Synthesis and Optimization of Interface Hardware between IP's Operating at Different Clock Frequencies

Bong-Il Park, Hoon Choi, In-Cheol Park, and Chong-Min Kyung
Div. of EE, Dept. of EECS, KAIST
E-mail: bipark@duo.kaist.ac.kr, hchoi7@samsung.co.kr, {icpark,kyung}@ee.kaist.ac.kr

Abstract

In system-on-a-chip design, interfacing of Intellectual Property(IP) blocks is one of the most important issues. Since most IP's are provided by different vendors, they have different interface schemes and different operating frequencies. In this paper, we propose a new interface synthesis method that enables one not only to handle the interface between IP's with different operating frequencies but also to minimize the hardware resource required for the interface. We have demonstrated the proposed algorithm by applying it to a real design example, MP3 decoder, and verified the IIS-to-PCI protocol converter on a real hardware system.

1 Introduction

Steady advances in design methodology and semiconductor technology has allowed the complexity of a single chip to contain more than one million transistors [1]. Time-to-market issue in the complex system-on-a-chip(SoC) design can only be solved by dealing with complex building blocks commonly called Intellectual Properties(IP's). It is to be noted that IP-based design methodology requires a large design space exploration and, therefore, a very efficient design verification methodology handling various IP's with different interface protocols.

The problem of interface synthesis, as originated from the network protocol conversion, has been addressed in various literatures [2–9]. In computer communication networks, it is often necessary to connect network components which stand on different network architectures. Previous works containing the analysis of conversion schemes [2–4], however, have been mainly focused on the software aspects without considering the hardware aspects.

Borriello and Katz [10] introduced the event graph to establish the correct synchronization and data sequencing. The limitation of this approach is that the two protocols should be made compatible manually by assigning labels to the data on both sides, since the protocol specification is given at the very low level of abstraction using waveforms.

In another approach taken by Narayan and Gajski [9], protocol is first classified into five types of atomic operations: (1) waiting for an event on an input control line, (2) assigning a value to an output control line, (3) reading a value from input data line, (4) assigning a value to an output data line, and (5) waiting for a fixed time interval. Then the protocol is represented as an ordered set of relations whose execution is guarded by a condition or by a time delay. Finally, relations between two protocols are grouped into a set of relation groups such that in each group the size of the data generated by the relations in the group from one protocol is identical to that expected by the relations in the group from the other protocol. Although this approach starts from HDL description and considers the data width mismatch between two modules, timing constraints are not considered, and thus it may not be applied to the real interface synthesis.

The approach proposed by Sun and Brodersen [8] provides a library of components to free the user from considering lower-level details, which is very similar to the ASIC design process. As in developing the library for ASIC design, the approach also requires a large library and cannot implement interface logic circuits not available in the library.

The approach proposed by Akella and McMillan [7] provides a protocol specification consisting of two FSM's for describing the producer/consumer part of the interface and specification describing the valid state transitions of the interface. Product of the two FSM's is taken as a solution, where the invalid state transitions are pruned according to the specification. Drawbacks are that no mismatch in data width can be handled and that the designer has to manually specify the intended behavior of the interface in the form of FSM.

Since most IP's are provided by different vendors with different design environments, they have inevitably different characteristics including operating frequency and interface scheme, etc. However, most works related to interface synthesis assume the two communicating parties are driven by the same clock. In this case, if a designer has some IP's with mutually different operating frequencies, all the IP's must be operated at the lowest frequency causing the performance degradation. In our approach, this problem is overcome by allowing different clocks in communication and optimizing the hardware resource. Supporting different clocks allows wider choice for candidate IP's to be integrated on a chip. The assumption of identical operating frequency is too hard constraint for selecting IP's and produces poor design quality in the eventual SoC design. Moreover, our approach directly synthesizes interface logic from the source description without additional modification for frequency matching, which produces better performance and ease of use.

This paper is organized as follows. In Section 2, the advantage using different frequencies in the interface is shown through the simple interface example. In Section 3, we formulate the interface synthesis problem and present interface synthesis for the system-on-a-chip

0-7695-0801-4/00 $10.00 © 2000 IEEE

considering different operating frequencies. Section 4 describes new algorithms developed for the selection of operating frequency and internal queue sizing. The results of application of the proposed algorithm to the MPEG decoder is shown in Section 5 followed by the conclusion.

