Reducing Register and Phase Requirements for Synchronous Circuits Derived Using Software Pipelining Techniques

Noureddine Chabini[1], El Mostapha Aboulhamid[1], Yvon Savaria[2]
1: *LASSO, DIRO, Université de Montréal C.P.6128, Suc. Centre ville, Montréal, Qc, Canada, H3C 3J7. Email:{chabinin, aboulham}@iro.umontreal.ca*
2: *GRM, DGEGI, École Polytechnique de Montréal, C.P. 6079, Suc. Centre-ville, Montréal, Qc, Canada, H3C 3A7. Email: savaria@vlsi.polymtl.ca*

Abstract

A method based on a modulo scheduling algorithm for software pipelining has been recently proposed to optimize clocked circuits. The resulting circuits are multi-phase clocked circuits, where all clocks have the same period. To preserve the functionality of the original circuit, registers must be placed after minimizing the clock period. The placement of these registers is derived from an arbitrary schedule determined during a clock period minimization step. A good schedule may allow to decrease the number of registers and the number of phases needed in the final circuit. Decreasing the number of registers contributes to minimizing the area occupied by the circuit and reduces its power consumption; while decreasing the number of phases reduces the complexity of the clock generation and distribution tasks. In this paper, we propose polynomial-time-solvable methods to choose a good schedule once the clock period is minimized. The methods have been tested on a subset of the ISCAS89 benchmarks. Experimental results show that the number of registers which must be inserted in the final circuit, and the number of phases, have been significantly decreased compared to the case where an arbitrary schedule is chosen.

1. Introduction

The performance of clocked digital circuits can be improved by minimizing their clock period. Retiming [7] is a technique often used for that purpose. This technique changes register placement in the circuit to minimize the maximum combinational delay between any two neighboring registers. Basic retiming supposes that the circuit is clocked by a single-phase clock, which limits the search space for effective timing solutions. Various extensions to retiming have been proposed [6, 8]. In [8], retiming with multi-phase clocks was proposed. With this method, the phases are fixed before retiming, which can give an optimal clock period if good phases are chosen.

Software pipelining has been proved to be a powerful technique for increasing the instruction-level parallelism for parallel processors. It has recently been used for optimizing clocked circuits [1]. The resulting circuit is a multi-phase clocked circuit, where all clocks have the same period. That method may be described as follows. First, the optimal clock period is determined, and a schedule of all the functional elements of the circuit is computed. Second, in order to preserve the behavior of the original circuit, registers are placed independently of their initial placement. The placement of registers is done using the fixed schedule. Finally, once the registers are placed, the phases are determined.

In the previous method, As Soon As Possible (ASAP) or As Late As Possible (ALAP) schedule is used. Choosing a good schedule may allow to decrease the number of registers that must be placed in the final circuit, and its number of phases. Decreasing the number of registers contributes to minimizing the area occupied by the circuit and reduces its power consumption; while decreasing the number of phases reduces the complexity of the clock generation task. In this paper, we propose polynomial-time-solvable methods to determine schedules for reducing the number of registers and the number of phases. The methods have been tested on various benchmarks selected from the ISCAS89 set. Experimental results show that the number of registers that must be inserted in the final circuit, and the required number of phases, can be substantially reduced compared to the case where an ASAP or an ALAP schedule is chosen.

The remainder of this paper is organized as follows. In Section 2, we introduce the notations and definitions used in this work. Section 3 briefly reviews the registers placement step in the method based on software pipelining, which was outlined above. Our methods to determine schedules for reducing the number of registers and the number of phases are presented in Sections 4 and 5. Section 6 presents experimental results. Conclusions are provided in Section 7.

2. Preliminaries

2.1. The cyclic graph model

In order to minimize the clock period of a synchronous sequential circuit, it is modeled (as in [1, 7]) as a directed cyclic graph $G = (V, E, d, w)$, where V is the set of functional elements in the circuit, and E is the set of edges which represent interconnections between vertices. Each vertex v in V has a non-negative integer propagation delay $d(v) \in N$. Each edge $e_{u,v}$, from u to v, in E is weighted with a register count $w(e_{u,v}) \in N$, representing the number of registers on the wire between u and v.

