Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

A New Approach to Synthesizing Pipelined Control Paths for Performance Optimization

Jer Min Jou and Shainn Rong Kuang

Department of Electrical Engineering
National Cheng Kung University
Tainan, Taiwan, R.O.C.

Abstract

This paper proposes a new and efficient approach to synthesizing the pipelined control paths, by which the performance of control path can be significantly improved at the expense of latency. The proposed approach first transforms the original control specification into a pipelinable intermediate control specification. And then, traditional control path synthesis is used to synthesize a pipelinable control path. Finally, the combinational logic in the pipelinable control path is pipelined to obtain a pipelined control path. In theory, the shortest clock cycle time of control path achieved by this approach is the time of one gate delay in the control path if available hardware resources are unlimited. Experimental results demonstrate that the proposed approach can effectively improve the performance of the control path.

I. Introduction

High performance is always the goal of ASIC design. To improve performance, more and more components are integrated into the VLSI circuit and the complexity of the circuit is rising so quickly that its synthesis needs automatic and starts from a high level. For simplicity, where synthesized from the behavior, a digital circuit is usually divided into two parts, data path and control path. In the previous years, many performance-driven efforts have been put on the pipelined data path synthesis. However, the situation is changing. Now a lot of circuits have the ability to execute multi-functions and are executed in parallel. Thus enlarges and complicates the control path. The complexity of control path may also be increased with choice of high clock frequency in the synthesis of a high performance system. Higher clock frequency may decrease the dead-time in a control state, but may increase the number of states required to realize the schedule. Thus the control path becomes more complex when the clock frequency becomes higher. Sometimes the control path will become the critical part and the clock cycle time becomes shorter than the control path execution time. To achieve high performance, the control path must be pipelined with itself and with the data path to meet the clock cycle constraint. So, not only the data path but also the control path must be pipelined.

Work about pipelined control path synthesis is few in the literatures. Conventional control path synthesis systems (e.g. [1, 2]) do not consider to synthesize pipelined control paths. They synthesize only serial control paths which are executed in serial in the same clock cycle with the data paths. In [3], Prabhu and Pangrle proposed a superpipelined control and data paths synthesis system which uses a modifiable clock cycle time and a pipelined control style to improve the throughput of the circuit. However, how to synthesize pipelined control paths is not clearly mentioned in it.

In this paper, we propose a new approach to pipelining the execution of control path. In it, two types of pipelined control path synthesis are supported:

(1) *Stage-Fixed Pipelining*: Given the fixed pipelined stage number k, find the k-stage pipelined control path with the smallest clock cycle time.

(2) *Time-Constrained Pipelining*: Given the maximal allowable clock cycle time, find the cheapest pipelined control path.

The time-constrained pipelining synthesis of control path finds its application in real time digital signal processing where the sample rate dictates how fast a data must be processed. The stage-fixed pipelining synthesis, on the other hand, is applicable to high performance applications with resource or latency constraints. The approach first transforms the original control specification, which is derived from the results of data path synthesis and is modeled as a cyclic behavioral state transition graph BSTG, into a pipelinable intermediate control specification. And then, the combinational logic synthesized from the pipelinable control specification is pipelined to obtain a pipelined control path. In theory, the shortest clock cycle time of control path achieved by the approach is the time of one gate delay in the control path if available hardware resources are unlimited. Experimental results demonstrate that the approach can effectively improve the performance of control path.

The rest of this paper is organized as follows. Basic concepts and definitions for pipelined control path synthesis are presented in Section II. Section III states the problems of pipelined control path synthesis and their

solutions. Section IV describes the synthesis algorithm of the pipelined control path. Experimental results are presented in Section V. Finally, conclusions are drawn in Section VI.

II. Preliminaries

The control path synthesis usually is carried out from a control specification after the data path synthesis. In this section, we introduce the BSTG which is used to model the control specification. In addition, the traditional serial control path synthesis and the basic idea of pipelining the control path are also explained.

Note that storage elements in the synthesized control path in this paper are classified into four categories: state registers *SR*, status registers *UR*, control registers *CR*, and pipelined registers *PR*. The state registers are used to keep track of the control state. The purpose of introducing status registers into the control path is to store the control conditions from the data path for a design with conditional branches. Control registers and pipelined registers only exist in pipelined control paths. They are designed to hold the control signals and the intermediate signals between stages of the pipelined control path, respectively.

2.1 Control Specification

The control specification derived from the result of data path synthesis is represented by a behavioral state transition graph BSTG. Fig. 1 illustrates the general concept of BSTG. Fig. 1(a) and Fig. 1(b) show a scheduled control data flow graph SCDFG and its corresponding BSTG, which uses vertices for states S_i and edges for state transitions. The vertex labels in each state are the condition-operation pairs [c:ε] which denote that the set of operations ε will be executed if control condition c is true. Note that the condition may be null. The edge labels in BSTG are the control conditions. The control conditions c1, c2, and c3 in Fig. 1(b) are produced by operations >1, >2, and >3, respectively. If there is an edge $S_i \rightarrow S_j$ in BSTG, then state S_j is called the *child state* of S_i, and S_i with more than one child state is a *branch state*. Each time only one of the child states will be reached. The decision of which child state is chosen to reach is taken according to the control condition attached to the corresponding edge. Following the basic notations above, we have some definitions.

