韓國産業應用學會論文誌 제3권 제4호 299-306, 2000.11
J. of Korea Society of Industrial Application, Vol.3, No.4, 299-306, November. 2000

상위수준 파이프라인 합성시스템에 관한 연구 : 데이터 경로 및 콘트롤 합성

A Study on High-Level Pipeline Synthesis System : Data Path Synthesis and Control Synthesis[1)]

김종태*
Jong-Tae Kim

<요 약>

이 논문은 파이프라인 합성을 위한 상위수준 데이터 경로 합성과 콘트롤 합성의 통합에 관한 연구이다. 현재 대부분의 상위수준 합성 방법은 콘트롤 영역의 영향을 무시하는데 보다 나은 설계를 위하여 데이터 경로 디자인 영역과 콘트롤 디자인 영역을 통합하여 탐색하는 파이프라인 상위수준합성 도구를 구현했다. 이 도구는 비용 제한 하에서 최고 성능의 파이프라인을 합성하는 비용제한합성과 성능 제한 하에서 최저 비용의 파이프라인을 합성하는 성능제한합성의 두 가지 방식을 제공한다.

Key Words : High-level Synthesis, Pipeline Synthesis

1. Introduction

Pipelining is a widely used approach for designing high performance digital circuits. As design size increases, pipelined architectures become quite complex and thus automated design tools in high level for pipeline ASICs are necessary to cope with such complexity and explore the design space efficiently. Task of high-level synthesis of pipelines may be divided into data path synthesis and control synthesis. The main tasks of data path synthesis include scheduling and resource allocation[1]. Scheduling is the process of partitioning the input specification so that each partition corresponds to a single time step, or a control step in execution and assigns operations in the input graph to time steps while observing the data precedence and satisfying constraints. At this point, high-level control specifications can be found. Currently most high-level synthesis systems

* 정회원, 성균관대학교 전기전자컴퓨터공학부, 부교수, 工博
경기도 수원시 장안구 천천동 300
email : jtkim@yurim.skku.ac.kr
Associate Professor, School of Electrical and Computer Engineering
(This study is supported by the academic research fund of Ministry of Education, Republic of Korea.)

ignore the effects of the control space and control synthesis usually follows data path synthesis, but optimal design can be guaranteed only if both processes occur simultaneously. Therefore it is necessary to combine data path and control synthesis. The main objective for this work is to provide the designer with a more precise and efficient tool for design space exploration.

Early works[2] in pipeline design focused on either scheduling and controlling existing pipelines or physically construction of stages with fixed functions. Pipeline stages are physically separated (structural pipelining). This may force non-optimal use or sharing of resources. The data path synthesis program we developed, called Sehwa[1], presents some theoretical foundations of pipelined synthesis. In Sehwa, the logical-stage concept (functional pipelining) was used. A logical stage corresponds to the set of operators, registers, and multiplexors which are activated during the same clock cycle. Two logical stages may share the same resource, which makes more complex and efficient sharing of resources possible. Other works on the scheduling of pipeline data paths can be found in [3][4].

Most of the previous work done in the control synthesis at the register transfer level was for non-pipeline systems. The CONSPEC[5] dealt with the automatic production of control specifications from high-level behavioral descriptions in control and timing graph form and is designed for interface processors. Bridge[6] is a high level synthesis sytem developed at AT&T Bell Laboratory and performs data path and control path allocation for non-pipeline systems. We developed a method to automatically synthesize time-stationary controller for pipeline data paths.[7]

To the best of knowledge, no comprehensive work was been published on the high-level pipeline synthesis which explores both data path and control design space.

2. Overview of Pipeline Synthesis System

The design tasks of high-level synthesis of pipelines include scheduling, resource allocation, and generation of a control specification.

2.1 Data Path Synthesis

The main design tasks of Sehwa are scheduling and resource allocation in such a way that the result preserves the behavior of the original data flow and meets all the design constraints in the following way. First the result of partitioning a data flow into time steps must reserve the original data precedence between operations in the input data flow graph to preserve the correctness of the design. Second the scheduling must be performed in such a way that each subtask can be completed within a given clock cycle time including the stage latch delays. And third, the resource requirement of each subtask never goes over the total available resources at any given time step and the sum of the resource requirements of the subtasks must not exceed the total available resources.

Sehwa is comprised of two classes of scheduling algorithms. They are polynomial-time scheduling algorithm and exhaustive algorithm. The polynomial time scheduling algorithm is further decomposed into feasible, maximal, and non-overlap schedulings. Feasible scheduling schedules a data flow graph with a fixed latency, a maximum stage-time limit (or a given clock cycle), and constraints on either the total cost of the pipeline implementation or the minimum required performance. Maximal scheduling schedules a data flow graph as short as possible with only a maximum stage-time limit, assuming there is no cost constraint (in this case, the latency is always 1), and non-overlap scheduling schedules a data flow graph as short as possible with a maximum stage-time limit and constraints on the total number of available modules. No execution overlap is considered.