2 Simple Interface Example

The IDCT block is shown in Fig. 1(a). It has three different operating phases which are data receiving, data processing, and data sending, as shown in Fig. 1(b). It must receive or send a data every one clock cycle through 16-bit data bus. As shown in Fig. 1(b), each phase needs at least 338 cycles which consists of 64 cycles for data receiving, 210 cycles for data processing, and 64 cycles for data sending. The IDCT block is designed with 0.35um standard cell library and can be operated up to 70MHz.

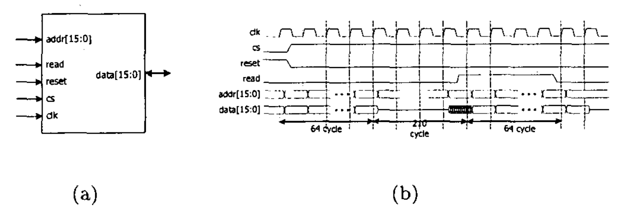

Figure 1. (a) Block diagram and (b) timing diagram of IDCT where a data is transmitted/received every clock.

Fig. 2(a) shows SRAM controller. The basic cycle for data access needs two cycles, which are address sending and data accessing as shown in Fig. 2(b). The SRAM controller can be operated up to 100MHz.

The two IP's have different interface signals and operating clock frequencies. Thus, interface logic is synthesized with two different clocking schemes. If they operate at their own clock frequencies, the performance can be maximized although the interface logic can be slightly complex. Table 1 shows the estimated performance of hardware interfacing between IDCT and SRAM. The second column is obtained when two clock inputs of each IP's are different, that is, the SRAM and IDCT block operates at 66MHz and 33MHz, respectively. Although the SRAM and IDCT block operates at more higher clock frequencies, their operating frequencies are limited by the system specification. The third column is obtained when the two clock inputs have the same clock frequency. At the different clock scheme, we can obtain several advantages which include shorter critical path delay, smaller area, and less power dissipation. In case of different clocks, internal storage elements were eliminated. This produces the smaller area. The critical path delay can be reduced due to the smaller area of protocol converter, although more states in the protocol converter are inserted. The reduced hardware and the shorter time for data transfer reduce the estimated power. In this paper, for these advantages, we propose a new interface synthesis scheme with different clock frequencies.

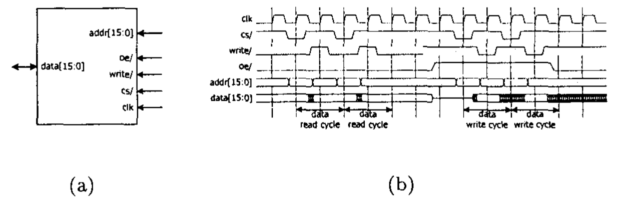

Figure 2. (a) Block diagram and (b) timing diagram of SRAM controller where a data is transmitted/received every two clock.

Table 1. Estimated performance of hardware interfacing between IDCT and SRAM

Operating Clock	same clock	different clocks
Critical Path Delay	9.34nsec	8.92nsec
Equivalent gate count	9424.68	610.69
Estimated Power	11.18mW	1.22mW

3 Interface Synthesis between IP's with Different Operating Frequencies

In this section, we introduce the interface synthesis process and propose a new process for handling IP's with different operating frequencies.

3.1 Problem formulation

Fig. 3 shows a simple example of two protocols where each protocol is shown as a pair of communicating finite state machines; Protocol A is the so-called *polling model* and protocol B is the *ack-nack model.* Each state is denoted by a circle with the state name, transition conditions and outputs shown on edges. The string with '?' character represents the transition condition and the string with '!' character means the output.

The aim of interface synthesis is to allow the communicating component of one protocol to communicate with that of another protocol. In Fig. 3, let us assume that A_0 and B_1 need to communicate with each other. To synthesize an interface between A_0 and B_1 is to find a new FSM C between A_0 and B_1 as shown in Fig. 4. In Fig. 4, A_0 'believes' that it is communicating with A_1 which consists of C and B_1. Similarly, B_1 regards C and A_0 as B_0. Therefore, interface synthesis is to find such an FSM C for given protocols A and B. In addition, the proposed approach do not have any constraint for operating frequency, while the given two protocols are operated at the same operating frequency in conventional approaches.