Definition 1: If there is a path from state S_i to state S_j in the BSTG, then S_i is an *ancestor* state of S_j and S_j is a *descendant* state of S_i for the path.

Definition 2: If S_i is an ancestor state of S_j in the BSTG, then the *distance* between S_i and S_j, denoted as D_{ij}, is the number of states between S_i and S_j.

Definition 3: If control condition c is produced by the data path at state S_i, then we say S_i *produces* c. If state S_j needs c to determine its next state or operations to be executed, then we say S_j *consumes* c. If c is produced and consumed at S_i and S_j, respectively, then state pair $<S_i, S_j>_c$ is a produce-consume pair associated with c.

2.2 Serial Control Path Synthesis

Traditional serial control path synthesis from a BSTG is classified into two phases: register-transfer (RT) level synthesis and logic level synthesis. *RT level synthesis* is to generate a state table from the BSTG as the input of logic level synthesis. To generate a state table, the inputs and outputs of control path at each state must first be determined. The inputs $i(t)$ of control path at state $s(t)$ are determined based on the results of status register allocation. The output control signals $o(t)$ activated by control path at state $s(t)$ can be obtained by combining the binding information of data path and the respective RT level network. After the state table is generated, *logic level synthesis* performs state minimization, state assignment, and logic optimization to get the final control path. Fig. 2 shows the structure of a serial control path. The state registers *SR* holds the present state, and a combinational logic *CL* decides the next state and output control signals. The state transition function δ and output function ρ satisfy

$$s(t+1) = \delta [\, i(t), s(t)] \quad (1)$$

$$o(t) = \rho [\, i(t), s(t)] \quad (2)$$

2.3 Basic Idea of Pipelining Control Path

The *CL* of a control path can be partitioned into two parts: state transition logic *SL* and output logic *OL* as shown in Fig. 3(a). *SL* solely implements Eq.(1) whereas *OL* generates control signals according to Eq.(2). Obviously, the key problem of pipelining control path is how to pipeline the circuit cycles passing through *SL* and *SR*. Let ϕ_π denote the critical path delay of a circuit π. The way is explained as follows.

First, *SL* and *SR* of the control path are doubled as shown in Fig. 3(b). The new *SL'* consisted of two *SL*s and one *SR* is a two-stage pipelined circuit. If both copies of *SR* are initialized to the same state values, the structure of Fig. 3(b) implements the same behavior as the structure of Fig. 3(a). Although the state transition logic of Fig. 3(b) has been pipelined, the clock cycle time of control path is not reduced. To reduce the cycle time, we move *SR* in the *SL'* out as *SR'*. For preserving the correct state sequence, the input $i(t)$ of the second *SL* is changed into $i(t+1)$. The result is shown in Fig. 3(c). The primary inputs of the formed combinational logic *SL"* are

$i(t)$, $i(t+1)$, and $s(t)$, and its primary outputs are δ [$i(t+1)$, δ [$i(t)$, $s(t)$]]. Then we fuse two *SLs* in *SL"* together into a new circuit λ whose delay is usually less than $2*\phi_{SL}$ (see Fig. 3(d)), partition it into 2 parts, and move *SR'* back into the partitioned λ to form λ' (see Fig. 3(e)). Finally, we directly pipeline *OL* into two stages by inserting *PR* and *CR* into it. A two-stage pipelined control path is formed (see Fig. 3(f)), whose delay is less than ϕ_{CL}.

The above example elucidates the basic idea of how to pipeline a cyclic control path to reduce its delay. Given a structural control path, the basic steps of synthesizing a two-stage pipelined control path described above can be summarized and extended to synthesize a k-stage pipelined control path, denoted as *k-PCP*, as follows:

1. derive the state table of a combinational logic with primary inputs $i(t)$, $i(t+1)$, $i(t+2)$, ... , $i(t+k-1)$, and state inputs $s(t)$, and state outputs δ [$i(t+k-1)$, δ [$i(t+k-2)$,..., δ [$i(t+1)$,δ [$i(t)$,$s(t)$]]...]] and control outputs ρ [$i(t)$, $s(t)$], then synthesize it; this synthesized circuit is called a k-factor combinational logic Γ_k.
2. pipeline the Γ_k into a k-stage pipelined circuit Ω_k, and then link Ω_k with control and state registers to form a *k-PCP*.