The outline of the synthesis procedure used by Sehwa is as follows. Maximal/Non-overlap synthesis is done to establish the boundary of the possible design space before cost or performance constraint

synthesis routines are invoked for the actual synthesis of the pipeline. Maximal synthesis produces the fastest design with the latency of 1 and minimal synthesis produces the cheapest and slowest design with the latency equal to the depth of the pipe. Then a range of designs close to a desired goal are produced using fast feasible scheduling procedures which is the main scheduling algorithm and is used to search the design space under cost or performance constraints. The design tasks are divide into two cases depending on the design constraint: design the fastest pipeline within the cost constraint, or design the cheapest pipeline that satisfies the minimum required performance.

An intelligent design analysis procedure is used to guide the design process toward certain optimal direction. The user can perform exhaustive search if none of the earlier designs meet his needs. The current best design found so far provides a tight bound on the design space for the exhaustive search. If none of the pipelined design meets the design constraints, Sehwa will design a non-pipelined data path using the same scheduling and resource allocation algorithms. Figure 1 depicts the design boundary for a pipeline design of a given data flow graph and the design search space is bounded by the cheapest design (point B) and the fastest design (point A). If the total cost is limited, point C is the optimal design. If the minimum required performance is given, point D is the cheapest design. The feasible design space is enclosed in the rectangle of the points C and D. While the scheduling is repeated with different cost and performance constraints, we only need to search the feasible space inside the possible design space.

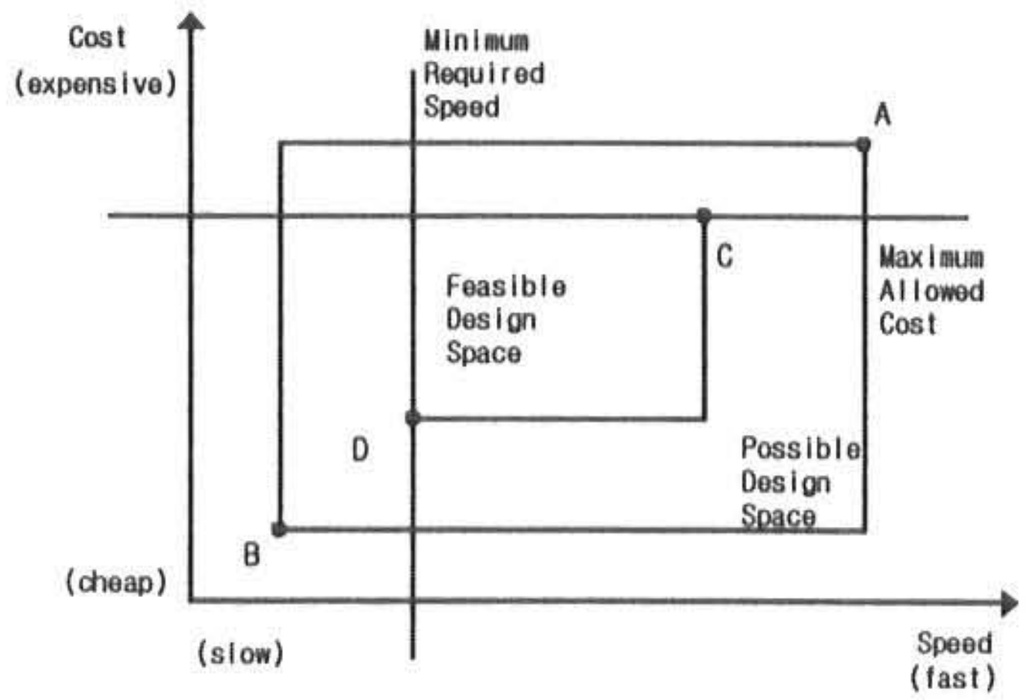

Figure 1. Design space boundary

2.2 Control Synthesis

Abstract control requirements such as the initiation latency and the number of clock cycles per task can be obtained from the results of scheduling and resource allocation. Ideally, we need to design the controller in parallel with the data path in order to perform optimal control-data path design tradeoff. However, this requires a complete understanding of the interaction between the two components and needs to be studied and experimented further.

The controller is modeled as a Moore style Finite State Machine (FSM) in which state memory holds its present state, and the combinational parts decide the next state and the primary output functions. The control specification procedure consists of two major steps: *state decision, and state transition.* The input is a scheduled data flow graph (DFG) which shows operator-to-time step assignments and interconnects between operations. The output is a FSM specification in the form of a state table. In the remainder of this section, we describe these steps and present our approach to solving each one. The detailed description on the control synthesis can be found in [7].