Figure 1 presents an example of a circuit and its directed cyclic graph model. In this figure, large rectangles represent functional elements, and small rectangles represent registers. Wires are oriented to show the propagation direction of the signals. The propagation delay of each functional element of this circuit is specified as a label on the left of each large rectangle. This example will be used through this paper, and will serve as an example of initial specification for the problem to optimize. The initial specification is in general a synchronous circuit with a single-phase clock period. The clock period of the circuit in Figure 1 is 6, which is equal to $d(v_4) + d(v_1)$.

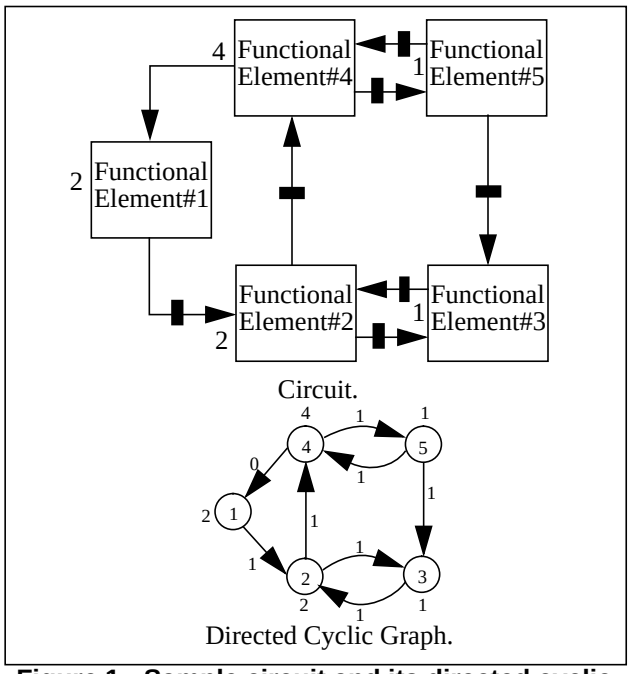

Figure 1 : Sample circuit and its directed cyclic graph model.

2.2. Periodic and *k*-periodic schedules

We define a schedule s [1, 5] as a function $s : N \times V \to Q$, where $s_n(v) \equiv s(n, v)$ denotes the schedule time of the n^{th} iteration of operation v. In multi-phase flip-flop based circuits, the schedule time is the start time of the operation. A schedule s is called periodic with period P, if:

$$\forall n \in N,\ \forall v \in V:\ s_{n+1}(v) = s_n(v) + P\,. \qquad (1)$$

A schedule s is called k-periodic if there exist integers n_0 and k, and a positive rational number P such that:

$$\forall n \geq n_0,\ \forall v \in V:\ s_{n+k}(v) = s_n(v) + k \cdot P\,. \qquad (2)$$

Both periodic and k-periodic schedules have the same throughput $T = 1/P$. The k-periodic schedules have a period of $(k \cdot P)$. When there is no resource constraint, a schedule s is said to be valid if and only if the operations terminate before their results are needed. In this case, we say that data dependencies are satisfied, which is equivalent to the following mathematical inequality:

$$\forall n \in N,\ \forall e_{u,v} \in E:\ s_{n+w(e_{u,v})}(v) \geq s_n(u) + d(u)\,. \qquad (3)$$

2.3. Maximum throughput of synchronous sequential circuit

The throughput, T, of synchronous sequential circuit is bounded by the inverse of the length, P, of the critical paths in the circuit. Based on data dependencies constraints only, the maximum throughput is [5]:

$$T = Min_{c \in C}\left(\left(\sum_{e_{u,v} \in c} w(e_{u,v})\right) \Big/ \left(\sum_{\forall v \in V \text{ and } e_{u,v} \in c} d(u)\right)\right), \qquad (4)$$

where C is the set of directed cycles in the directed cyclic graph modeling the circuit. Determining the maximum throughput is a Minimal Cost-to-Time Ratio Cycle Problem [3, 4], which can be solved in the general case in $O(|V| \cdot |E| \cdot \log(|V| \cdot d_{max}))$ [4], where $d_{max} = Max_{v \in V}(d(v))$. A possible method to solve this problem is to iteratively apply Bellman-Ford's algorithm for longest paths on the graph $G_P = (V, E, d, w_P)$ derived from G by letting:

$$w_P(e_{u,v}) = d(u) - P \cdot w(e_{u,v})\,, \qquad (5)$$

where $e_{u,v} \in E$ and $P = 1/T$. A binary search may be used to find the minimal value of P for which there is no positive cycle in G_P [5].