Fig. 4 shows the basic structure of a *k-PCP* synthesized by the above process. It consists of Ω_k, *SR*, and *CR*. And Ω_k consists of k combinational logics (CL_1, CL_2, ..., CL_k) with stage j communicating with stage $j+1$ through pipelined registers PR_j. Let δ_j and ρ_j represent the state transition and output functions of combinational logic CL_j in it, and In_t^{t+k-1} denote the set of inputs $i(t)$, $i(t+1)$, $i(t+2)$, ... , and $i(t+k-1)$. The state transition function and output function of the *k-PCP* satisfy

$$s(t+k) = \delta_k[\delta_{k-1}[...\delta_2[\delta_1[In_t^{t+k-1}, s(t)]]...]] \quad (3)$$

$$o(t+k) = \rho_k[\rho_{k-1}[...\rho_2[\rho_1[i(t),s(t)]]...]] \quad (4)$$

Since the Ω_k of *k-PCP* is obtained by pipelining Γ_k into k stages, thus the $\delta_k[\delta_{k-1}[...\delta_2[\delta_1[In_t^{t+k-1}, s(t)]]...]]$ and $\rho_k[\rho_{k-1}[...\rho_2[\rho_1[i(t),s(t)]]...]]$ will be identical with the δ [$i(t+k-1)$, δ [$i(t+k-2)$, ... , δ [$i(t+1)$, δ [$i(t)$,$s(t)$]]...]] and ρ [$i(t)$, $s(t)$] of Γ_k, respectively. That is, the state transition function and output function of the *k-PCP* satisfy

$$s(t+k) = \delta\,[i(t+k-1), \delta\,[i(t+k-2),..., \delta\,[i(t+1), \delta\,[i(t),s(t)]]...]] \quad (5)$$

$$o(t+k) = \rho\,[i(t), s(t)] \quad (6)$$

Eq.(5) and Eq.(6) state that the state sequence and output sequence of the *k-PCP* are identical with those of the serial control path. The difference between the behaviors of *k-PCP* and serial control path is the time of sending the control signals to data path. In the serial control path, the control signals ρ [$i(t)$, $s(t)$] are activated and sent to data path at state $s(t)$. In the *k-PCP*, however, the control signals ρ [$i(t)$, $s(t)$] are activated at state $s(t)$ but sent to data path at state $s(t+k)$, this situation is called *output-delay k steps*. Moreover, the serial control path at state $s(t)$ needs input $i(t)$, but the *k-PCP* at state $s(t)$ needs inputs In_t^{t+k-1}, this is called *input-borrow k steps*.

III. Problem Statement and Analysis

Section 2.3 has introduced the process of synthesizing a *k-PCP* from a structural serial control path. However, sometimes the control dependency is not satisfied by the *k-PCP* due to the features of input-borrow and output-delay k steps. *Control dependency* is the timing relation between states caused by the production and consumption of control conditions. Consider the BSTG shown in Fig. 1(b). Let $CS(\varepsilon)$ denote the required control signals for executing data path operations ε. The *2-PCP* synthesized from the BSTG of Fig. 1(b) will activate the control signals CS(>2, -2) at state S_2, but send CS(>2, -2) to data path at state S_4 or S_7 due to output-delay 2 steps. That is, state S_4 or S_7 produces c2≡>2. On the other hand, c2 is consumed by the *2-PCP* at state S_3 due to input-borrow 2 steps, and state S_3 (the consuming state) is reached before state S_4 or state S_7 (the producing state). Therefore, the control dependency is violated.

Fortunately, we can modify the original BSTG by inserting no operation states, NOOPs, into it to ensure that the synthesized *k-PCP* satisfies control dependencies. Although control path is idle when it is at NOOP, inserting NOOPs into the BSTG doesn't destroy its original behavior. A *k-PCP* is *well-behaved* if it satisfies all control dependencies.

Definition 4: A BSTG is *k-well* if a well-behaved *k-PCP* can be derived from it. Otherwise, it is *k-ill*.

Definition 5: The *k-step descendant set* of state S_i in the BSTG, denoted as $\theta_{i,k}$, is the set of descendant states of state S_i whose distance with respect to S_i is equal to k-1.

The following Theorem states the conditions that a k-well BSTG must satisfy.

Theorem 1: A BSTG is k-well if the distance D_{ij} of states in each produce-consume state pair $<S_i, S_j>_c$ satisfies one of the following conditions:

Condition 1: if S_j is not a branch state, then $D_{ij} \geq k$.