3.2 Basic Interface Synthesis

In our work, a protocol is modeled by an FSM, which is converted into an LTS(Labeled Transition System) as used in [7]. We adopted the approach of [7] as the

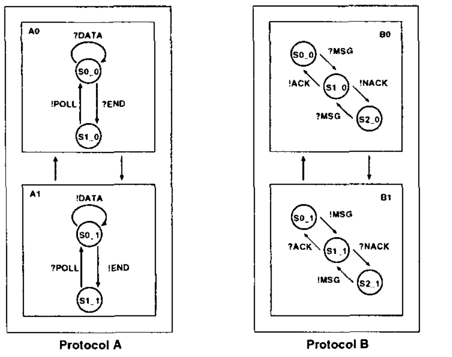

Figure 3. An example of each of the two model protocols: each protocol is represented as a pair of communicating FSMs where protocol A is the so-called *polling model* **and protocol B is the** *ack-nack model.*

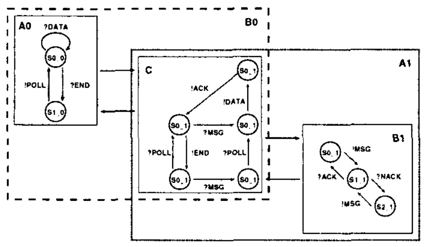

Figure 4. An example of interface logic between A_0 and B_1: C and B_1 mimics A_1, while C and A_0 mimics B_0

basic interface synthesis algorithm, which is briefly reviewed here. As stated in the problem formulation, the input is the description of the two protocols used by each module. Fig. 5 shows the relation between interface FSMs of the two modules,A and B. Each descriptions of the two protocols include the behavior of data and control over which the transfer occurs. Since the protocol converter has to communicate with A and B, Fig. 5 shows that the FSM of the protocol converter must be a dual FSM of A if observed from A and a dual FSM of B if observed from B.

In [7], every FSM is described with the general labeled transition system(LTS). The transitions in the protocol converter are generated by obtaining the product of the two dual LTSs. For example, $C = A^d \times B^d$ in Fig. 5. The resulting product machine includes all possible combinations of the transitions. Since all possible transitions may not be legal transitions and the result usually is state explosion [11], a new LTS called an interface specification LTS is required to describe the legal transitions. Thus we can eliminate illegal transitions and uncontrollable transitions in the product of the dual LTSs by using the interface specification LTS [7,11].

3.3 IP's with different operating frequencies

In Fig. 5, let us assume that the producer,A, is operated at the clock frequency f_A, while the consumer,B, is operated at f_B. Let us assume that f_A is smaller than f_B, that is, the producer is operated at lower clock frequency without loss of the generality. Additionally, we assume that two clocks have the same clock source and they are synchronized at their edges. The simplest approach for the protocol converter is very trivial. All the LTSs, *i.e.*, the producer, the consumer, and the protocol converter, are operated at the same clock, which is the slowest clock. This approach does not cause any problem except for the performance degradation.

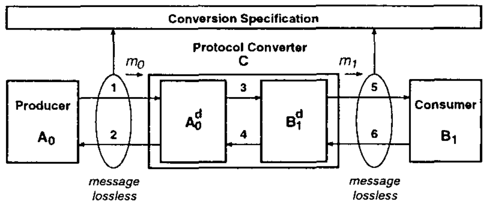

Figure 5. Constructing a protocol converter using dual machines and conversion specification

The next approach provides the better performance by allowing different clock frequencies. The producer and the consumer are operated at their own operating frequency, f_A and f_B respectively, while the protocol converter is operated at certain frequency which may be either f_B or f_A. However, this solution causes some problems. If it is operated at f_B, the protocol converter receives too many redundant messages from the producer. As the clock period of the producer is longer than that of the protocol converter. On the other hand, it sends too many redundant messages to the consumer. In the following we classified the situation in two cases according to the value of $\frac{f_B}{f_A}$.

