A Scheduling Method for Synchronous Communication in the Bach Hardware Compiler

Ryoji Sakurai[1], Mizuki Takahashi[1], Andrew Kay[2], Akihisa Yamada[1], Tetsuya Fujimoto[1], and Takashi Kambe[1]

1 Design Technology Development Center,
IC Group, Sharp Corporation
2613-1 Ichinomoto-cho, Tenri-shi, Nara JAPAN
E-mail: sakurai@edag.ptdg.sharp.co.jp

2 Sharp Laboratories of Europe
Edmund Halley Road, Oxford Science Park, Oxford
OX4 4GA United Kingdom
E-mail: akay@sharp.co.uk

Abstract – In this paper, we propose a scheduling method for synchronous communication between threads in the Bach hardware compiler. In this method, all communications are extracted from a behavioral Bach-C description and statically prescheduled to synchronize communications between threads if possible. Then all the operations and communications of each thread are synthesized independently according to the prescheduling result. Consequently, we can synthesize large system LSIs efficiently, because we do not need to synthesize the whole system descriptions at once to synchronize communications.

Experimental results show that our method improves throughput of synthesized circuits and is applicable to large systems designed with the Bach hardware compiler.

1 Introduction

In order to handle large-scale system design with increasing size and complexity, many behavioral level synthesis techniques have been proposed[1][2]. Some of them are now embedded in commercial CAD tools, and practically used[2]. They release a designer from detailed timing and control logic design, then the designer can concentrate on the essential algorithm and architecture design. However, known behavioral level synthesis techniques are limited to handling single data-flow. In other words, they cannot handle a system that consists of many subsystems communicating with each other. The integration of many subsystems is still a major problem for design automation.

We have proposed the Bach hardware compiler[5] to describe and to synthesize whole systems, as discussed above. The Bach hardware compiler realizes successful communications between subsystems by synthesizing handshake circuitry. Then the system synthesis task is partitioned into smaller behavioral level synthesis tasks, each of which can be handled by conventional high-level synthesis tools.

This strategy works well in the case of a system with coarse granularity where the system consists of large subsystems and loose communication channels. On the other hand, when a system consists of many tightly coupled subsystems, the cost and performance of the synthesized circuit will be dominated by the implementation of communication protocols. This is caused by generating the handshake circuitry to implement communications. Although it guarantees successful communication, additional circuitry and clock cycles are necessary.

In this paper, we propose a scheduling method to determine the communication timings between subsystems statically in the whole system. Using this method, each communication is implemented by a simple interconnection to reduce the communication circuit cost and clock cycle, and each partitioned circuit is synthesized independently.

This paper is organized as follows. In the next section, we introduce the Bach hardware compiler and its source language Bach-C. In Section 3, a *synchronous communication scheduling* problem is defined, and its efficient solution is proposed. A model and algorithm for *communication operation scheduling* problem are shown in section 4. We give experimental results and show the effectiveness of our proposed technique in Section 5. In the last section, we make concluding remarks.

2 The Bach Hardware Compiler

In the conventional LSI design flow, system algorithms are often implemented using programming languages C or C++, then the equivalent functionality is implemented using HDLs such as VHDL or Verilog-HDL. This means that different descriptions for the same function must be written and verified twice, which often causes long design period. To cope with the issue, we developed the Bach hardware compiler. Bach's source language, Bach-C, is based on ANSI C with some extensions to describe hardware algorithms. Bach hardware compiler enables us to design a large-scale system with Bach-C, verify the system description using Bach-C simulator, and also synthesize a circuit from the same description.

2.1 The features of Bach-C

Bach-C supports most constructs in ANSI C, and adds a few more constructs which are specially tailored to hardware description and simulation.

Bach-C has **par** statement to support explicit

```
                chan int ch;
par {           par {
  a = b * c ;     send(ch, send_data);
  d = e * f ;     receive_data=receive(ch);
}               }
```

(a) par syntax (b) synchronous channel

Figure 1. Examples of Bach-C description

parallelism (concurrency). Bach-C also provides **chan** declaration for synchronous communication, as in CSP[3] or occam[4]. Using **par** and **chan**, we can describe the behavior of a complete system including communication between parallel subsystems.