First the *mutual exclusion set* (MES) is identified for given time step i. A set M of nodes is said to be a MES if all the nodes in M are pairwise-mutually exclusive and M is not included in any larger MES M′. MESs are the maximal groups of mutually exclusive operations within a given time step. Let $M_{i,1}$, $M_{i,2},\cdots$, $M_{i,n}$ denote the MES covering time step i, a *Possible Execution Mode or* PEM, P is defined as a set of n operations, one from each MES. $P_i = \{o_1,\cdots,o_n \mid o_h \in M_{i,h}, h=1,\cdots,n\}$. We will denote by $P_{i,1}$, $P_{i,2},\cdots$, the different PEM's in time step i. Next, we find sets of operations $P_{i,j}$, PEM, which can be executed concurrently in each time step by picking one operation from each MES and combining them. Thus, each $P_{i,j}$ represents a

subset of nodes that can be executed in parallel during time step i. Since the schedule is pipelined, time steps i, $i+L$, $i+2L$,$\cdots$, are overlapping and therefore, a *state* can now be defined as follows: Given $1 \le i \le L$, a *state Si* is defined as a set of PEM's corresponding to overlapping time steps i, $i+L$, $i+2L$,$\cdots$. $S_i = \{P_{k,l} \mid \forall P_{k,l}, P_{m,n} \in S_i, k \bmod L = m \bmod L = i\}$, and S_i is not included in any larger state S_i'. We will denote by $S_{i,1}$, $S_{i,2}$,$\cdots$, $S_{i,ni}$ all the states that can be generated by different combinations of PEM's in i and the time steps that overlap with it, and n_i is the number of such different combinations. Since $1 \le i \le L$, we can define groups of states G_1, G_2,$\cdots$, G_i,$\cdots$, G_L such that $G_i = \{S_{i,j}, 1 \le j \le n_i\}$.

After identifying the states, we need to determine the state transitions. Given a CDFG pipeline-scheduled with a latency L, we observe that state transitions occur between adjacent groups of states in the following sequence: $G_1 \rightarrow G_2 \ldots G_i \rightarrow G_{i+1} \ldots G_L \rightarrow G_1$. This is mainly due to the pipelined nature of the scheduling and is shown in Figure 2. This is a key property in our optimization scheme, as will be discussed later. Another important factor affecting the control specifications are the distribution nodes (D). If the present state has m D nodes, there are 2^m possible combinations of input conditions. Given a particular state, the next state is the one which has all the compatible (i.e. not mutually exclusive) nodes of the present state. We find compatible

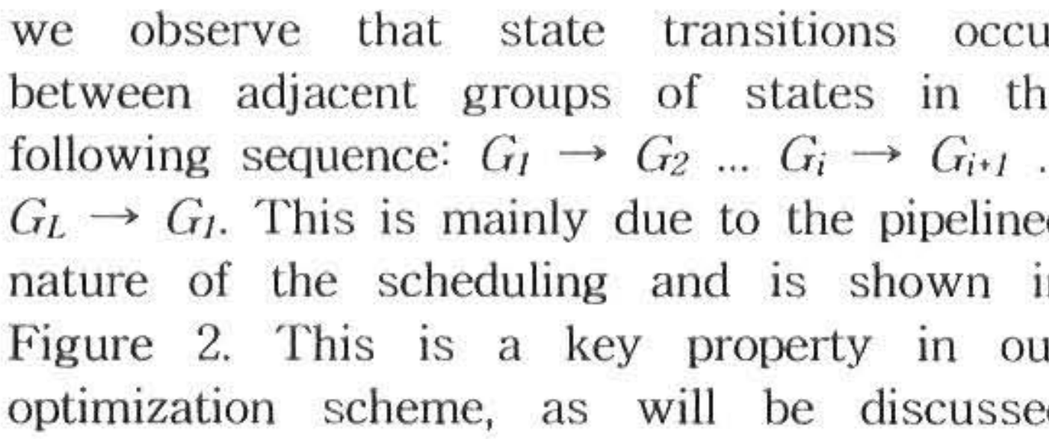

Figure 2. Overall timing and state transitions in a pipelined system

nodes by searching only the PEMs corresponding to the next time step within states in the next group.

