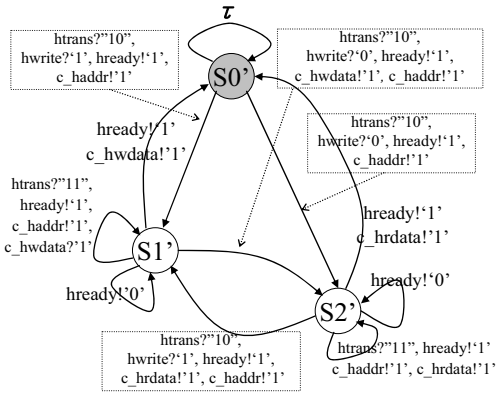


Fig. 17. A IPC(complementary FSM) for AHB master

B. Building IPC

IPC generation flow is shown in Figure 16. A slave IPC recognizes the behavior of a master IP and manages the external buffer signals to store/read data to/from external buffers. A master IPC starts operations depending on the status of the external buffers. An IPC is a complementary FSM of the interface FSM of each IP. We changed the port direction on each transition and checked whether a transition has distinguishable action or not (Fig. 16 a). Also in order to control buffers, buffer control signals such as "load", "request", "empty", and "full" are inserted in a transition with actions concerning addresses and data (Fig. 16 b). Next, the dead state should be removed (Fig. 16 b). A dead state means that the next state of all transitions in a state indicates itself. Figure 17 is an IPC of the AHB master shown in Figure 6(a). Buffer signals are omitted as space is limited in Figure 17.

V. Experiments

We performed experiments to compare automatic generated circuits with manual designs. The comparisons of the synthesis results of interface circuits are shown in Table 2. The manual designer has two years experience using digital circuit design with VHDL and Verilog. Interface circuits are synthesized by Xilinx ISE with Xilinx Virtex II XC2V6000. In Table 2, 'Area' means the number of slices and 'f_{max}' is the maximum frequency of an interface circuit. We could not show all our experimental results as space is limited. Automatic designs of "Type I" are 1.8 times larger in area and 0.7 times faster in the maximum frequency than manual designs on average. The test examples "OCN:AHB" and "PCI:AHB" show a particularly larger area than others because a 32bit adder is inferred from the generated VHDL description, which is used to generate addresses for "OCN" and "PCI" which, in turn, does not send addresses except for the first address in a burst. We hope to reduce the size of adders inferred later. In "Type II," there are examples for IPs with different data widths and different clock frequencies. We assume that the depth of buffers that store data temporarily is four. Automatic designs of "Type II" are 1.2 times larger in area and 0.9 times faster in maximum frequency than manual designs on the average. However, every address and data has at least one cycle delay because all data have to be stored and then passed to other IPs. In addition, we could generate and verify the interface circuit for an IP with no address. Table 2 shows that our proposed approach generates interface circuits that are comparable with manual designs especially in the case of IPC-based structure (i.e., "Type II").

In the second experiment, as shown in Table 3, we compare product FSM-based structure with IPC-based structure for the test examples where we can apply two structures. "Type I" usually has more states and transitions than "Type II." The area of "Type I" is also 1.2 times larger and the maximum frequency of "Type I" is 0.9 times faster than "Type II" on average. "AHB:PVCI" is a case where the area of "Type I" can be smaller than that of "Type II". In that case, the area for buffer control of "Type II" is larger than that of a product FSM that has a few states and transitions. However, "Type I" has smaller latency than "Type II." In Table 3, "W" and "R" mean the number of cycles to finish a write transaction and a read transaction (length 4) on a master IP, respectively. We assume that slaves are always ready to respond. In case of the read transaction, "Type II" needs two more cycles than "Type I" because data has to pass through the buffers. With this experiment, we observed that "Type I" has a generally less data transmission latency but a somewhat larger area and slower maximum operating frequency than 'type II'.

We compared our method with a previous work [3] in Table 4. This experiment also shows the effectiveness of IPC-based interface structure when there are differences in clock frequencies or data widths. The experiment of the approach [3] only considers read transactions. We observed

that the approach [3] and our approach show similar results in cases where product FSM-based structure was employed. However, our approach results in a smaller number of states(# S) and transitions(# T) over product FSM-based approach [3] in cases where there are differences in clock frequencies and/or data widths. In that case, our approach automatically chooses IPC-based structure.

We verified the functionalities of each generated circuit. We especially observed that the generated interface circuits (AHB:OCN, OCN:AHB) could replace manually-designed interface circuits and work correctly on a H.264 decoder system[13].

VI. Conclusion

In this paper, we have presented a method to automate the synthesis of interface circuits based on protocol categorization. In previous studies, the structure of interface circuit was fixed, but our synthesis algorithm chose an appropriate structure of interface circuits based on the categorization and the differences in clock frequencies and data widths. Moreover, we observed that, generally, product FSM-based structure has less data transmission latency but somewhat larger area and slower maximum operating frequency than IPC-based structure.

We could generate interface circuits for IPs with different clocks, with different data widths, and with no address concepts. Through our experiments, we noticed that the performance of the generated circuits is comparable with that of the manual designs especially when the corresponding IPs have different clock frequencies and/or data widths.