Condition 2: if S_j is a branch state, then $D_{ij} \geq 2k - 1$.

proof: The Theorem is briefly proved as follows. In the BSTG, the control condition c is produced at S_i and consumed at S_j for $<S_i, S_j>_c$. However, in the *k-PCP*, c is

produced at the state $S_n \in \theta_{i,k}$ due to output-delay k steps. Moreover, let $\sigma_{j,k}$ denote the set of ancestor states of state S_j in the BSTG whose distance with respect to S_j is equal to k-1. If S_j is a branch state in BSTG, then in k-*PCP* c is consumed at the earliest at the state $S_m \in \sigma_{j,k-1}$ due to input-borrow k steps. Otherwise, c is consumed at the earliest at state S_j. In order to keep the state S_n which produces c is reached before the state S_m or S_j which consumes c in k-*PCP*, D_{ij} must be larger than or equal to $2k$-1 (k) if S_j is (not) a branch state. As a result, if Condition 1 or Condition 2 is satisfied, the BSTG must be k-well. Q.E.D.

By Theorem 1, a k-ill BSTG can be made k-well by inserting some NOOPs between produce-consume state pair to increase the distance between the producing state and the consuming state. Consider the BSTG in Fig. 1(b), the BSTG is 2-ill. We can insert four NOOPs N_1, N_2, N_3, and N_4 into it to make it 2-well as shown in Fig. 5.

By the above explanation, the process of synthesizing a k-*PCP* can be revised as follows.

1. make the original BSTG k-well by inserting NOOPs into it according to Theorem 1, and then derive the behavioral specification, called a k-factor BSTG, of k-factor combinational logic Γ_k from the k-well BSTG, and synthesize it;
2. pipeline the Γ_k into the k-stage pipelined circuit Ω_k, and then link Ω_k with control and state registers.

The two steps will be explained in detail in next section.

```
Algorithm PCP_Synthesis(BSTG, k, T)
  if ( k=0 ) {
    Synthesize the serial control path and get its delay;
    k = ⌈φ_CL/T⌉;  }
  while( true ) {
    if (BSTG is k-ill)  Make BSTG k-well;
    Construct k-factor BSTG for the k-well BSTG;
    Perform state assignment and logic optimization
       from the k-factor BSTG to obtain a Γ_k;
    if ( T=0 )     /* stage-fixed pipelining */
    { Pipeline Γ_k into Ω_k and minimize φ_Ωk; break; }
    else {         /* T>0, time-constrained pipelining */
      Estimate the φ_Γk of Γ_k;
      if ( φ_Γk/k ≤ T) {
        Pipeline the Γ_k into a Ω_k under constraint T,
        if (φ_Ωk > T)  k = k + 1;  else  break;
      }
      else k = k + 1;
    }
  }
  Link the Ω_k with SR and CR to complete the k-PCP;
  return the k-PCP;
end Algorithm;
```

Fig. 6. Algorithm of synthesizing pipelined control paths.

IV. Synthesis of Pipelined Control Paths

The algorithm of synthesizing the pipelined control path is outlined in Fig. 6. The inputs of the algorithm include the original BSTG, pipelined stage number k, and cycle time constraint T. By setting different inputs, it can perform stage-fixed pipelining or time-constrained pipelining optionally. Stage-fixed pipelining is done by setting k>0 and T=0. When k=0 and T>0, time-constrained pipelining is performed. The main steps of the algorithm will be explained in detail in the following subsections.

4.1 Making a k-well BSTG

This section explains the process of making a k-ill BSTG k-well. By Theorem 1, if at least N_{ij} NOOPs will be inserted between each produce-consume pair $<S_i, S_j>_c$ in the k-ill BSTG, then N_{ij} must satisfy the following two equalities:

$$N_{ij} = k\text{-}D_{ij}, \quad \text{if } S_j \text{ is a branch state;} \tag{7}$$

$$N_{ij} = 2k\text{-}D_{ij} - 1, \quad \text{otherwise.} \tag{8}$$

The larger the number of NOOPs inserted is, the more registers and area of the k-*PCP* are. The number of NOOPs inserted must be minimized as much as possible. For each produce-consume pair $<S_i, S_j>_c$ there are at least N_{ij} NOOPs to be inserted according to Eq.(7) or Eq.(8), and an weighted dotted edge $S_i \overset{N_{ij}}{\cdots\!\rightarrow} S_j$ is graphed on BSTG to represent the constraint. Fig. 7 shows an example of BSTG with all its constraint edges, which is transformed from BSTG of Fig. 1(b) for k=2.

Each constraint edge $S_i \overset{N_{ij}}{\cdots\!\rightarrow} S_j$ in the k-ill BSTG corresponds to a path $S_i \rightarrow \cdots \rightarrow S_j$ of the k-ill BSTG. Two constraint edges have *intersection* if their corresponding paths have common edges. The intersection of two constraint edges is the set of the common edges in BSTG. For two constraint edges having an intersection, the intersection is the best position for inserting NOOPs. The minimal number of NOOPs inserted for the two intersected constraint edges is equal to the maximal value of N_{ij}s between the two edges. An integer linear programming (ILP) formulation is proposed to get the optimal solution for the minimal NOOP insertion problem. First, a relation graph $G(V,E)$ for all constraint in BSTG is constructed; each vertex v with weight $\alpha_v = N_{ij}$ of G represents a constraint: $S_i \overset{N_{ij}}{\cdots\!\rightarrow} S_j$ in BSTG, and each edge between vertex i and j represents that there is a intersection between the two vertices. A clique of a graph is a complete subgraph that is not contained in any other complete subgraph of it. Variable ω_i represents the number of NOOPs to be inserted at the intersection among the constraints in the clique i. Let AZ be the set of