3. Unification of Data Path and Control Synthesis

The main objective for this work is to provide the designer with a more precise and efficient tool for design space exploration. The use of automatic synthesis permits the designer to examine a large number of non-inferior designs under various design constraints in a relatively short period of time. From the set of possible solutions, the design that best optimizes the designer's objective function can be selected. Once a pipeline schedule and a operation allocation are obtained, the RT implementation must be completed in order to compute the total cost. The total cost includes operators, multiplexors, latches, controller, and wiring space. It is necessary to consider the total cost in order for the pipeline synthesis system to do the following: compare the total cost and speed of candidate solutions and choose the optimal solution, and determine valid design search space. Using fewer operators for a cheap data path will increase the number of latches and multiplexors. This is because sharing operators requires more input/output multiplexors (or buses) and more latches to store values which cannot be consumed at the next clock cycle after they are produced. The area and delay of controllers mainly depend on the following factors. First, the number of time steps in the pipeline schedule (pipe size in our definition), longer the number of time steps, larger the number of time steps in the overlapped time steps. Second, the number of conditional branches existing in the input behavior and where those conditional constructs are scheduled. If there are many conditional branches scheduled in the overlapped time steps, then the number of states is increased significantly and therefore, the controller is very expensive.

3.1 Cost constrained Synthesis

The objective of this cost constrained synthesis is to find the design which is the

fastest pipeline within the given cost constraint. Thus, the cost constrained synthesis routine starts with the minimum possible latency which is possible within the given data path cost constraint, since the smaller latency always results in faster design, as compared to designs with larger latency. After determining all the possible stage times (and make sorted list from smallest to largest), scheduling begins with the smallest stage time. Both data path and controller are synthesized to produce RT-level design according to the resulting schedule. The cost and delay of data path and controller are calculated and the cost of the current solution is compared with the given cost constraint. If the solution is feasible, (i.e., within cost constraint), then two actions are taken: put the solution in the buffer to be evaluated later and, compare the delay of the current design with the next possible stage time. If it is bigger, then do synthesis with next small stage time (reiterate the loop). Otherwise the following steps will carry out (break out of the loop). This second action is taken to reduce the design space search in minimal because if the delay of the current solution is less than the next possible stage time the new solutions found from the next iteration are always inferior (i.e., larger delay) to the previous solution of smaller stage time. Next, choose the fastest design among the possible solutions in the buffer as a solution and analyze it and determine whether there will be any improvement by providing more modules. In cost constrained synthesis, there are module counters associated with each module type that counts the operations that has been delayed to some later time steps only due to the resource limit. If any of these counter values is a positive number, the total number of modules allocated to that type is increased by the counter value as long as addition of modules does not violated the data path cost constraint. This might result in finding shorter schedule. However, if no solution is found, the latency is incremented or readjust data path cost constraint within the total cost constraint, and the possible stage times are again considered with the new latency. Increasing the latency enables more sharing of resources, thereby, reducing the total operator cost, but increasing the number of muxes used. These processes are repeated until a suitable solution is found. The outline of the new cost constrained synthesis procedure is shown in Figure 3. The control synthesis is incorporated into the cost constrained synthesis in Sehwa.

3.2 Performance Constrained Synthesis

The aim of performance constrained synthesis is to find the cheapest design with the minimum required performance as design constraint. Thus, scheduling starts with the maximum possible latency. All the possible stage times are determined and scheduling starts with the smallest stage time. Data path and controller are synthesized to produce RT-level design according to the resulting schedule. The cost and delay of the design are computed and the delay is compared with the given total performance constraint. If the delay of the current design is within the total performance constraint, put the solution in the solution buffer and