References

[1] Seawright A., and Brewer F., "Clairvoyant: a synthesis system for production based specification," Very Large Scale Integration (VLSI) Systems, IEEE Transaction, Volume 2 Issue 2, June 1994, Page(s):172-185

[2] *Passerone* R., Rowson J.A., Sangiovanni-Vincentelli A., "Automatic synthesis of interfaces between incompatible protocols," Proceedings of Design Automation Conference, 15-19 Jun 1998 Page(s):8 - 13

[3] Vijay D'*silva*, S. Ramesh and Arcot Sowmya, "Bridge Over Troubled Wrappers: Automated Interface Synthesis," Proceedings of the 17th International Conference on VLSI Design 2004 Page(s):189 – 194

[4] Vijay D'*silva*, S. Ramesh, "Synchronous Protocol Automata: A Framework for Modeling and Verification of SoC Communication Architectures," Proceedings of Design, Automation and Test in Europe Conference and Exhibition, Volume 1, Feb. 2004 Page(s): 390-395

[5] Yin-Tsung Hwang and Sung-Chun Lin, "Automatic Protocol Translation and Template Based Interface Synthesis for IP Reuse in SoC," Proceedings of the 2004 IEEE Asia-Pacific Conference on Volume 1, Dec. 2004, Page(s): 565-568

[6] Dongwan Shin and Daniel Gajski, "Interface Synthesis from Protocol Specification," Technical Report (CECS-02-13), April 12 2002, Center for Embedded Computer Systems University of California, Irvine

[7] ChangRyul Yun, YoungHwan Bae, HanJin Cho, KyoungSon Jhang, "Automatic Synthesis of Interface Circuits from Simplified IP Interface Protocols," Proceedings of the Eleventh Asia-Pacific Computer Systems Architecture Conference (ACSAC 2006), September 6-8th, Page(s):581-587

[8] O.C.P.I.P.A. Inc. Http://www.ocpip.org

[9] ARM Inc. AMBA™ Specification Rev 2.0m, document Number ARMIHI0011A

[10] ChangRyul Yun, KyoungSon Jhang, "An Interface Protocol Component Modeling Language," Proceedings of the 15th ASIC/SOC Conference, Sept. 2002, Page(s): 456-460

[11] http://www.opencores.org

[12] Se-Joong Lee, Seong-Jun Song, Kangmin Lee, Jeong-Ho Woo, Sung-Eun Kim, Byeong-Gyu Nam, Hoi-Jun Yoo, "An 800MHz star-connected on-chip network for application to systems on a chip," Proceedings of 2003 IEEE International on Solid-State Circuits (ISSCC 2003), Page(s):468 – 469

[13] Jin Ho Han, Mi Young Lee, Younghwan Bae, and Hanjin Cho, "Application Specific Processor Design for H.264 Decoder with a Configurable Embedded Processor," ETRI Journal, vol.27, no.5, Oct. 2005, Page(s):491-496

Table 2. The comparison between manual designs and automatic designs

Type	Master : Slave	Area (Slices)		f_{max}		A / M	
		M	A	M	A	Area	f_{max}
I–1	AHB:OCN	57	103	234	181	1.8	0.8
	OCN:AHB	112	374	197	105	3.3	0.5
	BVCI:AHB	44	61	580	356	1.4	0.6
	AHB:APB	66	121	268	249	1.8	0.9
I–2	AHB:PCI	50	72	315	238	1.4	0.9
	PCI:AHB	82	212	258	144	2.6	0.6
II–1	AHB:PVCI (50Mhz, 32bit: 12.5Mhz, 32bit)	310	356	214	191	1.1	0.9
II–2	AHB:PVCI (50Mhz, 32bit: 50Mhz, 8bit)	244	316	225	208	1.3	0.9
II–3	AHB:PVCI (12.5Mhz, 32bit: 50Mhz, 8bit)	237	316	204	204	1.3	1.0
II-4	AHB:DES	134	145	225	197	1.1	0.9

(**M**: Manual Design, **A**: Automatic Generated Design)

Table 4. The comparison with previous work [3]

Master:Slave	Clocks	Data Width	Previous Method		Our Method	
			# S	# T	# S	# T
AHB:PLB	1:1	1:1	7	12	6	12
OCP:AHB	1:1	1:1	8	18	9	20
AHB:PLB	1:2	1:1	12	15	6	8
OCP:PLB	1:1	1:3	17	32	6	8

Table 3 The comparison between "Type I" and "Type II" structure

Master:Slave	Type I (Product FSM Based)						Type II (IPC Based)						Type I/ Type II	
	# S	# T	Area	f_{max}	W	R	# S	# T	Area	f_{max}	W	R	Area	f_{max}
AHB:OCN	9	53	203	181	5	12	6	34	174	198	5	14	1.2	0.9
OCN:AHB	12	89	374	105	4	12	6	35	256	121	4	14	1.5	0.9
AHB:BVCI	18	48	324	219	5	8	6	20	276	228	5	10	1.2	1.0
BVCI:AHB	10	42	249	201	4	8	7	21	205	217	4	10	1.2	0.9
AHB:PVCI	10	39	252	204	5	8	6	19	267	212	5	10	0.9	1.0