all cliques in $G(V,E)$, and Z_v be the set of cliques whose vertex set including vertex v. Then, the ILP formulation for the minimal NOOP insertion problem is listed as follows:

$$\text{Minimize} \quad \sum_{i \in AZ} \omega_i$$

$$\text{Subject to:} \quad \sum_{i \in Z_v} \omega_i \geq \alpha_v \ , \ \forall v \in V \text{ in the } G.$$

The AZ is found by using the approach proposed by Bron and Kerbosch [4]. After the weight ω_i of each clique $i \in AZ$ is determined, the BSTG will be made k-well by inserting ω_i NOOPs into the respective intersection positions.

Although the algorithm applying the ILP formulation has an exponential time complexity for the worst case, the minimal NOOP insertion problem can be quickly solved by it for most cases. This is because First, the number of control conditions in the BSTG are not so many and therefore the number of produce-consume state pairs are also not many. Thus, the number of constraint edges in the BSTG and the vertices in G are typically few. Second, the control conditions usually are consumed very soon after they are produced, thus their lifetimes are short. If the lifetime of a control condition is long, the distance between the corresponding produce-consume state pair may be large enough so that no NOOP is necessarily inserted. As a result, few weighted dotted edges intersect and the cliques in G are few. Therefore, the algorithm based on the ILP formulation can obtain the globally optimal solution in reasonable time.

4.2 Constructing k-factor BSTG and Synthesizing Γ_k

After the BSTG has been made k-well, we then construct the k-factor BSTG, which specifies the behavioral specification of Γ_k. The vertices of k-factor BSTG are identical with the vertices of k-well BSTG. Each edge of k-factor BSTG represents a state transition from S_i to $S_j \in \theta_{i,k}$. The $\theta_{i,k}$ of S_i is found by the depth-first search method. After all edges in the k-factor BSTG are generated, the control conditions for each edge will be found. Since the edge $S_i \rightarrow S_n$ in the k-factor BSTG corresponds to a path $S_i \rightarrow \cdots \rightarrow S_n$, denoted as P_{in}, in the k-well BSTG, let C_{mn} be the control conditions on the edge $S_m \rightarrow S_n$ of the original k-well BSTG, then the control condition, C_{in}, attached to edge $S_i \rightarrow S_n$ of the k-factor BSTG can be calculated by the following equation:

$$C_{in} = \bigwedge_{S_m \rightarrow S_n \in P_{in}} C_{mn} \qquad (9)$$

where $\wedge$ denotes logical AND. Eq.(9) represents the feature of input-borrow k steps of the k-factor BSTG. Fig. 8 shows a 2-factor BSTG example derived from the 2-well BSTG of Fig. 5.

After the k-factor BSTG is constructed, we can derive the state table of Γ_k. Then, Γ_k is synthesized from the state table by performing state assignment and logic optimization using the tools in SIS [8]. Note that the k-factor BSTG is an intermediate specification for synthesizing the k-PCP. Therefore, when performing status register allocation to determine the inputs of Γ_k, the lifetimes of the control conditions in the k-factor BSTG must be analyzed by using the state sequence of the final k-PCP, which is identical with the state sequence of the k-well BSTG.

4.3 Pipelining k-factor Combinational Logic

Now, the final pipelining the k-factor combinational logic Γ_k is performed to obtain the pipelined circuit Ω_k of k-PCP. There are two types of pipeline synthesis: stage-fixed pipelining and time-constrained pipelining. When performing stage-fixed pipelining, the another goal of pipelining Γ_k is to obtain the Ω_k with the smallest clock cycle time. This can be achieved by the following two steps. First, we can convert Γ_k into a initial pipelined circuit by inserting k-1 registers in each primary output of Γ_k. Then, the retiming algorithm in [9] with time complexity $O(|V'|\,|E'|\,log|V'|)$ can be employed to pipeline it to minimize the cycle time, where V' and E' are the collection of combinational logic elements and interconnect wires in the circuit.

When performing time-constrained pipelining, on the other hand, Γ_k is pipelined into k stages for minimizing the number of pipelined registers under cycle time constraint T. The retiming algorithm in [9] also can be used to solve the problem, but it requires $O(|V'|^3\,log|V'|)$ computation steps. It is not practical when $|V'|$ is large.