Figure 1 shows examples of the **par** statement. In Figure 1(a), two statements a=b*c and d=e*f are executed concurrently. Figure 1(b) shows an example of synchronous communication. Channel variables are first declared with the keyword **chan**. Each channel transfers data of a given type from one thread to another synchronously. The sender uses the function **send(**ch,send_data**)** to send the value *send_data* down a channel *ch*. The receiver uses **receive(**ch**)** to receive the value from channel *ch*. Both sender and receiver must be ready in order for the transfer to occur. If either a sender or a receiver is not ready, the other one waits it.

The synchronous communication in Bach-C supports only one-to-one and one way communication. We refer to a pair of the send and receive operations using one channel as *communication pair*.

2.2 System designing with the Bach

When designing large-scale systems, it is very important to decide how subsystems communicate each other. If the protocol is inconsistent between them, the behavior of the whole system will be incorrect. Most of conventional behavioral synthesis tools are limited to handling single data-flow. This means that they cannot handle system descriptions composed of communicating subsystems. When we design large systems which consists of many tightly coupled subsystems using conventional tools, we must flatten these subsystems into one unified system if we want to synthesize interactions between subsystems. Otherwise, we must design each subsystem separately keeping consistency of communications between subsystems.

To release designers from these problems, Bach generates handshake circuitry for communicating between *thread*s (subsystems). Because handshake communication dynamically determines the timing at run-time, we do not need to schedule each thread in consideration of communication timing. In other words, we can synthesize each thread independently. Therefore, Bach can divide a full system synthesis into smaller behavioral synthesis problems, and guarantee communications by handshaking.

3 Synchronous communication scheduling

When we design a system consists of many tightly coupled subsystems with the Bach, sometimes handshake communication becomes critical issues in terms of cost and performance. These problems are caused by over constraint (too much safe assumption) that all communications need the *dynamic* synchronization provided by handshaking. In many applications, however, the most communication timings can be *statically* determined. Then we can realize the communications between threads by simple interconnection and can decrease area overhead and increase circuit performance.

In this section, we propose an efficient solution for *synchronous communication scheduling* problem which determines the communication timings between threads and which enables us to synthesize each thread separately.

3.1 Definition of synchronous communication scheduling problem

In this paper, we assume that each channel variable in a Bach-C description is assigned to an individual channel resource in the synthesized circuit. Then, a synchronous communication scheduling problem is defined as a problem to schedule all operations in the whole system description satisfying the following constraints.

C1: *Execution order for communication operations within a thread must be kept.*

C2: *For each communication pair, the send operation must be executed no later than the receive operation.*

C3: *Two communication operations must be scheduled to guarantee the data dependency between them.*

C4: *Two send (receive) operations on the same channel within a thread must be scheduled at different cycles.* (Remark: Send and receive operations are not executed on the same channel within a thread, since Bach-C supports only one way channel as mentioned in Section 2.1.)

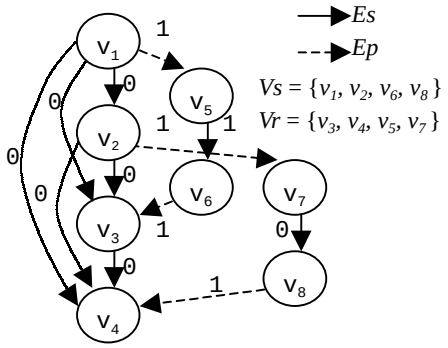
This scheduling problem can be solved in this way: first join several CDFGs corresponding to each thread into one large unified CDFG by making edges for each communication pairs, and schedule this unified CDFG all at once. However, this simple method may not work for large designs, because the unified CDFG may exceed the capacity of behavioral synthesis tools.

