9N−7

VHDLの記述による、パイプライン回路の自動合成

徐 行倹 石塚 満

東京大学

1. INTRODUCTION

In the design automation of ASIC with the behavioral specifications written in programming languages like VHDL, it is possible to create a pipelined circuit for pipeline iteration statements. Some thesis [1], [2] present scheduling technologies of pipeline circuits synthesis. But they do not explain the way to extract and create the pipeline from behavioral specifications. Some pipeline technologies have been well developed for general purpose compiler [3]. However, most of them are based on so-called static data flow model. In this paper, we present a methodology, which is able to synthesis a pipeline processing circuit from its behavioral specification written in VHDL, which iteration range is not computable during compile-time. Mainly the following statements to appear in the VHDL language are discussed in this paper:

- ***while*** iteration;
- ***exit***, which breaks an iteration;
- ***if-then-else*** and
- ***next***.

Section 2 discusses the creation of pipeline from an iteration statement. Section 3 presents a way to process the ***exit*** statement. Section 4 and 5 are about the discussions of ***if-then-else*** and ***next*** statements, and in section 6, our future research projects are introduced.

2. ITERATION

In VHDL, there are three kinds of iterations: ***for***, ***loop*** and ***while***. Since these forms can be mapped from one to another easily, we use only ***while*** statement to represent iterations in this paper in the format of:

while condition ***loop*** Label
  statements;
end loop Label;

where statements within the ***while*** statement, or iteration body, are repeated until the **iteration condition** becomes false. If the range of iteration is computable during compile-time, the control signals to activate and stop the pipeline can be created easily in the form of $F^{n1}T^{n2}F^{n3}T^{n4}$, where, T^n and F^n mean true and false signals, respectively, to be remained for n clocks. By applying such control signal sequences to the ***enable*** ports of the functional units of a pipeline circuit, the pipeline process can be activated or inactivated. But in some cases, the ranges of iteration are not computable during compile-time. Accordingly, the iteration condition has to be checked every clock to control the pipeline processing dynamically. Figure 1 illustrates an example of a pipeline data-path with five pipeline stages, which have a latency of one clock.

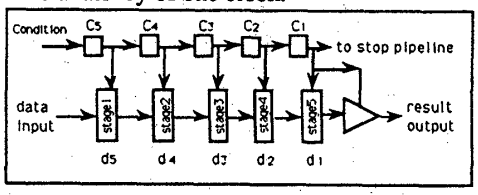

Fig. 1 The control of iteration

An one-bit register is allocated for each stage. Its output port is connected to data-path of corresponding stage for the necessary usage. One status of iteration condition is calculated per clock, and is propagated with its instance, so that the pipeline can be controlled dynamically.

3. EFFECT OF EXIT

block	statements	condition
	while C1 loop L1	
1	S1;	C1 and $\overline{Ce}$
2	exit L1 when Ce;	C1 and Ce
3	S2;	C1 and $\overline{Ce}$
4	end loop L1;	

Table 1 Execution conditions

Pipelined Circuit Synthesis From Its Specification in VHDL

Xing-jian XU Mitsuru ISHIZUKA

University of Tokyo

Table 1 lists a simple ***while*** iteration with some sub-blocks in it, and all the execution conditions of these sub-blocks. When the condition of***exit*** in while statement, C_l AND C_e, becomes true, the ***exit*** statement is executed, and as a result of that, the iteration is braked, or the pipeline is stopped in other words. The condition keeping the pipeline running is C_l AND NOT C_e instead of C_l only. The condition of an ***exit*** statement is propagated to the outer block specified in it, so that the calculation of pipeline condition is taken over the iteration condition and the ***exit*** conditions.

4. IF STATEMENT

An *if* statement can be written as:

if C ***then*** S1;
else S2;
end if

As S1 and S2 have different delays, inserting shift registers to make S1 and S2 have the same delay is a common way in pipeline circuit design.

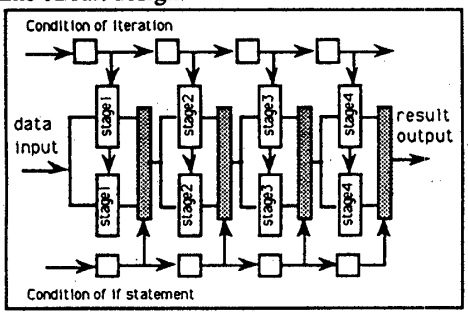

Fig. 2 A pipeline circuit for *If* statement

The way to synthesis a data-path for ***if-else*** statement is to synthesis the data-paths for both TRUE and FALSE cases, and choose the correct result according to the status of the condition. Since S1 and S2 are not executed simultaneously, sharing the same hardware resources for these two data-paths have to be considered when the delays of two data-paths are balanced. An example of an ***if-else*** statement with five pipeline stages is illustrated in Fig. 2. The status of the condition of *if* statement is propagated with its data instance.

5. NEXT STATEMENT

Table 2 shows a ***while*** iteration with a ***next*** statement in it. When Ci is true, the ***next*** statement will be executed. As a result of executing the ***next*** statement, the flow of the program jumps from the ***next*** statement to the beginning of ***while*** iteration instead of S2. When Ci is false, ***next*** statement will not be executed, but S2. It is clear that the ***next*** statement in a ***while*** iteration can be converted into an equivalent *if* statement like:

while C_l ***loop*** L1
S1;
if not C_i ***then***
S3;
end if
end loop L1;

block	statements	condition
	while C1 loop L1	
1	S1; if C1 then	C1
2	next L1; end if	C1 AND C1
3	S2;	C1 AND $\overline{C1}$
	end loop L1;	

Table 2 ***while*** iteration with ***Next*** statement

Unlike the ***exit*** statement propagates its execution condition to the outer blocks it specified, the ***next*** statement has an effect only on the statements from the one following the ***next*** to the end of the block the ***next*** belongs to.

7. FUTURE RESEARCH

We have introduced a methodology to create a pipelined circuit from its behavioral specification without any restrictions. If there is no ***else*** statement of ***if-then-else*** statement introduced in section 5, the path for the false condition, consists of all delay components. It means that in some special cases, the pipeline technologies do no always improve the performance of a circuit. Such research will be continued. Our researches in the near future are listed as follows:

- Determine the number of clocks per stage;
- Minimization of shift registers;
- Performance evaluation of pipeline circuit.

REFERENCES

[1] Cheng-Tsung Hwang etc.: Scheduling for Functional Pipeline and Loop Winding. 28th ACM/IEEE DA Conf. pp. 764-769, 1991

[2] C. Y. Roger Chen etc.: Data Path Scheduling for Two-Level Pipeline. 28th ACM/IEEE DA Conf. pp. 603-606, 1991

[3] Guang R. Gao: A Code Mapping Scheme for Dataflow Software Pipeline. Kluwer Academic Publishers, 1991