For the example in Figure 1, we have that $P = 4$. This value corresponds to the cycle defined by vertices v_1, v_2, and v_4.

2.4. Schedule for a given throughput

From equation (1) and inequality (3), we have that:

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v})\,. \qquad (6)$$

In the case of periodic schedules, determining a valid schedule of all the instances of each vertex v in V is equivalent to determining $s_0(v)$ for each v in V, which is also equivalent to determining solutions to the system of inequalities described by (6). To resolve this system, the graph G_P, previously described, may be used. To find an ASAP schedule, Bellman-Ford's algorithm for longest

paths, from a chosen vertex v_x to the others, may be applied on the graph G_P. Finding an ALAP schedule may be done as follows. Step 1, a graph G' has to be derived from G_P by inverting the direction of each edge in G_P. Step 2, Bellman-Ford's algorithm for longest paths, from the vertex v_x to the others, has to be applied on the graph G', where the weights of its edges are defined by equation (5). Finally, step 3, the ALAP schedule is obtained by multiplying each result in step 2 by -1. Table 1 gives the ASAP and ALAP schedules, relatively to $v_x = v_1$, of vertices of the circuit in Figure 1.

Table 1 : ASAP and ALAP schedules.

	Vertices				
	v_1	v_2	v_3	v_4	v_5
ASAP	0	-2	-4	-4	-4
ALAP	0	-2	1	-4	-1

2.5. Schedule graph

As in [1], a periodic schedule, with period P, of vertices of directed cyclic graph modeling a circuit is presented by a schedule graph $G_s = (V, E, d, w_s, P)$. Here V, E and d have the same definition given for the case of the graph G previously defined, and $w_s : E \to Q$ is a weight function, which associates to each edge $e_{u,v}$ in E the time distance between the schedule times of u and v. Mathematically, $w_s(e_{u,v})$ is defined as follows:

$$\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_{w(e_{u,v})}(v) - s_0(u)\,. \qquad (7)$$

Because s is periodic with period P, equation (7) may be written as follows:

$$\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_0(v) - s_0(u) + P \cdot w(e_{u,v})\,. \qquad (8)$$

The graph G_S is consistent if and only if for each edge $e_{u,v}$ in E, $w_s(e_{u,v}) \geq d(u)$. This is derived from equation (3). Figure 2 shows a consistent schedule graph, where edges are labeled with w_s values, for the circuit in Figure 1 using the ALAP schedule from Table 1.

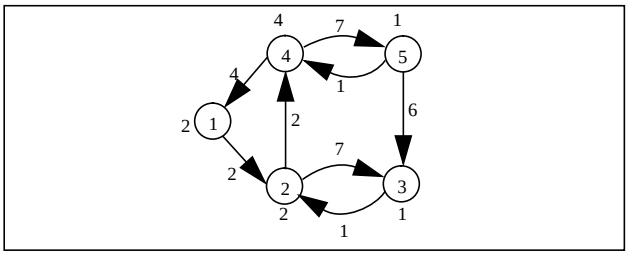

Figure 2 : Schedule graph.

2.6. Incidence, unimodular and eulerian matrices

In this paper, the incidence matrix, M, of a directed graph, G, is a matrix whose lines are indexed by the edges of G while the columns are indexed by its vertices. The entries of M are defined as follows: $M_{e,i}$ is equal to 1 if i is the tail of the edge e, to -1 if i is the head of e, and to 0 otherwise. For instance, the incidence matrix of the graph in Figure 1 is:

$$M = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}.$$

A matrix M is *totally unimodular* if every one of its square submatrices has a determinant equal to -1, 0 or 1 [9]. A submatrix of a matrix is said to be *Eulerian* [9, 10] if the sum of the entries of its lines and the sum of the entries of its columns are both even. This will be used to prove Theorem 1 in Section 4.2.

3. Register placement

In the method proposed in [1], which was outlined in Section 1, a register placement step is needed in order to preserve the behavior of the original circuit. The placement of registers is derived from a schedule graph G_S, by breaking every path in G_S that is longer than the optimal clock period P. For paths having a length less than P, no register is required because operations chaining is assumed.

For the circuit in Figure 1, applying the algorithm in [1] for register placement on the schedule graph G_S in Figure 2, starting from v_1, gives the placement of registers and their schedules as it is summarized by Table 2. Using the results of this table, we present the placement of registers in the circuit as depicted by Figure 3. The number of registers that are placed is 8 and the number of phases is 4.