3.4 Case I: $\frac{f_B}{f_A}$ is an integer number

Fig. 6 shows the case when f_B is some integer multiple of f_A. The protocol converter, which consists of two FSMs(A_0^d and B_1^d), operates at f_B, because if it operates at the lower clock frequency, f_A, the signal of the fast FSM(B) changes within a clock period and , therefore, interfacing is not working. Thus it operates at the higher clock frequency, f_B. Since the interface between B and B_1^d operates at the same clock frequency, we do not need any special consideration. However, some modifications are necessary between A and A_0^d.

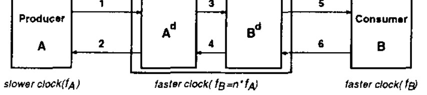

Figure 6. Case I: $f_B = n \times f_A$, where n is an integer.

Interface synthesis is performed after the states and edges are modified according to the following path.

Thus let us focus on the output generation paths of A_0^d which can be classified into the following four paths:

Path 1(signal 1$\rightarrow$ signal 3): For this path the input signal(signal 1) has the longer period than the output signal(signal 3). An obvious and efficient solution for this case is to insert a frequency matching counter which counts from 0 to $(n-1) = (\frac{f_B}{f_A} - 1)$. If f_A and f_B are 10MHz and 20MHz, respectively, an input signal has to be sampled at 10MHz. Otherwise, *i.e.*, if sampled at 20MHz, the FSM of A_0^d transits twice as if an input signal has been asserted for two successive cycles. This situation can be avoided with the clock cycle counter shown in Fig. 6. The output of the counter is asserted to enable input sampling at every $T_0(= n{\cdot}T_1)$. T_0 and T_1 represents the clock periods corresponding to f_A and f_B, respectively. In Fig. 7 which shows the modified FSM for frequency matching, (a) shows an original FSM which waits for the signal `poll` and transmits `data` until the signal `last` is asserted, while (b) shows the modified FSM in which the transition of state occurs only when the output signal of the counter(`cntN`) is asserted.

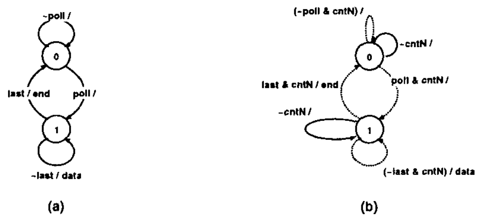

Figure 7. Path 1: (a) original FSM, and (b) Modified FSM for frequency matching.

Path 2(signal 4 $\rightarrow$ signal 3): This path does not need to be considered since the input(signal 4) and output(signal 3) signals are connected between the inter module of the protocol converter operating at the same operating frequency.

Path 3(signal 1 $\rightarrow$ signal 2): Similar to the path 1, the input signal needs to be sub-sampled. In addition, since the output signal is the input to A, it must be extended to cover one T_0. The transition edges of the FSM corresponding to input signals are modified in the same way as path 1, and those corresponding to output signals are also inserted. The inserted edges help assert the output signals during one T_0. In Fig. 8, newly inserted transition edges are represented as solid arrows.

Path 4(signal 4 $\rightarrow$ signal 2): This requires more complex modifications which include some new states and transition edges. In this path, the input signals(signal 4) are valid during only T_1. If an input signal is changed, this information is stored within a new state and is restored when the two clocks are matched every $T_0(= n \cdot T_1)$. Thus one input signal makes two new states which are used for storing a signal transition and asserting an output signal during $T_0(= n{\cdot}T_1)$. In the example of Fig. 9(b), if the signal `last` changes and the current state is '`1`', state transits to '`1_1`' or '`1_2`' according to the time when the signal changes.

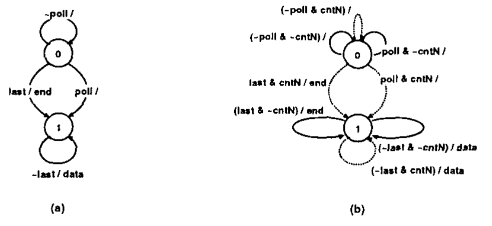

Figure 8. Path 3: (a) original FSM, and (b) its Modified FSM for frequency matching obtained by inserting new edges

Then new input signal can be asserted after the output signal of the counter(`cntN`) is asserted.