We have developed a simple and efficient method with time complexity $O(k^2|V'|^2)$ to solve the problem. The method first partitions Γ_k into k stages arbitrarily to obtain a initial Ω_k, then, it iteratively minimizes the clock cycle time of Ω_k or minimizes the number of pipelined registers in Ω_k under cycle time constraint T. The technique of min-cut graph partitioning proposed by Kernighan and Lin [5] is applied. Due to space limitation, the detailed algorithm is omitted here.

V. Experimental Results

We have implemented the pipelined control path synthesis algorithm in C on a Sun/Sparc2 workstation, and applied it to synthesize the control paths for some typical digital filters and benchmarks in high-level synthesis. These examples are sixth-order IIR filter (6-

IIR), 16-point FIR filter (16-FIR), fifth-order elliptic wave filter (5-EWF), differential equation solver (Diff), conditional branches example taken from MAHA (Cond1), and conditional branches example taken from SEHWA (Cond2). The last three benchmarks are with conditional branches. The data paths and initial control specifications of these examples are generated by using the high level synthesizer MASS developed by us [6]. The module library used to synthesize these examples is shown in Table 1. The first four examples and the last two examples are scheduled with 10ns and 20ns clock cycle time, respectively. The initial BSTGs are then derived from the scheduling results. Table 2 shows the data path synthesis results. The column "OPs" and "FUs" denote the number of operations in the CDFG and the number of functional units for realizing the data path, respectively. The column "conditions" and "states" represent the number of control conditions and states in the BSTG, respectively.

The experiment explains the relationship between the pipelined stage number k and the achievable shortest control path cycle time for the examples. During the pipelined control path synthesis process, the state table of each Γ_k is represented by using the KISS [7] format, and then synthesized using the *state-assign* (Nova) routine in SIS [8], optimized using the standard SIS script. Each generated Γ_k is mapped into 2-input NAND gates using the *tech_decomp -a 2* option in SIS. The delay is calculated using the unit-delay model. The results of the stage-fixed pipelining for these examples are shown in Table 3, where the column "pipelined stage number k" represents the stage number of pipelined control path; when k=0 denotes serial control. The row "$\phi_{k\text{-}PCP}$", "lits", "PRs", and "states" represent the clock cycle time of *k-PCP*, the literals of Γ_k, the number of pipelined registers in *k-PCP*, and the number of states in *k-PCP*. All pipelined control paths are synthesized within 2 minutes. The number of states in each *k-PCP* without condition branches, which is the same with its serial control path, is not listed in Table 3. The curves of the delay of Γ_k versus pipelined stage number for example 5_EWF and Cond2 are graphed in Fig. 9. The results show that each ϕ_{Γ_k} for different k is approximate to the delay of combinational logic in the serial control path. Thus, the larger the stage number k is, the shorter its clock cycle time is. The $\phi_{k\text{-}PCP}$ versus pipelined stage number for 5_EWF and Cond2 examples are graphed in Fig. 10. The shortest clock cycle time of control path (i.e. one gate delay) always can be achieved when the stage number is large enough. Since the literals of Γ_k is not necessary increasing, the main penalty paid for reducing the clock cycle time is the increased number of pipelined registers. Moreover, for the examples with conditional branches, the number of states in the control path increases when their stage number increases. Thus the number of state registers may increase when the stage number increases.

VI. Conclusions

We have proposed an efficient approach to synthesizing the pipelined control path from high level to logic level for performance optimization. We presented a technique to generate a k-well BSTG from a k-ill BSTG by inserting the minimal NOOPs. Then, the combinational logic synthesized from the k-well BSTG was pipelined into k-stage pipelined circuit in polynomial time. In the future, we would like to extend the approach to pipelining control paths for reducing power consumption.

References

[1] J. J. Kim, F. J. Kurdahi, and N. Park, "Automatic Synthesis of Time-Stationary Controllers for Pipelined Data Paths," *in Proc. of the International Conference on Computer-Aided Design*, pp.30-pp.33, 1991.

[2] J. P Weng and A. C. Parker, "CGS: Control Path Synthesis in the ADAM System," *6th International Workshop of High Level Synthesis* , pp.52-64, 1992.

[3] Usha Prabhu and Barry M. Pangrle, "Superpipelined Control and Data Path Synthesis," in *Proc. of the 29th Design Automation Conference*, pp.638-643, 1992.

[4] C. Bron and J. Kerbosch, "Finding all cliques of an undirected graph," Communications of the ACM, Vol. 16, No. 9, pp.575-577, 1973.

[5] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, Vol. 49, No. 2, pp.291-308, 1970.

[6] J. M. Jou and S. R. Kuang, "Library-Adaptively Integrated Data Path Synthesis for DSP Systems," in *Proc. of the International Conference on Computer Design*, pp.379-382, 1993.

[7] R. Lisanke. Logic Synthesis benchmark circuits for the International Workshop on Logic Synthesis, May, 1989.

[8] E.M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design using Synthesis and Optimization," in *Proc. of the International Conference on Computer Design*, pp.328-333, October 1992.

[9] C. Leiserson and J. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, 6, pp.5-35, 1991.