3.2 An efficient solution for synchronous communication scheduling problem

We propose an efficient scheduling method to

```
par {
  /* thread 1 */
  while(1){
    send(ch1, a);    /* v1 */
    send(ch2, b);    /* v2 */
    c = receive(ch3);/* v3 */
    d = receive(ch4);/* v4 */
  }
                    /* thread 2 */
                    while(1){
                      e = receive(ch1);/* v5 */
                      send(ch3, f1(e));/* v6 */
                    }
                    /* thread 3 */
                    while(1){
                      f = receive(ch2); /* v7 */
                      send(ch4,  f2(f));/* v8 */
                    }
                  }
```

(a) Bach-C



(b) Communication Graph

Figure 2. Examples of Bach-C description and generated communication graph

determine the communication timings. The outline of our algorithm is as follows:

Step1: For each thread, select all communications whose timing can be statically determined.

Step2: Extract execution order of the communication operations within each thread.

Step3: Find the communication pair in the whole threads.

Step4: Estimate execution cycles required between the communication operations within a thread.

Step5: Determine timing of all the selected communications to satisfy constraint **C1** to **C4** for synchronous communication (we call this step *communication operation scheduling*).

Step6: Synthesize all operations and communication operations in each thread based on the result of **Step5** using a conventional behavioral synthesis tool.

In **Step1**, we select communications whose timing can be statically determined. The other communications should be implemented by handshaking. In this way, we solve the synchronous communication scheduling problem by dividing into small behavioral synthesis tasks.

In the next section, we focus on communication operation scheduling executed in **Step5**, and explain the details of the algorithm.

4 Communication operation scheduling

In this section, we define the *communication graph* for modeling **C1** to **C4**. Then we show the algorithm for communication operation scheduling problem.

4.1 Communication graph

A communication graph, $G=(V, E)$, is to represent **C1** to **C4** which are obtained from a Bach-C description. Here V is a node set and $E \subseteq (V \times V)$ is a directed edge set.

Each node $v_i \in V$ corresponds to a communication operations (send or receive operation) in the Bach-C description. V is partitioned into two subsets Vs and Vr such that $V = Vs \cup Vr$ and $Vs \cap Vr = \phi$. Vs is a set of all send operations and Vr is a set of all receive operations.

A directed edge $e_{ij} = (v_i, v_j) \in E$ represents an execution order on v_i and v_j. E is partitioned into two subsets Es and Ep such that $E = Es \cup Ep$ and $Es \cap Ep = \phi$. If e_{ij} belongs to Es then v_j is executed after v_i, and v_i and v_j are in the same thread. This edge set Es denotes the **C1**. If e_{ij} belongs to Ep then $v_i \in Vs$ and $v_j \in Vr$ is a communication pair. This edge set Ep denotes the **C2**.

Each edge e_{ij} has an associated nonnegative integer C_{ij} used as scheduling constraint. C_{ij} represents **C3** and **C4**, that is, v_j must be executed at least C_{ij} cycles after v_i. Each C_{ij} is calculated as follows.

If e_{ij} belongs to Ep then C_{ij} is always 1. Here, we assume that a generated circuit has latched output port for each send operation. Thus a receive operation must be executed at least 1 cycle after the corresponding send operation.

If e_{ij} belongs to Es and there exists data dependency between nodes v_i and v_j, C_{ij} denotes the number of cycles required to execute non-communication operations between v_i and v_j, which represents **C3**. We estimate C_{ij} by performing an initial scheduling in each thread. If e_{ij} belongs to Es and nodes v_i and v_j use the same channel, C_{ij} is 1, which represents **C4**. Otherwise, C_{ij} is 0.

In Figure 2, we show examples of a Bach-C description (a) and a communication graph (b) which is generated from (a). For clarity, the node names corresponding to communication operations are annotated in Figure 2(a) using comments. The value along edge e_{ij} denotes the corresponding C_{ij}. We suppose that functions `f1` and `f2` are defined elsewhere.