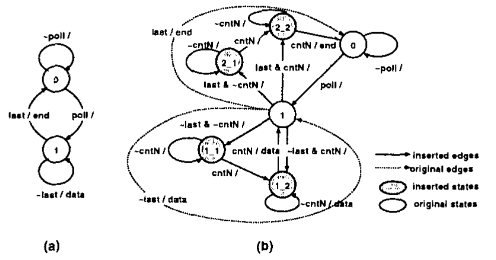

Figure 9. Path 4: (a) original FSM, and (b) its Modified FSM obtaining by inserting new states and edges

3.5 Case II: $\frac{f_B}{f_A}$ is a real number

If the ratio, $\frac{f_B}{f_A}$, is not an integer but a real number, then the several solutions are possible. The simplest solution is that both IP's are operated at the lower clock frequency. This requires no additional hardware although it degrades some performance. The next solution is to find the least common multiplier(LCM) of both clock frequencies since any real number can be represented as a ratio of two integers. For example, the ratio of 1.5 can be represented as 2:3. Then the LCM clock is inserted to all the blocks including the protocol converter, the producer, and the consumer. In this case, the interface between the protocol converter and the producer(consumer) is synthesized as in the case I. Thus the protocol converter needs two frequency matching counters as shown in Fig. 10.

However, if the two clock frequencies are slightly different, the LCM clock may be infeasible to implement. For example, if f_A and f_B are 59MHz and 60MHz respectively, then the 3.54GHz clock must be generated! It is nearly impossible to make such high frequency clock. In this case, the clock frequency should be slightly changed a *priori* to make the LCM a reasonable value.

4 More Considerations for Interface Synthesis

Until now, we assumed that IP's operate at certain single clock frequency. However, each IP has generally its operating range, bounded by the lowest and the

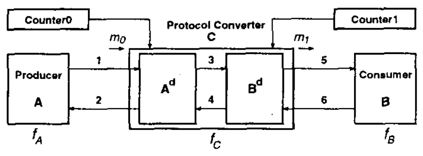

Figure 10. Case II: $f_A = r \times f_B$, where r is a real number

highest operating frequency. In this condition, we must first determine the operating frequency. Fortunately, if the operating ranges of two IP's are overlapped, we can just select any frequency within the overlapped range. In our interface synthesis approach, if two IP's are operated at different frequencies, then we must find an LCM frequency. Here our synthesis approach is extended to the case having the operating range of IP's.

4.1 Determination of operating frequency

Fig. 11 shows the pseudo code for determining the operating frequency. In this algorithm, inputs of the algorithm are the operating ranges of two IP's ,*i.e.*, a range,(f_{L1}, f_{H1}) and another range,(f_{L2}, f_{H2}). The algorithm determines the common operating frequency which equals an integer times the operating frequency of each IP. In the inner **while** loop, it searches an overlapping range between the ranges as the slow IP range is widened by multiplying an integer number. If the overlapping range cannot be found in the current range of the fast IP, the fast range is widened in the outer **while** loop. This iteration is continued until the overlapped range is found. For example, one IP operates from 15MHz to 20MHz, and the other IP operates from 42MHz to 44MHz. Then, in the first inner loop, overlapped ranged is not found, since the integer multiple ranges of the lower IP are (30MHz, 40MHz), (45MHz, 60MHz), etc. In outer loop, the range of the fast IP is changed into (84MHz, 88MHz). The range is overlapped with (75MHz,100MHz) which equals the five times of the range (15MHz, 20MHz). Finally, the frequency, 88MHz, is determined as the operating frequency, since it is the highest frequency of overlapped range, *i.e.* (84MHz, 88MHz).

4.2 Determining of queue sizing

Some protocols may have strict timing constraints between one message and the next message. If IP's based on these protocols are communicated through the protocol converter and have different operating frequencies, the protocol converter needs internal storage elements. As shown in Fig. 5, if a message must be sent every three clocks and received every single clock, the protocol of the consumer can be violated because messages are not available in time. This problem is solved with an internal queue which first stores all messages for sending and sends the stored messages according to the consumer protocol. In this case, the objective is to minimize the internal queue size which can be formulated as follows:

$$\text{Queue Size} = D_{total} - \lfloor D_{total} \frac{R_{m0}}{R_{m1}} \rfloor \qquad (1)$$

```
input: $f_{L1}, f_{H1}, f_{L2}, f_{H2}$
output: $f_{common}$
while $f_{L2} <= f_{max}$ do
    while $n \cdot f_{L1} <= m \cdot f_{H2}$ do
        if $(n \cdot f_{L1}, n \cdot f_{H1})$ overlap with $(m \cdot f_{L2}, m \cdot f_{H2})$
            return $f_{common} = \min(\, n \cdot f_{L1}, m \cdot f_{H2}\, )$;
        else
            $n = n + 1$;
        end if
    end while
    $m = m + 1$;
end while
return $f_{common}$ = can_not_find;
```