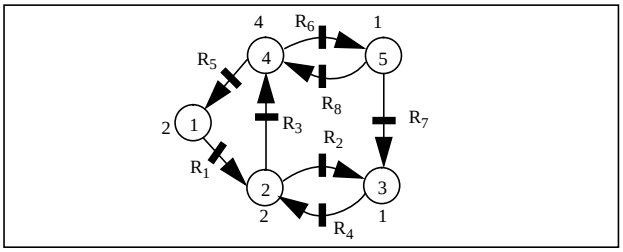

Figure 3 : Registers placement using ALAP.

Table 2 : Registers placement and their schedules.

Registers	Registers Placement	Schedule
R_1	$e_{1,2}$	2
R_2	$e_{2,3}$	1
R_3	$e_{2,4}$	0
R_4	$e_{3,2}$	2
R_5	$e_{4,1}$	0
R_6	$e_{4,5}$	3
R_7	$e_{5,3}$	1
R_8	$e_{5,4}$	0

4. Reducing register requirements

As explained in Section 3, the algorithm to place registers breaks only paths that are longer than the optimal clock period P, because operations chaining is assumed. Recall that this algorithm works on a schedule graph, G_S, where its edges are weighted as defined by (8). By examining equation (8), no register is required between u and v if $w_s(e_{u,v})$ is less than P. Hence, having $w_s(e_{u,v})$ as small as possible for each edge $e_{u,v}$ in G_S may allow to decrease the number of registers that will be placed by this algorithm. Having edges with this characteristic depends on the schedule chosen to construct the graph G_S. The problem of reducing the number of registers thus transforms to determining a schedule that allow $w_s(e_{u,v})$ to be as small as possible for each $e_{u,v}$ in G. To achieve that new goal, we develop a mathematical formulation summarized in Figure 4.

Let us define:

$$\varepsilon_{u,v} \geq w_s(e_{u,v}), \quad (9)$$

for each edge $e_{u,v}$. Minimizing $\sum \varepsilon_{u,v}$ over all edges tends to minimize the number of edges where a register will be necessary. The combination of (8) and (9) gives the inequality (10) in Figure 4. In this figure, inequality (11) represents the data dependency constraints that every valid schedule must satisfy. In the mathematical formulation in Figure 4, the variables are the schedule time of the first instance of each vertex v in V (i.e., $s_0(v)$) and the $\varepsilon_{u,v}$, which are defined for all vertices u and v such that the edge $e_{u,v}$ is in E.

For the circuit in Figure 1, 0, -2, -4, -4, and -1 is a valid schedule for the functional elements 1, 2, 3, 4 and 5, respectively. This schedule is an optimal solution for the formulation in Figure 4. Using this schedule, the optimized circuit presented in Figure 5 has the same functionality as the original one. But, only 7 registers and 3 phases are needed instead of 8 registers and 4 phases that are required for the circuit in Figure 3; recall that the circuit in Figure 3 is derived using an ALAP schedule. The schedule time of registers 1, 2, 3, 4, 5, 6, and 7 in the circuit in Figure 5 is 0, 0, 2, 0, 3, 0, and 0, respectively. Registers 1 and 2 are in the output of the same functional element and they have the same schedule time. Hence, only one of them is needed. We have the same thing for the case of registers 6 and 7. Hence, only 5 registers are required in this circuit

To solve the formulation in Figure 4, we examine three cases. Case 1: when there is no restriction on the type of each variable in this formulation. Case 2: when the variables have to be integers, and P is integer. Case 3: like case 2 but P is rational. But before examining these cases, let us first check if this formulation has a solution, and if it is possible to prune the solution space.

.

Formulation F_1:

$$\text{Minimize} \sum_{\forall e_{u,v} \in E} \varepsilon_{u,v}$$

Subject to:

$$\forall e_{u,v} \in E,\ \varepsilon_{u,v} \geq s_0(v) - s_0(u) + P \cdot w(e_{u,v}) \quad (10)$$

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v}) \quad (11)$$

Figure 4 : Scheduling for reducing register requirements.

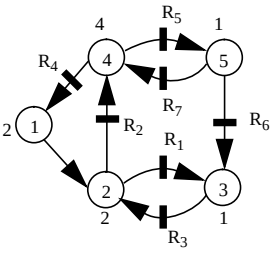

Figure 5 : Final circuit using Figure 4.