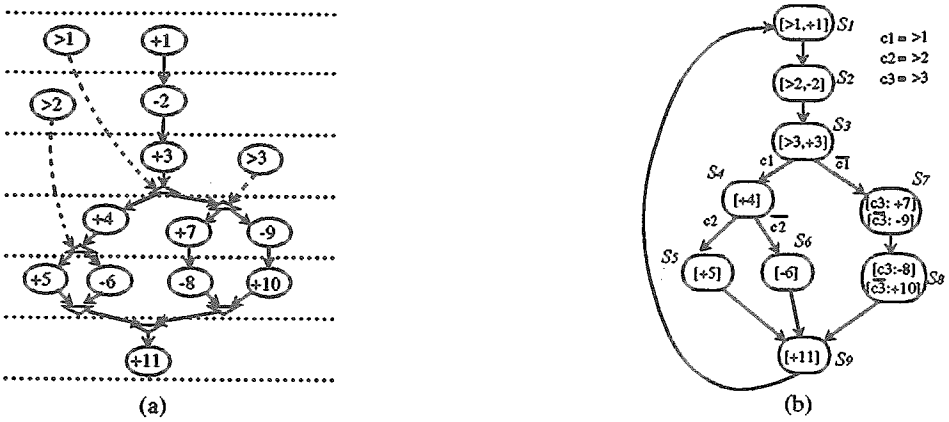

Fig.1. A simple example. (a) the SCDFG, (b) the corresponding BSTG of Fig.1(a).

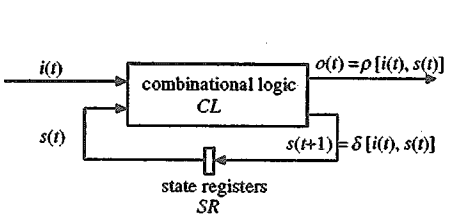

Fig.2. Structure of a serial control path.

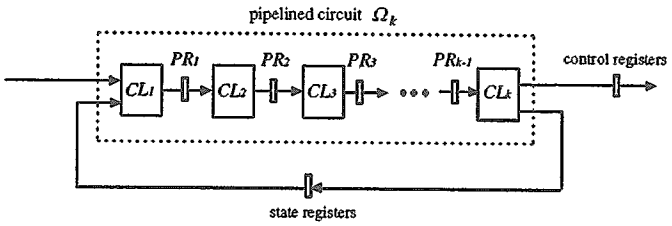

Fig.4. Stucture of a k-stage pipelined control path.

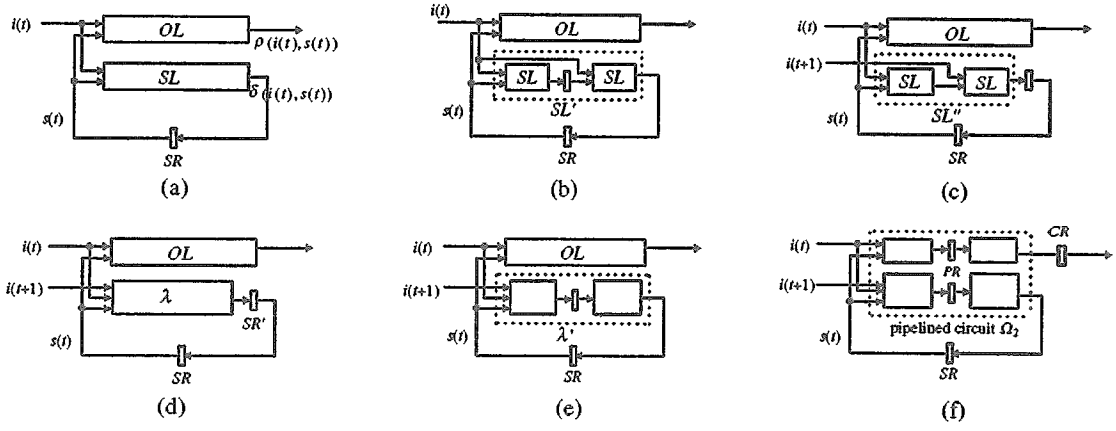

Fig. 3. The process of pipelining a serial control path into a two-stage pipelined control path.

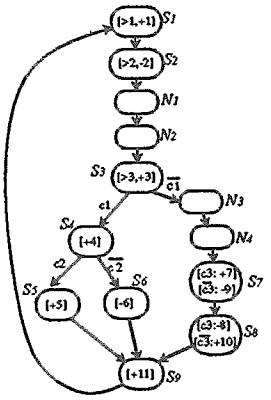

Fig.5. The 2-well BSTG of Fig.1(b).

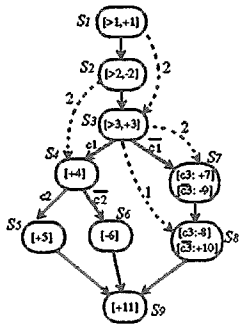

Fig.7. The BSTG with dotted constraint edges.