Figure 11. Algorithm for determining of operating frequency, where assume that $f_{L1} < f_{L2}$ and $f_{H1} < f_{H2}$

where D_{total} is the total amount of message sent, R_{m0} represents the rate of message sending, R_{m1} represents the rate of message reception, where we assumed that receiving rate is higher than the sending rate.

If a queue size equals D_{total}, then all data can be stored in the queue and the stored data are sent to the consumer sequentially. Since a queue can handle receiving and sending of data concurrently, sending operation can be started just after certain amount of data is stored. In the equation 1, the right term, $\lfloor D_{total} \cdot R_{m0}/R_{m1} \rfloor$, corresponds to the data operated concurrently. For example, if we have a 256 bit messages and the sending rate and receiving rate are 1Mbps and 2Mbps respectively, then the required queue size is 128 bits. The consumer protocol starts just after 128 bit messages are all stored in the internal queue. After this time, the internal queue stores the remaining 128 bits and sends messages concurrently.

5 Experiment

The proposed approach is applied to an MPEG 2 audio layer 3(MP3) player system through Virtual Chip System [12, 13]. In the MP3 decoder example, the interface part deals with three ports consisting of one IIC port, two IIS ports, and PCI interface connects between these ports and the software model. PCI interface operates at 33MHz, while one IIC and two IIS operate at 500KHz and 3MHz, respectively. Moreover, the data width of each protocol is not matched.

The decoded bit stream must be supplied continuously to the D/A Converter(DAC) of the MP3 player system at the bit rate of 3Mbps using IIS protocol. If the decoded bit stream is infinite, an infinite size queue which stores all the decoded bit stream is required, which is infeasible. Moreover, since the MP3 software model requires time for decoding a input bit stream, a chunk of 16-bit data should be occasionally supplied through PCI interface. Therefore, the decoded bit stream must be divided by certain amount. In this example, the decoded bit stream is divided by 90.9K bits which correspond to 1M cycles at 33MHz

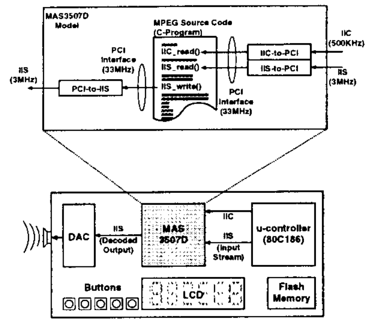

Figure 12. The lower part shows the block diagram of an MPEG 2 audio layer 3(MP3) player system. MP3 decoder(MAS3507D) has three interface ports which consist of one IIC port, two IIS ports

as shown in equation 2, though the protocol converter must infinitely supply it. Thus the required queue size can be calculated as follows:

$$
\begin{aligned}
D_{total} &= \frac{1M\text{-cycle}}{33MHz} \times 3M\text{bps} = 90.9K\text{bit} && (2)\\
R_{m0} &= 3M\text{bps} && (3)\\
R_{m1} &= \frac{33MHz \times 16\text{bit}}{P0 + P1} = 132M\text{bps} && (4)\\
\text{Queue Size} &= D_{total} - \lfloor D_{total}\frac{R_{m0}}{R_{m1}} \rfloor = 88.8K\text{bit} && (5)
\end{aligned}
$$

Equation 2 shows the IIS protocol during 1M cycle transfers 90.9Kbits. In equation 4, $(P0 + P1)$ means the number of clock for single PCI write transaction which requires minimum 4 cycles. Although this example does not show much reduction of queue size, if the decoded output stream has higher bit rate, queue size can be significantly reduced. As shown in Fig. 13, the IIS-to-PCI protocol converter is operated at 33MHz. Thus the PCI FSM makes a data valid signal for IIS protocol using a counter which counts from 0 to 10.