Lemma 1 : *Given that P is a feasible clock period, the formulation in Figure 4 has always a solution.*

Proof: an ASAP (or an ALAP) schedule satisfies (10) and (11). Hence, this schedule is a solution to the mathematical formulation in Figure 4. ❑

Lemma 2 : *If P and d(v) are integers, then for each v in V, ASAP(v) and ALAP(v) are integers.*

Proof: The procedure to find the ASAP and ALAP schedules for each vertex v in V is given in Section 2.4. Also, we have seen that this can be done by applying the Bellman-Ford's algorithm for longest paths on the graph G_P or G', where each $w_P(e_{u,v})$ is defined as in (5). Because P and, for each v in V, $d(v)$ are integers, then $w_P(e_{u,v})$ is integer. Hence, this algorithm produces integer values for ASAP or ALAP schedule. ❑

The ASAP and ALAP schedules may be used to prune the solution space of the mathematical formulation in Figure 4. This can be done by letting: $ASAP(v) \leq s_0(v) \leq ALAP(v)$ for each v in V.

The following subsections examine how to solve the mathematical formulation in Figure 4 for the three cases of interest.

4.1. Case 1: No restriction on the type of variables

In this case, the mathematical formulation in Figure 4 is a linear program. Hence, it can be solved efficiently by using the simplex method [13] or by using one of the two polynomial-time methods such that the ellipsoid method [14] or the interior point method [15]. In this paper, we have used the LP_Solve tool [11] (which is in the public domain) to obtain the experimental results.

4.2. Case 2: All variables and *P* are integers

In this case, the mathematical formulation in Figure 4 is an integer linear program which is NP-hard in the general case. But, in this mathematical formulation, the right hand sides of the inequalities are integers. From linear programming theory [12], we know that if the constraint matrix of an integer linear program is totally unimodular, and if the right hand sides of the inequalities (as in Figure 4) are integers, then the integer linear program and its relaxation, obtained by ignoring the constraint *integer*, have the same optimal solution. The relaxed formulation is a linear program and hence it can be solved as it was discussed in Section 4.1. Now, to solve our integer linear program as a linear program, we must prove that the constraint matrix, *A*, of the formulation in Figure 4 is totally unimodular.

Theorem 1 : *The constraint matrix, A, of the formulation in Figure 4 is totally unimodular.*

To prove Theorem 1, let us first recall the following theorem which is proved in [9].

Theorem 2 : *A matrix, X, is totally unimodular if and only if for every (square) Eulerian submatrix, Y, of X, we have that the sum of the entries of X divides by 4.*

Proof of Theorem 1: Let *m* be the number of edges and *n* be the number of vertices in the directed cyclic graph, *G*, modeling a circuit. The constraint matrix, *A*, of the mathematical formulation in Figure 4 is:

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}$$

where A_1, A_2, A_3, and A_4 are matrices of size $m \times m$, $m \times n$, $m \times m$ and $m \times n$, respectively. The matrix A_1 is the identity matrix. A_3 is the zero matrix. A_2 is the incidence matrix of *G* and $A_4 = -A_2$. Each line of A_2 contains exactly an 1 and an -1; the other entries of this line are equal to zero.

Let *B* be an arbitrary square eulerian submatrix of *A*. The sum of the entries of each line of *B* is even; all columns of *B* have this propriety too. Consequently, *B* and A_1 do not have any common element. Now, let us discuss the rest of the cases. First, it is well known that every incidence matrix is totally unimodular [10]. It is evident that the zero matrix is totally unimodular too. Hence, A_2, A_3, and A_4 are totally unimodular. Now, if *B* is a submatrix of A_2, or A_3, or A_4, then it is totally unimodular; hence, Theorem 2 is satisfied and *A* is totally unimodular. For the case where *B* is formed by elements in A_2 and by elements in A_4, or by elements in A_3 and by elements in A_4, then by the eulerian condition, each line of *B* has exactly one entry equal to 1 and one entry equal to -1 (the other entries are equal to zero); the sum of the entries of this line is equal to zero; consequently, the sum of the all entries of *B* is equal to zero and then it divides by 4; in this case, the condition of Theorem 2 is satisfied; hence, we have that *A* is totally unimodular. ❑

As a summary, we have proved that the constraint matrix of the mathematical formulation in Figure 4 is totally unimodular. The right hand sides of the inequalities in this formulation are integers. Consequently, solving the corresponding linear program, obtained by ignoring the *integer* constraint, gives the optimal solution of our integer linear program; this solution is guaranteed to be integer [12].