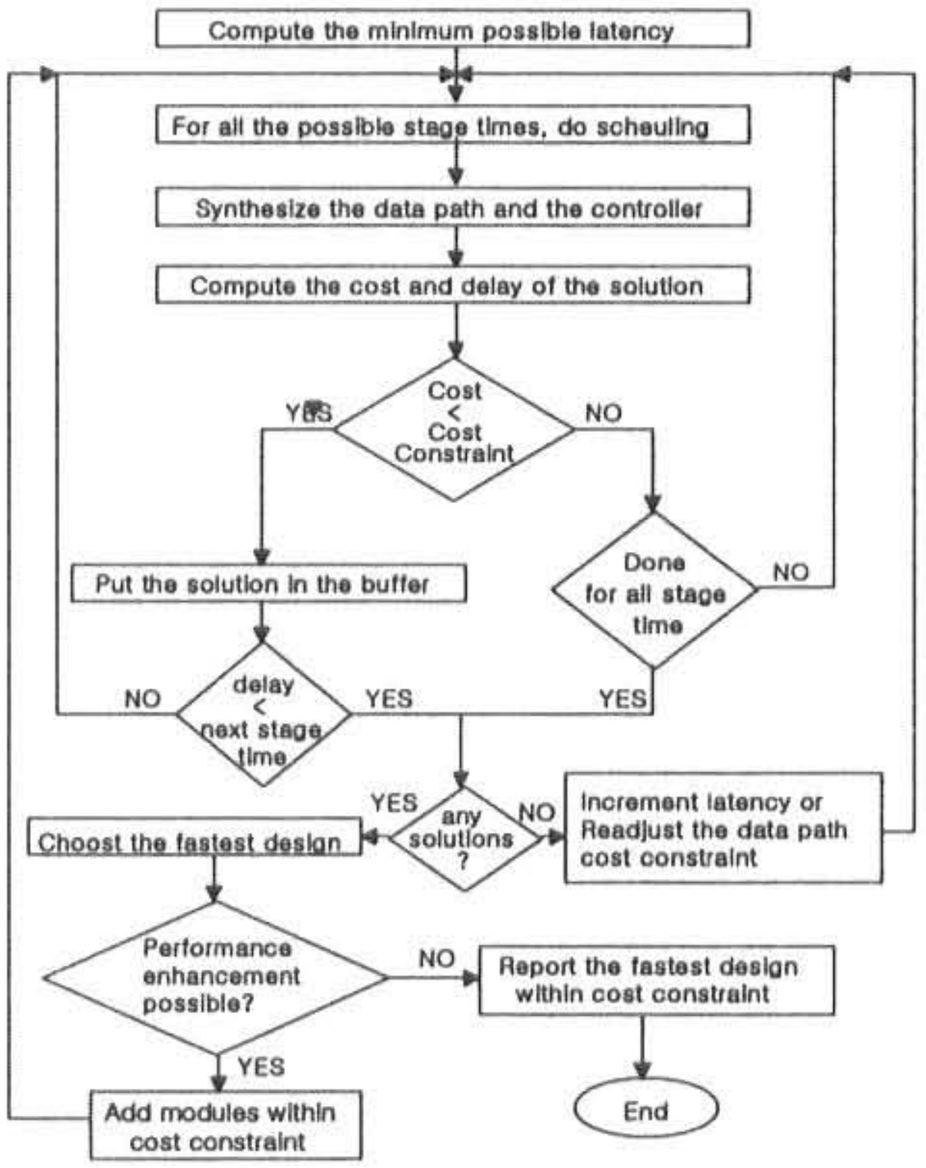

Figure 3. Cost constrained synthesis

increment the latency to search for a cheaper solution since the feasible solution is already found. If the condition is not met, resynthesis begins with next small stage time within the given performance constraint. If no feasible solution is found, the latency is decremented (i.e., a faster design is sought). Using smaller latency will require more modules to be allocated for scheduling, thus increasing the performance, but it is more expensive. This is repeated until possible solutions are found. If solutions are found, choose the cheapest design satisfying the overall performance constraint as the constrained synthesis in Sehwa.