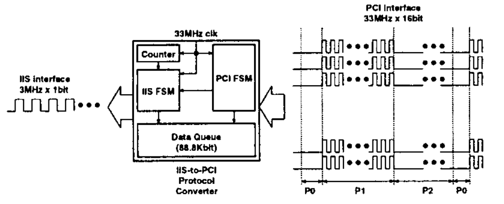
PCI Interface 33MHz x 16bit
33MHz clk
Counter
IIS FSM
PCI FSM
IIS interface 3MHz x 1bit
Data Queue (88.8Kbit)
IIS-to-PCI Protocol Converter
P0
P1
P2
P0

Figure 13. IIS-to-PCI protocol converter: decoded bit stream is connected to DAC using IIS protocol which is 1bit serial output and is received from software using PCI protocol which consists of three phases. $P0$-phase is required to initiate PCI transaction, $P1$-phase is used to send decoded data, and $P2$ is the time for MP3 software decoding of new IIS data.

6 Conclusion

Integrating IP's that are developed by different vendors is an important issues in the SoC design, as the different design environment produces different interface protocols and operating frequencies. For the exploration of large design space in SoC design, the automatic interface synthesis is one of the most important part. The approach proposed in this paper can be useful to a designer not only to make automatic interface synthesis but also to consider multi-clock operating environment. This approach can be started with the operating frequency and queue size determined. Then the input FSM is modified according to the input and output path combinations. Finally the conventional interface synthesis is performed. This approach was successfully applied to Virtual Chip System in order to verify the IIS-to-PCI protocol converter on a real hardware system.

References

[1] Semiconductor Industry Association, San Jose, CA. *International Technology Roadmap for Semiconductors 1998 Update*, 1998.

[2] K.Okumura. A Formal Protocol Conversion Method. In *ACM Symposium on Communications, Architectures, and Protocols(SIGCOMM)*, pages 30–37, Aug. 1986.

[3] S.S.Lam. Protocol Conversion. *IEEE Transactions on Software Engineering*, 14(3):353–362, Mar. 1988.

[4] K.L.Calvert and S.S.Lam. Formal Methods for Protocol Conversion. *IEEE Journal on Selected Areas in Communications*, 8(1):127–142, Jan. 1990.

[5] J.A.Nestor and D.E.Thomas. Behavioral Synthesis with Interface. In *IEEE/ACM International Conference on CAD*, pages 112–115, Nov. 1986.

[6] G.Borriello. *A New Interface Specification Methodology and its Applications to Transducer Synthesis.* Ph.D. dissertation, Univ. of California, Berkeley, 1988.

[7] J.Akella and K.McMillan. Synthesizing Converters between Finite State Protocols. In *IEEE International Conference on Computer Design*, pages 410–413, Oct. 1991.

[8] J.S.Sun and R.W.Brodersen. Design of System Interface Modules. In *IEEE/ACM International Conference on CAD*, pages 478–481, Nov. 1992.

[9] S.Narayan and D.D.Gajski. Interfacing Incompatible Protocols using Interface Process Generation. In *IEEE/ACM Design Automation Conference*, pages 468–473, Jun. 1995.

[10] G.Borriello and R.H.Katz. Synthesis and Optimization of Interface Transducer Logic. In *IEEE/ACM International Conference on CAD*, pages 274–277, Nov. 1987.

[11] R.Passerone and J.A.Rowson. Automatic Synthesis of Interfaces between Incompatible Protocols. In *IEEE/ACM Design Automation Conference*, pages 8–13, Jun. 1998.

[12] N.Kim, H.Choi, S.Lee, I.-C.Park, and C.-M.Kyung. Virtual Chip: Making Functional Models Work on Real Target Systems. In *IEEE/ACM Design Automation Conference*, pages 170–173, Jun. 1998.

[13] C.-J.Park, S.Lee, B.-I.Park, H.Choi, J.-G.Lee, Y.-I.Kim, M.-K.Jung, I.-C.Park, and C.-M.Kyung. Early In-System Verification of Behavioural Chip Models. In *IEEE International High Level Design Validation and Test Workshop*, pages 61–65, Nov. 1999.