4.3. Case 3: Variables are integers, and *P* is rational

In this case, the mathematical formulation in Figure 4 is an integer linear program. But, because the right hand side of the inequalities of this formulation are not integers, we cannot solve this formulation as it was done in Section 4.2. In [2, 5], a theorem says that given a valid periodic schedule *s*, the schedule *s** defined by $s^*_n(v) = \lfloor s_n(v) \rfloor$, where $n \in N$ and $v \in V$, is a *k*-periodic schedule with the same throughput. Hence, to solve this formulation, we can ignore the constraint *integer* and solve the resulting linear program as it was done in Section 4.1. Then, the *k*-periodic schedule, *s**, can be determined.

5. Reducing the number of phases

In the method based on software pipelining [1], outlined in Section 1, once registers are placed and their schedule is fixed, the phases are determined. If a register, *R*, is placed on edge $e_{u,v}$, then its schedule time is:

$$s_v(R) = s_0(v) \bmod P, \tag{12}$$

where *P* is the optimal clock period. The number of phases is the number of the different $s_v(R)$ for all the required registers. Let us now analyze how to determine a schedule that gives a small number of phases. By equation (12), we have that:

$$\exists k_v \in N : s_v(R) = s_0(v) - k_v \cdot P, \tag{13}$$

and

$$s_v(R) < P. \tag{14}$$

By letting,

$$\delta_{u,v} \geq |s_v(R) - s_u(R)|, \tag{15}$$

for each two pair of different vertex *u* and *v* in *V*, and by minimizing $\sum \delta_{u,v}$, then the number of the phases tends to be reduced. Because the schedule we want to determine for producing a small number of phases must be valid, the inequalities described by (6) must not be violated. Using (13) and (15), we have that:

$$\delta_{u,v} \geq |s_0(v) - s_0(u) + (k_u - k_v) \cdot P|. \tag{16}$$

Putting together the equalities and inequalities: (16), (14) and (6), we have the mathematical formulation, for determining a valid schedule with a small number of phases, which is presented in Figure 6. In this figure,

inequalities (17) and (18) are equivalent to (16); by examining inequalities (17) and (18), we conclude that only one of them is required. Inequality (19) is derived from (13) and (14). Inequality (19) is not in the standard form of linear programs, but if P is integer, then it may be replaced by $s_0(v) - k_v \cdot P \le P - 1$. Inequality (20) is equivalent to (6). If P is integer, then this formulation is a mixed integer linear program. But, relaxing this formulation by ignoring the term $(\pm(k_u - k_v) \cdot P)$ in (17) and (18) and restricting the inequalities to the vertices having edges between them, and by removing (19), we have the mathematical formulation in Figure 7. In this formulation, the variables are the schedule time of the first instance of each vertex v in V (i.e., $s_0(v)$) and the $\delta_{u,v}$, which are defined for all vertices u and v such that the edge $e_{u,v}$ is in E. The resolution of the linear program in Figure 7 may be done as we have done for the case of the formulation in Figure 4.

For the circuit in Figure 1, 0, -2, -2, -4, and -4 is a valid schedule for the functional elements 1, 2, 3, 4 and 5, respectively. This schedule is an optimal solution for the formulation in Figure 7. Using this schedule, the optimized circuit presented in Figure 8 has the same functionality as the original one. It operates using 8 registers and 2 phases instead of 8 registers and 4 phases that are needed for the circuit in Figure 3; note that the circuit in Figure 3 is obtained using an ALAP schedule. The schedule time of registers 1, 2, 3, 4, 5, 6, 7 and 8 in the circuit in Figure 8 is 2, 2, 0, 2, 0, 0, 2, and 0, respectively. In this circuit, registers 5 and 6 are in the output of the same functional element, and they have the same schedule time; hence, only one of them is needed; consequently, the circuit can operate with 7 registers instead of 8.