We consider nodes v_1 and v_5. There is an edge $e_{15} \in Ep$ between them. Then C_{15} of edge e_{15} is 1. Next, let us consider nodes v_5 and v_6. There is a funciton `f1` between them. Suppose that clock period is 50 ns and function `f1` takes 70 ns. Then C_{56} of edge e_{56} is $\lfloor 70/50 \rfloor = 1$. Note that C_{ij} is 0 if there exists no data dependency between two nodes

v_i and v_j. In Figure 2, for example, C_{12} is 0, since there is no data dependency between node v_1 and v_2.

4.2 Algorithm for communication operation scheduling

As mentioned in Section 3.1, we do not consider channel resource sharing among different channel variables. Therefore, we can use the well known ASAP (As Soon As Possible) scheduling method to determine the execution cycle S_i of node v_i in V. In Figure 3, S (v_i) denotes the scheduled cycle of node v_i. $Pred_{vi}$ denotes a set of nodes that are immediate predecessors of the node v_i.

A scheduling result for the graph from Figure 2(b) is as follows.

$S_1 = S_2 = 1$; $S_5 = S_7 = S_8 = 2$; $S_6 = 3$; $S_3 = S_4 = 4$;

We can handle this scheduling problem for large scale system design, because a target of the scheduling problem is only communication operations. Moreover, each communication timing (scheduling cycles in high-level synthesis) can be fixed for each thread with communication operation scheduling. This shows that we can partition the whole system synthesis problem into small behavioral synthesis problems for synchronous communication scheduling.

```
Procedure CommunicationOperationScheduling (G );
  while V ≠ φ  loop
    foreach v_i ∈ V  loop
      if Pred_vi = φ  then
        S_i = 1;
      else if Pred_vi are already scheduled then begin
        S_i = Max_{vj∈Predvi} (S(v_j) + C_ij);
        V = V – { v_i };
      end;
```

Figure 3. Algorithm for communication operation scheduling

5 Experimental results

We applied the proposed method to several samples and compared synthesis results with those of the handshaking approach. The results are summarized in Table 1. In this table, *Orig* shows the synthesis results using the handshake communication, and *New* shows the results of proposed method. The number in parentheses along each sample name indicates the number of communication operations in the Bach-C description. *Throughput* shows the number of interval clock cycles required to receive each data item.

This table shows that the proposed method can generate a high-speed circuit for all samples, and remove handshake circuitry for all synchronous communications. The speed advantage is due to the following features:

(1) The reduction of the number of cycles required for each communication between threads to half clocks.
(2) The ability to schedule and synthesize a circuit which operate several communication operations concurrently.

These experimental results demonstrate that the presented method improves both cost and performance of generated circuit.

		Gate count (gates)	Through-put (cycles)
Sample1 (10)	Orig	2,382	8
	New	1,580	4
Sample2 (18)	Orig	9,130	10
	New	8,188	2
Sample3 (32)	Orig	135,534	32
	New	113,105	3

Table 1. A comparison with handshake circuitry

6 Conclusion

In this paper, we have proposed a synchronous communication scheduling method in the Bach hardware compiler. In this method, the communication timings among subsystems are statically determined. Then synchronous communications are realized by just an interconnection. Unlike communication using handshake circuitry, we can guarantee the communication without sacrifice in circuit overhead and clock cycles. In particular, our method is effective for large system design, since the method partitions the whole system synthesis problem into small behavioral synthesis problems.

Experimental results show that the throughput and area of synthesized circuits are improved using our method, and that the Bach hardware compiler has become applicable to communication intensive applications.

References

[1] D. Gajski, A. Wu, N. Dutt and S. Lin, “High-level Synthesis: introduction to Chip and System Design,” Kluwer Academic Publishers, 1992

[2] “Behavioral Compiler User Guide,” version 1997.01, Synopsys, 1997

[3] C.A.R. Hoare, “Communicating Sequential Processes,” Prentice-Hall, 1985

[4] INMOS Ltd, “occam2 reference manual”, Prentice-Hall International, 1988

[5] K. Nishida, A. Kay, A. Yamada, T. Kambe and T. Nomura, “Bach Hardware Compiler for System Level Synthesis,” IEICE Technical Report, CPSY97-87 (In Japanese), 1997