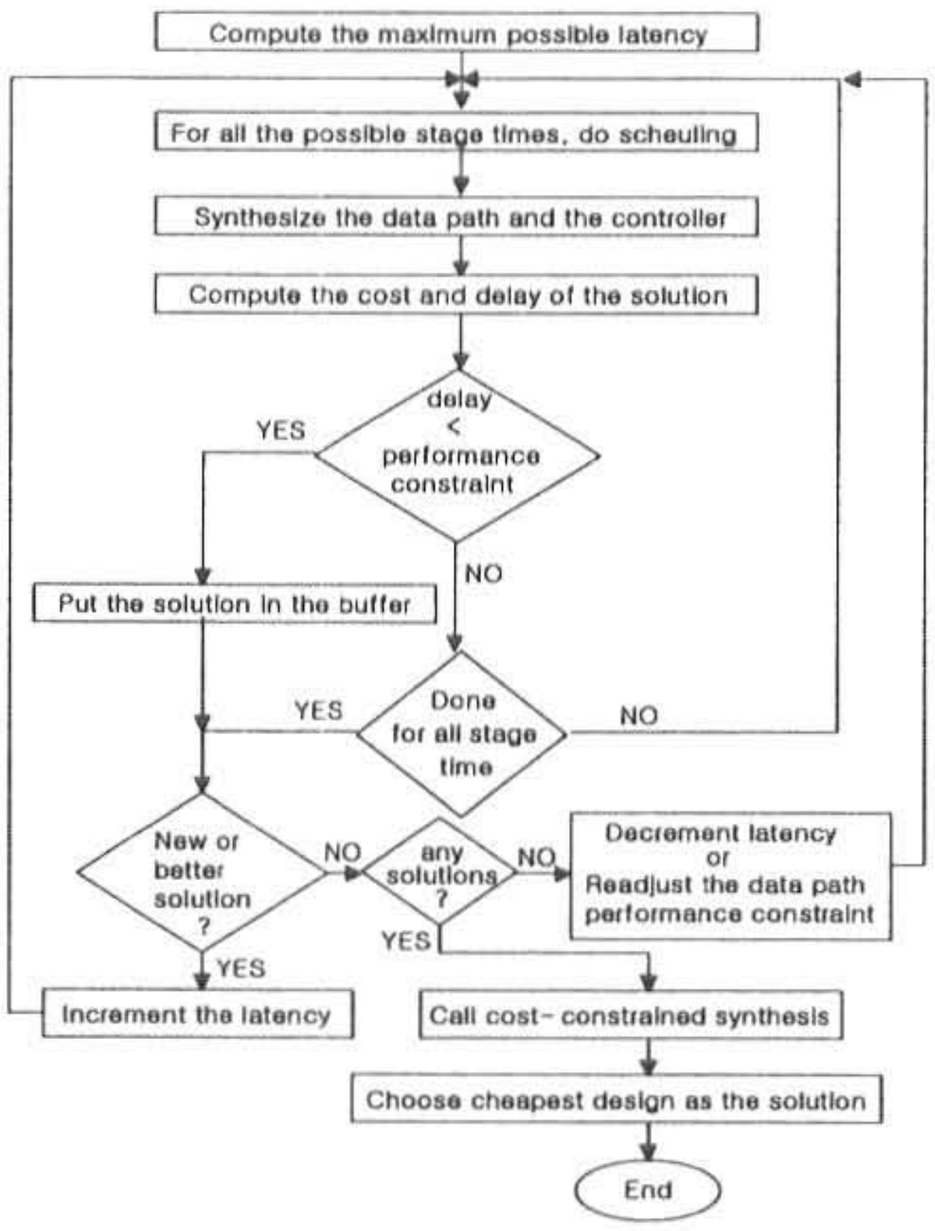

Figure 4. Performance-constrained synthesis

4. Experiments

In this section, we present some experimental results which were obtained by applying our approach to two design examples. The first example is from [1], and the second is a quadratic equation.

The data flow graph of the first example is shown in Figure5. We performed high-level pipeline synthesis with speed constraint. Figure 6 shows the area vs. the delay of the design points generated by synthesis routines. In the second example we assume that a, b,