Formulation F_2:

$$\textit{Minimize} \sum_{\forall u \in V,\, \forall v \in V,\, u \neq v} \delta_{u,v}$$

Subject to:

$$\forall u \in V,\ \forall v \in V,\ \forall k_u \in N,\ \forall k_v \in N \text{ and } u \neq v:$$

$$\delta_{u,v} \ge s_0(v) - s_0(u) + (k_u - k_v) \cdot P \quad (17)$$

$$\forall u \in V,\ \forall v \in V,\ \forall k_u \in N,\ \forall k_v \in N \text{ and } u \neq v:$$

$$\delta_{u,v} \ge s_0(u) - s_0(v) + (k_v - k_u) \cdot P \quad (18)$$

$$\forall v \in V,\ \forall k_v \in N:\ s_0(v) - k_v \cdot P < P \quad (19)$$

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \ge d(u) - P \cdot w(e_{u,v}) \quad (20)$$

Figure 6 : Scheduling for reducing the number of phases.

Formulation F_3:

$$\textit{Minimize} \sum_{\forall e_{u,v} \in E} \delta_{u,v}$$

Subject to:

$$\forall e_{u,v} \in E,\ \delta_{u,v} \ge s_0(v) - s_0(u) \quad (21)$$

$$\forall e_{u,v} \in E,\ \delta_{u,v} \ge s_0(u) - s_0(v) \quad (22)$$

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \ge d(u) - P \cdot w(e_{u,v}) \quad (23)$$

Figure 7 : The relaxed formulation of Figure 6.

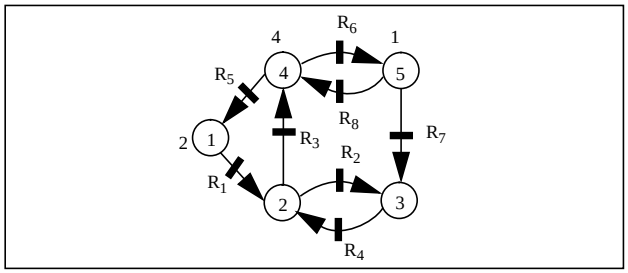

Figure 8 : Final circuit using Figure 7.

6. Experimental results

We have implemented the two methods presented in Sections 4 and 5 in a tool coded in C++, called CircuitOptimizer, which has the architecture described by Figure 9. Recall that the first method determines schedules for reducing register requirements, while the second one gives schedules for reducing the number of required phases. Starting from a given directed cyclic graph specification of a synchronous sequential circuit, the tool determines the optimal clock period and the ASAP and ALAP schedules, which could be used by the other components of the tool. Then, depending on the choice of the type of schedule required, automatic generation of the constraints of the mathematical formulation of this schedule is done. The LP_Solve tool [11] (in the public domain) is used to solve the generated mathematical formulation. Finally, the schedule found by the LP_Solve is parsed and used to place registers. The phases and their number are then determined.

To test the effectiveness of the methods that we have developed to determine schedules for reducing register and phase requirements, we have experimented the CircuitOptimizer tool on benchmarks selected from the ISCAS89 set. Table 3 reports results when the tool have been used to place a small number of registers. The results of the case when the target is to place registers for reducing the number of required phases are presented in Table 4. To determine the gain obtained by using our methods, we compare the results of these methods with the results of the original method [1] that uses ALAP. In Table 3, the number of the registers is reduced by a factor ranging from 24% to 38%. Note that even though it was not the objective of the method to reduce the number of phases, we nevertheless obtained reductions ranging from 19% to 67%. In Table 4, a substantial reduction of the number of phases has been obtained; the gain factor is between 12% and 70%; the number of registers is also reduced, except for the case of the circuit S5378 where it has increased as a result of reducing the number of phases. All the results in Tables 3 and 4 have been obtained in less than 20 minutes on an UltraSPARC-10 with 1GB RAM.

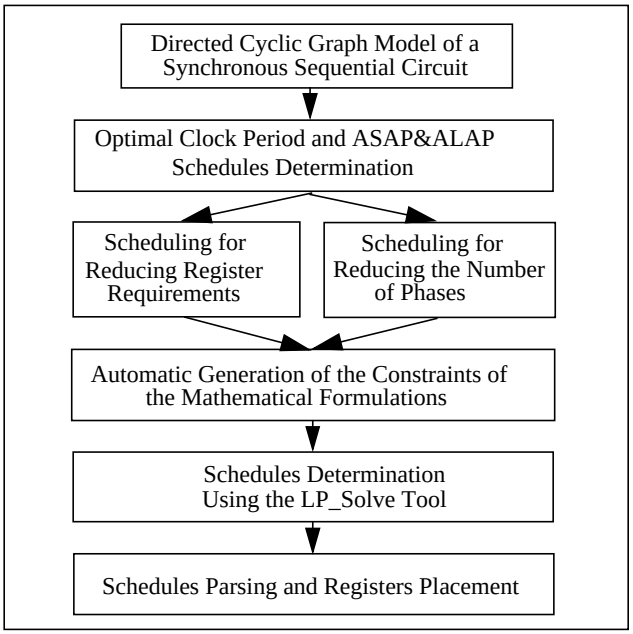

Figure 9 : The architecture of the CircuitOptimizer Tool.