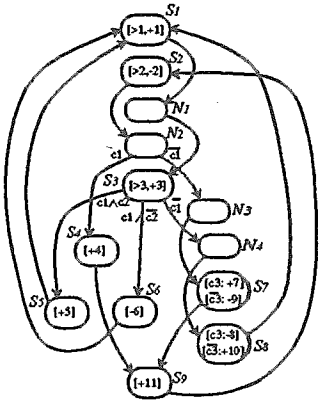

Fig.8. The 2-factor BSTG of Fig.5.

Table 1. The module library used.

function unit	notation	execution time (ns)
adder	ADD	20
subtractor	SUB	20
multiplier	MUL	50
comparator	COM	10

Table 2. The results of data path synthesis.

example	OPs	FUs	conditions	states
6_IIR	*: 12, +: 12	MUL: 2, ADD: 1	-	36
16_FIR	*: 8 +: 15	MUL: 1, ADD: 1	-	44
5_EWF	*: 8, +: 26	MUL: 1, ADD: 2	-	50
Diff	*: 6, +: 1 <: 1, -: 2	MUL: 2, ADD: 1 COM: 1, SUB: 1	1	17
Cond1	+: 8, -: 8	ADD: 1, SUB: 1	5	8
Cond2	+: 8, -: 7	ADD: 1, SUB: 1	5	6

Table 3. The results of stage-fixed pipelining.

example		pipelined stage number k											
		0	1	2	3	4	5	6	7	8	9	10	11
6_IIR	ϕ_{Γ_k}	9	9	9	10	11	10	10	11	10	10	10	.....
	$\phi_{k\text{-}PCP}$	9	9	5	4	3	2	2	2	2	2	1	.....
	lits	269	269	397	341	311	308	422	345	303	370	335	.....
	PRs	-	-	44	64	93	124	180	174	205	254	271	
16_FIR	ϕ_{Γ_k}	9	9	14	11	12	11	11	13	10	10	11	11
	$\phi_{k\text{-}PCP}$	9	9	7	4	3	3	2	2	2	2	2	1
	lits	290	290	375	389	347	379	413	402	351	360	362	416
	PRs	-	-	34	67	93	112	153	176	171	183	208	323
5_EWF	ϕ_{Γ_k}	12	12	12	14	11	11	11	11	11	11	11	11
	$\phi_{k\text{-}PCP}$	12	12	6	5	3	3	2	2	2	2	2	1
	lits	507	507	472	547	508	525	479	557	561	508	492	534
	PRs	-	-	58	99	150	197	223	294	320	351	375	473
Diff	ϕ_{Γ_k}	8	8	8	9	8	8	8	10	8	.....	.....	.....
	$\phi_{k\text{-}PCP}$	8	8	4	3	2	2	2	2	1	.....	.....	.....
	lits	169	136	150	120	109	146	191	140	147	.....	.....	.....
	PRs	-	-	23	32	45	67	81	83	124	.....	.....	.....
	states	17	18	19	20	21	22	23	24	25	.....	.....	.....
Cond1	ϕ_{Γ_k}	6	9	8	8	8	8	9	10	11	11	9	.....
	$\phi_{k\text{-}PCP}$	6	9	4	3	2	2	2	2	2	2	1	.....
	lits	154	147	198	173	282	212	279	254	338	267	278	.....
	PRs	-	-	24	42	95	97	126	140	177	206	240	.....
	states	8	10	13	16	19	22	25	28	31	34	37	
Cond2	ϕ_{Γ_k}	8	8	8	9	9	10	9	11	9	8	.....	.....
	$\phi_{k\text{-}PCP}$	8	8	4	3	3	2	2	2	2	1	.....	.....
	lits	178	226	216	174	280	208	310	300	262	260	.....	.....
	PRs	-	-	29	49	85	101	133	147	182	218	.....	.....
	states	6	8	10	12	14	16	18	20	22	24		

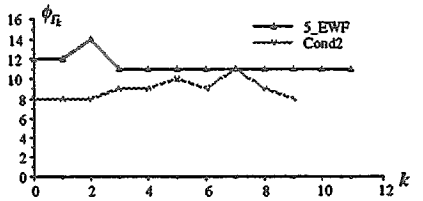

Fig.9. The curves of the delay of Γ_k, ϕ_{Γ_k}, versus its pipelined statge number k.

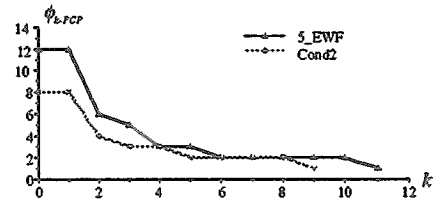

Fig.10. The curves of the cycle time of k-*PCP* versus its pipelined statge number k.