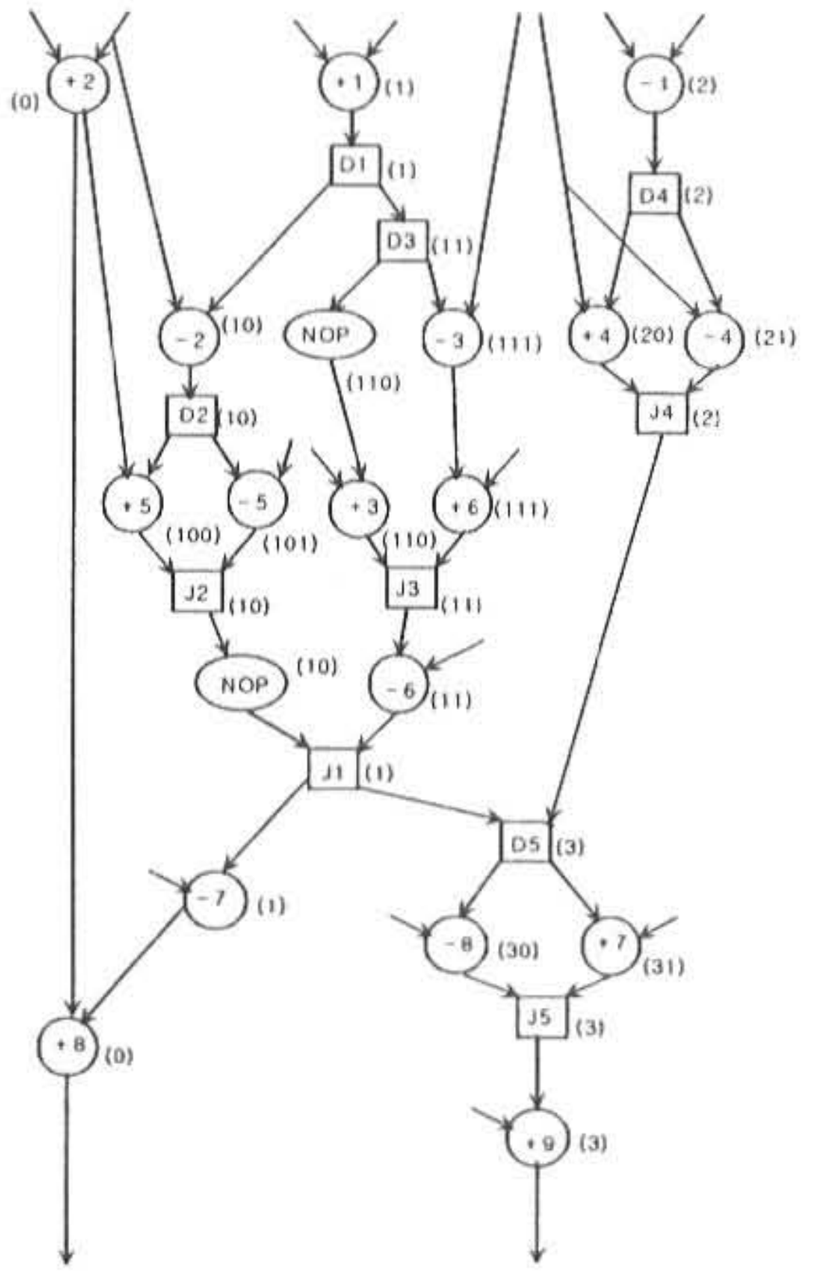

Figure 5. Data flow graph for example 1

candidate for the solution. Cost constrained synthesis routine is invoked once to look for a cheaper design with the cost for the solution candidate as its cost constraint and it is appended to the solution set. Finally, the cheapest design within the performance constraint is selected as a solution. Figure 4 shows an outline of the performance constrained synthesis. The control synthesis is incorporated into the performance

and c in the equation $ax^2+bx+c=0$ are 8-bit integers so that the number of iterations to compute the square root is 7. We unrolled the loop completely and ran synthesis routines with both cost and speed constraints. Figure 7 shows the possible design points and the example of speed constrained synthesis with the initiation interval of 100 and the design point D is

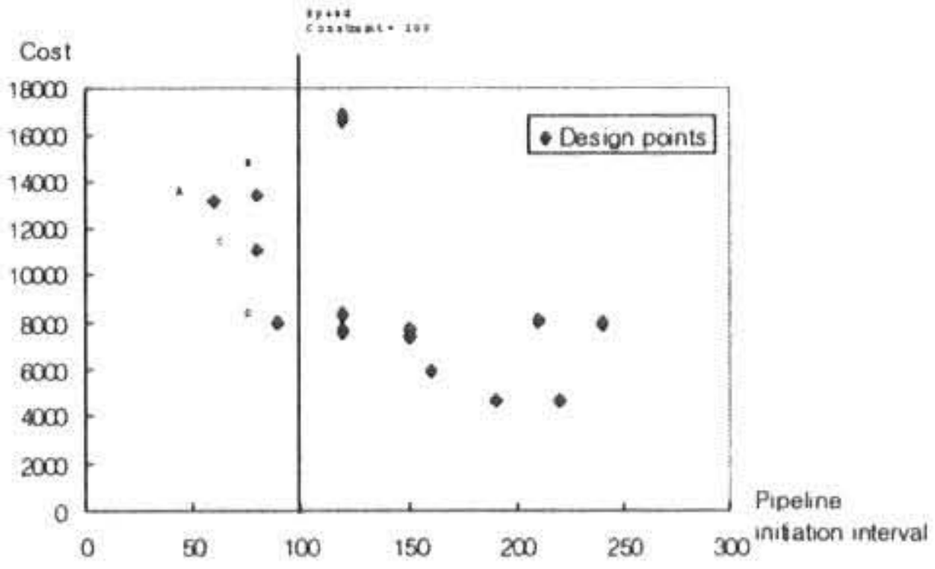

Figure 7. Speed constrained synthesis result of quadratic equation example

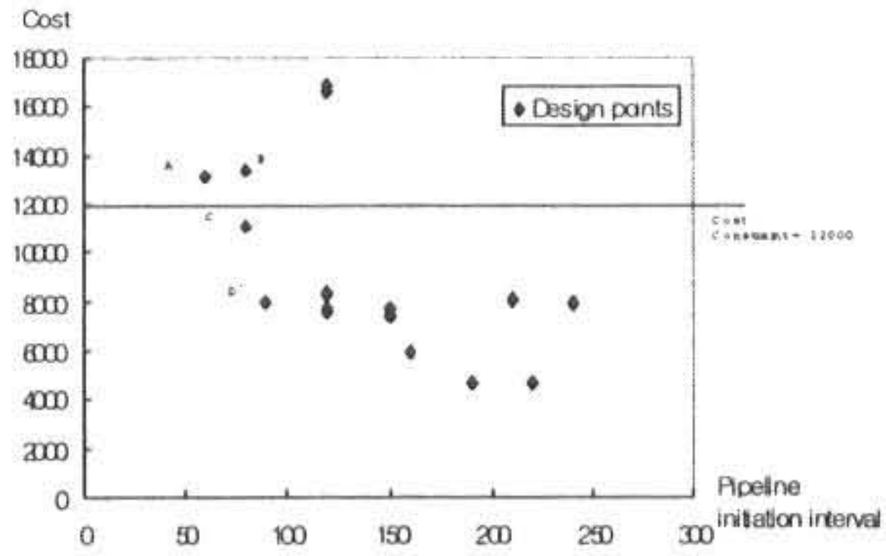

Figure 8. Cost constrained synthesis result of quadratic equation example

selected as a solution in the design space which is the cheapest design among the possible solutions. We also performed cost constrained synthesis with the cost of 12,000 and the result is shown in Figure 8. The solution is found in the design point C′ which is the fastest among the possible solutions.