Table 3 : Reducing register requirements.

	Arbitrary schedule: ALAP		Schedule produced by our method		Gain	
	# registers	# phases	# registers	# phases	# registers	# phases
S344	131	15	96	7	27%	53%
S641	142	16	90	13	37%	19%
S1423	422	65	320	47	24%	28%
S5378	1033	53	692	27	33%	49%
S9234	1042	48	643	16	38%	67%

Table 4 : Reducing phase requirements.

	Arbitrary schedule: ALAP		Schedule produced by our method		Gain	
	# registers	# phases	# registers	# phases	# registers	# phases
S344	131	15	79	6	40%	60%
S641	142	16	90	14	37%	12%
S1423	422	65	366	31	13%	52%
S5378	1033	53	1168	16	-13%	70%
S9234	1042	48	849	16	18%	67%

7. Conclusion

In this paper, we showed that choosing a good schedule has an impact on the number of registers that must be placed in the circuit derived using software pipelining techniques, and on the required number of phases. Reducing the number of registers contributes to the minimization of the area occupied by the circuit and reduces its power consumption, while reducing the number of phases reduces the complexity of the clock generation and distribution tasks.

We have developed two polynomial-time-solvable methods for determining schedules to reduce register and phase requirements. As demonstrated by the experimental results using a subset of the ISCAS89 benchmarks, the methods have proved to be very efficient for reducing the number of registers that must be inserted in the final circuit, and its number of phases.

References

[1] F.-R. Boyer, E.-M. Aboulhamid, Y. Savaria and M. Boyer, "Optimal Design of Synchronous Circuits Using Software Pipelining Techniques", *ACM Transactions on Design Automation of Electronic Systems*, Vo. 7, Num. 2, 2002.

[2] C. Hanen, "Study of NP-hard Cyclic Scheduling Problem: the Recurrent Job-Shop", *European Journal of Operation Research*, Vo. 72, 1994.

[3] S.-H. Gerez, S.-M.-H. de Groot, and O.-E. Herrmann, "A Polynomial-Time Algorithm for the Computation of the Iteration-Period Bound in Recursive Data- Flow Graphs", *IEEE Trans. on Circuits and Syst.*, No. 1, Vo. 39, 1 (Jan. 1992).

[4] E.-L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Reinhart, and Winston, New York, NY, USA, 1976.

[5] I.-E. Bennour, and E-M. Aboulhamid, "Les problèmes d'ordonnancement cycliques dans la synthèse de systèmes numeriques", *Technical Report* 996 (Oct. 1995), DIRO, Université de Montréal. `http://www.iro.umontreal.ca/~aboulham/pipeline.pdf`.

[6] A.-T. Ishii, C.-E. Leiserson, and M. C. Papaefthymiou, "Optimizing two-phase, level-clocked circuitry", *Journal of the ACM* 44, 1 (Jan. 1997).

[7] C.-E. Leiserson, and J.-B. Saxe, "Retiming synchronous circuitry", *Algorithmica* 6, 1, 1991.

[8] B. Lockyear, C. and Ebeling, "Optimal retiming of level-clocked circuits using symmetric clock schedules", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13, 9 (Sept. 1994).

[9] P. Camion, "Characterisation of totally unimodular matrices", *Proc. of the Amer. Math. Soc*., Vo. 16, 1965.

[10] C. Berge, *théorie des graphes et ses applications*, Dunod, Paris, 1958.

[11] The LP_Solve Tool: `ftp://ftp.ics.ele.tue.nl/pub/lp_solve/`

[12] A. Schrijver, *theory of linear and integer programming*, John Wiley and Sons, 1986.

[13] V. Chvatal, *Linear programming*, W. H. Freeman and Company, 1983.

[14] L.-G. Khachian, "A Polynomial algorithm in linear programming", *Soviet Math. Doklady*, Vo. 20, 1979.

[15] N. Karmakar, "A New polynomial-time algorithm for linear programming", *Combinatorica*, Vo. 4, 1984.