With two design examples, we cannot claim that our synthesis system can guarantee to find the optimal design every time. We rather say that integration of data path and control syntheses can search the design space more efficiently to find the better design. Clearly more experiments must be done in order to better understand the interaction between data path and control, and access the effect of the latter on the overall design process.

5. Conclusions

Nowadays, there is a high demand for high-speed computing with limited resources and for fast design of systems while guaranteeing their correctness. Pipelining has been a good methodology for designing fast digital circuits, but pipeline design is usually far more complex than nonpipeline design due to the execution overlap. Automated design tools for pipeline designs are necessary to cope with such complexity and explore the design space efficiently. In this paper we have addressed the problem of high-level pipeline synthesis and investigated of both data path and control design space. The main objective of this research is to provide designer with a more precise and efficient tool for design space exploration. We modified Sehwa's synthesis routines to be incorporated with control synthesis engine so that the both data path and control design spaces are explored. This effort leads one step forward to developing a unified methodology for combined data path and control synthesis of pipelined system.

Reference

1) N. Park and A. Parker, "Sehwa: a software package for synthesis of pipelines from behavioral specifications," IEEE Trans. on CAD, 7(3): 356-370, March 1988.
2) E. Davidson, "The design and control of pipelined function generators," in Proc. of 1971 International IEEE Conf. on Systems, Networks, and Computers, pp. 19-21, 1971.
3) E. Girczyc, "Loop winding - a data flow approach to functional pipelining," in Proc. ISCAS, IEEE, May 1987.

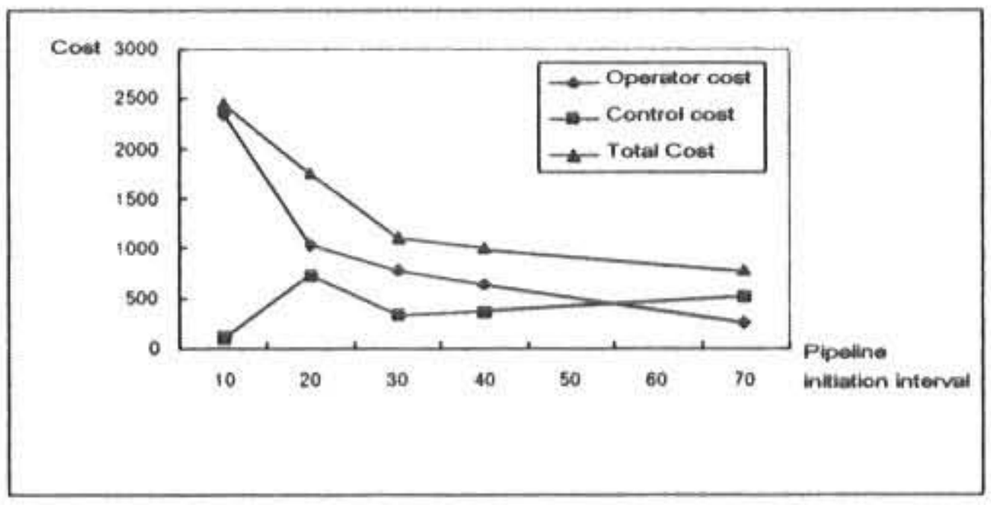

Figure 6. Pipeline synthesis results for example 1.

4) H. Lee, S. Hwang, "Design of a high-level synthesis system for automatic generation of pipeline data path", J. of KITE, 31(3), pp53-67, March 1994.
5) S. Hayati and A. Parker, "Automatic Production of Controller Specifications from Control and Timing Behavioral Descriptions", in Proc. of 26th Design Automation Conference, pp. 75-80, June 1989.
6) C. Tseng et al., "Bridge: a Versatile Behavioral Synthesis System," in Proc. of 25th Design Automation Conference, pp.415-420,June 1988.
7) J.T. Kim, F.Kurdahi, N. Park, "System-level time-stationary control synthesis for pipelined data paths", VLSI Design, pp. 159-180, 9(2), April 1999.

(2000년 6월8일 접수, 2000년 11월22일 채택)